CHAPITRE 1

Introduction

L'univers des télécommunications numériques est composé de réseaux locaux - LANs (*Local Area Networks*) et de réseaux longue distance - WANs (*Wide Area Networks*).

Historiquement, les technologies du réseau local sont nées dans les années 80 avec la nécessité de connecter de multiples ordinateurs sur le même site. Ces technologies ont permis de lier plusieurs ordinateurs sur le même support physique. Traditionnellement, les réseaux locaux ont une couverture limitée à quelques kilomètres et un débit de quelques mégabits par seconde.

Un réseaux local peut être décrit à l'aide de plusieurs caractéristiques:

- le débit nominal (bits/s),
- la topologie (anneau, bus, étoile,..),
- la distance de couverture,
- la méthode d'accès au support commun,
- les media et la connectique (type de câbles, ..),
- le format de trames.

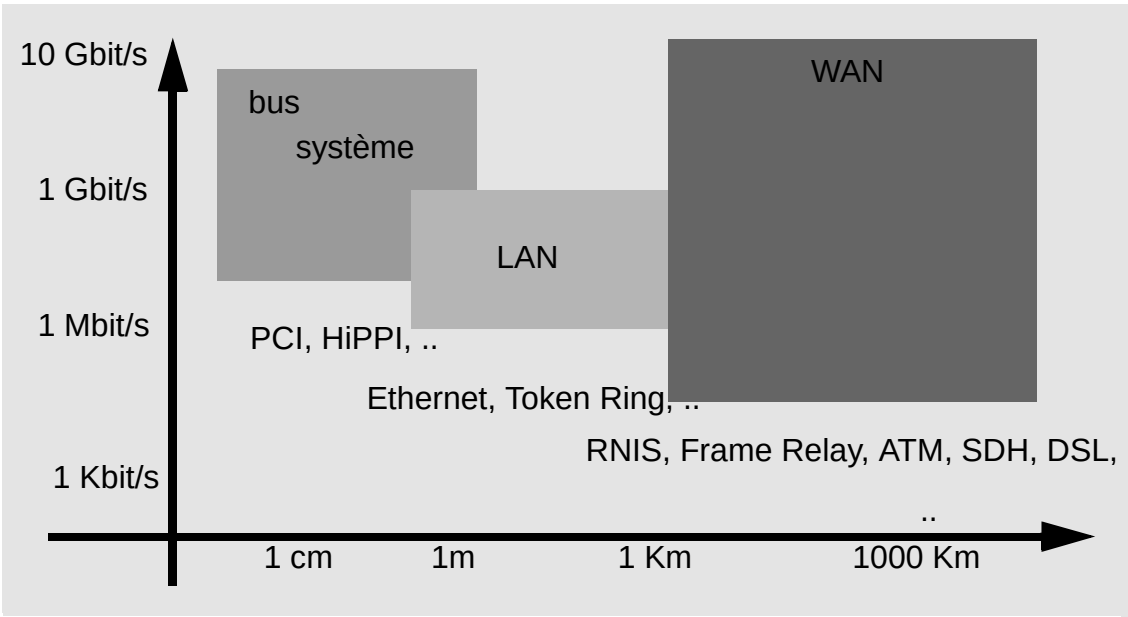
Les réseaux longue distance transportent des données numériques d'un site à l'autre. Les technologies WAN peuvent être déployées sur n'importe quelle distance à l'échelle nationale ou internationale.

L'ensemble des réseaux de télécommunication est basé sur une hiérarchie de communication synchrone SDH (*Synchronous Digital Hierarchy*). Parmi les technologies traditionnelles utilisées pour le transfert de données via le système SDH, on trouve des liaisons louées, des relais de trames (*frame relay)* et la technologie ATM. Depuis peu l'utilisateur peut profiter de la technologie xDSL (*Digital Subscriber Loop*) permettant de porter des données numériques entre le centre de raccordement et le site de l'abonné.

Les liaisons louées offrent un débit entre 64 Kbit/s et 34 Mbit/s entre deux points. *Frame relay* est une technologie à commutation par paquets, permettant d'exploiter les réseaux publics par une multitude de clients indépendants. Le débit d'accès au *frame relay* est entre 64 Kbit/s et 34 Mbit/s. La technologie ATM exploite la commutation de cellules pour répondre à la demande de différents services avec un débit élevé pourant atteindre plusieurs centaines de mégabits par seconde.

La technologie xDSL permet de connecter les utilisateurs finaux (postes individuels, réseaux locaux) par les liens haut débit (2 Mbit/s) créés sur les connections téléphoniques de la boucle locale.

FIGURE 1. Distances de couverture et débits des réseaux informatiques



Topologies

Différents réseaux de communication possèdent différentes topologies. Selon le type de la topologie, l'information peut être diffusée sur un support commun (multi-point), ou être acheminée par les liens à l'accès unique (point-à-point). En principe, une installation réseau est composée de deux parties:

- les terminaux (postes simples ou sous-réseaux),
- l'infrastructure du réseaux (longue distance).

Les canaux de communication peuvent fonctionner en mode de diffusion ou en mode point-à-point. La première technique est largement répandue dans les réseaux locaux à courte distance. Le mode point-à-point est nécessaire dans les réseaux filaires longue distance.

Parmi les topologies utilisées pour la construction/interconnexion des réseaux locaux nous trouvons:

- le bus,
- l'anneau,
- l'étoile,
- l'arborescence,
- le maillage.

Le bus correspond à la mise en parallele de tous les accès réseau. Le débit réel global sur le bus est partagé dans le temps par les utilisateurs.

