Université Libre de Bruxelles - Faculté des Sciences - Département d'informatique
INFO-F-303 : Réseaux

Réseaux
Résumé personnel

Christophe DUMEUNIER

Chapitre 1

Les réseaux d'ordinateurs et l'Internet

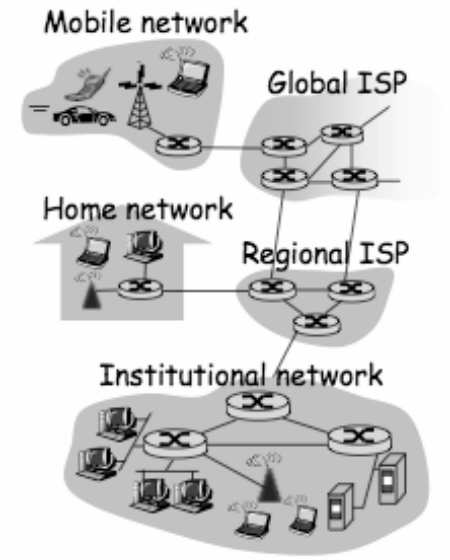
1.1 Qu'est-ce que l'Internet ?

L'Internet doit son existence et son bon fonctionnement aux *normes internet* définies par l'IETF (*Internet Engineering Task Force*). Elles se présentent sous forme de documents RFC (*Request for Comments*) qui sont pris comme base pour tout développement d'une application utilisant Internet comme moyen de communication.

L'*Internet* est une infrastructure de communication, un réseau d'ordinateurs à l'échelle planétaire, il relie ainsi des millions d'appareils informatiques (ordinateurs de bureau, serveurs, ordinateurs portables, PDA, téléviseurs, voitures, grille-pains) appelés *hôtes* ou *terminaux*. Ces terminaux sont reliés les uns aux autres au moyen de *liaisons de communication* (dont il existe une grande variété : fibres optiques, câbles coaxiaux, ondes radioélectriques,...).

Les terminaux ne sont généralement connectés au réseau que par le biais d'une seule liaison. En revanche ils sont indirectement reliés les uns aux autres au moyen de *routeurs*. Ceux-ci prennent en charge des *paquets* d'informations sur les liaisons entrantes et les redirigent sur les liaisons sortantes. Le chemin emprunté par un paquet entre l'expéditeur et le destinataire est appelé *route* ou *voie d'acheminement*.

Les terminaux accèdent à l'Internet grâce à des *fournisseurs d'accès à l'Internet* (FAI ou ISP (*Internet Service Provider*)). L'infrastructure d'un ISP n'est rien d'autre qu'un réseau de routeurs et de liaisons, et les ISP étant eux-mêmes reliés entre eux selon une structure de réseau. Internet est un réseau de réseaux !



L'Internet permet aux terminaux d'accéder à des *applications distribuées* en vue d'échanger des données. Pour cela l'Internet procure deux types de services à ses applications : un *service fiable orienté connexion* (données transmises par ordre chronologique et dans leur intégralité) et un *service non fiable sans connexion* (aucune garantie sur la bonne transmission, mais plus rapide).

Les terminaux, routeurs et autres rouages de l'Internet utilisent des *protocoles* qui contrôlent l'envoi et la réception d'informations au sein du réseau. Le fonctionnement d'un protocole peut se comparer aux conventions sociales, par exemple *demander l'heure*. La première phase serait d'entamer la conversation par un *bonjour* ou autre formule de politesse, à laquelle la personne répondra favorablement ou vous enverra balader. S'en suit la réelle question *Auriez-vous l'heure svp ?* suivie de la réponse *Il est 14h30*. Enfin la communication se clôture par un second échange de politesse tel que *Merci* suivi d'une réponse *A votre service*. Il s'agit là d'un protocole humain pour demander l'heure à un inconnu. Le processus est similaire au sein d'un réseau et toute communication requiert un minimum de deux entités utilisant le même protocole. De manière plus formelle :
Un *protocole* définit le format et l'ordre des messages échangés entre deux entités interlocutrices ou plus, ainsi que les actions générées au moment de la transmission ou réception d'un message ou de tout autre événement.

1.2 Réseaux d'accès

Le réseau de l'Internet est constitué de routeurs interconnectés, de réseaux interconnectés, d'hôtes et d'applications. Les *hôtes* (ou *systèmes terminaux*) sont les ordinateurs connectés à l'Internet. On les appelle ainsi car ils sont en périphérie du réseau. Ils peuvent être de natures très diverses, avec ou sans interface pour l'utilisateur. Les *serveurs* sont des terminaux chargés d'héberger des logiciels d'application et des

données. Contrairement aux autres terminaux, ils ne permettent pas un contact direct avec l'utilisateur.

De façon générale, les hôtes sont classés en *clients* et *serveurs* ; en particulier, une paire d'hôtes en communication est presque toujours une paire (*client*,*serveur*), un hôte pouvant prendre tantôt le rôle de client et tantôt le rôle de serveur selon les besoins.

Les réseaux TCP/IP, en particulier l'Internet, proposent deux types de services de bout-en-bout : un *service sans connexion* et un *service orienté connexion*. Lorsqu'une application a recours au *service orienté connexion*, le logiciel client et le logiciel serveur s'envoient mutuellement des paquets de contrôle avant de procéder à l'envoi de paquets contenant des données utiles. Une connexion est alors établie. Ce service est systématiquement accompagné du *transfert de données fiable* (données transmises dans le bon ordre et sans erreur), du *contrôle de flux* (surveiller qu'aucun noeud de la connexion ne sature l'autre par un trop grand nombre de paquets à une trop grande vitesse) et du *contrôle de congestion* (empêche le réseau d'arriver à saturation). Le service orienté connexion de l'Internet s'appelle TCP (Transmission Control Protocol). En revanche, le *service sans connexion* n'utilise pas la procédure de mise en présence : lorsque l'un des côtés d'une application souhaite envoyer des paquets, le programme les envoie tout simplement. De plus ce service n'est associé à aucun contrôle de flux ou de congestion. Ce service s'appelle UDP (User Datagram Protocol).

1.3 Coeur du réseau

L'architecture du coeur d'un réseau peut suivre deux approches différentes : la *commutation de circuits* ou la *commutation par paquets*.
Au sein de la commutation de circuits, les ressources requises le long d'un chemin sont réservées pour toute la durée de la session (par exemple, les réseaux téléphoniques). Avant que la source ne puisse transmettre, elle doit se mettre en relation avec le destinataire, et établir une connexion (appelée *circuit*) pour laquelle les commutateurs situés sur le chemin maintiennent la communication. Le débit est alors constant et garanti. Dans le cas de la commutation par paquets, ces ressources ne sont pas réservées et les messages envoyés doivent utiliser les ressources à la demande, devant parfois attendre leur tour. Des paquets sont ainsi transmis sur le réseau, mais sans aucune réservation de débit.

Lors d' une liaison, un circuit peut être mis en oeuvre au moyen de *multiplexage fréquentiel* (FDM) ou d'un *multiplexage temporel* (TDM). En FDM, la plage de fréquence est partagée entre les différentes connexions. TDM (qui tend à se substituer au FDM) utilise une répartition dans le temps grâce à des trames d'une durée précise, chaque trame étant divisée en un nombre d'intervalles bien défini. (Dans le cadre de la commutation par paquets, on parlera de *multiplexage statistique*.)

Parmi ces deux moyens d'envoyer des données, la commutation par paquets est la plus efficace. Voici un petit exemple pour s'en persuader. Supposons que différents

utilisateurs doivent se partager une liaison à $1Mbit/s$. Supposons également que chacun de ces utilisateurs alterne entre des périodes d'activité (où il génère des données à un taux constant de $100kbit/s$) et des périodes d'inactivité. Supposons enfin qu'un utilisateur n'est en moyenne actif que 10% du temps. En commutation de circuits, $100kbit/s$ doivent être en permanence réservés pour chaque utilisateur, la ligne ne peut donc supporter plus de 10 utilisateurs. En commutation par paquets, la probabilité qu'un utilisateur choisi soit actif à un instant donné est 1/10. S'il y a n utilisateurs sur la ligne ($n \geq 10$), la probabilité que plus de 10 utilisateurs soient actifs en même temps est $P_n = 1 - \frac{n!}{10^n} \sum_{k=0}^{10} \frac{9^{n-k}}{(n-k)!k!}$. A titre indicatif, $P_{15} = 0.0000000093$, $P_{25} = 0.0000117$, $P_{40} = 0.0014697$, $P_{70} = 0.0872691$. On voit que jusqu'à plus ou moins 40 utilisateurs, la probabilité de ne pas dépasser la capacité du lien est proche de 99,9%, les paquets traversent le lien presque sans délai. La commutation par paquets offre presque toujours la même performance de délai que la commutation de circuits, mais en autorisant un nombre d'utilisateurs beaucoup plus élevé.

L'utilisation de la commutation par paquets implique une segmentation fine des messages à envoyer à travers le réseau, les messages ainsi scindés devront être assemblés par le destinataire pour reconstruire le message. Cette procédure un petit peu fastidieuse se justifie largement. Imaginons qu'un gros paquet (par exemple $10000000bits$) doive traverser un réseau entre deux hôtes A et B reliés par deux commutateurs et trois liaisons, chacune d'un débit de $2Mbit/s$. Si le message n'est pas scindé, il faut exactement 5 secondes à A pour envoyer le paquet au premier commutateur. Le mode de transmission des commutateurs étant *store and forward*, le premier commutateur ne peut commencer à transmettre qu'après réception complète du message. Après 5 secondes il peut commencer à retransmettre le message au second commutateur. Nous obtenons un total de 15 secondes pour transmettre le message de A vers B. Supposons maintenant que le paquet soit découpé en 5000 paquets de $2000bit$ chacun. Il faut exactement une milliseconde pour transmettre le premier paquet au premier commutateur, qui peut alors commencer à le transmettre au second commutateur pendant que A transmet le second paquet. A aura ainsi fini de transmettre le premier paquet après 5 secondes, et il restera au dernier paquet deux segments à parcourir, ce qui fait un total de $5,002$ secondes pour transmettre la totalité du message. Nous voyons qu'une transmission en parallèle (*pipe-line*) est beaucoup plus efficace. Un deuxième avantage non négligeable de la scission des messages est la détection d'erreur et retransmission ou omission des données erronées. Non seulement, la probabilité qu'une erreur survienne dans un paquet est d'autant plus petite que le paquet est léger, et de plus, le temps de retransmission ou la perte de données occasionnés par ce paquet erroné n'aura que peu d'impact. Cependant, les paquets ne peuvent efficacement être scindés que dans certaines limites. En effet, tout paquet transmis sur le réseau possède une entête de taille fixe. Trop découper les paquets reviendrait à transporter plus d'en-têtes que de contenu réel. Un compromis doit donc être trouvé.

Il existe deux grandes catégories de réseaux à commutation par paquets : les *réseaux à datagramme* (réseaux transportant des paquets selon des adresses de destination) et les *réseaux à circuits virtuels* (réseaux transmettant les paquets en fonction de numéros de circuits virtuels).

Un *circuit virtuel* consiste en un chemin établi entre l'expéditeur et le destinataire, des numéros de circuits virtuels (un numéro pour chaque liaison le long du chemin) et des entrées dans des tables de conversion des commutateurs. (Ce résumé ne s'étendra pas plus sur les circuits virtuels, plus de détails dans [1] 35-37)

Les réseaux à datagrammes sont similaires aux services postaux. Un message envoyé est emballé sous enveloppe sur laquelle est apposée une adresse respectant une hiérarchie précise. Dans un réseau à datagrammes, l'en-tête de chaque paquet contient l'adresse du destinataire. Lorsqu'une adresse arrive à un commutateur, celui-ci observe une portion de celle-ci lui permettant de déduire le prochain commutateur (une table de conversion est prévue à cet effet dans chaque commutateur afin de déduire le port de sortie approprié). Notons que lors du passage d'une adresse, un commutateur ne garde aucune trace du passage du message (contrairement aux réseaux à circuits virtuels).

1.4 Réseaux d'accès et supports physiques

Réseaux d'accès

Il existe trois grands modèles d'accès au réseau : l'*accès résidentiel* (qui assure la connexion des particuliers), l'*accès d'entreprise* (qui assure la connexion des terminaux au sein d'une institution) et l'*accès mobile* (qui assure la connexion des terminaux mobiles).

Le mode d'accès *résidentiel* consiste à connecter un terminal à un routeur périphérique. Le plus souvent on a recours à l'utilisation d'un *modem à numérotation automatique* et d'une *ligne téléphonique*. Pour transmettre les données par la ligne téléphonique, le modem doit convertir les messages du format numérique au format analogique. Cette *ligne analogique* est constituée d'une *paire de fils de cuivre torsadée*. A l'autre extrémité de la ligne, le modem du fournisseur d'accès rétablit le signal au format numérique pour le transmettre à un routeur. Le débit autorisé par les modems permet de se connecter au maximum à $56kbit/s$. Ce procédé est très lent, de plus, le modem monopolise la ligne téléphonique pendant le temps de la connexion.

Pour remédier à cela, il existe deux grands modes d'*accès résidentiels à large bande* : la *ligne d'abonné numérisée* (DSL, Digital Subscriber Line) et la *liaison hybride fibre et coaxial* (HFC, Hybrid Fiber Coaxial Cable) appelée aussi *câble*. La technologie DSL, qui utilise les lignes téléphoniques déjà en place, repose sur le multiplexage fréquentiel. La liaison de communication est divisée en trois bandes de fréquences indépendantes : un *canal à haut débit* vers l'aval dans la bande $[50kHz, 1MHz]$, un *canal à débit intermédiaire* vers l'amont dans la bande $[4kHz, 50kHz]$ et un *canal téléphonique* dans la bande $[0Hz, 4kHz]$. DSL a été spécialement conçu pour des courtes portées, condition

indispensable pour augmenter la capacité de débit des lignes.

Tout comme pour le DSL, la technologie du câble a recours à des modems particuliers appelés *modems câble* qui divisent la connexion en deux canaux (vers l'amont et vers l'aval). Il s'agit d'un support de diffusion partagé : chaque paquet envoyé par la tête de réseau est transporté en aval le long de chaque liaison.

Les *entreprises* ont généralement recours à un *réseau local* (LAN, Local Area Network) pour connecter leurs terminaux à un routeur périphérique. La technologie *Ethernet* est la plus fréquemment employée, elle propose un débit de $10Mbit/s$ ou $100Mbit/s$ (ou même plus). Les connexions se font à l'aide de *lignes en cuivre à paire torsadée* ou de *câbles coaxiaux* qui relient les utilisateurs entre eux et au routeur. Ethernet a recours à un *support partagé*, ce qui signifie que les utilisateurs se partagent un débit global.

Il existe deux principaux modes d'*accès sans fil* à l'Internet. Au sein d'un *réseau LAN sans fil* (WLAN) reposant sur la technologie *Wi-Fi*, les utilisateurs itinérants peuvent transmettre et recevoir des paquets avec un *point d'accès sans fil* (*Hotspot*) dans un rayon de quelques dizaines de mètres. Dans les *réseaux d'accès étendus sans fil*, la station radio dessert les usagers dans un rayon de plusieurs kilomètres.

Le *WAP* (Wireless Access Protocol) et le *i-mode* sont deux technologies utilisant le mode d'accès mobile à l'Internet.

Supports physiques

Il existe deux catégories de supports physiques : les *supports guidés* (tels que fils de cuivre à paire torsadée, câbles coaxiaux,...) et les *supports non-guidés* (tels que Wi-Fi, programmes par satellite,...).

Le support physique de transmission le moins cher et le plus répandu est la *ligne téléphonique en cuivre à paire torsadée*. Les paires torsadées sont constituées de deux fils en cuivre isolés, d'un diamètre d'un millimètre, enroulés l'un sur l'autre en une spirale régulière réduisant l'interférence électrique avec les lignes voisines. La *paire torsadée non blindée* (UTP, Unshielded Twisted Pair) est généralement utilisée à l'intérieur d'un bâtiment. Le débit disponible varie de $10Mbit/s$ à $1Gbit/s$ selon le diamètre du fil de cuivre et de la distance.

Le *câble coaxial* est constitué de deux conducteurs en cuivre à structure concentrique muni d'un bon isolant et d'une gaine protectrice adaptée pour permettre des débits très élevés. Il existe deux types de câbles coaxiaux : le *câble coaxial en bande de base* et le *câble coaxial à large bande*. Le premier a un diamètre d'environ un centimètre, il est habituellement utilisé sans les réseaux locaux. Le terme *bande de base* signifie que les flux de bits sont directement transmis sur le câble, sans action de modulation du signal. Le câble coaxial à large bande est plus épais. Ce type de câble est habituellement utilisé dans les systèmes de télévision par câble.

Une *fibre optique* est un médium conducteur de lumière fin et flexible, qui permet la propagation sous forme d'impulsions lumineuses représentant chacune un bit d'information. Une simple fibre optique peut prendre en charge des débits de l'ordre de plusieurs

centaines de $Gbit/s$. Les fibres optiques ne sont pas sujettes au phénomène d'interférence électromagnétique, ne créent qu'une très faible atténuation du signal sur des tronçons allant jusqu'à une centaine de kilomètres et sont très difficiles à mettre sur écoute. Cela en fait un support de prédilection pour les transmissions longue distance. Néanmoins il s'agit d'une technologie très chère.

Les *canaux radio* envoient des signaux dans le spectre radio-électromagnétique et permettent ainsi d'éviter toute connexion physique. Les conditions de propagation déterminent l'*affaiblissement de trajet* et l'*affaiblissement par diffraction*, l'*affaiblissement par trajets multiples* et le *phénomène d'interférences*. Les *canaux radio terrestres* opèrent au niveau du globe terrestre, en voisinage (quelques dizaines de mètres) ou sur grandes distances (plusieurs dizaines de kilomètres). Les *canaux radio satellites* assurent les liaisons entre différents émetteurs ou récepteurs à haute fréquence en différents points du globe et des *satellites géostationnaires* ou *en orbite basse*. Le débit de ces satellites est de l'ordre du $Gbit/s$. La distance importante entre les satellites géostationnaires et le globe (36000 kilomètres) impose au signal un délai de propagation de 250 millisecondes.

1.5 Les ISP et les réseaux fédérateurs de l'Internet

Les réseaux d'accès situés en périphérie de l'Internet sont connectés aux autres éléments de cette entité gigantesque par une hiérarchie nivelée constituée de différents types de *fournisseurs d'accès à l'Internet*. Les fournisseurs d'accès sont placés à la base de cette hiérarchie, le sommet est formé d'un petit noyau d'ISP, dits de niveau 1. Tous les ISP de niveau 1 sont connectés entre eux deux à deux et sont connectés à un grand nombre d'ISP de niveau 2, formant une couverture internationale. Les ISP de niveau 1 sont également appelés *réseaux fédérateurs de l'Internet*. Un ISP de niveau 2 se caractérise généralement par une couverture régionale ou nationale, et il n'assure une connexion qu'à un nombre limité d'ISP de niveau 1.

Dans un réseau d'ISP, les noeuds par lesquels un ISP se connecte à d'autres ISP sont appelés *points de présence* (POP).

1.6 Retards et pertes au sein des réseaux

Au fil de son transport à travers les différents noeuds de son trajet, la transmission d'un paquet est affectée par plusieurs types de retards. Les plus importants sont le *temps de traitement à chaque noeud*, le *temps d'attente*, le *temps de transmission* et le *temps de propagation*. La somme de ces types de retards constitue le *temps total de transmission*. Le temps d'examen de l'en-tête d'un paquet et la détermination de la liaison de sortie constitue le *temps de traitement*. Dans la file d'attente du routeur, le paquet attend son tour, cela constitue le *temps d'attente*. Si L est la longueur du paquet et R le débit de la liaison entre A et B, le *temps de transmission* est égal à L/R. Le temps nécessaire au trajet entre l'origine et la destination est le *temps de propagation*. On définira généralement les variables $d_{\text{total}} = d_{\text{trait}} + d_{\text{att}} + d_{\text{trans}} + d_{\text{prop}}$. Regardons plus attentivement le temps d'attente, qui est particulier. Soit a le taux moyen d'arrivée

des paquets dans la file d'attente (en paquets/seconde), et par souci de simplicité, soit L la taille de tous les paquets. Le quotient La/R est l'*intensité du trafic*. Si $La/R > 1$, alors la vitesse d'arrivée moyenne des paquets est supérieure au rythme auquel ceux-ci quittent la file d'attente, ce qui aura pour conséquence de faire augmenter sans cesse la file d'attente. Si $La/R = 1$, alors la nature du trafic entrant affecte le temps d'attente. Enfin, si La/R est beaucoup plus petit que 1 les temps d'attentes seront souvent très faibles voire inexistants.

