Introduction aux réseaux et programmation en C des protocoles TCP et UDP

Version originale : V. Ribaud - février 2002
Adaptation : P. Le Parc
Département d'Informatique - Faculté des Sciences et Techniques
Université de Bretagne Occidentale

Janvier 2011

Note : Les chapitres 1 et 2 doivent être considérés comme obsolètes et le chapitre 3 comme partiellement obsolète. Ils sont laissés dans ce document à titre d'information. Les autres chapitres sont à jour et utilisables pour la programmation réseaux en C avec les protocoles TCP et UDP.

Table des matières

Table des figures

Chapitre 1

Travailler en réseau

Pour que le réseau fonctionne, une “syntaxe” commune est nécessaire afin que deux machines différentes échangent des données. Cette “syntaxe” est régie par les protocoles réseaux. Ils participent chacun à leur niveau : alors que IP sert principalement à envoyer des paquets sur un câble, TCP et UDP donnent des possibilités supérieures, basées sur IP, en rendant plus aisée l'écriture de programmes d'applications. TCP, par exemple, fournit un moyen de contrôler que toutes les données parviennent à leur destinataire, même si des paquets sont perdus en route. Pour utiliser des applications réseau, il n'est (heureusement) pas nécessaire de connaitre la façon dont travaillent ces protocoles. Parmi les applications réseau d'usage courant, on trouve `rlogin` qui permet de se connecter sur une machine distante (à condition d'y avoir un compte), `ftp` pour faire du transfert de fichiers, et `mail` pour envoyer des messages électroniques.

Il y a plusieurs sortes de réseaux. Les différences entre eux ne sont pas fondées sur les câbles qu'ils utilisent, mais surtout sur le protocole utilisé sur ce câble. A titre d'exemple, il existe SNA, DECNET, INTERNET avec les protocoles TCP/IP, et la famille des protocoles ISO, sensés devenir un jour des protocoles standards. A ce jour, TCP/IP est le plus largement répandu. Il existe sur la majeure partie des systèmes UNIX. Ce qui va suivre se réduit à étudier les programmes basés sur TCP/IP.

1.1 Domaines

Les applications qui utilisent le réseau ont besoin que les machines soient nommées. L'ensemble initial des machines constituait un espace de noms "à plat", sans structuration particulière.

On a été vite confronté au problème de gérer un ensemble de noms, sans avoir recours à un site central pour le gérer.
La réponse consiste à décentraliser le mécanisme de nommage en déléguant une partie de l'autorité pour des sous-ensembles de l'espace des noms. Le plan de nommage est donc hiérarchique. Le niveau le plus élevé de la hiérarchie divise l'espace des noms et délègue la responsabilité pour chaque division.
La responsabilité peut être subdivisée à chaque niveau, appelé sous-domaine.

Chaque machine a donc un nom unique appelé “domain name”. Un domaine ressemble à ça :

```
machine.sous-domaine1.....sous-domaine.domaine_general
```

Il se compose d'un nom de machine, et de plusieurs sous-domaines, séparés par un point. Le dernier domaine est dit domaine général (“top-level”).

Conceptuellement, les noms de domaine supérieurs permettent l'utilisation de deux systèmes hiérarchiques diffférents : géographique et organisationnel.

Chaque pays participant se voit assigné un domaine supérieur, habituellement le code à deux lettres de l'ISO.
Les machines françaises appartiennent au domaine *fr*, dont le responsable administratif est l'INRIA [1].
Les USA ont choisi de diviser le domaine *us* à raison d'un sous-domaine par état (*ca.us* correspond donc à l'état de Californie).

Une autre possibilité offerte est la décomposition par secteurs d'activité.
Il existe sept domaines généraux :

EDU : institutions à but éducatif

COM : organismes à vocation commerciale

ORG : organismes non commerciaux

GOV : institutions gouvernementales (laboratoires nationaux)

MIL : institutions militaires

NET : principaux nœuds du réseau

ARPA : obsolète, à l'origine, les institutions spécifiques d'Internet.

Parmi les codes des pays, on a :

AU	Australie	AT	Autriche
CH	Suisse	DE	Allemagne
DK	Danemark	FI	Finlande
FR	France	IL	Israël
IT	Italie	JP	Japon
KR	Corée du sud	MX	Mexique
NL	Pays-Bas	NO	Norvège
SE	Suède	UK	Grande Bretagne

Pour l'utilisation du réseau local de votre site, des alias sont définis pour vous éviter d'avoir à taper de longs noms de domaines. Par exemple, la machine :

```
machine.toto.ailleurs
```

peut être appelée seulement `machine` sur le réseau local.

1.2 Connexion à distance

Une fois que vous êtes connectés à une machine reliée au réseau, vous pouvez aussi vous connecter aux autres machines sur lesquelles vous avez un compte. Il faut cette fois utiliser le programme `rlogin` (`r` pour *remote*, à distance) ou `telnet` . Contrairement à la connexion initiale, il faut donner le nom de la machine sur laquelle on se connecte. Par exemple :

```
$ rlogin machine.toto.ailleurs
```

1. Institut National de Recherche en Informatique et en Automatique

Si, sur l'autre machine, votre nom d'utilisateur est différent, on utilisera :

```
$ rlogin -l mon_autre_nom machine.toto.ailleurs
```

Une session telnet ressemble à ça :

```
telnet machine.toto.ailleurs
Trying 7.7.7.7 ...
Connected to machine.
Escape character is '^]'.


BIDULE OS UNIX (machine)

login: nom_utilisateur
Password:
```

Pour se déconnecter, vous pouvez taper la commade `logout` ou taper `^d`, qui signifie "maintenir la touche Ctrl enfoncée, et, pendant ce temps, appuyer sur la touche `d`" ou taper `^]`.

Différences entre `telnet` et `rlogin` :

- `rlogin` ne marche qu'entre machines UNIX. Pour se connecter sur des machines non UNIX, il faut utiliser `telnet`.
- Pour se connecter sur des machines très lointaines, utilisez `telnet`. Si vous êtes connectés sur une machine distante, et que vous tapez une commande, chaque caractère est envoyé sur le réseau, enveloppé dans un paquet de données. Il est plus économique que toute la ligne de commande soit envoyée dans un paquet de ce type. Ceci devient essentiel si les paquets doivent aller très loin, sollicitant beaucoup les réseaux. Sous `telnet`, tapez le caractère d'échappement (en général `^]`) et la commande `mode line`. La ligne de commande sera alors traitée sur la machine locale, et sera envoyée sur le réseau après que vous aurez pressé la touche `RETURN`. Ceci, bien sûr, ne marche pas pour les éditeurs. Pour revenir au mode précédent, tapez `mode character`.
- `rlogin` permet de vous connecter à distance sans avoir à retaper votre mot de passe, grâce à l'utilisation du fichier `.rhosts` sur la machine de destination. Dans ce fichier, figure la liste des machines à partir desquelles un `rlogin` peut être autorisé sans demander le mot de passe. Pour des raisons de sécurité, il est fortement recommandé de ne PAS utiliser cette propriété : si quelqu'un arrive à s'introduire sur votre compte, il ou elle peut s'introduire sur tous les comptes qui vous appartiennent par un simple `rlogin`, sans mot de passe. Un fichier `.rhosts` peut se justifier s'il est limité à quelques machines d'un réseau local où vous avez le même mot de passe. Dans tous les cas, votre `.rhosts` DOIT avoir le niveau de sécurité `400`, c'est à dire seulement lisible par vous.
- `telnet` vous donne plus de possibilités que la seule connexion à distance. Référez-vous aux pages de manuel.

1.3 Transfert de fichiers

Il existe deux utilitaires pour le transfert de fichiers : `rcp` et `ftp`. `rcp` n'est pas recommandé car il nécessite l'existence d'un fichier `.rhosts`. `rcp` ne convient pas au transfert de gros fichiers.

Le début d'une session ftp (File Transfer Protocol) pour l'utilisateur `arthur` se connectant à la machine `machine.toto.ailleurs` ressemble à ça :

```
$ ftp machine.toto.ailleurs
Connected to machine.toto.ailleurs
220 machine FTP server (BIDULE OS 77.1) ready.
Name (machine.toto.ailleurs:arthur): arthur
331 Password required for arthur.
Password:
230 User arthur logged in.
ftp>
```

Vous donnez le nom de la machine comme argument de la commande `ftp`. Vous devez entrer votre nom d'utilisateur et votre mot de passe pour avoir accès à la machine. `ftp` attend ensuite les commandes. Un point d'interrogation (?) vous donnera la liste des commandes accessibles. Parmi les commandes importantes, on a :

`ls` ou `dir` liste le répertoire courant de la machine distante.

`cd répertoire` va dans le répertoire `répertoire` sur la machine distante.

`lcd répertoire` va dans le répertoire `répertoire` sur la machine locale.

`binary` transfère les fichiers en mode binaire. Ceci est utile pour le transfert de gros fichiers, car plus rapide. Pour revenir à la valeur par défaut, tapez `ascii`. Le mode binaire devrait être utilisé pour tous les transferts, sauf si :
- au moins une des machines n'est pas une machine UNIX.
- vous transférez des fichiers ASCII entre deux architectures différentes.

`hash` inprime des # pendant le transfert, pour que vous puissiez en évaluer la vitesse. Le nombre d'octets correspondant à un # dépend du serveur et est affiché lors de la commande.

`get nom_du_fichier` transfère le fichier `nom_du_fichier` de la machine distante vers la machine locale.

`put nom_du_fichier` transfère le fichier `nom_du_fichier` de la machine locale vers la machine distante.

`mget liste_de_fichiers` transfère les fichiers de la liste `liste_de_fichiers` de la machine distante vers la machine locale.

`mput liste_de_fichiers` transfère les fichiers de la liste `liste_de_fichiers` de la machine locale vers la machine distante.

`prompt` bascule du mode interactif vers le mode non-interactif, et *vice-versa*. Ceci est utile pour les commandes `mget` et `mput` qui vous demandent confirmation à chaque transfert de fichier.

`quit` termine la session.

Vous ne pouvez pas transférer des nombres en virgule flottante entre deux architectures de machines différentes en utilisant le mode binaire (Par exemple entre un DEC et un SUN SPARC) [2]

Certains sites font du ftp anonyme. Ces sites ont des archives où ils stockent des programmes du domaine public. Vous pouvez utiliser le ftp anonyme sans avoir de compte sur ces machines, pour récupérer les programmes qui vous intéressent. Il suffit de donner `anonymous` comme nom d'utilisateur et votre adresse e-mail (courrier électronique) comme mot de passe. Ne donnez JAMAIS votre mot de passe ! L'information sur les serveurs ftp anonyme peut être obtenue en lisant les forums réseau (voir plus loin).

2. Il faut faire la différence entre la représentation d'un nombre en virgule flottante dans la machine, et sa représentation sous forme de caractères qui peut être générée par la fonction C `fwrite` qui le transforme en chaine de caractères.

En tous cas, avant d'aller chercher des fichiers au loin, commencez par regarder les sites les plus proches, afin d'éviter de surcharger les réseaux.

1.4 Messagerie électronique

Le réseau vous permet d'envoyer et de recevoir des courriers électroniques. Il y a de nombreux programmes pour gérer votre messagerie. Les plus simples sont `mail` et `mailx` qui existent sur presque tous les systèmes UNIX. Au sein d'emacs, il existe `rmail`. Vous pouvez y entrer en tapant `ESC x rmail` sous emacs. En tapant `^h m` vous aurez une description du mode dans lequel vous êtes. Un autre gestionnaire pratique de messagerie est `mh`, qui est accessible à partir du shell, d'une interface X ou d'emacs. Il se peut qu'il y ait d'autres programmes de messagerie sur votre site. La façon dont on peut lire son courrier dépend completement du programme que vous utilisez, et il n'est pas possible de tous les détailler. Il y a quand même diverses choses à savoir pour envoyer des "mail".

Vous devez connaitre l'adresse de votre correspondant. Supposons pour le moment que cette personne est reliée au réseau Internet. Les adresses y ont la forme suivante :

```
nom@domaine
```

`nom` est le nom de la boite aux lettres. Sur les systèmes UNIX, il est courant d'utiliser le nom d'utilisateur comme nom de boite aux lettres. Certains systèmes acceptent le nom complet. Il peut y avoir aussi d'autres noms, par exemple une liste de distribution, où le courrier qui y est envoyé est redistribué à plusieurs personnes.

`domaine` est le nom du domaine où se trouve la machine qui doit recevoir le courrier. Ce n'est pas forcement la machine sur laquelle travaille habituellement votre correspondant. Vous devez connaitre la machine réceptrice précise. Le `domaine` peut aussi être un alias défini par le site où vous voulez envoyer vos messages, afin de vous faciliter la tâche (voyez le champ `cc:` dans l'exemple d'en-tête qui suit).

Un en-tête typique ressemble à cela – avec de petites variations selon votre gestionnaire de messagerie :

```
To: juser@foo.bar.edu
cc: arthur@machine.toto.ailleurs
Bcc: moi
Subject: comment vas-tu ?
-----------
Ici figure le corps du message....
```

`To:` contient l'adresse de la personne à laquelle on envoie la lettre. `cc:` est la liste des adresses des personnes qui recevront des "copies carbone" du message. Cette liste d'adresses est communiquée à tous les récepteurs du message. Le champ `Bcc:` s'utilise de même, mais la liste de ses adresses ne sera pas transmise aux destinataires. Il est souvent utilisé pour s'envoyer une copie du message. Puisque pour envoyer un message sur le réseau local, vous n'avez pas besoin de définir le domaine, vous pouvez ne mettre que votre nom d'utilisateur dans le champ `Bcc:`. Dans l'en-tête par défaut, vous trouverez probablement le champ `cc:` ou `Bcc:`, rarement les deux, bien que vous puissiez les utiliser. Tous ces champs peuvent recevoir une liste d'adresses, séparées par des virgules.

Lorsque vous envoyez un message électronique, vous ne savez pas s'il est bien arrivé ou non. Le fait que vous ayez reçu votre copie carbone ne signifie pas que les autres destinataires aient reçu leur copie. Par contre, vous serez informés si la distribution a échoué. Si vous n'êtes pas informé

d'un problème de distribution dans les jours qui suivent, vous pouvez être raisonnablement sûr que tout s'est bien passé.

Si votre message n'a pas pu être distribué, vous le recevez en retour, accompagné d'une note vous expliquant les causes de l'échec. Les raisons classiques sont :

`unknown user` Il n'existe pas de boite aux lettres à ce nom sur cette machine. Vérifiez le nom de votre correspondant.

`unknown host` Le domaine que vous avez spécifié n'est pas connu.

`host has been down for ...` Signifie que la machine de votre correspondant existe, mais ne répond pas aux demandes de transfert de messages.

Si vous voulez envoyer du courrier à une personne qui n'est pas sur Internet, mais sur Bitnet, utilisez l'adresse Bitnet, suivie d'un "`.bitnet`".

Certaines adresses de messagerie contiennent un signe %, comme dans :

```
utilisateur%machine1@machine2.domaine
```

Ceci indique que vous devez passer par une passerelle (`machine2`). Ceci n'a pas d'importance pour vous.

Vous pourrez trouver deux autres types d'adresses qui risquent de poser plus de problèmes :

- `machineN! ... !machine2!machine1!utilisateur`
  est une adresse UUCP. Essayez de la traduire en :

  ```
  utilisateur@machine1.uucp
  ```

  Si cela ne marche pas, vous devez spécifier au moins une passerelle dans l'adresse. Essayez quelque chose comme ça :
  `utilisateur%machine1% ... %machineN-1@machineN.uucp`
  ou
  `machineN-1! ... !machine1!utilisateur@machineN.uucp`
  ou alors, demandez à votre spécialiste local de la messagerie.
- `machine::utilisateur`
  est une adresse DECnet. Ces adresses sont très difficiles à traduire en domaine, à supposer que ce soit possible. Essayez d'obtenir de la part de votre correspondant une autre adresse !

1.5 Forums réseaux

Il existe un système international d'information appelé `netnews` ou `news`. Ce système comporte plus d'un millier de groupes, allant des plus pures discussions techniques au sujet des ordinateurs, jusqu'aux disputes politiques et aux informations récréatives. Le nombre de groupes auxquels vous avez accès dépend du site.

Pour lire ces forums, vous devez avoir un programme adapté. Il en existe des quantités, et chacun fonctionne à sa manière. Demandez lequel est disponible sur votre site.

Attention ! La première fois que vous lirez les "news", vous serez inscrit à tous les groupes de discussion, avec environ 120 000 articles qui vous attendent ! Bien sûr, vous pouvez vous désinscrire des groupes qui ne vous intéressent pas.

1.6 Le web

Ce paragraphe est inspiré de [Puj00].

WWW (World-Wide Web) est un système de documents hypermédias distribués, créé par le CERN (notamment Tim Berners-Lee) en 1989. Ce système travaille en mode client-serveur. Les clients sont des logiciels appelés *browsers*[3], comme Netscape, Internet Explorer, ...
Les clients et les serveurs du Web utilisent un protocole de communication appelé HTTP, *Hypertext Transfer Protocol*. Le langage HTML, *Hypertext Markup Language*, permet de définir une utilisation spécifique du document et notamment la navigation avec des liens. HTML est un langage à balise (*mark-up*), une balise encadre une partie du document qui doit être traitée selon les règles associées au type de balise (navigation pour les balises de type hypertexte, présentation et formatage de documents pour les balises de type commandes d'édition). Les liens hypertextes indiqués par des zones de texte reconnaissables contenant des URL, *Uniform Resource Locator*, permettent de relier les documents entre eux, quelle que soit la localisation géographique de ces documents. L'ensemble de tous ces liens entre documents forment la toile (d'araignée) ou Web, sur laquelle il est possible de naviguer.

Le groupe de travail SGML, *Standard Generalized Markup Language*, du consortium W3C travaille sur un standard en train de remplacer HTML ; il s'agit de XML, *eXtensible Markup Language*. A la différence de HTML qui est une DTD (*Document Type Definition*) particulière, XML permet de définir une DTD par classe de documents qui définit les éléments constituant le document. Tout document XML est constitué de sa DTD et du document physique composé d'éléments imbriqués à l'aide de balises. En simplifiant on peut dire que l'apparence et les caractérisitiques du document physique sont définis et contrôlés par sa DTD.

Audio et vidéo

L'audio et la vidéo constituent des applications potentielles très importantes pour les réseaux IP. De nombreuses évolutions des protocoles audio et vidéo voient le jour qui permettent un fonctionnement sur Internet comme dans un Intranet (un réseau local ou étendu privé qui utilise les protocoles de l'Internet).

1.7 Le réseau local

Le réseau n'est pas seulement important pour la connexion à distance ou la messagerie. S'il y a plusieurs machines sur votre site, il est probablement utilisé pour vous rendre l'utilisation de ces machines plus transparente. On réalise cela grâce à ce qui suit :

- Network Information System (NIS), couramment appelé Yellow Pages (YP)[4] conserve les informations sur les utilisateurs, comme dans le fichier `/etc/passwd` sur une machine qui est le serveur central YP. Ces fichiers sont disponibles pour les autres machines. Ceci entraine que votre mot de passe et votre shell sont les mêmes sur toutes les autres machines de ce domaine YP, c'est à dire qu'elles récupèrent leurs informations sur le même serveur YP. Si vous changez votre shell ou votre mot de passe sur une machine, peu importe laquelle, le changement sera stocké sur le serveur YP, et sera donc connu de toutes les autres machines de ce domaine YP.

3. brosseurs si on veut parler français
4. NIS/YP et NFS ont été développés par SUN Microsystems et sont utilisés par nombre d'autres constructeurs. Ces programmes sont devenus un standard.

- Le Network File System (NFS) permet de partager l'espace disque à travers le réseau. Ceci est particulièrement important pour votre répertoire-utilisateur. Avec NFS, il est possible que votre répertoire-utilisateur soit le même sur toutes les machines du réseau local. Physiquement, il reste sur le disque d'une seule machine. Les autres machines peuvent alors accéder au contenu de ce disque (`mount`), et il devient l'équivalent d'un disque local. La machine qui exporte (`export`) les fichiers de son disque local est appelée un serveur de fichiers. Ceci est utilisé pour certains programmes, car il est plus facile de mettre à jour une seule copie de ce programme, partagée à travers le réseau, que 20 réparties sur diverses machines. Souvent, il n'y a que très peu de programmes sur le disque local, le reste est stocké sur des disques reliés à une ou plusieurs machines.
- L'impression peut aussi être un service partagé sur le réseau. Ceci vous permet d'utiliser n'importe quelle imprimante sur le réseau local, quelque soit la machine à laquelle elle est physiquement raccordée.

Un réseau local ne ressemble pas forcément à ça. De toute façon, vous ne le trouverez pratique que s'il l'est réellement.

Chapitre 2

Les réseaux locaux

2.1 Les réseaux locaux

On peut les classer en trois grandes catégories, réseaux locaux informatiques, réseaux téléphoniques et réseaux large bande.

2.1.1 Les réseaux locaux informatiques (ou en bande de base)

Ils sont principalement destinés à l'acheminement des données numériques. Tous les matériels informatiques (terminaux, stations, mainframe, périphériques, ...) peuvent être reliés, en transmission bande de base.
Ils sont mal adaptés au transport de la voix et de l'image pour deux raisons :

- l'absence d'une garantie de temps de traversée du réseau et donc de synchronisation,
- la nécessité de débits importants (de 2 à 150 Mbits) pour le transport de l'image animée.

2.1.2 Les réseaux locaux téléphoniques (autocommutateurs privés)

Ils servent à véhiculer des voies téléphoniques. On les désigne souvent par leur nom anglais *PABX, PBX Private [Automatic] Branch Exchange*. Ils sont parfaitement adaptés au transport de la voix, réutilisent en général le câblage téléphonique, ce qui du fait de la mauvaise qualité des câbles peut imposer une limitation aux volumes d'informations numériques transportés.

La quatrième génération de PABX (on parle quelquefois de la troisième génération "et demie") traite de la même façon en commutation et en transmission les données et la voix (numérisée). Un PABX-4G peut apparaître comme une alternative à un réseau local. Les avantages sont énormes et leurs désavantages aussi. Le nombre de point de connexion est élevé et le débit global (jusqu'à 500 Mbits/s) est supérieur à l'offre actuelle des LAN. Cependant les échanges point à point sont limités à un débit de l'ordre de 64 Kbits/s (canal B du RNIS). Les PABX pratique la commutation de circuits adaptée à des trafics faibles, réguliers et continus, mais totalement inadaptée aux fortes rafales d'un transfert de fichier ou d'une image graphique.

Enfin un PABX est un dispositif centralisé de topologie étoilée et très complexe, dont l'arrêt est inconcevable.

En fait, PABX et réseaux locaux sont complémentaires. De nombreuses organisations installent les deux, le PABX prenant en charge le transport de la voix ainsi que les transferts de données à faible débit et le réseau local les communications haut débit.

2.1.3 Les réseaux locaux dits large bande

Ce type est à différencier des réseaux large bande numériques qui sont des réseaux nationaux à très grande bande passante.

Les informaticiens appellent réseaux large bande, les réseaux locaux câblés de diffusion de la télévision (en anglais CATV). Ces réseaux utilisent le multiplexage en fréquence où la bande passante est divisée en sous-bandes disjointes, chacune affectée au transport d'un certain type d'information (voix, donnée, image, ...).

Chaque appareil sur le câble est équipé d'un modem particulier, adapté au type d'information transmis.

Ce type de réseau est particulièrement adapté au transport simultané de toutes les informations de l'entreprise mais nécessite des équipements coûteux et distincts pour chaque application.

2.1.4 Décomposition d'un réseau local

[Ser97] définit un réseau local comme un ensemble d'équipements ou nœuds reliés entre eux pour échanger des informations et partager des ressources physiques, ce qui implique :

- un cablâge reliant les différents nœuds selon une certaine topologie,
- une méthode d'accès au support pour assurer son partage,
- une méthode d'adressage pour identifier chaque nœud,
- un ensemble cohérent de protocoles (pile) pour permettre la communication,
- un système d'exploitation réseau (*NOS, Network Operating System*) capable de prendre en charge les périphériques distants partagés et d'en contrôler l'utilisation (administration et sécurité),
- un ensemble de programmes utilisant les ressources mises en commun.

2.2 Technologie d'un réseau Ethernet

2.2.1 Composants

Un réseau local Ethernet a principalement cinq composants.

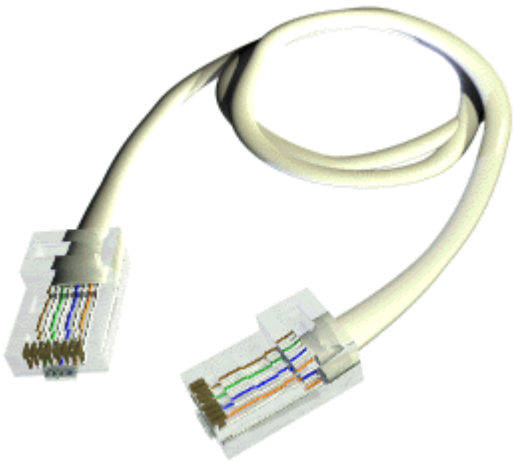
Le support physique de transmission

Les performances attendues du système déterminent le débit escompté et donc la bande passante nécessaire. Les principaux supports sont les paires métalliques, le câble coaxial et la fibre optique.

