

LES RESEAUX A TECHNOLOGIE

IP

TABLE DES MATIERES

RESEAU ASPECT LOGICIEL

Le modèle OSI page N°5
Les technologies IP page N°9
Partie N°1 La couche liaison et physique ETHERNET page N°16
Partie N°2 La couche réseau page N°21
Partie N°3 La couche transport page N°33
Partie N°4 La couche application page N°39
Partie N°5 Les protocoles particuliers page N°45
Annexe : Les réseaux locaux page N°48

LES RESEAUX TD1 ET TD2

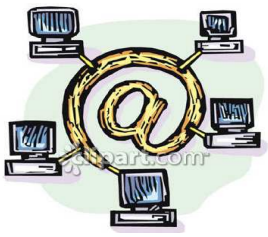
page N°51

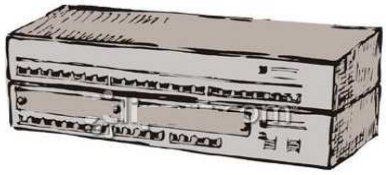
RESEAU ASPECT MATERIEL

1) Les médias de transport page N°56
2) Les principaux types de câblage matériel page N°57
3) Annexe : le matériel page N°58

CONTRÖLE ET COMMANDE A DISTANCE D'UN SYSTEME ASSERVI

1) Les sockets page N°73
2) TD : Les sockets page N°88





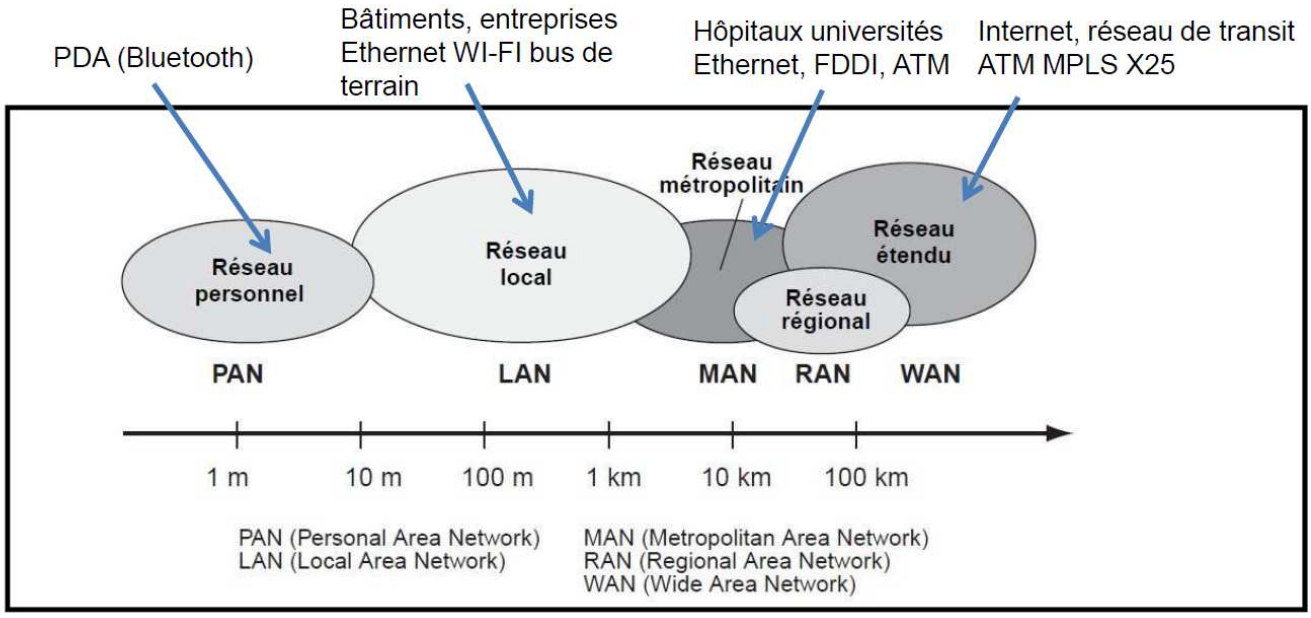
LES RESEAUX A TECHNOLOGIE

TCP-IP : ASPECT LOGICIEL

Les réseaux : aspects logiciels

1) Introduction

Les réseaux actuels permettent l'interconnexion de tous types d'ordinateurs que ce soit de gros serveurs, des stations de travail, des ordinateurs personnelle ou de simples terminaux graphiques. On peut faire une première classification des réseaux à partir de leurs tailles :



Les bus que l'on trouve dans les ordinateurs pour relier les différents composants.

Un réseau personnel **(personnal area network)** interconnecte souvent par des liaisons sans fils des équipements personnel (téléphone portable, agenda électronique …)

Un réseau local **(Local area network)** peut s'étendre de quelques mètres à quelques kilomètres réseau d'entreprise.

Un réseau métropolitain **(Métropolitan area network)** plusieurs lieux situés dans une même ville université administration.

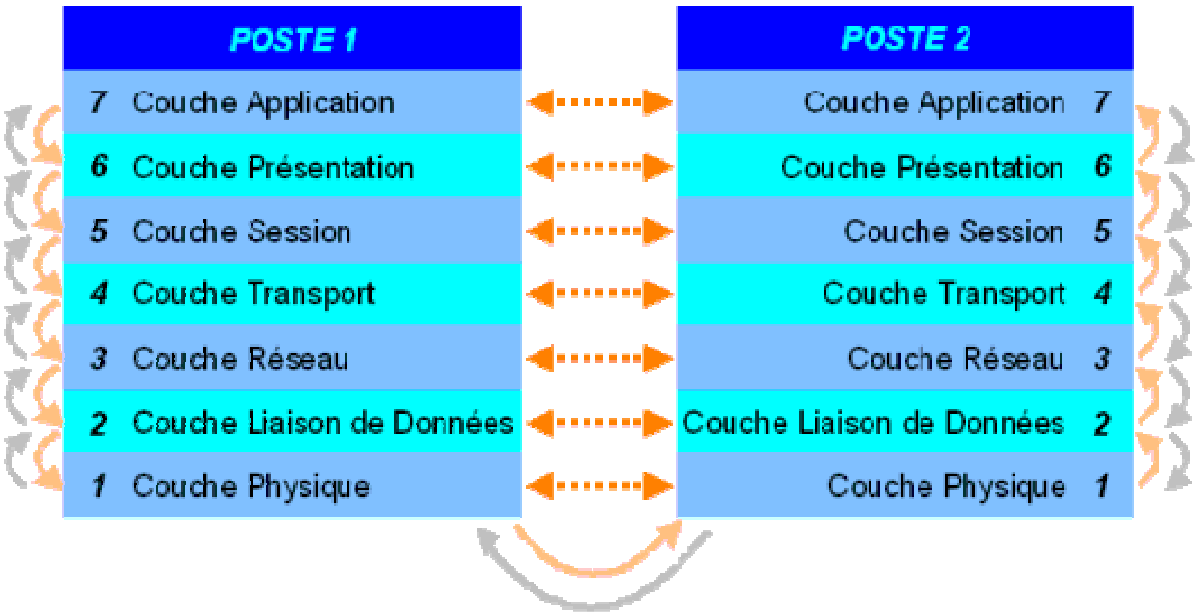
Un réseau étendu (**wide area network**) permet de communiquer à l'échelle d'un pays ou de la planète les infrastructures physiques pouvant être terrestres ou spatiales à l'aide de satellites de télécommunication réseau haut débit et de transit

Le modèle OSI

1) Introduction

Le fondement d'un bon réseau, c'est que le système d'exploitation soit capable:
- De gérer la transmission de données.
- De fournir aux applications des interfaces standard pour leur permettre d'exploiter les ressources du réseau.
C'est le cas de tous les systèmes d'exploitation à jour.
A priori, rien ne devrait obliger les plate formes client et serveur à fonctionner avec le même système d'exploitation. Même si un réseau Microsoft dispose d'outils qui lui sont spécifiques (protocole netbeui), les hôtes de ce réseau peuvent tout de même dialoguer avec ceux d'un réseau Unix.
Pour arriver à cette interopérabilité, il faut que les divers protagonistes se mettent d'accord sur un protocole ou un ensemble de protocoles . **Un protocole est un ensemble de rêgles et de conventions nécessaires pour la communication entre ordinateurs**. Comme ces protocoles sont complexes, ils sont découpés en plusieurs couches pour faciliter leurs implémentation. C'est l'objectif du modèle théorique **O.S.I.** qui décrit comment l'O.S. réseau, encore appelé N.O.S. doit être construit. Il **décrit l'architecture en 7 couches logicielles** présentant chacune des interfaces standard pour communiquer entre elles.
. Le nombre de couches, leur nom et leur fonction varie selon les réseaux. Cependant, dans chaque réseau, l'objet de chaque couche est d'offrir certains services aux couches plus hautes.

2) Le modèle OSI



3) Description succincte des couches.

La couche physique: 1.

C'est la couche spécifique à la "tuyauterie" du réseau. Elle permet de transformer un signal binaire en **un signal compatible avec le support choisi** (cuivre, fibre optique, HF etc.) et réciproquement. C'est à ce niveau que se situe l'adresse MAC (Medium Access Control).
C'est également à ce niveau que l'on trouve le protocole de diffusion de l'information: Ethernet, Token Ring, ATM, etc.
Pour un réseau domestique, c'est Ethernet qui est utilisé.
Cette couche fournit des outils de transmission de **bits** à la couche supérieure, qui les utilisera sans se préoccuper de la nature du médium utilisé.

La couche liaison: 2.

Cette couche assure le contrôle de la transmission des données. Une trame doit être envoyée ou reçue en s'affranchissant d'éventuels parasites sur la ligne. Le contrôle est effectué au niveau trame, au moyen d'un "checksum".
Cette couche Définit le format ainsi que le codage logique de la **trame**. Elle permet également la sécurisation du lien physique.

La couche Réseau: 3.

Cette couche assure la transmission des données sur les réseaux. C'est ici que la notion de routage intervient, permettant l'interconnexion de réseaux différents. C'est dans le cas de TCP/IP la couche Internet Protocol.
Cette couche est la plus haute dans la partie purement "réseau".
Cette couche Permet l'acheminement et le contrôle des données sous forme d'un **datagramme**.
Les transmissions sont routées.

La couche Transport: 4.

Cette couche apparaît comme un superviseur de la couche Réseau. Qu'est-ce à dire? Il n'est par exemple pas du ressort de la couche réseau de prendre des initiatives si une connexion est interrompue. C'est la couche Transport qui va décider de réinitialiser la connexion et de reprendre le transfert des données.

La couche transport assure une communication de bout en bout en régulant le flux de données et en assurant un transport fiable des données.

La couche Session: 5.

Cette couche fournit donc à la couche supérieure des outils plus souples que ceux de la couche transport pour la communication d'informations, en introduisant la notion de session.

La couche Présentation: 6.

Cette couche assure :

Le format de codage interne des données

Les "mots" sont une suite d'octets. Un mot de 32 bits est donc une collection de 4 octets. Chez Intel, les octets sont numérotés de droite à gauche, alors que chez Motorola, ils le sont de gauche à droite. Il s'en suit que si une machine à base Intel envoie des mots à une machine à base Motorola, il vaut mieux tenir compte de ce détail pour ne pas s'y perde...

D'autres exemples pourraient être trouvés dans le style, comme la complémentation à 1 ou à 2 pour les représentations négatives, les divers dialectes ASCII etc.

La compression des données.

Certains transferts de données se font par le biais d'algorithmes de compression. C'est intéressant pour certains types de documents comme le son, l'image ou la vidéo. Dans le modèle OSI, ces algorithmes sont fournis par la couche présentation.

Le cryptage des données.

Même chose pour la cryptographie.

La couche Application: 7.

A priori, cette couche pourrait être la plus simple à comprendre, ce n'est pas obligatoirement le cas. En effet, dans le modèle OSI, cette couche propose également des services: Principalement des services de transfert de fichiers, (FTP), de messagerie (SMTP) de documentation hypertexte (HTTP) etc.
Dans le modèle, les applications ayant à faire du transfert de fichiers utilisent le service FTP fourni par la couche 7.

4) Simplification du modèle OSI : le modèle DOD

Le modèle O.S.I. est tellement théorique qu'il va à l'encontre de l'efficacité. Il est donc souvent simplifié. Les simplifications se bornant toutefois à regrouper les fonctions de plusieurs couches O.S.I. en une seule. Exemple: Le modèle D.O.D. utilisé dans le protocole TCP/IP:

Modèle OSI	Modèle DOD
7 Couche Application	Couche Application
6 Couche Présentation	
5 Couche Session	
4 Couche Transport	Couche Hôte à Hôte
3 Couche Réseau	Couche Internet
2 Couche Liaison de Données	Couche Accès Réseau
1 Couche Physique	

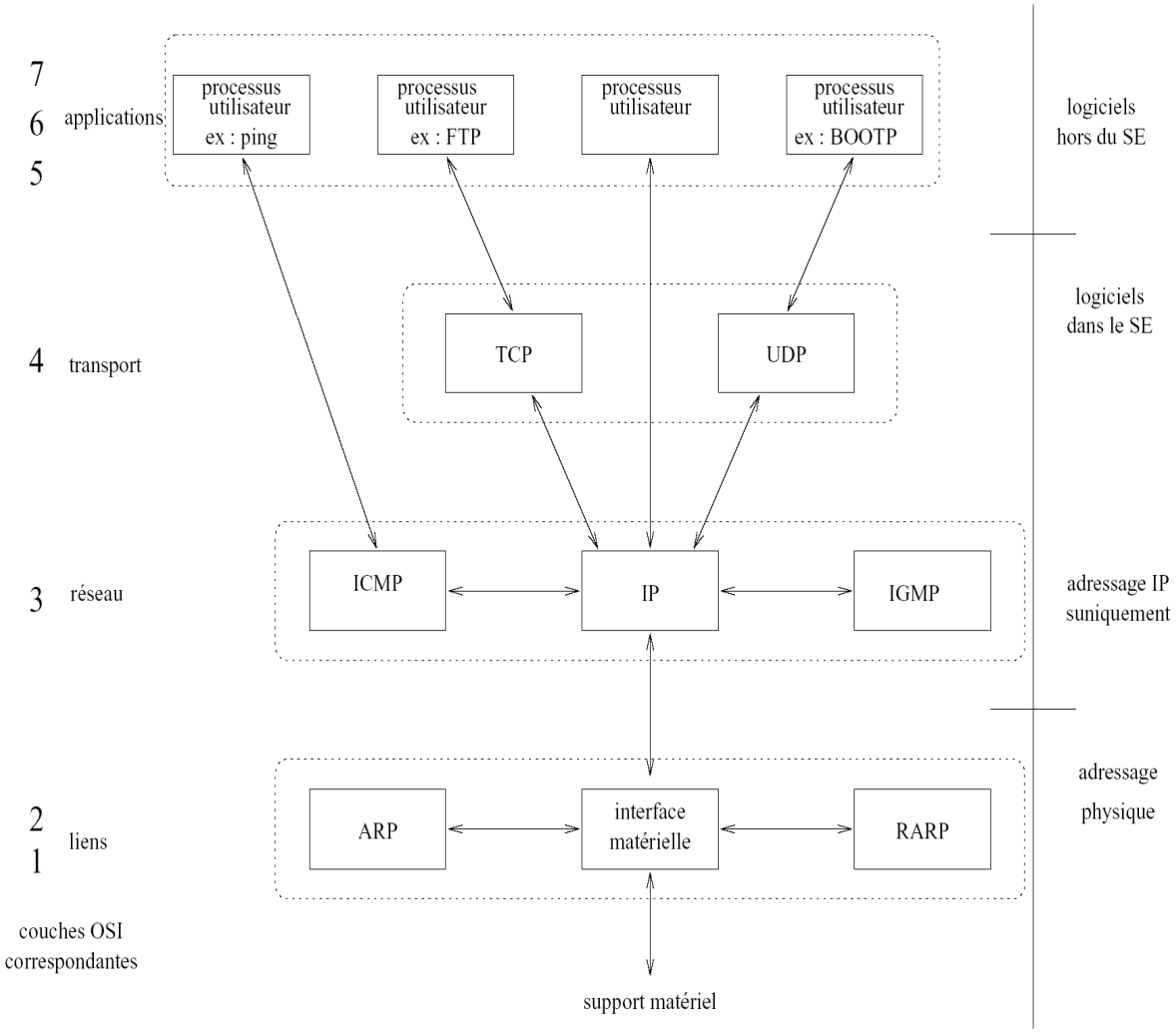
Les technologies IP (internet protocol) et les différentes couches réseaux

1) Le langage réseau de l'INTERNET

Le langage adopté dans l'INTERNET pour communiquer entre machines est le langage réseau **TCP-IP**.
TCP-IP est de fait le premier véritable langage réseau indépendant de tout constructeur d'informatique, ce qui en fait son succès.
Si un réseau parle 'TCP-IP', il n'est pas forcément connecté à l'INTERNET c'est en fait une fédération de réseaux **qui mettent en place un protocole commun**.
.

2) Architecture du protocole TCP-IP

Les logiciels TCP-IP sont structurés en quatres couches de protocole qui s'appuient sur une couche matérielle :



3) La communication dans les réseaux LAN

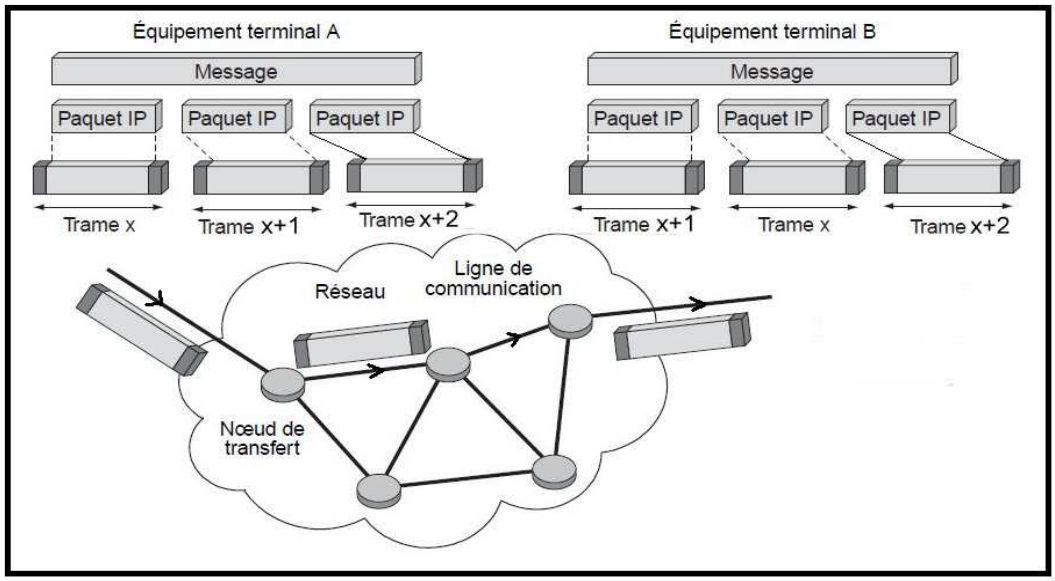
Ils existent différents types de communication dans les réseaux. Dans les réseaux LAN, du type **ETHERNET TCP-IP** cette communication se fait par commutation de paquet.

3.1) La commutation de paquets :

Un paquet est une suite d'information dont la taille ne peut dépasser une valeur fixée.
Un **paquet** ne forme pas toujours un tout logique comme les messages. Un g**rand message sera découpé** pour être acheminé en **plusieurs paquets**. Les différents paquets **ne passent pas forcément par le même chemin**. Une nouvelle fonction apparaît : le **découpage** des messages en paquets à l'émission et le **ré assemblage** des paquets en messages à la réception.

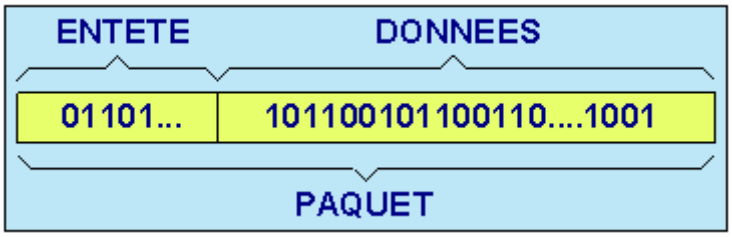
Cette méthode permet d'utiliser la ligne d'une façon optimale et les délais de transmission sont généralement très courts car :

- les besoins en mémoire dans les nœuds sont diminués et leur gestion est facilitée par le fait que tous les paquets ont la même taille à défaut une taille maximale.
- en cas d'erreur, la quantité d'information à ré émettre est plus petite que dans le cas de messages.



Remarque : Les paquets ne sont pas forcément reçus dans le bon ordre

Cette solution implique que chaque hôte du réseau sera astreint à découper les messages qu'il souhaite transmettre en des séquences plus courtes appelées PAQUETS. ces paquets ne seront pas envoyés en vrac dans le réseau !. Comment chaque routeur pourrait-il connaître la destination de chaque paquet ?
Comment le récepteur pourrait ré-assembler tous ces paquets dans le bon ordre et sans en oublier aucun ?. Ces paquets devront donc être en quelque sorte **étiquetés** avant leur envoi.
L'étiquette est une information codée binaire ajoutée **en tête de chaque paquet**. Elle contient les informations nécessaires au routage et au réassemblage des paquets à l'arrivée. On l'appelle l'**entête** de paquet.



Bien sûr, le fait d'ajouter de l'information à chaque paquet "alourdit" le message.
Celui-ci est globalement plus long que le message initial du fait des entêtes de chaque paquet.

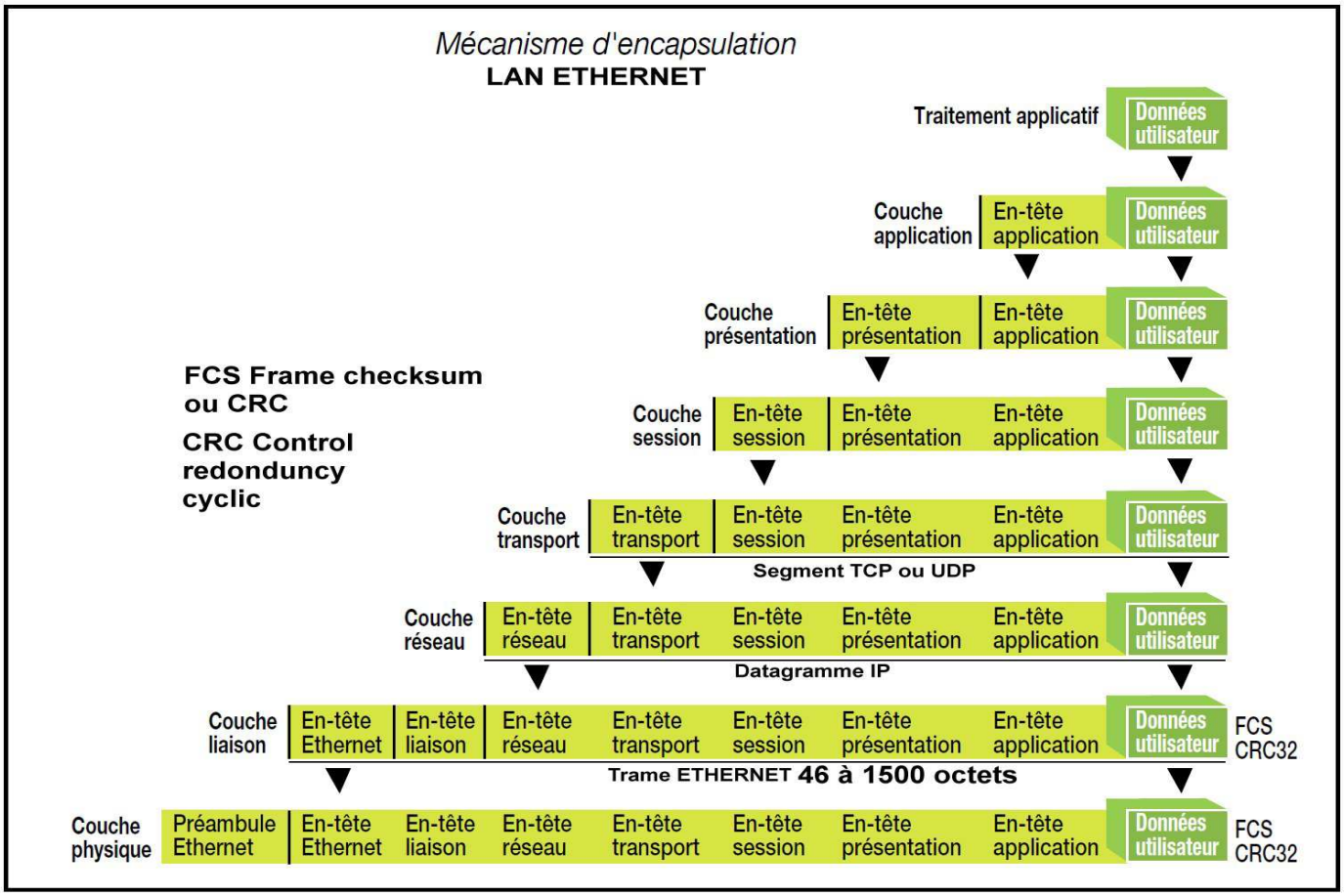
3.2) L'encapsulation

Le nombre d'en tête avant les données sont beaucoup plus conséquent car
Pour réaliser une communication sur un réseau il faut :

- ✓ relier les systèmes par un lien physique (couche PHYSIQUE) ;
- ✓ contrôler qu'une liaison peut être correctement établie sur ce lien (couche LIAISON) **correction** d'éventuelles des **erreurs** de transmission ;
- ✓ s'assurer qu'à travers le relais (réseau) les données sont correctement acheminées et délivrées au bon destinataire (couche RÉSEAU) **protocole IP** ;
- ✓ –contrôler, avant de délivrer les données à l'application que le transport s'est réalisé correctement de bout en bout (couche TRANSPORT) et assembler et réassembler les paquets **protocole TCP ou UDP ;**
- ✓ –organiser le dialogue entre toutes les applications, en gérant des sessions d'échange (couche SESSION) ;
- ✓ traduire les données selon une syntaxe de présentation aux applications pour que celles-ci soient compréhensibles par les deux entités d'application (couche PRÉSENTATION) ;
- ✓ –fournir à l'application utilisateur tous les mécanismes nécessaires à masquer à celle-ci les contraintes de transmission (couche APPLICATION).

La couche de liens est l'interface avec le réseau et est constitué d'un driver du système d'exploitation et d'une carte interface d'ordinateur avec le réseau (le plus souvent une carte ethernet)

L'encapsulation consiste pour chaque couche **à ajouter de l'information aux données** en les commençant par des en têtes.
On donne ci-dessous un exemple d'encapsulation pour un réseau dont la couche physique est du type ETHERNET



la couche réseau ou couche IP gère la circulation des paquets à travers le réseau

la couche transport assure une communication de bout en bout en régulant le flux de données et en assurant un transport fiable des données.

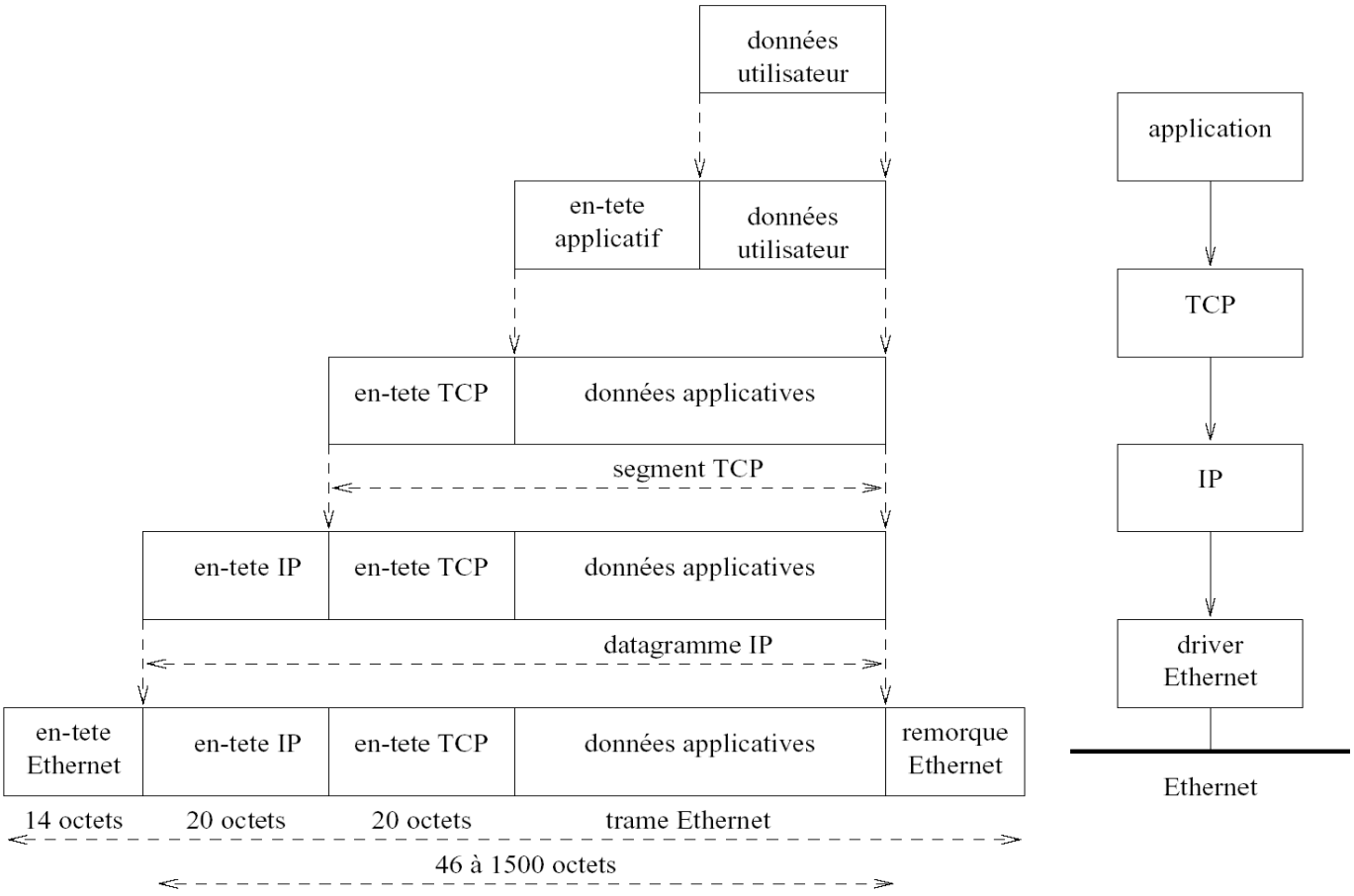
Le protocole TCP (transmission control protocol) assure un transport fiable des données.

L'UDP ne garantit pas cette fiabilité (user datagram protocol) c'est la couche application qui doit s'en assurer.

La couche application est celle des programmes utilisateurs (FTP (file transfert protocol), smtp (simple mail transfert protocol), http)
Lorsqu'une application envoie des données à l'aide de TCP-IP les données traversent de haut en bas chaque couche jusqu'à aboutir au support physique ou elles sont émises sous formes d'une suite de bits.

La couche de liens est l'interface avec le réseau et est constitué d'un driver du système d'exploitation et d'une carte interface d'ordinateur avec le réseau (le plus souvent une carte ethernet)

L'encapsulation consiste pour chaque couche **à ajouter de l'information aux données** en les commençant par des en têtes.
On donne ci-dessous un exemple d'encapsulation pour un réseau dont la couche physique est du type ETHERNET



la couche réseau ou couche IP gère la circulation des paquets à travers le réseau

la couche transport assure une communication de bout en bout en régulant le flux de données et en assurant un transport fiable des données.

Le protocole TCP (transmission control protocol) assure un transport fiable des données.

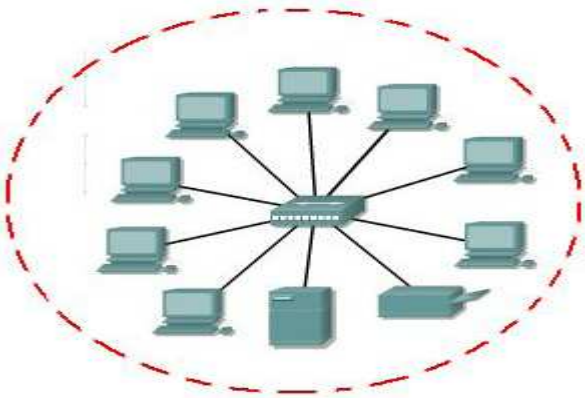
L'UDP ne garantit pas cette fiabilité (user datagram protocol) c'est la couche application qui doit s'en assurer.