En réalité, les files d'attente des routeurs sont finies, ce qui signifie que lorsqu'un paquet arrive dans une file d'attente qui est déjà pleine le routeur l'abandonne et le paquet est *perdu*.

1.7 Les couches de protocoles

Afin de réduire la complexité de la conception d'un réseau, les architectes de réseau ont choisi d'organiser les protocoles, ainsi que les matériels et logiciels qui en dépendent, en différentes *couches*, chaque protocole étant associé à une couche particulière. Au sein d'un réseau, chaque couche est susceptible d'assurer une ou plusieurs des fonctions suivantes : la *détection d'erreurs*, le *contrôle de flux*, la *segmentation* puis le *réassemblage*, le *multiplexage*, l'*établissement de connexion*.

La pile de protocoles Internet est constituée de cinq couches successives : La *couche application* est responsable de l'exécution des différentes applications réseau. La *couche transport* assure le transport des messages de la couche application entre les terminaux client et serveur ; elle a recours à deux protocoles de transport : TCP (orienté connexion) et UDP (service minimum). La *couche réseau* est responsable de l'acheminement des datagrammes entre les serveurs ; elle est constituée principalement d'un protocole qui définit les champs du datagramme (protocole IP). La *couche liaison* assure le transfert des paquets de données entre les noeuds du réseau. Enfin la *couche physique* se charge de transmettre les bits d'information d'un noeud à un autre.

Chapitre 2

La couche application

2.1 Principes des protocoles de la couche Application

Le logiciel d'une application se répartit sur plusieurs systèmes (l'application Web par exemple est le fruit de deux logiciels communicant l'un avec l'autre : le navigateur de l'utilisateur et le programme correspondant sur le serveur. On parlera donc plus de *processus* que programme pour désigner les entités interlocutrices. Des processus ne cohabitant pas dans la même machine communiquent par l'envoi et la réception de *messages* à travers le réseau. Les applications réseau disposent de *protocoles* définissant le format et l'ordre des messages échangés. Attention à faire la distinction entre *application réseau* et *protocole de la couche application* ; par exemple, le Web est une application de réseau qui permet aux utilisateurs d'obtenir des documents hébergés par des serveurs Web, et HTTP serait son protocole principal.

Une application réseau comprend généralement deux *pôles* : un *pôle client* et un *pôle serveur*. De façon générale les communications s'établissent entre le pôle client d'un terminal et le pôle serveur d'un autre terminal distant sur le réseau. De nombreuses applications utiliseront aussi bien le pôle client ou le pôle serveur, selon le terminal qui a ouvert la connexion.

Deux processus interagissant font passer les messages qu'ils s'envoient au travers de leurs *interfaces de connexion* (ou *sockets*, ou *interface API*). Une interface de connexion agit comme une porte. De l'autre côté de l'interface se trouve la couche transport, cette interface sert de liaison entre la couche application et la couche transport. Cependant ce sera au développeur de choisir parmi les protocoles de transport disponibles.

Dans les applications internet, le serveur de destination est identifié au moyen de son *adresse IP* unique. L'application expéditrice doit aussi savoir à quelle application elle s'adresse chez le destinataire, cette information est un *numéro d'accès* (*numéro de port*).

Certaines applications (telles que messagerie électronique, transfert de fichiers, applications financières,...) exigent une fiabilité de transfert maximale, sans aucune perte de données. Par contre, d'autres applications (dites *à tolérance d'erreurs*) (transfert audio ou vidéo,...) ne se soucient pas d'une quantité raisonnable de pertes de données qui ne se traduisent que par des petites perturbations locales. D'autres applications auront

besoin d'un débit minimal disponible en permanence (téléphonie, vidéoconférence,...) et doivent être en mesure d'envoyer des données et de s'assurer de leur arrivée à ce même débit, codant au besoin à un rythme différent si le débit venait à manquer ; on parle alors d'applications *sensibles au débit*. Enfin, certaines applications ont des contraintes de temps (jeux en réseaux, ou toute forme d'interactivité).

Le modèle de services TCP comprend un service orienté connexion et un service de transfert de données fiable (l'un ne va pas sans l'autre, ce ne sont pas des options). Le *service orienté connexion* signifie qu'il y a échange d'informations de contrôle de la couche transport entre le client et le serveur avant l'échange des messages venant de la couche application. On dit alors qu'une *connexion TCP a été établie* entre les interfaces. La connexion est dite *duplex* : les deux processus peuvent s'envoyer et recevoir des messages simultanément. Le *service de transport fiable* offre la possibilité au processus de compter sur TCP pour assurer le transfert de leurs données sans erreur et dans l'ordre. TCP comprend également un mécanisme de contrôle de congestion, qui tend à brider les processus d'expédition lorsque le chemin entre la source et le destinataire semble saturé. En contre partie, TCP n'apporte aucune garantie de débit minimum ni temps minimum de transfert (le données arriveront, mais personne ne sait en combien de temps).

UDP quant à lui est un processus de transport extrêmement simple et léger, doté d'un modèle de service minimum. Il ne nécessite aucun échange préalable entre deux processus, de plus les échanges de données sont non-fiables, ce qui signifie que lorsqu'un processus expédie un message, il ne se soucie pas de son arrivée.

2.2 Le Web et HTTP

Le protocole HTTP (*HyperText Transfert Protocol*) est le protocole de la couche application utilisé par le Web. Il est mis en oeuvre par l'interaction de deux logiciels : un logiciel client et un logiciel serveur, par le biais du protocole TCP de la couche transport.

Un petit peu de vocabulaire :
Une *page Web* est constituée de différents fichiers accessibles au moyen d'adresses propres (URL, *Uniform Resource Location*). La plupart des pages Web sont formées d'un fichier HTML de base et d'un certain nombre de fichiers référencés. Chaque URL comprend deux parties : le nom du serveur hébergeant l'objet et le chemin d'accès. Par exemple : `www.nomduserveur.com/chemin01/chemin02/fichier.xml`.
Un *navigateur* est un agent utilisateur pour le Web, il permet d'afficher les pages Web sollicitées et propose de nombreuses fonctions. Le navigateur est également chargé de mettre en oeuvre le pôle client de HTTP.
Un *serveur Web* héberge des objets Web, tous accessibles au moyen d'un URL, il est chargé de mettre en oeuvre le pôle serveur de HTTP.

HTTP définit les modalités des requêtes et de transfert de pages Web entre un client et un serveur. Lorsque l'utilisateur sollicite une page Web, le navigateur envoie au serveur des messages de demande HTTP concernant les objets contenus dans cette page. Celui-ci répond au moyen de messages HTTP contentant les dits objets. Avec HTTP, le serveur ne stock pas d'informations d'état relatives au client, ce qui signifie que si un client sollicite deux fois de suite le même objet dans un court délai, l'objet sera renvoyé une seconde fois. On parle de *protocole sans mémoire*.

HTTP peut mettre en oeuvre deux types de connexions : des *connexions persistantes* et des *connexions non-persistantes*. En présence de connexions non-persistantes un seul objet Web peut être transféré à la fois sur une connexion TCP. Dans le cas de connexions persistantes il n'est pas nécessaire d'établir une nouvelle connexion pour le transfert de chaque nouvel objet.

Lors du transfert d'une page Web (par exemple un fichier HTML et dix images jpg) entre un serveur et un client dans les cas de connexions non-persistantes :

- Le client HTTP établit une connexion TCP avec le serveur sur l'accès 80.
- Le client HTTP envoie un message de demande HTTP au serveur via l'interface de connexion associée à la connexion TCP établie. Ce message contient le chemin d'accès de la page sollicitée.
- Le serveur HTTP reçoit le message de demande via l'interface de connexion, récupère l'objet à son adresse de stockage, l'intègre dans un message de réponse et envoie le tout au client via l'interface de connexion.
- Le serveur HTTP ordonne à TCP d'interrompre la connexion.
- Le client HTTP reçoit le message de réponse via son interface, la connexion TCP est interrompue. Le message indique que le fichier qu'il transporte est de type HTML, avec dix références vers les dix images.
- Les quatre premières étapes sont alors répétées pour chacun des objets référencés.

Il aura ainsi fallu 11 connexions TCP distinctes pour transférer la page Web dans son intégralité. Notons tout de même que les navigateurs permettent de solliciter les dix images via des connexions TCP parallèles ou successives.

Avec des connexions persistantes, la connexion TCP est maintenue par le serveur après l'envoi de la réponse. Le serveur HTTP interrompra la connexion TCP après un temps d'inactivité déterminé par l'administrateur du serveur.

Il existe deux types de connexions persistantes : avec ou sans *pipelinage*. En l'absence de pipelinage, le client émet une nouvelle requête uniquement après réception de la réponse à sa requête précédente. Il subit donc un RTT pour la requête et la réception de chaque objet contenu dans la page, et de plus, cela laisse la connexion TCP inactive plus de la moitié du temps. Avec pipelinage, le client HTTP émet une requête dès qu'il rencontre une référence à un objet, c'est-à-dire qu'il peut émettre une nouvelle requête avant même d'avoir reçu la réponse de la précédente. Cela réduit le temps total à un seul RTT (pour des petits objets).

Il existe deux types de messages HTTP : les *demandes* et les *réponses*. Ces messages sont rédigés au format ASCII. La première ligne est la *ligne de demande* composée de trois champs (méthode, adresse URL et la version du protocole utilisé). Le champ relatif à la méthode peut prendre différentes valeurs : GET (sollicite un objet), POST (données relatives à un formulaire), HEAD (sollicite un objet, mais sans le recevoir (débug)), PUT (chargement d'un objet) et DELETE (suppression d'un objet), PUT et DELETE n'étant disponible que pour HTTP/1.1. Elle est suivie des *lignes d'en-tête*.

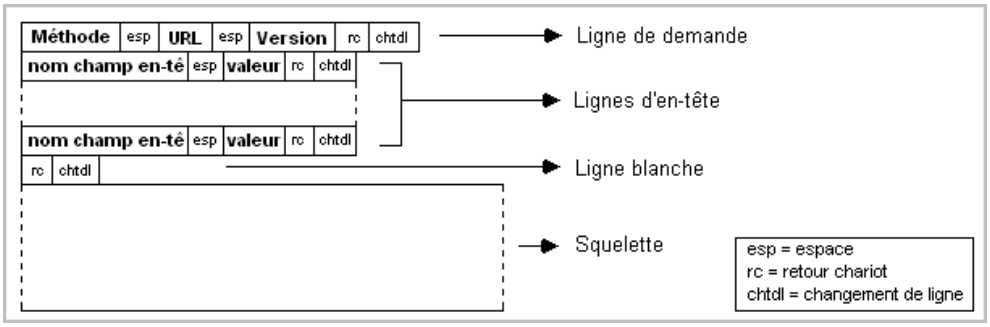


Schéma général d'un message de demande HTTP.

Exemple d'en-tête :

```
GET /somedir/page.html HTTP/1.1
Host: www.someschool.edu
Connection: close
User-Agent: Mozilla/4.0
Accept-language:fr
```

Un message de réponse est constitué de trois sections : une ligne d'état, plusieurs lignes d'en-tête et le squelette. Ce dernier contient la partie la plus importante, à savoir l'objet sollicité. La ligne d'état contient entre autres un code d'état qui rend compte des résultats de la requête ; quelques exemples : `200 OK` → la requête est réussie et l'information est contenue dans la réponse, `301 Moved Permanentely` → l'objet sollicité a été définitivement déplacé et une nouvelle URL est spécifiée, `400 Bad Request` → la demande n'a pas été comprise par le serveur, `404 NotFound` → le document est introuvable sur le serveur.

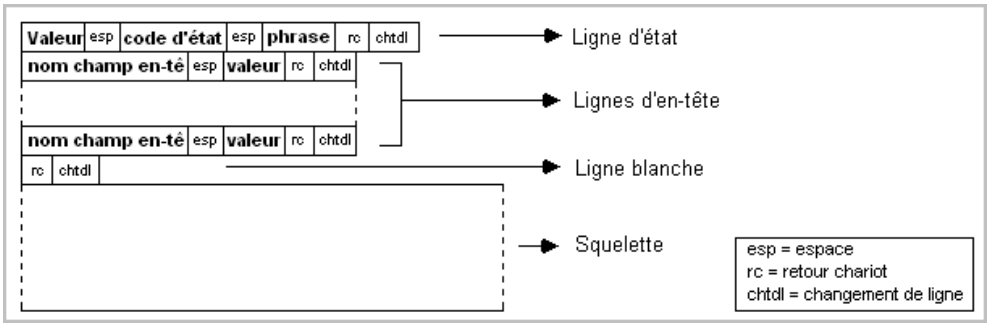


Schéma général d'un message de réponse HTTP.

Exemple d'en-tête :

```
HTTP/1.1 200 OK
Connection: close
Date: Thu, 06 Aug 1998 12:00:15 GMT
Server: Apache/1.2.0 (Unix)
Last-Modified: Mon, 22 Jun 1998 09:23:24 GMT
Content-Length: 6821
Content-Type: text/html
```

Nous avions spécifié qu'un serveur HTTP est sans mémoire et qu'il ne conserve pas d'informations d'état. Cependant, il est souvent utile de pouvoir identifier ses visiteurs (accès réduits, privilèges, panier personnel,...). HTTP procure deux mécanismes d'identification : la *procédure d'autorisation* et les *cookies*.

Une procédure d'autorisation repose sur l'utilisation des en-têtes HTTP et de codes d'états particuliers. Un client enverra tout d'abord un message de demande ordinaire, auquel le serveur répond par un code d'état `401 Authorization Required` et spécifie les détails de la procédure d'authentification (en général, une demande de login et de mot de passe). La demande est alors réexpédiée avec un en-tête contenant le login et le mot de passe.

Les cookies constituent un autre système permettant aux sites Web de rassembler des données sur leurs visiteurs. Le déroulement se fait en quatre étapes :

- L'insertion d'une ligne d'en-tête particulière dans le message de réponse HTTP ;
- La réitération de la demande avec la ligne d'en-tête correspondante ;
- L'envoi d'un fichier témoin qui est conservé par le navigateur dans le post de l'utilisateur ;
- l'intégration des informations dans une base de données sur le serveur Web.

En enregistrant les objets déjà demandés, la mise en mémoire cache sur le Web tend à réduire les temps de réponse et à diminuer le trafic encombrant l'Internet. Les *caches Web* peuvent se trouver chez le client ou au sein d'un serveur intermédiaire sur le réseau. Ce procédé pose cependant un problème : la version d'un objet résidant dans la mémoire cache d'un serveur peut être périmée. Heureusement, HTTP est doté d'un mécanisme s'assurant de l'actualité des objets, connu sous le nom de *GET conditionnel*.

2.3 DNS, le service d'annuaire de l'Internet

Les serveurs de l'Internet peuvent être identifiés de nombreuses manières, l'approche la plus fréquente étant le nom de serveur (tels que `www.yahoo.com` ou `www.tf1.fr`) pour des raisons évidentes de mémorisation. Un nom de serveur ne fournit pourtant aucune information quant à sa localisation sur le réseau, l'idée est donc d'attribuer à chaque serveur un identifiant à structure hiérarchique fixe, d'une longueur de 4 octets , appelée *adresse IP*, entre `0.0.0.0` et `255.255.255.255`. On parle de structure hiérarchique car sa lecture de gauche à droite révèle progressivement davantage d'informations sur l'emplacement du serveur sur l'Internet.

Pour assurer la correspondance entre les noms de domaines et les adresses IP, il faut un service d'annuaire permettant de traduire les noms de serveur en adresse IP. C'est ce pour quoi est fait le DNS (*Domain Name System*). Il s'agit à la fois d'une base de données distribuée mise en oeuvre au sein d'une hiérarchie de *serveurs de nom* et un protocole de la couche application qui permet aux serveurs ordinaires et aux serveurs de nom de communiquer les uns avec les autres pour assurer le service de conversion. Le protocole DNS est un protocole qui utilise UDP sur l'accès 53 sur des machines Unix souvent équipées du logiciel BIND (Berkeley Internet Name Domain).

Le DNS est fréquemment utilisé par d'autres protocoles (tels que HTTP, SMTP, FTP) pour délivrer les adresses IP correspondant aux noms de serveurs demandés par les utilisateurs. Par exemple, pour qu'un utilisateur puisse envoyer un message au serveur `www.candiulb.be/forum/index.php`, il doit d'abord connaître son adresse IP, qui s'obtient de la manière suivante : le navigateur extrait le nom du serveur (`www.candiulb.be`) de l'URL et le remet au pôle client de DNS, celui-ci formule une requête auprès d'un serveur de nom, à laquelle il reçoit une réponse concernant l'adresse IP correspondante ; puis le navigateur établit une connexion TCP avec le processus serveur répondant à cette adresse.

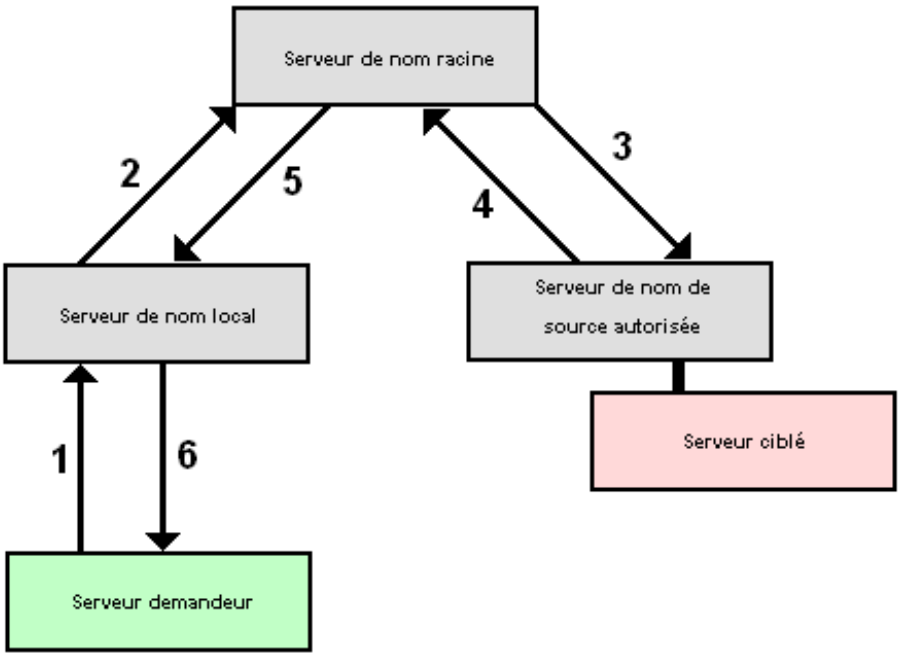
En plus de ses fonctions de traduction, DNS propose plusieurs autres services importants tels que la gestion d'alias pour les serveurs (un serveur peut donc avoir plusieurs noms), l'attribution d'alias pour les serveurs de messagerie, ou une répartition de charge pour les serveurs extrêmement sollicités et qui seront répliqués pour diminuer le taux d'affluence.

Lorsqu'une application exploitée sur le poste d'un utilisateur a besoin de convertir un nom en adresse IP, elle sollicite tout d'abord le pôle client du DNS qui prend le relais et lance une recherche à travers le réseau ; après un certain temps, celui-ci reçoit une réponse contenant l'adresse IP correspondant à sa demande. Ainsi au point de vue de l'application, le DNS fonctionne comme une boîte noire.