La prise

La prise est chargée de la connexion mécanique : elle permet le branchement sur le support.
Le type de prise dépend du support, du mode de diffusion et de la topologie.

De plus en plus souvent, les prises utilisées sont du type prise téléphonique type RJ45 (celles qui relient la ligne au poste). C'est le cas du réseau Ethernet sur paires torsadées ou 10BAST. Les liaisons utilisent des paires métalliques d'au plus 100 mètres et sont de type point à point. La topologie est en général en étoile ou arborescente, construite autour de boîtiers passifs ou actifs (hubs ou switchs) (capables d'isoler une liaison point à point défaillante par exemple).



Pris de www.3com.com/ solutionstechno

FIGURE 2.1 – Un câble RJ45-RJ45

Dans le cas de câble coaxial utilisé en bande de base, on rencontre deux modes principaux de raccordement. Le premier consiste à sectionner le câble et à raccorder les deux tronçons à une prise en forme de T qui réalise une dérivation vers l'équipement à connecter (cas du câble Ethernet dit fin ou 10BAS2 : les brins font 200 mètres maximum avec 30 stations au plus par brin). Le sectionnement nécessite l'arrêt du réseau pendant la durée de raccordement, ce qui peut être insupportable. Cependant, plus il y a de tronçons de câble raccordés par des prises en T, plus cela entraîne des modifications électriques du signal.

Pris de www.3com.com/ solutionstechno

FIGURE 2.2 – Connecteur BNC

Le second mode de raccordement ne présente pas ces inconvénients mais sa mise en place doit être effectuée avec soin. Elle consiste en l'installation sur le câble d'une prise dite vampire qui sert de support à une sonde très fine pénétrant jusqu'au coeur (ou âme) du coaxial et permettant à une partie de l'énergie circulant sur le câble d'être dérivée vers l'équipement à connecter. En principe, le retrait d'une telle prise n'affecte pas les caractéristiques du câble. Les câbles utilisés avec des prises vampires sont robustes, relativement rigides et de forte dimension (cas du câble Ethernet dit épais ou jaune ou 10BAS5 : les brins font 500 mètres maximum avec 30 stations au plus par brin).

Dans le cas de câble coaxial utilisé en large bande, l'ensemble prise-adaptateur remplit les fonctions de modulation et démodulation. De plus les réseaux large bande nécessitent l'utilisation d'amplificateurs afin de régénérer les signaux sinusoïdaux. Or ces amplificateurs sont unidirectionnels et la présence de l'un d'eux entre deux ordinateurs limite la transmission des signaux à un seul sens. Pour autoriser des échanges bidirectionnels soit :

- on double les câbles, un est raccordé aux émetteurs (modulateurs), l'autre aux récepteurs (démodulateurs),
- on utilise un seul câble avec deux bandes de fréquence, une affectée au canal d'émission de tous les ordinateurs, l'autre au canal de réception.

Dans les deux cas, on trouve à une des extrémités du réseau un dispositif appelé tête de bus ou tête de pont (head end) qui assure la liaison entre le canal d'émission (câble ou bande de fréquence) et le canal de réception (idem). Le point sensible des réseaux large bande est la tête de bus dont la défaillance entraîne l'arrêt complet du réseau.

Les liaisons à base de fibre optique nécessitent une technologie plus complexe. La difficulté majeure réside dans la réalisation des dérivations nécessaires aux raccordements. Une solution contournant cette difficulté consiste à réaliser des réseaux en anneaux constitués de liaison point à point reliant deux à deux les ordinateurs du réseau local. Les prises de raccordement sont actives, c'est à dire que les signaux optiques reçus sont transformés en signaux électriques, transmis à l'ordinateur puis régénérés en signaux lumineux pour être transmis sur le tronçon optique aval. L'inconvénient majeur est que la défaillance d'un récepteur actif entrave la mise hors service du réseau local.

Une autre solution utilise des prises passives constituées de deux dérivations, l'une alimente la prise en direction de l'ordinateur et l'autre le tronçon suivant du réseau. On sait faire des dérivations mais l'affaiblissement associé est élevé.

L'adaptateur

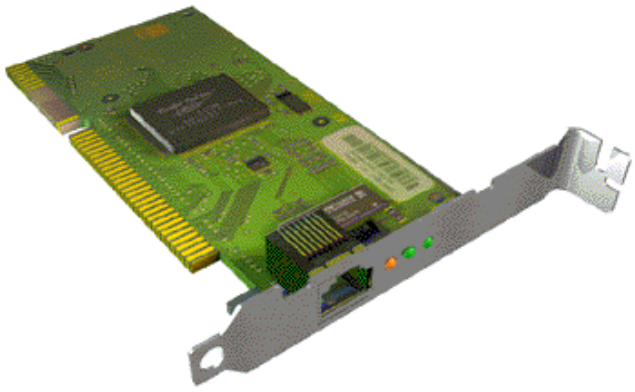
L'adaptateur est responsable de la connexion électrique. Il est chargé de la sérialisation et désérialisation des paquets, de la transformation des signaux logiques en signaux transmissibles sur le support, de leur émission et de leur réception (un modem peut être vu comme un adaptateur). Dans le cas d'Ethernet l'ensemble prise-adaptateur s'appelle un transceiver (abréviation de transmitter-receiver). Dans les normes IEEE de la sous-couche MAC, le transceiver est appelé *MAU, Medium Attachment Unit*.

Pris de www.3com.com/˜solutionstechno

FIGURE 2.3 – Transceiver

Le communicateur (ou coupleur de réseau)

Cet organe est chargé de contrôler les transmissions sur le câble. Il doit gérer la procédure d'accès au support physique ainsi que le protocole de liaison. Le communicateur assure le formatage et le déformatage des trames, souvent la détection d'erreur mais rarement la reprise sur erreur. Il comporte un microprocesseur qui contrôle les échanges avec l'adaptateur, ainsi que l'interface avec l'extérieur. Notons que le terme communicateur fait souvent référence à l'ensemble communicateur-adaptateur (surtout sur les PC actuels où le transceiver est intégré avec le coupleur, sur une carte distincte ou sur la carte mère).



Pris de www.3com.com/~solutionstechno

FIGURE 2.4 – Coupleur Ethernet

L'interface vers l'utilisateur

Cet élément constitue la prise de connexion du coupleur. Dans le cas d'un PC ou d'une station, le coupleur peut être sur le bus ou accessible via une interface. Il existe une interface standard : l'interface V24 (elle est aussi utilisée comme standard entre l'adaptateur et le communicateur).

2.2.2 Mode de transmission et codage

Trois approches sont possibles pour le transport de l'information :

- Soit l'information est sous forme numérique, c'est le mode bande de base.
  Il n'y a pas de modulation nécessaire. La suite binaire est directement transmise sur le support par des changements discrets dans les signaux représentant l'information (transition de tension). Les signaux bande de base sont sujets à l'atténuation et doivent être régénérés. Selon le choix du medium et de la topologie sous-jacente, l'IEEE a normalisé une série de réseaux Ethernet : 10BAS5 (Ethernet jaune ou épais), 10BAS2 (Ethernet brun ou fin), 10BAST (Twised-Ethernet, Ethernet sur paire torsadées).
- Soit chaque type d'information se voit allouer une bande passante en fonction de ses besoins et différente des autres : c'est le mode large bande. Cette méthode utilise le multiplexage en fréquence, ceci permet des transmissions simultanées indépendantes. La transmission ne peut s'effectuer dans deux directions simultanément aux mêmes fréquences. Comme on l'a vu, on utilise deux câbles distincts ou deux canaux de fréquences séparées pour l'émission et la réception. L'IEEE a normalisé ce type de réseau Ethernet large bande sous l'appellation 10BRO36.

– Soit la couche physique est réalisée avec de la fibre optique ; selon les techniques employées on peut voir la transmission comme étant en bande de base (changements discrets d'une impulsion lumineuse) ou comme étant en large bande, surtout lorsque le spectre optique est divisé en quelques sous-canaux (*WDM, Wave length Division Multiplexing*), ou encore DWDM (*Dense WDM*) lorsque le nombre de longueurs d'onde est important.

Les systèmes bande de base sont plus simples à installer et moins coûteux que les systèmes large bande qui offrent un plus haut débit et des distances de transmission plus grandes.

2.2.3 Partage du canal

Adressage

Chaque interface Ethernet a une adresse formée de 6 octets garantie unique mondialement par partage entre les constructeurs. Une adresse Ethernet est écrite sous la forme xx :xx :xx :xx :xx :xx, où x est un caractère hexadécimal. Ainsi Sun a des adresses de la forme 8 :0 :20 :xx :xx :xx, HP a 8 :0 :9 :xx :xx :xx, NCD a 0 :0 :a7 :xx :xx :xx.

Un paquet Ethernet est formé de :

– 8 octets de préambule (7 fois 0xaa + 1 fois 0xab) : destiné à synchroniser les horloges entre machines.
– l'adresse Ethernet du destinataire. Ce peut être soit l'adresse Ethernet de la machine destinataire soit ff :ff :ff :ff :ff :ff. Ce dernier cas est un broadcast, i.e. un paquet émis à destination de toutes les machines.
– l'adresse Ethernet de l'émetteur du paquet.
– les données.
– un check-sum.

Partage du bus

Ethernet a d'abord été conçu pour un support physique de type bus. L'ethernet sur paire torsadées (10BAST) utilise un élément actif appelé hub [1] dont le rôle est de simuler un bus. Bien que chaque station soit seule sur son brin de paires torsadées, lorsque deux stations émettent en même temps, la collision est détectée par le hub et propagée sur les autres brins [2].

L'algorithme de partage du canal est CSMA/CD :

– si le bus est occupé (i.e. il y a un signal dessus), on attend la fin et 10 μs après la fin, on se met à émettre. Si le bus est libre, on se met à émettre. (Carrier Sense Multiple Access(CSMA))
– pendant les 10 μs du début d'émission, on regarde si le signal que l'on essaye de mettre est celui que l'on entend, autrement dit on regarde s'il y a collision. (principe de Collision Detect (CD)). Ces 10 μs correspondent à environ 3km de distance : effectivement un câble Ethernet est limité à 5 brins de 500m maximum.
– si aucune collision n'est détectée, la machine considère le paquet comme émis.
– s'il y a collision (i.e. si deux machines se sont mises à émettre simultanément), on arrête et on attend un temps aléatoire pris dans l'intervalle [0,T] avant de réessayer.
– si une seconde collision est détectée, on attendra un temps aléatoire pris dans [0,2T].
– pour une troisième collision, l'intervalle sera [0,4T].
– etc. jusqu'à 16 essais, l'intervalle étant majoré par [0,1024T].

1. *hub*, moyeu
2. Un commutateur, ou *switch* ne propage pas les collisions.

– au bout de 16 essais, on abandonne et on remonte une erreur.

L'étude mathématique montre que cette méthode est équitable, et résiste bien à une charge élevée.

Un paquet a :

– une taille minimale : 64 octets. Afin que 2 machines aux 2 extrémités du réseau soient sûres de détecter la collision.
– une taille maximale : afin qu'une machine ne puisse prendre le bus pour elle seule pendant trop longtemps. C'est 1536 octets. On parle ici de MTU (Maximal Transmission Unit).

Exemple sous Unix

```
$ netstat -ian
Name  Mtu  Net/Dest      Address        Ipkts  Ierrs Opkts  Oerrs Collis Queue
lo0   8232 127.0.0.1     127.0.0.1      604    0     604    0     0      0
le0   1500 129.199.96.17 129.199.96.17  1631322 6     1289914 2     106071 0
```

Ici le pourcentage de collisions sur Ethernet est 106071/1289914 = 8.22%, valeur très élevée signe de saturation. Les erreurs en entrée sont en général dues à des parasites électriques, celles en sortie à des problèmes de transceivers.

Une vision plus moderne

La commutation Ethernet prend de plus en plus d'ampleur. Au lieu de simuler un bus (et donc de propager le trafic et les collisions . . .sur tous les câbles), seuls les liaisons nécessaires sont utilisées.

Contrairement au concentrateur (hub) où le segment est partagé entre tous les utilisateurs connectés sur chaque port, le commutateur "personnalise" chaque port en limitant le trafic par "port" aux seuls utilisateurs connectés sur chaque port.

Chaque "paquet" reçu par le commutateur est mémorisé totalement (Store and Forward) ou partiellement (cut-through) puis envoyé uniquement vers le port sur lequel la station, correspondant à l'adresse de destination indiquée dans le paquet, est détectée.

Les performances d'un commutateur résident dans la capacité de traitement des paquets par seconde, sa capacité de stockage des paquets et le temps de traitement par paquet (latency).

La compatibilité avec CSMA/CD est effectuée (de manière à pouvoir migrer progressivement les installations existantes). Cela pose des problèmes avec l'Ethernet 1000 MBit/s, les trames trop courtes doivent être artificiellement prolongées à 512 bits, ce qui rend ce type de réseau inadapté pour le transport de petits échantillons (comme la voix).

2.3 Interconnexion

L'interconnexion est née de la nécessité de mettre en relation des machines appartenant à des réseaux différents. L'interconnexion ne se limite pas au niveau physique, en fait chaque couche du modèle ISO peut utiliser des relais d'interconnexions (le terme le plus générique employé est passerelle, mais on devrait limiter son emploi à la couche 7, aussi il est préférable d'employer le terme de relai).

Les types de matériels les plus utilisés sont :

– Répéteur (*repeater*)

- Multirépéteur (étoile, *hub*)
- Pont (*bridge*)
- Commutateur (*switch*)
- Routeur (*router*)
- Passerelle (*gateway*)

2.4 Répéteurs

2.4.1 Généralités

Les répéteurs réalisent une connexion physique entre deux réseaux possédant un niveau de trame commun (même couche 2). Un répéteur travaille au niveau de la couche physique (ISO 1), ce qui signifie qu'il ne regarde pas le contenu de la trame [3]. Un répéteur ne fait que retransmettre d'un côté les bits reçus sur l'autre ; il ne peut que réaliser la prolongation ou l'adaptation d'un support (d'un support 10BAST vers un support coaxial ou optique, par exemple) ou l'isolation électrique (galvanique).

Comme avantages, on trouve :
- débit en général égal à celui des réseaux interconnnectés,
- simplicité d'installation,
- pas d'administration.

Les désavantages sont les suivants :
- un répéteur ne diminue pas la charge du réseau,
- un répeteur ne filtre pas les collisions.

2.4.2 Exemple d'un répéteur Ethernet

Un répéteur Ethernet interconnecte des segments, il n'a pas d'adresse Ethernet. On l'utilise généralement pour augmenter la distance maximale entre 2 stations en reliant 2 segments Ethernet, mais aussi pour adapter deux segments de technologie différentes.

Selon la norme, les réseaux Ethernet sont soumis à des contraintes de peuplement (nombre de stations par segment), de taille (longueur des segments), de structure (nombre maximum de segments peuplés ou non peuplés). Les fonctions d'un répéteur Ethernet sont les suivantes :
- faire passer tous les signaux (pas uniquement les bits "corrects") émis sur un segment sur l'autre ;
- le délai de propagation au travers du répéteur est environ < 7.5 temps-bit (0,125 μs dans le cas d'un Ethernet à 10MBit/s) ;
- régénérer électroniquement le signal ;
- reformer le préambule de 56 bits (si nécessaire) ;
- ajouter du padding si la trame est < 12 octets ;
- si il existe une collision sur un segment, générer uun signal appelé "jam" (32 bits) sur les 2 segments ;
- disposer d'une fonction Jabber (trames trop longues tronquées) ;
- disposer d'une fonction SQE test.

Un répéteur Ethernet est connecté sur les segments coaxiaux comme une station :
- un répéteur peut être au milieu du câble ;
- le raccordement : câble de transceiver + Transceiver ;

3. Agissant au niveau physique, les réseaux interconnectés doivent être homogènes.

– il suit les règles d'emplacement d'une station (2.5 m) ;
– il y a un maximum de 4 répéteurs entre 2 stations.

2.4.3 Concentrateur

FIGURE 2.5 – Un hub à 8 ports

Aussi appelé étoile, hub, multirépéteur

Un concentrateur assure une fonction de répéteur dans une topologie en étoile (obligatoire avec la fibre optique et la paire torsadée).

La fonction de segmentation s'est généralisée, à l'extrême il n'y a qu'un équipement par segment. Ce sont des éléments souvent modulables (sous la forme de cartes dans un rack par exemple, avec un type de cartes par media). Dans le cas d'un hub Ethernet, il n'a pas d'adresse Ethernet.

2.5 Pont

2.5.1 Généralités

Les ponts (*briges*) permettent une interconnexion entre deux ou plusieurs réseaux (ponts multiports) dont les couches physiques sont dissemblables. Chaque réseau (ou brin) est relié au pont via un port. Un pont travaille au niveau de la couche liaison (ISO 2), ce qui signifie qu'il est transparent aux protocoles de niveau supérieur.

Agissant au niveau 2, les ponts ont accès à l'adresse MAC et peuvent acheminer les trames en fonction de cette adresse (on parle quelquefois de "routage de niveau 2"). Les ponts ne peuvent interconnecter que des réseaux dont l'espace d'adressage est homogène. La couche 2 étant divisée en 2 sous-couches dans le modèle de référence IEEE, on peut distinguer deux sortes de ponts, selon que les couches MAC sont compatibles ou non :

– un pont MAC relie des réseaux du même type (Ethernet, Token-ring, ...) ;

– un pont LLC remonte la trame au niveau LLC, puis réencapsule la trame dans la couche MAC ; le niveau MAC peut être différent d'un sous-réseau à l'autre (pont Ethernet/Token-ring, ...).

Le travail du pont s'appuie sur des tables qui stockent la localisation des stations sur ses différents ports. Il existe trois modes de fonctionnement :

– Auto apprentissage (*learning bridge*), qui lui permet de déduire quelles sont les stations présentes sur chaque port à partir de l'observation des trames echangées ;
– Table figée avec les adresses des stations, la reconfiguration étant à la charge des administrateurs-systèmes ;
– Mixte avec des filtres manuels permettant d'isoler ou de limiter l'accès de ou à certaines stations.

Un pont réalise généralement un filtrage automatique (on parle de pont filtrant) : les trames echangées entre deux stations situées sur le même port ne traversent pas le pont. Les trames nécessitant un acheminement au travers du pont sont généralement stockées puis réémises (*store and forward*), un pont dispose donc d'un processeur et de mémoire. Grâce à la fonction de filtrage, chaque brin peut être isolé du trafic des autres brins. Cette faculté associée à la non-retransmission des trames erronées et des collisions, permet de découper un réseau physique en plusieurs sous-réseaux logiques. Chaque port d'interconnexion est une interface MAC. Sur chaque sous-réseau, un pont se comporte comme une station du sous-réseau et dispose d'une adresse MAC.

Certains protocoles ne peuvent utiliser que des ponts, comme LAT (DEC).

Un réseau interconnecté avec des ponts doit s'assurer que les trames ne bouclent pas indéfiniment.

Pour cela, l'IEEE 802.1 a normalisé un algorithme du *spanning tree* (arbre recouvrant) pour éviter les boucles. Ce protocole permet de constituer, à partir de n'importe quelle topologie physique, un arbre interconnectant tout le réseau, en minimisant une fonction de coût (qui peut être fonction de la distance ou du débit de chaque liaison ou du tarif associé au débit ...).

Il existe aussi une technique d'origine IBM, utilisée dans les réseaux Token-ring, le routage par la source (*Source Routing*). Lors du premier échange entre deux stations, une trame de découverte du chemin est émise et peut être dupliquée lorsqu'il existe plusieurs chemins possibles. Lorsque les trames de découverte arrivent à destination, le destinataire choisit le meilleur chemin et renvoie une trame à l'expéditeur. Par la suite, toutes les trames échangées contiennent la route complète à suivre. Les ponts Token-ring n'entretiennent donc aucune table d'acheminement, ils se contentent d'aiguiller les trames selon les informations contenues dans l'en-tête de la trame.

2.5.2 Exemple d'un pont Ethernet

Les fonctions d'un pont Ethernet sont les suivantes :

– augmenter la distance maximum entre 2 stations Ethernet,
– interconnecter deux sous-réseaux physiques différents,
– diminuer la charge des réseaux en isolant le trafic sur chaque sous-réseau.

Un pont Ethernet a deux ou plusieurs adresses Ethernet.

Avantages

– Débit presque 10 Mb/s
– Filtre les trames inutiles et les collisions
– Pas de limite de distance

- Peu d'administration

Désavantages
- Ne filtre pas les broadcast ou multicast
- Supplanté par le routeur

2.6 Commutateurs

2.6.1 Principes

- Fonctionnement type "ponts"
- Processeurs spécialisés (ASICS,..)
- Ports avec bande passante "dédiée" et non partagée
- Communication port à port parallèles entre elles.

2.6.2 Méthodes de commutations

La commutation "On the fly" ou "Cut throught"

- Arrivée de la trame
- Lecture des premiers octets de la trame
  - début en-tête Eth
- Commute la trame vers le destinataire en fonction de l'adresse destination
- Avantages
  - temps de latence très faible
    - atteint 15 micro-secondes
    - indépendant de la longueur de la trame
- Inconvénients
  - Retransmission des erreurs
    - CRC, fragments de collision
  - Impossibilité de commutter 10/100/uplink ATM

La commutation "Store & Forward"

- Arrivée de la trame
- Stockage de la trame
- Commutation vers le port de sortie
- Avantages
  - Traitement des erreurs
  - Possibilité de traitements
  - Adaptée aux commutateurs 10/100/uplink ATM
- Inconvénients
  - Plus lent que la commutation "on the fly"
  - Temps de latence fonction de la longueur de la trame

2.6.3 Critères de choix

- Architecture
  - "switch fabric"

  - matrice de commutation (Crossbar, Batcher-Banyan
  - mémoire partagée
  - bus partagé (TDM)
- bande passante globale
  - actuellement de 500 Mbps à 4 Gbps
- taille des buffers pour les ports I/O
  - en entrée
  - en sortie
  - partagés

- Performances
  - Débit
    - vitesse maximale à laquelle le commutateur peut transmettre des paquets sans perte
  - Taux de perte
    - pourcentage de trames envoyées mais non retransmises par le commutateur dans une fenêtre de temps prédéterminée
  - Temps de latence
    - "Cut throught" 1er bit entrée - 1er bit sortie
    - "Store & Forward" dernier bit entrée - 1er bit sortie

FIGURE 2.6 – Un switch à 8 ports sur un hub à 8 ports

2.7 Routeur

Un routeur est un élément d'interconnexion de niveau 3 qui achemine (route) les données vers un destinataire défini par une adresse de niveau 3 (X121, IP, ...). Au départ, les routeurs étaient mono-protocoles (IP, IPX, Appletalk, DECNET, ...) et très dépendants du protocole ; aujourd'hui les routeurs sont multi-protocoles mais restent dépendants des protocoles supportés bien qu'on tende vers une standardisation du protocole de routage.

Un routeur est une unité de traitement disposant d'un CPU puissant, d'une bonne quantité de mémoire, d'un bus interne très rapide. Ce peut être un matériel dédié ou non (station ou PC utilisé comme routeur, le plus souvent IP).

Un routeur a une (en fait 2 au moins) adresse(s) MAC connue(s) des stations. Un routeur permet le relayage des paquets entre duex réseaux d'espace d'adressage homogène (IP/IP, Appletalk/Appletalk, ...), mais pas entre deux réseaux d'adressage hétérogène (IP/Appletalk par exemple), il faut alors utiliser une passerelle.

Les routeurs orientent les paquets selon des informations contenues dans des tables de routage. Ils utilisent essentiellement deux modes de routage :

- le routage statique ou fixe, où les tables sont introduites et maintenues par l'administrateur réseau ;
- le routage par le chemin le plus court, dans lequel les tables de routage indiquent, pour chaque destination, le chemin de coût le moins elevé ; périodiquement, des échanges d'informations entre routeurs permettent de maintenir ces tables à jour.

Il existe de nombreux protocoles de routage, certains normalisés par l'ISO, d'uatres utilisés dans l'internet et recommandé par l'IETF (*Internet Engineering Task Force*). Certains protocoles utilisent des algorithmes à vecteur de distance (*Distance Vector Routing*), d'autres des algorithmes à état des liaisons (*Link State Routing*).

Pour limiter l'espace de routage, les réseaux ont été subdivisés en domaine de routage, appelé *AS, Autonomous System*). Le problème du routage utilise alors souvent deux types de protocole, l'un à l'intérieur du domaine (routage intra-domaine), l'autre entre deux domaines différents (routage inter-domaine).