La couche application est celle des programmes utilisateurs (FTP (file transfert protocol), smtp (simple mail transfert protocol), http)
Lorsqu'une application envoie des données à l'aide de TCP-IP les données traversent de haut en bas chaque couche jusqu'à aboutir au support physique ou elles sont émises sous formes d'une suite de bits.

4) L'interconnexion des réseaux IP du type LAN

4.1) Connexion d'un seul LAN

Dans ce cas précis étant donné le peu de machines présentes on utilisera un switche ou un HUB en étoile ou en anneau

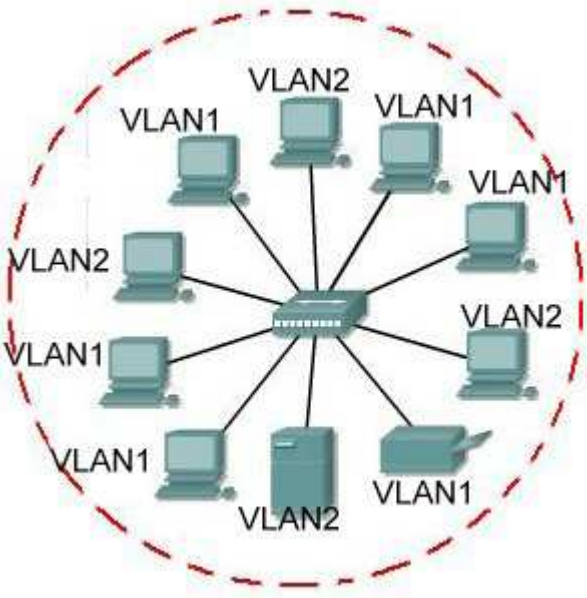


4.2) L'interconnexion de plusieurs LAN

Lorsqu'il ya trop de machines, pour éviter les problèmes **de congestions** (trop d'abonnés sur le réseau) **ou de collisions** (plusieurs abonnés qui parlent en même temps), **on doit segmenter** les réseaux. Cette segmentation peut se faire sur la couche 2 en utilisant des switches ou sur la couche 3 par l'intermédiaire des routeurs en créant des réseaux ou des sous-réseaux. Ils existent trois possibilités :

4.2.1) Les VLAN (réseau locaux virtuels)

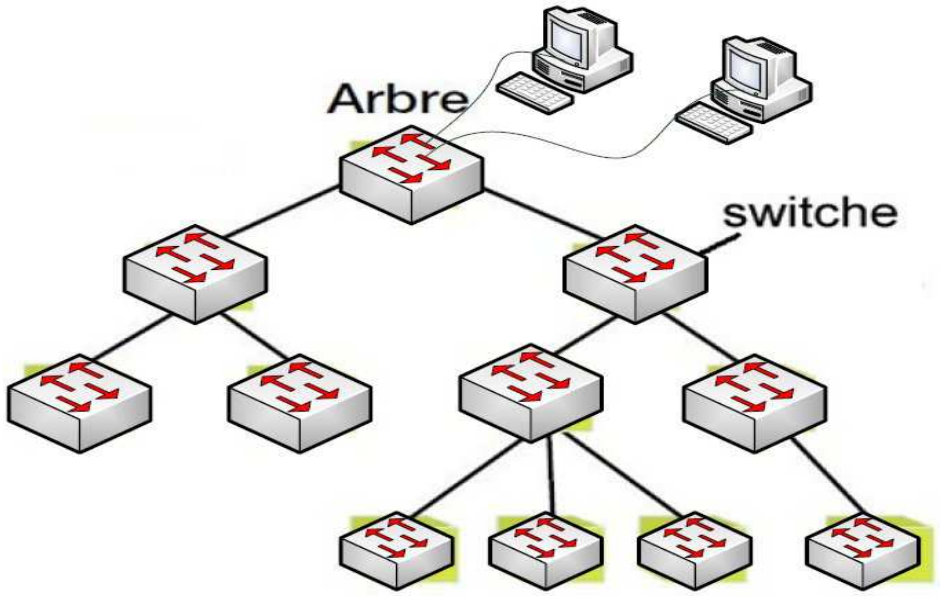
Utilisation d'un seul switche en utilisant des réseaux locaux virtuels (VLAN)



Les éléments du VLAN1 ne seront visibles qu'entre eux et même principe pour le VLAN2

4.2.2) La topologie en arbre

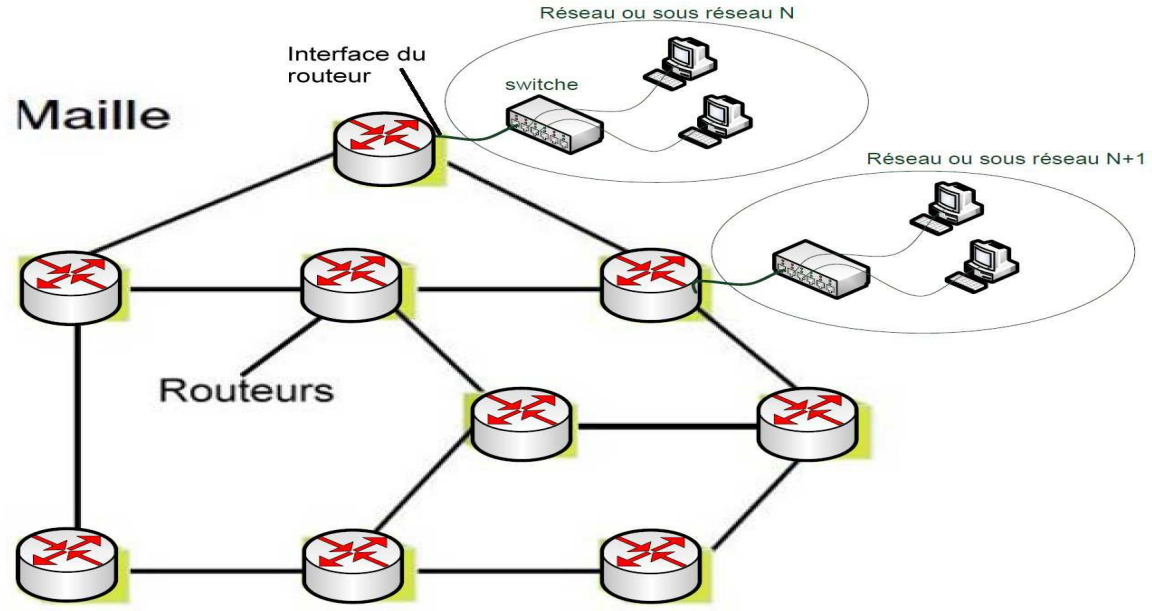
On va cascader les switches entre eux de façon à avoir une topologie en arbre



Si le nombre de machine est trop important on utilisera les routeurs avec une topologie maillée

4.2.3) Les routeurs en topologie maillée

Chaque réseau ou sous réseau branché sur une interface différente des routeurs doivent avoir pour ses machines des adresse IP de réseau ou de sous réseau différentes



Partie 1 :La couche liaison et physique ETHERNET

1) Présentation

ETHERNET a été développé **par Xerox** Corporation au Palo Alto Center (PARC) vers le milieu des **années 70**. Le principe est donc de mettre un support physique en commun, et de faire du très haut débit sur des distances moyennes (>100m).
On utilise un câble commun pour relier des dizaines voire des centaines de machines. Ce câble commun va véhiculer les informations à destination de l'ensemble des stations, la méthode utilisée est le **CSMA/CD (Carrier Sense Multiple Access / Colision Detection)**.
Le Câble forme un BUS dans le jargon réseau, reliant les stations. La vitesse est fixée par la norme : 10 Mbs. (10 Millions de bits par seconde). Un bit est une valeur binaire : 0 ou 1.
Des prix : début 80 une carte ETHERNET vaut 10000Fr, maintenant 150Fr.
La notation IEEE802.3 :
10Base5 10=10Mbs Base=Bande de Base 5 = 5*100mètres ex :

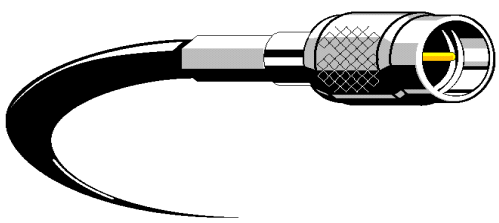
Nom	10 Base 5	10 base2	1 Base5	10BaseT (1985)	10Broad 36
Vitesse Mbps	10	10	1	10	10
Signal	Baseband	Baseband	Baseband	Baseband	Broadband
Longueur Max	500	185	250	100	1800
Media	50 Ohm coax (thick)	50 Ohm coax (thin)	Unshielded Twisted Pair	UTP	75 ohm coax
Topologie	Bus	Bus	Bus	Etoile	Bus

2) Les JONCTIONS et la connectique

10Base2

Un câble coaxial fin avec des connecteurs en **T**. Facile a mettre en place. Par contre les connecteurs affaiblissent le signal , du coup on ne peut mettre que 30 stations sur le câble. Tend à être remplacé par 10BaseT. Un problème sur le câble et 30 stations en panne contrairement à 10BaseT.

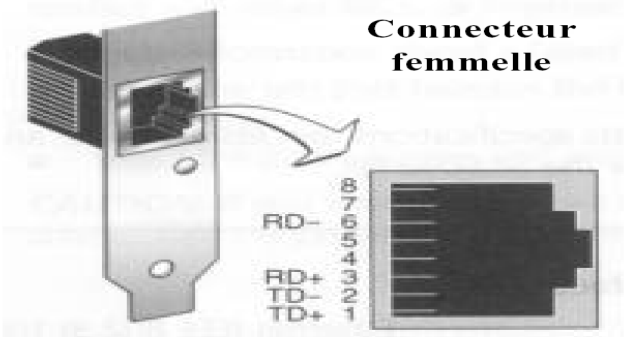
Connecteur BNC

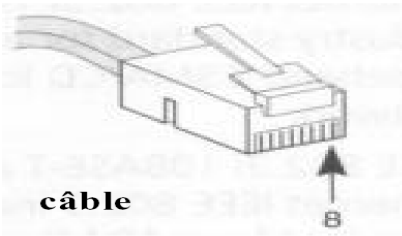


10 Base T :

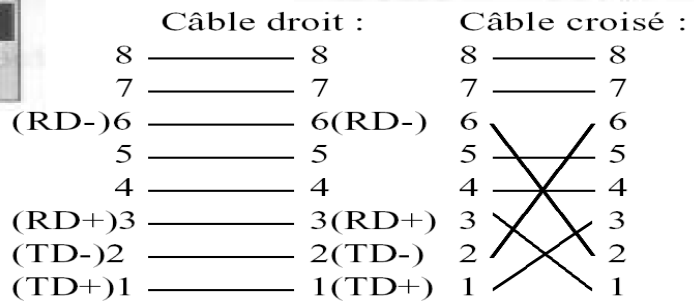
Le support est constitué de 2 paires de fils torsadés (twisted pairs), prolongés par des connecteurs d 'extrémité appelés **RJ45**. Ces câbles vont dans des appareils appelés HUB qui connectent les machines. Il existe des HUB 8 ports 12 /16/24 ports. On ne doit pas connecter par des câbles, plus de 3 Hubs. Les câbles de connections qui les relient sont des câbles croisés. Voir schéma du cours. En 85, la porte sur le Hub valait 2000Fr , maintenant 300 à 400 Fr. Les Hub peuvent être cascadés en local avec des câbles propriétaires. Ils ne forment alors qu'un seul ensemble. Dés qu'ils sont éloignés , il faut des câbles croisés. Les machines ne doivent pas être à plus de 100 mètres du Hub. Idem pour les Hubs entre eux.

Connecteurs RJ45





• Un câble droit connecte un terminal au hub ;
• Un câble croisé interconnecte 2 hubs



100 Base T

En fait , en gardant le principe de ETHERNET, on transmet à 100 Mbs. Ceci ne peut marcher que sur un réseau qui ne fait que du 100BaseT. Ce sont donc des Hubs particuliers qui utilisent les câbles habituels du 10BaseT, toutefois les connecteurs d'extrémité sont blindés.
Pour avoir à la fois du 100Mbs et du 10Mbs sur le même réseau, il faut interconnecter avec des switchs. Un switch vaut au moins 1000Fr la porte (200Fr pour un Hub), par contre la carte ETHERNET 100Mbs est à peine plus chère que son ancêtre.

***GIGA* Bit Ethernet**

Le concurrent de l'ATM pour les hauts débits. Même principe mais la vitesse est de 1Gigabit/sec. Le prix des cartes et des liens 1Gbs étant assez bon marché, le Gbs risque de faire une sérieuse concurrence à l'ATM.
Le Gigabit Ethernet utilise en 1999 uniquement les fibres optiques. Cependant, une normalisation sur paire métalliques va avoir lieu. Le cablage recommandé devra suivre les spécification 5E. Il faudra faire très attention à avoir les mêmes types de cable (impédance). Curieusement, le Gigabit sur cuivre sera un protocole de transport parallèle qui utilise les 8 fils, 4 en émission et 4 en réception. Les émissions étant à 250 Mbs sur chaque fil. La transmission se fait à 100Mhz par codage des informations.

3) Les différens types de trame ETHERNET

Deux types de protocole (couche réseau) seront abordés :

Les trames envoyées sur le réseau auront un format différent suivant les principaux protocoles :

ETHERNET II

Source	Destination	Type	46 à 1500 octets	FCS

ETHERNET 802.3

Source	Destination	Longueur	46 à 1500 octets	FCS

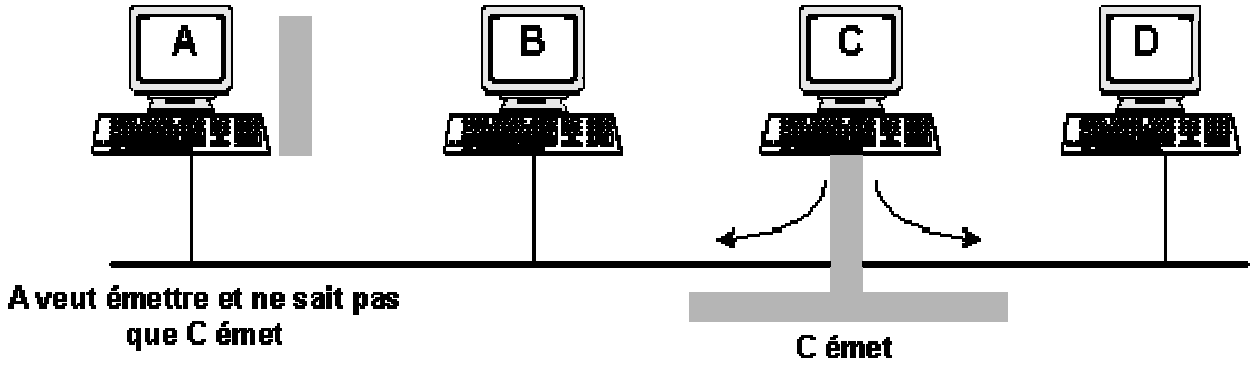
Source et destination sont des adresses MAC de l'expediteur et du destinataire
C'est une adresse écrite en "dur" dans le "firmware" d'un équipement réseau, le plus souvent une interface réseau.
Cette adresse est définie sur 6 octets.

- Les trois premiers (les plus à gauche) sont attribués au constructeur.
- Les trois derniers sont spécifiques à un équipement matériel donné.

Au total, une adresse MAC est sensée être unique au monde.
Son but est d'identifier sans aucune ambiguïté possible un nœud sur un réseau. Elle est utilisée par le niveau 2 du modèle OSI pour l'acheminement des données d'une source vers une cible.
Il faut bien comprendre que cette adresse est indispensable, parce qu'elle est la seule qui soit définie à la mise en route d'un système, puisqu'elle réside dans une ROM. D'ailleurs, certains protocoles réseaux simples se contentent de cette adresse pour fonctionner. NetBEUI en est un exemple. De plus, au niveau 2 du modèle OSI, c'est la seule adresse en mesure d'être utilisée.
Toute autre adresse qui sera ajoutée avec l'installation du système sera une adresse plus évoluée, destinée à gérer les réseaux de façon logique, mais l'adresse MAC demeure indispensable.

4) l'accès au support

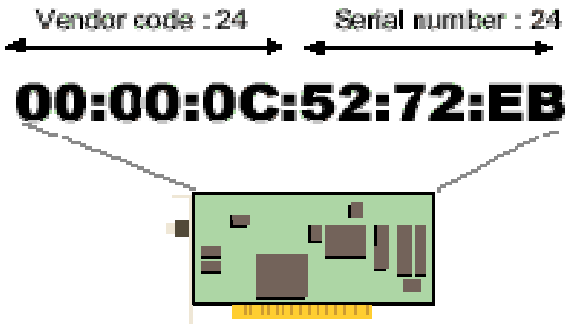
Ethernet utilise le protocole **CSMA/CD (Carrier Sense Multiple Access / Collision Detection)** pour accéder au support. C'est un protocole **non-déterministe**, c'est à dire que nous ne pouvons déterminer le temps maximum que peut mettre une station pour émettre. Dans le cas de trafic de plus en plus important les délais d'attente augmentent. De plus, il n'y a **pas de priorité entre les stations**.. Lorsque 2 stations émettent au même instants il y a une **collision.**
Dans le protocole CSMA, on va écouter le support pour savoir s'il y a une porteuse (Carrier Sense) avant d'émettre et si personne n'émet, on émet. Cela n'élimine pas les collisions mais réduit leur probabilité. En effet, soit une station A qui veut émettre, elle écoute le support à son point de raccordement. Une autre station B peut être en train d'émettre mais le signal n'est pas arrivé encore au point de raccordement de A.



Pendant toute l'émission, la station émettrice continue à écouter le support pour détecter s'il y a collision (CD). S'il y a une collision, une trame spécifique de brouillage (Jam) est émise puis la station recommence le processus d'écoute CSMA après un temps aléatoire tiré dans un intervalle déterminé. S'il y a de nouveau collision, l'intervalle de temps pour le tirage aléatoire est doublé. On double l'intervalle jusqu'à la 10ème collision. On effectue toutefois 16 tentatives d'émission. Si au bout de la 16ème tentative, il y a toujours des collisions, on avertit la couche supérieure de l'impossibilité d'émettre.

5) la réception

Chaque station dispose d'une adresse physique Ethernet inscrite dans la ROM de la carte. Cette adresse est affectée par le constructeur de la carte.



L'adresse destinataire est comparée avec l'adresse de la carte dans le cas d'une adresse **unicast (adresse individuelle)**. S'il y a correspondance alors le récepteur conserve la trame et passe l'information à la couche au dessus sinon la trame est détruite. Dans le cas d'une adresse **multicast (de groupe)**, le récepteur détermine s'il appartient au groupe auquel la trame est destinée et dans l'affirmative conserve la trame. Si l'adresse est l'adresse de diffusion d'Ethernet (tous les bits sont à 1) alors le récepteur conserve la trame.

Partie 2 : La couche réseau

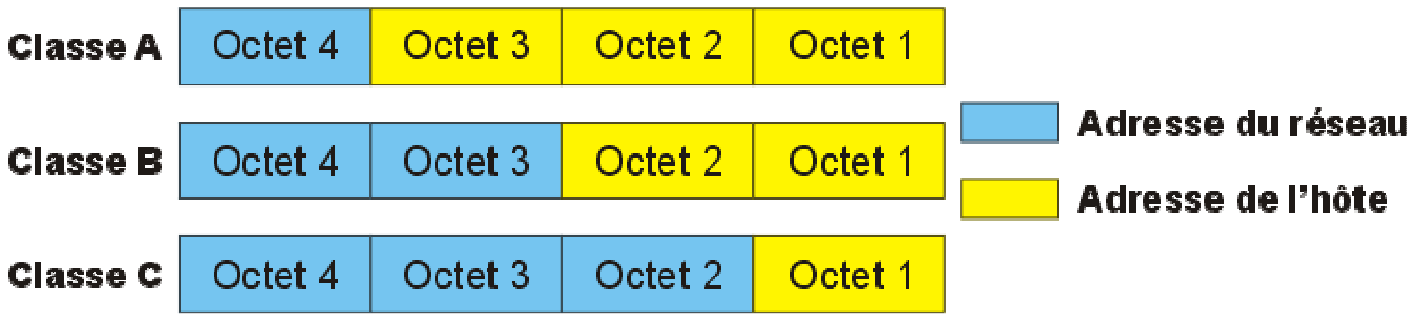
Dans le chapitre précédent, nous avons eu l'occasion de voir comment un réseau est construit et quels protocoles de bas niveau (niveau 1 du modèle OSI) sont employés pour transporter ses données "brutes" sur le réseau.
Une bonne compréhension de TCP/IP est nécessaire si l'on souhaite d'une part savoir comment les données transitent sur les réseaux.
la couche réseau ou couche IP gère la circulation des paquets à travers le réseau grâce à des adressesIP

1) l'adressage logique

Un PC est relié au réseau via une carte d'interface dite "carte réseau" laquelle possède une adresse physique unique attribuée lors de la fabrication par le fabricant de la carte. Ainsi la carte réseau d'une des machines sur laquelle ce que vous êtes en train de lire a été écrit possède l'adresse physique hexadécimale C0B35CDD (adresse MAC (media access control)), adresse qui est utilisée par le réseau local du laboratoire. Dans ce cas la liaison Ethernet est directe entre machines et chacune d'entre elles, via sa carte réseau, va examiner la totalité du trafic sur le réseau (coaxial) pour récupérer simplement ce qui lui est destiné. A l'échelle d'Internet cette procédure est évidemment *inimaginable*. On va donc **subdiviser Internet en sous-réseaux hiérarchisés** et grâce à l'adressage logique élaboré par le logiciel de communication il sera possible de s'y retrouver. Sous TCP/**IP l'adresse logique d'une machine est ce qu'on appelle une adresse IP** qui va inclure un numéro ID d'identification de votre réseau (par ex.194.254), un numéro d'identification de sous-réseau du réseau (par ex. 17) et un numéro de machine identifiant la machine sur le réseau (par ex. 200) d'où une adresse IP complète 194.254.17.200.

2) Les classes d'adresses IPV4.

Chaque ordinateur du réseau internet dispose d'une adresse IP unique **codée sur 32 bits** (norme IPV4) IP définit donc une adresse sur 4 octets. Une partie définit l'adresse du réseau l'autre partie définit l'adresse de l'hôte dans le réseau ou le sous réseau. La taille relative de chaque partie varie suivant la classe choisie.

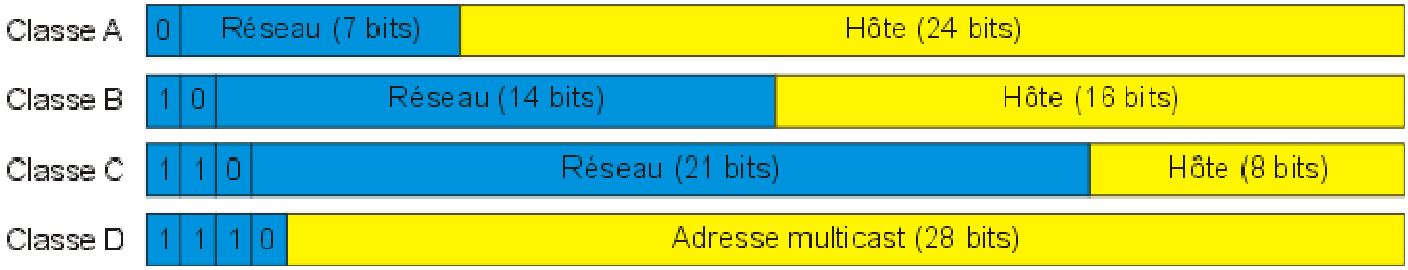


Comme vous le voyez, la classe A permet de créer peu de réseaux, mais avec beaucoup d'hôtes dans chaque réseau, La classe C faisant l'inverse.

Les adresses de classe A sont utilisées par les grands réseaux qui comportent plus de 65536 ordinateurs (défense américaine)

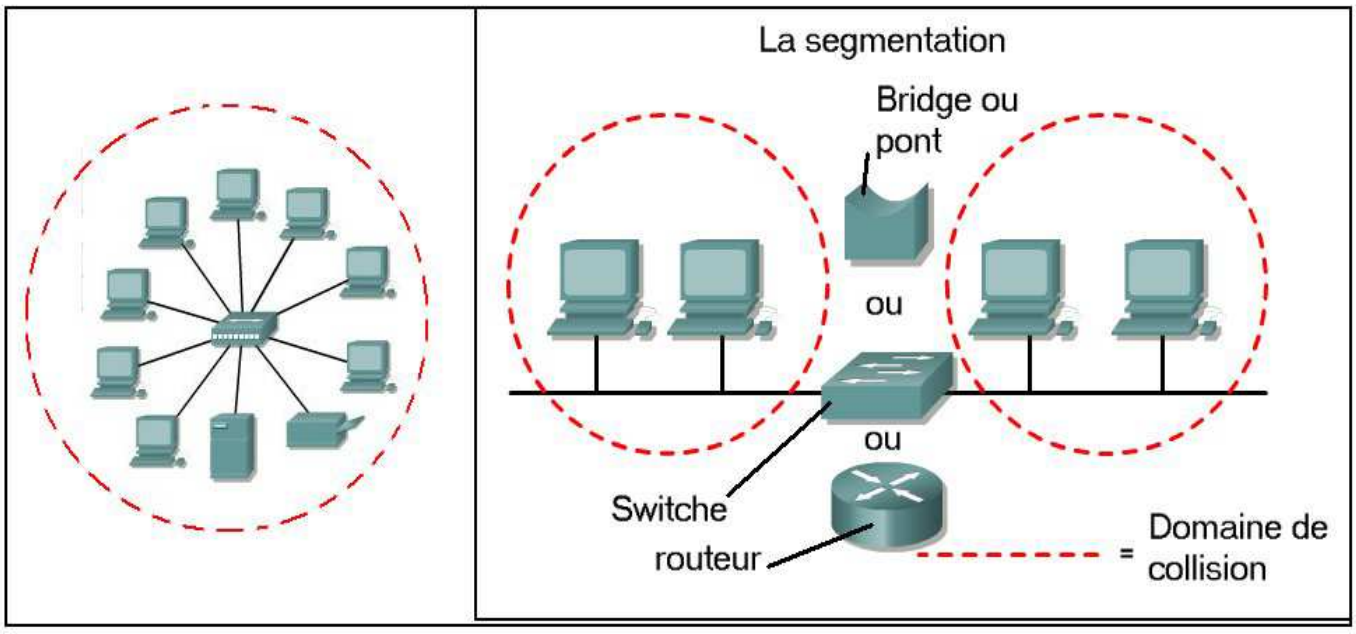
L'adresse de l'hote va permettre de déterminer :
- la machine dans un sous réseau
- un sous réseau particulier

La classe est définie par les bits les plus forts (les plus à gauche)



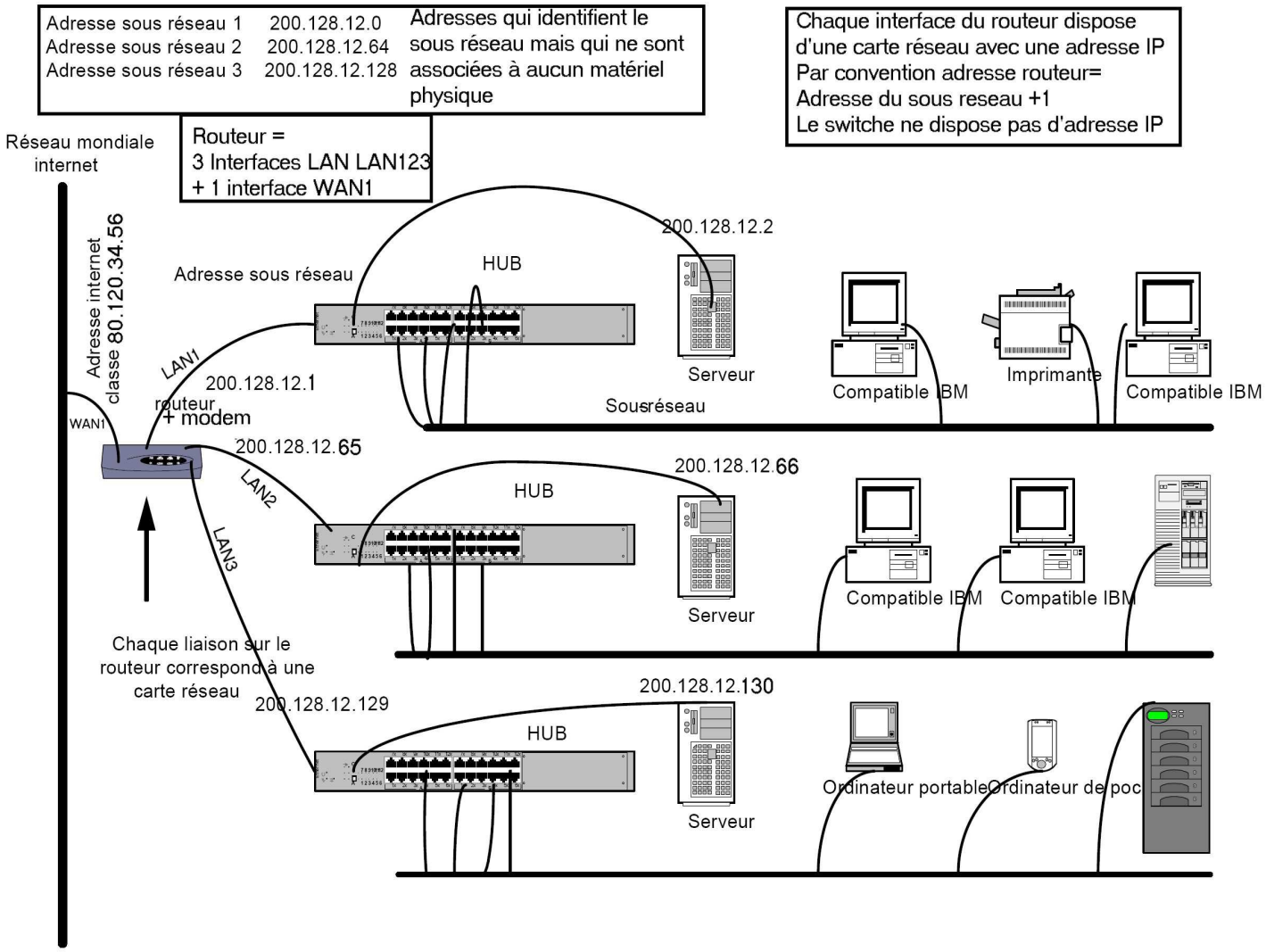
3) La segmentation d'un réseau

Le principe est de segmenter les réseaux afin de limiter les domaines de collision. Cette segmentation peut se faire sur la couche 2 en utilisant des ponts ou des switches ou sur la couche 3 par l'intermédiaire des routeurs en créant des réseaux ou des sous-réseaux



Ces plus petits domaines de collision vont permettre d'augmenter le trafic.

3.1) Le SUBNETTING



a) Vérifiez si ce réseau est de type classe C ?
b) Combien de machines peut on brancher par sous réseau ?
c) Si on effectue un partage linéaire des adresses IP combien de sous réseau peut on créer ?

3.2) Le masque de sous réseau.

Le masque de sous-réseau est composé de 4 octets. Il permet au routeur ou au machines **de déterminer** le **nombre de sous réseau** ainsi que **le nombre de machine par sous réseau** (le routeur peut être un serveur dans le cas d'un réseau intranet). Il aiguille ainsi l'information vers la bonne machine dans le bon sous réseau.

Grâce à ce masque on extrait :
- l'adresse de réseau et de sous réseau en effectuant un ET logique bit à bit entre l'adresse IP et le masque.
- l'adresse de la machine ou de l'hôte en effectuant un ET logique bit à bit entre l'adresse IP et le complément du masque (!masque).

Soit le masque de sous réseaux suivant :
1111 1111 . 1111 1111 . 1111 1000 . 0000 0000

Soit l'adresse IP suivante 62.161.99.115 est-ce une adresse réseau ou l'adresse d'un sous réseau ?

Donnez la partie de l'adresse qui concerne l'hôte ou le host id ?

Quelle est la plus petite adresse machine possible dans ce sous réseau ?

Quelle est la plus grande adresse possible dans ce sous réseau ?