Une version simplifiée du DNS consisterait en un serveur de noms unique contenant toutes les correspondances existantes. Ce système a l'air simple, mais impossible à mettre en oeuvre, pour causes : fragilité d'un site central unique, volume de trafic trop important, base de données centralisée trop éloignée de certains utilisateurs, problèmes de maintenance dus au volume énorme des données à stocker. DNS se doit donc d'être

un système distribué.

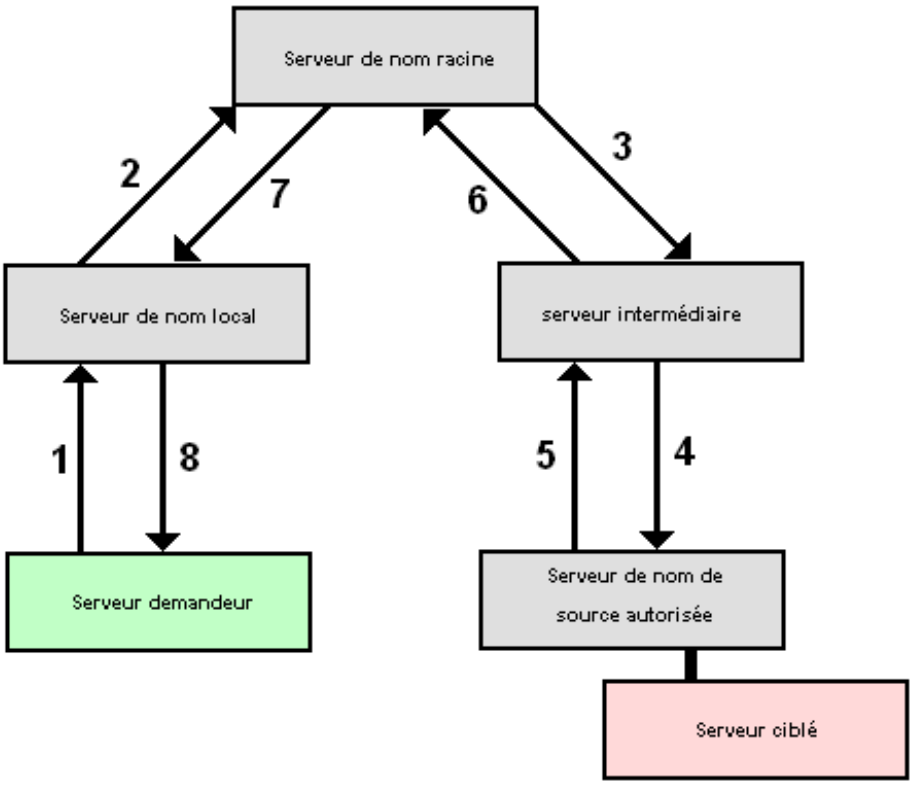
DNS utilise un grand nombre de serveurs de noms, organisé de manière hiérarchique et distribué dans le monde entier. Il existe trois types de serveurs de noms : les *serveurs de noms locaux*, les *serveurs de nom racine* et les *serveurs de nom de source autorisée*. Chaque fournisseur d'accès possède un serveur de noms local, vers lequel vont toutes les recherches DNS formulées au sein de ce réseau local. Un serveur de noms local est forcément *proche* du client. L'Internet compte environ une douzaine[1] de serveurs de noms *racine*, la plupart étant situés aux Etats-Unis. Lorsqu'un serveur local de noms n'est pas en mesure de répondre à une demande il se transforme en client DNS et interroge un serveur de nom racine, si celui-ci a une réponse il l'envoie au serveur de noms DNS, qui la transmet alors à l'auteur de la demande ; si le serveur de nom racine ne peut lui non plus satisfaire la demande directement, il répond en donnant l'adresse IP d'un serveur de nom de source autorisée qui connaîtra certainement la correspondance recherchée. Tout serveur est enregistré auprès d'au moins deux serveurs de noms de source autorisée, en général il s'agit tout simplement du fournisseur d'accès. Un serveur de nom est qualifié *de source autorisée* pour un serveur donné, s'il dispose en permanence d'archives DNS permettant d'établir la conversion pour ce serveur.



Recherches successives pour l'obtention de l'adresse IP
du serveur ciblé par le serveur demandeur.

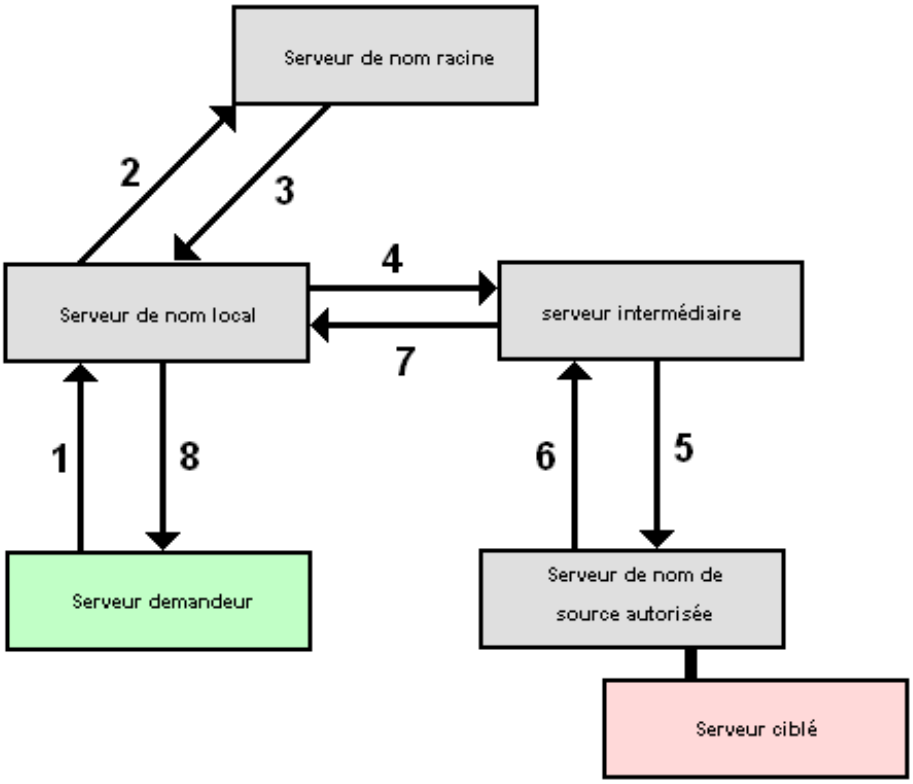
[1] chiffre datant de 2003 dans [1]

Nous avons considéré un principe un peu trop simplificateur, en effet, pour un serveur donné, le serveur de nom racine ne connaît parfois que l'adresse IP d'un serveur intermédiaire de noms qui lui sera en mesure de conduire au serveur approprié, ou même parfois de plusieurs serveurs de noms intermédiaires.



Recherches successives comprenant la participation
d'un serveur de noms intermédiaire.

Toutes ces recherches que nous venons de voir étaient du type *récursives*, mais DNS autorise également des *recherches itératives* à n'importe quel moment du processus de recherche : si un serveur n'est pas en mesure de répondre favorablement à la demande, il renvoie directement l'adresse IP du prochain serveur de nom de la chaîne. En général, toutes les demandes d'une même recherche DNS sont récursives, mis à part celle émanant du serveur local de nom adressée au serveur racine qui est de nature itérative. Cette démarche est préférable, les serveurs racines traitant généralement de grands volumes de demandes.



Recherches successives comprenant une recherche itérative.

Une dernière propriété importante de DNS est la *mise en mémoire cache*, qui permet de minimiser le temps de réponse des recherches. Lorsqu'un serveur de nom reçoit l'adresse IP correspondant à un nom de serveur, il la place dans sa mémoire cache en même temps qu'il transmet l'information. Toute demande concernant le même nom de serveur pourra ainsi être satisfaite immédiatement. Cependant, comme ces associations sont susceptibles de changer souvent, la mémoire cache doit être fréquemment réactualisée.

2.4 Distribution de contenu

La notion de *distribution de contenu* fait référence aux mécanismes permettant de dupliquer du contenu sur différents serveurs au sein de l'Internet et de fournir aux utilisateurs les moyens de déterminer les serveurs les plus aptes à leur fournir les objets sollicités.

Un *cache Web* (ou *serveur Proxy*) est une entité réseau qui exécute les requêtes HTTP pour le compte d'un serveur d'origine ; il dispose de sa propre capacité de stockage, au

sein de laquelle il conserve une copie d'objets récemment sollicités. La procédure entre un navigateur et un proxy associé serait la suivante :

- Le navigateur établit une connexion TCP avec le cache Web, et lui envoie une requête HTTP concernant l'objet désiré.
- Le cache Web vérifie s'il dispose de cet objet en local ; si c'est le cas il transmet directement l'objet au navigateur client au sein d'un message de réponse HTTP.
- Si le cache Web ne dispose pas de l'objet, il se connecte au serveur d'origine auquel il envoie une requête HTTP concernant l'objet ; et reçoit ainsi l'objet du serveur d'origine.
- Le cache Web fait une copie locale de l'objet reçu puis l'envoie par message de réponse HTTP au navigateur client.

Cette méthode réduit sensiblement le temps de réponse des requêtes HTTP ; de plus cela limite le trafic sur l'Internet pour rester en local, ce qui se traduit aussi en économie de débit vers l'extérieur.

Afin d'augmenter leur efficacité, il est possible de faire coopérer plusieurs caches Web, même s'il s'agit de postes éloignés géographiquement. Un serveur cache peut par exemple être configuré de manière à envoyer ses requêtes HTTP à un cache placé au sein d'un ISP. De cette manière si le premier ne dispose pas de l'objet sollicité dans sa mémoire locale, il transmet au suivant. Le fait de faire ensuite appel à un cache Web plus étendu, permet d'augmenter le taux de réussite de la requête, et est par conséquent très efficace. Un autre mode de mise en mémoire cache coopérative est un regroupement *en grappe*, souvent au sein d'un même réseau local. L'accès au bon serveur se fait alors par un hachage prédéfini.

Chapitre 3

La couche transport

3.1 Les services de la couche transport

Un protocole de la couche transport fournit une *communication de type logique* entre des processus d'application exploités sur les différents serveurs : du point de vue des applications, les serveurs sont en communication directe et échangent des messages sans se soucier des caractéristiques de l'infrastructure physique utilisée pour les véhiculer. Les protocoles de transport sont mis en oeuvre au niveau des terminaux, les routeurs du réseau n'ont aucun lien avec la couche transport (nous verrons qu'ils se limitent aux couches réseau, liaison et physique). Pour le transport des messages venant de la couche application, ceux-ci doivent être fragmentés en morceaux de taille plus réduite, précédés chacun d'un en-tête de couche transport. Ces fragments sont alors envoyés, via la couche réseau, au terminal à l'autre extrémité de la liaison qui se chargera de reconstituer le message sous sa forme originale pour le remettre de son côté à la couche application. Ces fragments de messages échangés au sein de la couche transport sont appelés *segments*.

L'Internet propose deux protocoles de transports : UDP (*User Datagram Protocol*) qui procure un service non fiable et sans connexion, et TCP (*Transmission Control Protocol*) qui procure un service fiable orienté connexion. A ce stade, il est nécessaire de préciser que la couche réseau (la couche inférieure, utilisée par la couche transport pour faire parvenir ses segments) offre un service dit *au mieux possible*, c'est-à-dire faisant au mieux pour faire parvenir les segments d'un serveur à l'autre, mais sans offrir aucune garantie. C'est donc à la couche transport d'élaborer une méthode permettant une parfaite fiabilité si elle le souhaite (pour TCP en particulier, mais aussi pour la détection d'erreur dans les messages pour UDP).

3.2 Multiplexage et démultiplexage

Il s'agit des phases qui permettent de transformer le service d'expédition de serveur à serveur procuré par la couche réseau en un service d'expédition de processus à processus de la couche application. Sur le serveur de destination, la couche transport reçoit

les segments en provenance de la couche réseau, elle est alors chargée de transmettre les messages reconstitués à partir des segments au processus d'application approprié. Un processus, en tant qu'élément d'application, dispose d'une *interface de connexion* (socket), ainsi la couche transport au sein du serveur destinataire n'envoie pas les données directement à un processus mais à une interface de connexion, chacune d'elles possédant un identifiant unique. L'action d'identifier l'interface d'un segment remis à la couche transport est appelé *démultiplexage* ; l'action inverse qui consiste à combiner les différents segments s'appelle *multiplexage*. Un segment de la couche transport est doté de certains champs prévus pour le multiplexage et le démultiplexage. Tout numéro d'accès est un nombre compris entre 0 et 65535.

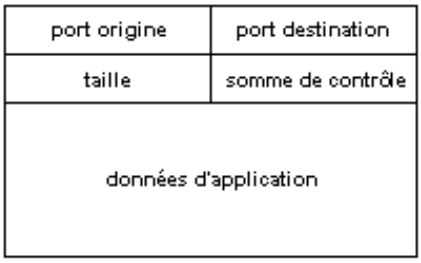
Multiplexage et démultiplexage sans connexion : Un programme exploité sur un serveur A quelconque génère une interface de connexion UDP. La couche transport affecte automatiquement un numéro d'accès (qui n'est pas utilisé, ou spécifié) à l'interface de connexion UDP créée. Imaginons qu'un processus de A avec l'interface de connexion UDP de numéro a souhaite envoyer des données à un serveur B sur l'interface de connexion UDP b. La couche transport du serveur A génère un segment incluant ses propres données et les données d'accès sur B, elle le remet ensuite à la couche réseau qui l'expédie (en faisant *au mieux*) au serveur B. Une fois arrivé, l'en-tête du segment est consultée pour identifier le numéro d'accès b et le segment est remis à l'interface de connexion correspondante. Remarquons qu'une interface est univoquement déterminée par son adresse IP et son numéro de port. Pour ce segment, le numéro d'accès d'origine a et l'adresse IP de A joue le rôle d'*adresse de retour*.

Multiplexage et démultiplexage orientés connexion : Nous venons de voir qu'une interface UDP est identifiable avec 2 champs : son adresse IP et son numéro de port. Une interface TCP nécessitera 4 champs : adresse IP et numéro de port des *deux* processus communiquant. L'en-tête d'un segment voyageant par TCP vers une interface doit donc également contenir ces 4 champs pour être redirigé lors du démultiplexage. Le fonctionnement général reste néanmoins le même.

3.3 Transport sans connexion : UDP

UDP réalise le strict minimum de ce qui peut être exigé d'un protocole de transport. Mis à part le service de multiplexage/démultiplexage et une légère détection d'erreur, il ne fait que lier la couche application à la couche réseau. Cette absence de complexité fait néanmoins gagner beaucoup de temps aux applications qui n'ont pas de temps à perdre et n'accordent que peu d'importance aux erreurs de transmission. Le faible volume des en-têtes qui en résulte fait également gagner du temps et réduit la quantité de données circulant sur le réseau.

Un segment UDP contient quatre champs d'en-tête : les numéros d'accès d'origine et de destination, la taille des données fournies dans le segment et une somme de contrôle qui permettra une détection d'erreur éventuelle.



Structure d'un segment UDP.

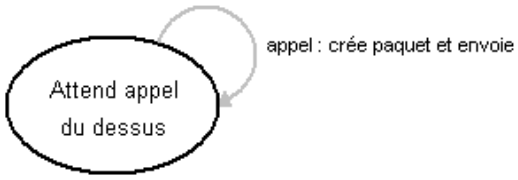
La somme de contrôle UDP permet de détecter l'altération de certains bits du segment UDP depuis sa création. Au niveau du pôle expéditeur, UDP fait le complément à 1 de la somme de tous les mots de 16 bits du segment en éliminant tout dépassement de capacité et reporte le résultat dans le champ de la somme de contrôle. Au niveau du pôle destinataire, tous les mots de 16 bits sont comptabilisés, y compris la somme de contrôle. Si aucune erreur ne s'est introduite dans le paquet, alors la somme à l'arrivée sera forcément 1111111111111111. Si l'un des bits prend la valeur 0, le protocole sait qu'une erreur s'est glissée ; UDP n'entreprend en aucun cas la *correction* de cette erreur, certains systèmes se débarrassent tout simplement du segment, d'autres le remettent à l'application avec un avertissement.

3.4 Le transfert de données fiable

Un *transport de données fiable* consiste à transférer des données sans corruption ou perte de données, de plus, les données sont fournies au destinataire dans l'ordre où elles ont été envoyées. Le problème est complexe par le fait que la couche réseau n'est *pas* fiable. Cette section présente un développement progressif d'un protocole de transfert de données fiables vu d'un pôle expéditeur vers un pôle destinataire. (Note : nous parlerons ici de *paquets* plutôt que de segments.)

3.4.1 Transfert de données fiable sur canal fiable : rdt1.0

Le cas de figure le plus simple considère un canal totalement fiable. Le protocole `rdt1.0` est extrêmement simple. Le pôle expéditeur attend de recevoir des données de la couche supérieure, lorsqu'il reçoit des données, il les envoie au niveau inférieur et se remet en attente de données. Le pôle destinataire attend de recevoir un paquet du niveau inférieur, lorsqu'il en reçoit un, il le transmet au niveau supérieur.



Pôle expéditeur de `rdt1.0`

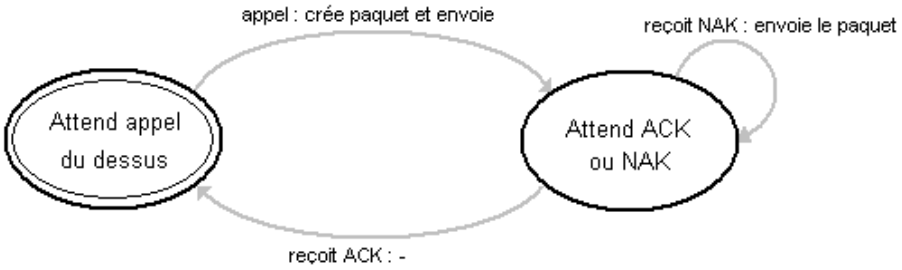
Pôle destinataire de `rdt1.0`

3.4.2 Transfert de données fiable sur canal avec erreurs : rdt2.0

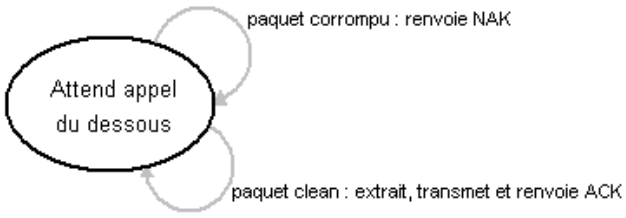
Pour ce second protocole, nous supposons toujours que les paquets arrivent dans le bon ordre, mais ils peuvent contenir des erreurs, c'est-à-dire des bits du message qui sont passés de 0 à 1 ou inversement. Un tel protocole a recours à des accusés de réception positifs et négatifs (protocole ARQ (*Automatic Repeat reQuest*)).

Pour cela, il faut tout d'abord un mécanisme de détection d'erreurs au pôle destinataire. Il faut également une réaction du destinataire, car lui seul est en mesure de décider si le paquet est arrivé avec ou sans erreur, ce qui se fait via des accusés de réception ACK (ok reçu) et NAK (paquet corrompu). Enfin, le pôle expéditeur doit être à même de renvoyer un paquet qui a été corrompu.

Le protocole `rdt2.0` va donc voir apparaître des états supplémentaires. Le pôle expéditeur doit toujours attendre les paquets du niveau supérieur, mais après un envoi, il doit se mettre en attente d'un accusé positif pour reprendre son attente d'un nouveau paquet. Tant qu'il reçoit des accusés négatifs, il devra réexpédier le même paquet. Le pôle destinataire quant à lui n'a pas beaucoup changé depuis la version 1.0, il se contente d'envoyer un ACK ou un NAK selon le résultat du test de validité du paquet.