Avantages des routeurs

- Un routeur est un très bon filtre :
  - il ne laisse pas passer les trames inutiles, les collisions, les broadcasts, les multicasts
  - il offre la possibilité de gérer des tables de filtrages au niveau 3, soit sur les adresses, soit sur les protocoles routés, ...
- Un routeur permet de séparer proprement 2 administrations (deux entités).
- Une erreur au niveau d'un réseau physique d'un coté n'affecte pas les autres réseaux interconnectés
- Un routeur est un équipement connu par les protocoles de niveau 3, il peut être administré à distance (les ponts aussi).
- Les performances sont bonnes, de 10000 à 15000 paquets/s (elles avoisinent même les 100000 paquets/s pour les commutateurs FDDI et FastEthernet, sans parler de la génération en cours d'arrivée ...).

Désavantages

- Le coût d'un routeur est plus élevé que celui d'un équipement de plus bas niveau.
- La configuration n'est pas toujours aisée, le marché des certifications se développe (Cisco notamment).

2.8 B-Routeur

Les ponts-routeurs (*B-routeur, pour Bridge-routeur*) allient dans un même équipement des fonctions de pont (niveau 2) et de routeur (niveau 3) :

- Fonction de pont :
  - pour les protocoles non routables : LAT, TOKEN-RING
  - pour ce qu'il ne sait pas router
- Fonction de routeur : pour les protocoles reconnus

Ils répondent à tous les besoins, mais il faut savoir les configurer.

2.9 Passerelle

L'interconnexion d'un réseau avec un autre réseau peut nécessiter des mécanismes d'adaptation. Ces fonctions sont réalisée par une passerelle (*gateway*) :

- Traduire un protocole dans un autre
- Tout ce qui n'est pas un répéteur, un pont ou un routeur et qui permet l'interconnexion de réseaux
- Travailler sur les couches >= 3
- Permettre à 2 mondes de communiquer, souvent avec des légères pertes de fonctionnalités

Les passerelles sont utilisées pour :

- Conversion de protocole :
  - DECNET-IP
  - LocalTalk AppleTalk-Ethernet IP
- Conversions d'applications :
  - Telnet - SETHOST
  - Telnet - PAD
  - ftp - COPY
  - Messagerie :SMTP - EARN/RJE - X400

Le matériel peut être dédié ou être une application sur une station. Cela demande généralement beaucoup d'administration (tables ...).

Chapitre 3

Le réseau Internet

A l'initiative de la DARPA (*Defense Advanced Research Project Agency*), des recherches ont été menées dans le but de batir un réseau permettant l'interconnexion de machines hétérogènes. Ces recherches ont abouti à un certain nombre de protocoles, généralement désignés sous le terme *pile de protocoles TCP/IP*.

3.1 Modèle d'Internet

3.1.1 Interconnexion d'applications

Pour l'utilisateur final, Internet se présente comme un ensemble de services qui utilisent le réseau pour coopérer et communiquer. On utilise généralement le terme d'*interopérabilité* pour évoquer la capacité des systèmes à coopérer à la réalisation d'activités communes. Par exemple, un système de courrier électronique comprend un porgramme de diffusion qui expédie ses messages un à un vers divers ordinateurs. Le chemin entre la source et le destinataire peut emprunter plusieurs réseaux différents, cela est sans importance dans la mesure où les systèmes de courrier électronique sur chaque ordinateur interopèrent à l'expédition des messages.

Les utilisateurs du Web, du courrier électronique, de transfert de fichiers, de connexion à distance, de téléphonie, de vidéophonie, ...le font en exécutant des programmes d'applications, sans connaître les protocoles TCP/IP, l'organisation des réseaux sous-jacents, ni même le chemin emprunté. Seuls les programmeurs qui écrivent les applications voient l'Internet comme un système dont ils doivent comprendre les détails techniques.

3.1.2 Interconnexion de réseaux

Lorsqu'une application met en jeu des milliers de réseaux interconnectés, nul n'est capable de concevoir les programmes d'applications nécessaires à son fonctionnement. De plus, deux types complétement différents d'application peuvent reposer sur les mêmes mécanismes de communication. La complexité de la communication est maîtrisée grâce à la décomposition en couches, chaque couche étant mise en œuvre indépendamment. Chaque couche ou niveau réalise un sous-ensemble des fonctions nécessaires pour communiquer entre systèmes et repose sur la couche inférieure pour réaliser des fonctions plus primitives sans s'occuper des détails de ces primitives.

Il existe une couche dont le modèle ne peut être ignoré : c'est celle qui réalise la connectivité entre les différents équipements. Sans surprise, cette couche est appelée la couche Réseau dans l'architecture OSI. Il s'agit d'assurer et de contrôler l'acheminenement des informations de la source

vers la destination finale. Dans l'Internet, les informations sont découpées en paquets (appelés datagrammes) qui sont commutés de proche en proche, en fait de réseau en réseau.

Comment les réseaux sont-ils interconnectés ? Physiquement, deux réseaux différents ne peuvent être reliés l'un à l'autre que par l'intermédiaire d'un équipement spécialisé raccordé à chacun d'entre eux. Mais cet équipement ne réalise pas forcément le service d'interconnexion dont l'utilisateur a besoin, comme par exemple la coopération avec les ordinateurs de chaque réseau.

L'Internet nécessite que les équipements d'interconnexion soient capables de faire transiter les paquets de données d'un réseau à l'autre. Ces équipements, initalement appelés passerelle IP (*Internet gateway*), sont plus simplement nommés routeurs IP ou routeurs (*Internet router*).

L'Internet est donc un ensemble de réseaux hétérogènes interconnectés par des routeurs. Les routeurs connaissent des informations relatives à la topologie globale de l'Internet bien au-delà des réseaux qu'ils raccordent.

3.1.3 Décomposition conceptuelle

Entre les applications et l'interconnexion de réseaux, il existe une plateforme de transport qui est la charnière de la communication. C'est d'ailleurs au travers de cette couche de transport qu'est perçue le réseau sous-jacent : transport d'électricité, d'eau, de voyageurs, de marchandises, d'informations, de sons, d'images, ...

Conceptuellement, l'Internet assure trois types de services. On les représente généralement en strates pour indiquer leur dépendance (cf. figure 3.1.3). L'interconnexion de réseaux qui se matérialise par la remise de datagrammes (en mode non connecté dans l'Internet) est le service de plus bas niveau. Ce service est utilisé pour réaliser un service de transport (généralement fiable), utilisé par les services d'application.

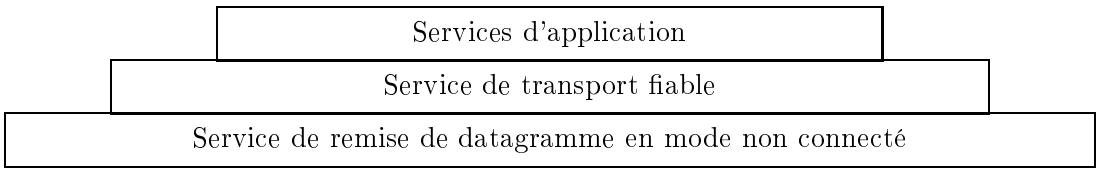


FIGURE 3.1 – Les trois couches conceptuelles de services de l'Internet

3.2 Architecture

La plupart des descriptions de TCP/IP définissent une architecture comportant 3 à 5 couches. Le modèle à 4 niveaux décrit à la figure 3.2 est une bonne représentation des protocoles TCP/IP.

On peut représenter les différents protocoles en 7 couches (voir fig. 3.2), sans qu'il soit tout à fait possible de comparer leur place avec leur équivalent dans le modèle d'architecture d'Interconnexion des Systèmes Ouverts (*modèle OSI*[1]) de l'Organisation de Standardisation Internationale ISO[2].

1. Open Systems Interconnection
2. International Standard Organization

Applications	Applications et processus utilisant le réseau
Transport Hôte à Hôte	Transmission de données de bout en bout
Internet	Routage des datagrammes
Accès au réseau	Routines d'accès aux réseaux physiques

Figure 3.2 – Couches de l'architecture de TCP/IP

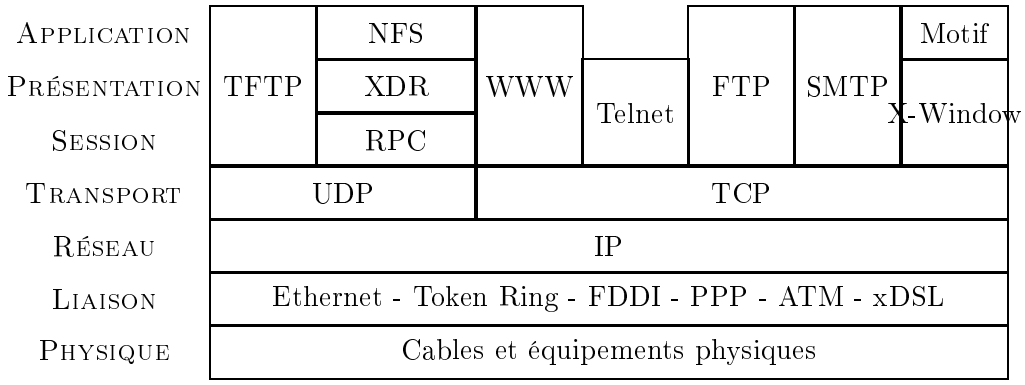


Figure 3.3 – Architecture de la pile de protocoles TCP/IP

3.3 La couche Accès réseau

Les protocoles de cette couche permettent de transmettre des paquets de données entre machines connectées au même réseau physique.

Les fonctions de cette couche comprennent l'encapsulation des paquets dans les trames transmises par le réseau et la mise en correspondance des adresses IP avec les adresses physiques qu'utilisent le réseau (protocole ARP, *Address Resolution Protocol* dans le cas d'IPv4).

3.4 La couche Internet

3.4.1 Le protocole Internet (*Internet Protocol*)

IP est un protocole d'interconnexion de réseau qui permet l'échange de *datagrammes*[3] en *mode non-connecté*[4]. IP ne garantit pas l'arrivée à bon port des messages, ni leur arrivée dans l'ordre d'émission, ni la détection d'erreurs.

Un datagramme est structuré selon le format de paquet utilisé par IP.

IP définit un plan d'adressage, qui permet d'identifier précisément chaque machine connectée à l'Internet.

3. lettre-grammes, si on veut parler français ... mais personne ne le fait

4. à la manière du courrier classique, le destinataire peut ne pas lire son courrier et son datagramme l'attend à la poste ou dans sa boite aux lettres

Lorsque l'émetteur et le destinataire n'appartiennent pas au même réseau, les messages transitent par plusieurs passerelles ou *nœuds de routage*.
L'algorithme de routage repose sur des *tables de routage* gérées par chaque passerelle.

IP est synonyme de IPv4, *Internet Protocol, version 4*. IPv4 est le protocole en service depuis le début des années 80. IPv4 fournit le service de livraison de paquets (datagrammes) pour TCP, UDP, ICMP et IGMP.
Dans le modèle ISO, IP correspond au niveau 3 (réseau). En réalité, il assure bien des fonctions de niveau 3 comme l'interconnexion mais peu d'autres fonctionnalités de ce niveau.

IPv6, *Internet Protocol, version 6* a été conçu au milieu des années 90 pour remplacer IPv4. Le changement le plus visible est que les adresses IPv6 sont sur 128 bits pour faire face à l'explosion de l'Internet. IPv6 joue réellement le rôle d'un protocole de niveau 3 : de nouvelles fonctionnalités ont été conçues notamment pour garantir des délais de transmission aux applications en temps réel et aussi pour transporter les paquets d'une extrémité à l'autre avec une certaine sécurité.
IPv6 fournit le service de livraison de paquets pour TCP, UDP, ICMPv6.

3.4.2 Le protocole ICMP (*Internet Control Message Protocol*)

Ce protocole utilise les fonctions de transmission de datagrammes IP pour envoyer des messages qui réalisent les fonctions de contrôle de congestion des passerelles, de détection des destinations inaccessibles, de redirection des routes et de vérification des machines hôtes à distance (*ping*).

Le *multicasting* ou multipoint permet l'envoi de datagrammes à plusieurs destinations efficacement. IP utilise les adresses de classe D et la notion de groupe de diffusion, si cela existe. La gestion des groupes par les passerelles est effectuée grâce au protocole IGMP (*Internet Group Management Protocol*).

ICMPv6, *Internet Control Message Protocol*, version 6, combine les fonctionnalités de ICMPv4, IGMP et ARP.

3.5 Les protocoles de la couche transport

3.5.1 Le concept de port

Les deux protocoles de la couche *transport* UDP et TCP ont pour but la communication entre activités (processus) sur des machines distantes. Dans l'Internet, il est évidemment nécessaire de permettre le dialogue entre activités sur des systèmes différents. La désignation des processus étant différente d'un système à l'autre, il faut définir une méthode d'identification indépendante des systèmes.

Le principe retenu est de séparer l'identité du processus réalisant une fonction donnée de cette fonction elle-même :
la notion de *service* est privilégiée par rapport à celle de processus.
Un même processus peut assurer plusieurs services et un même service peut être assuré par plusieurs processus.

Le concept Internet correspondant est celui de *port*.
Pour chacun des deux protocoles de la couche transport, il existe un ensemble de port identifiés

par un nombre entier.
A chaque application Internet standard correspond un numéro de port spécifique. C'est par l'intermédiaire des numéros de port que les modules TCP et UDP réaliseront le multiplexage.

3.5.2 User Datagram Protocol (UDP)

Il permet une application d'envoyer des messages à une autre en mode datagramme non connecté. Il fournit le même type de service que IP et donc ne garantit ni l'arrivée ni l'ordre. Il n'y a pas de mécanismes d'acquittement, ni de contrôle de flux en cas d'engorgement du réseau.

UDP utilise indifféremment IPv4 ou IPv6.

3.5.3 Transmission Control Protocol (TCP)

Il s'agit d'un protocole orienté connexion [5], qui offre un service **sûr** de transport de flot d'octets : les octets émis d'un côté de la connexion sont délivrés dans le même ordre de l'autre côté.

Le protocole repose sur des mécanismes d'acquittement dans un laps de temps donné, et de réémission si l'accusé de reception n'arrive pas dans le temps imparti. Les duplications et le séquencement sont gérés avec des numéros d'ordre.

Comme UDP, TCP utilise indifféremment IPv4 ou IPv6.

3.6 La couche haute des applications

3.6.1 RPC et XDR

Les RPC (Remote Procedure Call) mettent en jeu un ensemble de mécanismes qui permettent l'exécution de procédures sur une machine distante, à partir d'une invocation demandée par la machine locale et permettant de récupérer le résultat de l'exécution distante sur la machine locale. Les RPC utilisent les protocoles TCP/IP ou UDP/IP. L'interfaçage réseau (*socket*) est complétement transparent pour le programmeur.
L'utilitaire *rpcgen* de Sun permet de mettre en place rapidement un système Client (local) /Serveur (local ou distant) RPC.

XDR (eXternal Data Representation) permet une représentation des données indépendante de la machine. De façon générale, les données "locales" sont codées dans le format XDR par sérialisation, transmises à la machine "distante", puis décodées par désérialisation. On s'affranchit ainsi de la contrainte de représentation des données qui reste propre à une machine.

Les RPC utilisent XDR pour la Sérialisation/Désérialisation des informations (données et résultats).

3.6.2 Applications usuelles

Un certain nombre d'applications de haut niveau s'appuie soit sur UDP, soit sur TCP.
- FTP pour le transfert fiable.
- TFTP pour un transfert de petits fichiers basé sur UDP.
- SMTP pour l'échange de courrier entre machines distantes.

5. le correspondant doit être présent et accepter la connexion

- Telnet qui permet de faire d'un terminal physiquement connecté à un système un terminal "logique" d'un autre et donc de travailler à distance (on parle de *terminal virtuel*).
- NFS et RFS qui sont des systèmes de gestion de fichiers répartis. NFS, developpé par Sun Microsystems a été repris sur de nombreuses plate-formes. Il s'appuie sur une représentation standard des objets (protocole XDR) et un mécanisme d'appels de procédures à distance (protocole RPC).
- X-Window est un protocole s'appuyant sur les protocoles Internet et permet la gestion de dispositifs d'affichage sur le réseau.
- sans oublier le Web ...

3.7 La qualité de service sur l'Internet

On désigne par le terme de multimédia un produit ou un service qui mêle, grâce à une traduction numérique, des données jusqu'ici exploitées séparément : du texte, des sons, de la vidéo, des photos, des dessins, ...[LM98]

3.7.1 Deux natures de trafic

L'Internet et le protocole IP ont été conçus pour fournir un service de livraison de datagrammes, qualifié de "pour le mieux" (*best-effort*). Sur ce schéma du "best-effort", l'Internet (ou un Internet privé [6]) traite les paquets équitablement. Lorsque le trafic croît sur le réseau et qu'il y a des congestions, toutes les livraisons de paquet sont ralenties. Si la congestion devient sévère, les paquets sont perdus plus ou moins aléatoirement pour résorber la congestion. Aucune distinction entre les paquets n'est faite, que ce soit en terme d'importance d'un flot de paquets, ou que ce soit en fonction d'exigences sur l'arrivée temporelle des paquets d'un flot.

Malheureusement, les besoins des utilisateurs ont changé : en plus du trafic habituel des données entre réseaux, de nouvelles applications multimédia, en temps réel, en diffusion, apparaissent. Le seul réseau conçu pour un trafic temps-réel est ATM, mais il est peu probable de construire un deuxième réseau mondial cohabitant (ou remplaçant) le réseau Internet basé sur TCP/IP.

Le trafic sur un réseau peut être réparti en deux grandes catégories : élastique ou inélastique.

Un trafic élastique peut s'adapter aux variations de délai et de débit de l'Internet et cependant satisfaire le besoin des applications. C'est le trafic typique supporté par TCP/IP et pour lequel l'Internet a été conçu. Le courrier électronique, le trasfert de fichier, la connexion à distance, la gestion de réseau, le Web sont des applications qui s'accommodent d'un trafic élastique, bien qu'à des degrés différents.

Un trafic inélastique ne peut facilement s'adapter, voire pas du tout, aux variations de délai et de débit de l'Internet. L'exemple-type est le trafic temps-réel comme la voix ou la vidéo. Un trafic inélastique a, au moins, les exigences suivantes :

débit Un débit minimum peut être demandé. A la différence des applications élastiques qui peuvent continuer à délivrer des données avec un service dégradé, la plupart des applications inélastiques nécessite un débit minimum garanti.

délai Un délai maximum peut être demandé, au dessus duquel le service ne peut être assuré.

6. appelé en général Intranet (déployé sur un réseau privé) ou Extranet (déployé sur plusieurs sites reliés entre eux par un réseau virtuel)

gigue (variation de délai) Plus le délai maximum autorisé est grand, plus le délai réel est long et plus il est nécessaire de mémoriser les informations reçues de manière à réguler le trafic réel délivré.

pertes de paquets Les applications temps-réel varient dans leur capacité, si elle existe, à gérer la perte de paquets.

3.7.2 Défauts intrinsèques du réseau IP

IP est un réseau transmettant des paquets de données, dont les défauts de transmission pour des applications critiques comme la téléphonie sont les suivants :

le délai il doit rester de bout en bout inférieur à 400 millisecondes (aller-retour).

la gigue c'est la variation de délai. Le délai, même important, pourrait être constant ce qui préserverait la synchronisation de l'émetteur et du récepteur. Il n'en est rien, et les variations de délai détruisent la référence de temps du signal reçu, obligeant le destinataire à gérer une mémoire tampon pour rétablir la synchronisation.

les pertes de paquets elles sont chroniques et sont dans la nature même d'IP, on dit qu'IP délivre son meilleur effort pour la transmission mais sans aucune garantie de livraison. Dans les moments de congestion du réseau, le taux de perte dépasse le seuil de tolérance.

Deux autres défauts plus traditionnels n'engendrent pas autant de problèmes :

la qualité sonore lorsque le temps de transport aller-retour dépasse 40 ou 50 ms, des phénomènes d'écho deviennent perceptibles et génants. Ces phénomènes peuvent être éliminés par un traitement du signal.

la fiabilité des équipements l'industrie des télécoms est habituée à une très haute fiabilité (supérieure à 99,999%), les autocommutateurs ne tombent pratiquement jamais en panne. En revanche, les équipements réseaux ont une fiabilité n'excédant souvent pas 80 % et une indisponibilité de plusieurs heures par mois.

Le tableau 3.1, tiré de [Sus00], compare les deux architectures (circuits téléphoniques et réseaux de paquets IP) du point de vue de certains paramètres de QoS.

3.7.3 Qualité de service et IP

La QoS (Qualité de Service) est la capacité d'un élément réseau (une application, un hôte, un routeur) d'avoir un certain niveau d'assurance que la demande de trafic et les besoins de service puissent être satisfaites, même en présence de congestion dans le réseau. Les ressources disponibles n'étant jamais infinies, le déploiement de la QoS passe forcément par un dimensionnement adapté des réseaux, afin de toujours bénéficier de suffisamment de ressources pour offrir des garanties (en terme de perte de délais etc.) aux différents applications.

Si l'utilisation de l'IP ne peut aujourd'hui être remise en question, il est néanmoins nécessaire d'étendre l'architecture Internet afin d'élargir l'éventail d'offres de services (intégrant notamment la QoS). Cette évolution des réseaux IP nécessite l'introduction d'une intelligence capable de différentier le trafic puis offrir différents niveaux de service à chaque classe de service identifiée.

Les deux solutions qui ont été proposées au sein de l'IETF sont :

Réservation de Ressources (Modèle Intserv)

	Télécoms/Circuits	Réseaux IP
Délais	Négligeables. Limités au temps d'acheminement et au passage dans les commutateurs (inférieur à 1 ms par nœud).	Importants. Entre 200 ms et 400 ms pour un aller simple, sur l'Internet, voire plus. Entre 50 et 150 ms sur des réseaux locaux ou des réseaux privés bien gérés
Gigue	Nulle. Les octets parviennent à des intervalles fixes de 125 μsecondes.	Importante. De 50 à 100 ms sur l'Internet, inférieure sur un réseau local. Dépend de l'état de remplissage des buffers des nœuds.
Perte d'information	Négligeable pour la voix.	Importante. De 3% à 30% des paquets sont perdus et irrécupérables.
Qualité sonore	Bonne. Pas d'écho, de diaphonie, de parasites. Son MIC non compressé	Moyenne. Echo important corrigé par des procédés d'annulation. Son compressé de qualité GSM à améliorer.
Fiabilité	Excellente. Fiabilité à 5 chiffres : 99,999%	Moyenne. De type informatique, à améliorer.

Table 3.1 – Défauts comparés des réseaux télécoms et IP

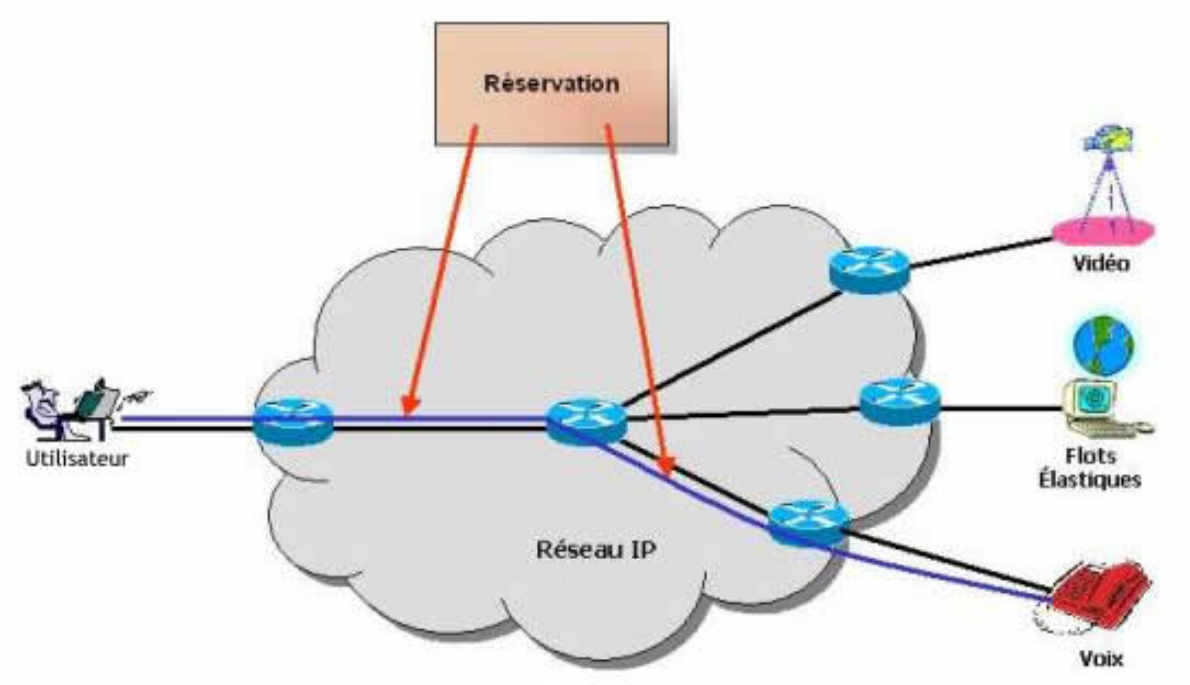
Les ressources du réseau sont réparties par rapport à des demandes explicites des applications. La demande est satisfaite et une réservation est effectuée si suffisamment de ressources sont disponibles. RSVP est utilisé dans ce contexte comme protocole de signalisation.