3.3) Les points clés à retenir

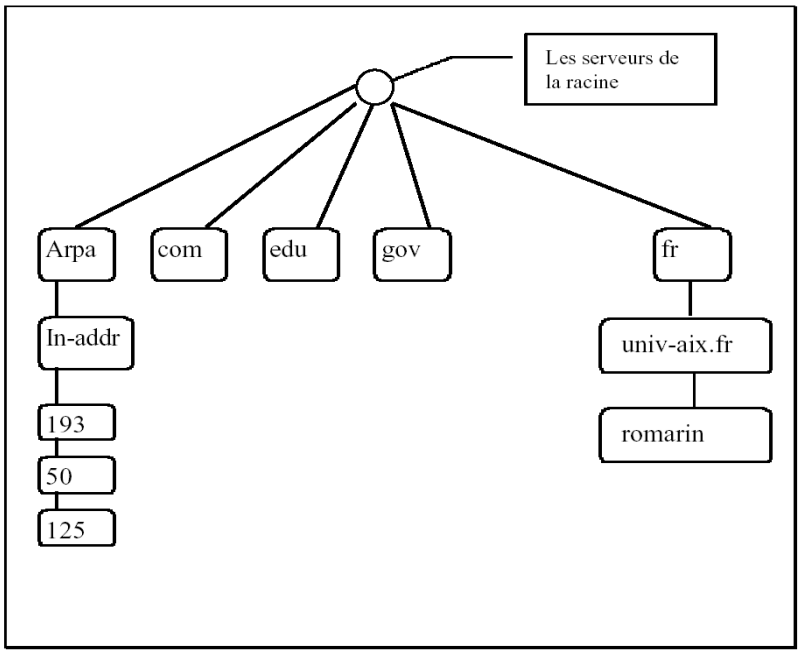
Le masque de sous réseau est programmé pour chacun des éléments du réseau (sauf le switche) car :
Le routeur se sert du masque de sous réseau pour déterminer l'adresse de sous réseau et diriger l'information vers le sous réseau auquel elle est destinée.
Un ordinateur A d'un sous réseau qui veut envoyer une information vers un autre ordinateur B a besoin de connaitre grâce au masque de sous réseau si l'ordinateur B se situe dans son sous réseau ou un autre réseau ou sous réseau.
Il faut retenir :

- Qu'on ne doit pas donner à une machine une adresse dont **la partie machine de l'adresse IP est nulle**. Elle est réservée à l'adresse réseau ou du sous réseau.
- Que la première adresse IP du sous réseau est par convention l'adresse IP de l'interface du routeur pour ce sous réseau.
- Qu'une adresse IP dont la partie machine ne contient que des '1' logiques est l'adresse de diffusion générale sur le sous –réseau

Pour un sous réseau 3 adresses IP sont donc intrinsèquement prises :
L'adresse IP de l'interface routeur
L'adresse IP du réseau ou du sous réseau (bits machines à 0)
L'adresse IP de diffusion générale sur le sous réseau (bits machines à 1)

4) la résolution d'adresses et les serveurs DNS

TCP/IP offre une structuration des adresses en langage clair, c'est à dire la possibilité de faire **correspondre une adresse numérique telle 194.254.17.200 avec un nom beaucoup plus convivial** via un service gérant ce qu'on appelle les noms de domaine (DNS) ainsi sera-t-il beaucoup plus facile de s'appeler toto.fr que la suite numérique précédente.
Les noms sont composés par une suite de caractères alphanumériques encadrés par des points36. Par ex romarin.univ-aix.fr correspond à l'adresse 193.50.125.2 et le mécanisme qui associe le nom au numéro s'appelle la résolution de noms. Cette représentation est hiérarchique.
Les serveurs qui traitent la conversion nom = adresse ou adresse = nom sont des serveurs de nom ou DNS



Domaine	Description
com	Organisations commerciales (hp.com)
edu	Institutions éducatives (américaines)
gov	Organisations gouvernementales US
int	Organisations internationales
mil	Militaires Us
net	Réseau
org	Organisation à but non lucratif
de	Allemagne
uk	Angleterre
fr	La France

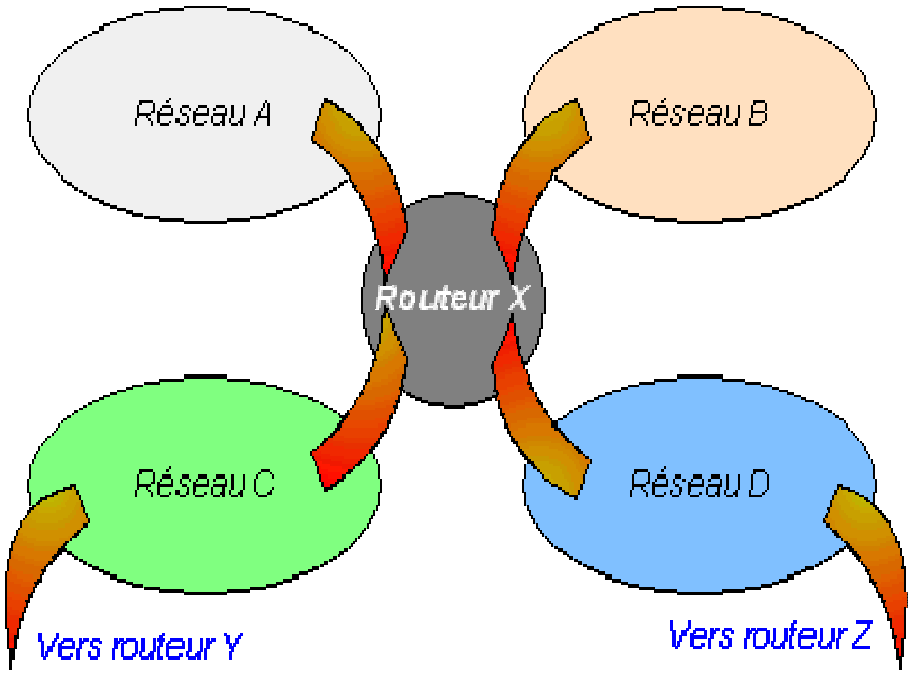
Comment ça marche ?.
Une organisation : le NIC (Network Information Center) a en charge la bonne marche des DNS et délègue son autorité sur des sous domaines. En France, l'autorité responsable est l'INRIA qui gère le domaine fr.
Quelques une des machines de l'INRIA sont les serveurs du domaine fr. (http://www.nic.fr)

Quant une application (TELNET, web..) a besoin de résoudre une adresse symbolique, elle va utiliser un renseignement de la configuration de la machine. Sous Unix, il s'agit du fichier /etc/resolv.conf. Dans ce fichier, on va trouver l'adresse de un ou 2 serveurs de noms. On envoie une requête UDP sur le port 53 du serveur de noms en demandant la résolution. Celui-ci va alors appeler une série de serveurs pour la résolution.

5) Le Routage des Datagrammes IP

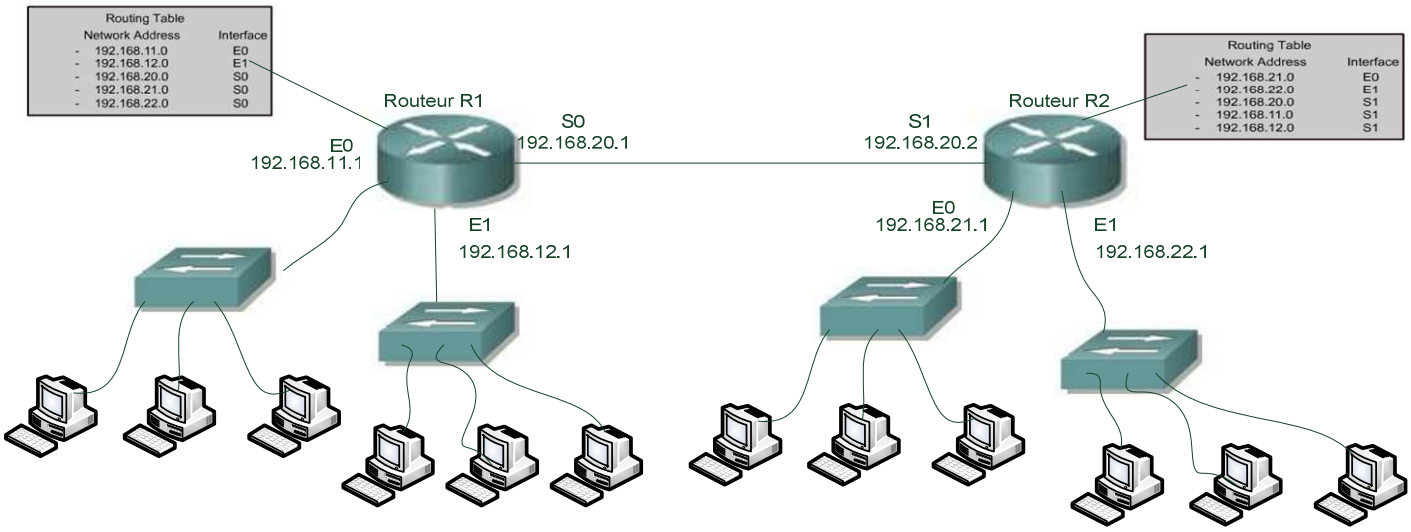
5.1) Présentation

Le routage est l'opération d'acheminer les paquets à bonne destination. Les machines effectuant cette opération sont appelées **routeurs** ou **passerelles**. Dans la terminologie Anglo-saxonne, on parle de « router ». Un routeur est souvent une machine spécialisée et sans disque dur (fiabilité). Cependant une station Unix ou un Windows SERVER peuvent faire le travail.
Le routeur va donc être un dispositif qui lit les adresses logiques et va ensuite orienter les données vers leur destinataire.

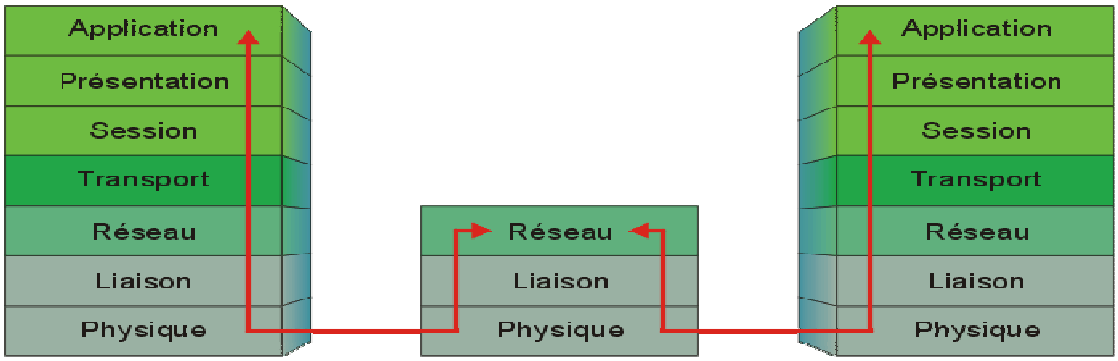


Les routeurs ont plusieurs fonctions: contrôler le trafic, **assurer la connexion** entre réseaux de caractéristiques physiques différentes (ex réseau de type Ethernet et réseau en anneau à jeton), **assurer l'adressage hiérarchique et enfin la régénération des signaux**.
. Ainsi quand un routeur reçoit un message il va identifier quel en est le destinataire, consulter sa table de routage pour identifier le chemin et enfin transmettre ce message via sa carte réseau ad hoc.

Table de routage



Un routeur agit au niveau de la couche réseau.
Le routeur ne lit que l'en-tête IP, Il ignore (à priori) tout du contenu du paquet.
Les 4 réseaux raccordés restent 4 réseaux physiques différents, les adresses MAC d'un côté restent totalement inconnues de l'autre côté



5.1) les tables de routage statiques

Les tables de routage statiques sont configurées manuellement par un opérateur
Le routage statique est adéquat si :
- le réseau est de petite taille
- il ne comporte que quelques points de connexion vers les autres réseaux
- La topologie du réseau change

Dans l'exemple précédent si l'on veut atteindre le réseau 192.168.22.0 à partir du routeur R1, il faudra tapez pour un routeur CISCO la ligne de commande suivante :

ip route 192.168.22.0 255.255.255.0 192.168.20.2

- Réseau ou Sous réseau Que l'on veut atteindre
- Nombre de machines Et de sous réseaux
- Adresse de l'interface du routeur ou l'on souhaite passer

5.2) Les tables de routage dynamiques

Les tables de routage sont mises à jour automatiquement au moyen de protocoles plus évolués (routage dynamique qui consiste en un protocole de communication entre routeurs qui informent chacun de leurs voisins des réseaux auxquels ils sont connecté).
Ils existent plusieurs protocoles de routage dynamique RIP, OSPF …
RIP est le protocole de routage le plus largement utilisé, le plus simple à mettre en œuvre, et souvent utilisé par défaut dans les routeurs.

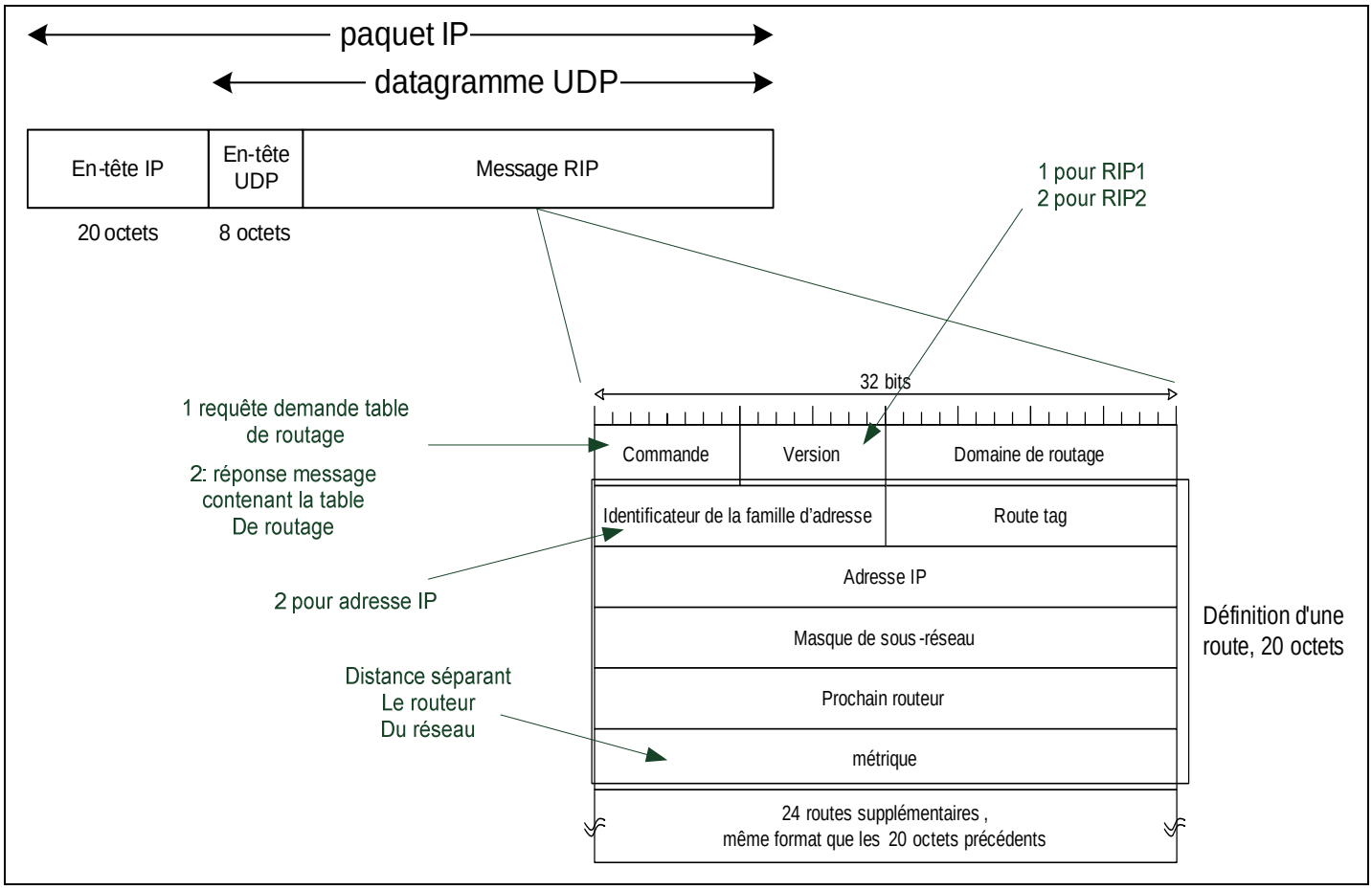
Le protocole de routage RIP (Routing Information Protocol)

a) *Présentation*

RIP est un protocole de routage de type « distance-vector », consistant à transmettre à ses voisins un vecteur de distance (compteur de sauts) associé à une destination. Il est conçu pour travailler sur des réseaux de petite taille.

b) *Format du message*

RIP utilise le protocole UDP pour transporter ses données.
Le port 520 lui est réservé.



c) Principe

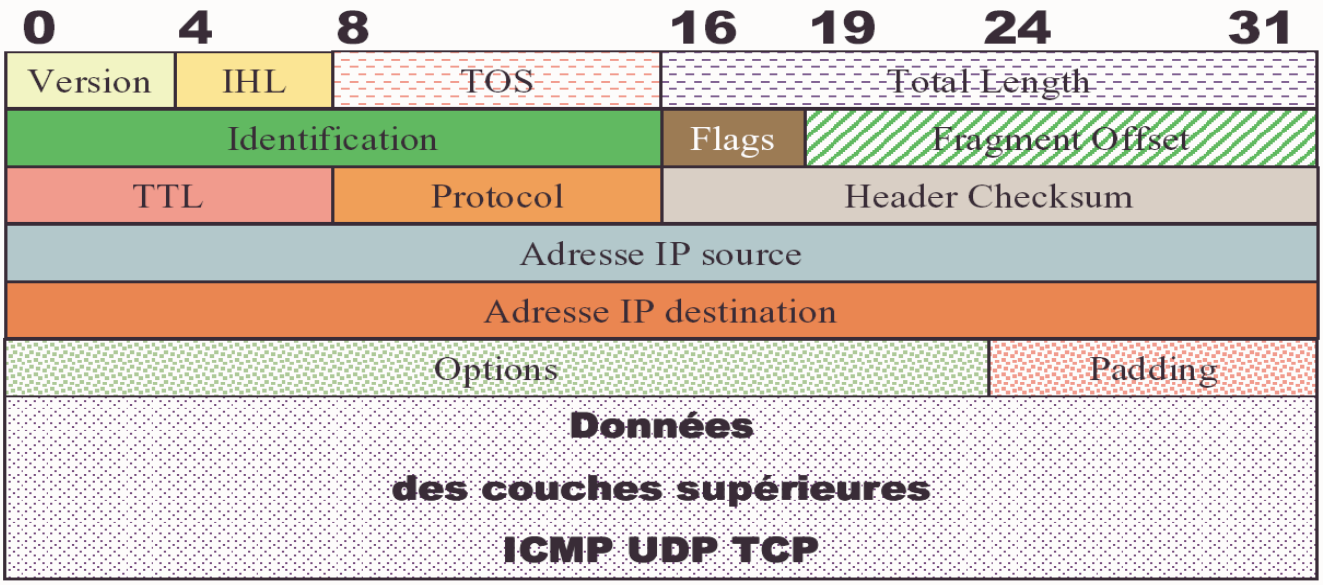
Au démarrage du routeur, le programme résident (démon) RIP détecte toutes les interfaces actives, et ainsi crée la table de routage des routes directes.
Il envoie alors une **requête** sur chacune des interfaces afin d'obtenir les tables de routages complètes des autres routeurs. Ce datagramme est un broadcast diffusé sur le port UDP 520 et configuré comme suit :
Command à 1
Metric à 16
Les réponses qui parviennent au routeur sont des listes de réseaux accessibles par les routeurs adjacents, avec une distance associée à chaque destination.
Lorsqu'une nouvelle destination est reçue, le routeur l'ajoute à sa table de routage, indiquant comme adresse de destination, l'adresse source à l'origine du message RIP et effectue une mise à jour anticipée

d) Inconvénient

- Limitation du métrique à 15.
- Pas d'optimisation des liens et donc absence de QoS (qualité de service).
- Les mises à jour ne sont pas incrémentales et consomment de la bande passante.
- La mise à jour des tables peut être lente à l'initialisation.
- Aucune confidentialité

6) Le datagramme IP

Un datagramme IP est constitué d'une en tête suivie d'un champ de données.



- **Version** : Indique la version du protocole, comme nous en sommes encore à IPv4, ce champ devra donc contenir une valeur fixée à 4.
- **IHL** : *Internet Header Lenght*, ce champ indique la longueur de l'entête, les entêtes IP contiennent toujours au moins 20 octets.
- **Lenght**, ce champ indique la longueur totale du paquet IP, comprenant donc l'entête et le payload.
- **TTL** : *Time To Live*, ce champ définit le temps de vie du datagramme après qu'il ait passé à travers un système. On l'initialise généralent à 255.
- **Protocole number** : Ce champ définit le numéro du protocole à employer en complément de IP. Les valeurs d'identifications sont :

1 : Pour le protocole ICMP (Internet Control Message Protocol).
2 : Pour le protocole IGMP (Internet Group Management Protocol).
6 : Pour le protocole TCP (Transmission control Protocol).
17 : Pour le protocole UDP (User Datagramm Protocol).

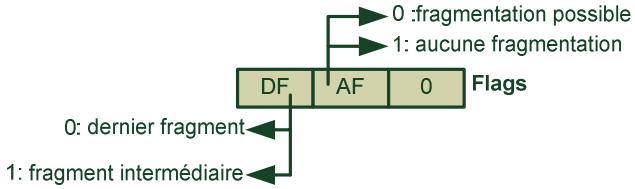
- **Checksum** : Le checksum est un petit supplèment qui permet de vérifier si le datagramme a correctement été forgé.
- **Source IP Adress** : L'adresse IP de l'emétteur, donc la votre.
- **Destination IP Adress** : L'adresse IP du récepteur.
- **Options** : Ce champ indique les options IP à employer, trop nombreuses pour être citées ici, désolé.
- **Padding** : Ce champ est un espace spécial, pour faire en sorte que l'entête IP ait bien une taille maximale de 32 octets (octets de bourrage).

a) La fragmentation

La fragmentation du datagramme Internet devient nécessaire dès lors qu'un datagramme de grande taille arrive sur une portion de réseau qui n'accepte la transmission que de paquets plus courts.
Les procédures de fragmentation et réassemblage Internet doivent pouvoir "casser" un datagramme Internet en un nombre de "fragments" arbitraire et quelconque pourvu que le réassemblage soit possible.
Le récepteur des fragments utilise **le champ d'identification** pour s'assurer que des fragments de plusieurs datagrammes ne puissent être mélangés. Le champ **"Fragment Offset"** indique au récepteur la position du fragment reçu dans le datagramme original.

Identification : 16 bits
Une valeur d'identification assignée par l'émetteur pour identifier les fragments d'un même datagramme.

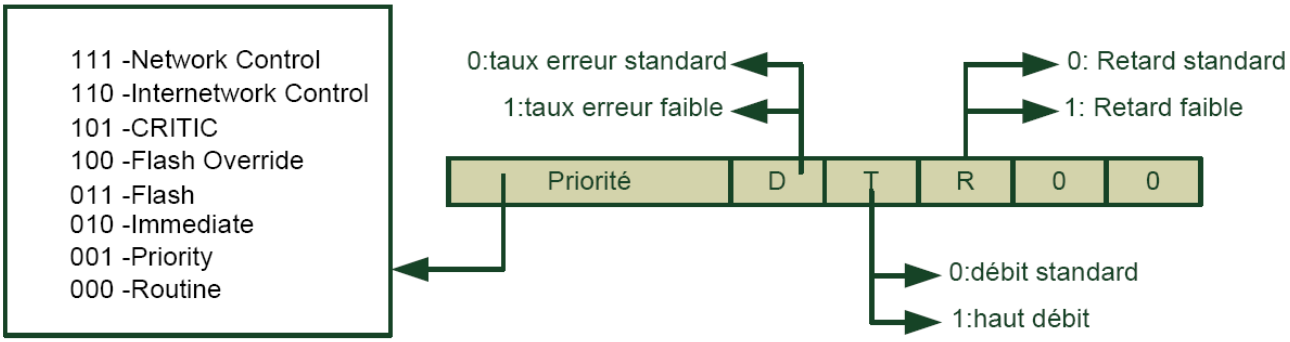
Flags : 3 bits



Fragment Offset : 13 bits
Ce champ indique le décalage du premier octet du fragment par rapport au datagramme complet. Cette position relative est mesurée en blocs de 8 octets (64 bits). Le décalage du premier fragment vaut zéro.

b) La qualité de service (QOS) :TOS

Le Type de Service donne une indication sur la qualité de service souhaitée, qui reste cependant un paramètre "abstrait". Ce paramètre est utilisé pour "guider" le choix des paramètres des services actuels lorsqu'un datagramme transite dans un réseau particulier.



Partie N°3 : la couche Transport

La couche transport **assure une communication de bout en bout en régulant le flux de données** et en **assurant un transport fiable des données**.

1) Le protocole TCP (transfert control protocol)

1.1) Le segment TCP

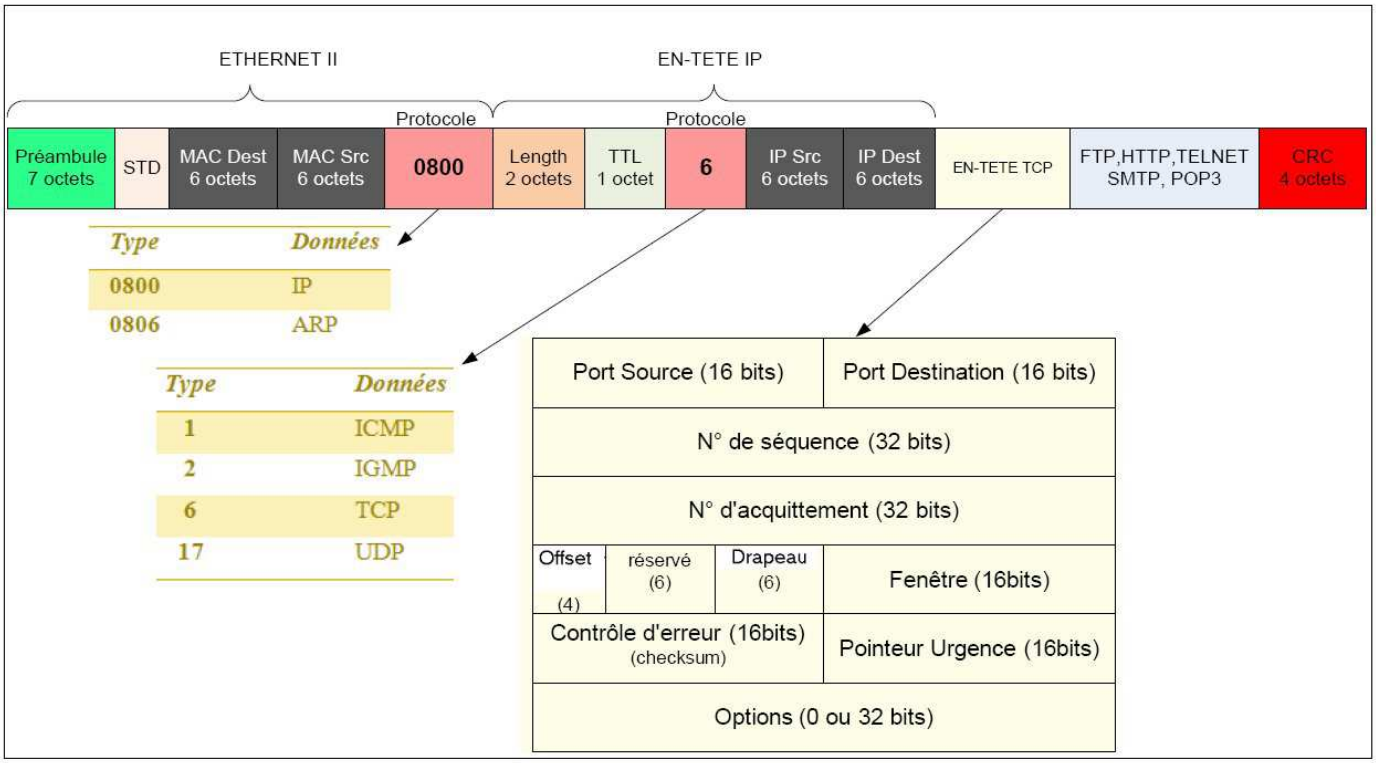
TCP est un **protocole connecté**. C'est à dire qu'il existe une phase de **création d'une connexion** où les deux machines négocient leurs options et réservent des ressources. TCP informe les applications **du succès ou de l'échec** et ensuite **contrôle le lien**.

· TCP va soit **découper, soit rassembler dans un paquet** suffisamment d'informations pour **minimiser** les transferts réseaux.

· Connexions Bidirectionnelles :

· Fiabilité des transferts et acquittements.

. Voilà donc à quoi ressemble un en tête de segment TCP :



trame Ethernet +TCP/IP

Le numéro du port source et celui du port de destination définissent les numéros de ports employés pour la communication. Chaque numéro de port est relié à un socket, un socket est une API (fonction) qui va permettre la communication entre deux processus. Cela peut être un service, exemple : ftp, comme une application.

Le numéro de séquence (Sequence number) et le numéro d'acquittement (Acknowledgment Number) identifient la position des données dans le flux de données envoyé, et la position des

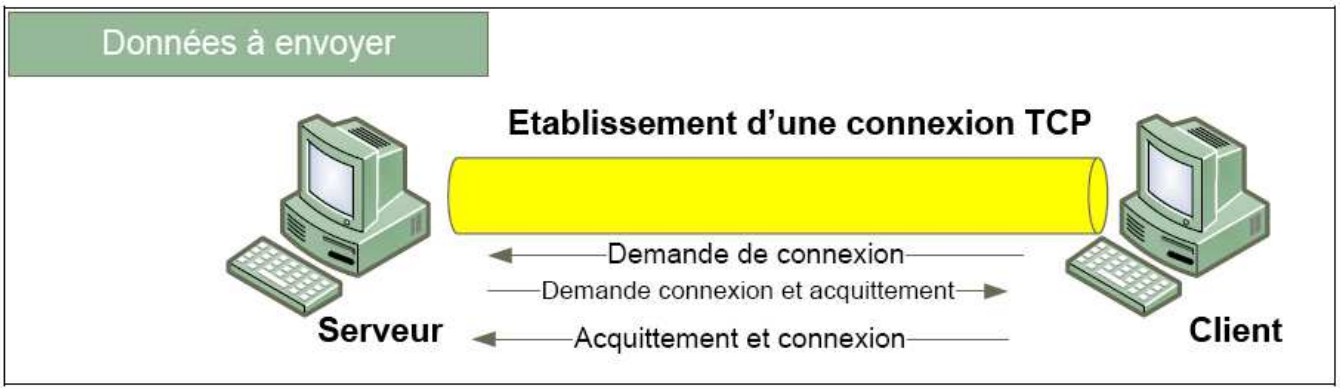
données dans le flux de données acquitté reçu correctement. L'acquittement d'une séquence d'octets jusqu'à un numéro N donné se fait en précisant dans le champ numéro d'acquittement la valeur N+1 : le numéro d'acquittement fait référence au prochain octet attendu.

Parlons maintenant du champ Flags (Drapeaux). Ces flags indiquent la nature du segment TCP. Citons les :

URG : Indique des données urgentes, sa valeur est 32
ACK : Indique un accusé de réception des données, indiquant que celles ci ont été correctement reçues, la valeur de ACK est 16.
PSH : Indique une poussée des données vers l'application, valeur : 8.
RST : Indique une remise à zéro de la connexion, sa valeur est 4.
SYN : Indique une synchronisation de la connexion, sa valeur est 2.
FIN : Indique une fin de transmission, sa valeur est 1.
TCP offre un mécanisme de transfert de données fiabilisées :

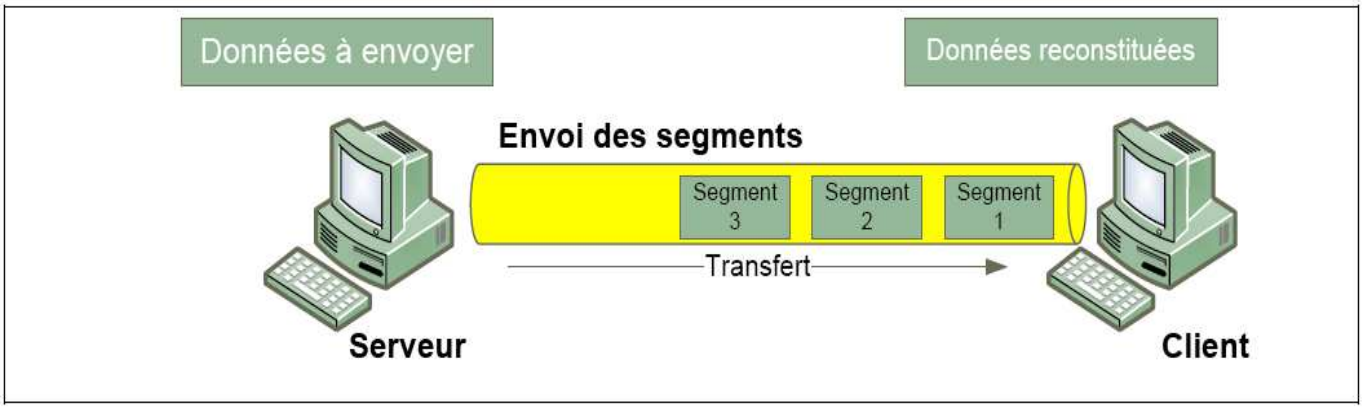
1.2) Principe du transfert de données entre 2 hôtes TCP

Le transfert de données entre 2 PC communiquant en **TCP/IP** utilise la notion de **client/serveur.** Un serveur est un programme lancé automatiquement au démarrage de la machine (ou par l'administrateur de la machine) et à l'écoute d'un **port** (notions de la couche 4) venant des trames TCP/IP reçues via une interface (Ethernet, Wifi ou bluetooth) de communication. Le protocole TCP utilise 3 phases de fonctionnement : la phase connexion, la phase de transfert et enfin la phase de déconnexion.
La phase de connexion : le client envoie une demande de connexion au serveur avec un numéro de ticket, le serveur acquitte ce numéro de ticket et envoie lui-même son numéro de ticket. Le client acquitte ce dernier, la connexion est faite entre les 2 PCs. Ce protocole dit de la double poignée de main permet d'être sur que les 2 hôtes sont prêts à communiquer.