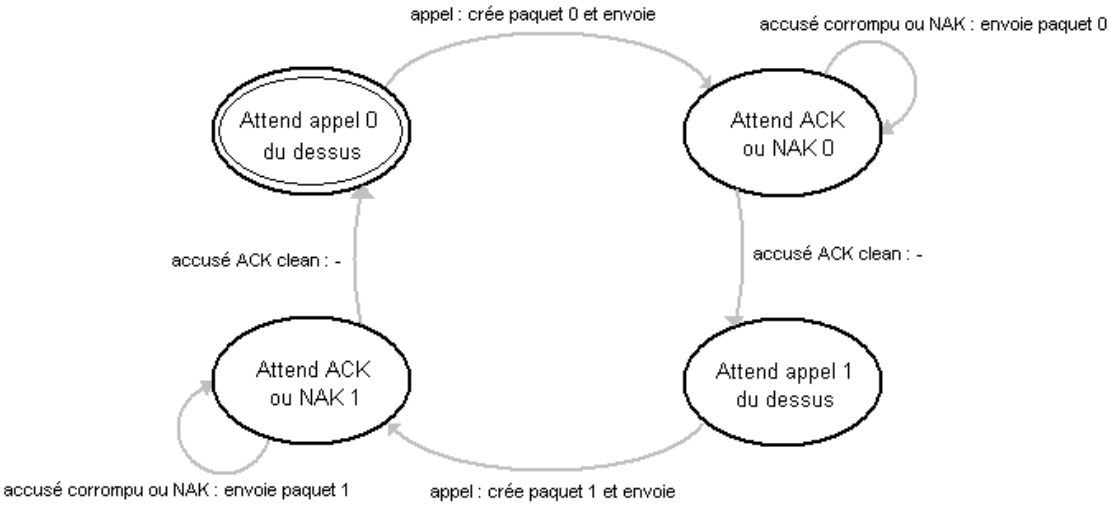


Pôle expéditeur de `rdt2.0`

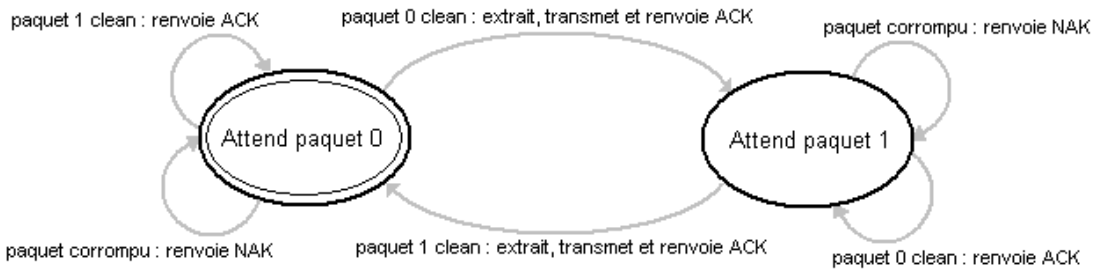


Pôle destinataire de `rdt2.0`

Ce protocole parait parfaitement utilisable, cependant il n'est pas du tout applicable, car il ne prend pas en compte le fait que les paquets contenant les accusés peuvent eux aussi être erronés, ce qui n'est pas identifiable par l'expéditeur qui n'a plus aucun moyen de savoir si le paquet qu'il avait envoyé à son destinataire est arrivé corrompu ou non. La solution à ce problème est d'utiliser d'une part un accusé avec un nombre suffisant de bits pour pouvoir déceler une erreur (un seul bit ne suffit pas), et d'autre part de faire réexpédier le paquet par l'expéditeur lorsqu'il reçoit un accusé corrompu. Cependant, cela pose encore un problème si l'accusé corrompu était un ACK, car le paquet réenvoyé sera considéré comme le suivant par le destinataire et donc dupliqué. La parade est d'ajouter un numéro de séquence aux paquets, attribués par l'expéditeur ; le destinataire n'a alors plus qu'à vérifier ce numéro de séquence pour déterminer s'il a déjà reçu ce paquet ou non. Notons que dans ce cas, un numéro de séquence d'un seul bit suffit. Cela forme un protocole `rdt2.1`, version améliorée du protocole `rdt2.0`.



Pôle expéditeur de `rdt2.1`



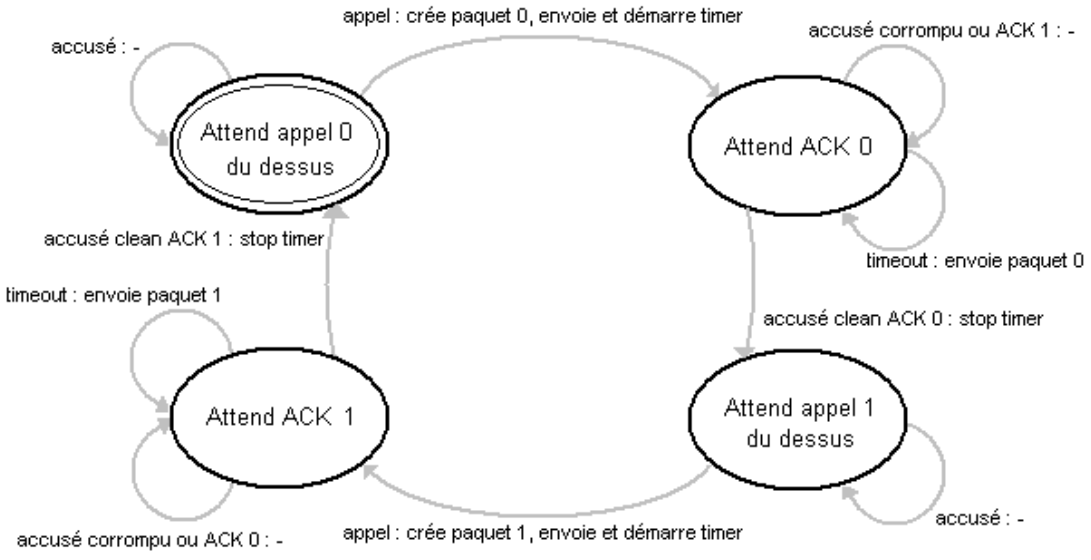
Pôle destinataire de `rdt2.1`

3.4.3 Transfert de données fiable sur canal avec pertes/erreurs : rdt3.0

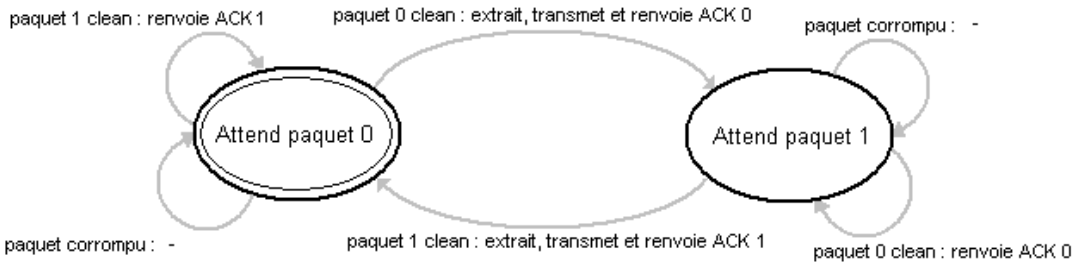
Nous supposons dans ce nouveau modèle que certains paquets peuvent ne jamais arriver à destination. De nouveaux problèmes sont alors à régler : quelles mesures à prendre lorsqu'un paquet n'arrive pas (pas de ACK ou NAK généré), ou lorsqu'un accusé émis n'arrive pas.

Une méthode parfaitement valable pour remédier à cela est de réexpédier le paquet si on n'a pas de nouvelles du destinataire après un temps déterminé. Mais combien de temps attendre ? Au moins aussi longtemps que le temps d'un aller retour, auquel s'ajoute les temps de traitements en différents points du réseau et chez le destinataire, ce qui n'est absolument pas fixe. Pour ne pas attendre trop de temps non plus, il faudra choisir un temps basé sur une bonne estimation du temps de réponse du destinataire. Il faut néanmoins remarquer que cela implique d'éventuels paquets dupliqués sur le réseau, qui ne seront pas pour autant corrompus et pour lesquels les accusés arriveront non-corrompus également. Du point de vue de l'expéditeur c'est une méthode efficace : il ne doit pas se soucier de savoir si c'est le paquet qui a été perdu ou l'accusé, ou simplement un retard de l'un des deux. Si le délai d'attente est dépassé, il renvoie le paquet tout simplement. Pour gérer ces échanges dans le temps, il suffit de mettre en place un système de *compte à rebours* qui interrompt les envois de données après l'écoulement d'un certain temps. De plus, les accusés NAK ne sont plus nécessaires, si un paquet reçu par le destinataire est corrompu, il le jette tout simplement, sans envoyer d'accusé de réception.

L'existence de paquets dupliqués ou perdu provoquent quelques complications. Lorsque l'expéditeur reçoit un accusé de réception, il n'est pas en mesure de déterminer s'il s'agit d'une réponse au dernier paquet de données qu'il a transmis ou d'une réponse tardive à un paquet plus ancien. Il faut donc doter l'accusé d'un signe qui le distingue des autres, par exemple le numéro de séquence du paquet concerné.



Pôle expéditeur de `rdt3.0`



Pôle destinataire de `rdt3.0`

3.4.4 Transfert de données fiable avec pipelinage

Le protocole `rdt3.0` est un protocole tout à fait correct et exploitable, cependant il est tout sauf performant ! Et ce à cause de son caractère *stop-and-wait*. Pour illustrer cela, prenons l'exemple de deux terminaux situés aux extrémités Est ou Ouest des Etats-Unis. Le temps de propagation aller-retour entre ces deux serveurs (un RTT) à la vitesse de la lumière (ce qui est une supposition un petit peu trop optimiste) est de l'ordre de 30 ms. Supposons qu'ils soient reliés par un canal d'un débit R de 1 Gbit/s, et fixons la taille des paquets (en-tête comprise) à 1000 octets. Nous obtenons

$$t_{\text{trans}} = \frac{L}{R} = \frac{8000 \text{ bits}}{10^9 \text{ bits/sec}} = 8 \ \mu\text{s}$$

Il faut donc 15.008 ms pour que le paquet atteigne le destinataire qui peut alors émettre un ACK (que l'on supposera de taille négligeable) arrivant 15 ms plus tard, soit au temps $t = 30.008$ ms. C'est seulement à ce moment qu'un second paquet peut être envoyé. On voit que le taux d'utilisation de la connexion est donné par

$$U = \frac{\frac{L}{R}}{RTT + \frac{L}{R}} = \frac{0.008}{30.008} = 0.00027 = 0.027\%$$

Ce qui correspond à un taux de transfert effectif de 267 kbit/s sur une ligne proposant 1 Gbit/s !

Il existe une solution à ce problème : l'expéditeur peut envoyer plusieurs paquets avant de se mettre en attente. Cette approche est connue sous le nom de *pipelinage*. Cela implique divers changements dans les protocoles. Premièrement, la gamme des numéros de séquence doit être augmentée pour pouvoir identifier les paquets malgré que certains (ou même beaucoup) se soient perdus ou arrivent dans un ordre arbitraire. Secondement, il se peut que les pôles expéditeurs et destinataires des protocoles aient à placer plusieurs paquets à la fois dans leur mémoire tampon (l'expéditeur doit le faire au moins avec les paquets qui n'ont pas encore reçu d'accusé). Il existe pour cela plusieurs méthodes.

Go-Back-N

Dans ce protocole, l'expéditeur est autorisé à transmettre plusieurs paquets à la suite dès leur disponibilité, sans attendre d'accusé de réception, mais tout en respectant un nombre limité N de paquets non confirmés dans le canal. On peut représenter ce fonctionnement par une *fenêtre* de taille fixe N appliquée à la gamme des numéros de séquence. Lors de l'exécution du protocole, cette fenêtre se déplace vers l'avant, c'est pourquoi on parle de *protocole à fenêtre glissante*.

Un expéditeur Go-Back-N (GBN) doit pouvoir réagir à trois types d'événements : invocation de la couche supérieure (vérification si la fenêtre est pleine, génération et envoi de paquet), réception d'un ACK (accusés de réception cumulatifs : si un accusé n arrive, cela signifie que $n-1, n-2, \ldots$ sont arrivés également, même si les accusés ne sont pas parvenus à l'expéditeur), expiration du délai imparti (après dépassement du temps imparti, tous les paquets non confirmés font l'objet d'une retransmission, le compte à rebours étant basé sur l'envoi du plus ancien paquet non confirmé). Du point de vue du destinataire, c'est très simple : il ne garde que le paquet qu'il attend et génère un ACK du dernier paquet correctement reçu à chaque réception.

Répétition sélective

GBN souffre malgré ses améliorations d'encore quelques problèmes de performances. En effet, la moindre erreur dans un paquet peut entraîner la retransmission superflue d'une grande quantité de paquets pourtant arrivés intègres. Les protocoles de *répétition sélective* évitent les retransmissions inutiles et demandent à l'expéditeur de ne retransmettre que les paquets susceptibles d'avoir été perdus ou corrompus. Cela implique que le destinataire envoie un ACK pour chaque paquet correctement reçu. Comme dans GBN, une fenêtre permet de limiter le nombre de paquets en attente de confirmation. En revanche certains paquets de cette fenêtre auront déjà fait l'objet d'un accusé de réception. Le destinataire renvoie un accusé pour chaque paquet valide qu'il reçoit, qu'il soit dans l'ordre ou non, et stocke dans un tampon ceux qui sont encore temporairement hors séquence (notons que ce tampon ne dépassera jamais la taille de la fenêtre d'envoi de l'expéditeur). Afin d'éviter les problèmes de confusions avec les numéros de séquence, la quantité de numéros disponible doit être au moins deux fois plus grande que la taille de la fenêtre.

3.5 Transport orienté connexion : TCP

La première étape est d'organiser un échange de présentations. Les processus s'envoient donc des segments préliminaires qui définissent les paramètres du transfert. Le processus initiant la connexion est le processus *client* et l'autre le processus *serveur*. Le processus d'application client informe d'abord la couche transport du client qu'il souhaite établir une connexion avec un processus situé sur un autre serveur.

La couche transport du client procède alors à l'établissement de la connexion TCP avec le protocole TCP du serveur. Le client envoie tout d'abord un segment TCP spécial

auquel le serveur répond par l'envoi d'un autre segment spécial, entraînant l'envoi d'un troisième par le client. Les deux premiers ne véhiculent aucune donnée de la couche application, mais le troisième peut en contenir.

La connexion TCP étant établie, les processus TCP peuvent s'envoyer des données. Le premier envoie un flux de données au travers de l'interface de connexion, puis les données sont prises en charge par le TCP du serveur client, ce dernier les dirige vers la mémoire tampon d'envoi de la connexion. De temps à autre, TCP s'empare de groupes de données contenues dans ce tampon pour en former un *segment TCP*, constitué d'un en-tête et des données, qui est remis à la couche réseau. A l'autre pôle de TCP, ces données sont placées dans un tampon de réception de la connexion TCP. L'application retire ainsi le flux de données qui lui sont destinées.

TCP, tout comme `rdt3.0`, a recours à un système de compte à rebours et de retransmission, mais des détails s'y ajoutent. Il faut en particulier décider d'une bonne estimation du RTT pour l'appliquer concrètement. TCP évalue le RTT d'un segment (variable `SampleRTT`) comme le temps écoulé entre l'envoi du segment et le retour de l'accusé. Plutôt que de le calculer pour chaque segment, TCP n'en mesure qu'un à la fois, ce qui le conduit à une nouvelle valeur à peu près à chaque fois que s'écoule un laps de temps correspondant au RTT. Les valeurs de `SampleRTT` varient de segment en segment, il est donc difficile d'en faire une bonne estimation ; par contre sa moyenne `EstimatedRTT` est aisément calculable selon l'idée suivante : A l'arrivée d'un nouveau `SampleRTT`, TCP réévalue `EstimatedRTT` :

$$\texttt{EstimatedRTT} = (1 - \alpha) \cdot \texttt{EstimatedRTT} + \alpha \cdot \texttt{SampleRTT}$$

Cette formulation accorde plus de poids aux éléments récents qu'aux plus anciens, on parle de *moyenne pondérée exponentielle flottante*. Il est également utile de disposer d'une mesure du coefficient de variation du RTT que l'on appelle `DevRTT` :

$$\texttt{DevRTT} = (1 - \beta) \cdot \texttt{DevRTT} + \beta \cdot |\texttt{SampleRTT} - \texttt{EstimatedRTT}|$$

Une fois que l'on connaît ces deux valeurs, une valeur judicieuse pour le compte à rebours est donnée par :

$$\texttt{TimeoutInterval} = \texttt{EstimatedRTT} + 4 \cdot \texttt{DevRTT}$$

Du point de vue du transfert de données fiable, TCP fonctionne à peu près comme Go-Back-N.

La temporisation ne va pas se faire seulement avec les estimateurs comme suggéré ci-dessus. Une première variante consiste à doubler le temps de l'intervalle à chaque retransmission plutôt que de le redéfinir sur base des variables `SampleRTT` et `EstimatedRTT`. Cette variante se traduit par un début de contrôle de congestion (qui sera vu en détail plus loin). Une seconde variante propose une solution au problème de temps souvent trop long des délais imposés, en effet, TCP est capable de déceler un

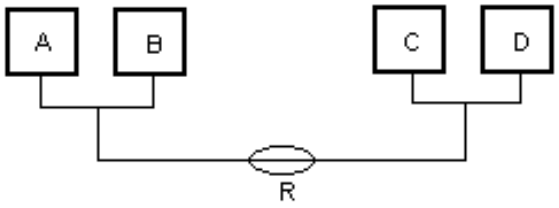
paquet perdu lorsqu'il reçoit des accusés dupliqués (ce qui signifie qu'un paquet $n + i$ ($i > 0$) a abouti alors que le paquet n était attendu). Dans la mesure ou l'expéditeur émet un grand nombre de paquets, trois accusés dupliqués successifs conforteront la thèse que le paquet en attente de confirmation s'est probablement perdu. TCP procède alors à une *retransmission rapide*.

TCP procure un service de contrôle de flux, c'est-à-dire s'assurer que l'expéditeur n'envoie pas ses messages trop rapidement pour le destinataire (à ne pas confondre avec le contrôle de congestion qui lui est lié à l'encombrement du réseau entre les deux terminaux). Le principe est simple, pour ne pas que l'expéditeur sature le tampon de réception du destinataire, celui-ci l'informe de l'espace disponible dans son tampon en insérant cette variable dans la fenêtre de réception de tous les segments qu'il lui envoie. Lorsque le tampon de réception est saturé, l'expéditeur se contente alors d'envoyer des *ping* à l'aide de segment contenant un seul octet qui n'a pas d'importance, dans le seul but de recevoir une réponse avec les nouvelles informations sur l'état du tampon.

3.6 Contrôle de congestion

3.6.1 Les causes et effets du phénomène de congestion

Considérons différentes situations, de complexité croissante. Le cas le plus simple concerne deux hôtes A et B connectés au moyen d'une liaison en un simple bond entre la source et le destinataire. On suppose que A et B émettent tous les deux au même débit λ_{in} octet/s. Nous supposons également que les paquets en provenance de A et B traversent un routeur et une liaison de sortie partagée de capacité R. Dans ce premier cas nous supposons que la quantité de tampons dont dispose le routeur est illimitée. Pour un débit d'émission entre 0 et $R/2$, le débit au niveau des destinataires est égal au débit d'émission ; en revanche, pour un débit d'envoi supérieur à $R/2$, le débit au niveau du destinataire ne dépassera pas $R/2$. un débit d'émission élevé est une bonne chose pour l'utilisation de tout le potentiel de la bande passante, cependant ça s'avère être très mauvais pour le temps de réponse, qui tend vers l'infini lorsque le débit tend inférieurement vers $R/2$, par le fait qu'une file d'attente de plus en plus grande se présente à l'entrée du routeur.