Différentiation par Priorités (Modèle Diffserv) Le trafic entrant dans le réseau est classifié et se voit attribuer des ressources, en fonction des critères de gestion du modèle de service. Aucune réservation n'est faite, mais un dimensionnement adéquat assure qu'il aura assez de ressources dans le réseau pour les demandes de toutes les applications. Les garanties données par le modèle vont dans le sens du partage des ressources disponibles. Pour offrir un certain niveau de QoS, les classifications donnent un traitement différentiel à des applications sensées avoir des besoins plus exigeants.

Les deux solutions proposées ne sont pas mutuellement exclusives ou concurrentes, mais bien au contraire, elles sont complémentaires. Ces solutions ont d'ailleurs été pensées pour travailler ensemble afin de faire face à des situations opérationnelles diverses et potentiellement complexes dans des contextes de réseaux variées.

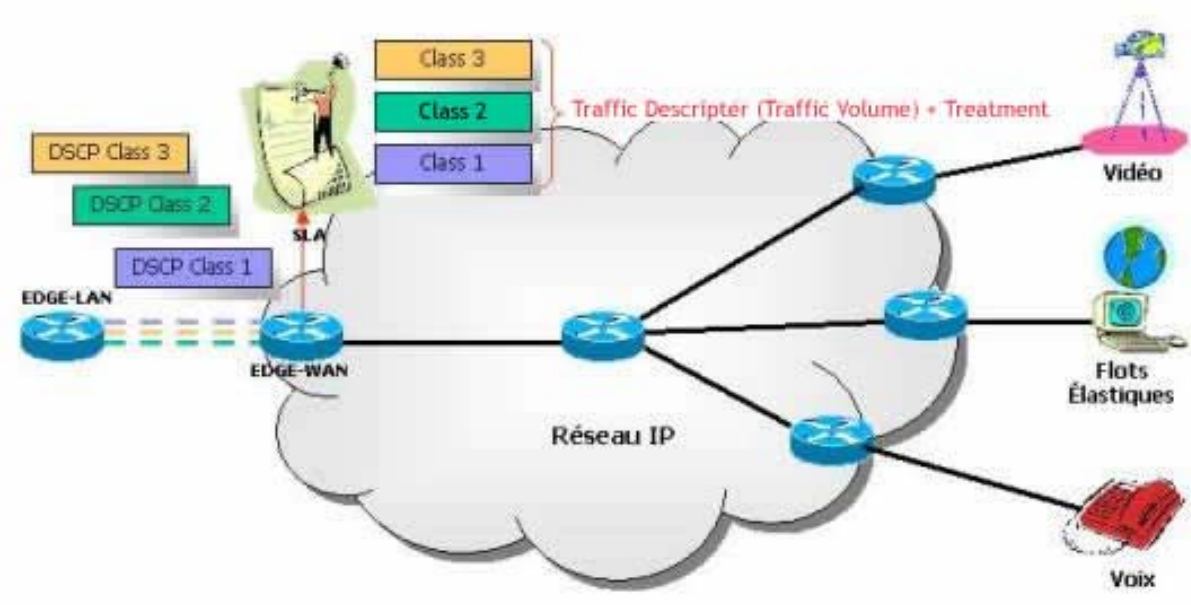
3.7.4 Modèle de QoS

Les architectures de QoS sont en cours de discussion au sein de l'IETF (Internet Engineering Task Force). Un modèle de services à trois classes a été proposé, en utilisant la solution Diffserv



Pris de www.vthd.org

FIGURE 3.4 – Réservation de ressources



Pris de www.vthd.org

FIGURE 3.5 – Différentiation de services

(Differentiated Services) dans le coeur du réseau. La grande quantité de bande passante déployée

sur un réseau haut débit rend possible l'utilisation et le développement d'applications nouvelles, conçues pour un Internet de nouvelle génération. Ces applications innovantes consomment de grandes quantités de bande passante et possèdent des contraintes de délai pour pouvoir fonctionner correctement. Le trafic généré par ces applications spécifiques, ainsi que le trafic généré par des applications traditionnelles, est classifié puis marqué dans l'une des trois classes de service définies. Les classes de service sont définies de façon à offrir des garanties minimales de QoS à chacun des types de trafic qui sera transporté par le réseau. La classification de trafic suivi est la suivant :

- Les applications qui travaillent en temps réel génèrent trafic de type stream (videostreamming, VoIP).
- Les applications qui ne travaillent pas en temps réel génèrent trafic de type élastique (Web browsing, e-mail, FTP).

Les classes de service sont définies par rapport à cette classification de trafic de la façon suivante :

- La Classe 1 pour le trafic de type stream.
- La Classe 2 pour le trafic de type élastique avec priorité.
- La Classe 3 (Best-Effort) pour le trafic de type élastique sans priorité (compatibilité avec ce qu'existe aujourd'hui).

Chapitre 4

Les adresses et les fichiers de configuration

4.1 L'adressage

Une ou plusieurs *adresses logiques* sont attribuées à chaque machine hôte. L'adresse INTERNET ou *adresse IP* est constituée d'une adresse de réseau et d'une adresse de machine sur ce réseau. Une adresse IPv4 occupe 4 octets ou 32 bits, une adresse IPv6 128 bits. Les adresses IPv4 sont en général données sous la forme n1.n2.n3.n4 (chacun des n représente la valeur d'un octet comprise entre 0 et 255).

4.1.1 Le principe d'adressage d'origine

Conceptuellement, chaque adresse IP est constituée d'une paire d'identificateur, le premier identifiant le réseau et le second identifiant une machine sur ce réseau. Dans le système d'adressage d'origine, à base de **classes**, les adresses IP devaient appartenir à une des classes A, B ou C (cf. tableau 4.1). Ceci a changé vers le milieu des années 90 avec l'arrivée de adresses sans classe (*classless*).

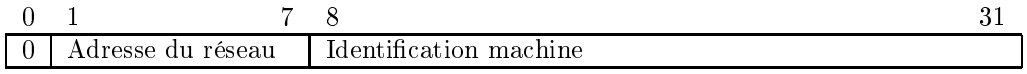
Classe	Intervalle
A	**0**.0.0.0 à **127**.255.255.255
B	**128**.0.0.0 à **191**.255.255.255
C	**192**.0.0.0 à **223**.255.255.255
D	**224**.0.0.0 à **239**.255.255.255
E	**240**.0.0.0 à **255**.255.255.255

Table 4.1 – Intervalles des adresses des différents classes de réseaux

Adresses de classe A

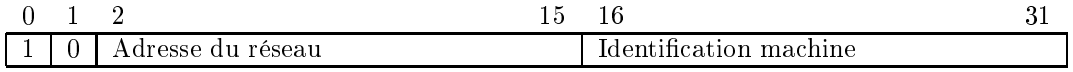
Elles correspondent aux grands réseaux (relativement peu nombreux et abritant un très grand nombre de machines).

L'adresse réseau occupe 1 octet, l'adresse de la machine sur ce réseau 3 octets.



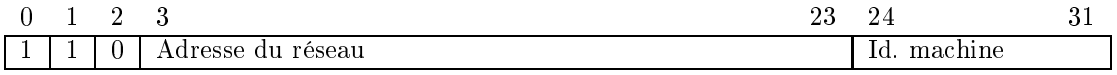
Adresses de classe B

Elles correspondent aux réseaux de taille moyenne.
L'adresse réseau occupe 2 octets, l'adresse de la machine sur ce réseau 2 octets.



Adresses de classe C

Elles correspondent aux petits réseaux (très nombreux et abritant un petit nombre de machines).
L'adresse réseau occupe 3 octets, l'adresse de la machine sur ce réseau 1 octet.



4.1.2 Adresses particulières

Par convention, l'identifiant 0 n'est attribué ni à aucun réseau ni à aucune machine. Ainsi, on peut utiliser une adresse IP ayant 0 comme identifiant de machine pour désigner le réseau lui-même (193.52.16.0 désigne le réseau où sont situées les machines Internet du département Informatique).

L'adresse dont tous les bits de l'identifiant de machine sont à 1 permet de désigner l'ensemble des machines de ce réseau, on l'appelle adresse de diffusion dirigée (*directed broadcast address*). Elle permet d'envoyer le même message à toutes les machines.

L'adresse de diffusion dirigée possède un identifiant de réseau, associée à un identifiant de machine, indiquant par sa valeur (tous les bits à 1) la diffusion. Une adresse de diffusion dirigée peut être interprétée sans ambiguïté parce qu'elle identifie le réseau cible sur lequel s'applique la diffusion mentionnée.

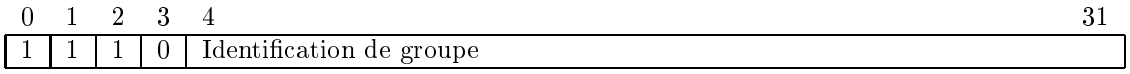
Une autre forme d'adresse de diffusion, appelée adresse de diffusion limitée ou adresse de diffusion locale, permet d'indiquer une diffusion locale, sur le réseau où se trouve la machine qui l'émet ou la reçoit, indépendamment de toute adresse IP.

Par convention l'adresse 127.0.0.1 est affectée à l'interface de bouclage (*loopback*). Tout ce qui est envoyé à cette adresse boucle et devient une réception. Cette adresse est normalement connue par le nom `INADDR_LOOPBACK`.

4.1.3 Adresses de multidistribution (classe D)

En plus de la remise à un destinataire unique (*unicast delivery*), dans laquelle un paquet atteint une unique machine, et de la remise en diffusion (*broadcast delivery*) où il s'agit de remettre un paquet à toutes les machines d'un certain réseau, il est possible d'avoir une remise multidestinataire (*multicasting*), dans laquelle un paquet est remis à un sous-ensemble spécifique de machines.

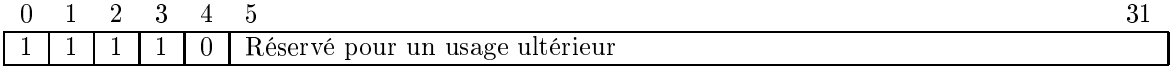
Ce sont les adresse de classe D qui sont utilisées dans l'adressage multipoint (*multicast*). Chaque groupe de diffusion possède une adresse de diffusion de groupe (classe D) unique. Une machine peut rejoindre un groupe à tout moment et appartenir à plusieurs groupes. Son appartenance à un groupe détermine le fait qu'elle reçoive l'information destinée à ce groupe. Une machine peut envoyer des informations à un groupe sans y appartenir. Les 4 premiers bits sont utilisés pour désigner la classe, les 28 bits suivants pour numéroter les groupes.



4.1.4 Adresses de classe E

Elles sont reservées pour un usage ultérieur.

Les 5 premiers bits sont utilisés pour désigner la classe, les 27 bits suivants pour l'instant inutilisés.



4.1.5 Multidomicile

Traditionnellement le terme de machine multidomiciliée (*multihomed*) fait référence à une machine avec plusieurs interfaces réseaux, comme deux Ethernet par exemple ou un Ethernet et un modem. Chaque interface a son adresse IPv4 unique.

Il est désormais possible d'attribuer plusieurs adresses IP à la même interface, ce qui lui donne ainsi plusieurs interfaces logiques. Aussi la définition d'une machine multidomiciliée doit être étendue à une machine à plusieurs interfaces, sans faire la distinction si ces interfaces sont physiques ou logiques.

4.2 Agrégation d'adresses et sous-adressage

4.2.1 Sous-réseaux

La structure standard d'une adresse IP peut être modifiée localement en utilisant des bits d'adresse de machine comme bits d'adresse de réseau supplémentaires.

Ces nouveaux bits de réseau permettent de définir un réseau au sein d'un réseau de plus grande taille. Ce nouveau réseau est appelé sous-réseau. Par exemple, dans un réseau de classe B où les machines sont identifiées sur 16 bits, utiliser 3 bits pour identifier un sous-réseau et les 13 autres bits pour identifier une machine, permet d'identifier 8 (2^3) sous-réseaux de 8190 (2^{13}) machines.

L'application d'un masque de bits, appelé masque de sous-réseau, à l'adresse IP permet de définir un sous-réseau. Les bits du maque de sous-réseau sont à 1 si les bits correspondants font partie de l'identifiant du réseau. Les bits sont à 0 si ils font partie de l'identifiant de la machine.

Le masque de réseau associé aux adresses de classe B correspond à la valeur 255.255.0.0. En faisant un ET entre une adresse IP et ce masque, on obtient l'adresse du réseau. Par convention, on fait suivre l'adresse d'un réseau où le sous-adressage est utilisé par un / et du nombre de bits du masque de sous-réseau.

Le masque de sous-réseau le plus souvent utilisé par les réseaux de classe B est 255.255.255.0. Il permet d'agrandir la partie réseau d'une adresse de classe B d'un octet supplémentaire.

Les deux premiers octets définissent le réseau de classe B, le troisième l'adresse du sous-réseau, le quatrième la machine reliée au sous-réseau.

Par exemple, le département d'Informatique utilise un réseau de classe B pour la pédagogie (ce réseau n'est pas relié à l'Internet). Il est utilisé avec des sous-réseaux et se note 172.18.0.0/24.

4.2.2 Adressage agrégé ou sur-adressage

En 1993, il devint évident que la croissance d'Internet allait conduire à l'épuisement des adresses de classe B. En attendant l'arrivée d'une nouvelle version d'IP, il fallait trouver une solution.

Appelé adressage hors classe (*classless addressing*), adressage agrégé ou sur-adressage (*supernet addressing*), voire adressage de sur-réseau, cette technique est à l'opposé de celle des sous-réseaux. Au lieu d'utiliser une seule adresse IP pour plusieurs réseaux appartenant à une certaine organisation, le sur-adressage consiste à disposer d'un grand nombre d'adresses réseau IP pour une unique organisation (en général des adresses de classe C).

L'allocation d'un grand nombre d'adresses de classe C au lieu d'une adresse de classe B permet d'économiser des numéros de classe B, mais il apparaît un autre problème : au lieu d'avoir une seule entrée par entreprise, une table de routage en comporte alors un grand nombre.

La technique dite CIDR (*Classless Inter-Domain Routing* ou routage inter-domaine sans classe) permet de résoudre ce problème. De façon conceptuelle, CIDR fusionne ou agrège un ensemble contigü d'adresse de classe C en une seule entrée représentée par un couple (`adresse_de_réseau`, `compteur`) dans lequel `adresse_de_réseau` est la plus petite adresse de réseau du bloc et `compteur` donne le nombre total de réseau du bloc. Par exemple, (`193.52.16.0, 3`) permet de spécifier les trois adresses réseau `193.52.16.0, 193.52.17.0, 193.52.18.0`.

En pratique, CIDR ne restreint pas les numéros de réseau aux adresses de classe C et n'a pas besoin de spécifier la taille du bloc comme un nombre entier. CIDR exige que chaque bloc d'adresse soit une puissance de deux et utilise un maque binaire pour identifier la taille du bloc.

Etand donné qu'identifier un bloc CIDR nécessite de définir deux items, une adresse et un masque, une notation abrégée a été définie (appelé notation slash). Cette notation indique la longueur du masque en décimal et utilise le signe slash (/) pour séparer cette valeur de l'adresse.

4.3 Routage

4.3.1 Tables de routage

Chaque machine du réseau (hôte ou passerelle) doit déterminer sa voie d'acheminement des datagrammes :

- si l'hôte de destination se trouve sur le même réseau local, les données sont transmises à l'hôte de destination,
- si l'hôte de destination se trouve sur un réseau distant, les données sont expédiées à une passerelle locale.

L'acheminement est donc déterminé en fonction de la partie réseau de l'adresse, selon les indications de la table de routage locale. Celle-ci est créée soit par l'administrateur-système, soit au moyen des protocoles de routage.

4.3.2 L'interface IP / Ethernet

Un réseau Ethernet identifie ses machines au moyen d'une adresse physique unique, appelée *adresse Ethernet.*

Le protocole spécifique ARP *Address Resolution Protocol* permet de retrouver l'adresse Ethernet correspondant à une adresse IP.

Le protocole RARP *reverse ARP* permet à une station sans disque ou un terminal X, ne connaissant pas sa propre adresse IP, de la demander par la diffusion d'un message Ethernet d'un type spécifié.

4.4 Les fichiers de configuration

4.4.1 Le fichier `/etc/hosts`

Il contient les informations relatives aux différentes machines du réseau local auquel appartient le système. Exemple :

```
193.52.16.26    kelenn-gw kelenn-gw.univ-brest.fr
```

Chaque ligne donne pour chaque site :

- l'adresse internet (`193.52.16.26`),
- le nom officiel (`kelenn-gw`),
- une liste d'alias (`kelenn-gw.univ-brest.fr`),
- un commentaire éventuel (apres un `#`).

4.4.2 Le fichier `/etc/networks`

Il constitue la base de données des réseaux connus. On y trouve une suite de lignes contenant :

- le nom officiel du réseau,
- son adresse internet,
- une liste d'alias,
- un commentaire éventuel (apres un `#`).

Du fait de la généralisation de l'adressage par nom, ce fichier n'est pratiquement plus utilisé et maintenu.

4.4.3 Le fichier `/etc/services`

Il donne la liste des services Internet connus. Exemple :

```
ftp             21/tcp
telnet          23/tcp
```

On y trouve une suite de lignes contenant :
- le nom du service (`ftp`),
- un numéro de port et un protocole (`21/tcp`)
- une liste d'alias,
- un commentaire éventuel (apres un `#`).

4.4.4 Le fichier `/etc/protocols`

Il donne la liste des protocoles utilisés dans le réseau Internet. Exemple :

```
ip      0       IP      # internet protocol, pseudo protocol number
```

On y trouve une suite de lignes contenant :
- le nom officiel du protoocole (`ip`),
- son numéro (`0`),
- une liste d'alias,
- un commentaire éventuel (apres un `#`).

4.4.5 Cas des NIS

Lors de la gestion à l'aide des NIS, ces fichiers ne sont pas à jour. Les renseignements sont centralisés sur la machine-maître des NIS et s'obtiennent grâce à la commande `ypcat nom_de_fichier`. Exemple :

```
$ ypcat hosts
172.16.1.18 cisco6 cisco6.univ-brest.fr
193.52.16.18    cisco5 cisco5.univ-brest.fr
```

4.5 Les processus démons

4.5.1 Principes

L'invocation de services standard tels que FTP ou `rlogin` est réalisée par l'intermédiaire de commandes (`ftp, rlogin`).
Ces services sont invoqués sur une machine distante, il est donc nécessaires que des processus dédiés, appelés *démons*[1] s'éxécutent sur la machine distante.

1. en anglais, daemon

4.5.2 Le super-démon `inetd`

On peut imaginer de lancer tous les processus-démons nécessaires au lancement du système, mais cela entrainerait l'existence permanente d'un nombre important de processus à l'état endormi.

On a préféré charger un processus serveur unique (le serveur `inetd`) de recevoir les demandes de services et d'activer le procesus démon approprié à la demande.

Un exemplaire unique de ce super-démon est lancé, qui scrute les différents ports des services qu'il supervise. Lorsqu'une requête arrive sur l'un des points de communication associés aux ports, le super-démon crée un nouveau processus correspondant au service demandé (il arrive qu'il traite lui-même la requête).

4.5.3 Le fichier `/etc/inetd.conf`

Le contenu de ce fichier est lu au lancement du démon ou lorsque le démon reçoit le signal `SIGHUP`. Exemple :

```
ftp stream  tcp     nowait  root    /usr/etc/in.ftpd        in.ftpd
name dgram   udp     wait    root    /usr/etc/in.tnamed      in.tnamed
systat stream  tcp     nowait  root    /usr/bin/ps             ps -auwwx
echo stream  tcp     nowait  root    internal
```

On y trouve une suite de lignes contenant :
- le nom du service (`ftp`),
- le type de point de communication utilisé (`stream` ou `dgram` qui dans le domaine Internet utilisent respectivement TCP et UDP),
- le protocole utilisé (`tcp` ou `udp`),
- une option `wait` ou `nowait` pour les services en mode `dgram` (l'option `wait` stipule qu'un seul serveur de ce type peut exister),
- un nom d'utilisateur qui sera le propriétaire du processus démon créé (`root`),
- la référence absolue du programme exécuté par le démon spécifique (par exemple `/usr/etc/in.ftpd/`) ; si la valeur est `internal` le service est réalisé par `inetd`,
- une liste de paramètres du programme à exécuter (par exemple des options).

4.6 Les fichiers d'équivalence UNIX

4.6.1 Le fichier `/etc/hosts.equiv`

Ce fichier définit les hôtes (machines) et utilisateurs fiables auxquels le droit d'accès aux commandes en `r` (cf. §5.7) est accordé. Ce fichier peut aussi définir les hôtes et utilisateurs auxquels le droit d'accès explicite n'est pas octroyé. Ne pas disposer des droits d'accès ne signifie pas nécessairement que l'utilisateur ne puisse pas accéder aux commandes ; cela signifie simplement que l'utilisateur doit spécifier un mot de passe avant de pouvoir y accéder.

Différents systèmes UNIX appartenant à un réseau local peuvent être déclarés donc équivalents du point de vue des `r`-commandes des utilisateurs, ce qui permet à un utilisateur connecté à un hôte d'accéder aux comptes utilisateurs de même nom sur un autre système qui aura précisé le nom de l'hôte dans son fichier `/etc/hosts.equiv`.

4.6.2 Le fichier `/.rhosts`

Dans le cas où des machines n'ont pas été déclarés équivalentes par les administrateurs systèmes, les utilisateurs ont la possibilité de définir personnellement des accès aux `r`-commandes sur différentes machines par l'intermédiaire du fichier `.rhosts` de leur *home directory*.

Si le fichier `.rhosts` de l'utilisateur `arthur` de la machine `ma_machine` contient les lignes :

```
autre_machine toto
une_autre_machine toto
```

l'utilisateur `toto` sur les machines `autre_machine` ou `une_autre_machine` pourra s'identifier sur la machine `ma_machine` sous le nom `arthur` sans avoir à donner le mot de passe de cet utilisateur.

N.B. : c'est un "trou de sécurité important", à utiliser avec précaution.

Chapitre 5

Les commandes d'administration

Toutes ces commandes sont à exécuter avec `man` sous la main ...

5.1 La commande `ping`

Elle permet de tester si une machine donnée est active. Exemple :

```
$ ping kelenn-gw
kelenn-gw.univ-brest.fr is alive
```

5.2 La commande `arp`

Elle permet de visualiser le contenu de la table du protocole ARP. Exemple :

```
$ arp -a
kelenn-gw.univ-brest.fr (193.52.16.26) at 8:0:20:74:f9:df
odet (172.16.2.2) at 8:0:20:20:83:31
```

Lorqu'aucune résolution d'adresse n'a été faite dernièrement, il n'y a pas d'entrée. En utilisant La commande `ping`, on force la conversion d'adresse et on peut ensuite visualiser l'adresse demandée.

```
$ arp terre
terre (193.52.16.59) -- no entry
$  ping terre
terre is alive
$ arp terre
terre (193.52.16.59) at 8:0:20:2:c8:38
```

5.3 La commande `ifconfig`

Elle permet de spécifier (pour les administrateurs) ou d'obtenir des informations sur les interfaces du système.

```
$ /sbin/ifconfig -a
lo0: flags=849<UP,LOOPBACK,RUNNING,MULTICAST> mtu 8232
        inet 127.0.0.1 netmask ff000000
le0: flags=863<UP,BROADCAST,NOTRAILERS,RUNNING,MULTICAST> mtu 1500
        inet 193.52.16.37 netmask ffffffc0 broadcast 193.52.16.0
le1: flags=863<UP,BROADCAST,NOTRAILERS,RUNNING,MULTICAST> mtu 1500
        inet 172.18.3.1 netmask ffffff00 broadcast 172.18.3.0
```

Cette machine est une passerelle qui a 3 interfaces :
- `lo0` interface de bouclage (loopback) pour les paquets à destination de la même machine,
- `le0` interface Ethernet associée à l'adresse IP `193.52.16.37`,
- `le0` interface Ethernet associée à l'adresse IP `172.18.3.1`.

Chacune de ces interfaces peut être utilisée (`UP`), autorise la diffusion sur le réseau (ou le sous-réseau) (`BROADCAST`), n'a pas d'en-queue (`NOTRAILERS`), est en fonctionnement (`RUNNING`), autorise le multi-adressage (`MULTICAST`).

Le Maximum Transmit Unit (MTU) est de 8232 sur l'interface de bouclage, de 1500 octets sur les interfaces Ethernet.

Le masque de sous-réseau associé à `le0` est `ffffffc0` (en binaire 11111111111111111111111111000000, 26 1 suivis de 6 0). Le réseau (de classe C) auquel est connecté `le0` utilise le sous-adressage (/26) : 2 bits sont utilisés pour l'identification des sous-réseaux, 6 bits pour l'identification des machines dans le sous-réseau (ce qui permet de définir 4 (2^2) sous-réseaux de 64 (2^6) machines).