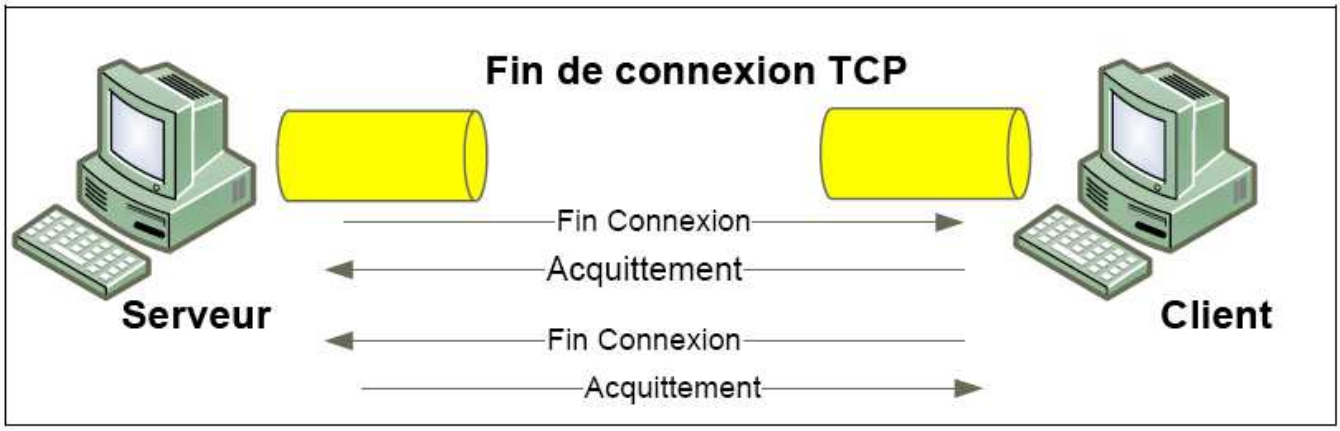
Etablissement de la connexion entre le client et le serveur

Phase de transfert : après la connexion, le transfert peut alors avoir lieu dans un sens ou dans l'autre. Chaque segment est envoyé dans le tuyau ainsi créé. Les segments envoyés sont vérifiés et acquittés par le client ce qui ici est montré par la notion de tuyau.



Transfert des segments

Fin de connexion : Après envoi des données, le serveur (ou le client) envoi une demande de fin de connexion qui est acquittée par le client qui envoie lui-même une fin de connexion qui est acquittée par le serveur. Toujours la double vérification.

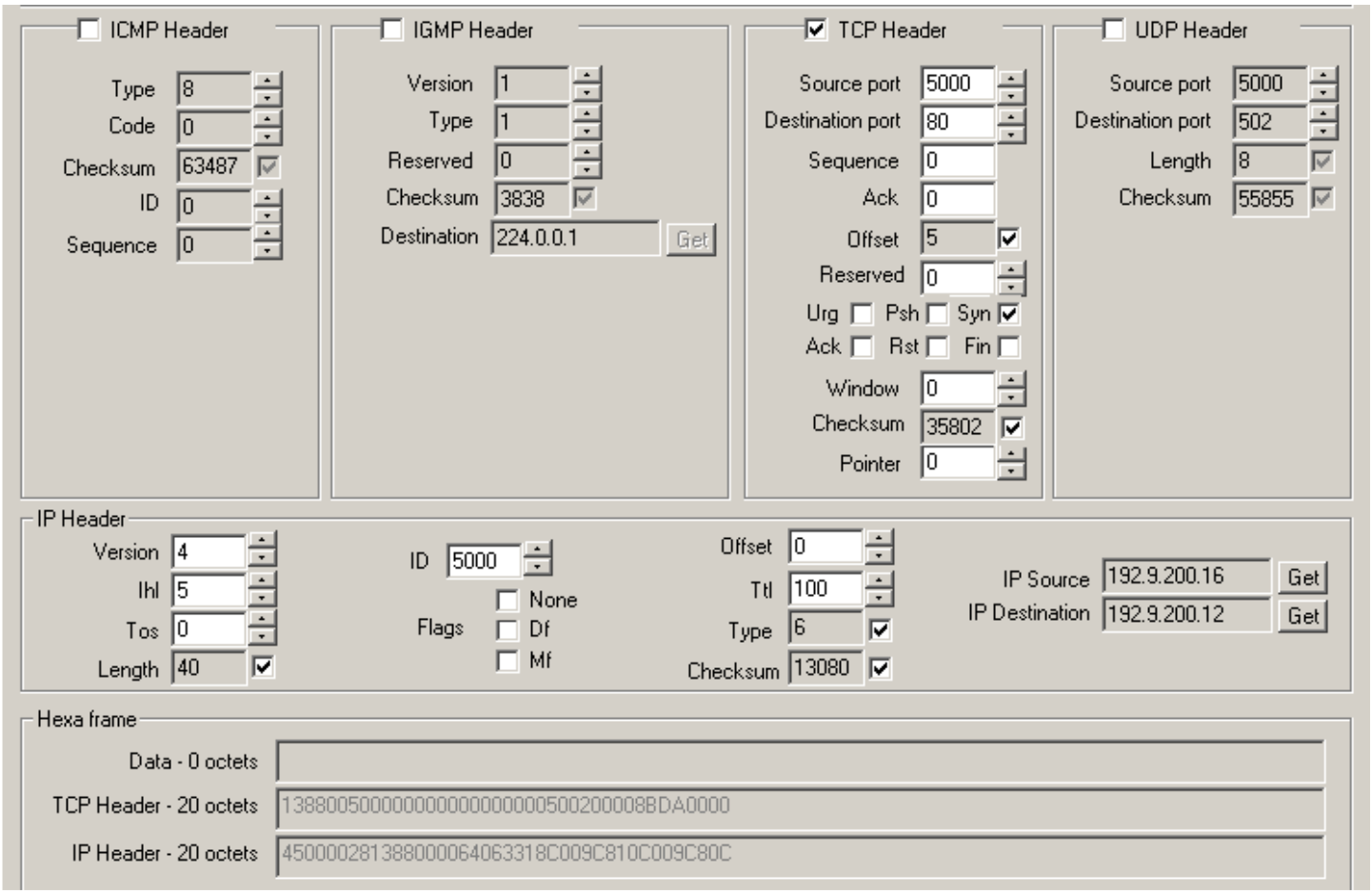


Déconnexion entre le serveur et le client

TCP utilise un mécanisme de transmission de données par "Fenêtre coulissante". Le but de ce mécanisme est de permettre l'envoi de données, sans attente d'acquittement, avec un certain niveau d'anticipation.

La fenêtre indique le nombre d'octet que l'on peut recevoir sans attente d'acquittement. La largeur de la fenêtre (en réception) indique l'espace mémoire disponible en réception. Une largeur de fenêtre à 0 indique que l'émetteur du segment TCP ne désire plus (pour l'instant) recevoir de données.

En utilisant le logiciel frameIP envoyez une trame du type TCP sur le réseau



Capturez cette trame et interprétez là

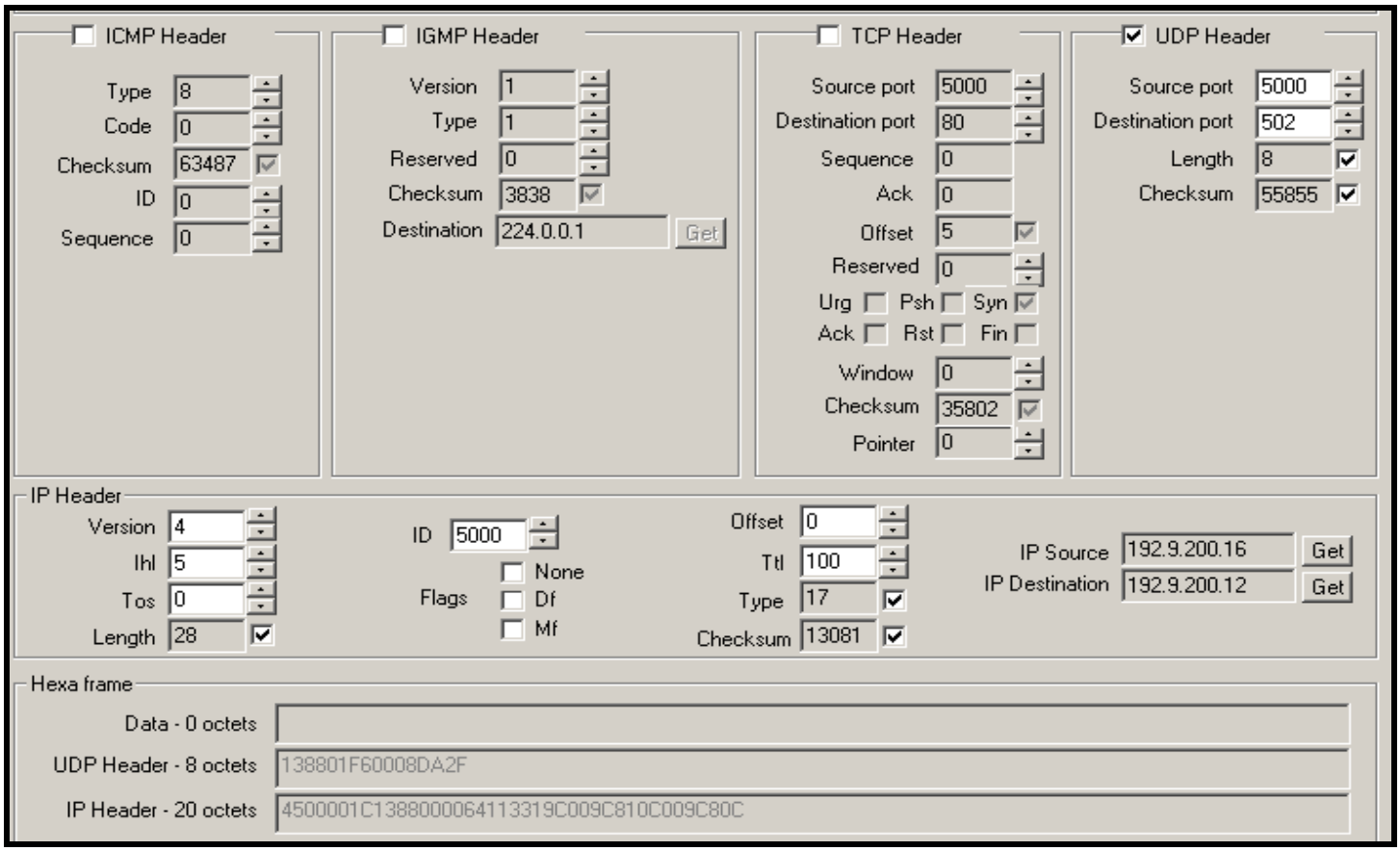
UDP (USER DATAGRAM PROTOCOL) service non – connecté

- Service sans connexion, sans garantie, utilisant IP pour le transport de messages entre machine

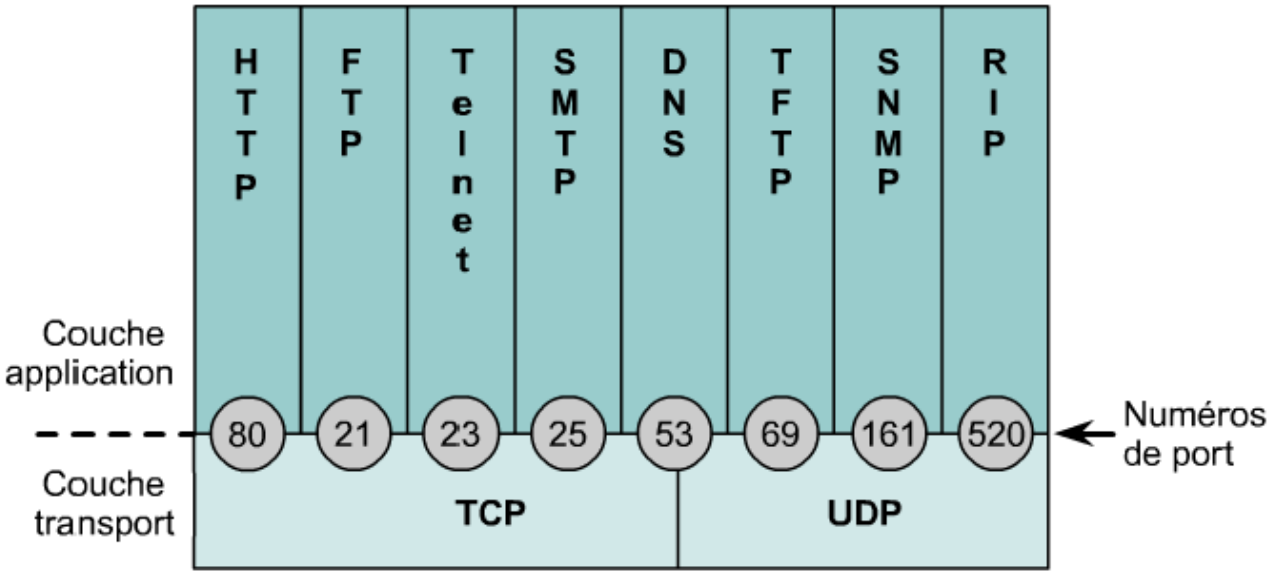
Source port	Destination port
Length	checksum

- Source port : n° de port
  - ⇨optionnel, identifie un port pour la réponse
- Destination port : n° port
- Length, taille de l'en-tête et des données
  - ⇨unité octet
  - ⇨taille max = 64 Koctets
- Checksum : fonction de l'en-tête et des données
  - ⇨optionnel
  - ⇨c'est la seule garantie sur la validité des données qui arrive à la destination

En utilisant le logiciel frameIP envoyez une trame du type UDP sur le réseau



Capturez cette trame et interprétez là

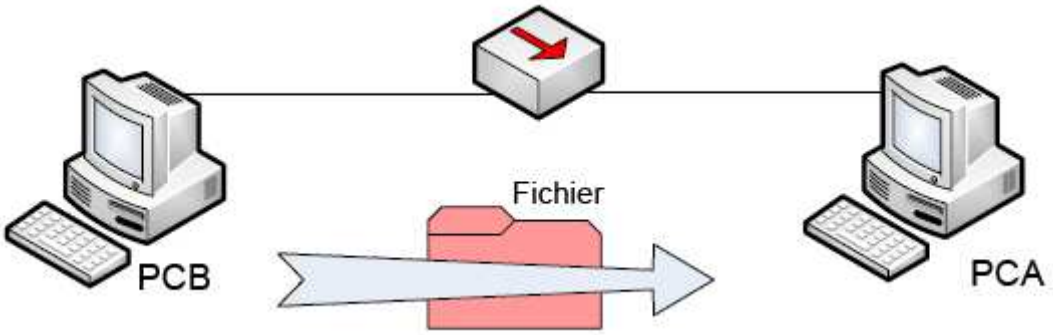


Les principales applications associées aux ports et protocole de la couche 4 (UDP ou TCP)

Partie 4 : La couche application

EXEMPLE N°1 LE PROTOCOLE TFTP (TRIVAIL FILE TRANSFERT PROTOCOLE), PORT 69

INTRODUCTION



Le protocole **TFTP** permet le transfert d'un fichier d'un PCA vers un PCB. Ce protocole utilise comme tous les protocoles applicatifs la notion de client/serveur. Le serveur **tftp** est capable d'envoyer (commande **get** pour le client) ou de recevoir (commande **put** pour le client) un fichier à un client. Ce protocole utilise **UDP** pour la couche transport. UDP va découper le fichier en segment, numéroter chacun de ces segments et les envoyer. Le récepteur grâce au CRC de la couche transport pourra dire si le segment reçu comporte des erreurs de transmission ou non et renvoyer une trame d'acquittement pour chaque segment reçu.

Caractéristiques de TFTP :

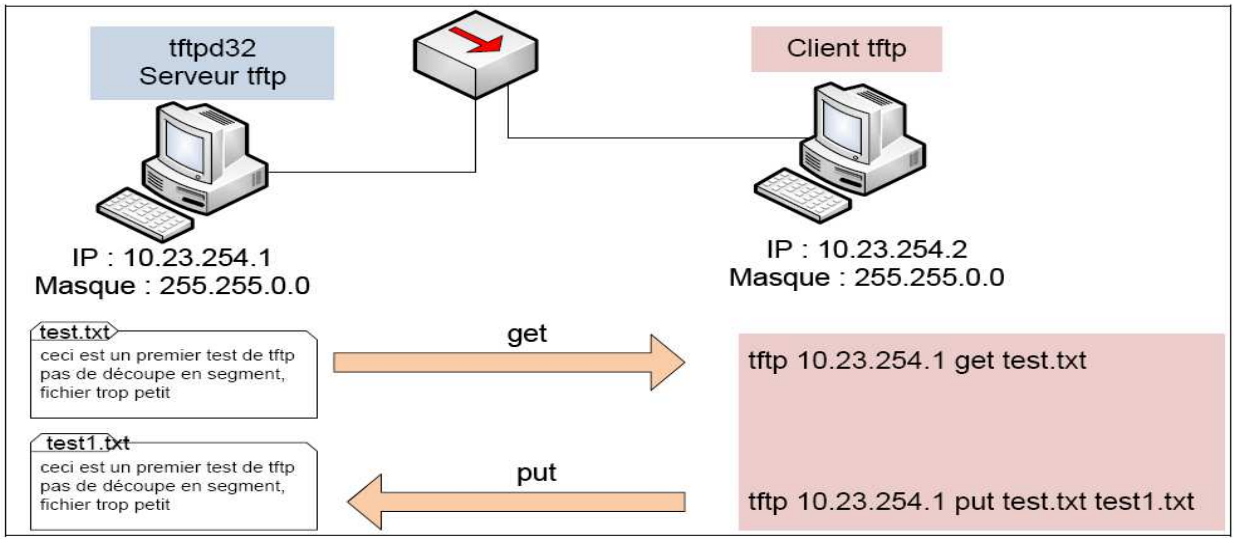
✓ permet de contrôler l'intégrité du fichier reçu (numérotation des segments et acquittement de chacun des segments).

✓ ne permet pas la connexion par login et mot de passe d'un client au serveur. Aucune sécurité n'est assurée quant à l'identité du possesseur du fichier transféré.

✓ Ne permet pas le transfert d'un ensemble de fichiers (un seul fichier possible)

✓ Est un protocole très simple.

Exemple d'utilisation d'un client tftp :



Supposons que le client **tftp** se trouve sur le poste à l'adresse **10.23.254.2** et que le **serveur** est à l'adresse **10.23.254.1**, sur le client supposons que dans le répertoire courant il existe le fichier test.txt.

✓ Pour envoyer le fichier test.txt dans le répertoire par défaut du serveur tftp

tftp 192.9.200.xxx put test.txt

✓ Même chose mais cette fois-ci on désire que le fichier envoyé ait le nom test.1 sur le serveur

tftp 192.9.200.xxx put test.txt test.1

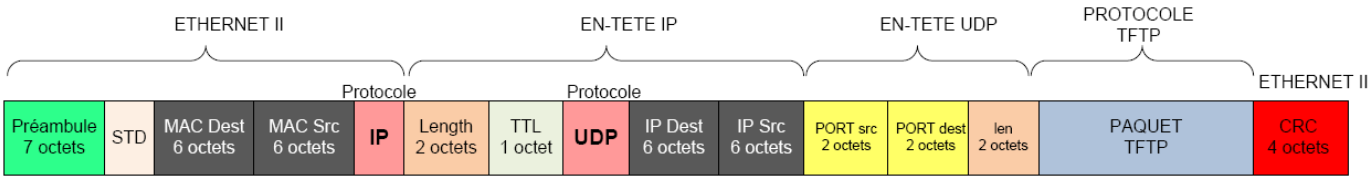
✓ Enfin pour téléchargé le fichier test.1 sur le client.

tftp 192.9.200.xxx get test.1

A la suite de ces 3 commandes, le répertoire courant du client a 2 fichiers : test.txt et test.1

ETUDE DU PROTOCOLE TFTP

La trame de la figure ci-dessous n'est pas complète, il manque notamment des informations dans **l'en-tête IP** et le CRC dans **l'en-tête UDP**. Par contre cette trame permet de visualiser les principales informations présentes dans la trame Ethernet TCP/IP (on parle de TCP/IP même si la couche transport utilisée n'est pas TCP) circulant lors des requêtes et réponse tftp. On pourra de plus noter la position du paquet TFTP dans cette trame. Nous allons étudier maintenant ce qui se trouve dans le paquet TFTP.



Il existe 5 types de paquets TFTP :

✓ paquet **RRQ** (READ REQUEST) ou se trouve le nom du fichier à télécharger du serveur vers le client

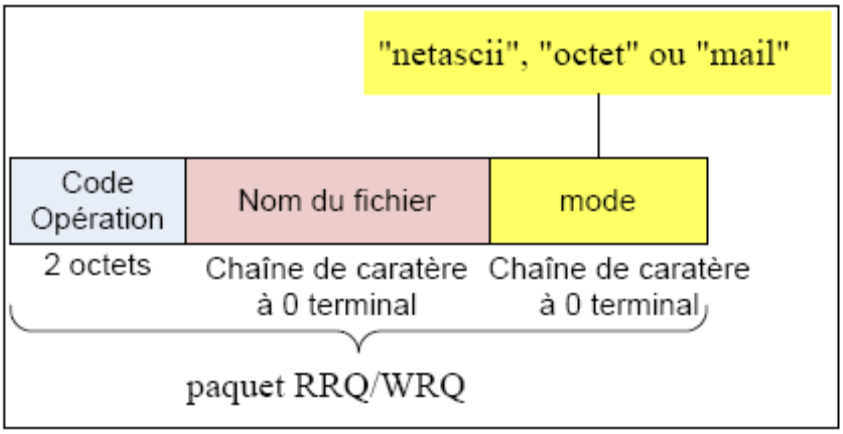
✓ paquet **WRQ** (WRITE REQUEST) ou se trouve le nom du fichier à télécharger du client vers le serveur

✓ Paquet **DATA** ou se trouvent le ou les segments avec le numéro de bloc associé.

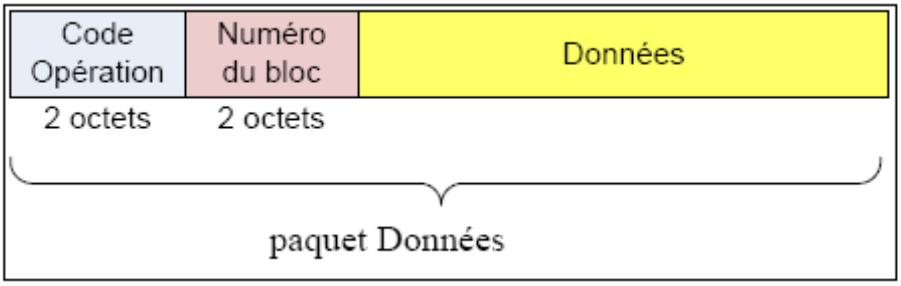
✓ Paquet **ACK** (ACKNOWLEDGE) permet au récepteur d'acquitter le numéro de segment courant

✓ Paquet **ERR**(ERROR) envoyé si une erreur est détectée dans le segment, dans ce cas le paquet est renvoyé.

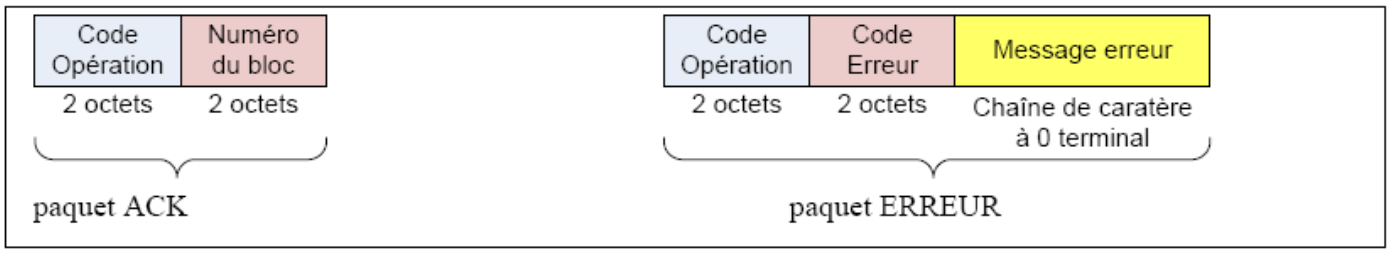
Le détail de ces paquets est donné ci-dessous :



paquet RRQ ou WRQ, envoi du nom du fichier et du codage employé



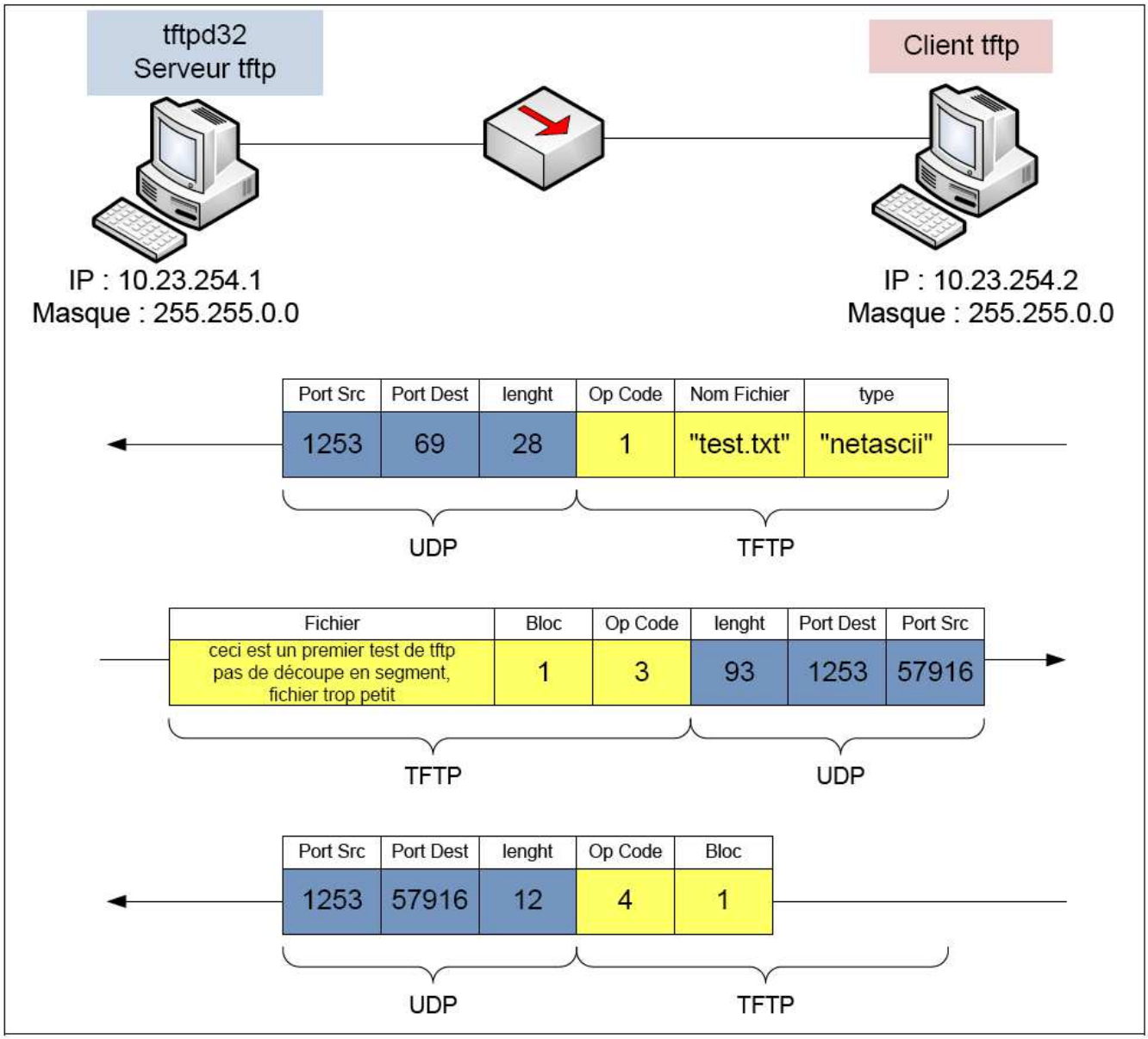
paquet DATA, envoi du segment et de son numéro



paquets ACK et ERR, envoyé aprés réception du segment

Chaque paquet possède un code opération codé sur 2 octets. Ce code ne peut avoir que 5 valeurs possibles permettant de préciser le type de paquet envoyé.

code opération	opération
1	Demande de lecture (RRQ)
2	Demande d'écriture (WRQ)
3	Données (DATA)
4	Accusé de réception (ACK)
5	Erreur (ERR)



résumé des paquets envoyés pour le transfert du fichier test.txt via le protocole tftp

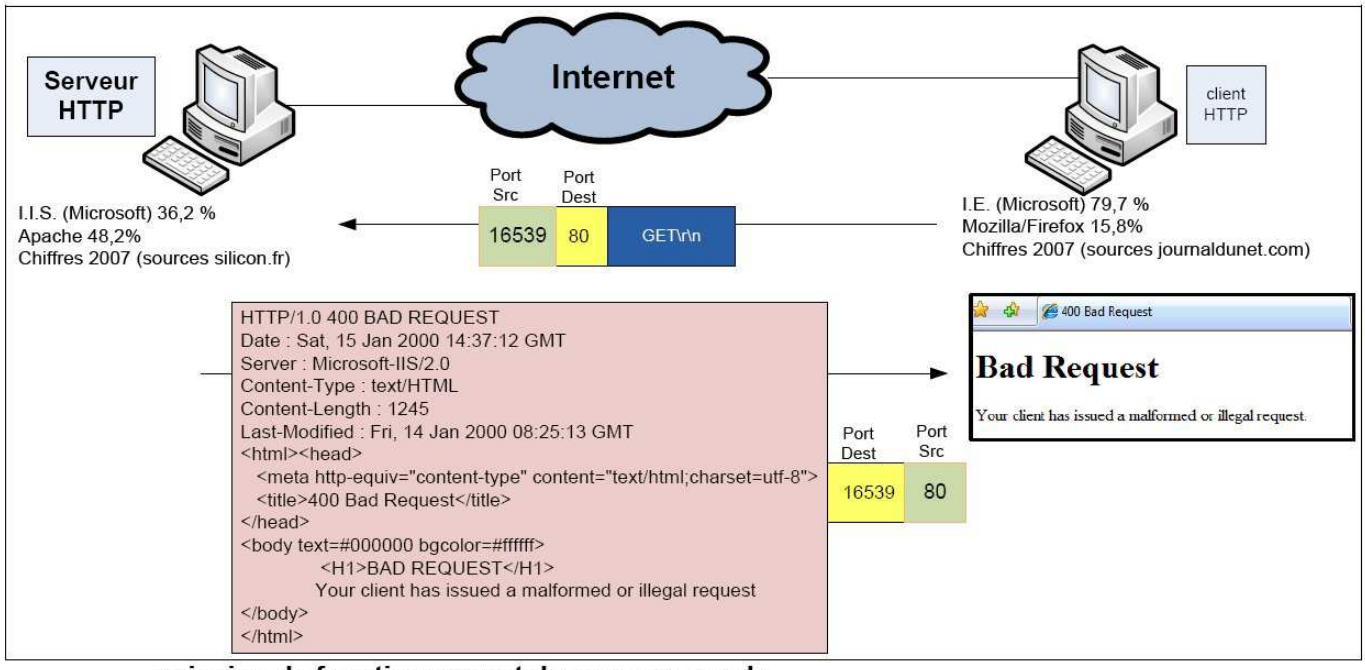
Exemple N°2 : Le protocole HTTP (Hyper Text Transport Protocol)

Le Protocole **HTTP** (Hypertext Transfert Protocol) est la couche application permettant de spécifier le dialogue entre un serveur WEB (principalement **Apache** et **IIS**) et un client WEB ou navigateur (**Internet Explorer** et **Firefox/Mozilla**).
C'est un protocole texte basé sur **TCP**, il suffit donc d'ouvrir une connexion **TCP** sur le port 80 du serveur et d'envoyer une requête texte vers le serveur.