Dans un second cas, nous supposons que les routeurs ne disposent que d'un tampon limité, ce qui est le cas en réalité. Une conséquence est la perte des paquets arrivant

lorsque le tampon est plein. Il y a cette fois des précisions à apporter à la notion de *taux d'envoi*. Puisque certains paquets, n'arrivent plus, certains seront renvoyés par la couche transport, ce qui fait donc une différence entre le taux d'envoi λ_{in} de la couche application et le taux d'envoi λ'_{in} de la couche transport, que l'on nomme *charge offerte au réseau*. Si A avait le pouvoir magique de savoir quand le tampon du routeur est saturé, il n'y aurait aucune perte, avec pour conséquence $\lambda'_{\text{in}} = \lambda_{\text{in}}$, avec un taux d'envoi maximum de $R/2$. Si l'expéditeur ne renvoie un paquet que s'il est sûr qu'il s'est perdu, nous pouvons observer que λ_{out} tend vers $R/3$ lorsque λ'_{in} tend vers $R/2$.

Un troisième cas de figure met en présence différents routeurs et différentes connexions qui se partagent les routeurs. Tant que les débits d'émission restent faibles, il y a vraiment peu de chance qu'un routeur perde un paquet, mais si les débits d'émission deviennent élevés on assiste à un phénomène particulier et destructeur pour l'efficacité du réseau : plusieurs types de trafic peuvent entrer en compétition pour l'espace tampon d'un même routeur, et apparaît un travail superflu effectué par le réseau. En effet,en cas d'un grand volume de trafic, les routeurs du réseau de l'amont transmettent des paquets qui risquent d'être perdus par un de leurs homologues en aval, et ils travaillent donc pour rien, saturant leur tampon et empêchant des paquets pour lesquels la suite du réseau est dégagée de passer. Le réseau aurait été mieux géré si le premier routeur avait tout de suite rejeté le paquet et était resté passif ! C'est donc là une des grandes causes de congestion d'un réseau : *lorsqu'un paquet est perdu sur son parcours, la capacité de transmission dépensée par chacun des routeurs en amont pour le faire progresser s'en trouve perdue.*

3.6.2 Contrôle de congestion de TCP

L'approche de TCP consiste à faire en sorte que chaque expéditeur régule sa vitesse d'envoi en fonction du niveau de congestion perçu au sein du réseau. Le phénomène de perte (qui est la principale conséquence d'une congestion sur le réseau) est identifié soit par l'expiration du temps imparti, soit par la réception de trois accusés de réception identiques de la part du destinataire. L'algorithme de contrôle de congestion de TCP présente trois composantes principales : Un accroissement additif et une décroissance multiplicative, un départ lent et une réaction au phénomène d'expiration du temps imparti.

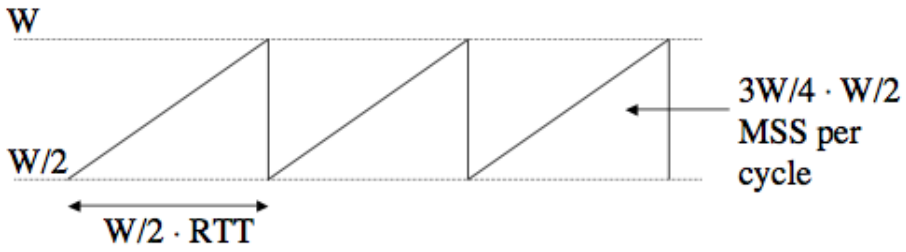
Avec le contrôle de congestion de TCP, chaque pôle maintient une variable dite *fenêtre de congestion* appelée `CongWin`, qui impose une limite au rythme auquel l'expéditeur est autorisé à charger des données sur le réseau. TCP a recours à une approche décroissante multiplicative, c'est-à-dire que la valeur `CongWin` est diminuée de moitié après chaque phénomène de perte, avec une limite en la valeur de 1 MSS. Une fois le phénomène de congestion résorbé, TCP doit pouvoir agir pour augmenter le taux d'envoi. TCP procède donc à un agrandissement progressif linéaire de sa fenêtre de congestion. Cet algorithme est appelé *algorithme AIMD* (*Additive Increase, Multiplicative Decrease*).

Au début d'une connexion, la variable `CongWin` a une valeur de un MSS, ce qui fait un taux d'envoi de MSS/RTT. Mais le réseau dispose peut-être d'un débit bien plus important, et il serait idiot de faire une progression linéaire jusqu'à cette valeur (ce qui peut prendre beaucoup de temps sur une connexion disposant d'un très bon débit). L'expéditeur aura donc recours à une accélération exponentielle en doublant la valeur de `CongWin` à chaque RTT, et ce jusqu'aux premiers phénomènes de perte, moment auquel il divise brusquement `CongWin` par deux et passe au mode de progression linéaire décrit ci-dessus. Cette phase initiale est appelée *départ lent*.

Il reste un point important à détailler : TCP fait une distinction entre un phénomène de perte signalé par trois accusés identiques (ce qui indique une perte de seulement quelques paquets sur le réseau) et un phénomène de perte décelé par expiration du temps imparti (qui laisse plus croire à une grosse congestion du réseau qui aurait perdu tout ce qui a été envoyé). Dans le premier cas, il se contente de diviser la fenêtre de congestion par deux, et de continuer une progression linéaire, comme nous l'avons déjà précisé (on parle de *récupération rapide*). Dans le second cas, l'expéditeur revient au départ lent, ramenant la taille de la fenêtre de congestion à un seul MSS. De plus TCP garde le contrôle sur les processus au moyen d'une variable `Seuil`, qui détermine la taille que doit atteindre la fenêtre avant que ne cesse la phase de départ lent et ne commence la phase d'évitement de congestion. Cette variable prend initialement une valeur élevée (65 Ko en pratique), et se trouve divisée par deux lors des phénomènes de perte.

L'algorithme de contrôle de congestion de TCP opère donc de la façon suivante :
- Tant que la taille de la fenêtre de congestion est inférieure à la valeur du seuil, l'expéditeur est en phase de départ lent et la fenêtre connaît une croissance exponentielle.
- Une fois la valeur du seuil dépassée, l'expéditeur entre dans la phase d'évitement de congestion durant laquelle la fenêtre s'accroît de manière linéaire.
- Lorsqu'il y a trois accusés de réception identiques, la valeur du seuil est réglée à la moitié de la taille de la fenêtre de congestion en cours et cette dernière est portée à la valeur du seuil.
- Lors de l'expiration du temps imparti, la valeur du seuil est réglée à la moitié de la fenêtre de congestion en cours et cette dernière est portée à 1 MSS.

Le débit à un instant donné sera `CongWin`/RTT, mais il est continuellement en dents de scie, il est donc naturel de se demander quel est le *débit moyen* d'une connexion TCP sur une durée plus longue. Notons w la taille de la fenêtre de congestion (en octets) et un temps aller-retour de RTT secondes. Le débit est approximativement w/RTT. Appelons W la valeur atteinte par w lorsqu'un phénomène de perte se présente. En partant du principe que RTT et W sont relativement constants dans le temps, le débit varie entre $\frac{1}{2}W$/RTT et W/RTT. Le débit moyen sur la connexion sera donc à peu près $\frac{3}{4}W$/RTT.



La quantité de données est donc de $\frac{3}{4}W \cdot \frac{1}{2}W = \frac{3}{8}W^2 = 1/p$ où p est le taux moyen de perte de paquets. Ainsi, nous pouvons écrire $W = \sqrt{\frac{8}{3p}}$. Nous pouvons écrire le débit moyen en bytes par seconde comme $B = \frac{3}{4}W/\text{RTT} = \sqrt{\frac{3}{2p}}\frac{\text{MSS}}{\text{RTT}}$

Chapitre 4

La couche réseau

4.1 Modèles de services

A la différence de la couche transport, on trouve un élément de la couche réseau au sein de chaque serveur et de chaque routeur du réseau, c'est une couche qui s'étend donc à tout le réseau.

Tout comme nous parlions de messages pour la couche application ou de segments pour la couche transport, la couche réseau véhicule des *datagrammes* de routeur en routeur.

La principale fonction des routeurs est d'assurer le *routage* des datagrammes entre leurs liaisons d'entrée et de sortie. Le rôle principal de la couche réseau est donc extrêmement simple : il consiste à déplacer des paquets d'un serveur expéditeur à un serveur destinataire, et ce en trois étapes :

- Choix de l'itinéraire à emprunter (algorithmes de routage) ;
- Réexpédition (lorsqu'un routeur reçoit un paquet il doit le faire suivre jusqu'à la liaison de sortie appropriée) ;
- Etablissement de l'appel (tout comme TCP établit une connexion avant le transfert des informations, certaines architectures de la couche réseau imposent aux routeurs de s'échanger des informations d'état avant le transfert des données proprement dites ; on parle d'*établissement d'appel*).

L'une des premières questions à se poser est *datagrammes ou circuits virtuels ?*.
Un réseau à commutation par paquet à circuits virtuels se compose de façon très semblable à un réseau téléphonique, qui est lui doté de circuits réels. L'utilisation d'un circuit virtuel se déroule en trois grandes étapes :

- *Etablissement du circuit virtuel.* Durant cette phase, l'expéditeur contacte la couche réseau et lui fournit l'adresse du destinataire. La couche réseau détermine le parcours à prendre (ce qui implique une mise à jour des tables de routage de chacun des commutateurs sur le parcours, et une éventuelle réservation des ressources).
- *Transfert de données.*
- *Fermeture du circuit virtuel.* Le circuit est libéré lorsque l'un des deux systèmes le

demande. Elle informe alors l'autre système et procède à l'actualisation des tables de routage.

On parle aussi de *service réseau orienté connexion.*

En présence d'une couche réseau à *datagrammes*, un système transmettant des données doit préciser l'adresse du destinataire à l'intérieur de chaque datagramme car les commutateurs d'un réseau à datagrammes ne conserveront aucune information. La redirection des datagrammes se fait via leur table de routage. On parle de *service réseau sans connexion.*

4.2 Les principes du routage

Pour transférer des paquets de la couche transport d'un serveur à l'autre, la couche réseau doit tout d'abord déterminer le parcours à emprunter. Un serveur est généralement directement connecté à un routeur spécifique (routeur de premier bond), de même pour le destinataire. Le problème revient donc à acheminer un paquet entre ces deux routeurs.

Un *algorithme de routage global* serait capable de déterminer le parcours le moins onéreux entre la source et la destination au moyen d'une connaissance globale du réseau, c'est-à-dire de connaissances complètes du réseau et des coûts associés aux liaisons. Ce n'est évidemment pas imaginable de façon réellement globale au sein de réseaux trop grands. Dans la mesure où les algorithmes doivent être informés de la valeur de chaque liaison du réseau, on parle d'*algorithme d'état de lien.*

Avec un *algorithme de routage décentralisé*, le calcul du parcours le moins cher se fait de manière itérative et distribuée. Aucun noeud ne connaissant le coût de toutes les liaisons, le calcul se fait au niveau de chaque noeud en partant de leurs connaissances locales, puis au moyen d'un calcul par itération. Grâce à un échange d'informations avec les voisins, chacun parvient à évaluer progressivement le parcours le moins onéreux vers une destination ou un groupe donné de destinations. Aucun noeud n'est en mesure de connaître l'ensemble du parcours, par contre il est à même de déterminer à quel voisin envoyer le datagramme pour minimiser le coût général.

Algorithme de routage par état de lien

Un algorithme par état de lien utilise la connaissance de la topologie du réseau et du coût des différentes liaisons pour ses calculs. Par ce protocole, chaque noeud du réseau diffuse à l'ensemble de ses homologues l'identité et la valeur des liaisons auxquelles il est rattaché. Cette *diffusion d'état de lien* peut s'effectuer sans que les différents liens aient initialement connaissance de l'existence de tous les noeuds du réseau, un noeud donné doit simplement connaître l'identité de ses voisins directs ainsi que la valeur de la liaison y conduisant ; les autres informations lui seront transmises en retour. Ainsi, tous les noeuds disposent des mêmes informations sur le réseau, chacun pouvant appliquer l'algorithme individuellement.

L'algorithme utilisé est l'*algorithme de Dijkstra*, qui détermine le parcours le moins onéreux entre un noeud et tous les autres noeuds du réseau (voir documentation plus précise pour les détails de l'algorithme, page 323 dans [1]). Une fois l'algorithme effectué, le prédécesseur de chaque noeud sur le parcours le moins onéreux partant du noeud d'origine est identifié. Connaissant également les prédécesseurs des prédécesseurs, il est possible de baliser progressivement le parcours dans son intégralité. Il s'agit d'un algorithme de routage en $O(n^2)$

Algorithme de routage à vecteur de distance

L'algorithme à vecteur de distance est de nature itérative, asynchrone et distribuée : les calculs se font au niveau de chaque noeud individuel, à partir des informations fournies par les noeuds voisins directs, et les résultats sont partagés de la même manière, de plus il n'impose pas du tout que les noeuds travaillent ensemble.

La principale structure de données utilisée par cet algorithme est constituée d'une *table des distances* tenue à jour au niveau de chaque noeud. Cette table contient une ligne pour chaque destination possible au sein du réseau et une colonne pour chacun des voisins directs. Imaginons un noeud X souhaitant acheminer un paquet vers une destination Y via son voisin direct Z. L'entrée de la table des distances du noeud X associée est $D^X(Y, Z)$ (*atteindre* Y *par* Z *depuis* X), elle correspond à la somme du coût $c(X, Z)$ de la liaison directe entre X et Z et le coût $D^X(Y, Z)$ de la meilleure liaison entre Z et Y ; ce qui conduit à l'expression

$$D^X(Y, Z) = c(X, Z) + \min_W\{D^Z(Y, W)\}$$

dans laquelle le terme W est choisi parmi tous les voisins directs de Z (y compris X).

Chaque noeud doit ainsi connaître la valeur du parcours le moins onéreux de tous ses voisins pour toutes les destinations possibles. Dès qu'un noeud trouve un nouveau parcours moins onéreux vers une destination donnée, il est chargé d'en informer tous ses voisins.

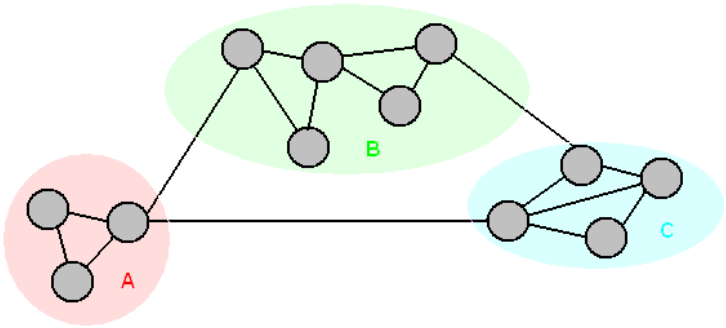
Lorsqu'un noeud détecte un changement dans la valeur de l'une de ses liaisons, il actualise sa table des distances et il en informe ses voisins si nécessaire. Dans le cas d'une bonne nouvelle, elle se transmet très rapidement : la nouvelle valeur restreignant le coût de parcours s'installe directement dans les routeurs avoisinant. Mais s'il s'agit d'une mauvaise nouvelle, le processus d'information est très mauvais et très lent : le nouveau chemin plus coûteux ne s'installera qu'itération après itération, ce qui peut prendre un temps très long si la nouvelle valeur est grande et que le nouveau vrai chemin le moins coûteux est également très coûteux. Ce phénomène porte le nom de *problème de la valeur infinie*. Il peut être évité en ayant recours à une technique connue sous le nom de *retour pollué*. L'idée est la suivante : Si un routeur Z passe par Y pour atteindre X, Z peut faire croire à Y qu'il se trouve à une distance infinie de X. Y étant alors persuadé que Z ne peut atteindre X, il opte pour une autre route. Cependant cette ruse ne résout pas

le problème pour les boucles de plus de trois routeurs voisins, et ne met pas le réseau à l'abri des boucles de routage.

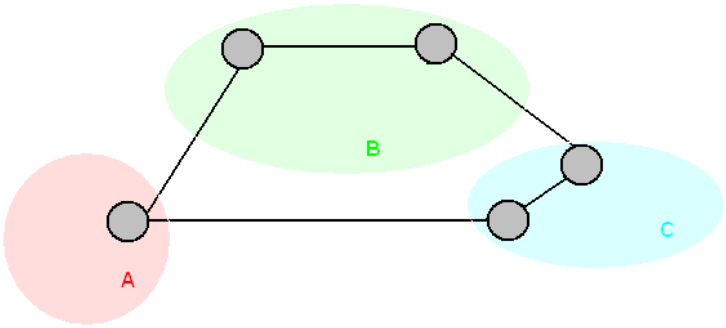
4.3 Routage hiérarchique

Jusqu'à présent nous avons mis tous les routeurs sur un même pied d'égalité, ce qui est un peu réducteur pour des raisons évidentes de dimensions (stockage d'information pour plusieurs centaines de millions de routeurs) et d'administration (chacun gère son routeur comme il l'entend). Les routeurs vont donc s'organiser en régions ou *systèmes autonomes* (AS, Autonomous Systems), les différents routeurs d'un même système utilisant le même algorithme et disposant de certaines informations l'un sur l'autre. On parle alors de *protocole de routage interne*. Certains routeurs doivent alors servir d'interface avec le reste du réseau (*routeurs frontaliers*) où un *protocole de routage externe* est mis en oeuvre. Ceci résout déjà fortement le problème de dimension qui était présent dans le modèle précédent.

Considérons par exemple trois systèmes autonomes A, B, C, munis chacun de plusieurs routeurs, mis en présence.



D'un point de vue externe, le réseau se résume à un réseau constitué de cinq routeurs :

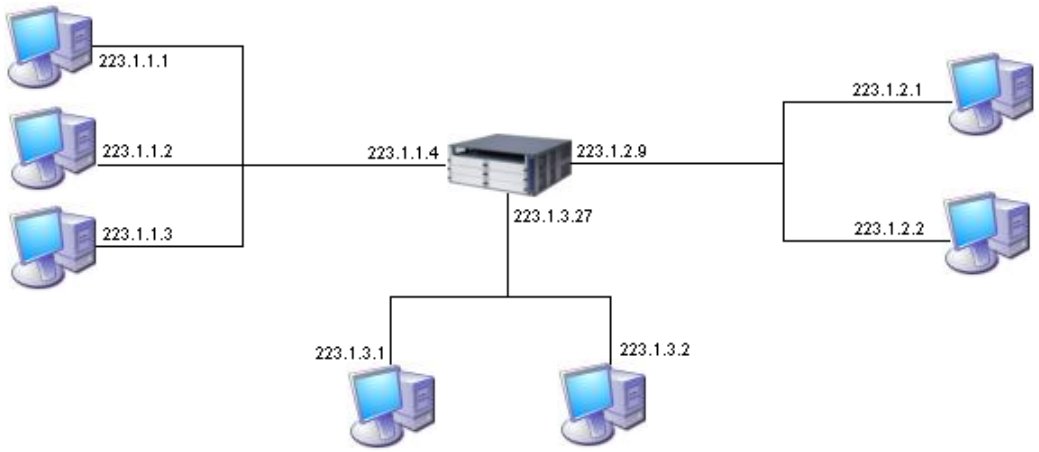


4.4 Protocole Internet (IP)

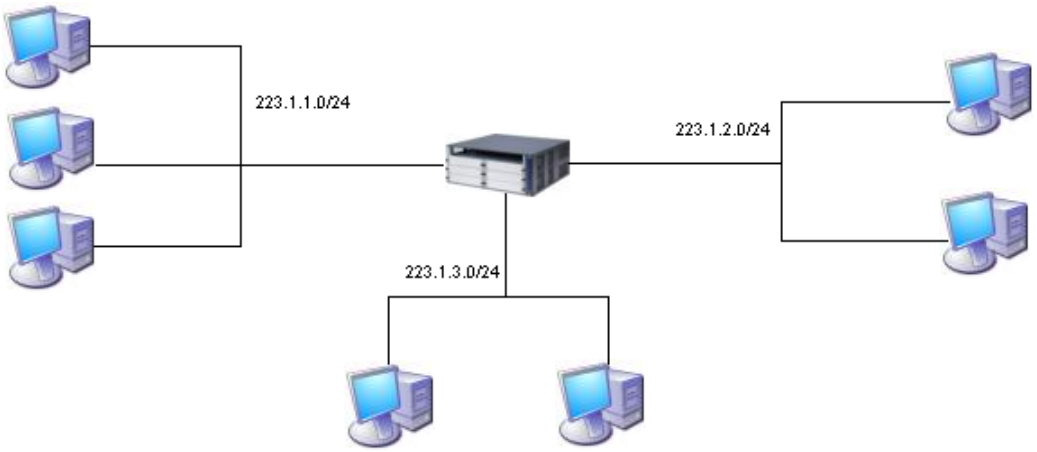
4.4.1 Adressage

Un hôte n'est généralement connecté au réseau que par l'intermédiaire d'une simple liaison qui représente la seule voie d'accès : un hôte et sa liaison coïncident au niveau de leur *interface réseau*. Un routeur possède ainsi autant d'interfaces que de liaisons, une *adresse IP* unique sera attribuée à chacune de ces interfaces. Une adresse IP a une taille fixe de 32 bits, ce qui permet 2^{32} adresses différentes. On l'exprime généralement sous une forme 123.456.789.12.