Le masque de sous-réseau associé à `le1` est `ffffff00` (en binaire 11111111111111111111111100000000, 24 1 suivis de 8 0). Le réseau (de classe B) auquel est connecté `le1` utilise le sous-adressage (/24) : 8 bits sont utilisés pour l'identification des sous-réseaux, 8 bits pour l'identification des machines dans le sous-réseau (ce qui permet de définir 256 (2^8) sous-réseaux de 256 (2^8) machines).

5.4 La commande `netstat`

Elle permet d'obtenir des informations sur l'activité réseau du système. Plus précisément, elle fournit des informations sur les interfaces, les tables de routage et les "points de communication" ou *socket* du système (une socket est une prise de communication reliée à un port et un protocole TCP ou UDP).

Avec l'option `-i`, on obtient des statistiques sur les interfaces du système local (nom de l'interface, taille maximum des paquets transmis, réseau connecté, nom logique, paquets émis et reçus, erreurs et collisions) :

```
$ netstat -i
Name  Mtu  Net/Dest      Address        Ipkts  Ierrs Opkts  Oerrs Collis Queue
le0   1500 193.52.16.0   penfeld-gw     1293594 11   983956  14   152016 0
le1   1500 172.16.2.0    penfeld        898285  0    1181685 28   1729   0
lo0   1536 loopback      localhost      16593   0    16593   0    0      0
```

Le champ `Name` contient le nom attribué à l'interface avec `ifconfig`. Une astérisque (*) indique que l'interface n'a pas été activée.

Le champ `MTU` contient le Maximum Transmit Unit.

Le champ `Net/Dest` indique le réseau, le sous-réseau (auquel cas il faut appliquer le masque) ou l'hôte auquel l'interface permet d'accéder.

Le champ `Address` contient l'adresse IP attribuée à cette interface.

Le champ `Ipkts` indique le nombre de paquets que cette interface a reçu.

Le champ `Ierrs` indique le nombre de paquets endommagés que cette interface a reçu.

Le champ `Opkts` indique le nombre de paquets que cette interface a envoyé.

Le champ `Oerrs` indique le nombre de paquets générant une erreur que cette interface a envoyé.

Le champ `Collis` indique le nombre de collisions Ethernet détectées par cette interface.

Le champ `Queue` indique le nombre de paquets figurant dans la file d'attente et attendant leur transmission par cette interface (normalement à 0).

L'option `-r` donne les informations sur les tables de routage :

```
$ netstat -r
Routing tables
Destination          Gateway              Flags    Refcnt Use        Interface
venan                paludenn-gw          UGHD     0      238        le0
default              cisco5               UG       10     67909      le0
193.52.16.0          penfeld-gw           U        36     24917      le0
172.16.2.0           penfeld              U        3      167449     le1
```

L'option `-s` donne des statistiques par protocole ou sur le routage (avec `-r`).

Enfin, avec l'option `-a`, on visualise toutes les sockets actives (on peut restreindre au domaine Unix `-f unix` ou au domaine internet `-f inet`).

5.5 La commande `nslookup`

Elle permet de se connecter au serveur de noms utilisé par défaut ou à un serveur de nom particulier et de l'interroger pour obtenir des informations sur les domaines (liste des machines) et sur les machines (adresses).

Elle s'utilise en mode interactif ou en mode non interactif.

```
$ nslookup
Default Server:  cassis-gw.univ-brest.fr
Address:  192.70.100.29

> kelenn
Server:  cassis-gw.univ-brest.fr
Address:  192.70.100.29

Name:    kelenn.univ-brest.fr
Address:  172.16.1.24
```

```
> kelenn-gw
Server:  cassis-gw.univ-brest.fr
Address:  192.70.100.29

Name:    kelenn-gw.univ-brest.fr
Address:  193.52.16.26

> whitehouse.gov
Server:  cassis-gw.univ-brest.fr
Address:  192.70.100.29

Non-authoritative answer:
Name:    whitehouse.gov
Address:  198.137.241.30

> exit
```

La commande `help` donne la liste des actions permises :

```
Commands:       (identifiers are shown in uppercase, [] means optional)

NAME            - print info about the host/domain NAME using default server
NAME1 NAME2     - as above, but use NAME2 as server
help or ?       - print help information
exit            - exit the program
set OPTION      - set an option
    all         - print options, current server and host
    [no]debug   - print debugging information
    [no]d2      - print exhaustive debugging information
    [no]defname - append domain name to each query
    [no]recurse - ask for recursive answer to query
    [no]vc      - always use a virtual circuit
    domain=NAME - set default domain name to NAME
    root=NAME   - set root server to NAME
    retry=X     - set number of retries to X
    timeout=X   - set time-out interval to X
    querytype=X - set query type to one of A,CNAME,HINFO,MB,MG,MINFO,MR,MX
    type=X      - set query type to one of A,CNAME,HINFO,MB,MG,MINFO,MR,MX
server NAME     - set default server to NAME, using current default server
lserver NAME    - set default server to NAME, using initial server
finger [NAME]   - finger the optional NAME
root            - set current default server to the root
ls NAME [> FILE]- list the domain NAME, with output optionally going to FILE
view FILE       - sort an 'ls' output file and view it with more
```

5.6 La commande finger

Elle permet de donner des informations sur les utilisateurs d'une machine donné ou du domaine NIS courant.
En particulier, cette commande liste les informations contenus dans les fichiers /etc/passwd ./plan ./project, indique le dernier login, quand le courrier a été lu pour la dernière fois ...
Exemple :

```
$ finger gire
Login name: gire                        In real life: Sophie Gire
Directory: /home/ens/gire               Shell: /bin/csh
Last login Wed Jul 30 12:43 on console
New mail received Mon Oct 13 14:35:44 1997;
  unread since Mon Aug 18 18:21:11 1997
Plan:
  Etre toute ma vie en "mise en disponibilite pour convenances personnelles"
  et tout le temps tout le temps tout le temps tout le temps
     faire des patates a l'ail a John Cassavetes
```

Le démon fingerd doit être actif sur la machine.

```
$ finger president@whitehouse.gov
[whitehouse.gov]

        Finger service for arbitrary addresses on whitehouse.gov is not
supported. If you wish to send electronic mail, valid addresses are
"PRESIDENT@WHITEHOUSE.GOV", and "VICE-PRESIDENT@WHITEHOUSE.GOV".
```

5.7 Les commandes en r

Les commandes en r (pour *remote-command*) permettent d'exécuter des commandes à distance sur d'autres machines. Pour des raisons de sécurité, certains sites désactivent l'utilisation des commandes en r.

5.7.1 La commande rlogin

Elle permet de se connecter sur les machines du réseau local (identique à telnet).

```
uneMachine $ rlogin autreMachine
Password:
autreMachine $
```

5.7.2 La commande rcp

Elle permet la copie de fichiers à partir et à destination des autres machines du réseau local.

```
uneMachine $ rcp autreMachine: fichierDistant fichierLocal
```

5.7.3 La commande `rsh`

Elle permet de transmettre une commande à une autre machine du réseau local, en vue de son exécution. La sortie standard et les erreurs standard sont renvoyées à la machine locale.

```
uneMachine $ rsh autreMachine ls -l /tmp
total 4
drwx------    2 ribaud   users        4096 Feb 26 07:06 xdvijjHQZh
```

5.7.4 La commande `ruser`

Elle liste les utilisateurs connectés sur les machines du réseau local.

```
$  rusers
kelenn3      root
kelenn-gw.un ballot ballot ballot ballot
```

5.8 La commande `uname`

Elle permet d'obtenir des informations sur le système local.

```
$ uname -a
SunOS penfeld-g 4.1.3_U1 2 sun4m
```

5.9 Les commandes `hostname` et `hostid`

Elles permettent d'obtenir le nom de la machine et un identificateur unique attribué par le constructeur.

```
$ hostname ; hostid
penfeld-gw
8074f8a8
```

Chapitre 6

Les structures et les fonctions d'accès

6.1 Les fichiers d'include

La plupart des définitions de structures et déclarations de fonctions de ce chapitre sont contenues dans les fichiers d'include :

```
#include<sys/types.h>           /* types de base */
#include<sys/socket.h>          /* definitions de base des sockets */
#include<netdb.h>               /* fonctions d'information sur le reseau */
#include<netinet/in.h>          /* sockaddr_in and co */
#include <arpa/inet.h>          /* fonctions de manipulation d'adresse */
```

6.2 Les librairies standard

La compilation dans les environnements Linux ne nécessite pas d'utilisation explicite de librairies. Par contre sous solaris, les différentes fonctions se trouve dans les librairies `nsl, socket` et `resolv`.

6.3 Les types Posix

Le type de données `in_addr_t` doit être un entier sans signe d'au moins 32 bits, le type de données `in_port_t` doit être un entier sans signe d'au moins 16 bits et type de données `sa_family_t` peut être n'importe quel entier sans signe.

La table 6.1, inspirée de [Ste98], liste ces trois types ainsi que d'autres types Posix.1g qui seront utilisés.

6.4 Les adresses des machines

En IPv4, l'adresse Internet d'une machine est un entier long (4 octets).

Historiquement la structure suivante était utilisée :

```
#include<netinet/in.h>
struct in_addr {
               u_long s_addr; };
```

Type de données	Description	Fichier d'en-tête
int8_t	Entier 8-bit signé	<sys/types.h>
uint8_t	Entier 8-bit sans signe	<sys/types.h>
int16_t	Entier 16-bit signé	<sys/types.h>
uint16_t	Entier 16-bit sans signe	<sys/types.h>
int32_t	Entier 32-bit signé	<sys/types.h>
uint32_t	Entier 32-bit sans signe	<sys/types.h>
sa_family_t	Famille d'adresse	<sys/socket.h>
socklen_t	Entier 32-bit sans signe	<sys/socket.h>
in_addr_t	Adresse IPv4, normalement un uint32_t	<netinet/in.h>
in_port_t	Port TCP ou UDP, normalement un uint16_t	<netinet/in.h>

Table 6.1 – Types de données Posix

En Posix, l'adresse internet est :

```
#include<netinet/in.h>
struct in_addr {
                uint32_t s_addr; };
```

Sur Sun, on trouve la définition suivante, compatible avec la première, et permettant d'accéder à différentes interprétations :

```
#include<netinet/in.h>
struct in_addr {
        union {
                struct { uint8_t s_b1,s_b2,s_b3,s_b4; } _S_un_b;
                struct { uint16_t s_w1,s_w2; } _S_un_w;
                uint32_t _S_addr;
        } _S_un;
#define s_addr  _S_un._S_addr             /* should be used for all code */
};
```

Sur certains Linux (qui devraient pourtant être Posix ...), on trouve la définition suivante :

```
#include<netinet/in.h>
struct in_addr {
               unsigned int s_addr; };
```

6.5 Les fonctions de manipulation des adresses

6.5.1 inet_aton

```
#include <arpa/inet.h>
int inet_aton(char *cp, struct in_addr *inp);
```

Cette fonction convertit la chaîne de caractères `cp` (représentant une adresse en notation pointée) en une adresse sous la forme d'un entier 32-bit pointée par `inp`. La fonction renvoie 1 si l'appel a réussi, 0 sinon.

6.5.2 inet_ntoa

```
#include <arpa/inet.h>
char * inet_ntoa(struct in_addr in);
```

Cette fonction convertit une adresse sous la forme d'un entier 32-bit nommée `in` en une adresse en notation pointée. La fonction renvoie l'adresse de la chaîne si l'appel a réussi, NULL sinon.

6.6 La représentation des nombres

La représentation des nombres sur des machines hétérogènes peut poser des problèmes d'interprétation (Little-Endian ou Big-Endian). Aussi, une représentation standard est adoptée : celle dite *Big Endian* où les octets de poids fort sont les plus à gauche.

Des fonctions de conversion (en général macro-définies) sont fournies dans le fichier `<netinet/in.h>`.

`htnol` et `ntohl` permettent la manipulation des adresses, et `htnos` et `ntohs` permettent celles des numéros de port :

```
u_short ntohs(u_short); /* network to host short */
u_short htons(u_short); /* host to network short */
u_long  ntohl(u_long); /* network to host long */
u_long  htonl(u_long); /* host to network long */
```

6.7 La structure `hostent`

Elle correspond à la liste des informations relatives à une machine au retour d'un appel à l'une des fonctions `gethostname`, `gethostbyname`, `gethostbyaddr`, `gethostent` :

```
#include<sys/socket.h>
#include<netdb.h>
struct  hostent {
  char    *h_name;        /* official name of host */
  char    **h_aliases;    /* pointer to array of pointers to alias names */
  int     h_addrtype;     /* host address type : AF_INET or AF_INET6 */
  int     h_length;       /* length of address : 4 or 16 */
  char    **h_addr_list;  /* ptr to array of ptrs with IPv4 or IPv6 addrs */
#define h_addr  h_addr_list[0]  /* first address in list */
};
```

6.8 Les fonctions de consultation

6.8.1 gethostname

On a souvent besoin du nom de la machine locale.

```
#include <unistd.h>
int gethostname(char *name, size_t len);
```

Cette fonction permet de récupérer à l'adresse `name` le nom de la machine locale (terminé par un `\0`). `len` contient la taille de la zone pointée par `name`. La fonction renvoie 0 si l'appel a réussi, -1 sinon.

6.8.2 `gethostbyname`

Il est possible d'obtenir une structure `hostent` associée à une machine de nom donné en paramètre : si un serveur de nom est actif, il est interrogé ; sinon les NIS et en dernier le fichier `/etc/hosts`.

```
#include<sys/types.h>
#include<sys/socket.h>
#include<netdb.h>
struct hostent  *gethostbyname(const char *name);
```

Le pointeur renvoyé pointe en zone statique et chaque appel écrase le résultat du précédent : il faut donc recopier les résultats avec `memcpy` par exemple.

6.8.3 `gethostbyaddr`

Lorsqu'on connait l'adresse Internet, on peut aussi obtenir une structure `hostent`.

```
#include<sys/types.h>
#include<sys/socket.h>
#include<netdb.h>
struct hostent  *gethostbyaddr(const char *addr, int len, int type);
```

Dans le cas d'une adresse Internet v4, `addr` pointe sur un entier long, et donc `len` est égal à `sizeof(long)` et `type` est égal à `AF_INET`.

Chapitre 7

Les sockets

7.1 Introduction

7.1.1 Qu'est-ce qu'une socket ?

Une *socket* est un point de communication par lequel un processus peut émettre ou recevoir des informations.

A l'intérieur d'un processus, une socket sera identifiée par un descripteur de même nature que ceux identifiant les fichiers (c'est-à-dire dans le même ensemble). Cette propriété est essentielle car elle permet la redirection des fichiers d'entrée-sortie standard (descripteurs 0, 1 et 2) sur des sockets et donc l'utilisation d'applications standards sur le réseau.

Cela signifie également que tout nouveau processus (créé par `fork`) hérite des descripteurs de son père.

A toute socket est associée dans le système un objet ayant la structure `socket`, définie dans le fichier standard `<sys/socketvar.h>` qui contient l'ensemble des caractéristiques de cet objet.

7.1.2 Type d'une socket

Le *type* d'une socket détermine la sémantique des communications qu'elle permet de réaliser.

L'ensemble des propriétés d'une communication dépend essentiellement du protocole de transport utilisé.
Dans le monde Internet, on utilise essentiellement TCP et UDP, bien qu'on puisse utiliser IP et que l'interface des sockets aie été étendue pour l'utilisation de protocoles de transport conformes au modèle OSI.

Une communication nécessitant l'échange de messages complets et structurés emploie une socket de *type* `SOCK_DGRAM` (datagramme), le protocole sous-jacent est UDP.

Une communication nécessitant l'échange de flots d'information emploie une socket de *type* `SOCK_STREAM` (stream ou flot), le protocole sous-jacent est TCP.

Du fait de la nature des protocoles TCP et UDP, une socket de *type* `SOCK_DGRAM` s'utilise en mode non-connecté et sans garantie de fiabilité
et une socket de *type* `SOCK_STREAM` s'utilise en mode connecté et avec garantie du maximum de fiabilité.

Les constantes symboliques `SOCK_DGRAM` et `SOCK_STREAM` sont définies dans le fichier standard `<sys/socket.h>`.

7.1.3 Domaine d'une socket

Le *domaine* d'une socket détermine l'ensemble des autres points de communication qu'elle permet d'atteindre (et donc les processus qui les utilisent).

Il existe plusieurs domaines dont les deux principaux dans le monde Unix sont :

- `AF_UNIX` : le domaine *local* qui permet d'atteindre les processus s'exécutant sur la même machine,
- `AF_INET` : le domaine de l'*internet* qui permet d'atteindre les processus s'exécutant sur une des machines de l'internet.

Dans le domaine `AF_UNIX`, une socket est désignée de la même manière que les fichiers et l'adresse d'une telle socket est définie dans le fichier standard `<sys/un.h>` comme correspondant à la structure suivante :

```
#include <sys/un.h>
struct  sockaddr_un {
        short   sun_family;       /* AF_UNIX */
        char    sun_path[108];    /* reference du "fichier" */
};
```

Dans le domaine `AF_INET`, une socket appartient à une machine et est identifiée par un *port*, l'adresse d'une telle socket est définie dans le fichier standard `<netinet/in.h>` comme correspondant à la structure suivante :

```
#include <netinet/in.h>
struct sockaddr_in {
        short   sin_family;       /* AF_INET */
        u_short sin_port;         /* numero du port associe */
        struct  in_addr sin_addr; /* adresse internet de la machine */
        char    sin_zero[8];      /* tableau de 8 caracteres nuls */
};
```

En Posix, on a la définition suivante :

```
#include <netinet/in.h>
struct sockaddr_in {
        uint8_t         sin_len;     /* longueur de la structure (16) */
        sa_family_t     sin_family;  /* AF_INET */
        in_port_t       sin_port;    /* numero du port associe */
        struct  in_addr sin_addr;    /* adresse internet de la machine */
        char            sin_zero[8]; /* tableau de 8 caracteres nuls */
};
```

7.2 Primitives communes

7.2.1 La création d'une socket : `socket`

Tout processus souhaitant communiquer doit demander à son système local la création d'une socket en spécifiant le type, le domaine et le protocole.

Pour un domaine et un type, il n'existe souvent qu'un seul type de protocole utilisable, dans ce cas on choisit 0 pour le protocole et le système choisit le protocole adapté.

```
#include <sys/types.h>
#include <sys/socket.h>
int socket(
  int domain,    /* AF_UNIX, AF_INET, ... */
  int type,      /* SOCK_DGRAM, SOCK_STREAM, ... */
  int protocol   /* 0 protocle par defaut */
  );
```

La valeur de retour est un descripteur sur la socket nouvellement créée si cette création est possible et -1 en cas d'erreur. En cas d'erreur, `errno` vaut :

- `EACCES` : incompatibilité type / protocole,
- `EMFILE` : table des descripteurs pleine,
- `EPROTONOSUPPORT` : protocole non-supporté,
- ... (peut dépendre du système utilisé).

7.2.2 La suppression d'une socket : `close`

```
#include <unistd.h>
int close(int fd);
```

Une socket est supprimée lors de la fermeture du **dernier descripteur** permettant d'y accéder (appel-système `close`).

Cette suppression libère toutes les ressources allouées.

Cette primitive peut être bloquante dans le cas d'utilisation d'une socket de type `SOCK_STREAM` dans le domaine `AF_INET` au cas où le tampon d'émission n'est pas vide.

7.2.3 L'attachement d'une socket à une adresse : `bind`

Une différence essentielle avec les fichiers est que le *nommage* d'une socket est une opération différente de sa création : ouverture et création vont de pair comme les *tubes*, mais il est ensuite possible et souvent nécessaire d'attacher une adresse de son domaine à l'objet créé au moyen de la primitive `bind`.

Sans nommage explicite de la socket, on se retrouve dans une situation analogue à celle de la création d'un *tube* sans nom :
seuls les processus ayant hérité du descripteur sur la socket peuvent lire ou écrire sur la socket (car elle est *duplex*), mais personne ne pourrait envoyer de l'extérieur, de données sur la socket (dans le monde réel, essayez de vous faire appeler dans une cabine téléphonique dont vous ne connaissez pas le numéro ...).

Un moyen de désigner la socket indépendamment de tout contexte (en dehors de son domaine) est fourni par `bind` :

```
#include <sys/types.h>
#include <sys/socket.h>
int bind(
  int sockfd,                     /* descripteur de la socket a attacher */
  const struct sockaddr *my_addr, /* pointeur sur l'adresse a utiliser */
```

```
  int addrlen                    /* longueur de l'adresse */
  );
```

La structure `sockaddr` est générique :

```
#include <sys/socket.h>
struct sockaddr {
        u_short sa_family;              /* address family */
        char    sa_data[14];            /* up to 14 bytes of direct address */
};
```

Il faut, en fonction du domaine, lui substituer la structure correspondante (voir 7.1.3).

N.B. Pour les appels `bind`, `accept`, `connect` et `sendto`, l'utilisation de la structure du domaine provoque un warning à la compilation[1], du genre "warning : passing arg 2 of 'bind' from incompatible pointer type". Pour éviter ce warning, il faudrait faire une conversion (*cast*) à l'appel.

La valeur de retour est 0 si cet attachement est possible et -1 en cas d'erreur. En cas d'erreur, `errno` vaut :

- `EACCES` : adresse protégée,
- `EADDRINUSE` : adresse déja utilisée,
- `EADDRNOTAVAIL` : adresse incorrecte,
- `EBADF` : `s` est un descripteur invalide,
- ... (peut dépendre du système utilisé).

Cas du domaine `AF_UNIX`

L'attachement d'une socket à une adresse du domaine `AF_UNIX` ne peut se faire que si la référence correspondante n'existe pas (elle peut d'ailleurs être utilisée par autre chose qu'une socket : un fichier régulier, un tube nommé, ...).

On visualise les sockets du domaine `AF_UNIX` avec la commande `netstat` :

```
% netstat -f unix
Active UNIX domain sockets
Address  Type   Recv-Q Send-Q Vnode     Conn Refs Nextref Addr
ff64668c stream      0      0     0 ff65f18c    0       0
ff656a8c stream      0      0     0 ff674e0c    0       0 /tmp/.X11-unix/X0
ff64688c dgram       0      0 ff13d6b8     0    0       0 /dev/log
```

Une socket de ce domaine apparait avec un `s` en début des permissions.

```
% ls -l /tmp/.X11-unix/X0
srwxrwxrwx  1 moi           0 Nov  3 08:07 /tmp/.X11-unix/X0
% ls -l /dev/log
srw-rw-rw-  1 root          0 Sep 19 09:57 /dev/log
```

La primitive `socketpair` permet de créer deux sockets *non nommées* et des les associer (utilisable pour les deux types de sockets `SOCK_DGRAM` et `SOCK_STREAM`).

1. C n'est pas un langage-objet, `struct sockaddr_in` et `struct sockaddr_un` devraient être des sous-types de `struct sockaddr` ...

```
#include <sys/types.h>
#include <sys/socket.h>
int socketpair(
  int  domain,   /* AF_UNIX */
  int  type,     /* SOCK_DGRAM, SOCK_STREAM */
  int  protocol, /* 0 */
  int sv[2]      /* tableau de 2 entiers qui contiendra les sockets */
  );
```

Les avantages par rapport à un tube nommé POSIX sont que la paire de sockets permet une communication duplex (alors qu'il faudrait deux tubes nommés et quatre descripteurs) et qu'on peut choisir son mode de communication `SOCK_DGRAM` ou `SOCK_STREAM` ce qui conduit à des sémantiques différentes.

Cas du domaine `AF_INET`

Rappelons le format des structures utilisées :

```
#include<netinet/in.h>
struct in_addr {
                u_long s_addr;
        };

struct sockaddr_in {
        short   sin_family;            /* AF_INET */
        u_short sin_port;              /* numero du port associe */
        struct  in_addr sin_addr;      /* adresse internet de la machine */
        char    sin_zero[8];           /* tableau de 8 caracteres nuls */
};
```

Les champs peuvent avoir, à l'appel de `bind`, des valeurs particulières :

- La valeur `INADDR_ANY` de `sin_addr.s_addr` permet d'associer la socket à toutes les adresses IP possibles de la machine (particulièrement important pour les machines "passerelles" : la socket pourra être indifféremment contactée sur n'importe quel des réseaux auquel appartient la passerelle),
  sinon il faut utiliser les primitives `gethostname` et `gethostbyname` pour connaître l'adresse de la machine locale.
- L'attachement à un numéro de port particulier est indispensable si ce numéro est public et connu (exemple d'un service dont les clients connaissent le numéro).
  Il existe des cas où le numéro de port peut être attribué par le système, en spécifiant 0 dans le champ `sin_port`. Ceci peut conduire le processus à ne pas connaître le port qu'il utilise, il peut le demander au système au moyen de la primitive `getsockname`.

```
#include <sys/types.h>
#include <sys/socket.h>
int getsockname(
  int s,                 /* descripteur de la socket */
  struct sockaddr *name, /* pointeur sur une structure adresse
                            a remplir par getsockname */
```

```
int *namelen            /* pointeur sur la taille de l'adresse */
);
```

A l'appel `*namelen` est la taille de la structure adresse reservée pour récuperer le résultat, au retour `*namelen` aura pour valeur la longueur effective de l'adresse.