L'utilisation de **telnet** permet d'écrire du texte directement sur la couche application et donc de simuler une requête **http, ftp, pop ou smtp**. Le client **telnet** est fourni sous Windows. Il suffit de taper dans la console les 2 lignes ci-dessous.

```
telnet www.google.fr 80
GET
```

La première ligne permet une connexion en **tcp** sur le port 80 vers l'adresse **IP** du site www.google.fr. La deuxième ligne envoie la méthode **GET** qui est la commande minimale de demande de page web par défaut sur le serveur web par défaut.
La réponse du serveur est composée pour la couche application d'un **en-tête** (texte se trouvant avant la balise ‹**html**›) et d'un **corps de texte** dans lequel se trouvent les balises **<head> et </head>** (informations non affichable dans le navigateur ou se trouvent les META) suivi du message à proprement parlé se trouvant entre les balises **<body> et </body>**.



principe de fonctionnement des serveurs web

Dans l'exemple de la figure précédente, le client **http** (simulé par la commande **telnet**) envoie 5 lettres (couche application) 'G' 'E' 'T' \r \n vers le port 80 à l'adresse **IP** de google . Les caractères \r et \n correspondent respectivement au code ASCII du retour chariot et saut de ligne. La réponse du serveur montre ici que la requête envoyée ne correspond pas à une requête valide (« Bad request »). En effet, comme nous allons le voir plus loin, une requête minimale doit

comporter la requête **GET** suivi du répertoire courant sur le serveur (ici la racine /) et de la version du protocole utilisé (http/1.0). Vient ensuite, au minimum le nom du serveur (host). La réponse du serveur commence alors par la ligne **HTTP/1.0 200 OK**, ce qui veut dire que la requête a été validée par le serveur. Cette ligne est ensuite suivi de l'en-tête (Date, Content-Type, Content-Length,…) terminé par la balise **<html>**. Vient ensuite le corps de texte.

```
telnet www.google.fr 80
GET / HTTP/1.0
host: www.google.fr
```

ETUDE DU PROTOCOLE : LA REPONSE

Une réponse **HTTP** est un ensemble de lignes envoyées au navigateur par le serveur (exactement comme la requête). Elle comprend: une ligne statut, une liste de champs, une ligne blanche, le corps de la réponse (généralement le contenu du fichier demandé).
La ligne de statut précise la version du protocole utilisée et l'état du traitement de la requête à l'aide d'un code et d'un texte explicatif. Le code est utilisé par le programme client alors que l'explication permet d'informer l'utilisateur en cas d'erreur.

Prenons un autre exemple de réponse du serveur. Nous pouvons voir l'en-tête suivi du corps de texte.

```
HTTP/1.0 200 OK
Date : Sat, 15 Jan 2000 14:37:12 GMT
Server : Microsoft-IIS/2.0
Content-Type : text/HTML
Content-Length : 1245
Last-Modified : Fri, 14 Jan 2000 08:25:13 GMT
```

EN-tête

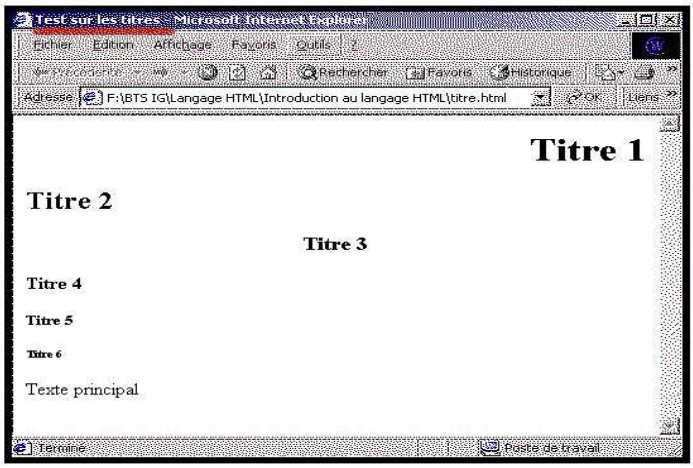
```
<html><head>
<meta http-equiv="content-type" content="text/html;charset=utf-8">
<title>400 Bad Request</title>
</head>
<body text=#000000 bgcolor=#ffffff>
<H1>BLABLABLA</H1>
Cequ'il faut afficher !
</body>
</html>
```

Corps de texte

Langage HTML

```
<HTML>
<HEAD><TITLE> Test sur les titres </TITLE></HEAD>

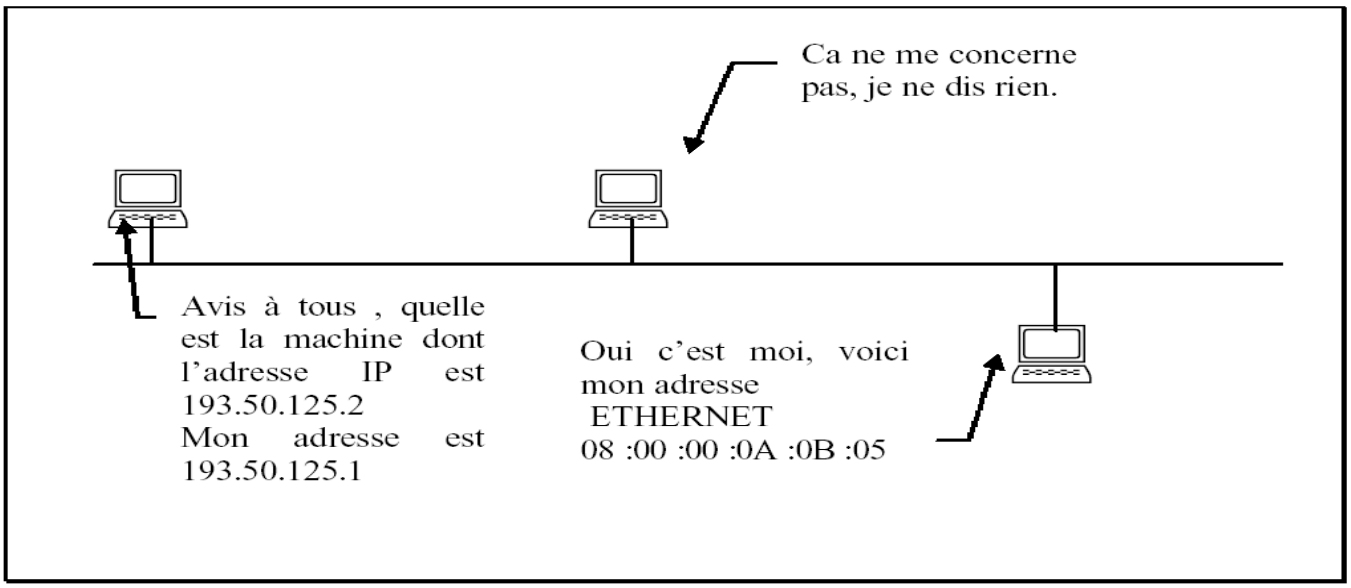
<BODY>
  <H1 align="right">Titre 1</H1>
  <H2>Titre 2</H2>
  <H3 align="center">Titre 3</H3>
  <H4>Titre 4</H4>
  <H5 align="left">Titre 5</H5>
  <H6>Titre 6</H6>
  Texte principal
</BODY>
</HTML>
```



Partie N°5 : Les protocoles particuliers

1) Les protocoles ARP et RARP (nécessaire en mode TCP)

Etant donné que le protocole IP et ses adresses peuvent être utilisés sur des architectures matérielles différentes (réseau ethernet, token ring) possédant leurs propres adresses physiques, il y a nécéssité d'établir les correspondances entre adresses IP et adresses matérielles des ordinateurs d'un réseau.
Ces protocoles sont la jonction entre la couche liaison et la couche réseau.
Plaçons nous dans le cas d'une correspondance à établir entre adresses IP et ethernet et le nécessité de la résolution d'adresse fournie par ARP *(Address Resolution Protocol)* lors d'une connexion FTP.



Le client FTP convertit l'adresse du serveur FTP en une adresse IP grâce à un serveur de nom DNS 193.50.125.1
Le client FTP demande à la couche FTP d'établir une connexion avec cette adresse
TCP envoie une requête de connexion à ce serveur en émettant un datagramme IP contenant l'adresse IP
La machine émettrice doit convertir l'adresse IP sur 4 octets en une adresse ethernet sur 6 octets avant d'émettre la trame ethernet contenant le paquet IP
Le module ARP envoie une requête ARP dans une trame ethernet avec une adresse de destination. Ainsi toutes les machines du réseau local reçoivent cette requête contenant l'adresse IP à résoudre
La couche ARP de la machine visée reconnaît que cette requête lui est destinée et répond par une réponse ARP contenant son adresse matériel 08 :00 :00 :0A :0B :05 Les autres machines du réseau ignorent la requête
La réponse ARP est reçue par l'émetteur de la requête
La réponse ARP est reçue par la couche ARP du client FTP et le driver ethernet peut alors émettre le paquet IP avec la bonne adresse ethernet de destination

adresse Ethernet de destination	adresse Ethernet de source	type de trame	type de mat.	type de prot.	taille mat.	taille prot.	op	adresse Ethernet de l'émetteur	adresse IP de l'émetteur	adfesse Ethernet cible	adresse IP cible
an-tete Ethernet			28 octets requete/réponse ARP/RARP								

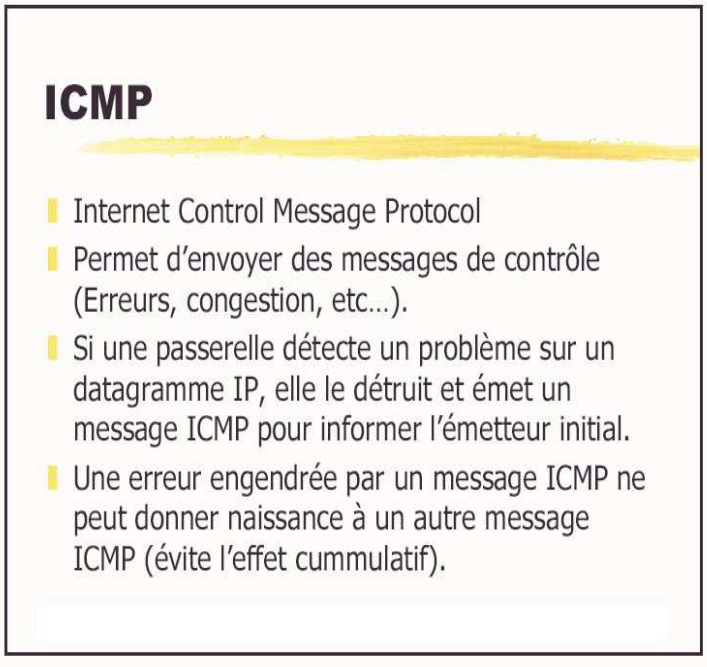
L'adresse ethernet de destination sera ff :ff : ff :ff :ff :ff l'adresse multicast désignant toutes les machines du réseau à la fois. La valeur du champ type de trame est 0x806 indiquant le protocole ARP. Le champ type de matériel est égal à 1 pour un réseau ethernet et est celui type de protocole est égal à 0x800 pour IP

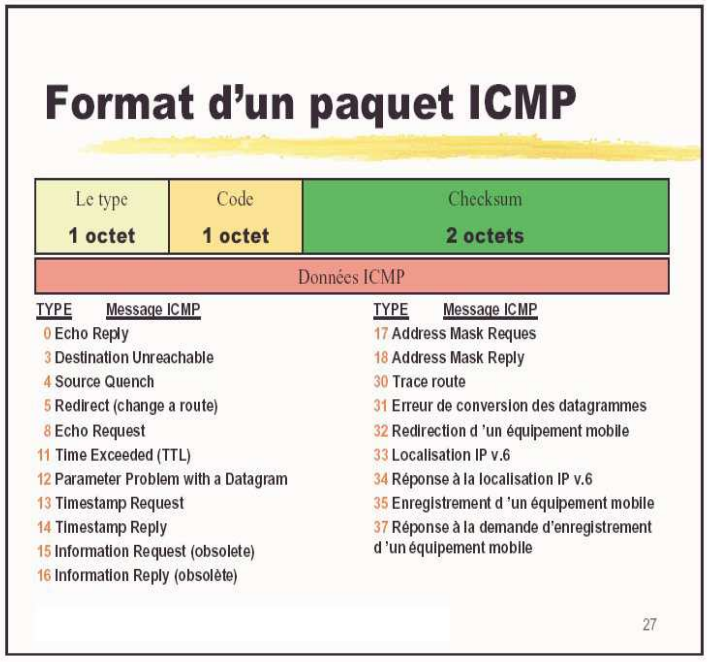
RARP

C'est ARP à l'envers, la machine diffuse une trame ETHERNET pour demander sa propre adresse IP. Un serveur va lui renvoyer son adresse dans une trame ETHERNET parfaitement définie (service **8035**). C'est utilisé pour des machines n'ayant pas de disque dur, serveurs de terminaux... Cependant, comme nous le verrons, l'adresse IP est insuffisante pour travailler sur un vaste réseau où nous devons définir l'adresse de la passerelle, des serveurs de noms, etc. RARP est donc abandonné au profit de DHCP qui assument ce genre de fonction de manière beaucoup plus évoluée.

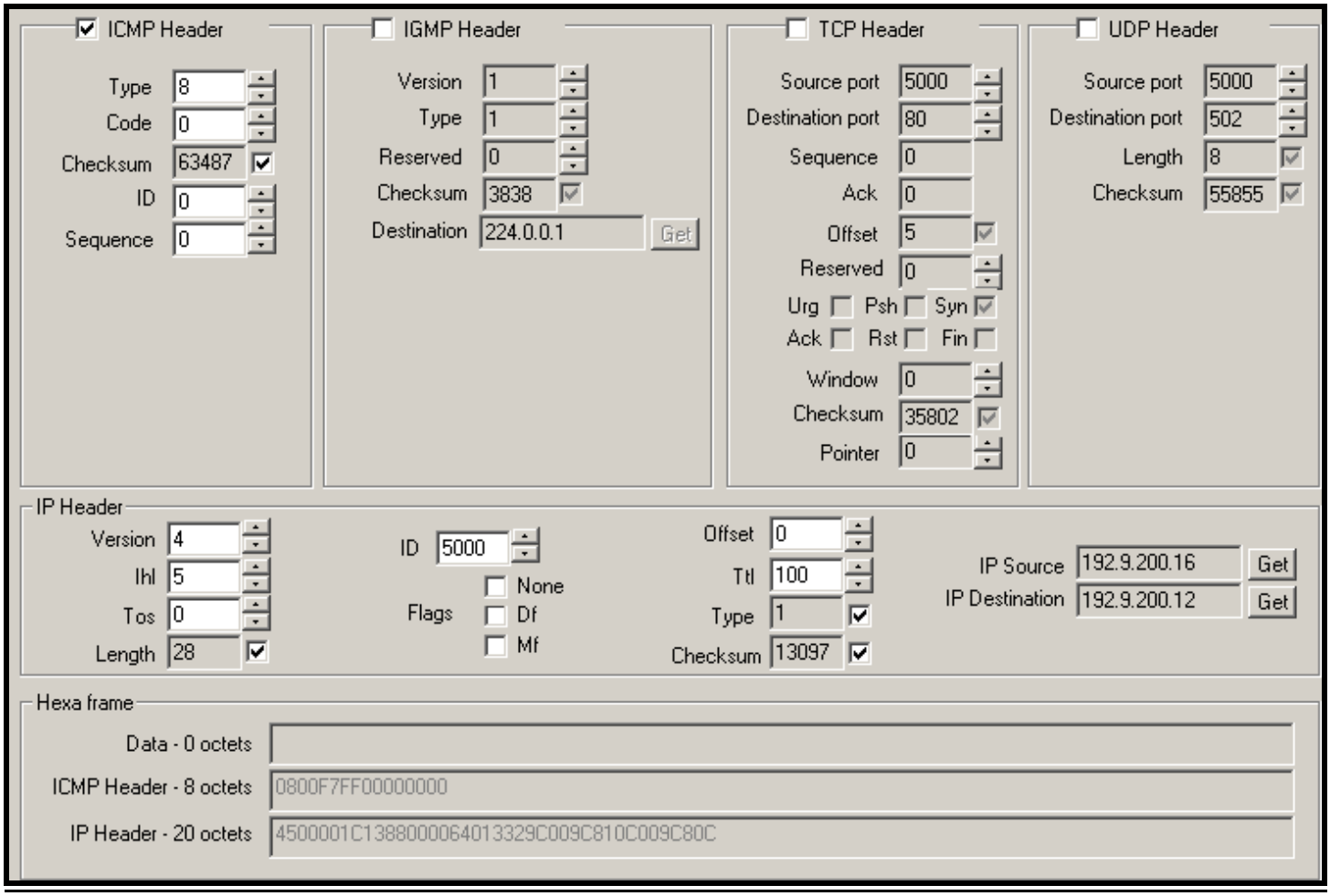
2) Le protocole ICMP

ICMP, sert à echanger une information de contrôle entre 2 machines, le plus souvent une erreur.





En utilisant le logiciel frameIP envoyez une trame du type ICMP sur le réseau



Capturez cette trame et interprétez là

ANNEXE : LES RESEAUX LOCAUX (LAN)

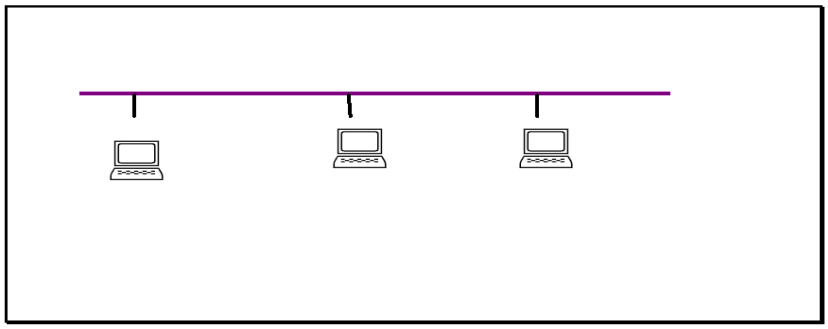
1) Les types de réseaux locaux

Le But : Raccorder sur un même support physique des ordinateurs, et permettre de communiquer avec un ensemble d'ordinateurs sur ce support. Un seul message sur le support peut être lu par plusieurs ordinateurs. Les modems sont remplacés par des cartes réseaux que l'on installe dans les ordinateurs. Ces réseaux sont de taille limitée. Cette limite est due au protocole lui-même.

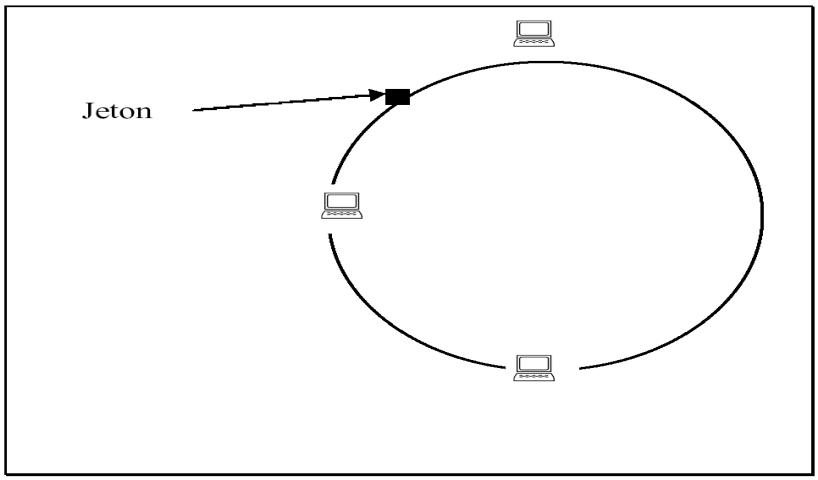
On trouve schématiquement deux types de réseaux, les **BUS** et les **Anneaux**

Dans le cas des BUS, tout le monde parle sur un même fil. Pour gérer les collisions inévitables, on s'empare du fil en émettant suffisamment longtemps (le temps de la propagation aller/retour du signal sur le support), pour s'assurer que le message a été correctement lu.

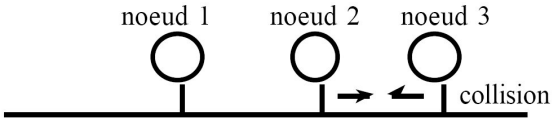
type BUS



type anneau



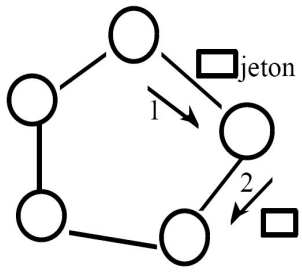
2) Les principales stratégies d'accès au réseau



(ex : bus Ethernet)

❒ Accès CSMA/CD («Carrier Sense Multiple Access/Collision Detection») :

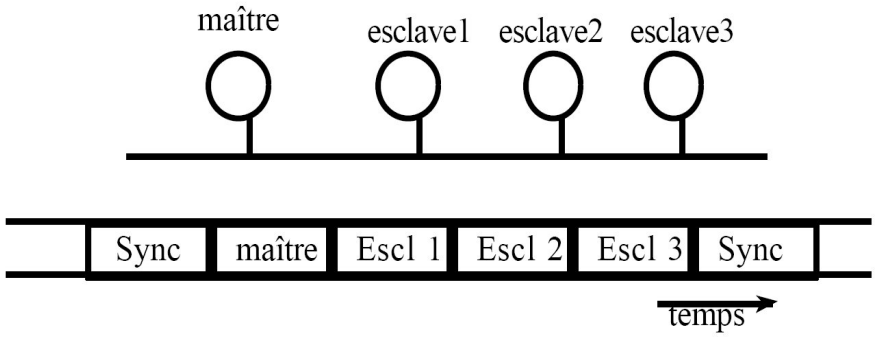
- ☞ Les nœuds attendent un blanc avant d'émettre
- ☞ Si plusieurs émissions simultanées, détection de la collision
- ☞ Accès multiples possibles
- ☞ En cas de contention, l'arbitrage se fait par durée d'attente aléatoire pour chaque nœud en collision
- ☞ Problème si charge élevée
- ☞ Peu déterministe (risque de blocage)



(ex : bus FDDI)

❒ Accès jeton sur anneau («Token Ring») :

- ☞ Topologie en anneau
- ☞ Circulation d'une trame particulière (jeton) de noeud en noeud quand pas d'émission
- ☞ Le noeud désirant émettre sur le médium garde le jeton, émet sa trame puis rend le jeton
- ☞ Connexion point à point, déterminisme
- ☞ Problèmes si médium rompu, perte ou duplication de jeton
- ☞ Variante : bus à jeton



❒ Accès par multiplexage temporel («Time Division Multiple Access») :

- ☞ Emission d'un mot de synchronisation par le noeud maître
- ☞ Emission des données par tous les noeuds à un Intervalle de Temps précis (IT)
- ☞ Taille des données fixe
- ☞ Meilleur efficacité que le polling

RECAPITULATIF :

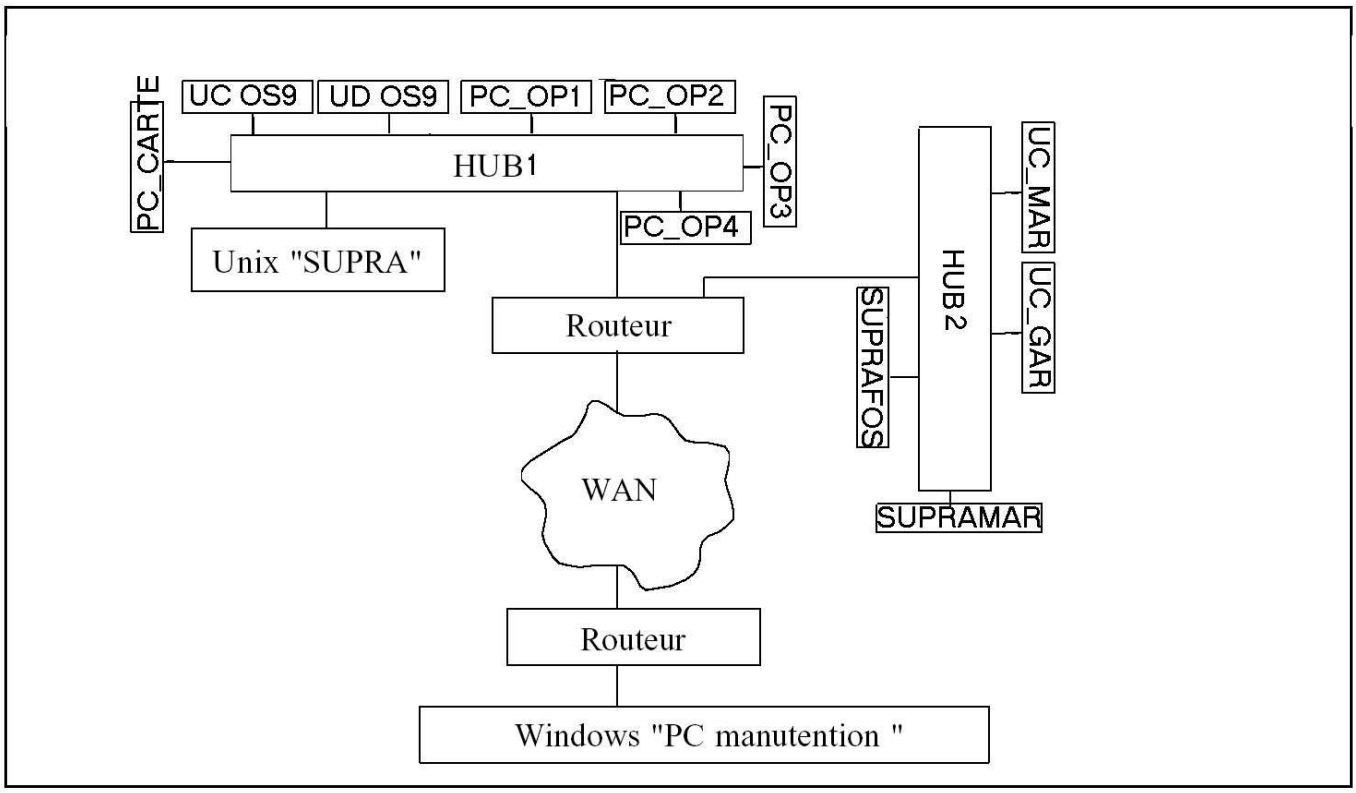
	Efficacité trafic faible	Efficacité trafic élevé	Déterminisme	Priorité	Robustesse	Flexibilité couche physique	Coût par noeud
TDMA	☹	☺	☺	☹	☹	☺	☹
TokenRing	☺	☺	☺	☺	✔	✔	✔
CSMA/CD	☺	☹	☹	✔	☺	☺	✔

☺ : bon
☹ : mauvais
✔ : OK

1) TDN°1 : Analyse du réseau

1.1) Première partie : Architecture réseau.

Les informations de mouvement (entrées et sorties des conteneurs) envoyés par les entreprises de manutention sont stockée dans une Base de données (station UNIX).
On donne ci-dessous la configuration partielle du réseau :



L'environnement TCP/IP a été configuré comme suit:

Dans le fichier /h0/internet/etc/hosts

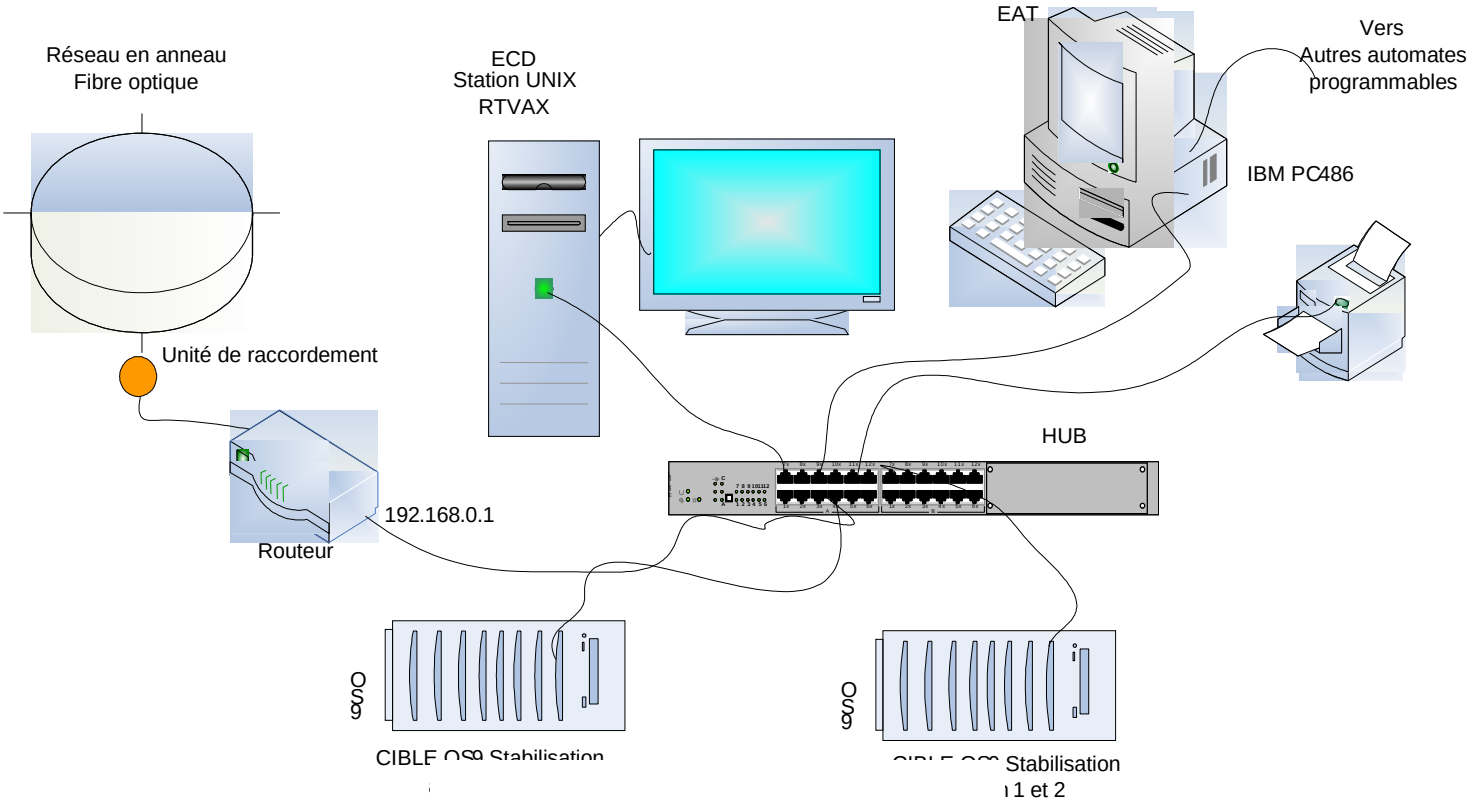
192.168.1.40 UC_MAR
192.168.1.41 UC_GARE
192.168.1.42 SUPRAMAR
192.168.1.43 SUPRAFOS
128.1.1.1 UC_MOUREPIANE
128.1.1.2 UD_MOUREPIANE
128.1.1.3 PC_OP1
128.1.1.4 PC_OP2
128.1.1.5 PC_OP3
128.1.1.6 PC_CARTE
128.1.1.7 SUPRA_MOUREPIANE

1.2) Deuxième partie : question

1.2.1) Combien le routeur dispose t'il de carte réseau ?, quel est le nombre de sous réseaux ou réseaux qu'il peut router ?
1.2.2) Quelles sont les classes de ce réseau ?
1.2.3) Donnez le masque de sous réseau qui permet de disposer sur le hub N°1 de 29 machines (Adresse de début du sous réseau 128.1.1.0).
1.2.4) Quelle sera l'adresse la plus haute de la machine connecté au hub N°1
1.2.5) Pour le HUB N°2, On veut connecter 61 machines possibles, déterminez le masque de sous réseau, l'adresse IP la plus basse possible de ce sous réseau est 192.168.1.1
1.2.6) Grâce à un analyseur de trames réseaux on obtient la trame ETHERNET suivante suite à une commande ARP :
00E07DE3AA98000AE6540AF608060001
080006040002000AE6540AF680010103
00E07DE3AA988001012D65666768696A
6B6C6D6E6F70717273747576291CEB5C
Déterminez l'adresse MAC source et destinataire, l'adresse IP source et destinataire.