Ces adresses ne peuvent pas être distribuées au hasard. En effet, une portion de l'adresse sera déterminée par le *sous-réseau* auquel elle est physiquement reliée. La figure suivante en donne un exemple : un routeur doté de trois interfaces sert à l'interconnexion de sept hôtes. Les trois hôtes se situant sur l'interface gauche partagent tous des adresses IP dont les 24 premiers bits sont identiques. Les interfaces de ces trois hôtes forment donc un *réseau IP*.



Les 24 premiers bits de ces adresses IP constituent la partie réseau de leur adresse, les 8 bits restants forment à *partie interface*. Un réseau possède lui aussi son adresse IP, bien qu'il regroupe plusieurs adresses. On a pour cela recours à un *masque de sous-réseau*. Par exemple, le réseau de gauche sera identifié par l'adresse 223.1.1.0/24 où "/24" est le masque de sous-réseau qui indique que les 24 premiers bits de l'adresse définissent l'adresse du réseau (on parle aussi de *préfixe réseau*). Toute entité venant se raccorder au réseau 223.1.1.0/24 est tenue d'avoir une adresse IP du type 223.1.1.xxx.



L'identification des différents réseaux au sens de l'IP dans un ensemble de routeurs et d'hôtes interconnectés s'effectue selon la règle suivante : dissocier dans un premier temps chaque interface du routeur ou hôte auquel elle appartient. Ceci engendre des îlots de réseaux isolés limités par des interfaces. On appelle ensuite *réseau IP* chacun de ces réseaux isolés. La normalisation du CIDR (*Classless InterDomain Routing*) permet de diviser l'adresse en une composante réseau et une composante hôte de taille quelconque (plutôt que la limite standard de 8,16 ou 24 bits). Une adresse réseau CIDR se présente sous la forme $a.b.c.d/x$ dans laquelle x indique le nombre de bits constituant la composante réseau. Par exemple, $a.b.c.d/21$ identifiera un réseau pouvant contenir jusqu'à 2048 interfaces. Dans la pratique, un tel réseau peut encore être subdivisé en sous-réseaux internes.

Pour pouvoir établir un nouveau parc d'ordinateurs, il faut tout d'abord obtenir de son fournisseur d'accès un contingent d'adresses ; par exemple 200.23.16.0/20. Lorsqu'une institution a reçu son contingent d'adresses, elle peut assigner des adresses IP individuelles aux différentes interfaces routeur et hôte constituant son réseau local. Ce procédé peut se faire manuellement par l'administrateur, mais il existe le protocole DHCP (*Dynamic Host Configuration Protocol*) permettant à un hôte d'obtenir une adresse IP de manière automatique, ainsi que certaines informations utiles comme l'IP du routeur de premier bond et celle de son serveur DNS (ce protocole sera vu en détail un peu plus loin).

4.4.2 Transport d'un datagramme

Sommairement, un *datagramme* est constitué de champs divers, suivis par l'adresse IP de la source et l'adresse IP de destination, et enfin les données qui sont généralement un segment TCP ou UDP. Après création du datagramme, la couche réseau doit assurer son transfert jusqu'à l'hôte destinataire. Le mode opératoire de transfert est différent selon que les deux hôtes concernés sont sur le même réseau ou non.

Si, pour reprendre notre exemple, l'hôte dont l'interface est 223.1.1.1 (appelons-le A) souhaite envoyer un datagramme à l'hôte B dont l'interface est 223.1.1.3 (donc sur le même réseau 223.1.1.0/24), le processus est le suivant : L'IP de A consulte tout d'abord sa table de routage interne et y décèle une entrée 223.1.1.0/24 dont l'adresse réseau correspond aux 24 premiers bits du son destinataire B. La table de routage indique que le nombre de bonds en direction du réseau 223.1.1.0 est 1, ce qui signifie que B est sur le même réseau que A, ce dernier est donc directement accessible par son interface de sortie, sans avoir à passer par un routeur intermédiaire. Il remet alors le datagramme au protocole de couche liaison de l'interface qui sera responsable du transfert jusqu'à B.

Cette fois A souhaite envoyer un datagramme à l'hôte C d'interface 223.1.2.2 situé sur le réseau 223.1.2.0/24. A consulte sa table de routage IP et y trouve une entrée dont l'adresse réseau correspond aux premiers bits d'adresse de C. Il s'agit d'un chemin à 2 bonds, A sait que son datagramme est destiné à un réseau différent du sien et qu'il doit passer par un routeur intermédiaire. La table de routage l'informe également que pour atteindre son destinataire il doit tout d'abord envoyer le datagramme à l'adresse IP 223.1.1.4, ce qui est effectué par la couche liaison. Le datagramme se trouve ensuite au niveau du routeur qui est chargé de le rediriger vers sa destination finale. Celui-ci assure cette fonction en consultant sa table d'acheminement et en y repérant l'entrée 223.1.2.0/24 indiquant que le datagramme doit être transmis à l'interface de routeur 223.1.2.9. Puisqu'il n'y a plus qu'un seul bond jusqu'à la destination, le routeur sait que C se trouve sur le même réseau que cette interface. Il remet donc le datagramme à cette interface qui le transmet à sa destination finale grâce à la couche liaison.

Dans notre exemple, la table d'acheminement se présente comme

Destination	Routeur suivant	Nbr de bonds	Interface
223.1.1.0/24	-	1	223.1.1.4
223.1.2.0/24	-	1	223.1.2.9
223.1.3.0/24	-	1	223.1.3.27

4.4.3 Structure d'un datagramme

Un datagramme IPv4 est constitué de nombreux champs :

- *Version.* Quatre bits pour spécifier la version du protocole IP.
- *Taille de l'en-tête.* Quatre bits pour spécifier la taille de l'en-tête, qui est généralement de 20 octets.
- *Type de service.* Huit bits pour différencier les types de datagrammes.
- *Taille du datagramme.* Seize bits pour spécifier la taille totale du datagramme IP (en-tête comprise) exprimée en octets. La taille maximale d'un datagramme est donc de 65535 octets, mais dépasse rarement 1500 octets, et est souvent limitée à 576 octets.
- *Identifiant,fanions,déplacement.* Trois champs liés au phénomène de fragmentation.

- *Durée de vie.* Le champ TTL (*Time To Live*) de huit bits permet d'éviter que des datagrammes ne circulent de façon ininterrompue à l'intérieur d'un réseau où ils se seraient perdus (par exemple dans une boucle de routage). Cette valeur est décrémentée à chaque fois que le datagramme est pris en charge par un routeur ; arrivé à 0 le datagramme est supprimé.
- *Protocole de couche supérieure.* Champ de huit bits utilisé uniquement lorsqu'il arrive à destination pour indiquer à quel protocole de la couche transport il doit être confié.
- *Somme de contrôle de l'en-tête.* Champ de deux octets, somme de contrôle des autres mots de 2 octets de l'en-tête. Tout datagramme possédant une erreur est généralement supprimé par le routeur qui la détecte.
- *Adresse IP source.* Trente-deux bits.
- *Adresse IP destinataire.* Trente-deux bits.
- *Options.* Ce champ permet d'étendre l'en-tête d'un datagramme si besoin est.
- *Données.*

4.4.4 Fragmentation

Le volume de données maximal qui peut être porté par un paquet de la couche liaison est appelé MTU (*Maximum Transfert Unit*). Les datagrammes se voient naturellement soumis aux mêmes contraintes de taille. Cela ne présente pas de problème, sauf lorsque les protocoles utilisés pas les liaisons composant le trajet de bout-en-bout présentent des MTU différents. Lorsqu'un datagramme possède une dimension plus grande que le MTU de la liaison sur laquelle il doit être envoyé, il est *fragmenté* en datagrammes plus petits, et les fragments sont expédiés les uns après les autres. Avant d'être transmis à la couche transport du destinataire, les fragments doivent être réassemblés, car TCP et UDP n'admettent que des segments entiers. Un réassemblage et une nouvelle fragmentation à chaque routeur entraînerait des complications, c'est pourquoi IPv4 confie ce travail au terminal qui recevra les fragments. Lorsqu'un routeur doit fragmenter un datagramme, il remplit les champs d'identification, de fanion et de déplacement, ce qui permettra d'identifier les différents fragments. Il est possible que des fragments se perdent en route, dans ce cas tout le fragment original est supprimé par le terminal.

4.4.5 Protocole ICMP

Le protocole ICMP (*Internet Control Message Protocol*) est utilisé par les serveurs, routeurs et passerelles pour échanger des informations de couche réseau, principalement des rapports d'erreurs. Bien que les messages soient véhiculés par des paquets IP, IMCP est souvent considéré comme partie intégrante de la couche IP, bien qu'il appartienne architecturalement à une couche supérieure. Lorsqu'un serveur reçoit un paquet IP annonçant ICMP (champ *Type de service*), il procède au démultiplexage en direction d'ICMP, de la même manière qu'en direction de TCP ou UDP.

4.4.6 Protocole DHCP

Nous savons déjà que le protocole DHCP (*Dynamic Host Configuration Protocol*) est souvent utilisé pour l'attribution dynamique d'adresses IP. Il s'agit d'un protocole client-serveur. Normalement chaque réseau est doté d'un serveur DHCP.

Ce protocole propose un processus en quatre étapes aux hôtes cherchant à se connecter à un réseau :

- Détection d'un serveur DHCP : Un nouvel arrivant envoie un *message de détection de DHCP*, via UDP, avec le numéro d'accès 67. Le nouvel arrivant ne connaissant aucune adresse IP du réseau, il l'envoie sur l'*adresse de diffusion* 255.255.255.255 avec pour adresse d'origine 0.0.0.0. Ce message sera reçu par toutes les interfaces du réseau, y compris celle(s) du/des serveur(s) DHCP.
- Proposition(s) du serveur DHCP : Le serveur DHCP répond d'un *message de proposition DHCP*, contenant une adresse IP, le masque de sous-réseau et une indication sur la durée de validité de l'adresse fournie.
- Requête DHCP : Le nouvel arrivant fait son choix parmi les propositions et y répond par une requête DHCP.
- Acquittement DHCP. Le serveur répond à cette requête au moyen d'un acquittement (ACK) qui confirme les paramètres sollicités.

Le client peut alors commencer à utiliser son adresse IP pendant la durée impartie.

4.4.7 Traducteur d'adresses NAT

Dès que l'on souhaite mettre en place un réseau local reliant différents ordinateurs d'un même espace de travail, il faut obtenir autant d'adresses IP auprès de son fournisseur d'accès. Mais que se passe-t-il s'il a déjà alloué les adresses contiguës à celles du réseau en extension à un autre client ? C'est pour régler ce genre de problème qu'existe la *traduction d'adresse réseau* (NAT, *network Address Traslation*).

La figure ci-dessous représente le fonctionnement d'un routeur NAT. Ce routeur est doté d'une interface appartenant au réseau domestique et une interface orientée vers l'Internet. Le type d'adressage s'effectuant au sein du mini-réseau est exactement le même que celui que nous avons déjà décrit : les quatre interfaces se partagent le réseau 10.0.0.0/24. Le routeur doté d'une fonction NAT se comporte vis-à-vis du monde extérieur comme un équipement isolé muni d'une seule adresse IP 138.76.29.7 et tout trafic entrant doit être doté de cette adresse de destination. Le routeur NAT obtient son adresse auprès de son FAI via DHCP et il a recours à un DHCP local pour l'attribution des adresses IP du mini-réseau.

Lorsqu'un datagramme arrive de l'extérieur, le routeur a besoin d'une table de traduction d'adresses pour savoir à quel hôte du mini-réseau le remettre. Il utilisera pour ça une astuce avec les numéros de port. Prenons l'exemple de l'hôte d'interface 10.0.0.1 qui sollicite une page Web (accès 80) sur l'interface 128.119.40.186. Le terminal 10.0.0.1 assigne à son datagramme le numéro d'accès d'origine (arbitraire) 3345 et l'envoie au travers du réseau local. Sur réception de ce datagramme, le routeur NAT remplace l'adresse IP de l'expéditeur (10.0.0.1) par son adresse orientée vers l'Internet (138.76.29.7), choisit un port arbitraire (5001 par exemple) nouvellement créé, retient ces correspondances et envoie le nouveau datagramme sur l'Internet. Le serveur Web renverra naturellement les données de la requête sur l'interface 138.76.29.7 et l'accès 5001, qui seront converties en 10.0.0.1 et 3345 par le serveur NAT et renvoyées sur le réseau local.

4.5 Routage au sein de l'Internet

On appelle *système autonome* tout groupe de routeurs placés sous le contrôle d'une même autorité administrative et technique et qui a recours au même protocole de routage. Les protocoles de routage internet se distinguent essentiellement par leur caractère interne ou externe.

Un protocole de routage interne sert à déterminer le mode de routage assuré au sein d'un système autonome. Les deux plus anciens protocoles sont RIP (*Routing Information Protocol*) et OSPF (*Open Shortest Path First*).

4.5.1 Le protocole RIP : informations de routage

RIP est un protocole à vecteur de distance très similaire à celui vu précédemment. Sa version originale a recours au nombre de bonds comme mesure du coût d'un chemin donné, attribuant à chaque liaison la même valeur unitaire, avec un coût par chemin ne pouvant dépasser 15 (ce qui limite RIP à des systèmes d'une portée inférieure à 15 bonds). Les informations échangées entre routeurs voisins ont lieu à peu près toutes les 30 secondes grâce au *message de réponse RIP*, contenant une liste pouvant aller jusqu'à 25 réseaux de destination au sein du système autonome.

Les tables de routage au sein des routeurs sont constituées de trois colonnes : le réseau de destination, l'identité du prochain routeur pour cette destination, et enfin le nombre de bonds jalonnant le chemin.

Les routeurs s'échangent des annonces environ toutes les 30 secondes. Si un routeur ne reçoit pas de nouvelles de la part d'un voisin au moins une fois toutes les 180 secondes, celui-ci est considéré comme hors de portée, la table est alors modifiée et les voisins informés des modifications.

S'il le souhaite un routeur peut interroger directement ses voisins au sujet du coût de ses chemins vers une destination donnée.

4.5.2 Le protocole OSPF : priorité au plus court chemin

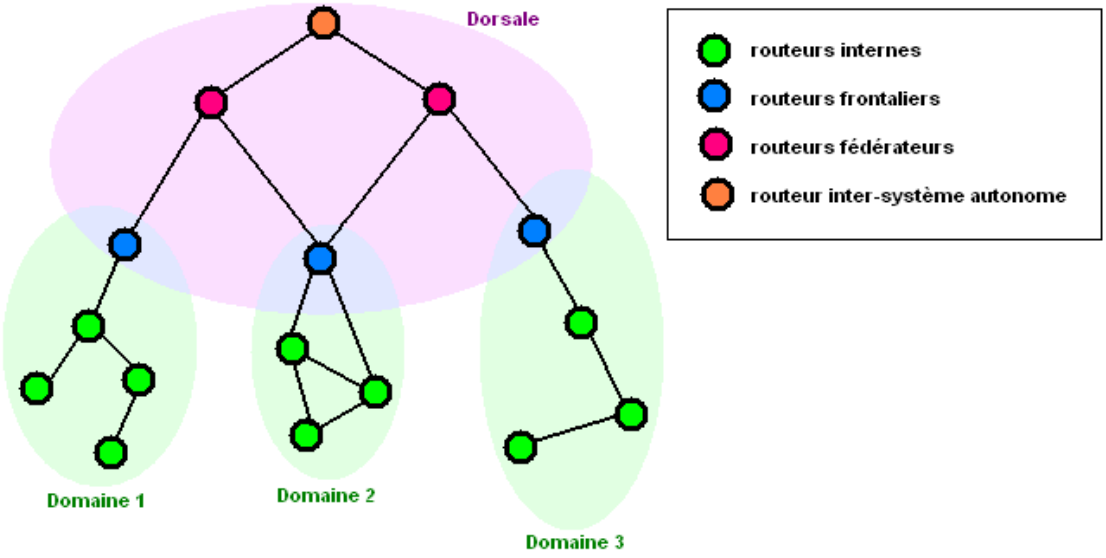
Tout comme RIP, OSPF (conçu pour succéder à RIP) sert au routage interne, et possède certaines propriétés avancées. A la base il s'agit d'un protocole à état de lien ayant recours à la technique d'inondations d'informations et à l'algorithme de Dijkstra. Les routeurs élaborent une carte topologique complète du système autonome puis déterminent un arbre de plus court chemin vers tous les réseaux du système, se considérant comme noeud racine. C'est à partir de cet arbre que peuvent être formées les tables de routage. Les valeurs de liaisons individuelles sont laissées à l'administrateur.

Les routeurs communiquent leurs informations de routage à tous les routeurs de leur système autonome. Ces informations sont diffusées à chaque changement d'état d'une liaison et de manière périodique.

OSPF apporte les innovations suivantes :

- *Sécurité* : Tous les échanges entre routeurs OSPF font l'objet d'une authentification, ce qui signifie que seuls les routeurs dignes de confiance peuvent participer au protocole.
- *Chemins de coût égal* : OSPF ne s'oppose pas à ce que différents chemins de même coût soient empruntés pour rejoindre une même destination.
- *Multidiffusion.*
- *Support d'une hiérarchie* : Capacité de ce protocole à structurer un système autonome de manière hiérarchique (voir ci-dessous).

Un système autonome OSPF peut être divisé en différents *domaines*, utilisant chacun son propre algorithme de routage d'information d'état de lien et communiquant ses informations à tous les routeurs du même domaine. Les détails internes à un domaine spécifique restent ainsi invisibles aux autres routeurs. Au sein de chaque domaine OSPF, un ou plusieurs *routeurs frontaliers* sont responsables de l'acheminement des paquets vers l'extérieur. Par ailleurs, chaque système autonome dispose d'un *domaine fédérateur* ayant pour fonction l'acheminement du trafic d'un domaine à un autre et rassemblant donc tous les routeurs frontaliers, ainsi qu'éventuellement d'autres routeurs. Le dessin ci-dessous illustre parfaitement cette situation.



4.5.3 Protocole BGP : routage inter-systèmes

BGP se définit comme un protocole à vecteur de chemin. Les routeurs BGP voisins, connus sous le nom de *partenaires BGP*, s'échangent des informations détaillées sur les chemins à emprunter (plutôt que des indications sur la distance comme dans les protocoles à vecteur de distance). BGP est un protocole distribué, dans le sens où les routeurs BGP ne communiquent qu'avec leurs voisins directs.

Le routage BGP se fait en direction de réseaux de destination plutôt qu'en direction de routeurs ou de serveurs ; une fois qu'un datagramme a atteint son réseau de destination, il est pris en charge par le routage interne de celui-ci jusqu'à son destinataire final. Dans le cadre de BGP, les systèmes autonomes s'identifient à l'aide d'un *numéro de système autonome unique* (ASN, *Autonomous System Number*).