Attention : les valeurs des champs `sin_addr.s_addr` et `sin_port` sont renvoyées en format réseau (voir 6.6).

N.B. Les noms `name` et `namelen` sont mal choisis, ce ne sont pas des noms de machines mais des adresses. Les noms corrects sont `my_addr` et `addrlen` (comme pour `bind`).

Une fonction de création et d'attachement de socket

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>

int creer_socket(int type, int *ptr_port, struct sockaddr_in *ptr_adresse)
{
struct sockaddr_in adresse;
int desc;
int longueur = sizeof(struct sockaddr_in);

/* cr'eation de la socket */
if ((desc = socket(AF_INET, type, 0)) == -1) {
        perror("Creation socket impossible\n");
        return -1;
        }

/* pr'eparation de l'adresse locale : port + toutes les @ IP */
adresse.sin_family=AF_INET;
adresse.sin_addr.s_addr=htonl(INADDR_ANY);
adresse.sin_port=htons(*ptr_port);

/* attachement de la socket a' l'adresse locale */
if (bind(desc, &adresse, longueur) == -1) {
        perror("Attachement de la socket impossible\n");
        close(desc);
        return -1;
        }

/* dans le cas ou' *port == 0, le syste'me a attribu'e un port */
/* on r'ecupe're ce num'ero de port avec getsockname */
if (ptr_adresse != NULL)
        getsockname(desc, ptr_adresse, &longueur);

/* mise sous forme hote du numero de port */
*ptr_port = ntohs(ptr_adresse->sin_port);

return desc;
}
```

7.3 La scrutation

7.3.1 Introduction

Dans un certain nombre de situations, un processus est amené à communiquer (de manière non régulière) avec différentes entités du système et utilise pour cela des objets de nature quelconque (fichiers, tubes, sockets, ...). Les opérations qu'il est amené à réaliser sont très souvent bloquantes et dans le cas d'un serveur, par exemple, il est essentiel que les requêtes soient prises en compte le plus rapidement possible après qu'elles aient été transmises. Le processus est donc amené à scruter ce qui se passe sur différents descripteurs *sans prendre le risque d'être bloqué*. Il existe deux possibilités :

- si les descripteurs ont étés positionnés en mode non-bloquant au moyen de la primitive `fcntl` ou `ioctl` :
  `int on=1; ... ; ioctl(desc, FIOSNBIO; &on); ... } /* ou FIONBIO */`
  on peut essayer de lire alternativement sur chacun des descripteurs (*polling*) ;
- sinon on peut demander à être averti lorsqu'une opération peut être réalisée sur l'un des descripteur au moyen de la primitive `select`, le processus appelant cette primitive se met alors en attente sur des événements relatifs aux lectures et écritures sur un ensemble de descripteurs.

7.3.2 La primitive `select`

Les ensembles de descripteurs

Le type `fd_set` prédéfini dans le fichier standard `<sys/types.h>` permet de définir des ensembles de descripteurs qui seront utilisés comme paramètres de la primitive `select`. Différentes macros définies dans ce fichier permettent la manipulation des `fd_set`.

Prototype de la fonction	*Effet*
`FD_ZERO(fd_set *ptr_set)`	`*ptr_set = {}`
`FD_CLR(int desc, fd_set *ptr_set)`	`*ptr_set = *ptr_set - {desc}`
`FD_SET(int desc, fd_set *ptr_set)`	`*ptr_set = *ptr_set + {desc}`
`FD_ISSET(int desc, fd_set *ptr_set)`	valeur non nulle : desc appartient à `*ptr_set`

La primitive `select`

Cette primitive permet de scruter plusieurs descripteurs (fichiers, tubes, sockets, ...) , en se mettant en attente des événements relatifs aux lectures et écritures.

Trois types d'événements sont susceptibles d'être examinés par la primitive `select` :

- la possibilité de lire sur un descripteur donné,
- la possibilité d'écrire sur un descripteur donné,
- l'existence d'une condition exceptionnelle sur une ressource d'un descripteur pour lequel ce concept existe (comme par exemple l'arrivée d'un message urgent sur une socket du domaine Internet et de type `SOCK_STREAM`).

```
int select(
int width,       /* ensemble de descripteurs a examiner 0, 1, ..., width - 1 */
fd_set *readfds,         /* ensemble de descripteurs en lecture */
fd_set *writefds,        /* ensemble de descripteurs en ecriture */
fd_set *execptfds,       /* ensemble de descripteurs en exception */
struct timeval *timeout  /* temporisation */
);
```

Interprétation des paramètres en entrée

`readfds` pointe en entrée sur un ensemble de descripteurs sur lequel on souhaite pouvoir réaliser une lecture.
Une valeur `NULL` de ce paramètre correspond à l'ensemble vide (rien à surveiller en lecture).
L'existence dans l'ensemble correspondant d'un descripteur sur un fichier régulier non verrouillé en mode non exclusif de manière non impérative provoque un retour immédiat de la primitive `select` (en effet, la lecture sur un tel fichier n'est pas bloquante).

`writefds` pointe en entrée sur un ensemble de descripteurs sur lequel on souhaite pouvoir réaliser une écriture.
Une valeur `NULL` de ce paramètre correspond à l'ensemble vide (rien à surveiller en écriture).
L'existence dans l'ensemble correspondant d'un descripteur sur un fichier régulier non verrouillé impérativement (que ce soit en mode partagé ou exclusif) provoque un retour immédiat de la primitive `select` (en effet, l'écriture sur un tel fichier n'est pas bloquante).

`execptfds` pointe en entrée sur un ensemble de descripteurs sur lequel on souhaite la réalisation d'un test de condition exceptionnelle.
Une valeur `NULL` de ce paramètre correspond à l'ensemble vide (rien à surveiller).

`width` doit au moins être égal à la valeur du plus grand descripteur appartenant à l'un des trois ensembles précédents augmentée de `1`. Il indique que l'opération porte sur les `width` descripteurs (`0, 1, ..., width -1`).

`timeout` pointe sur une structure de type `timeval` qui définit un temps maximal d'attente avant que l'une des opérations souhaitées soit possible.
Une valeur `NULL` correspond à un temps d'attente infini.

Retour de la primitive

Un processus appelant est bloqué jusqu'à ce que l'une des conditions suivantes se réalise :

- L'un des évenements attendus sur *un des descripteurs de l'un des ensembles* s'est produit. Les ensembles `*readfds`, `*writefds` et `*execptfds` sont mis à jour et contiennent les descripteurs sur lesquels l'opération correspondante est possible : la macro `FD_ISSET` permet de tester l'appartenance des descripteurs à ces ensembles.
  En cas d'utilisation de `select` dans une boucle, il est donc nécessaire de réinitialiser les trois ensembles avant de rappeler la primitive `select`.
  La valeur de retour est le nombre total de descripteurs sur lesquels une opération est possible.

- Le temps d'attente maximum s'est écoulé.
  La valeur de retour est alors 0, et les trois ensembles sont également mis à jour.
- Un signal capté par le processus est survenu : la valeur de retour est alors -1 et la variable `errno` a la valeur `EINTR`.

En cas d'erreur, la primitive renvoie -1 et `errno` vaut :

- `EBADF` : `s` est un descripteur invalide,
- `EINTR` : le processus appelant `select` a reçu un signal avant qu'un des événements attendus soit arrivé, et le signal a interrompu l'appel-système,
- `EFAULT` : un des pointeurs a une valeur incorrecte,
- `EINVAL` : le temps a une valeur incorrecte.

Chapitre 8

La communication en mode connecté (TCP)

8.1 Principes

C'est le mode de communication associé aux sockets de type `SOCK_STREAM`. Il permet d'échanger des séquences de caractères continues et non structurées en messages.
Dans le cas du domaine `AF_INET`, le protocole sous-jacent est TCP, la communication est donc fiable.

Ainsi que l'indique son nom, le mode connecté nécessite l'établissement d'une connexion (un *circuit virtuel*) entre deux points.
Alors que le mode de communication par datagrammes présente un aspect symétrique dans la mesure où chacun des points de transport (les sockets) sont dans le même état et peut prendre l'initiative de la connexion, le mode connecté met en lumière la dissymétrie des deux processus impliqués :

- l'une des deux entités, en position de *serveur*, est en attente passive de demande de connexion,
- l'autre entité, en position de *client*, doit prendre l'initiative de la connexion en demandant au serveur s'il accepte la connexion.

Après cette phase initiale, on retrouve la symétrie : chaque extrémité peut envoyer et recevoir des caractères.

8.2 Le point de vue du serveur

8.2.1 Introduction

Le rôle du serveur est passif pendant l'établissement de la connexion :

- le serveur doit disposer d'une *socket d'écoute* ou *socket de rendez-vous* attachée au port TCP correspondant au service (et donc supposé connu des clients).
- le serveur doit aviser le système auquel il appartient qu'il est prêt à accepter les demandes de connexion, sur la socket de rendez-vous (par la primitive `listen`).
- puis le serveur se met en attente de connexion sur la socket de rendez-vous (par la primitive **bloquante** `accept`).
- lorsqu'un client prend l'initiative de la connexion, le processus serveur est **débloqué** et une nouvelle socket, appelée *socket de service*, est créée :

c'est cette socket de service qui est connectée à la socket du client.
- le serveur peut alors déléguer le travail à un nouveau processus créé par `fork` et reprendre son attente sur la socket de rendez-vous,
  ou traiter lui-même la demande (mais il risque de faire attendre des nouveaux clients).

8.2.2 L'ouverture du service : `listen`

```
#include <sys/types.h>
#include <sys/socket.h>
int listen(int s,      /* socket de rendez-vous */
int backlog            /* nombre maximum de connexions pendantes */
);
```

Cette primitive permet à un serveur de déclarer un service ouvert auprès de son entité TCP locale.
Après cet appel, l'entité TCP locale commence à recevoir les demandes de connexions (appelées des *connexions pendantes*), le paramètre `backlog` définit la taille d'une file d'attente où sont mémorisées les connexions pendantes.

La valeur de retour est 0 en cas de réussite. En cas d'erreur, la valeur de retour est -1 et `errno` vaut :
- `EBADF` : `s` est un descripteur invalide,
- `ENOTSOCK` : `s` n'est pas une socket,
- `EOPNOTSUPP` : type de socket incorrect.

8.2.3 La prise en compte d'une connexion : `accept`

```
#include <sys/types.h>
#include <sys/socket.h>
int accept(int s,      /* socket de rendez-vous */
struct sockaddr *addr, /* pointeur sur l'adresse de la socket connectee */
int *addrlen           /* pointeur sur la longueur de l'adresse */
);
```

Cette primitive permet à un serveur d'extraire une connexion *pendante* de la file d'attente associée à la socket de rendez-vous `s`.
Ainsi le serveur prend en compte un nouveau client : une nouvelle socket *de service* est créée, est connectée au client et son descripteur est renvoyé par la primitive `accept`.
L'adresse de la socket du client avec lequel la connexion est établie est écrite dans la zone pointée par `addr` si sa valeur est différente de `NULL` (la longueur est alors donnée par `*addrlen`).

Un appel à `accept` est normalement *bloquant* s'il n'y a aucune connexion pendante. En cas de succès, il renvoie un entier non-négatif, constituant un descripteur pour la nouvelle socket. En cas d'erreur, la valeur de retour est -1 et `errno` vaut :
- `EBADF` : `s` est un descripteur invalide,
- `EINTR` : le processus appelant `accept` a reçu un signal avant qu'un des clients attendus soit arrivé, et le signal a interrompu l'appel-système,
- `ENOTSOCK` : `s` n'est pas une socket,
- `EOPNOTSUPP` : type de socket incorrect

- EWOULDBLOCK : la socket est non-bloquante et il n'y a pas de connexions pendantes à accepter,
- ... (peut dépendre du système utilisé).

8.2.4 Un exemple de serveur

```
/* Lancement d'un serveur :  serveur_tcp port */

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <sys/types.h>
#include <netdb.h>
#include <sys/socket.h>
#include <netinet/in.h>

#define TRUE 1

main (int argc, char **argv)
{
int socket_RV, socket_service;
char buf[256];
int entier_envoye;
struct sockaddr_in adresseRV;
int lgadresseRV;
struct sockaddr_in adresseClient;
int lgadresseClient;
struct hostent *hote;
unsigned short port;

/* creation de la socket de RV */
if ((socket_RV = socket(AF_INET, SOCK_STREAM, 0)) == -1)
        {
   perror("socket");
   exit(1);
        }
/* preparation de l'adresse locale */
port = (unsigned short) atoi(argv[1]);

adresseRV.sin_family = AF_INET;
adresseRV.sin_port = htons(port);
adresseRV.sin_addr.s_addr = htonl(INADDR_ANY);
lgadresseRV = sizeof(adresseRV);

/* attachement de la socket a l'adresse locale */
if ((bind(socket_RV, (struct sockaddr *) &adresseRV, lgadresseRV)) == -1)
        {
```

```
   perror("bind");
   exit(3);
        }

/* declaration d'ouverture du service */
if (listen(socket_RV, 10)==-1)
        {
  perror("listen");
   exit(4);
        }

/* boucle d'attente de connexion */
while (TRUE)
   {
   printf("Debut de boucle\n");
   fflush(stdout);

   /* attente d'un client */
   lgadresseClient = sizeof(adresseClient);
   socket_service=accept(socket_RV, (struct sockaddr *) &adresseClient, &lgadresseClient);
    if (socket_service==-1 && errno==EINTR)
                {  /* reception d'un signal */
                continue;
                }
   if (socket_service==-1)
                {  /* erreur plus grave */
                perror("accept");
                exit(5);
                }

   /* un client est arrive */
   printf("connexion acceptee\n");
   fflush(stdout);

   /* lecture dans la socket d'une chaine de caractères */
   if (read(socket_service, buf, 256) < 0)
                {
                perror("read");
                exit(6);
                }
   printf("Chaine recue : %s\n", buf);
   fflush(stdout);

   /* ecriture dans la socket d'un entier */
   entier_envoye = 2006 ;
   if (write(socket_service, &entier_envoye, sizeof(int)) != sizeof(int))
               {
```

```
                perror("write");
                exit(7);
                }
   printf("Entier envoye : %d\n", entier_envoye);

   close(socket_service);
   }
close(socket_RV);
}
```

8.3 Le point de vue du client

8.3.1 Introduction

Le rôle du client est actif pendant l'établissement de la connexion :
- le client doit disposer d'une socket attachée à un port TCP quelconque.
- le client doit construire l'adresse du serveur, dont il doit connaître le numéro de port de la socket de rendez-vous, (il obtient éventuellement l'adresse IP du serveur à partir de son nom par la primitive `gethostbyname`).
- puis le client demande la connexion de sa socket locale à la socket de rendez-vous du serveur (par la primitive **bloquante** `connect`).
- lorsque le serveur accepte la connexion, le processus client est **débloqué** et peut dialoguer avec le serveur.

8.3.2 La demande de connexion : `connect`

```
#include <sys/types.h>
#include <sys/socket.h>
int connect(int sockfd,      /* socket locale */
struct sockaddr *serv_addr,  /* pointeur sur l'adresse du serveur */
int addrlen                  /* longueur de l'adresse */
);
```

Contrairement à ce qui se passait dans le cas d'une demande de pseudo-connexion (voir 9.5), le processus appelant `connect` est bloqué jusqu'à ce que la négociation entre les deux entités TCP concernées soit achevée (il est cependant possible de rendre l'appel non bloquant).

La demande de connexion réussit, et la primitive renvoie 0 si les conditions suivantes sont réalisées :

1. les paramètres sont localement corrects,
2. il existe une socket de type `SOCK_STREAM` attachée à l'adresse `*name` (dans le même domaine que la socket locale `s`) et cette socket est dans l'état `LISTEN` (c'est-à-dire qu'un appel à `listen` a été réalisé pour cette socket),
3. la socket locale `s` n'est pas déjà connectée,
4. la socket d'adresse `*name` n'est pas actuellement utilisée dans une autre connexion (sauf paramètrage exceptionnel de cette socket),
5. la file des connexions pendantes de la socket distante n'est pas pleine.

Dans le cas où l'une des quatre premières conditions n'est pas remplie, la fonction renvoie -1 et `errno` vaut :
- `EBADF` : `s` est un descripteur invalide,
- `EINTR` : le processus appelant `connect` a reçu un signal avant que la connexion aie eu lieu, et le signal a interrompu l'appel-système,
- `ENOTSOCK` : `s` n'est pas une socket,
- `EOPNOTSUPP` : type de socket incorrect
- `EISCONN` : la socket locale est déjà connectée,
- `EADDRINUSE` : la socket distante est déjà connectée,
- ... (peut dépendre du système utilisé).

Si la file d'attente est pleine, le comportement est particulier :

- Si la socket locale est en mode bloquant, le processus est bloqué. La demande de connexion est itérée pendant un certain temps : si au bout de ce laps de temps la connexion n'a pas pu être établie, la demande est abandonnée. La fonction renvoie alors -1 et `errno` vaut ETIMEDOUT.
- Si la socket locale est en mode non-bloquant, le retour est immédiat avec la valeur de retour -1 et `errno` qui vaut EINPROGRESS. Cependant ce retour ne correspond pas à une véritable erreur, la demande de connexion n'est pas abandonnée mais automatiquement réitérée comme dans le mode bloquant.
  Pour déterminer si la connexion a pu être établie, le processus pourra utiliser la primitive `select` en testant le descripteur en écriture.
  Si une autre demande de connexion est formulée avant que cet essai de connexion soit terminé, la fonction renvoie -1 et `errno` vaut EALREADY.

8.3.3 Un exemple de client

```
/* Lancement d'un client : client_tcp port machine_serveur */

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <sys/types.h>
#include <netdb.h>
#include <sys/socket.h>
#include <netinet/in.h>

main (int argc, char **argv)
{
int sock;
char buf[256];
int entier_recu;
struct sockaddr_in adresse_serveur;
struct hostent *hote;
unsigned short port;

/* creation de la socket locale */
if ((sock = socket(AF_INET, SOCK_STREAM, 0)) == -1)
   {
   perror("socket");
   exit(1);
   }

/* recuperation de l'adresse IP du serveur (a partir de son nom) */
if ((hote = gethostbyname(argv[2])) == NULL)
   {
   perror("gethostbyname");
```

```
   exit(2);
   }

/* preparation de l'adresse du serveur */
port = (unsigned short) atoi(argv[1]);

adresse_serveur.sin_family = AF_INET;
adresse_serveur.sin_port = htons(port);
bcopy(hote->h_addr, &adresse_serveur.sin_addr, hote->h_length);
printf("L'adresse du serveur en notation pointee %s\n", inet_ntoa(adresse_serveur.sin_addr));
fflush(stdout);

/* demande de connexion au serveur */
if (connect(sock, (struct sockaddr *) &adresse_serveur, sizeof(adresse_serveur))==-1)
   {
   perror("connect");
   exit(3);
   }
/* le serveur a accepte la connexion */
printf("connexion acceptee\n");
fflush(stdout);

/* ecriture dans la socket d'une chaine */
strcpy(buf, "Bonne année");
if (write(sock, buf, strlen(buf)+1) != strlen(buf)+1)
   {
   perror("write");
   exit(4);
   }
printf("Chaine envoyée : %s\n", buf);
fflush(stdout);

/* lecture dans la socket d'un entier */
if (read(sock, &entier_recu, sizeof(int)) < 0)
   {
   perror("read");
   exit(5);
   }
printf("Entier reçu : %d \n", entier_recu);
fflush(stdout);
}
```

8.4 Le dialogue client/serveur

8.4.1 Introduction

Une fois la connexion établie, la communication redevient symétrique.
Les deux processus peuvent échanger des suites de caractères en *duplex* par l'intermédiaire des deux sockets connectées.

Une différence essentielle est que, dans la communication via des sockets de type `SOCK_DGRAM`, le découpage de l'envoi en messages n'est pas préservée à la reception (une lecture peut provenir de plusieurs écritures).

Par ailleurs dans le cas du domaine `AF_INET`, le protocole sous-jacent (TCP) garantit que les caractères seront reçus dans le même ordre que celui de leur envoi, à l'exception de la possibilité d'adresser un caractère dit urgent ou hors bande (*Out Of Band*).

8.4.2 L'envoi de caractères

L'interface standard `write`

Une première solution est d'utiliser `write` :

```
#include <sys/types.h>
#include <sys/uio.h>
int write(int fd,      /* descripteur */
const void *buf,       /* adresse du buffer a envoyer */
int count              /* nombre d'octets maximum \'a envoyer */
);
```

La fonction renvoie le nombre de caractères envoyés (éventuellement 0) et -1 en cas d'erreur.

L'interface standard `send`

```
#include <sys/types.h>
#include <sys/socket.h>
int send(int s,
const void *msg,
int len,
unsigned int flags
);
```

Les trois premiers paramètres sont identiques à ceux de `write`. La primitive `send` permet pour les sockets de type `SOCK_STREAM` d'utiliser l'option `MSG_OOB` dans le domaine `AF_INET` qui indique que les données à transmettre ont un caractère urgent.

La fonction renvoie le nombre de caractères envoyés (éventuellement 0). Dans le cas où le tampon d'émission est plein, le processus est bloqué. Dans le cas où la connexion a été fermée par l'autre extrémité, on se trouve dans la même situation que l'écriture dans un *tube* sans lecteur : le processus écrivain reçoit le signal `SIGPIPE` ce qui entraine sa terminaison (comportement par défaut).

En cas d'erreur, la valeur de retour est -1 et `errno` vaut :

- `EBADF` : `s` est un descripteur invalide,
- `EINTR` : le processus appelant `send` a reçu un signal avant que les caractères puissent être "bufferises" pour être envoyées, et le signal a interrompu l'appel-système,
- `ENOTSOCK` : `s` n'est pas une socket,
- `EWOULDBLOCK` : la socket est non-bloquante et l'opération demandée serait bloquante,
- ... (peut dépendre du système utilisé).

8.4.3 La réception de caractères

L'interface standard `read`

Une première solution est d'utiliser `read` :

```
#include <sys/types.h>
#include <sys/uio.h>
int read(int fd,       /* descripteur */
void *buf,             /* adresse du buffer pour recevoir les donnees */
int count              /* nombre d'octets maximum \'a recevoir */
);
```

La fonction renvoie le nombre de caractères reçus (éventuellement 0) et -1 en cas d'erreur.

L'interface standard `recv`

```
#include <sys/types.h>
#include <sys/socket.h>
int recv(int s,
void *msg,
int len,
unsigned int flags
);
```

Les trois premiers paramètres sont identiques à ceux de `read`.
Le dernier paramètre a soit la valeur 0, soit une combinaison bit à bit des constantes symboliques `MSG_PEEK` (pour consulter sans extraire) et `MSG_OOB` dans le domaine `AF_INET` (pour lire une donnée urgente).

La fonction renvoie le nombre de caractères reçus (éventuellement 0, ce qui indique que la connexion a été fermée). Dans le cas où le tampon de réception est vide, le processus est bloqué.

En cas d'erreur, la valeur de retour est -1 et `errno` vaut :

- `EBADF` : `s` est un descripteur invalide,
- `EINTR` : le processus appelant `send` a reçu un signal avant que les caractèrs puissent être reçus, et le signal a interrompu l'appel-système,
- `ENOTSOCK` : `s` n'est pas une socket,
- `EWOULDBLOCK` : la socket est non-bloquante et l'opération demandée serait bloquante,
- ... (peut dépendre du système utilisé).

8.5 La fermeture de la connexion

8.5.1 La primitive `close`

L'appel de la primitive `close` sur le dernier descripteur d'une socket entraîne la suppression de la socket.
Cependant, dans le cas d'une socket de type `SOCK_STREAM` du domaine `AF_INET`, s'il existe encore des données dans le tampon d'émission de la socket, le système continue d'essayer de les acheminer avant de libérer complètement les ressources (il est d'ailleurs possible de rendre cette primitive bloquante).

8.5.2 La primitive `shutdown`

```
#include <sys/types.h>
#include <sys/socket.h>
int shutdown(
int s,
int how
);
```

Elle permet de rendre compte, au niveau de la fermeture d'une socket, de l'aspect *duplex* de la communication.

Le paramètre `how` détermine le niveau de fermeture souhaité :
- 0 : la socket n'accepte plus d'opérations de lecture : un appel à `read` ou à `recv` renverra 0 ;
- 1 : la socket n'accepte plus d'opérations d'écriture : un appel à `write` ou à `send` provoquera la génération d'un exemplaire du signal `SIGPIPE`, et s'il est capté, une valeur de retour -1 (`errno = EPIPE`) ;
- 2 : la socket n'accepte plus ni opérations de lecture, ni opérations d'écriture.

8.6 Squelettes de serveur et de client

Les squelettes qui suivent peuvent être utilisés pour fabriquer plus facilement des serveurs et des clients TCP.

8.6.1 Le squelette d'un serveur créant un processus de service

Le code de ce serveur-type, une fois un client accepté, confie la gestion du dialogue avec ce client à un processus dédié à cette gestion (grâce à `fork` puis `exec`).

Le lancement du serveur sera réalisé avec deux paramètres :
- le premier correspond au numéro du port d'attachement de la socket d'écoute (de rendez-vous),
- le deuxième correspond à la référence du fichier binaire exécuté par un processus de service : dans ce processus, la socket sera associée à son entrée standard (qui est donc redirigée sur la socket de service avant le recouvrement par `exec`).