TD N°2) configuration réseau

On donne les différents éléments informatiques du réseau. Même si l'architecture du réseau est basée sur 4 anneaux en fibre optique protocole X25, les sous réseaux connectés à chaque unité de raccordement utilisent les réseaux ETHERNET à technologie TCP/IP. On pourra associer à chaque sous réseau une adresse IP unique le routeur se situe alors dans chaque unité de raccordement.



1.1) On donne le masque du sous réseau suivant :

255.255.255.224

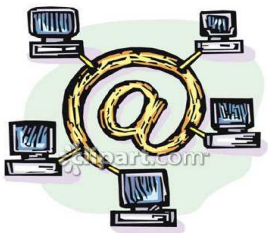
Combien de machines pourra t'on au maximum brancher sur le HUB de ce sous-réseau.

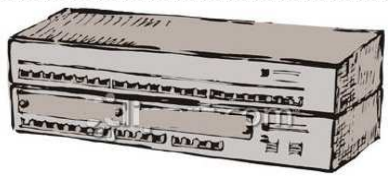
1.2) Pourquoi peut on affirmer que c'est un sous réseau ?

1.3) Quelles sont les couches du modèle OSI que va traiter le HUB

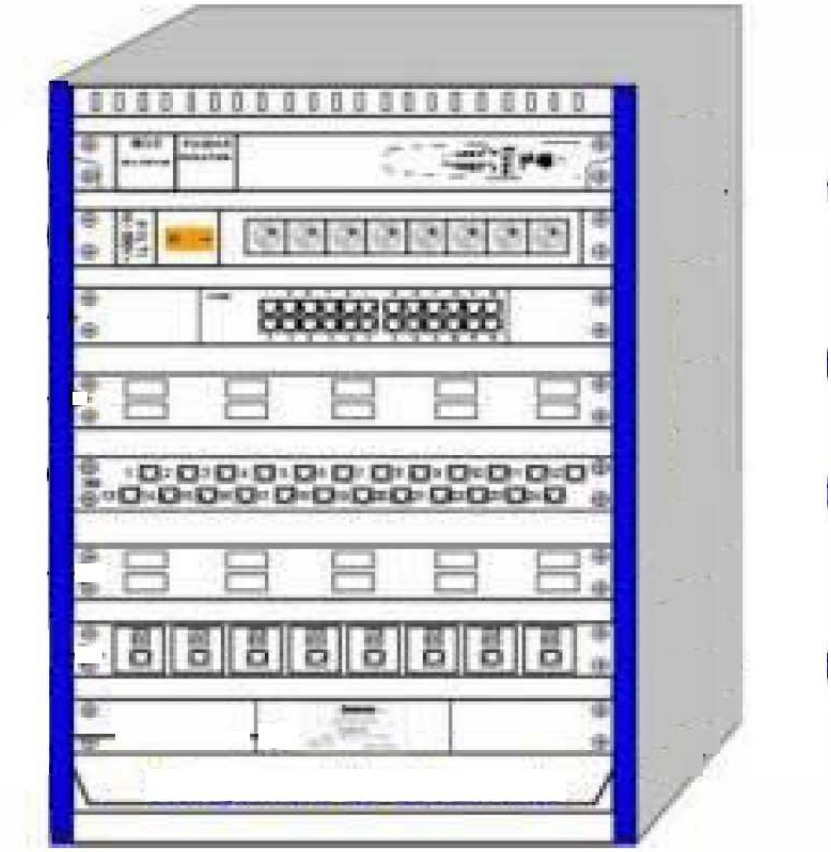
1.4) Quel est la couche du modèle OSI que va interpréter le routeur

1.5) Quelle est l'adresse de diffusion générale pour ce sous réseau





LES RESEAUX LAN A TECHNOLOGIE TCP-IP : ASPECTS MATERIELS



Réseaux aspects matériels

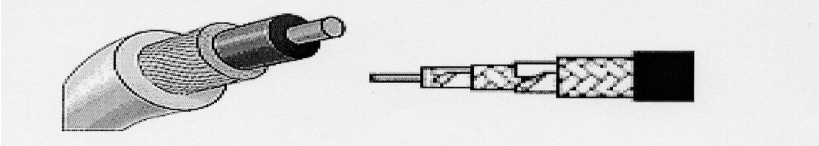
IL existe différentes architectures au niveau des réseaux LAN MAN WAN (Ethernet, Token Ring, FDDI, ATM, X25) que se soit au niveau du matériel et des protocoles.
Ce polycopié expose les différents types de matériel que l'on rencontre dans les réseaux locaux du type ETHERNET (le plus populaire). La première partie énumère les différents types de support de transmission, la seconde partie les principaux types de câblage ETHERNET et leurs matériels associés.

1) Les médias de transport.

Dans les médias de transport, nous trouvons:

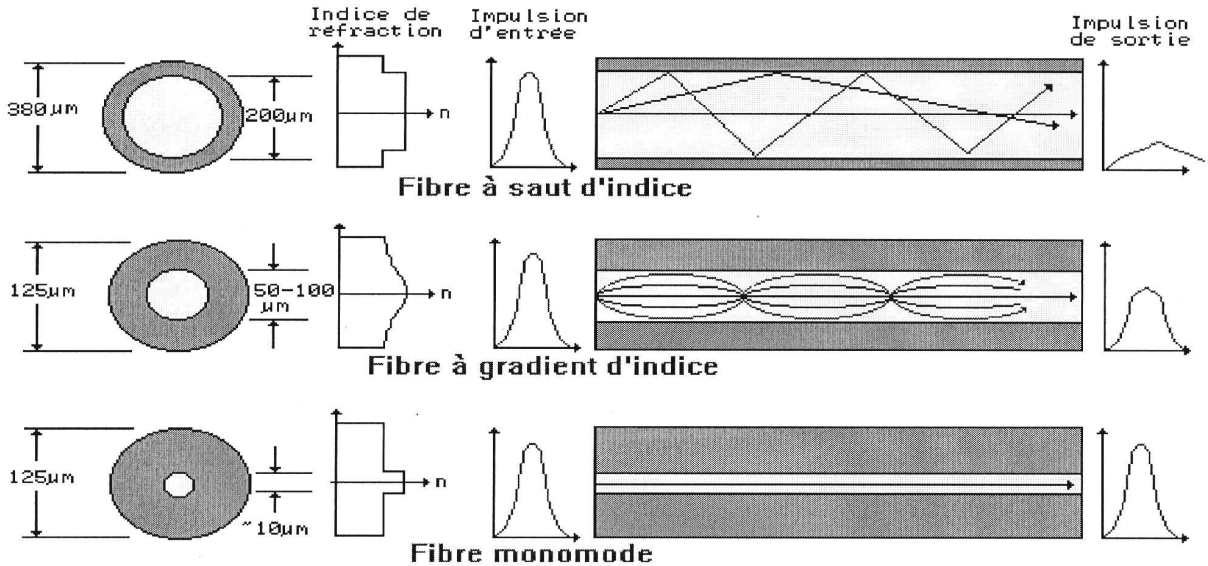
1.1) Les câbles en "cuivre"

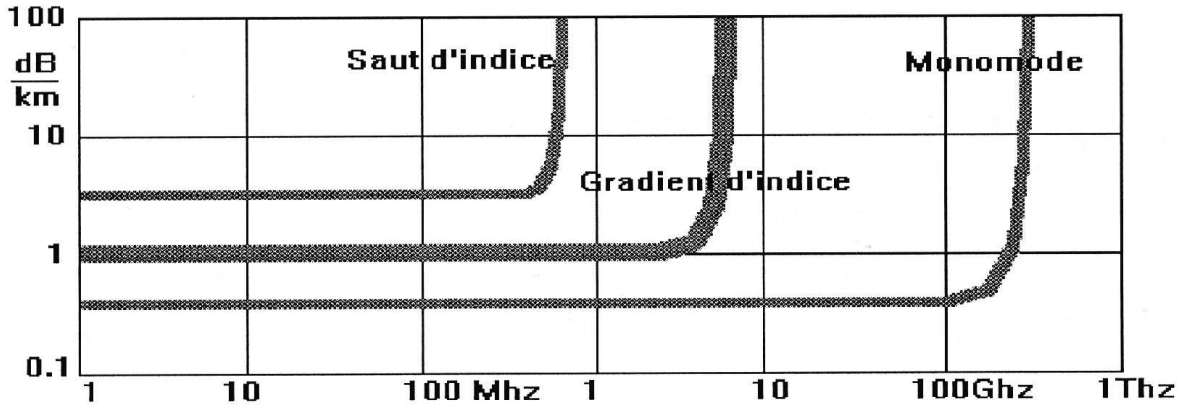
C'est en général le média le moins coûteux, mais également le plus limité, surtout en terme de distance.



1.2) Les fibres optiques qui ont l'avantage d'être insensibles aux perturbations électromagnétiques dans lesquelles nous baignons. De plus, la vitesse de propagation de la lumière dans ces fibres autorise de longues distances et de nombreuses solutions permettent une très grande bande passante, donc un gros débit de données.
Malheureusement, la fibre optique souffre de quelques défauts:

Sa relative fragilité.
La difficulté d'y adapter de la connectique
Le prix de cette connectique.





La fibre monomode dont le cœur est si fin que le chemin de propagation des différents modes de propagation est pratiquement direct. La dispersion devient quasiment nulle. La bande passante transmise est presque infinie (>10GHz/km). Cette fibre est utilisée essentiellement pour les sites à distance.

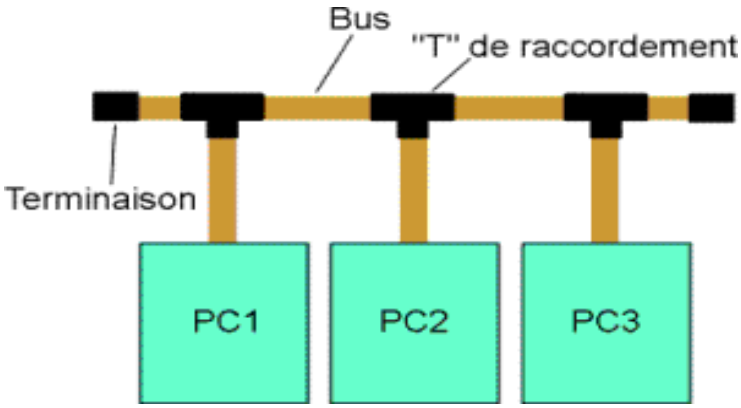
2) les principaux types de câblage ETHERNET

2.1) Câblage 10base2

Ethernet s'appuie sur la norme IEEE 802.3 qui définit la méthode d'accès CSMA/CD (Détection de Porteuse avec Accès Multiples/Détection de Collisions). Le nombre maximum de stations sur un réseau Ethernet est de 1024. Le débit défini par l' IEEE est de 10, 100 ou 1000 Mbps. C'est l'une des normes les plus anciennes devenues un peu obsolète. Son débit est de 10Mbits/s, sa topologie est du type anneau

2.1.1) Principe.

Le principe du "BUS" est extrêmement simple: Un conducteur unique représente le réseau. Chaque extrémité est bouclée sur un "bouchon" dont l'impédance électrique est égale à l'impédance caractéristique du conducteur, ceci afin d'éviter les réflexions des signaux en bout de câble. Chaque poste est "piqué" sur ce bus au moyen d'un "T" de raccordement. La technologie présentée ci-dessous est adaptée aux petits réseaux.



2.1.2) La connectique

Dans ce cas de figure, le câble le plus souvent utilisé est du coaxial de type RG58. Il est souple, fin et relativement facile à mettre en œuvre.



Le câble RG58, présente une impédance caractéristique de 50 Ohms
Les prises de type "BNC" sont facilement montées si l'on dispose de la pince à sertir adéquate.



Le "T" de type BNC permet les raccordements, avec son type de prise à verrouillage par baïonnette.



Le bouchon de terminaison, également de type BNC, se trouve aisément dans le commerce, au même titre d'ailleurs que les autres accessoires décrits plus haut.



Au final, il est très simple et rapide de connecter entre eux quelques PC par ce moyen.



2.1.3) Conclusion

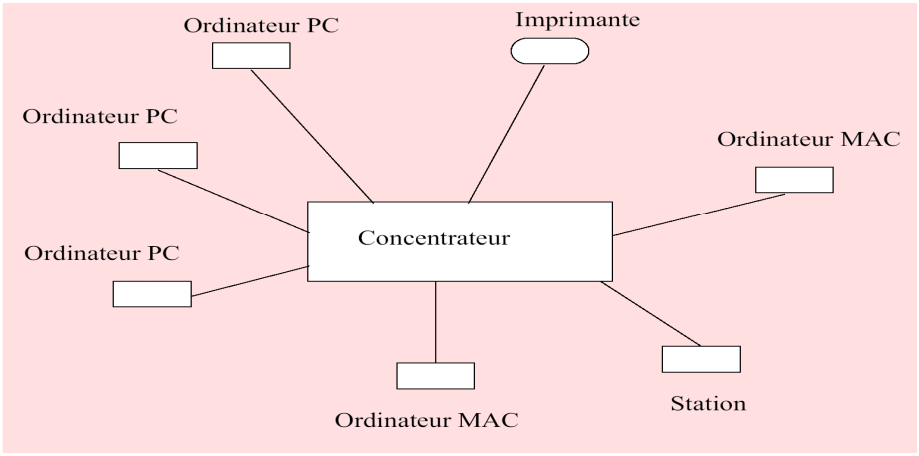
Avantages.
Il n'y a qu'un seul avantage à utiliser cette technologie, mais il est de taille:
Après avoir vu les divers constituants, il devient évident que ce procédé est peu coûteux, facile et rapide à mettre en œuvre.
Inconvénients.
Ils sont hélas nombreux:
Lorsque le réseau dépasse les dimensions d'une pièce, il faut alors passer les murs, ce qui "fige" considérablement la topologie et diminue les possibilités d'extension.
Si un défaut de connectique apparaît, c'est tout le réseau qui devient inopérant. En effet, tout se passe alors comme si l'on avait deux réseaux, mais chacun d'eux ayant une extrémité non adaptée. Plus rien ne fonctionne et le défaut n'est pas toujours visible. Les investigations sont longues et laborieuses
Conclusions.
Malheureusement, ce type de réseau est limité à 10 Mbits/s et n'a plus d'avenir, bien qu'encore suffisant pour un réseau domestique.
Il devient de plus en plus difficile de trouver ce genre d'adaptateur réseau. Les derniers modèles encore vendus sont souvent de type "combo", c'est à dire qu'ils permettent aussi bien un câblage coaxial en BUS qu'un câblage en étoile avec des paires torsadées, comme nous allons le voir tout de suite. Naturellement, un seul de ces deux modes est utilisable pour une interface donnée.

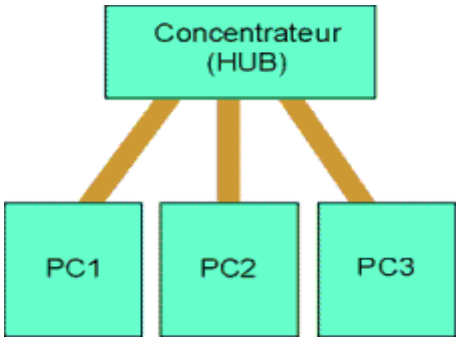
2.2) Câblage 10baseT

2.2.1) Principe

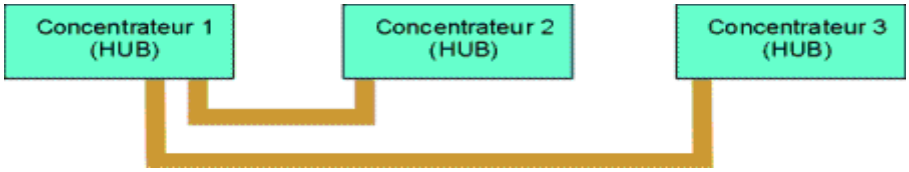
Chaque PC est relié par un câble constitué de 4 paires torsadées (dont deux seulement servent, normalement, l'une pour l'émission et l'autre pour la réception) à un concentrateur, encore appelé "HUB". Sa topologie est du type étoile

2.2.2) Topologie en étoile et concentrateur

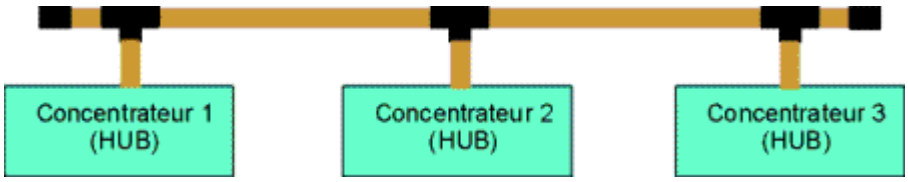




Les HUBS peuvent être "cascadés" en utilisant un média de type "paires torsadées": Normalement, il ne peut y avoir qu'un seul niveau de cascade. Autrement dit, sur l'illustration ci dessous, il ne devrait pas y avoir de hubs cascadés sur les hubs 2 et 3.

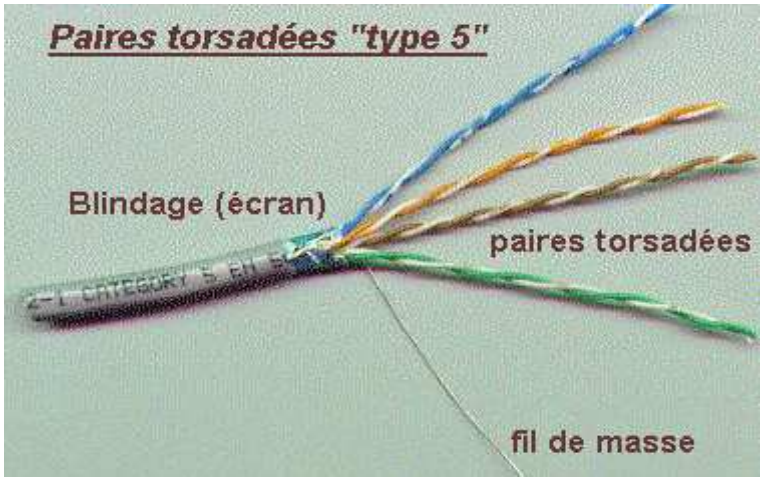


Ou montés sur un BUS, en utilisant un média de type "coaxial" ou "fibre optique":



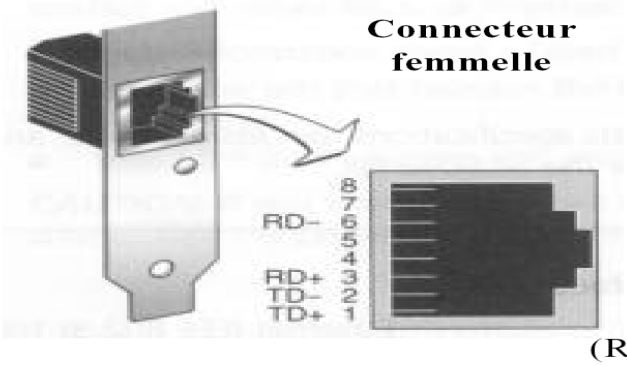
2.2.3) la connectique

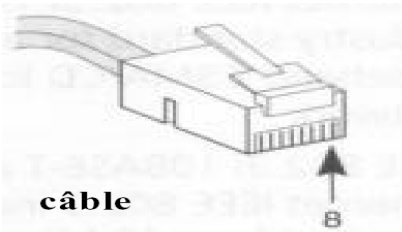
Le support est constitué de 2 paires de fils torsadés (twisted pairs), prolongés par des connecteurs d 'extrémité appelés **RJ45**.
Le câble de type 5 est constitué de 4 paires torsadées. il peut être blindé (écranté) ou non.
Le type 5 est certifié pour les réseaux 100 Mb/s. Le câble écranté offre une meilleure immunité au bruit électronique, il est à utiliser de préférence, même si son coût est plus élevé.



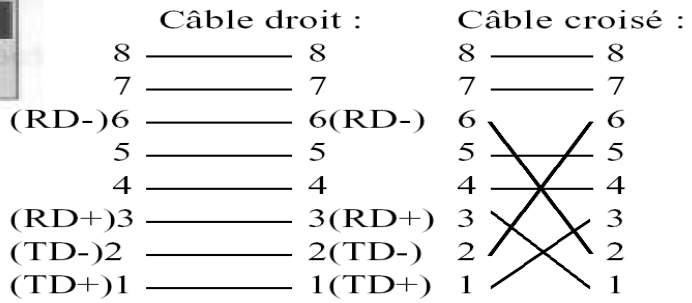
Ce type de câble est terminé par des connecteurs "RJ45".

Connecteurs RJ45





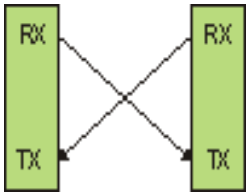
• Un câble droit connecte un terminal au hub ;
• Un câble croisé interconnecte 2 hubs



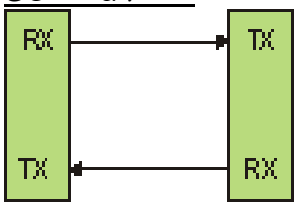
Deux équipements connectés doivent faire correspondre le TX (Emission) de l'un au RX (Réception) de l'autre. Normalement, il faudrait donc des câbles croisés. C'est ce qui est nécessaire si l'on souhaite relier directement deux PC entre eux.

Mais si l'on s'arrange pour que l'un des équipements ait sa prise déjà croisée, alors, il faut un câble droit. Les HUBS ont leurs prises croisées, c'est pour cela qu'il y a un X marqué sur ses prises. Il faut donc un câble droit pour connecter un PC à un HUB.

De PC à PC



De PC à HUB



2.2.4) Conclusion

Avantages.

D'un fonctionnement beaucoup plus sûr que le bus, si un lien vient à se rompre, seul le PC connecté par ce lien est absent du réseau.
Il est aisé d'ajouter des postes au réseau, même s'ils sont dans une pièce.
Cette technologie permet de réaliser un réseau 100 Mbits/s

Inconvénients.

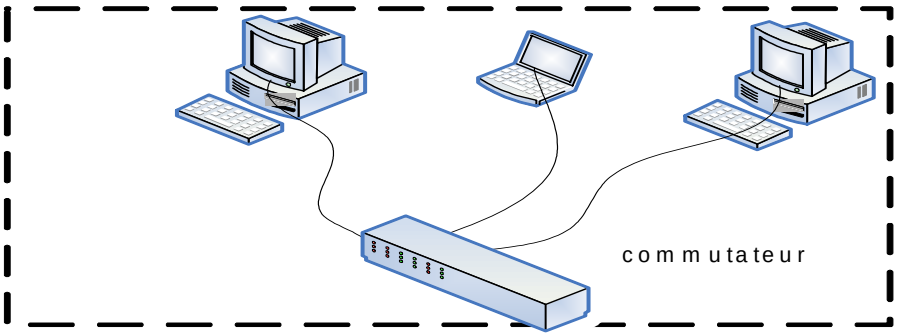
La longueur totale de câble mise en œuvre est importante.
Au voisinage du HUB, on obtient un faisceau de câbles imposant.
Le coût est tout de même plus élevé que dans une architecture BUS.

Conclusions.

L'exemple est fourni pour un petit réseau. Cependant, il existe des HUBS 100 Mb/s de 8, 12 ou 24 ports, que l'on peut monter en BUS sur une fibre optique, autorisant des réseaux à haut débit et de très grande taille.

2.3) Câblage 100baseT

FAst Ethernet ou 100BaseT est l'extension naturelle de 10BaseT et s'appuie sur le même protocole CSMA/CD, permettant ainsi aux données de transiter d'un noeud réseau à 10Mbps vers un nœud à 100Mbps sans conversion de protocoles. C'est la solution la plus économique et facile à mettre en place lorsqu'on souhaite faire évoluer le réseau Ethernet vers les hauts débits. Le matériel utilisé reste très similaire, le HUB peut être remplacé par un switch de conception plus récente et plus évolué (voir dossier annexe).



2.4) GIGABIT ETHERNET : 1000BaseT

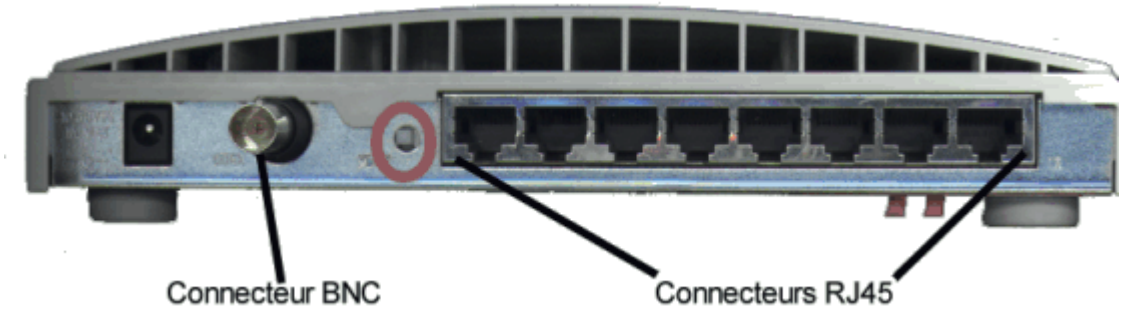
Si au départ, le gigabit utilisait une connexion en fibre optique, elle est remplacée par une connexion de type RJ45 de classe 5e (avec une limitation de distance limitée à 100 mètres). Le gigabit utilise le même format de trames de données que le 10 Base –T et le 100 Base TX et le même protocole anticollision, à savoir le CSMA-CD (Carrier Sense Multiple Access with Collision Avoidance).

2.5) Résumé

Médias	Vitesse max	Distance max
Câble coaxial 50 ohms - 10 Base 5 - "Ethernet épais"	10 Mbps	500 m
Câble coaxial 50 ohms - 10 Base 2 - "Ethernet fin"	10 Mbps	200 m
Paires Torsadées Non Blindées (UTP) – Catégorie 5 10 Base T	10 Mbps	100 m
Paires Torsadées Non Blindées (UTP) - Catégorie 5 100 Base TX	100 Mbps	100 m
Paires Torsadées Non Blindées (UTP)- Catégories 5 ou 6 - 1000 Base TX	1000 Mbps	100 m
Fibre Optique multimode – 100 Base FX	100 Mbps	220 m
Fibre Optique multimode – 1000 Base FX	1000 Mbps	220 m
Fibre Optique multimode – 1000 Base SX	1000 Mbps	550 m
Fibre Optique monomode – 1000 Base LX	1000 Mbps	5000 m
Câble coaxial 50 ohms - 10 Base 5 - "Ethernet épais"	10 Mbps	500 m
Câble coaxial 50 ohms - 10 Base 2 - "Ethernet fin"	10 Mbps	200 m

3) Les concentrateurs ou HUB

Il permet de relier différents éléments informatiques **sur la même branche d'un réseau** (voir ci-dessous)
Voici un exemple de HUB 10 Mb/s: Il possède huit ports de type RJ 45 et un port BNC, permettant un montage sur un BUS RG58. De marque "3COM" (sans aucune intention publicitaire).



Notez le petit bouton en face arrière qui nécessite quelques explications. Nous avons vu que pour connecter deux équipements entre eux, il faut faire correspondre le RX de l'un au TX de l'autre, raison pour laquelle les prises des HUBs sont croisées et marquées d'un X.
Seulement voilà, lorsque l'on cascade des HUBs avec de la paire torsadée, si les deux prises sont croisées, ça ne va plus parce que les deux croisements s'annulent. Sur ce modèle de HUB, il y a une prise particulière que l'on peut à volonté croisé ou non au moyen de ce petit poussoir.
Normalement, tout HUB dispose d'un moyen d'être cascadé et offre donc une prise non croisée. Sur les modèles d'entrée de gamme, il y a une prise dédiée non croisée. Sur d'autres modèles, il y a un port disposant de deux prises: L'une croisée, l'autre non. Dans ce cas, il ne faut pas oublier que les deux prises étant reliées au même port, une seule des deux peut être utilisée.
La face avant de ce HUB donne des informations concernant l'état de chaque port, la densité du trafic sur le réseau et même les "collisions" (cas ou 2 cartes réseaux sont en mode émission au même moment sur le réseau).



Les Hubs sont utilisés en Ethernet base 10 et base 100. L'Hub est le concentrateur le plus simple. Ce n'est pratiquement qu'un répétiteur (c'est son nom en Français). Il amplifie le signal pour pouvoir le renvoyer vers tous PC connectés. Toutes les informations arrivant sur l'appareil sont donc renvoyées sur toutes les lignes.
Des concentrateurs plus performants appelés "switches" permettent une meilleure répartition de la bande passante du réseau et augmentant encore les capacités de ce dernier.

4) Les répéteurs

Le répéteur est un équipement qui permet d'outrepasser la longueur maximale imposée par la norme d'un réseau. Pour se faire il amplifie et régénère le signal électrique. Il est également capable d'isoler un tronçon défaillant (Câble ouvert par exemple) et d'adapter deux médias Ethernet différents. (Par exemple 10base2 vers 10baseT). Cette dernière utilisation qui est la principale actuellement.

5) Le commutateur ou Le switch

Un switch reconnaît les différents PC connectés sur le réseau. En recevant une information, il décode l'entête et l'adresse physique pour connaître le destinataire et ne l'envoie que vers celui-ci dans le cas d'une liaison PC vers PC. Ceci réduit le trafic sur le réseau complet. Au contraire, dans le cas d'un HUBs, les informations circulent toutes sur tout le câblage RJ45 et donc vers toutes les stations connectées. Un switch travaille donc sur le niveau 1, 2. Les switches remplacent de plus en plus les HUBS. Les prix deviennent pratiquement équivalents.
D'aspect extérieur, il est équivalent à un HUBS.



Certains switches sont configurables grâce à un port console (Connecteur DB9 RS232)

6) Le routeur

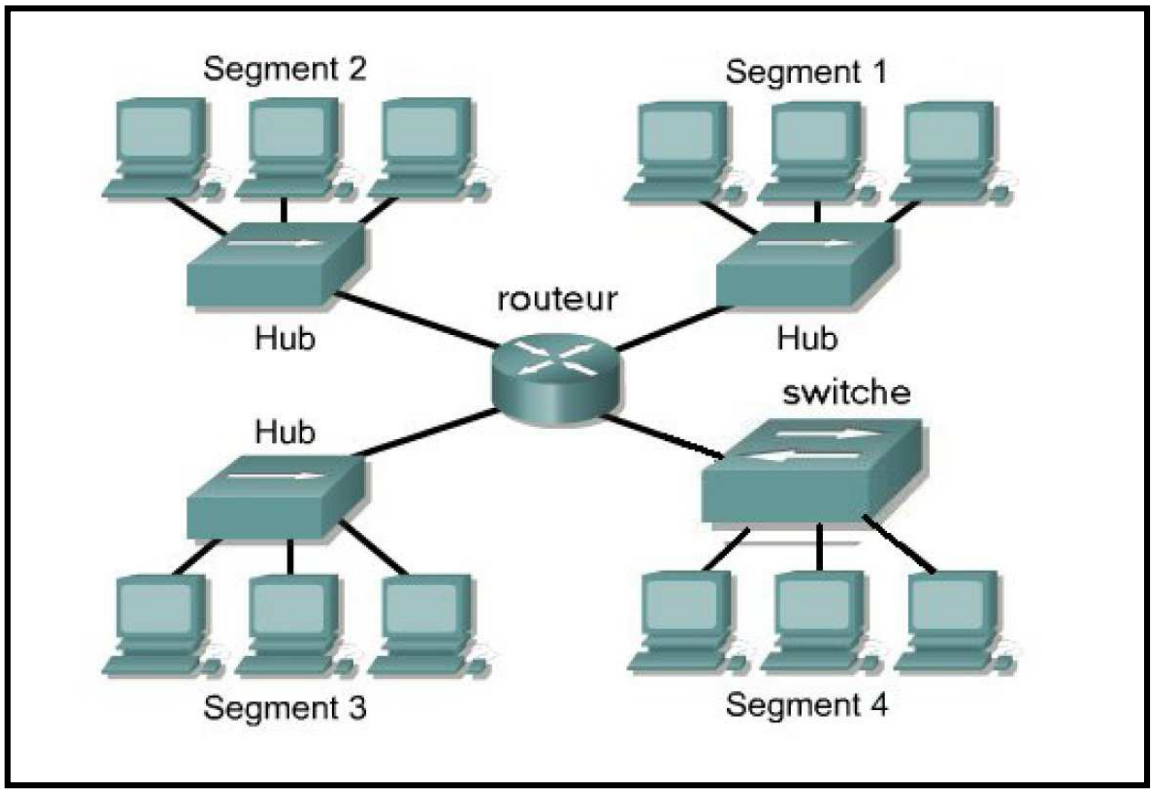
6.1. LES ROLES DU ROUTEUR.