BGP assure trois grandes fonctions :

- Réception et filtrage d'annonces de parcours en provenance de routeurs voisins. Les annonces de parcours sont des *promesses* que tout paquet reçu sera transmis au prochain routeur placé sur le chemin vers la destination. Il en profite pour supprimer les chemins le faisant passer par lui-même (ce qui provoquerait des boucles de routage).
- Sélection de chemin. Un routeur BGP peut recevoir plusieurs annonces de parcours différentes pour un même système autonome, parmi lesquelles il doit faire un choix.
- Envoi d'annonces de parcours aux routeurs voisins.

4.6 Au coeur d'un routeur

Quatre éléments principaux peuvent être identifiés dans l'architecture d'un routeur :

- *Port d'entrée.* Prend en charge les fonctionnalités de la couche physique et de la couche liaison. Il assure une fonction de consultation et d'acheminement des paquets entrant dans la matrice de commutation vers le port de sortie approprié.
- *Matrice de commutation.* Relie les ports d'entrée et les ports de sortie.
- *Port de sortie.* Emmagasine les paquets qu'il reçoit de la part de la matrice de commutation. Assure les fonctionnalités inverses de la couche liaison et de la couche physique.
- *Processeur de routage.* Chargé de l'exécution des protocoles de routage, de la mise à jour des informations et des tables.

4.6.1 Les ports d'entrée

Les fonctions de terminaison de ligne et de traitement de liaison de données qu'il doit assurer mettent en oeuvre les couches physiques. Les opérations de consultation et de ré-acheminement effectuées par le port d'entrée à l'arrivée des paquets jouent un rôle essentiel dans ses fonctions de commutation. Souvent c'est durant cette étape que le routeur détermine les ports de sortie vers lesquels rediriger les paquets arrivant. Les ports d'entrée gardent une copie de la table de ré-acheminement, qu'il actualise régulièrement. Cela permet de prendre des décisions directement au niveau des ports, sans solliciter le processeur de routage central (ce qui pourrait provoquer des phénomènes de goulet d'étranglement).

La consultation d'une table de ré-acheminement est simple conceptuellement : il suffit d'y chercher l'entrée de destination la plus apparentée à l'adresse. Dans la pratique ce n'est néanmoins pas si simple. Le point le plus problématique est le fait que les routeurs fédérateurs doivent opérer à des vitesses élevées, impliquant des millions de consultations par seconde ; on souhaite en effet que l'opération de consultation soit plus rapide que le temps qu'il faut au routeur pour recevoir un nouveau paquet. Dans de telles conditions, il est évident qu'un simple balayage linéaire du contenu de la table de réacheminement est parfaitement impossible. Une structure d'arbre serait déjà plus appropriée. Mais cela n'est pas encore suffisant. Une solution réside dans des mémoires à contenu adressable (temps de recherche constant).

Dès que le port de sortie a été identifié, le paquet peut entrer dans la matrice de commutation. Cependant un paquet peut se voir refuser temporairement l'accès si elle est occupée à traiter des paquets qui ont été reçus sur d'autres liaisons d'entrée. Il est alors placé en file d'attente sur son port.

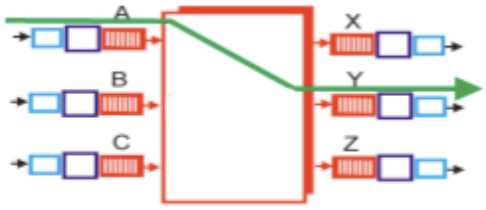
4.6.2 La matrice de commutation

La matrice de commutation forme le coeur du routeur, elle assure le transfert des paquets entre le port d'entrée et le port de sortie approprié. La commutation peut se faire de plusieurs manières :

Commutation par action sur les mémoires

A l'origine, les deux types de ports fonctionnaient comme des entrées/sorties d'un système d'exploitation conventionnel. Dans cette configuration, le port d'entrée signalait l'arrivée des paquets au processeur de routage par un signal d'interruption. Puis une copie du paquet était placée dans la mémoire du processeur qui effectuait les actions de routage pour connaître le port de sortie, puis transmettait une copie au port de sortie.

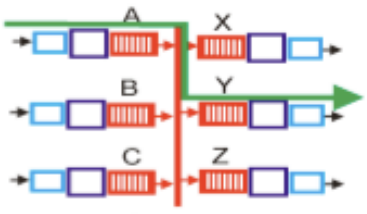
La plupart des routeurs actuels continuent d'avoir recours à une commutation par action sur les mémoires, mais la grande différence par rapport aux prédécesseurs porte sur le fait que la consultation de l'adresse de destination et la commutation du paquet sont sous contrôle de processeurs placés dans les cartes de communication d'entrée.



Commutation par action sur les mémoires.

Commutation par bus

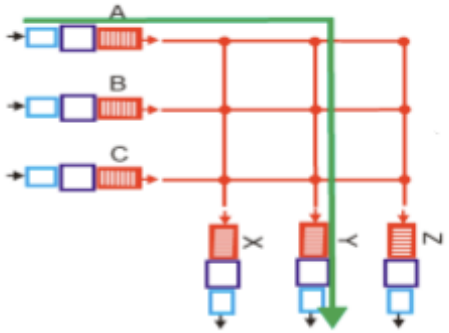
Avec cette méthode, le port d'entrée transfert les paquets directement au port de sortie grâce à un bus partagé, sans passer par le processeur de routage. L'inconvénient est qu'un seul paquet peut être transféré à la fois. Il y aura donc aussi un phénomène de file d'attente. C'est le débit de ce bus qui détermine la vitesse de commutation du routeur. Etant donné qu'on trouve des bus à plusieurs Gbit/s, cette méthode de commutation s'adapte à la plupart des réseaux d'accès et aux réseaux locaux d'entreprise.



Commutation par bus.

Commutation par réseau d'interconnexions

Pour réduire les limitations de débits associées au partage d'un bus commun, on peut utiliser un réseau d'interconnexions de $2n$ bus reliant n ports d'entrée à n ports de sortie. Un paquet qui arrive sur le port d'entrée du routeur est acheminé sur le bus horizontal jusqu'à l'interconnexion avec le bus vertical correspondant au port de sortie. Si ce bus vertical est libre, le paquet est transféré jusqu'au port de sortie, sinon il retourne au port d'entrée et est mis en attente.



Commutation par réseau d'interconnexions.

4.6.3 Les ports de sortie

Le port de sortie prend les datagrammes stockés dans sa mémoire et les transmet à la liaison de sortie. Il fait alors appel à la couche liaison et la couche physique.

4.6.4 Mise en attente

On peut constater que des files peuvent se former aux ports d'entrée et de sortie (matrice de commutation trop et/ou pas assez rapide selon les ports). A mesure de l'augmentation de ces files, le risque de perte de paquets augmente. Ce phénomène de perte se situe exactement au niveau des files d'attente, mais l'endroit exact dépend de la densité du trafic, de la vitesse de la matrice de commutation et du débit de la liaison.

Supposons que les liaisons d'entrée et de sortie disposent d'un débit identique, et que le routeur compte n ports d'entrée, et n ports de sortie. Si la vitesse de la matrice de commutation est au moins n fois supérieure à la vitesse de la liaison, aucun phénomène de mise en attente ne se produit à l'entrée, mais si tous les paquets venaient à être dirigés vers le même port de sortie il serait très rapidement saturé. Un gestionnaire de paquets doit déterminer quel paquet de la file doit être transmis. Cette règle est généralement la règle FIFO (*First In First Out*).

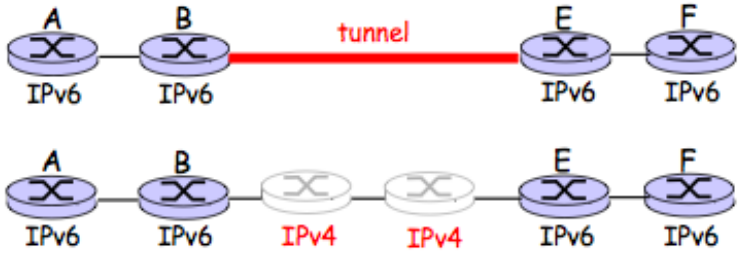
Si la matrice de commutation n'est pas assez rapide par rapport aux débits des liaisons d'entrée, le phénomène de mise en attente se produit alors au niveau des ports d'entrée.

4.7 IPv6

Au début des années 1990, l'épuisement progressif des adresses IPv4 à 32 bits menace. Une nouvelle version (IPv6) remédie à ce problème, et propose également de nombreuses innovations.

Avec IPv6, la taille des adresses passe de 32 bits à 128 bits, ce qui éloigne presque définitivement la menace de leur épuisement. L'en-tête d'un datagramme IPv6 devient fixée à 40 octets, ce qui facilite son traitement. La possibilité de fragmenter un datagamme disparaît, cette option devient du ressort de l'expéditeur et du destinataire exclusivement. Si un routeur reçoit un datagramme IPv6 trop volumineux il le supprime tout simplement et renvoie une erreur ICMP. La somme de contrôle disparait également, la couche transport et la couche liaison assurent déjà ce genre de service qui devient redondant.

Il est évident que tous les routeurs ne peuvent pas tous se déconnecter simultanément pour s'upgrader. Il faut donc une méthode transitionnelle pour passer de IPv4 à IPv6, capable de faire passer des datagrammes IPv6 sur des routeurs ne supportant pas encore ce format. La solution est une *mise en tunnel* : si deux noeuds IPv6 séparés par des routeurs IPv4 souhaitent s'échanger des datagrammes IPv6, ils considéreront la suite de routeurs IPv4 comme un tunnel. Le noeud IPv6 du côté du pôle expéditeur se munit du datagramme IPv6 tout entier et l'insère dans le champ de données d'un datagramme IPv4, et ce dernier est envoyé aux routeurs IPv4. Lorsque le datagramme parvient au routeur IPv6, il extrait les données pour récupérer un datagramme IPv6, comme illustré ci-dessous.



Cohabitation IPv4 et IPv6 grâce aux *tunnels*.

Chapitre 5

La couche Liaison et les réseaux locaux

5.1 Services de la couche liaison

Un protocole de la couche liaison est chargé du transfert des datagrammes de noeud en noeud, c'est-à-dire sur des liaisons individuelles dont l'ensemble constitue la voie de communication. Dans ce chapitre, nous appellerons *noeuds* aussi bien les hôtes que les routeurs, sans se préoccuper de savoir de quel type de noeud il s'agit. On appelle *liaison* les canaux de communication reliant deux noeuds adjacents du parcours.

Les services qui peuvent être fournis par un protocole de couche liaison sont les suivants :

- *Tramage.* Avant la transmission, le protocole procède à une *encapsulation* des datagrammes à l'intérieur d'une *trame*, qui comporte par ailleurs un certain nombre de champs d'en-têtes (tels que l'adresse physique par exemple).
- *Accès à la liaison.* Un protocole de contrôle d'accès au médium décrit les règles selon lesquelles la trame doit être transmise sur la liaison.
- *Transfert fiable.* Un protocole de la couche liaison proposant un transfert fiable garantit la bonne livraison sans erreur de tous les datagrammes qui lui sont soumis.
- *Contrôle de flux.* Les noeuds disposent généralement d'une mémoire tampon limitée pour mémoriser temporairement les trames qu'ils reçoivent. Ceci peut poser des problèmes lorsque les trames arrivent à un débit supérieur à celui qu'ils sont en mesure d'absorber.
- *Détection d'erreurs.* Des erreurs binaires peuvent se produire, la plupart des protocoles de couche liaison sont en mesure de les détecter.
- *Correction d'erreurs.* Procédure similaire, à la différence qu'elle peut déterminer l'emplacement précis de l'erreur et la corriger.

5.2 Techniques de détection et correction d'erreurs

La méthode de détection d'erreur la plus simple est celle qui repose sur l'emploi d'un bit de parité. Soit D des informations à transmettre de d bits. L'expéditeur ajoute aux données un unique bit tel que la somme des $d+1$ bits soit paire (ou impaire selon ce qui est convenu par le modèle). Cette méthode est un peu faible, car si un nombre pair d'erreurs s'est glissé dans les données, elles ne seront pas détectées ; elle ne peut donc être appliquée que sur des systèmes pour lesquels les erreurs n'arrivent pas en rafale et dont la probabilité est extrêmement faible.

On peut généraliser cette méthode en répartissant les d bits sur une matrice, et en calculant un bit de parité pour chaque ligne et chaque colonne. Cela permet d'une part de déterminer qu'il y a plusieurs erreurs, mais aussi de les localiser (dans une certaine mesure).

Avec les techniques de somme de contrôle, les d bits sont regroupés en octets. Chaque octet est considéré comme un nombre de 16 bits et tous les octets sont additionnés, puis le complément à 1 de cette somme est pris comme somme de contrôle. La somme des octets de la trame doit donc avoir tous ses bits à 1.

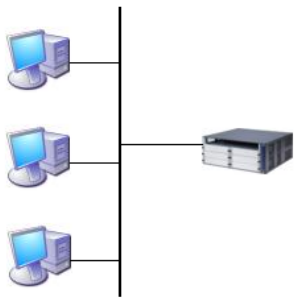
Une technique de détection d'erreur très fréquente repose sur les *codes de redondance cyclique* (CRC). Les codes CRC opèrent de la manière suivante : Soit un groupe de données de d bits qu'un noeud souhaite envoyer à un autre, l'expéditeur et le destinataire devant d'abord s'accorder sur une configuration binaire de $r+1$ bit, le *polynôme générateur* G. Le bit le plus significatif (à gauche) de G doit être réglé à 1. Pour un groupe de données D, l'expéditeur choisit r bits additionnels, R, et les ajoute à D de sorte que la configuration binaire de $r+1$ bits résultante soit exactement divisible par G en utilisant l'arithmétique modulo-2. La procédure de détection d'erreur est alors très simple : le destinataire divise les $d+r$ bits reçus par G, si le reste n'est pas nul il sait qu'une erreur s'est produite. Le calcul de R effectué par l'expéditeur se fait de façon à ce que $D \cdot 2^r \oplus R = nG$ (rappel : $\oplus$ = XOR), c'est-à-dire calculer

$$R = \text{reste}\left(\frac{D \cdot 2^r}{G}\right)$$

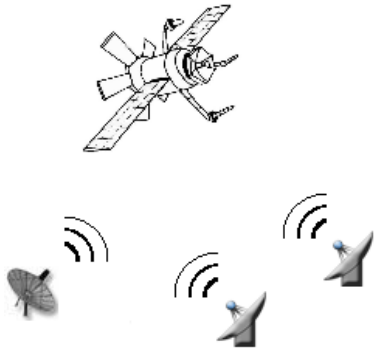
Cette technique est en mesure de détecter les erreurs en rafales inférieures à $r+1$ bits. De plus, une rafale d'erreurs affectant plus de $r+1$ bits sera détectée avec une probabilité $1 - \frac{1}{2^r}$.

5.3 Protocoles d'accès multiple

Il existe deux catégories de liaisons de données : les *liaisons point-à-point* (échanges entre un seul expéditeur et un seul destinataire) et les *liaisons à diffusion* (échanges entre plusieurs expéditeurs et/ou plusieurs destinataires). Le terme *diffusion* fait référence au fait que les trames transmises par un expéditeur sont diffusées sur un même canal partagé entre plusieurs destinataires.



Câble partagé (par exemple Ethernet).



Satellite.

Espace radioélectrique partagé (par exemple Wi-Fi).

Les réseaux d'ordinateurs ont recours à des *protocoles d'accès multiple* pour gérer les transmissions des noeuds reliés au canal de diffusion partagé. On en distingue trois types : les *protocoles à partage de canal*, les *protocoles d'accès aléatoire* et les *protocoles à partage de ressources*. Un protocole d'accès multiple opérant sur un canal à diffusion d'un débit R doit présenter les propriétés suivantes :

- Lorsqu'un seul noeud a des données à envoyer, il dispose d'un taux d'envoi de R bit/s.
- Si M noeuds ont des données à envoyer, chacun d'entre eux doit disposer d'un taux d'envoi instantané de R/M bit/s.
- Le protocole doit disposer d'une structure décentralisée, sans station maîtresse susceptible de bloquer tout le système en cas de panne.
- Le protocole doit être simple, de sorte que sa mise en place ne soit pas coûteuse.

Protocoles de partage de canal

Le multiplexage par répartition dans le temps (TDM) et en fréquence (FDM) sont deux techniques permettant de partager la bande passante d'un canal de diffusion entre toutes les stations qui y sont reliées. En TDM, chaque intervalle de temps est assigné à l'une des stations. Lorsque l'une d'entre elles doit envoyer un paquet, elle procède à cet envoi durant l'intervalle de temps qui lui est alloué (en général ce temps est suffisant pour transmettre un paquet entier). FDM procède à un partitionnement de la bande des fréquences, et alloue une partie du canal à chaque station. FDM et TDM présentent les mêmes avantages et inconvénients : ils évitent les collisions et assurent une répartition équitable de la bande passante, mais limitent fortement le débit maximal des stations les plus actives.

Le protocole d'accès multiple par répartition de code (CDMA, *Code Division Multiple Access*) alloue aux différentes stations un code particulier leur permettant de coder les bits d'information qu'ils envoient. Le CDMA permet alors à des paquets envoyés simultanément d'être compris par les différents destinataires, en dépit des interférences (permet également un anti-brouillage).

Protocoles d'accès aléatoire

Avec ce type de protocoles d'accès multiple, les stations disposent en permanence du débit intégral du canal. Lors d'une collision entre deux trames, chaque station impliquée procède à la retransmission de sa trame, jusqu'à ce que celle-ci parvienne à son destinataire sans aucune collision. Pour éviter une nouvelle collision immédiate, le protocole laisse s'écouler un laps de temps aléatoire avant de procéder à la retransmission.

Le protocole *ALOHA à allocation temporelle* en fait partie. Pour des raisons de simplicité, nous supposons que toutes les trames ont une taille de L bits, que le temps est divisé en intervalles de L/R secondes, que les expéditeurs commencent à transmettre leur trame au début d'un intervalle de temps, qu'ils fonctionnent de façon synchrone et que tout expéditeur est informé d'une collision avant la fin de l'intervalle de temps au cours duquel elle se produit. Lorsqu'un expéditeur a une nouvelle trame à envoyer, il attend le début d'un nouvel intervalle de temps. Dans le cas d'une collision, l'expéditeur retransmet sa trame au cours de l'intervalle suivant avec une probabilité p convenue, et ce jusqu'à ce qu'elle finisse par passer sans collision. On peut constater que les seuls intervalles de

temps qui ne sont pas gâchés sont ceux au cours desquels une seule station est active. L'*efficacité* dépend donc de la proportion d'intervalles réussis dans le cas où un grand nombre de stations sont actives en même temps. Pour simplifier le calcul, supposons que, pour tout intervalle, chaque station tente d'émettre une trame avec une probabilité p. Dans un environnement avec N stations, la probabilité qu'une station quelconque réussisse sa transmission est de $Np(1-p)^{N-1}$. Des paramètres adéquats ($p = 1/N$ et $N \to \infty$) amènent à une efficacité maximale de $1/e = 0.37$, soit une efficacité maximale de 37%. ALOHA offre un débit maximal à une station seule à transmettre. Etant donné que les stations sont à même de détecter une collision, le procédé est fortement décentralisé ; il requiert cependant une synchronisation (ce qui n'est pas le cas pour ALOHA pur qui est totalement décentralisé).

Protocole d'accès multiples avec détection de porteuse

Dans le monde des réseaux, *écouter avant de prendre la parole* est ce qu'on appelle procéder à une *détection de porteuse*. Cela impose à une station de consulter l'activité en cours sur le canal avant d'engager la transmission. Si une autre station est occupée, elle patiente un temps aléatoire avant de reconsulter le canal. Si deux stations commencent à émettre en même temps, elles interrompent leur transmission et se remettent en attente ; cela n'est possible que si une station reste à l'écoute du canal pendant sa transmission. Ces deux règles sont mises en application dans le protocole CSMA et CSMA/CD (CSMA avec détection de collisions).