```
/* Lancement d'un serveur :  serveur_tcp port reference */

#include <stdio.h>
```

```
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <signal.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>

#define TRUE 1

struct sigaction action;

int creer_socket(int type, int *ptr_port, struct sockaddr_in *ptr_adresse)
{
struct sockaddr_in adresse;
int desc;
int longueur = sizeof(struct sockaddr_in);

if ((desc = socket(AF_INET, type, 0)) == -1) {
perror("Creation socket impossible\n");
return -1;
}

adresse.sin_family=AF_INET;
adresse.sin_addr.s_addr=htonl(INADDR_ANY);
adresse.sin_port=htons(*ptr_port);

if (bind(desc, &adresse, longueur) == -1) {
perror("Attachement de la soccket impossible\n");
close(desc);
return -1;
}

if (ptr_adresse != NULL)
getsockname(desc, ptr_adresse, &longueur);
return desc;
}

void eliminer_zombie(int sig){
printf("terminaison d'un processus de service\n");
wait(NULL);
}

main(int argc, char *argv[]){
struct sockaddr_in adresse;
int lg_adresse;
int port;
```

```
int socketRV, socket_service;
if (argc!=3){
fprintf(stderr, "Nombre de parametres incorrect\n");
exit(2);}

/* test d'existence du programme a executer pour le service */
if (access(argv[2], X_OK)!=0){
fprintf(stderr, "Fichier %s non executable\n", argv[2]);
exit(2);}

/* detachement du terminal */
if (fork()!=0) exit(0);
setsid();
printf("serveur de pid %d lance\n", getpid());

/* handler de signal  SIGCHLD */
action.sa_handler=eliminer_zombie;
sigaction(SIGCHLD, &action, NULL);

/* creation et attachement de la socket d'ecoute */
port=atoi(argv[1]);
lg_adresse=sizeof(adresse);
if ((socketRV=creer_socket(SOCK_STREAM, &port, &adresse))==-1){
fprintf(stderr, "Creation socket ecoute impossible\n");
exit(2);}

/* declaration d'ouverture du service */
if (listen(socketRV, 10)==-1){
perror("listen");
exit(2);}

/* boucle d'attente de connexion */
while(TRUE){
socket_service=accept(socketRV, &adresse, &lg_adresse);

/* reception d'un signal (probablement SIGCHLD) */
if (socket_service==-1 && errno==EINTR)
continue;

/* erreur plus grave */
if (socket_service==-1){
perror("accept");
exit(2);}
printf("connexion acceptee\n");

/* lancement du processus de service */
if (fork()==0){
```

```
/* il n'utilise plus la socket d'ecoute */
close(socketRV);
/* redirection de l'entree standard sur socket_service */
dup2(socket_service, STDIN_FILENO);
close(socket_service); /* descripteur inutile */
execlp(argv[2], argv[2], NULL);
perror("execlp");
exit(2);}

/* le serveur principal n'utilise pas socket_service */
close(socket_service);
}
}
```

8.6.2 Le squelette d'un client

Le code de ce client-type pourra se connecter à un serveur-type donné plus haut. On suppose qu'une fois la connexion avec le serveur établie , le client exécute la fonction `client_service` qui reçoit en premier paramètre le descripteur de la socket du client, en second le nombre de paramètres spécifiques et en troisième un vecteur de ces différents paramètres.

Le lancement du client sera réalisé avec plusieurs paramètres :

- le premier est le nom de la machine où s'exécute le serveur,
- le deuxième correspond au numéro du port du service contacté,
- les différents paramètres spécifiques.

```
/* Lancement d'un client : client_tcp serveur port liste_parametres */

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>
#include <errno.h>
#include <netdb.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>

void client_service(int socket_client, int argc, char *argv[]);

int creer_socket(int type, int *ptr_port, struct sockaddr_in *ptr_adresse)
{
struct sockaddr_in adresse;
int desc;
int longueur = sizeof(struct sockaddr_in);

if ((desc = socket(AF_INET, type, 0)) == -1) {
perror("Creation socket impossible\n");
return -1;
```

```
}

adresse.sin_family=AF_INET;
adresse.sin_addr.s_addr=htonl(INADDR_ANY);
adresse.sin_port=htons(*ptr_port);

if (bind(desc, &adresse, longueur) == -1) {
perror("Attachement de la socket impossible\n");
close(desc);
return -1;
}

if (ptr_adresse != NULL)
getsockname(desc, ptr_adresse, &longueur);
return desc;
}

main(int argc, char *argv[]){
int port;
int socket_client;
struct hostent *hp;
struct sockaddr_in adresse_serveur, adresse_client;

if (argc<3){
fprintf(stderr, "Nombre de parametres incorrect\n");
exit(2);}
/* test d'existence du serveur */
if ((hp=gethostbyname(argv[1]))==NULL){
fprintf(stderr, "Serveur %s inconnu\n", argv[1]);
exit(2);}

/* creation et attachement de la socket du client (port quelconque) */
port=0;
if ((socket_client=creer_socket(SOCK_STREAM, &port, &adresse_client))==-1){
fprintf(stderr, "Creation socket client impossible\n");
exit(2);}
printf("client sur le port %d\n", ntohs(adresse_client.sin_port));

/* preparation de l'adresse du serveur */
adresse_serveur.sin_family=AF_INET;
adresse_serveur.sin_port=htons(atoi(argv[2]));
memcpy(&adresse_serveur.sin_addr.s_addr, hp->h_addr, hp->h_length);

/* demande de connexion au serveur */
if (connect(socket_client, &adresse_serveur, sizeof(adresse_serveur))==-1){
perror("connect");
exit(2);}
```

```
printf("connexion acceptee\n");
client_service(socket_client, argc-3, argv+3);
}
```

Chapitre 9

La communication en mode non connecté (UDP)

9.1 Principes

Quelque soit le domaine (`AF_UNIX` ou `AF_INET`), les principales caractéristiques de la communication par datagrammes sont :

- lorsqu'un datagramme est envoyé, l'expéditeur n'obtient aucune information sur l'arrivée de son message à destination,
- les limites des messages sont préservées.

La communication avec UDP (échange de datagrammes) est réalisée par l'intermédiaire de sockets de type `SOCK_DGRAM` dans le domaine `AF_INET`.
Un processus souhaitant émettre un message à destination d'un autre doit :

- d'une part disposer d'un point de communication local (descripteur de socket sur le système local obtenu avec la primitive `socket`, et éventuellement nommé avec la primitive `bind`),
- d'autre part connaître une adresse (adresse IP et numéro de port) sur le système auquel appartient son interlocuteur (en espérant que cet interlocuteur dispose d'une socket **attachée** à cette adresse ... ce type de communication ne permettant pas d'en être certain).

Les sockets sont utilisés en mode non connecté : en principe, toute demande d'envoi doit comporter l'adresse de la socket destinataire (bien qu'il soit possible d'établir une *pseudo-connexion* entre deux sockets de type `SOCK_DGRAM`).

9.2 Primitives d'envoi et de réception

9.2.1 Introduction

Un processus désirant communiquer avec le monde extérieur par l'intermédiaire d'une socket de type `SOCK_DGRAM` doit réaliser, selon les circonstances, un certain nombre des opérations suivantes :

- demander la création d'une socket dans le domaine adapté aux applications avec lesquelles il souhaite communiquer,
- demander éventuellement l'attachement de cette socket sur un port convenu ou un port quelconque selon qu'il joue un rôle de serveur attendant des requêtes ou celui de client prenant l'initiative d'interroger un serveur,
- construire l'adresse de son interlocuteur en mémoire : tout client désirant s'adresser à un serveur doit en connaître l'adresse et donc la préparer en mémoire (éventuellement avec la primitive `gethostbyname` pour obtenir l'adresse IP à partir du nom) ;
  s'il s'agit d'un serveur, il recevra l'adresse de son émetteur avec chaque message ;
- procéder à des émissions et des réceptions de messages ; les sockets du type `SOCK_DGRAM` sont en mode non connecté ; chaque demande d'envoi s'accompagne de la spécification complète de l'adresse du destinataire (bien qu'il soit possible de réaliser des "pseudo-connexions" cf. paragraphe 9.5).

9.2.2 La primitive `sendto`

```
#include <sys/types.h>
#include <sys/socket.h>
int sendto(
int s,                      /* descripteur de la socket d'emission */
const void *msg,            /* adresse du message a envoyer */
int len,                    /* longueur du message */
unsigned int flags,         /* option = 0 pour le type SOCK_DGRAM */
const struct sockaddr *to, /* pointeur sur l'adresse du destinataire */
int tolen                   /* longueur de l'adresse de la socket destinataire */
);
```

Un appel à `sendto` correpond à la demande d'envoi, via la socket `s`, du message pointé par `msg` de longueur `len` (éventuellement nulle), à destination de la socket attachée à l'adresse pointée par `to` de longueur `tolen`.

La valeur de retour est le nombre de caractères envoyé en cas de réussite, et -1 en cas d'échec. Seules les erreurs locales sont détectées, notamment il n'y a aucune détection qu'une socket est effectivement attachée à l'adresse destinataire (si c'est faux, le message sera perdu et l'émetteur n'en sera pas avisé).

En cas d'erreur, `errno` vaut :

- `EBADF` : `s` est un descripteur invalide,
- `EINTR` : le processus appelant `sendto` a reçu un signal avant que les données aient été "bufferisés", et le signal a interrompu l'appel-système,
- `EINVAL` : longueur d'adresse incorrecte,
- ... (peut dépendre du système utilisé).

9.2.3 La primitive `recvfrom`

```
#include <sys/types.h>
#include <sys/socket.h>
int recvfrom(
int s,                  /* descripteur de la socket de reception */
void *buf,              /* adresse de recuperation du message recu */
int len,                /* taille de la zone allouee a l'adresse buf */
unsigned int flags,     /* 0 ou MSG_PEEK */
struct sockaddr *from, /* pointeur pour recuperer l'adresse d'expedition */
int *fromlen            /* pointeur sur la longueur de l'adresse
                           de la socket d'expedition */
);
```

Cette primitive permet à un processus d'extraire un message sur une socket dont il possède un descripteur, s'il existe un message (sinon l'appel est bloquant).
Le rôle du paramètre `len` est de donner la longueur de la taille reservée pour mémoriser le message (des caractères seront perdus si cette taille est inférieure à la longueur du message).
Dans le cas où `from` est différent de `NULL`, au retour `from` pointe sur l'adresse d'expédition et `*fromlen` contient la longueur de cette adresse.

En cas d'erreur, `errno` vaut :

- `EBADF` : `s` est un descripteur invalide,
- `EINTR` : le processus appelant `recvfrom` a reçu un signal avant que les données aient été reçus, et le signal a interrompu l'appel-système,
- `EWOULDBLOCK` : la socket est non-bloquante et l'opération demandée serait bloquante,
- ... (peut dépendre du système utilisé).

La valeur de retour est le nombre de caractères reçus en cas de réussite, et -1 en cas d'échec.

9.3 Un exemple

L'"émetteur" envoie la chaîne de caractères "salut" et attend la réponse. Le "receveur" attend l'envoi et répond "tchao".

Code de l'"émetteur" Utilisation : `emetteur port machine_distante`

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <sys/types.h>
#include <netdb.h>
#include <sys/socket.h>
#include <netinet/in.h>

main(int argc, char **argv)
{
  int sock, envoye, recu;
  char buf[256];
  struct sockaddr_in adr;
  int lgadr;
  struct hostent *hote;

/* cr'eation de la socket */
  if ((sock = socket(AF_INET, SOCK_DGRAM, 0)) == -1) {
    perror("socket"); exit(1);}

/* recherche de l'@ IP de la machine distante */
  if ((hote = gethostbyname(argv[2])) == NULL){
    perror("gethostbyname"); exit(2);}

/* pr'eparation de l'adresse distante : port + la premier @ IP */
  adr.sin_family = AF_INET;
  adr.sin_port = htons(atoi(argv[1]));
  bcopy(hote->h_addr, &adr.sin_addr, hote->h_length);
  printf("L'adresse en notation pointee %s\n", inet_ntoa(adr.sin_addr));

/* echange de datagrammes */
  strcpy(buf, "salut");
  lgadr = sizeof(adr);
  if ((envoye = sendto(sock, buf, strlen(buf)+1, 0, &adr, lgadr)) != strlen(buf)+1) {
    perror("sendto"); exit(1);}
  printf("salut envoye\n");
  lgadr = sizeof(adr);
  if ((recu = recvfrom(sock, buf, 256, 0, &adr, &lgadr)) == -1) {
    perror("recvfrom"); exit(1);}
  printf("j'ai recu %s\n", buf);
}
```

Code du "receveur" Utilisation : `receveur port`

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <sys/types.h>
#include <netdb.h>
#include <sys/socket.h>
#include <netinet/in.h>

main (int argc, char **argv)
{
  int sock,recu,envoye;
  char buf[256], nomh[50];
  struct sockaddr_in adr;
  int lgadr;
  struct hostent *hote;

/* cr'eation de la socket */
  if ((sock = socket(AF_INET, SOCK_DGRAM, 0)) == -1) {
    perror("socket"); exit(1);}

/* r'ecup'eration du nom de la machine pr'esente */
  if (gethostname(nomh, 50) == -1) {
    perror("gethostbyname"); exit(1);}
  printf("Je m'execute sur %s\n", nomh);

/* pr'eparation de l'adresse locale : port + toutes les @ IP */
  adr.sin_family = AF_INET;
  adr.sin_port = htons(atoi(argv[1]));
  adr.sin_addr.s_addr = htonl(INADDR_ANY);

/* attachement de la socket a' l'adresse locale */
  lgadr = sizeof(adr);
  if ((bind(sock, &adr, lgadr)) == -1) {
    perror("bind"); exit(1);}

/* 'echange de datagrammes */
  lgadr = sizeof(adr);
  if ((recu = recvfrom(sock, buf, 256, 0, &adr, &lgadr)) == -1) {
perror("recvfrom"); exit(1);}
  printf("j'ai recu %s\n", buf);
  strcpy(buf, "tchao");
  if ((envoye = sendto(sock, buf, strlen(buf)+1, 0, &adr, lgadr)) != strlen(buf)+1) {
    perror("sendto"); exit(1);}
  printf("reponse envoyee ...adios\n");
}
```

9.4 Les messages

Les primitives `sendmsg` et `recvmsg` permettent d'envoyer et de recevoir un message dont le contenu est constitué de fragments non contigus en mémoire.

Les différents fragments, de type `struct iovec` (défini dans le fichier standard `<sys/uio.h>` sont stockés dans un tableau `msg_iov` d'élements de type `struct iovec`, ce tableau appartient à une structure de type `struct msghdr` (définie dans le fichier standard `<sys/socket.h>`).

Les fragments sont rassemblés en un seul message avant envoi, et lors de la reception le message reçu est récupéré sous forme de fragments qui sont stockés selon les indications données par une structure de type `struct msghdr`.

9.5 Les pseudo-connexions

9.5.1 Introduction

Chaque demande d'émission d'un message nécessite la spécification de l'adresse de la socket destinatrice.

Il est possible de mémoriser dans la structure `socket` associée à une socket l'adresse d'une socket *paire* vers laquelle seront automatiquement dirigés les messages émis, sans préciser l'adresse à chaque envoi (ceci ne change rien aux caractéristiques de la communication qui sont celles d'UDP).

```
#include <sys/types.h>
#include <sys/socket.h>
int connect(int s,
struct sockaddr *serv_addr,
int addrlen
);
```

La primitive `connect` permet d'associer à une socket locale, identifiée par `s` à la socket d'adresse `*serv_addr` (de longueur `addrlen`).
Tout nouvel appel à `connect` annule la demande précédente, et si `serv\_addr` a la valeur `NULL`, la socket `s` n'est plus associée à aucune autre.

9.5.2 Les primitives d'envoi et de réception

Une fois pseudo-connectée, une socket ne permet plus d'émettre vers des sockets différentes de celle de la socket *paire*.

En particulier, toute demande d'envoi par `sendto`, même avec l'adresse de la socket *paire*, échouera (valeur `EINVAL` de `errno`).
De même, si un message arrive d'une autre socket, toute tentative de réception entrainera une erreur (valeur `EISCONN` de `errno`) ; il faudra rompre la connexion pour llire ce message.

La destination étant implicite, l'envoi de message se fait soit par la primitive standard sur les descripteurs : `write` ou par la primitive spécifique aux sockets :

```
#include <sys/types.h>
#include <sys/socket.h>
int send(int s,
const char *msg,
int len,
unsigned int flags
);
```

Par rapport à `write`, la primitive `send` n'apporte rien pour les sockets de type `SOCK_DGRAM` dans la mesure où aucune option n'est supportée.

La réception de message peut se faire par la primitive standard sur les descripteurs : `read`, dans ce cas, s'agissant d'une communication en mode datagramme, un appel à `read` provoquera l'extraction d'un message complet.

On peut utiliser la primitive spécifique aux sockets :

```
#include <sys/types.h>
#include <sys/socket.h>
int recv(int s,
char *buf,
int len,
unsigned int flags
);
```

L'emploi de la primitive `recv` permet de consulter la socket sans rien extraire, en utilisant l'option `MSG_PEEK`.

Chapitre 10

XDR

10.1 Rappel sur la couche Présentation

La couche Présentation (couche 6) spécifie ou, optionnellement, négocie la façon dont l'information est représentée pour l'échange par des entités d'application. La couche Présentation fournit la représentation :

1. des données transférées entre entités d'application,
2. de la structure de données utilisée par les entités d'application,
3. et des opérations effectuées sur la structure des données.

La couche Présentation ne s'occupe que la syntaxe des données transférées. La signification des données n'est connue que des entités d'application et non de la couche Présentation. La couche Présentation soutient également un service de présentation simple sans connexion.

10.2 Le protocole XDR (*eXternal Data Representation*)

XDR et RPC ont été créés par Sun Microsystem en 1986, en vue du développement de NFS (Network File System) et NIS (Network Integration Service).

XDR est un protocole du niveau de la couche présentation du modèle OSI. XDR n'assure pas réellement toutes les fonctions de la couche Présentation (au sens de l'ISO). La correspondance entre l'ISO et le monde TCP/IP n'étant pas parfaite pour les couches hautes, XDR s'utilise plutôt à l'intérieur des RPC ou directement lors d'un échange via des sockets.

XDR est un protocole standard (norme ISO 8825) de représentation, de description et de codage de données. Il permet de transférer des données entre systèmes hétérogènes. Il a pour but de régler les problèmes posés par la diversité de représentation interne des objets :

- taille des objets typés (exemple des `int` sur 16 ou 32 bits),
- ordre des octets (Little-Endian [1] ou Big-Endian [2]),
- représentation des objets (nombres négatifs, nombres flottant, ...),
- alignement des objets en mémoire,
- pointeurs et valeurs pointées.

1. Little-Endian : la partie la moins significative se trouve à l'adresse la plus petite
2. Big-Endian : la partie la plus significative se trouve à l'adresse la plus petite

Le principe de base de XDR est de transmettre les données à l'aide d'un flot ordonné d'information. On met bout à bout les différentes valeurs suivant un certain format (appelé quelquefois format canonique ou format neutre).

Sérialisation - Désérialisation On dispose d'un ensemble de fonctions de sérialisation (désérialisation) pour les types de base ainsi qu'une méthode de description standard de structures de données permettant la sérialisation (désérialisation) des différentes données de la structure.

Un processus désirant transmettre une séquence de valeurs encode (sérialise) ces valeurs dans un flot. L'encodage consiste à convertir la représentation locale (interne) dans la représentation neutre. Le flot est transmis à un autre processus, qui décode (désérialise) les valeurs en extrayant du flot des représentations canoniques (neutres) et les convertit en représentation locale (interne).

10.3 Typage explicite et implicite

Un typage explicite signifie que chaque morceau de données contient des informations sur son type (le type est échangé avec la valeur). L'émetteur et le récepteur ne partagent rien d'autre que les données echangées.

Un typage implicite signifie que les morceaux de données ne contiennent aucune information sur leur type. Cela implique que l'émetteur et le recepteur sont d'acord sur l'ordre et le type des données.

10.4 Un exemple illustratif

10.4.1 Echange de données sans XDR

Considérons un exemple où un programme (`writer.c`) écrit 8 entiers sur la sortie standard et un autre programme (`reader.c`) lit 8 entiers de l'entrée standard.

```
#include <stdio.h>

main()          /* writer.c */
{  long i;

for (i = 0; i < 8; i++) {
   if (fwrite((char *) &i, sizeof(i), 1, stdout) != 1) {
    fprintf(stderr, "failed!\n");
    exit(1);
   }
  }
  exit(0);
}
```

fichier writer.c

```
#include <stdio.h>

main()       /* reader.c */
{ long i, j;

 for (j = 0; j < 8; j++) {
   if (fread((char *) &i, sizeof(i), 1, stdin) != 1) {
    fprintf(stderr, "failed!\n");
    exit(1);
   }
   printf("%ld ", i);
  }
  printf("\n");
  exit(0);
}
```

fichier reader.c

En connectant la sortie du programme écrivain avec l'entrée du programme lecteur (au moyen d'un tube (*pipe*), on obtient des résultats identiques sur Sun (Big-Endian) comme sur PC (Little-Endian).

```
sun% writer | reader
0 1 2 3 4 5 6 7
sun%
PC% writer | reader
0 1 2 3 4 5 6 7
PC%
```

Grâce aux commandes distantes (*r-commands*), on peut établir un tube entre deux processus s'exécutant sur des machines différentes, ici entre un programme écrivain sur Sun et un programme lecteur sur PC.

```
sun% writer | rsh PC reader
0 16777216 33554432 50331648 67108864 83886080 100663296
117440512
sun%
```

Des résultats identiques sont obtenus en exécutant le programme écrivain sur PC et le programme lecteur sur Sun. Ces résultats sont dus au fait que l'ordre des octets dans un entier long diffèrent entre Sun et PC. Notez que 16777216 correspond à un 1 placé au 24^{eme} bit.

10.4.2 Echange de données avec XDR

Lorsque des données sont échangées entre machines hétérogènes, on a besoin de données portables. Les programmes sont rendus portables en remplaçant les appels `write()` et `read()` par des appels à une fonction , appelée un filtre, de la librairie XDR `xdr_long()` Ce filtre sait convertir un entier long d'une représentation locale (interne) à une représentation canonique (neutre).

Les versions révisées sont données dans les programmes `writer_XDR.c` et `reader_XDR.c`

```
#include <stdio.h>
#include <rpc/rpc.h> /* xdr is a sub-library of rpc */

main()       /* writer_XDR.c */
{ XDR xdrs;
  long i;

xdrstdio_create(&xdrs, stdout, XDR_ENCODE);
  for (i = 0; i < 8; i++) {
   if (!xdr_long(&xdrs, &i)) {
    fprintf(stderr, "failed!\n"); exit(1); }
  }
  exit(0);
}
```

fichier writer_XDR.c

```
#include <stdio.h>
#include <rpc/rpc.h> /* xdr is a sub-library of rpc */

main()                /* reader_XDR.c */
{ XDR xdrs;
  long i, j;

xdrstdio_create(&xdrs, stdin, XDR_DECODE);
  for (j = 0; j < 8; j++) {
   if (!xdr_long(&xdrs, &i)) {
    fprintf(stderr, "failed!\n"); exit(1); }
   printf("%ld ", i);
  }
  printf("\n");
  exit(0);
}
```

fichier reader_XDR.c

Les programmes révisés sont exécutés sur un Sun, sur un PC et d'un Sun à un PC. Les résultats (corrects) sont donnés ci-dessous.