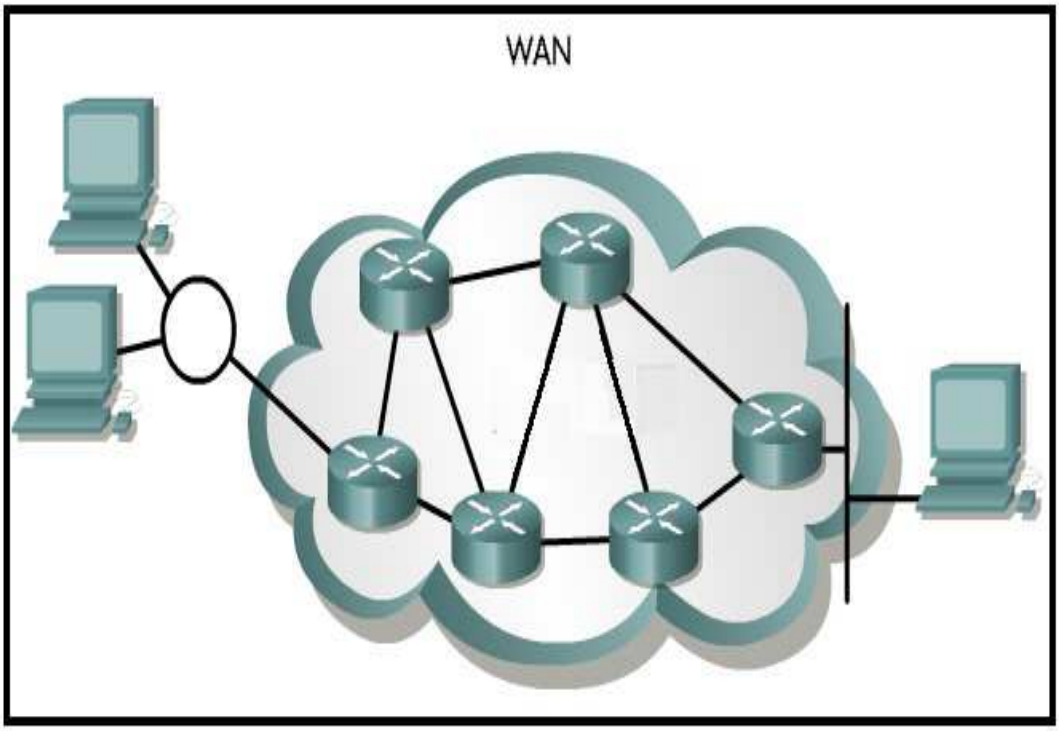
6.1.1. Place du routeur dans le modèle OSI

A l' origine, le **routeur** se situait uniquement au niveau de la couche 3 (réseau) du modèle OSI. A présent, avec ('intégration de la technologie, il **couvre les couches 3 a 7**. A la différence du **HUB qui repose sur la couche 1 (physique)** et du **SWITCH qui repose sur la couche 2** (liaison).Ces deux derniers matériels servant uniquement dans la conception de RLE (ici de *type* Ethernet).

6.1.2. Relier des réseaux entre eux et différences avec le HUB et le SWITCH.

Le premier rôle du routeur est de relier des réseaux (LAN comme WAN) entre eux afin de partager ou de faire circuler de l'information. C'est ici que la différence entre l'utilisation de routeur, HUB et SWITCH se fait : le routeur peut relier des RLE composés de HUB et SWITCH, alors que ces derniers relient des machines informatiques (PC, Serveurs, SWITCH, HUB, Sonde, Imprimantes réseau, etc. ...) afin de créer des RLE. Donc, un routeur ne peut entrer dans la composition seule d'un RLE.





6.1.3. Sécurisation

Aujourd'hui, l'administrateur réseau ne peut négliger la sécurisation de son réseau. Cela est encore plus vrai lorsque ce dernier doit être intégré dans un Intranet. Actuellement, les routeurs permettent de mettre en place des sécurités à certains niveaux :

6.1.3.1. Sécuriser l'accès au RLE

Dans un premier temps, L'administrateur veut protéger son RLE contre toute sorte d'intrusion, mais ne veut pas empêcher les informations de transiter par le routeur attenant à son RLE. Lors de la programmation du routeur, il va, à l'aide de **listes d'accès ou access-lists,** interdire l'accès au RLE à plusieurs niveaux, en fonction des couches sur Lesquelles repose le routeur :

- **Au niveau de la couche 3 du modèle OSI se trouve l'adresse logique.** L'administrateur va définir dans une **liste d'accès,** une, plusieurs, voire des plages d'adresses qui ont ou n'ont pas le droit de venir sur son réseau.

. **Au niveau de la couche application de l'empilage IP** (qui regroupe les couches 5 à 7 du modèle OSI), se trouvent **les ports logiques.** Derrière ces ports logiques peut se trouver une application. Par exemple, au port 80 correspond http, et au port 23 correspond le TELNET. Ce qui veut dire qu'une adresse ne faisant pas parti du RLE est peut-être autorisée à venir surfer sur son serveur par le biais du port 80, mais pas pour administrer le port telnet 23. Dans ce cas, l'administrateur va définir dans un autre type de **listes d'accès,** les applications par Lesquelles certaines adresses sont autorisées à accéder à son réseau.

6.1.3.2. Sécuriser l'information circulant entre deux routeurs.

L'information qui circule entre deux routeurs, sur un WAN en l'occurrence, n'est pas protégée. C'est à dire qu'elle circule en clair. Ainsi, quelqu'un de mal intentionné peut recueillir des informations vitales en vue d'attaquer un réseau, soit de faire de l'espionnage. II est ici possible de masquer ces informations. Le but n'est pas de faire croire que rien ne circule, mais d'empêcher toute utilisation du trafic. Pour ce faire, I' administrateur va créer, par le biais de deux interfaces virtuelles, un réseau virtuel appelé **tunnel.** En fait, le paquet IP va être encapsulé dans un autre paquet IP. Cela a pour résultat de rendre illisible le paquet IP d'origine. En plus de cette manipulation, le tout sera chiffré à l'aide d'un algorithme.

6.2. LE ROUTAGE

6.2.1. Types de routage il existe deux types de routages :

- **Le routage statique ;**
- **Le routage dynamique.**

Le routage statique a pour avantage :

- d'être é**crit une fois pour toute ;**
- d'être **sécurisant** (car aucune table de routage n'est échangée sur le réseau) ;

IL a pour inconvenient:

- d'être **lourd dans sa mise en œuvre** (quand il y a beaucoup de routeurs) ;

  De ne pouvoir dérouter le trafic en cas de rupture d'une liaison (utilisation d'une seule route) ;

Le routage dynamique a pour avantage d'être flexible :

- dans la mise en œuvre de nombreux routeurs ;
- en cas de rupture de liaison (reroutage du trafic de manière automatique) ;

Le routage dynamique a pour inconvénient :

- de ne pas titre sécurisé (bavard sur le réseau)

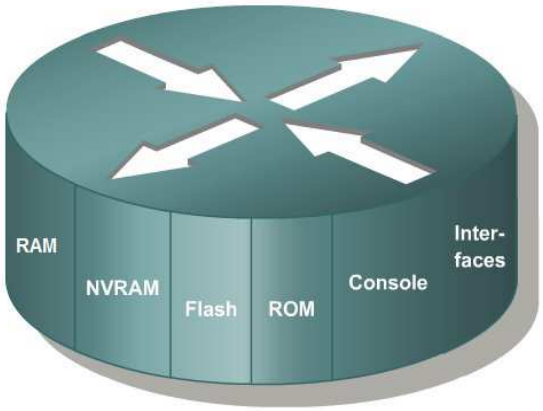
6.2.2. Le rôle du routage

Le routage permet de définir le chemin par lequel doit passer l'information pour atteindre son destinataire, en allant d'un réseau vers un autre. IL y a ici la notion d'échange d'informations entre des machines appartenant à des réseaux différents.

Pour cela, il faut créer des **tables de routage** qui sont remplies ainsi : le réseau de destination, via l'interface du routeur en vis-à-vis.

6.3. L'ARCHITECTURE INTERNE DU ROUTEUR.

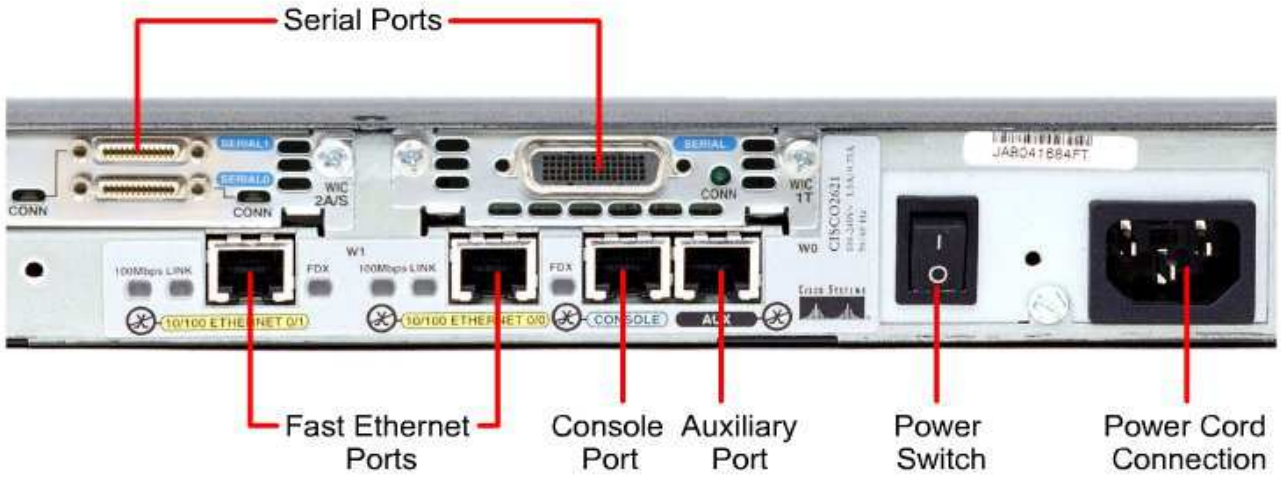
Le routeur contient quatre types de mémoire



ROM : Contient le « bootstrap » pour le démarrage du routeur et un noyau IP.
FLASH : Mémoire non volatile. Elie contient l'IOS (Internetworking Operating System) du routeur qui est le système d'exploitation (SE ou OS) du routeur. IOS est exécuté en flash.
NVRAM : Contient la configuration du routeur (suite d'instructions ASCII).
RAM : Contient les instructions (Routage - Filtres - NAT ...) exécutées par le routeur dans sa phase opérationnelle. C'est dans cette mémoire qu'est écrit dans un premier temps la configuration. Son contenu est volatile. Mais il peut être sauvegarde dans la NVRAM.

6.4. LES PORTS PHYSIQUES DU ROUTEUR.

Pour communiquer, le routeur utilise trois ports physiques différents. Ces ports sont des interfaces qui appartiennent chacune à un réseau différent. A ce titre, elles possèdent chacune une adresse appartenant au réseau sur lequel elles sont connectées.



6.4.1. Les interfaces

6.4.1.a) Le port Ethernet.

C'est par ce port que le LAN (RLE) est relié au routeur et accède ainsi à d'autres réseaux.

6.4.1.b) Le port série

IL permet l'interconnexion entre routeur. Cette interconnexion est considérée comme étant un WAN.
Contrairement au réseau Ethernet ou l'horloge est donnée par la bande de base, sur une liaison série il faut déterminer un DCE qui donne cette horloge. Cela ce fait au niveau du câble série. Donc l'interface série d'un routeur est DCE en fonction du câble qui lui est connecté. Ce dernier est câblé en DCE. En conséquence, l'interface en vis-à-vis est automatiquement DTE, et possède un câble DTE.

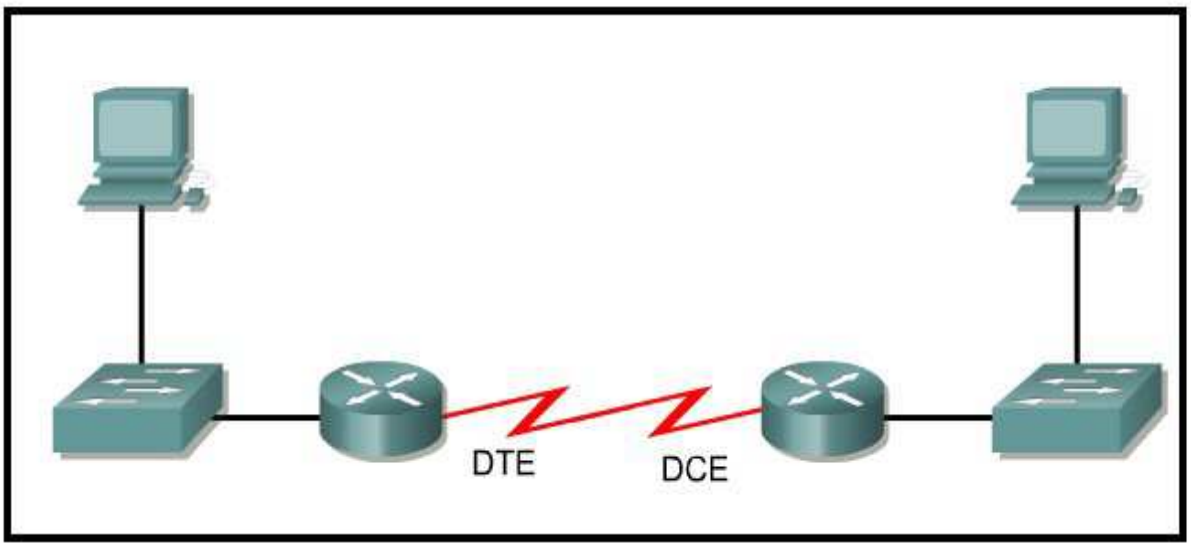
6.4.2). Le port console.

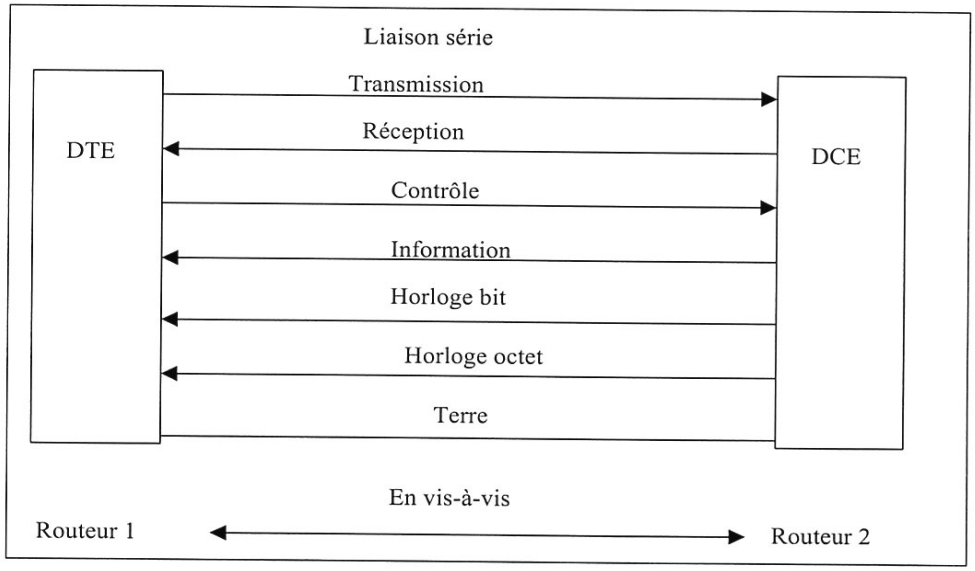
IL permet a L'administrateur de programmer, configurer, administrer le routeur, via le port COM d'un PC.

6.4.3) Spécificité du port série : L'horloge

Dans un RLE, en couche 1 du modèle OSI l'**horloge** est retrouvée dans le signal : la bande de base.

Dans un WAN, le signal qui circule est de la modulation. Donc, il faut qu'un élément sur la liaison série donne l'horloge afin de synchroniser les machines. Cet élément est le DCE (data computer equipment). II faut donc déterminer qui, physiquement, sur la liaison série est DCE, et donc son vis-a-vis est DTE (data terminal equipment). L'horloge est donnée par la norme fonctionnelle en couche. Celle utilisée dans ce cours est en générale la norme X21.





6.5. Mise EN OEUVRE ET EXPLOITATION DU ROUTEUR

6.5.1. Les câbles

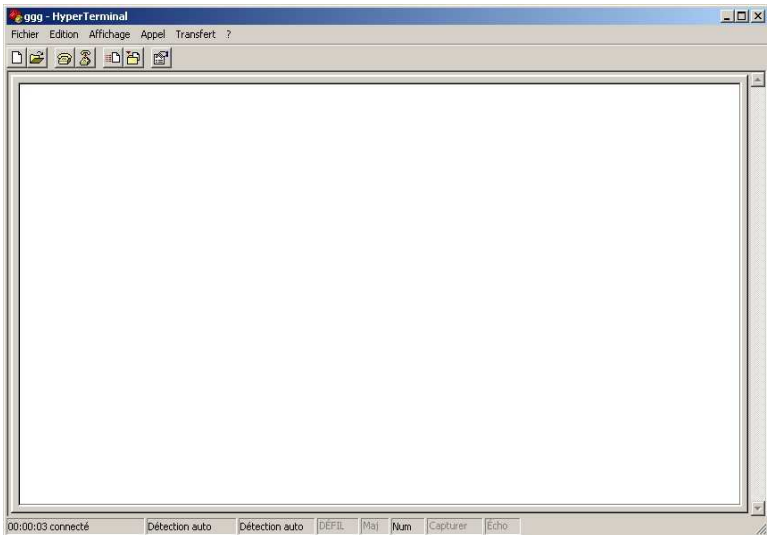
Pour relier les interfaces sur leur réseau respectif, il faut utiliser des câbles.
L'interface Ethernet utilise un câble avec connectique en RJ45, de type 10 base T, 100 base T ou encore Giga Ethernet. .

L'interface série utilise un câble série normalise X21. La connectique est différente suivant le type de routeur

6.5.2. Les outils de programmation

II existe plusieurs outils permettant de programmer un routeur. Dans le cadre de cette formation, seuls deux outils sont utilisés. L'outil est différent en fonction du moment et du lieu ou se trouve L'administrateur :

Premier outil : **Hyper terminal.**

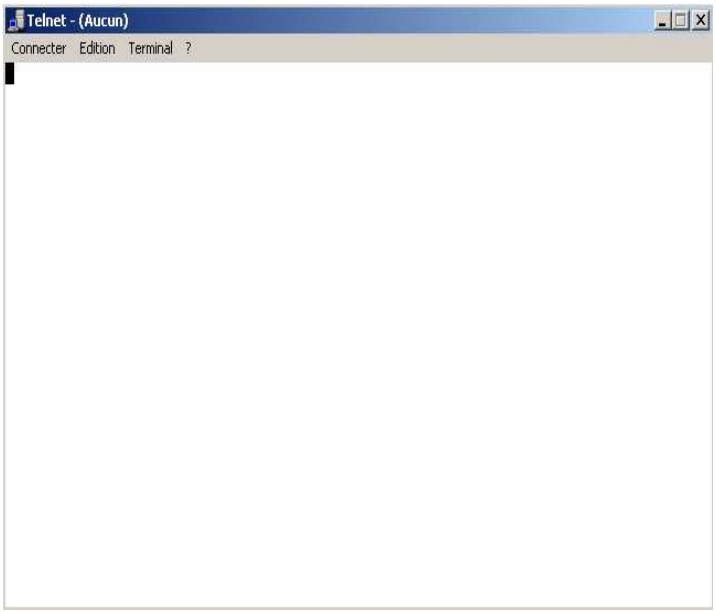


Cet outil est utilisé :

- lorsque le routeur est vierge de toute configuration
- lorsque l'administrateur ne passe pas par le réseau, quand le routeur est configuré.

Cet outil utilise la liaison port Com du PC - port console du routeur. C'est une liaison locale de quelques mètres seulement.

Deuxième outil : **TELNET**.



Cet outil est utilisé:

- Uniquement si le routeur est déjà configuré ;
- En local ou a distance.

Cet outil utilise le réseau et donc les interfaces réseau du PC et du routeur. Cette liaison peut être de quelques mètres comme de plusieurs centaines de kilomètres (via un réseau comme Internet, par exemple).

CONCLUSION

IL est maintenant possible de bien situer le routeur, le Hub et le Switch dans le réseau, ainsi que leurs différentes utilisations. La compréhension des rôles du routeur et de son architecture permettent au lecteur de mieux appréhender la mise en œuvre et l'utilité de ce matériel dans un réseau.
Le routeur est bien l'élément qui reliera des RLE entre eux, le tout formant un intranet.
Tout le contenu de ce cours est la base minimale à connaitre sur les routeurs. Cependant, ce type de matériel est plus complexe que cela, et peut donc permettre à l'administrateur de mettre en place dans son réseau d'autres spécificités en matière de sécurité comme de fonctionnalités ou d'outils.

CONTRÔLE ET COMMANDE A DISTANCE D'UN SYSTEME ASSERVI

L'interface des sockets

L'interface des sockets .....
1. INTRODUCTION....74
2. LES SCHEMAS DE PRINCIPE....75
3. PRESENTATION DE TCP/IP....76
4. CREER UN SOCKET : SOCKET()....77
5. INITIALISER UN SOCKET : BIND()....78
6. INITIALISER UNE FILE D'ATTENTE : LISTEN()....79
7. ATTENDRE UNE COMMUNICATION : ACCEPT()....79
8. CONNECTER UN CLIENT AU SERVEUR : CONNECT()....80
9. RECEPTION DE DONNEES : READ()....81
10. EMISSION DE DONNEES : WRITE()....81
11. CLORE UNE COMMUNICATION : CLOSE()....82
12. VOUS AVEZ DIT DNS?....82
12.1 Recherche de machine par nom : gethostbyname()....83
12.2 Recherche de machine par adresse : gethostbyaddr()....83
13. ADRESSES MACHINES, ADRESSES RESEAUX?....84
13.1 Définitions....84
13.2 Conversion d'adresses : htons() -- htonl() -- ntohs() -- ntohl()....84
13.3 Les adresses IP : inet_addr() -- inet_ntoa()....85
14. CONCLUSION....86

1. Introduction

Le concept de socket est assez difficile à appréhender. C'est BSD qui l'a choisi pour accomplir les communications inter-processus (IPC). Cela veut dire qu'un socket est utilisé pour permettre aux processus de communiquer entre eux de la même manière que le téléphone nous permet de communiquer entre nous. L'analogie entre le concept de socket et le téléphone est assez significative. Elle sera utilisé pour éclaircir la notion de socket.

Ainsi pour recevoir des appels téléphoniques, vous devez d'abord installer le téléphone chez vous. De la même façon, vous commencez par créer un socket avant d'attendre des demandes de communications. La commande `socket()` est alors utilisée pour créer un nouveau socket (que nous désignerons *master*).

Seulement il faut créer un socket avec les bonnes options. Vous devez spécifier les type d'adressage du socket, le numéro de port et surtout le type de socket que vous voulez utiliser. Nous allons détailler ces points un peu plus tard.

Ensuite, de la même façon qu'on vous attribue un numéro de téléphone pour recevoir vos appels, on doit spécifier au socket une adresse à laquelle il doit recevoir les messages qui lui sont destinés. Ceci est réalisé par la fonction `bind()` qui associe un numéro au socket.

Les sockets du type que nous allons utiliser (`SOCK_STREAM`) ont la possibilité de mettre les requêtes de communication dans une file d'attente, de la même façon que vous pouvez recevoir un appel pendant une conversation téléphonique. C'est la fonction `listen()` qui permet de définir la longueur de la queue d'attente. Il n'est pas indispensable d'utiliser cette fonction mais c'est plutôt une bonne habitude que de ne pas l'oublier.

L'étape suivante est d'attendre les demandes de communication. C'est le rûle de la fonction `accept()`. Cette fonction retourne un nouveau socket (désigné *slave*) qui est connecté à l'appelant. Le socket initial (*master*) peut alors se remettre à attendre les demandes de communication. C'est pour cette raison qu'on exécute un `fork()` à chaque demande de connexion dans le cas du modèle à accès concurrents. Dans le cas d'un serveur à accès unique, on ne se préoccupe plus du socket principal.

On sait maintenant comment créer un socket qui reçoit des demandes de communications, mais comment l'appeler?

Pour téléphoner vous devez d'abord avoir le numéro. Le rôle de la fonction `connect()` est de connecter un socket à un autre socket qui est en attente de demande de communication. Maintenant qu'une connexion est établie entre deux sockets, la conversation peut enfin commencer. C'est le rôle des fonctions `read()` et `write()`.

Enfin, à la fin d'une communication on doit raccrocher le téléphone ou fermer le socket correspondant. C'est la fonction `close()`.

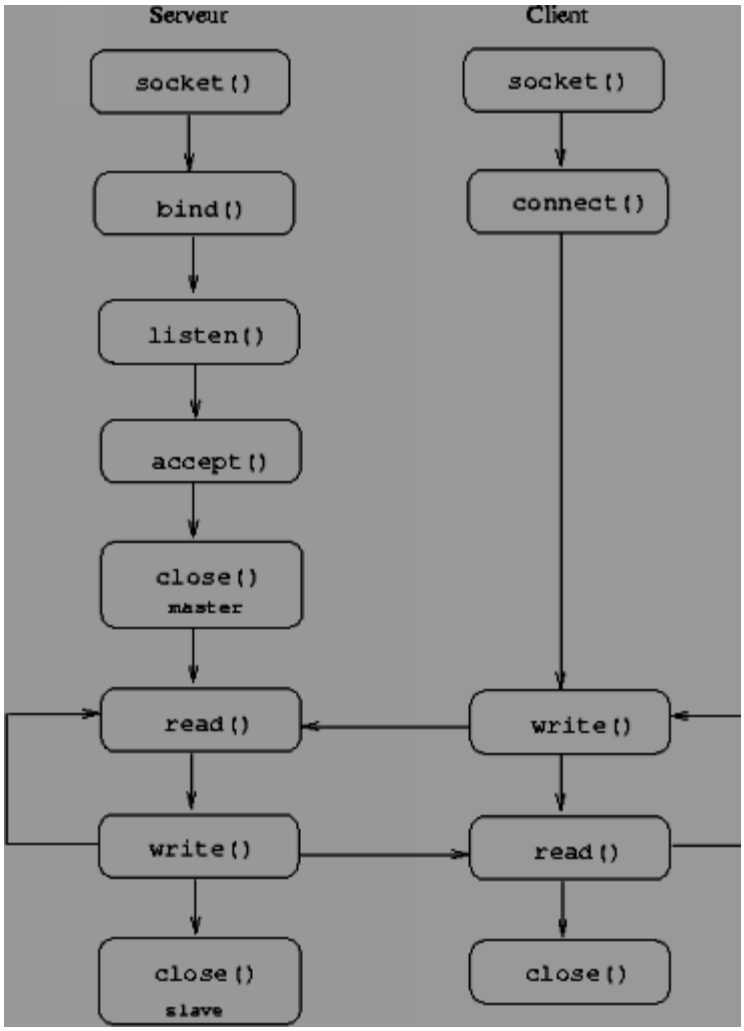
Il faut aussi noter qu'il existe deux types de connexion:

- Le mode connecté correspond au protocole `TCP`. Lorsque deux sockets sont en mode connecté, ce que l'on envoie à l'une est forcement reçu par l'autre. Le protocole se charge de vérifier l'intégrité de la communication et de renvoyer les données perdues.
- Le mode déconnecté correspond au protocole `UDP`. Ce protocole ne tient pas compte de la qualité de la liaison. C'est au programmeur de vérifier qu'aucune information n'a été altérée pendant la transmission

Dans la suite, nous ne nous intéresserons pas au protocole `UDP`. Nous citerons seulement les primitives qui permettent de l'utiliser.

2. Le schéma de principe

- Le modèle à accès uniques : le serveur attend une requête de connexion. Dès qu'un client se connecte, le serveur ferme le socket principal et ne s'intéresse qu'au socket connecté au client. Donc, avec ce modèle, on ne traite qu'un client à la fois. La figure schématise un tel serveur.



3. Présentation de TCP/IP

TCP/IP est sûrement le moyen de communication entre machines le plus utilisé dans le monde. Nous ne rentrerons pas dans les subtilités de ses protocoles. Nous donnerons seulement quelques notions pour que le lecteur puisse imager les propos qui suivront.

TCP/IP est un ensemble de protocoles organisés en couche. Les protocoles définissent un ensemble de commandes quune machine doit envoyer à une autre pour qu'elles se comprennent. En fait les protocoles supposent l'existence d'un moyen de communication fiable entre les deux machines.

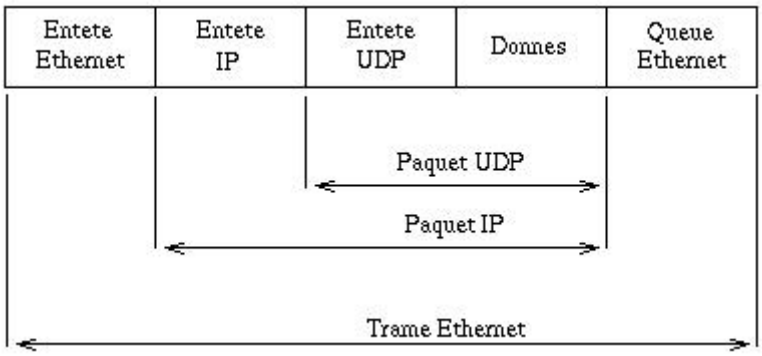
Pour envoyer un segment de données à une machine distante, il faut d'abord découper le segment en morceaux plus petits : des paquets. C'est la couche TCP ou *Transmission Control Protocol*, qui de chaque côté du fil découpe et recolle les morceaux.

D'autre part, les différents paquets ne sont pas obligés de se suivre et peuvent employer des voies différentes pour arriver à destination. C'est le protocole IP ou *Internet Protocol*, qui assure le voyage ou *routage* des différents paquets.

L'émission et la réception des paquets eux-mêmes sont gérées par le protocole UDP ou `User Datagram Protocol`. Ce protocole ne vérifie pas l'intégrité des données. Mais TCP, au recollage, fait des tests de validité et demande la retransmission des paquets douteux.

Enfin, le protocole ethernet permet de faire communiquer les machines entre elles au plus bas niveau : le matériel.

L'emploi de tous ces protocoles implique l'habillage ou *encapsulation* des données pour que l'information voyage sans se perdre dans le réseau des réseaux.



Nous voyons donc sur le schéma que seule une partie des informations des paquets est disponible à lutilisateur. Le reste étant réservé aux différents protocoles.

4. Créer un socket : `socket()`

Synoptique

```
#include <sys/types.h>
#include <sys/socket.h>

int socket(family, type, protocole)
int family;    /* détermine la famille du socket*/
int type;      /* détermine le type du socket*/
int protocole; /* option sur le mode de transfert*/
```

Description

Les tableaux et résument l'utilisation des variables `family` et `type`.

`family`	détermine la famille du socket
`AF_UNIX`	protocoles internes UNIX
`AF_INET`	protocoles internet
`AF_NS`	protocoles Xerox NS
`AF_IMPLINK`	protocoles spéciaux

`type`	détermine le type du socket
`SOCK_STREAM`	utilisé en mode connecté (TCP)
`SOCK_DTGRAM`	utilisé en mode déconnecté (UDP)
`SOCK_RAW`	??
`SOCK_SEQPACKET>`	??

L'argument `protocole` est généralement mis à `0` sauf dans certaines applications. Cependant, d'une façon générale on peut lui donner une autre valeur. Le tableau suivant résume l'utilisation de la variable `protocole`.

`protocole`		`type`
`0`	dans tous les cas	tous
`IPPROTO_UDP`	protocole UDP	`SOCK_DTGRAM`
`IPPROTO_TCP`	protocole TCP	`SOCK_STREAM`
`IPPROTO_ICMP`	protocole ICMP	`SOCK_RAW`
`IPPROTO_RAW`	protocole RAW	`SOCK_RAW`

L'appel système `socket()` retourne un entier dont la fonction est similaire à celle d'un descripteur de flux. Nous appellerons cet entier un descripteur de socket : `sockf}`.