Protocoles à partage de ressource

Le *protocole de sondage* implique la désignation d'une station maîtresse qui invite à transmettre toutes les stations de façon cyclique. Elle procède en informant la première station sur la liste du nombre maximal de trames qu'elle peut transmettre, puis se tourne vers la deuxième, et ainsi de suite. Cette procédure se répète jusqu'à ce que toute les stations aient été contactées. Cependant ce processus implique un temps d'invitation à transmettre, et, plus grave, si la station maîtresse est défaillante, tout le réseau est par terre.

Le *protocole de passage de jeton* n'a pas le problème de désignation d'une station maîtresse, il utilise une trame particulière, appelée *jeton*, qui circule d'une station à l'autre selon un ordre défini. La station qui possède le jeton a le droit d'envoyer des trames, et elle ne garde le jeton que si elle en a à envoyer. Puis le jeton passe à la station voisine. La panne de la machine ayant le jeton est encore un problème.

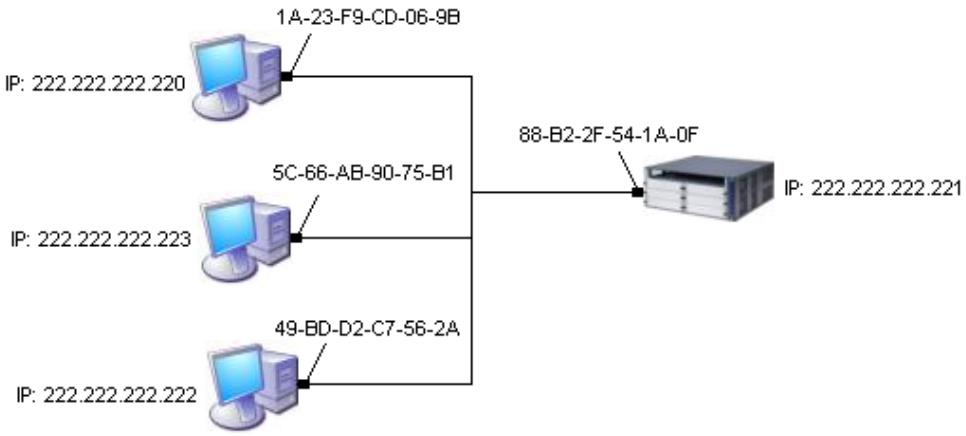
5.4 Adresses LAN et ARP

Un *réseau LAN* (*Local Area Network*) est un réseau d'ordinateurs concentré dans un espace privé. Au sein de ce type de réseau, les stations s'échangent des trames sur le canal de diffusion, ce qui signifie que lorsqu'une station transmet une trame, toutes les autres la

reçoivent. Cela n'est généralement pas nécessaire, il serait préférable que les différentes stations du réseau puissent s'adresser les unes aux autres au moment d'envoyer leurs trames. Une station qui recevra une trame qui n'est pas attribuée à son adresse LAN la rejettera tout simplement. Il peut néanmoins arriver qu'un expéditeur souhaite envoyer une trame à toutes les stations du réseau. Il aura alors recours à une *adresse de diffusion LAN* (qui n'est autre que FF-FF-FF-FF-FF-FF).

Chaque station est donc munie d'un *adaptateur* possédant une adresse LAN (on parle aussi d'*adresse physique*, d'*adresse Ethernet* ou encore d'*adresse MAC*). Il s'agit généralement d'une adresse de 6 octets, présentés sous forme hexadécimale. Lorsque l'adaptateur est réalisé, cette adresse est gravée dans une mémoire ROM et reste immuable. Ces adresses sont parfaitement uniques.

L'utilisation de deux systèmes d'identification d'une station (IP et LAN) implique un système de conversion : le *protocole de résolution d'adresses* ARP (*Address Resolution Protocol*). Chaque hôte internet et chaque routeur d'un réseau local dispose d'un *module ARP*.



Dans l'exemple ci-dessus, chaque station dispose d'une adresse IP et d'une adresse LAN. Si la station 222.222.222.220 cherche à envoyer un datagramme IP à la station 222.222.222.222, elle doit fournir à son adaptateur le datagramme muni de l'adresse LAN du destinataire. Une fois ces deux informations obtenues, l'adaptateur crée une trame de la couche liaison libellée à cette adresse et l'envoie à travers le réseau. Pour connaître l'adresse LAN du destinataire, elle doit soumettre l'adresse IP à son module ARP qui en assure la conversion et lui envoie l'adresse LAN 49-BD-D2-C7-56-2A.

Le module ARP de chaque station contient dans sa mémoire RAM une table appelée *table ARP*, qui contient les correspondances entre adresses IP et adresses LAN des différentes stations du réseau. Pour chaque entrée, la table contient également un temps d'expiration. Il faut noter que la table ne contient pas nécessairement l'adresse de toutes les stations du LAN. Dans le cas où la station expéditrice ne possède pas la conversion,

elle élabore un *paquet ARP*, contenant entre autres des champs d'adresses LAN et IP, qu'elle envoie sur le réseau sur l'adresse de diffusion ; sa fonction est d'interroger toutes les stations du réseau pour connaître l'adresse LAN correspondant à l'adresse IP en cours de conversion. La station concernée répond avec un message du même format à l'expéditeur, qui peut alors mettre sa table à jour.

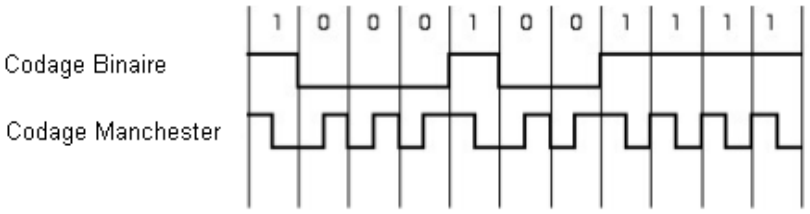
5.5 Ethernet

Ethernet reprend plusieurs systèmes. L'élément unificateur de ces divers systèmes est la structure des trames, qui reste la même dans tous les systèmes Ethernet. Ces trames possèdent six champs :

- *Champ de données.* 46 à 1500 octets. C'est là que vient se loger le datagramme IP.
- *Adresse de destination.* 6 octets. Ce champ contient l'adresse LAN de l'adaptateur du destinataire.
- *Adresse d'origine.* 6 octets.
- *Champ de type.* 2 octets. Ce champ permet à Ethernet de multiplexer les protocoles de la couche réseau, afin que le datagramme soit remis au bon protocole de la couche réseau.
- *Cyclic Redundancy Check (CRC).* Ce champ permet de détecter les erreurs qui se sont glissées dans le datagramme.
- *Préambule.* 8 octets. Toute trame Ethernet commence par un préambule dont les sept premiers octets ont une valeur 10101010 et le dernier une valeur 10101011, qui servent à réveiller l'adaptateur du destinataire et à synchroniser son horloge.

Le service réseau procuré par les différentes versions d'Ethernet est un service *sans connexion* et *non fiable*. En effet, aucune vérification n'est effectuée, et bien que les liaisons soient parfois fortement isolées, elles ne sont jamais à l'abri d'une erreur binaire due à un champ magnétique ou des interférences par exemple. Une trame erronée sera malgré cela remise à la couche réseau qui la rejettera.

Ethernet repose sur une transmission en bande de base, dans laquelle les adaptateurs envoient des signaux numériques directement sur le canal de diffusion. La carte d'interface ne transpose pas le signal sur une autre bande de fréquences, comme c'est le cas en ADSL ou avec les systèmes à modems câble. Ethernet utilise un *codage Manchester*. Dans ce codage, chaque bit contient une transition : vers le bas pour 1 et vers le haut pour 0. L'utilisation de ce codage s'explique par le fait que les horloges de l'expéditeur et du destinataire ne sont par parfaitement en phase.



Ethernet repose sur un algorithme d'accès multiple CSMA/CD :
- Les noeuds peuvent commencer à transmettre n'importe quand.
- Les adaptateurs ne transmettent jamais lorsqu'ils détectent une activité sur le canal (propriétés de détection de porteuse).
- Les adaptateurs interrompent leur transmission dès qu'ils détectent de l'activité d'un autre adaptateur au sein du canal (propriété de détection des collisions).
- Avant de procéder à la retransmission d'une trame, les adaptateurs patientent pendant une durée aléatoire (relativement courte par rapport à la durée de transmission d'une trame).

5.6 Hubs, ponts et commutateurs

La façon la plus simple d'interconnecter des réseaux LAN est l'utilisation d'un *hub*. Un bit rejoignant l'interface d'un hub est automatiquement diffusé sur toutes les interfaces du hub. Ce n'est rien d'autre qu'un répéteur de signal. Il est possible de concevoir des systèmes avec plusieurs niveaux de hubs. Ceux qui ne sont pas directement reliés à des stations sont appelés *hubs fédérateurs*.

L'utilisation abondante de hubs pose néanmoins des problèmes :
- Les différents domaines de collision sont regroupés en un seul même domaine.
- Les différentes technologies Ethernet ne seront pas toujours compatibles.

A la différence des hubs qui sont des équipements de la couche physique, les *ponts* sont des éléments de la couche liaison. En effet, les ponts sont des sortes de commutateurs de paquets, capables de transmettre et filtrer les trames au moyen de leur adresse de destination. Lorsqu'un pont reçoit une trame sur son interface d'entrée, il examine son adresse de destination de couche 2 et tente de la transmettre à l'interface appropriée ; si cette interface est celle d'où vient la trame, elle est tout simplement absorbée. Ces fonctions sont assurées via l'utilisation d'une table contenant des informations sur certaines stations du réseau. De plus, les ponts mettent CSMA/CD en oeuvre pour limiter les collisions sur le LAN.

L'un des grands problèmes liés à une interconnexion purement hiérarchique des différents segments de LAN a trait à la fiabilité de l'ensemble. Si l'un des hubs tombe en panne, c'est toute une partie du réseau qui perd sa connexion. Il est donc souhaitable de mettre en place plusieurs chemins entre les différents segments d'un LAN. Cependant, en l'absence de contrôle, les trames peuvent former des boucles et se multiplier au sein du réseau. Pour éviter ce problème, il faut avoir recours au *protocole d'arbre recouvrant*. Grâce à ce protocole, les ponts des différents segments LAN s'échangent des informations sur les différents arbres de recouvrement du réseau, c'est-à-dire des sous-ensembles du réseau initial comprenant tous les ponts, mais ne présentant pas de boucle. Si à un moment donné, l'une des liaisons de l'arbre venait à être coupée, les ponts peuvent automatiquement reconnecter les interfaces et réappliquer l'algorithme.

5.7 Protocole PPP (point-à-point)

PPP (*Point-to-Point Protocol*) est le protocole le plus utilisé pour les liaisons point-à-point, autrement dit les liaisons directes entre deux stations. Quelques conditions lui sont imposées :

- *Mise en trame du paquet.*Le pôle expéditeur doit être en mesure de prendre en compte un paquet de la couche réseau et de l'encapsuler à l'intérieur d'une trame de couche liaison PPP, de telle sorte que le destinataire puisse identifier le début et la fin, aussi bien de la trame que du paquet de données associé.
- *Transparence.* PPP ne doit imposer aucune contrainte aux données globales associées au paquet réseau (il n'a par exemple pas le droit d'interdire certaines configurations binaires dans le paquet.
- *Compatibilité.* PPP doit être compatible avec différents protocoles de couche réseau. Il doit également être en mesure d'être exploité sur différents types de liaisons.
- *Détection d'erreur.* Un destinataire PPP doit être en mesure de détecter des erreurs binaires dans la trame reçue.
- *Simplicité.* PPP doit être de moindre coût pour des raisons évidentes.

Une trame PPP est constituée de :

- *Fanion.* Toutes les trames PPP débutent et se terminent par un fanion de 1 octet qui prend pour valeur 01111110.
- *Champ d'adresse.* L'unique valeur que peut prendre ce champ est 11111111.
- *Champ de contrôle.* L'unique valeur que peut prendre ce champ est 00000011.
- *Protocole.* Ce champ indique au destinataire PPP le protocole de couche supérieure auquel les données encapsulées doivent être transmises.
- *Informations.* Ce champ contient le paquet venant de la couche réseau.
- *Somme de contrôle.* Ce champ sert à la détection d'erreur au sein des trames PPP.

Un problème persiste. Que se passe-t-il si un octet de fanion 01111110 apparaît en plein milieu d'une trame ? La solution consiste à avoir recours à des *octets de bourrage.* PPP définit un octet de bourrage de valeur 01111101, qu'il insère devant tout fanion 01111110 qui se trouve en dehors du champ prévu à cet effet. De la même manière, si cette configuration de cet octet apparaît au sein de la trame, celui-ci devra également être précédé d'un octet de bourrage, et ainsi de suite tant que nécessaire.

Chapitre 6

Liste des abréviations

AIMD : *Additive Increase, Multiplicative Decrease*. Algorithme de contrôle de congestion de TCP.
API : *Application Programmers' Interface*. Interface entre l'application et le réseau.
ARP : *Address Resolution Protocol*. Protocole permettant la conversion des adresses LAN et IP.
AS : *Autonomous Systems*. Régions de routeurs s'autogérant.
ASN : *Autonomous System Number*. Numéro d'identification unique des systèmes autonomes BGP.
BIND : *Berkeley Internet Name Domain*. Logiciel standard pour les serveurs de nom Unix.
CARP : *Cache Array Routing Protocol*.
CDMA : *Code Division Multiple Access*. Protocole d'accès multiple par répartition de code.
CDN : *Content Distribution Networks*. Réseaux de distribution de contenu.
CIDR : *Classless InterDomain Routing*.
DHCP : *Dynamic Host Configuration Protocol*. Protocole d'obtention d'IP automatique.
DNS : *Domain Name System*. Annuaire de correspondance entre les noms de domaines et les adresses IP.
CRC : *Cyclic Redundancy Check*. Champ de détection des erreurs au sein d'une trame Ethernet.
DSL : *Digital Subscriber Line*. Ligne d'abonné numérisée.
FTP : *File Transfert Protocol*. Protocole de transfert de fichiers.
GBN : *Go-Back-N*. Protocole de transport à fenêtre glissante.
GPRS : *General Packet Radio Service*. Technologie d'accès mobile par paquets.
HFC : *Hybrid Fiber Coaxial Cable*. Liaison hybride fibre et coaxial (câble).
HTTP : *Hypertext Transfert Protocol*.
ICMP : *Internet Control Message Protocol*. Protocole de rapport d'erreur de couche réseau.
IP : *Internet Protocol*.
IETF : *Internet Engineering Task Force*. Groupe d'étude de l'ingénierie de l'Internet.

ISP : *Internet Service Provider*. Fournisseur d'accès à l'Internet.
FDM : *Frequency Division Multiplexing*. Multiplexage fréquentiel.
LAN : *Local Area Network*. Réseau local à grande vitesse.
MTU : *Maximum Transfert Unit*. Volume maximum de données qui peut être porté par un paquet de la couche liaison.
NAP : *Network Access Points*. Points d'accès au réseau.
NAT : *Network Address Traslation*. Traduction d'adresses réseau.
OSPF : *Open Shortest Path First*. Protocole de routage avec priorité au plus court chemin.
POP : *Point Of Presence*. Point de présence d'un ISP.
PPP : *Point-to-Point Protocol*. Protocole de la couche liaison pour les liaisons point à point.
P2P : *Peer-to-Peer*.
PDA : *Personal Digital Assistant*.
RFC : *Request for Comments*. Format de documents contenant les normes IETF.
RIP : *Routing Information Protocol*. Protocole d'information de routage.
RTT : *Round-trip Time*. Temps nécessaire à un petit paquet pour faire un aller-retour entre un client et un serveur.
SMTP : *Simple Mail Transfert Protocol*. Protocole de gestion des systèmes de messageries électroniques.
TCP : *Transmission Control Protocol*. Protocole de transfert de données fiable orienté connexion.
TDM : *Time Division Multiplexing*. Multiplexage temporel.
TTL : *Time To Live*. Durée de vie d'un datagramme.
UDP : *User Datagram Protocol*. Protocole de transfert de données non fiable sans connexion.
URL : *Uniform Resource Location*. Adresse propre d'un fichier sur le Web.
UTP : *Unshielded Twisted Pair*. Paire torsadée non blindée.
WAP : *Wireless Access Protocol*. Protocole d'accès sans fil.
WLAN : Réseau LAN sans fil.
WML : *WAP Markup Language*. Equivalent d'HTML pour le WAP.

Bibliographie

[1] James Kurose & Keith Ross. *Analyse structurée des réseaux. Des applications de l'internet aux infrastructures de télécommunication.* Pearson Education France 2003.

[2] Guy Leduc. *Réseaux, INFO-F-303.* Slides de cours. Université Libre de Bruxelles 2008.

Table des matières

1 Les réseaux d'ordinateurs et l'Internet **2**
- 1.1 Qu'est-ce que l'Internet ? . . . 2
- 1.2 Réseaux d'accès . . . 3
- 1.3 Coeur du réseau . . . 4
- 1.4 Réseaux d'accès et supports physiques . . . 6
- 1.5 Les ISP et les réseaux fédérateurs de l'Internet . . . 8
- 1.6 Retards et pertes au sein des réseaux . . . 8
- 1.7 Les couches de protocoles . . . 9

2 La couche application **10**
- 2.1 Principes des protocoles de la couche Application . . . 10
- 2.2 Le Web et HTTP . . . 11
- 2.3 DNS, le service d'annuaire de l'Internet . . . 15
- 2.4 Distribution de contenu . . . 18

3 La couche transport **20**
- 3.1 Les services de la couche transport . . . 20
- 3.2 Multiplexage et démultiplexage . . . 20
- 3.3 Transport sans connexion : UDP . . . 21
- 3.4 Le transfert de données fiable . . . 22
  - 3.4.1 Transfert de données fiable sur canal fiable : rdt1.0 . . . 22
  - 3.4.2 Transfert de données fiable sur canal avec erreurs : rdt2.0 . . . 23
  - 3.4.3 Transfert de données fiable sur canal avec pertes/erreurs : rdt3.0 . . . 25
  - 3.4.4 Transfert de données fiable avec pipelinage . . . 26
- 3.5 Transport orienté connexion : TCP . . . 27
- 3.6 Contrôle de congestion . . . 29
  - 3.6.1 Les causes et effets du phénomène de congestion . . . 29
  - 3.6.2 Contrôle de congestion de TCP . . . 30

4 La couche réseau **33**
- 4.1 Modèles de services . . . 33
- 4.2 Les principes du routage . . . 34
- 4.3 Routage hiérarchique . . . 36

4.4 Protocole Internet (IP) . . . 37
4.4.1 Adressage . . . 37
4.4.2 Transport d'un datagramme . . . 38
4.4.3 Structure d'un datagramme . . . 39
4.4.4 Fragmentation . . . 40
4.4.5 Protocole ICMP . . . 40
4.4.6 Protocole DHCP . . . 41
4.4.7 Traducteur d'adresses NAT . . . 41
4.5 Routage au sein de l'Internet . . . 42
4.5.1 Le protocole RIP : informations de routage . . . 42
4.5.2 Le protocole OSPF : priorité au plus court chemin . . . 43
4.5.3 Protocole BGP : routage inter-systèmes . . . 44
4.6 Au coeur d'un routeur . . . 45
4.6.1 Les ports d'entrée . . . 45
4.6.2 La matrice de commutation . . . 46
4.6.3 Les ports de sortie . . . 47
4.6.4 Mise en attente . . . 47
4.7 IPv6 . . . 48

5 La couche Liaison et les réseaux locaux 49
5.1 Services de la couche liaison . . . 49
5.2 Techniques de détection et correction d'erreurs . . . 50
5.3 Protocoles d'accès multiple . . . 50
5.4 Adresses LAN et ARP . . . 53
5.5 Ethernet . . . 55
5.6 Hubs, ponts et commutateurs . . . 56
5.7 Protocole PPP (point-à-point) . . . 57

6 Liste des abréviations 58

Bibliographie 60

Table des matières 61