```
sun% writer | reader
0 1 2 3 4 5 6 7
sun%
PC% writer | reader
0 1 2 3 4 5 6 7
PC%
sun% writer | rsh PC reader
0 1 2 3 4 5 6 7
sun%
```

Chapitre 11

La bibliothèque XDR

11.1 La mise en œuvre de la librairie XDR

Contrairement aux sockets, les fonctions XDR ne sont pas des appels système mais des fonctions de librairie externe au système.

11.1.1 Les fichiers d'include

```
#include <rpc/types.h>
#include <rpc/xdr.h>
```

11.1.2 L'édition de liens

La librairie XDR est automatiquement liée avec un programme C.

11.2 Fonctions de creation de flot XDR et d'accès aux enregistrements

Les données converties du format machine dans leur représentation XDR sont placées dans un flot XDR.

Inversement, les données sont lues dans un flot XDR et transformées de leur représentation XDR vers la représentation machine.

11.2.1 Constantes et types prédéfinis

- `bool_t -> FALSE, TRUE` : booléen
- `xdr_op -> XDR_ENCODE, XDR_DECODE, XDR_FREE`ă : type du flot
- `XDR` : structure opaque de description de flot (<=>`FILE`)

11.2.2 Flot XDR sur disque

Création du flot :

```
void xdrstdio_create (XDR*pt_xdr,FILE*pt_file,enum xdr_op type);
```

Permet de créer un nouveau flot XDR associé à un fichier tel que : les données à encoder seront écrites dans un fichier et les données à décodées seront lues dans le fichier, suivant le type demandé.

Paramètres :
– Avant appel

1. `pt_xdr` : pointeur sur une zone mémoire réservée de taille `sizeof(XDR)`
2. `pt_file` : pointeur sur structure `FILE` associé à un fichier déjà ouvert en lecture (resp. écriture) dans le cas d'un décodage (resp. encodage)
3. `type` : type d'opération effectuée sur le flot (`XDR_ENCODE`, `XDR_DECODE`)

– Après appel

1. `pt_xdr` : pointe sur une zone mémoire qui contient la structure décrivant le flot XDR nouvellement créé

11.2.3 Flot XDR en mémoire

Création du flot :
`void xdrmem_create (XDR*pt_xdr,void*pt, u_int taille, enum xdr_op type);`
Permet de créer un nouveau flot XDR associé à une zone mémoire, tel que : les données à encoder seront écrites en mémoire et les données à décodées seront lues en mémoire, suivant le type demandé.
Paramètres :
– Avant appel

1. `pt_xdr` : pointeur sur une zone mémoire réservée de taille `sizeof(XDR)`
2. `pt, taille` : pointeur sur une zone mémoire réservée de taille, `taille` (`RNDUP`) permettant de stocker les données encodées (x4), ou contenant les données à décoder.
3. `type` : type du flot XDR (`XDR_ENCODE`, `XDR_DECODE`)

– Après appel

1. `pt_xdr` : pointe sur une zone mémoire qui contient la structure décrivant le flot XDR nouvellement créé

11.2.4 Flot XDR structuré

Structuration du flot enregistrements permettant le recouvrement en cas d'erreur
Chaque enregistrement est décomposé en fragments comme suit :

0	Taille	Donnée
0	Taille	Donnée
...	...	...
0	Taille	Donnée
1	Taille	Donnée

TABLE 11.1 – Fragments d'un flot XDR structuré

Pour le dernier fragment, le premier bit vaut 1 et est suivi de la taille des données sur 31 bits. Pour les autres fragments, le premier bit vaut 0 et est suivi de la taille des données sur 31 bits.
Création du flot :

`void xdrrec_create (XDR *pt_xdr, u_int taille_envoi, u_int taille_rec, void* io_handle, (int (*) (void*,char*,int)) read_f, (int (*) (void*,char*,int)) write_f);`

Paramètres :

– Avant appel

1. `pt_xdr` : pointeur sur une zone mémoire réservée de taille `sizeof(XDR)`
2. `taille_envoi, taille_rec` : tailles des tampons d'écriture et de lecture alloués lors de l'appel
3. `io_handle` : pointeur sur structure qui sera passé en premier paramètre des fonctions de lecture et d'écriture (par exemple : `FILE *`)
4. `read_f` : pointeur sur fonction à appeler lorsque le tampon de lecture est vide
5. `write_f` : pointeur sur fonction à appeler lorsque le tampon d'écriture est plein

– Après appel

1. `pt_xdr` : pointe sur une zone mémoire qui contient la structure décrivant le flot XDR nouvellement créé

Ce flot peut être utilisé en encodage et en décodage.
Il suffit de positionner le champ `x_op` de la structure `XDR` à `XDR_ENCODE` ou `XDR_DECODE`
Exemple :
`pt_xdr ->x_op = XDR_ENCODE`

11.2.5 Accès aux enregistrement dans les flots

L'accès aux enregistrements dans les flots se fait à l'aide des filtres XDR (voir la section suivante). Les fonctions `xdr_setpos` et `xdr_getpos` permettent de se positionner dans à un enregistrement particulier ou savoir à quelle position on se trouve. Le premier enregistrement porte le numéro 1.

Après lecture d'enregistrements dans un flot, il est nécessaire de se repositionner en début du flot si celui-ci est modifié et que l'on souhaite accéder aux nouvelles informations qu'il contient. On utilisera alors `xdr_setpos(&flotXDRconsidéré, 1)`.

11.3 Les filtres XDR

Un filtre a trois fonctionnalités. Il permet l'encodage, le décodage mais également la libération de la mémoire qui a pu être allouée par le filtre.

Tous les filtres renvoie `TRUE` en cas de réussite et `FALSE` en cas d'échec.
Les deux arguments d'un filtre sont de même type en encodage et en décodage.
`bool_t xdr_`*type* `(XDR *pt_xdr,type*pt);`
Paramètres :

1. `pt_xdr` : pointeur sur le flot XDR en encodage ou décodage
2. `pt` : pointeur sur la zone contenant la valeur à encoder ou sur la zone réservée pour recevoir la valeur décodée

Retour :
– `TRUE` : en cas de succès
– `FALSE` : en cas d'échec

Type	Filtre	Type XDR
char	xdr_char()	int
short	xdr_short()	int
u_short	xdr_u_short()	u_int
int	xdr_int()	int
u_int	xdr_u_int()	u_int
long	xdr_long()	int
u_long	xdr_u_long()	u_int
float	xdr_float()	float
double	xdr_double()	double
void	xdr_void()	void
enum	xdr_enum()	int

Table 11.2 – Filtres XDR de base

11.3.1 Un exemple illustratif

Echange de données sans XDR

Considérons un exemple où un programme (`emetteur_udp.c`) envoie un entier (`unEntierEnvoye` qui vaut 1) à un autre programme (`receveur_udp.c`) puis reçoit un entier (`unEntierRecu` qui devrait valoir 16777216) de ce même programme.

Le programme (`receveur_udp.c`) reçoit donc un entier (`unEntierRecu` qui devrait valoir 1) en provenance du programme (`emetteur_udp.c`) puis envoie un entier (`unEntierEnvoye` qui vaut 16777216) à ce même programme.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <sys/types.h>
#include <netdb.h>
#include <sys/socket.h>
#include <netinet/in.h>

main(int argc, char **argv)
{
  int sock, envoye, recu;
  int unEntierEnvoye, unEntierRecu;
  struct sockaddr_in adr;
  int lgadr;
  struct hostent *hote;

/* cr'eation de la socket */
  if ((sock = socket(AF_INET, SOCK_DGRAM, 0)) == -1){
    perror("socket"); exit(1);}

/* recherche de l'@ IP de la machine distante */
  if ((hote = gethostbyname(argv[2])) == NULL){
    perror("gethostbyname"); exit(2);}

/* pr'eparation de l'adresse distante : port + la premier @ IP */
  adr.sin_family = AF_INET;
  adr.sin_port = htons(atoi(argv[1]));
  bcopy(hote->h_addr, &adr.sin_addr, hote->h_length);

/* echange de datagrammes */
  unEntierEnvoye = 1;
  lgadr = sizeof(adr);
  if ((envoye = sendto(sock, &unEntierEnvoye, sizeof(unEntierEnvoye), 0, &adr, lgadr))
        != sizeof(unEntierEnvoye)){
    perror("sendto"); exit(1);}
  printf("unEntier envoye : %d\n", unEntierEnvoye);
  lgadr = sizeof(adr);
  if ((recu = recvfrom(sock, &unEntierRecu, sizeof(unEntierRecu), 0, &adr, &lgadr)) == -1){
    perror("recvfrom"); exit(1);}
  printf("unEntier recu : %d\n", unEntierRecu);
}
```

fichier emetteur_udp.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <sys/types.h>
#include <netdb.h>
#include <sys/socket.h>
#include <netinet/in.h>

main (int argc, char **argv)
{
  int sock,recu,envoye;
  int unEntierEnvoye, unEntierRecu;
  struct sockaddr_in adr;
  int lgadr;
  struct hostent *hote;

/* cr'eation de la socket */
  if ((sock = socket(AF_INET, SOCK_DGRAM, 0)) == -1){
    perror("socket"); exit(1);}

/* pr'eparation de l'adresse locale : port + toutes les @ IP */
  adr.sin_family = AF_INET;
  adr.sin_port = htons(atoi(argv[1]));
  adr.sin_addr.s_addr = htonl(INADDR_ANY);

/* attachement de la socket a' l'adresse locale */
  lgadr = sizeof(adr);
  if ((bind(sock, &adr, lgadr)) == -1){
    perror("bind"); exit(1);}

/* 'echange de datagrammes */
  lgadr = sizeof(adr);
  if ((recu = recvfrom(sock, &unEntierRecu, sizeof(unEntierRecu), 0, &adr, &lgadr)) == -1){
     perror("recvfrom"); exit(1);}
  printf("unEntier recu : %d\n", unEntierRecu);
  unEntierEnvoye = 16777216;
  lgadr = sizeof(adr);
  if ((envoye = sendto(sock, &unEntierEnvoye, sizeof(unEntierEnvoye), 0, &adr, lgadr))
        != sizeof(unEntierEnvoye)){
      perror("sendto"); exit(1);}
  printf("unEntier envoye : %d\n", unEntierEnvoye);
}
```

fichier receveur_udp.c

En exécutant ces deux programmes sur deux machines, on obtient

```
sun1% emetteur_udp 2000 sun2
unEntier envoye : 1
unEntier recu : 16777216
sun1%

PC1% emetteur_udp 2000 PC2
unEntier envoye : 1
unEntier recu : 16777216
PC1%

PC1% emetteur_udp 2000 sun2
unEntier envoye : 1
unEntier recu : 1
PC1%
```

```
sun2% receveur_udp 2000
unEntier recu : 1
unEntier envoye : 16777216
sun2%

PC2% receveur_udp 2000
unEntier recu : 1
unEntier envoye : 16777216
PC2%

sun2% receveur_udp 2000
unEntier recu : 16777216
unEntier envoye : 16777216
sun2%
```

Explications

Lorsque les entiers sont échangés entre deux machines ayant la même représentation des entiers, l'ordre des octets n'a pas d'importance.

Lorsque les entiers sont échangés entre deux machines ayant une représentation différente des entiers, le nombre 1 (qui en binaire est codé avec 3 octets valant 0 et un octet valant 1) est transformé en 16777216 (2 à la puissance 24, qui en binaire est codé avec un octet valant 1 et 3 octets valant 0). Selon l'ordre des octets (Big-Endian ou Little-Endian), l'interprétation de la même séquence de 32 bits est soit 1, soit 16777216.

Echange de données avec XDR

Le programme (`emetteur_udp_XDR.c`) encode puis envoie un entier (`unEntierEnvoye` qui vaut 1) à un autre programme (`receveur_udp_XDR.c`) puis reçoit et décode un entier (`unEntierRecu` qui devrait valoir 16777216) de ce même programme.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <sys/types.h>
#include <netdb.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <rpc/xdr.h>

#define TAILLE 1024

main(int argc, char **argv) {
  int sock, envoye, recu;
  int unEntierEnvoye, unEntierRecu;
```

```
  struct sockaddr_in adr;
  int lgadr;
  struct hostent *hote;
  XDR xdr_encode, xdr_decode;
  char datagrammeEnvoye[TAILLE];
  char datagrammeRecu[TAILLE];

/* creations des flots */
  xdrmem_create(&xdr_encode, datagrammeEnvoye, TAILLE, XDR_ENCODE);
  xdrmem_create(&xdr_decode, datagrammeRecu, TAILLE, XDR_DECODE);

/* cr'eation de la socket */
  if ((sock = socket(AF_INET, SOCK_DGRAM, 0)) == -1){
    perror("socket"); exit(1);}

/* recherche de l'@ IP de la machine distante */
  if ((hote = gethostbyname(argv[2])) == NULL){
    perror("gethostbyname"); exit(2);}

/* pr'eparation de l'adresse distante : port + la premier @ IP */
  adr.sin_family = AF_INET;
  adr.sin_port = htons(atoi(argv[1]));
  bcopy(hote->h_addr, &adr.sin_addr, hote->h_length);

/* envoi d'un datagramme */
  unEntierEnvoye = 1;
/* encodage en memoire */
  if (xdr_int(&xdr_encode, &unEntierEnvoye) == FALSE){
    printf("Echec d'encodage\n"); exit(1);}

/* envoi de la memoire contenant l'entier encode */
  lgadr = sizeof(adr);
  if ((envoye = sendto(sock, datagrammeEnvoye, 1024, 0, &adr, lgadr)) != 1024){
    perror("sendto"); exit(1);}
  printf("unEntier envoye : %d\n", unEntierEnvoye);

/* reception d'un datagramme contenant un entier encode */
  lgadr = sizeof(adr);
  if ((recu = recvfrom(sock, datagrammeRecu, 1024, 0, &adr, &lgadr)) == -1){
    perror("recvfrom"); exit(1);}

/* decodage de cet entier */
  if (xdr_int(&xdr_decode, &unEntierRecu) == FALSE){
    printf("Echec de decodage\n"); exit(1);}
  printf("unEntier recu : %d\n", unEntierRecu);
}
```

fichier emetteur_udp_XDR.c

Le programme (`receveur_udp_XDR.c`) reçoit et décode donc un entier (`unEntierRecu` qui devrait valoir 1) en provenance du programme (`emetteur_udp_XDR.c`) puis envoie et encode un entier (`unEntierEnvoye` qui vaut 16777216) à ce même programme.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <sys/types.h>
#include <netdb.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <rpc/xdr.h>

#define TAILLE 1024

main (int argc, char **argv)
{
  int sock,recu,envoye;
  int unEntierEnvoye, unEntierRecu;
  struct sockaddr_in adr;
  int lgadr;
  struct hostent *hote;
  XDR xdr_encode, xdr_decode;
  char datagrammeEnvoye[TAILLE];
  char datagrammeRecu[TAILLE];

/* creations des flots */
  xdrmem_create(&xdr_encode, datagrammeEnvoye, TAILLE, XDR_ENCODE);
  xdrmem_create(&xdr_decode, datagrammeRecu, TAILLE, XDR_DECODE);

/* cr'eation de la socket */
  if ((sock = socket(AF_INET, SOCK_DGRAM, 0)) == -1){
    perror("socket"); exit(1);}

/* pr'eparation de l'adresse locale : port + toutes les @ IP */
  adr.sin_family = AF_INET;
  adr.sin_port = htons(atoi(argv[1]));
  adr.sin_addr.s_addr = htonl(INADDR_ANY);

/* attachement de la socket a' l'adresse locale */
  lgadr = sizeof(adr);
  if ((bind(sock, &adr, lgadr)) == -1){
    perror("bind"); exit(1);}

/* 'echange de datagrammes */
/* reception d'un datagramme contenant un entier encode */
  lgadr = sizeof(adr);
```

```
  if ((recu = recvfrom(sock, datagrammeRecu, 1024, 0, &adr, &lgadr)) == -1){
    perror("recvfrom"); exit(1);}

/* decodage de cet entier */
  if (xdr_int(&xdr_decode, &unEntierRecu) == FALSE){
    printf("Echec de decodage\n"); exit(1);}
  printf("unEntier recu : %d\n", unEntierRecu);

  /* envoi d'un datagramme */
  unEntierEnvoye = 16777216;
/* encodage en memoire */
  if (xdr_int(&xdr_encode, &unEntierEnvoye) == FALSE){
    printf("Echec d'encodage\n"); exit(1);}

/* envoi de la memoire contenant l'entier encode */
  lgadr = sizeof(adr);
  if ((envoye = sendto(sock, datagrammeEnvoye, 1024, 0, &adr, lgadr)) != 1024){
    perror("sendto"); exit(1);}
  printf("unEntier envoye : %d\n", unEntierEnvoye);
}
```

fichier receveur_udp_XDR.c

En exécutant ces deux programmes sur deux machines, on obtient

```
sun1% emetteur_udp 2000 sun2
unEntier envoye : 1
unEntier recu : 16777216
sun1%

PC1% emetteur_udp 2000 PC2
unEntier envoye : 1
unEntier recu : 16777216
PC1%

PC1% emetteur_udp 2000 sun2
unEntier envoye : 1
unEntier recu : 16777216
PC1%
```

```
sun2% receveur_udp 2000
unEntier recu : 1
unEntier envoye : 16777216
sun2%

PC2% receveur_udp 2000
unEntier recu : 1
unEntier envoye : 16777216
PC2%

sun2% receveur_udp 2000
unEntier recu : 1
unEntier envoye : 16777216
sun2%
```

Explications

L'échange d'entiers encodés avec XDR permet de s'affranchir de l'ordre des octets. XDR ne résoud pas seulement ce problème, la bibliothèque XDR permettant d'échanger tout type de données.

11.3.2 Aucune donnée

Occasionnellement, un filtre XDR doit être fourni, même si aucune donnée n'est requise.
`bool_t xdr_void(void);`

11.3.3 Enumérations

`bool_t xdr_enum(XDR *pt_xdr, enum_t *pt);`
Ce filtre assume l'hypothèse qu'une énumération C a la même représentation machine qu'un `int`.

11.3.4 Booléens

`bool_t xdr_bool(XDR *pt_xdr, bool_t *pt);`
Cas particulier de `enum`

11.3.5 Chaînes de caractères

`bool_t xdr_string (XDR *pt_xdr,char **pt,u_int taille_max);`
Paramètres :

1. `pt_xdr` : pointeur de flot XDR
2. `pt` : pointe sur le pointeur de la chaîne de caractères à encoder ou sur une zone mémoire contenant l'adresse de la zone réservé pour stocker la chaîne à décoder
3. `taille_max` : taille maximale de la chaîne

Attention Une chaîne de caractères au sens XDR est un pointeur de caractères qui pointe sur une zone contigue de caractères.

```
/* Exemple */
char *chaine;
chaine = malloc(80);
char *chaine2;
char tableau[80];
chaine2 = &tableau[0];
```

Dans l'exemple ci-dessus, `chaine` et `chaine2` sont des chaînes de caractères (au sens XDR), alors que `tableau` n'en est pas une. Une chaîne est un pointeur sur une zone contiguë de caractères, son type est char *. Un pointeur sur une chaîne de caractères est donc un pointeur sur un pointeur sur une zone contiguë de caractères, son type est char **.

Pour utiliser la fonction `xdr_string`, on écrira donc par exemple :

```
char machaine[80];
char * pt;
...
strcpy(machaine, "chaine à envoyer");
...
xdr_string(&flotXDR, &pt, 80)
...
```

Désérialisation (décodage) d'une chaîne La longueur de la chaîne est calculée, elle ne doit pas excéder `taille_max`. Ensuite, on examine la valeur pointée par `pt`. Si `*pt` est `NULL`, il y a allocation de mémoire dynamique de la longueur nécessaire Si `*pt` est non-nulle, la bibliothèque XDR suppose que la place nécessaire a été allouée pouvant contenir une chaîne d'au plus `taille_max` caractères. La chaîne est alors désérialisée du flot XDR et `*pt` pointe sur la chaîne décodée.

`xdr_wrapstring(XDR *pt_xdr,char **pt)` appelle `xdr_string` avec `taille_max=UINT_MAX`

De nombreuses fonctions comme `xdr_vector`, `xdr_array` et `xdr_pointer` nécessitent en paramètre un pointeur de fonction de type `xdrproc_t`. Le type `xdrproc_t` n'accepte que deux arguments, alors que `xdr_string` en a trois. Il faut dans ce cas utiliser `bool_t xdr_wrapstring`.

11.3.6 Suite d'octets non-interprétées

```
bool_t xdr_opaque(XDR *pt_xdr, char *pt,int taille);
```

Paramètres :

1. `pt_xdr` : pointeur sur le flot XDR
2. `pt` : pointeur sur suite d'octets à encoder ou sur la zone réservée pour stocker la suite d'octets à décoder
3. `taille` : taille de la suite d'octets

11.3.7 Tableau d'octets

```
bool_t xdr_bytes(XDR *pt_xdr,char **pt,u_int *pt_taille, u_int max);
```

Paramètres :

1. `pt_xdr` : pointeur sur le flot XDR
2. `pt` : pointe sur le pointeur de la zone mémoire qui contient le tableau d'octets à encoder ou sur la zone mémoire réservée pour contenir le tableau d'octets à décoder.
3. `pt_taille` : pointe sur la taille du tableau d'octets à encoder ou sur zone réservée contenant avant appel la taille de la zone réservée sous `*pt` et après appel la taille du tableau d'octets décodé.
4. `max` : taille maximale du tableau d'octets à décoder

Si `*pt=NULL` en décodage, il y a allocation dynamique de la mémoire n'ecessaire.

11.3.8 Tableau de taille fixe

```
bool_t xdr_vector(XDR *pt_xdr, void *pt, u_int taille, u_int taille_case,
xdrproc_t xdr_type);
```

Paramètres :

1. `pt_xdr` : pointeur sur le flot XDR
2. `pt` : pointeur sur le tableau
3. `taille` : nombre de case du tableau
4. `taille_case` : taille des éléments du tableau
5. `xdr_type` : pointeur sur la fonction d'encodage et de décodage du type des éléments du tableau

11.3.9 Tableau de taille variable

`bool_t xdr_array(XDR *pt_xdr, void **pt, u_int *taille, u_int max, u_int taille_case, xdr proc_t xdr_type);`

Paramètres :

1. `pt_xdr` : pointe sur le flot XDR
2. `pt` : pointe sur le pointeur de début du tableau à encoder ou sur le pointeur de la zone mémoire réservée pour contenir le tableau décodé
3. `taille` : pointe sur le nombre de cases du tableau à encoder ou sur le nombre de cases de la zone réservée (celle pointée par `*pt`) et après appel pointe sur le nombre de cases effectivement utilisées.
4. `max` : taille maximale de la zone à décoder
5. `taille_case`, `xdr_type` : même chose que xdr_vector

Allocation automatique de mémoire si `*pt=NULL`

11.3.10 Structures

Il n'y a pas de fonction prédéfinies pour l'encodage des structures.
Exemple :

```
struct structure {
     type1 ch1;
     type2 ch2;
     ...
     typen chn;
};

bool_t xdr_structure(XDR *pt_xdr, struct structure *pt)
{
return(
xdr_type1(pt_xdr,&(pt->ch1)) &&

xdr_type2(pt_xdr,&(pt->ch2))&&

...

xdr_typen(pt_xdr,&(pt->ch1)));
}
```

11.3.11 Unions

`bool_t xdr_union(XDR *pt_xdr, enum_t *disc, void *union_pt, struct xdr_discrim *choices, xdr_proc_t defaultarm);`

Paramètres :

1. `pt_xdr` : pointeur sur flot XDR
2. `disc` : pointeur sur discriminant de l'union

3. `union_pt` : pointeur sur l'union
4. `choices` : tableau de pointeur de fonction de codage en fonction du discriminant (cf .exemple)
5. `defaultarm` : pointeur de fonction de codage par défaut.

Exemple :

```
struct xdr_discrim filter[]= {
{INT,xdr_int},
{FLOAT,xdr_float},
{END, NULL}};

typedef struct {
enum type {END=0,INT,FLOAT} disc;
union {
int entier;
float flottant;
} valeur;
} mumerique;

XDR xdr_handle;
...
numerique nb;

nb.valeur.flottant = 3.2;
nb.disc = FLOAT

xdr_union (&xdr_handle, &nb.disc, &nb.valeur, filter, xdr_int);
```

11.3.12 Pointeurs simples

`bool_t xdr_reference(XDR *pt_xdr, char **pt, u_int taille, xdrproc_t f);`
Permet d'encoder ou de décoder un pointeur et l'objet pointé
Paramètres :

1. `pt_xdr` : pointeur sur flot XDR
2. `pt` : pointe sur le pointeur à encoder ou sur zone qui contiendra le pointeur décodé
3. `taille` : taille de l'objet pointé
4. f : fonction XDR du type de l'objet pointé

Errreur si `*pt=NULL` en encodage
Si `*pt=NULL` en décodage alloue la zone necessaire.

11.3.13 Pointeurs

`bool_t xdr_pointer(XDR *pt_xdr, void **pt, u_int taille, xdrproc_t f);`
Mêmes paramètres que `xdr_reference`.
Encode le pointeur `NULL` ce qui permet d'encoder les structures récursives.
Exemple :

```
typedef struct maillon {
int val;
struct maillon *next;
}liste;
bool_t xdr_liste(XDR*pt_xdr,liste*pt_liste) {
return ( xdr_pointer (pt_xdr,&pt_liste->next,sizeof(liste),xdr_liste) &&
xdr_int(pt_xdr,&pt_liste->val));
}
```

Bibliographie

[Com06] D. Comer. *TCP/IP : architecture, protocoles et applications - 5eme édition.* Pearson Education, 2006.

[LM98] François Leslé and Nicolas Macarez. *Que sais-je ? Le multimédia.* P.U.F., 1998.

[Puj00] G. Pujolle. *Les réseaux.* Eyrolles, 2000.

[Puj04] G. Pujolle. *Cours rséaux et télécoms.* Eyrolles, 2004.

[Ser97] C. Servin. *Télécoms De l'ingéniérie aux services.* Masson, 1997.

[Ste98] W. Richard Stevens. *UNIX Network Programming Volume 1.* Prentice Hall, 1998.

[Sus00] Jean-François Susbielle. *Internet multimédia et temps réel.* Eyrolles, 2000.