Exemple

Création d'un socket pour communiquer en mode connecté via le réseau Internet.

```
int sockfd; /* descripteur de socket*/

...

if ((sockfd=socket(AF_INET,SOCK_STREAM,0))==-1)
{
```

```
        perror("Oops! Impossible de créer un socket");

        exit(1);

}
```

5. Initialiser un socket : `bind()`

Synoptique

```
#include <sys/types.h>
#include <sys/socket.h>

int bind(sockfd, myaddr, addrlen)
int             sockfd;  /* descripteur de socket*/
struct sockaddr *myaddr; /* structure d'adresse*/
int             addrlen; /* taille de la structure*/
```

avec:

```
struct sockaddr {
        unsigned short  sa_family;
        char            sa_data[41];
};
```

Description

L'appel à `bind()` se fait dans trois cas:

- Le serveur enregistre sa propre adresse auprès du système pour que tous les messages qui lui sont destinés lui parviennent.
- Un client enregistre une adresse spécifique pour lui même.
- Un client sans connexion doit s'assurer que le système lui a affecté une unique adresse que ses correspondants utiliseront afin de lui envoyer des message.

`bind()` renvoie `0` si tout se passe bien et `-1` en cas de problème.

Nous n'utiliserons que le premier point dans le cadre de ce projet.

Pour cela, il est décrit dans `<netinet/in.h>` deux structures simplifiant l'utilisation de `bind()`.

```
struct sockaddr_in { /* équivalent de struct sockaddr*/
        short          sin_family;  /* voir socket()*/
        unsigned short sin_port;    /* le port au format réseau*/
        struct in_addr sin_addr;    /* adresse*/
        char           sin_zero[8]; /* inutilisé*/
};

struct in_addr {
        unsigned long  s_addr;      /* adresse au format réseau */
};
```

Une simple conversion de cast transforme une `struct sockaddr` en une `struct sockaddr_in`.

Exemple

On suppose que `sockfd` est le descripteur créé par l'exemple de `socket()`. Il est donc de type `SOCK_STREAM` et de famille `AF_INE`T. Pour initialiser ce socket :

```
struct sockaddr_in sockadr          /* structure d'adresse associée*/
                                    /* au socket sockfd*/
...
sockadr.sin_family=AF_INET;         /* comme a l'appel socket()*/
sockadr.sin_port=htons(1976);       /* port sur lequel fonctionne le*/
```

```
                                    /* serveur. Nous verrons plus*/
                                    /* tard l'utilisation de htons()*/
sockadr.sin_addr.s_addr=INADDR_ANY; /*informonsle systeme qu'un*/
                                    /* serveur attend des connexions*/
if (bind(sockfd,(struct sockaddr*)&sockadr,sizeof(sockaddr_in))==-1) {
        perror("Oops! Impossible d'initialiser la socket");
        exit(1);
}
```

Note

Les format réseaux seront décrit un peu plus tard. Sachez que la primitive `htons()` est dans cette exemple, le moyen simple de faire la conversion entre le format interne de la machine hûte et le réseau. `INADDR_ANY` est décrit dans `<netinet/in.h>`. Cette constante informe le système qu'un serveur tourne sur la machine et que toute connexion sur le port concerné doit être lui transmis.

6. Initialiser une file d'attente : `listen()`

Synoptique

```
#include <sys/types.h>
#include <sys/socket.h>

int listen(sockfd, backlog)
int sockfd;  /* descripteur de socket*/
int backlog; /* longueur de la file d'attente*/
```

Description

L'appel à `listen()` initialise la queue d'attente pour les requêtes de connexion non encore traitées. L'argument `backlog` spécifie la longueur de cette file d'attente. Il semble que le nombre de place de cette file soit limité de façon implicite à `20` sur la plupart des systèmes. Cela signifie, que le serveur pourra assurer le traitement de `20` requêtes intervenant exactement au même instant.

`listen()` renvoie `-1` en cas d'echec ou `0` si tout va bien.

Exemple

On aura pour notre serveur les lignes suivantes : (on n'a évidemment pas d'équivalent pour le client)

```
if (listen(sockfd,5)<0) {
         perror("Oops! Impossible de creer la file d'attente");
         exit(1);
}
```

Note

Il faut bien faire la différence entre file d'attente et nombre de clients connectés au serveur. En effet, un client fait d'abord une requête de connexion auprès du serveur. Il rentre de ce fait dans la file d'attente. Dès qu'il obtient la connexion, il sort de la file d'attente mais reste connecté au serveur.

7. Attendre une communication : `accept()`

Synoptique

```
#include <sys/types.h>
#include <sys/socket.h>

int accept(sockfd, peer, addrlen)
int             sockfd;   /* descripteur de socket*/
```

```
struct sockaddr *peer;    /* struct d'adresse*/
int             *addrlen; /* longueur de cette structure*/
```

Description

L'argument `sockfd` désigne le descripteur de socket sur lequel seront attendues les connections.

Lorsqu'une requête arrive, le système enregistre l'adresse du client dans la structure `*peer` et la longueur de l'adresse dans `*addrlen`. Il crée alors un nouveau socket connecté avec la destination spécifié par le client, et renvoie à l'appelant un descripteur de socket. Les échanges futurs avec ce client se feront donc par l'intermédiaire de ce nouveau socket.

Le socket `sockfd` n'a donc pas de destination et reste ouvert pour accepter de nouvelles demandes de communications.

Tant qu'il n'y a pas de connexion, le serveur reste bloqué sur l'appel à `accept()`}.

Exemple

Pour un serveur qui attend des connexions, on peut avoir:

```
struct sockaddr_in client_adr; /* structure d'adresse du client*/
...
if ((slave=accept(sockfd,(struct sockaddr*)&client_adr,sizeof(sockaddr_in)))<1) {
         perror("Oops! Impossible d'etablir la connexion");
         exit(1);
}
/*
 * on récupere un descripteur de socket 'slave' connecté sur le client
 * et aussi l'adresse du client, contenue dans 'client_adr'
 */
```

8. Connecter un client au serveur : `connect()`

Synoptique

```
#include <sys/types.h>
#include <sys/socket.h>

int connect(sockfd, servaddr, addrlen)
int             sockfd;    /* descripteur de socket*/
struct sockaddr *servaddr; /* adresse du serveur*/
int             addrlen;   /* longueur de la structure pécédente*/
```

Description

Un socket est initialement créé dans un état non connecté, ce qui signifie qu'il n'est associé à aucune destination éloignée. L'appel système `connect()` associe de façon permanente un socket à une destination éloigné et le place dans un état connecté.

`connect()` renvoie `-1` en cas d'echec et `0` si tout va bien.

Exemple

Dans notre client, on aura les lignes suivantes :

```
struct sockaddr_in serv_adr    /* structure d'adresse du serveur*/
...
serv_adr.sin_family=AF_INET;   /* comme a l'appel a socket()*/
serv_adr.sin_port=htons(1976); /* port sur lequel fonctionne*/
                               /* le serveur*/
```

```
serv_adr.sin_addr.s_addr=adrress_du_serveur_sur_32_bits_au_format_réseau;
if (connect(sockfd,(struct sockaddr*)&serv_adr,sizeof(sockaddr_in))<0) {
        perror("Oops! Impossible de connecter le serveur");
        exit(1);
}
```

Note

L'appel à `connect()` ne s'utilise qu'avec des sockets utilisables en mode connecté comme le type `SOCK_STREAM`.

9. Réception de données : `read()`

Synoptique

```
#include <sys/types.h>
#include <sys/socket.h>

int read(sockfd, buff, nbytes)
int  sockfd; /* descripteur de socket*/
char *buff;  /* zone de donnees tampon*/
int  nbyte;  /* taille de la zone de données*/
```

Description

De la même façon que l'on lit dans un fichier, on lit dans un socket. La seule différence majeure est que la connexion peut se terminer à tout moment. Il faut donc être attentif et vérifier régulièrement que client et serveur soient toujours dans l'état connecté.

`read()` renvoie `-1` en cas d'echec ou `0` si tout va bien.

Exemple

On suppose que `sockfd` soit un descripteur de socket dans l'état connecté:

```
char buff[256];
if ((length=read(sockfd,buf,sizeof(buff)-1))<0) {
                /* -1 pour pouvoir rajouter le '\0'*/
        perror("Connexion perdue");
        exit(1);
}
buff[length]=0; /* pour 'finir' la chaîne de caracteres*/
                /* et par sécurité*/
printf("Message recu:\n%s",buf);
```

Note

En cas de perte de connexion, on lira un `0` dans le descripteur de socket, puis des `EOF`.

Nous ne nous intéressons qu'à la fonction `read()`, sachez quand même qu'il existe d'autres fonctions comme `recv()` ou `readv()` qui permettent la gestion des données hors-bande (i.e. urgentes).

10. Emission de données : `write()`

Synoptique

```
#include <sys/types.h>
#include <sys/socket.h>

int send(sockfd, buff, nbytes)
int  sockfd; /* descripteur de socket*/
char *buff;  /* zone de donnees a envoyer*/
```

```
int  nbyte;  /* taille de la zone de données*/
```

Description

C'est la fonction symétrique de `read()`.

`write()` renvoie `-1` en cas d'echec et `0` si tout va bien.

Exemple

On suppose que `sockfd` est un descripteur de socket dans le mode connecté:

```
char buff[256];
buff="MESSAGE PIPO";
if (send(sockfd,buff,strlen(buff))<0) {
        perror("Connexion perdue");
        exit(1);
}
```

Note

Nous ne nous intéresserons qu'à la fonction **`write()`**, sachez quand même qu'il existe d'autres fonctions comme **`send()`** ou **`sendmsg()`** qui permettent la gestion des données hors-bande (i.e. urgentes).

11. Clore une communication : close()

Synoptique

```
int close(sockfd)
int sockfd; /* descripteur de socket*/
```

Description

L'appel à `close()` est une manière élégante de clore une communication entre deux sockets. Cependant il faut noter que `close()` n'interrompt pas la communication. Le système essaye de transmettre les données sockées dans les buffer du coté serveur ou client.

`close()` renvoie `-1` en cas de problème ou `0` si tout va bien.

Exemple

On a `sockfd` un descripteur de socket:

```
close(sockfd); /* fin de connexion*/
```

Note

On a vu qu'un descripteur de socket est à peu de choses près assez semblable à un descripteur de flux. C'est pour cette raison qu'il n'existe pas de fonction du style `close_socket()`, mais une fonction commune à tous les descripteurs de flux: `close()`. Pour un meilleur contrôle de la fin de connexion, voir l'appel à `shutdown()`page.

12. Vous avez dit DNS?

Un DNS ou *Domain Name Serveur*, est une méthode pour stocker l'adresse d'une machine, non pas par une suite de numéro, (par exemple `147.38.94.17`), mais avec une notation plus 'lisible' de la forme: machine.noeud.indicatif_de_pays (par exemple `morbier.imt-mrs.fr`). Pour convertir une adresse numérique en une adresse alphanumérique et vice-verça, deux primitives sont à la disposition du programmeur.

12.1 Recherche de machine par nom : gethostbyname()

Synoptique

```
#include <netdb.h>

struct hostent gethostbyname(hostname)
char *hostname;              /* nom en toute lettre de l'hûte*/
```

avec

```
struct hostent{
        char *h_name;         /* nom officiel de la machine*/
        char *h_aliases;      /* liste des surnoms de la machine*/
        int  h_addrtype;      /* type d'adressage (ex.AF_INET)*/
        int  h_length;        /* longueur de l'adresse*/
        char **h_addr_list;   /* liste des adresses de la machine*/
};                            /* au format NetworkByteOrder*/

#define h_addr h_addr_list[0] /* c'est l'adresse principale*/
                              /* d'une machine*/
```

Description

En général on se sert de cette fonction pour obtenir l'adresse d'une machine dont on connait le nom.

Exemple

```
struct hostent *h;
char host[]="maroilles.imt-mrs.fr";
struct sockaddr_in addr;
...
if((h=gethostbyname(host))==NULL) {
        perror("Impossible d'obtenir l'adresse IP");
        exit(1);
}
printf("Nom de l'hote: %s\n",h->h_name);
bcopy((char*)&h.h_addr,(char *)&addr.sin_addr.s_addr,h.h_length);
/*
 * on a addr.sin_addr.s_addr qui pointe sur l'adresse
 * de maroilles.imt-mrs.fr
 *
 */
```

Note

`bcopy()` est décrit dans `<bstring.h>`. Cette appel copie une zone de mémoire vers une autre. Cependant `<bstring.h>` n'est pas implémenté sur tous les systèmes d'exploitation. Vous trouverez en annexes les *listings* des fonctions de cette bibliothèque.

12.2 Recherche de machine par adresse : `gethostbyaddr()`

Synoptique

```
#include <netdb.h>
struct hostent gethostbyaddr(addr, length, type)
char *addr;  /* adresse de l'hote au format "xxx.xxx.xx.xx"*/
int  lenght; /* longueur de cette adresse*/
int  type;   /* type d'adresse (AF_INET par exemple)*/
```

Description

En général on se sert de cet appel pour connaitre le nom d'une machine dont on connait l'addresse.

Exemple

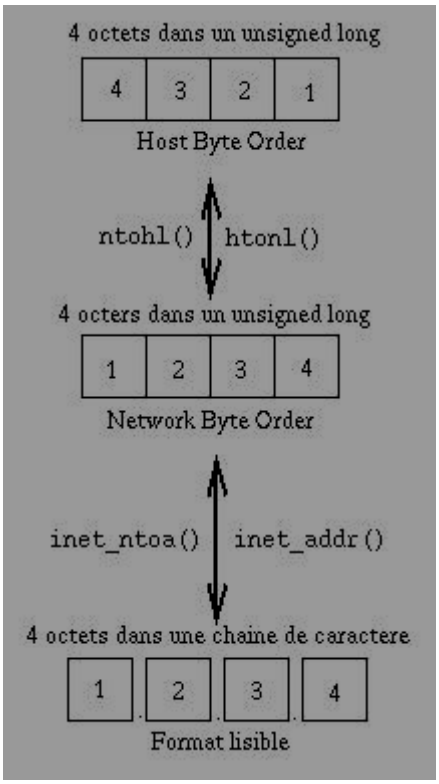
Supposons que l'on connaisse l'adresse IP d'une machine, ces quelques lignes permettent d'en connaitre le nom.

```
struct hostent client;
...
client=gethostbyname("147.94.38.17",sizeof("147.94.38.17"),AF_INET);
printf("La machine s'appelle %s\n",client.h_name);
```

13. Adresses machines, adresses réseaux?

13.1 Définitions

Les adresses sont stockées par la machine sous forme d'entier. Malheureusement, toutes les machines ne traitent pas ces entiers de la même façon que le réseau. Ainsi sur une machine, l'adresse est stockée sous la forme du *Host Byte Order* alors que le réseau attend du *Network Byte Order*. En fait certaines machines travaillent en *Network Byte Order* par défaut. Mais la portabilité étant le maître mot en UNIX, l'utilisation des commandes suivantes est vivement recommandée.



La figure schématise les différents formats d'adresse et montre les domaines d'applications des fonctions que nous détaillons dans les paragraphes suivants.

Note

Il n'y a aucun rapport avec le DNS ici.

13.2 Conversion d'adresses : `htons()` -- `htonl()` -- `ntohs()` -- `ntohl()`

Synoptique

```
#include <sys/types.h>
#include <netinet/in.h>

short htons(short int arg) /* Host TO Network Short*/
```

```
long  htonl(long int arg)  /* Host TO Network Long*/
short ntohs(short int arg) /* Network TO Host Short*/
long  ntohl(long int arg)  /* Network TO Host Long*/
```

Description

Ces fonctions assurent la conversion entre le format *Network Byte Order* et le format *Host Byte Order*.

Note

`sin_family` ne doit pas être converti à la notation réseau, au contraire de `sin_addr` et `sin_port`. En effet `sin_family` est juste utilisée par le noyau et ne voyage pas dans le réseau.

Exemple

Voir l'appel à `bind()` page : on convertit le numéro de port au format réseau.

13.3 Les adresses IP : `inet_addr()` -- `inet_ntoa()`

Synoptique

```
#include <sys/types.h>
#include <netinet/in.h>
#include <arpa/inet.h>

unsigned long inet_addr(addr)
char *addr; /* adresse au format "xxx.xxx.xx.xx"*/
```

Description

Cette primitive convertit une chaîne de caratères en une adresse au format réseau (*Network Byte Order*). On s'en sert pour l'appel à `bind()`. On évite ainsi l'emploi de `htonl()` et `htons()`.

Exemple

On veut connecter sock sur l'adresse `147.94.38.17`, une solution pour initialiser l'adresse du socket :

```
struct sockaddr_in sock;
...
sock.sin_addr.s_addr=inet_addr("147.94.38.17");
...
```

Synoptique

```
#include <sys/types.h>
#include <netinet/in.h>
#include <arpa/inet.h>

char *inet_ntoa(addr)
struct in_addr addr; /* adresse au format Network Byte Order*/
```

Description

Cette primitive a la fonction inverse de la précédente. Elle permet de rendre lisible une adresse au format réseau. Cette primitive se contente juste de convertir une adresse au format réseau au format dit IP. Elle ne fait en aucun cas appel au serveur de noms.

Exemple

Supposons que l'on veuille connaître la provenance d'un client. Le serveur pourra comporter les lignes suivantes.

```
struct sockaddr_in client; /* structure d'adresse d'un client */
...
printf("Connexion provenant du %s\n",inet_ntoa(client.sin_addr));
```

Note

L'appel à `htons()` ou `htonl()` est inutile si on utilise ces deux primitives.

14. Conclusion

L'interface des socket permet donc de faire communiquer des processus tournants sur deux ou plusieurs machines distantes. On pourait même imaginer deux programmes tournants sur une même plateforme communiquer grâce aux sockets. Cependant, le type de communication que nous voulons établir entre les processus de la machine serveur ne sont pas aussi simples. Nous ne pourons donc pas nous contenter des sockets pour faire communiquer tous les processus. De plus, la découverte d'autres moyen d'intercommunication est assez importante en elle-même pour que l'on s'interresse à elle.

1) TDN°1 : Etude préliminaire : Les sockets

a) Lire le dossier fourni sur les sockets

b) complétez le document réponse N°1 en remplissant les cases vides par le nom des fonctions correspondantes.

c) L'adresse d'une communication possède plusieurs informations réunit dans une structure (chaque champ de la structure représente une information). Complétez le document réponse N°2 en indiquant dans le cadre vide les différentes informations présentent dans une adresse.

d) Combien d'adresses différentes vont être gérés lors d'une communication client serveur, donnez le nom utilisé pour chacune des adresses ou structures utilisées dans le polycopié.

e) remplir le document réponse N°3

Document réponse N°1

Action réalisée	Fonction langage C
Mise en place du matériel pour établir la communication 	
Spécification d'une adresse pour la réception des messages 	
Définition de la longueur de la file d'attente des messages 	
Attendre les demandes de communication 	
Demande de communication et connexion 	
Début de la conversation 	
Clore la communication 	

Document réponse N°2





Document réponse N°3

Fonctions	Description	Paramètres d'entrées	Paramètres de sorties
Socket()			
Bind()			
Listen()			
Accept()			
Read()			

Write()			
Close()			
Connect()			

TD N°2 Le programme client tcpsend

On demande lors de ce TD de mettre en place un programme client sur la cible OS9 afin qu'il interroge un programme serveur présent sur un ordinateur PC

Procédure à suivre pour la mise en œuvre du programme client

1) Créez sur microware HAWK un nouveau projet tcpsend en incluant le fichier source tcpsend.c et les librairies suivantes (en respectant l'ordre) :
C:\Mwos\OS9\68020\LIB\socklib.l;
C:\Mwos\OS9\68020\LIB\os_lib.l;
C:\Mwos\OS9\68020\LIB\netdb.l;
C:\Mwos\OS9\68020\LIB\sys_clib.l;
C:\Mwos\OS9\68020\LIB\sclib.l;
C:\Mwos\OS9\68020\LIB\clib.l

2) Chargez le fichier code machine dans la cible OS9 et complétez le fichier source tcpsend.c

3) Chargez un fichier texte dans la cible OS9 en utilisant ws--->FTP32
démarrez---> programmes----> wsftp32
hostname ms9-targetx.egc.es.tpn3
puis double cliquez sur whatsnew.txt

4) Dans le répertoire réseau MP12 lancez le programme serveur.exe et notez l'adresse IP du PC

5) Lancez telnet et la commande tcpsend 192.9.xxx.xxx whatsnew.txt
ou 192.9.xxx.xxx et l'adresse réseau du PC notée précédemment

6) observez avec le programme réseau le transfert du fichier

7) Questions

on pourra faire une analyse de trame avec le programme finisar surveyor web demo

7.1) Comment simplement en analysant le fichier source tcpsend, peut on affirmer que c'est un programme client?

7.2) Dans quel type de trame circule le texte <whatsnew.txt>

```
/* <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< tcpsend.c >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>> */
/*
**      Syntax:    tcpsend <adresse IP > <file> programme client
**      Function: envoyer un fichier à travers un socket du type TCP programme client vers serveur
*/

/* Header Files */
#include <stdio.h>
#include <stdlib.h>
#include <types.h>
#include <errno.h>
#include <string.h>
#include <INET/socket.h>
#include <INET/in.h>
#include <INET/netdb.h>

/* Macro Definitions */

#define PORT_NUM  1200
#define INIT          77              /* commands */
#define DATA          78
#define END                   79

/* Type Definitions */
struct data {
        int code, count;
        char data[512];
};

/* Global Variables */

struct sockaddr_in ls_addr;/* paramètres de connexion vers le serveur */
char msgbuf[2048] = {0};

/* Function Prototypes */
void main(int argc, char* argv[], char* envp[]);

void main(int argc, char* argv[], char* envp[])
{
  int s;
        FILE *ifile;
        u_int32 n;
        u_int32 count = 10;
        struct hostent *host;

        /* initialisation des paramètres de connexion vers le serveur
        */
        /*************************************************************************************
        *************************************************************************************
        */
```

```
        ls_addr.sin_family = host->h_addrtype;
        ls_addr.sin_port =  htons(_____________________________);

        /*************************************************************************************
        *************************************************************************************
        */

        /* vérification du format de l'adresse IP du serveur
        tapez en ligne de commande mode MSDOS
        tcpsend 192.299.9.3 whatsnew.txt
        mise au bon format pour les paramètres de connexion au serveur
        structure struct sockaddr_in ls_addr;
        */
        /*************************************************************************************
        *************************************************************************************
        */

        if ((host = gethostbyname(argv[1])) != NULL) {
                memcpy(&ls_addr.sin_addr.s_addr, host->h_addr, host->h_length);
        } else {
                ls_addr.sin_addr.s_addr = inet_addr(_____________________);
        }

        /*************************************************************************************
        *************************************************************************************
        */
        /* ouverture du fichier
        tapez en ligne de commande mode MSDOS
        tcpsend 192.299.9.3 whatsnew.txt
        si erreur à l'ouverture ecrire
        can't open file
        */
        /*************************************************************************************
        *************************************************************************************
        */
                if ((ifile = fopen(argv[2],"r")) ==-1) {
                printf("can't open file '%s'\n", _____________________________);                exit(0);
        }

        /*************************************************************************************
        *************************************************************************************
        */

while (1) {

        /* Mise en place de la connexion
        s descripteur de socket
        family=AF_INET
```

```
        type=SOCK_STREAM
        protocole=0
        /*si impossibilité d'ouvrir le socket affichez à l'écran can't open socket
        fermez le fichiez et sortir du programme*/
        */
        /*****************************************************************************************
        *****************************************************************************************
        */

                if ((s = socket(_____________________________________)) == -1) {
                        printf("can't open socket\n");
                        close(s);
                        exit(0);
                }

        /* Connecter un client(le programme suivant) au serveur les paramètres de connexion sont contenus dans la
struture ls_addr
        s descripteur de socket
        si impossible de se connecteur au serveur affichez    "connect failed"
        attente de la connexion en imprimant le nombre de tentative de connexionsi connexion impossible après
l'attente affichez      fermez le socket et fermez le fichier
        */
        /*****************************************************************************************
        *****************************************************************************************
        */
                if (connect(s, (struct sockaddr*)&ls_addr, sizeof(ls_addr)) == -1) {
                        printf("connect failed");
                        close(s);
                        fclose(ifile);
                        exit(0);
                }
                break;
        }

        /*****************************************************************************************
        *****************************************************************************************
        */
        printf("Connection established\nSending file '%s'...", *argv); fflush(stdout);

        count = sizeof(msgbuf);
        if ((errno = fread(_____________________________________________)) >count) {
                printf("lecture impossible du fichier \n");
                close(s);
                fclose(ifile);
                exit(0);}
  else{
                if ((errno = write(_____________________________________)) == -1) {
                                printf("socket write error\n");
                                        close(s);
                                        fclose(ifile);
```

```
                                        exit(errno);}
        count = sizeof(msgbuf);
        }
        if (errno ==-1){
                printf("read error on file\n");
                exit(0);
        }

        close(s);
        fclose(ifile);
        printf("sent %u bytes\n", count);
        exit(0);
}
        /*********************************************************/
```

On donne ci-dessous le programme partiel du serveur de la cible OS9 que l'on trouve sur le réseau sous le nom de tcp_recv, complémentez ce programme afin qu'il affiche l'adresse IP de la cible OS9. Compilez et testez ce programme (on pourra utiliser le programme windows client.exe pour le tester)

```
/* <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< tcprecv.c >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>> */
/*      Syntax:    tcprecv <file>
**      Function:  receives a file across a stream (TCP) socket
*/

/* Header Files */

#include <stdio.h>
#include <stdlib.h>
#include <types.h>
#include <errno.h>
#include <string.h>
#include <signal.h>
#include "pit.h"

#include <INET/socket.h>
#include <INET/in.h>
#include <INET/netdb.h>

/* Macro Definitions */

#define PORT_NUM   1200

#define INIT        77          /* commands */
#define DATA        78
#define END         79

/* Global Variables */
char msgbuf[20480] = {0};
```

```
char *buff;
int sx;
_PIT_REG  *pit;

void main(int argc, char* argv[])
{
        unsigned char tab[12]="bonjour";
        unsigned char c;
        unsigned char reste=1;
        unsigned char quotient;
        int s;
        int i;
        int size;
        int totbytes = 0;
        u_int32 count = 1;
        struct sockaddr_in ls_addr;
        struct sockaddr_in to;
        setbuf(stdout,NULL);
        setbuf(stdin,NULL);

        /* création du socket pour la connexion avec les clients
        */
        /*************************************************************************************
        */
        if ((sx = socket(_____________________________________)) == -1) {
                printf("can't open socket\n");
          close(_____________________);
                exit(errno);
        }

        /* initialisation des paramètres de connexion du serveur
        */
        /*************************************************************************************
        */
        ls_addr.sin_family = AF_INET;
        ls_addr.sin_port = ____________________________________
        ls_addr.sin_addr.s_addr = 0;

        /* initialisation du socket
        */
        /*************************************************************************************
        */
        if (bind(sx,(struct sockaddr*)&ls_addr,sizeof ls_addr)== -1) {
                printf("can't bind socket\n");
          close(sx);
          exit(errno);
        }

        /* initialisation de la file d'attente une connexion possible
        */
```

```
/*****************************************************************************************
*****************************************************************************************
*/
if (listen(_____________________________) < 0) {
        printf("tcp_listen - failed!\n");
        close(sx);
        exit(errno);
}
        size = sizeof(struct sockaddr_in);

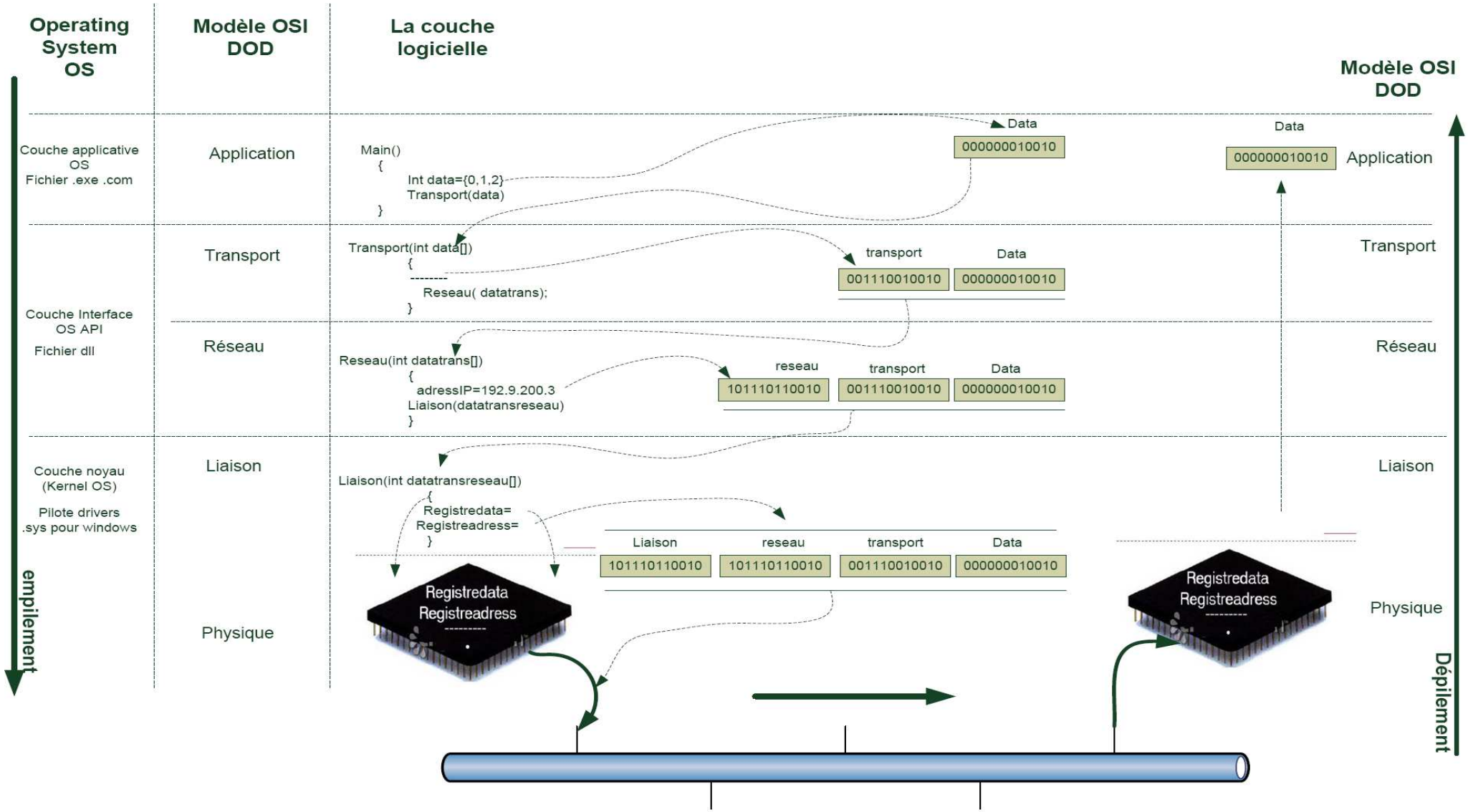
/* attendre la communication
*/
/*****************************************************************************************
*****************************************************************************************
*/
        if ((s = accept (sx,(struct sockaddr*)&to,&size)) < 0) {
        printf("can't accept\n");
        close(s);
        close(sx);
        exit(errno);
}
close(sx);
printf("connected to %s port %d\n",
        inet_ntoa(to.sin_addr), to.sin_port);

while (count) {
        count = (sizeof(msgbuf))-1;

/* Lecture des données issues du client
*/
/*****************************************************************************************
*/

                if ((errno =read(_____________________________________)) <0) {
                printf( "can't recv (cnt=%d)\n");
                close(s);
                exit(errno);
        } else if (count == 0) {
                break;
        } else {
                msgbuf[errno]=0;
                printf("message recu \n %s",msgbuf);
                write(_____________________________________________________);
                }
                totbytes += count;
        }
close(s);
printf("read %d bytes\n",totbytes);

exit(0);}
```


Operating System OS
Modèle OSI DOD
La couche logicielle
Modèle OSI DOD
Couche applicative OS
Fichier .exe .com
Application
Main()
{
Int data={0,1,2}
Transport(data)
}
Data
000000010010
Data
000000010010
Application
Transport
Transport(int data[])
{
Reseau(datatrans);
}
transport
Data
001110010010
000000010010
Transport
Couche Interface OS API
Fichier dll
Réseau
Reseau(int datatrans[])
{
adressIP=192.9.200.3
Liaison(datatransreseau)
}
reseau
transport
Data
101110110010
001110010010
000000010010
Réseau
Couche noyau (Kernel OS)
Pilote drivers .sys pour windows
Liaison
Liaison(int datatransreseau[])
{
Registredata=
Registreadress=
}
Liaison
reseau
transport
Data
101110110010
101110110010
001110010010
000000010010
Liaison
Registredata
Registreadress
Registredata
Registreadress
Physique
Physique
empilement
Dépilement