L'étoile est constituée d'un élément central appelé *hub* (moyeu) ou concentrateur. Théoriquement, si l'élément central est une matrice de commutation spatiale, le débit global d'un réseau en étoile peut atteindre n-1 fois la bande passante de chaque liaison.

L'anneau est composé d'un chemin bouclé sur lequel sont connectés les adaptateurs réseau. Sur un anneau l'odre de circulation des informations est implicite et les adaptateurs sont atteints consécutivement.

Le maillage est une topologie où les adaptateurs sont reliés par couple par des liaisons en point-à-point. C'est la topologie la plus coûteuse; elle necessite **n*(n-1)/2** liaisons pour **n** adaptateurs.

FIGURE 2. Terminaux et infrastucture du réseau

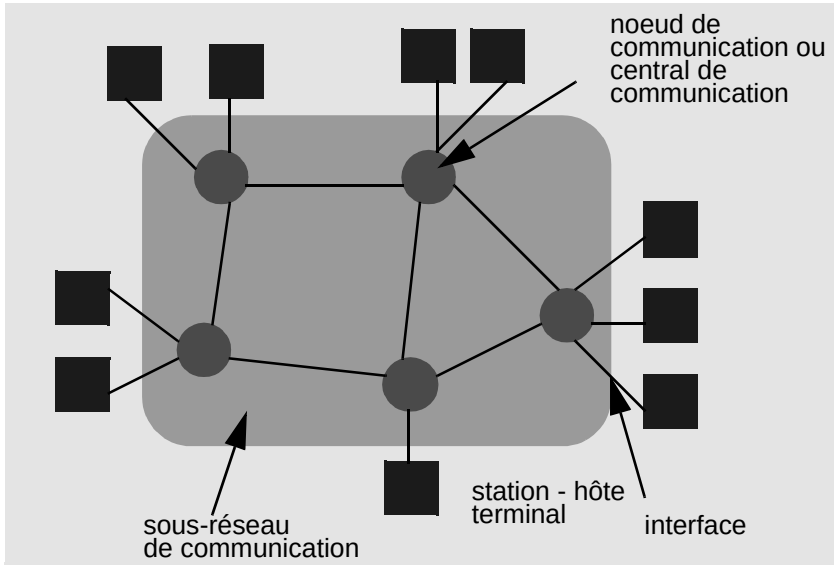
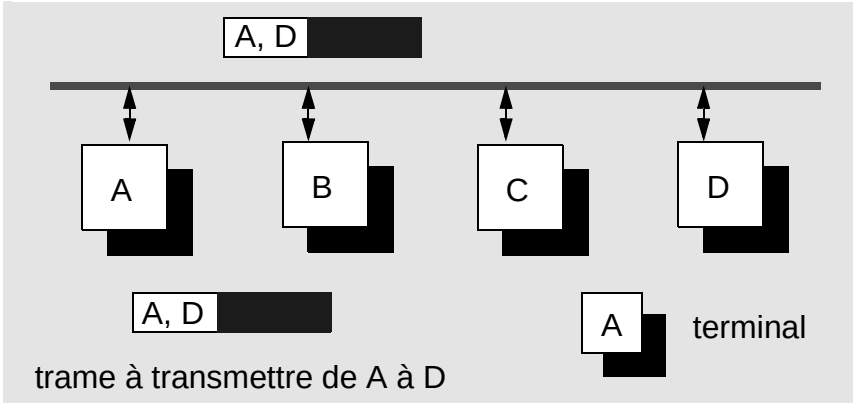


FIGURE 3. Mode de communication en diffusion



Réseaux point-à-point

Les noeuds de communication permettent de choisir la route de communication en fonction de l'adresse de destination. Selon la nature du réseau et de l'implémentation utilisée on trouve deux sortes de noeuds de communi-

cation: les commutateurs et les routeurs. Dans les deux cas, un noeud de communication intègre les fonctions de traitement et de mémorisation, et fonctionne selon le principe «*store and forward*». Dans le cas des commutateurs (e.g. commutateur ATM), le temps de stockage est très court, ce qui permet d'effectuér les communications en temps réel.

FIGURE 4. Mode point-à-point

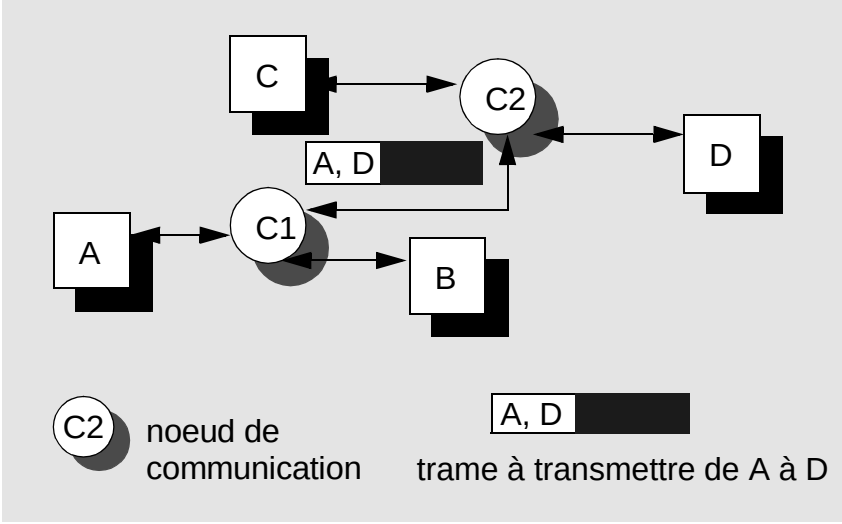


FIGURE 5. Topologies point-à-point

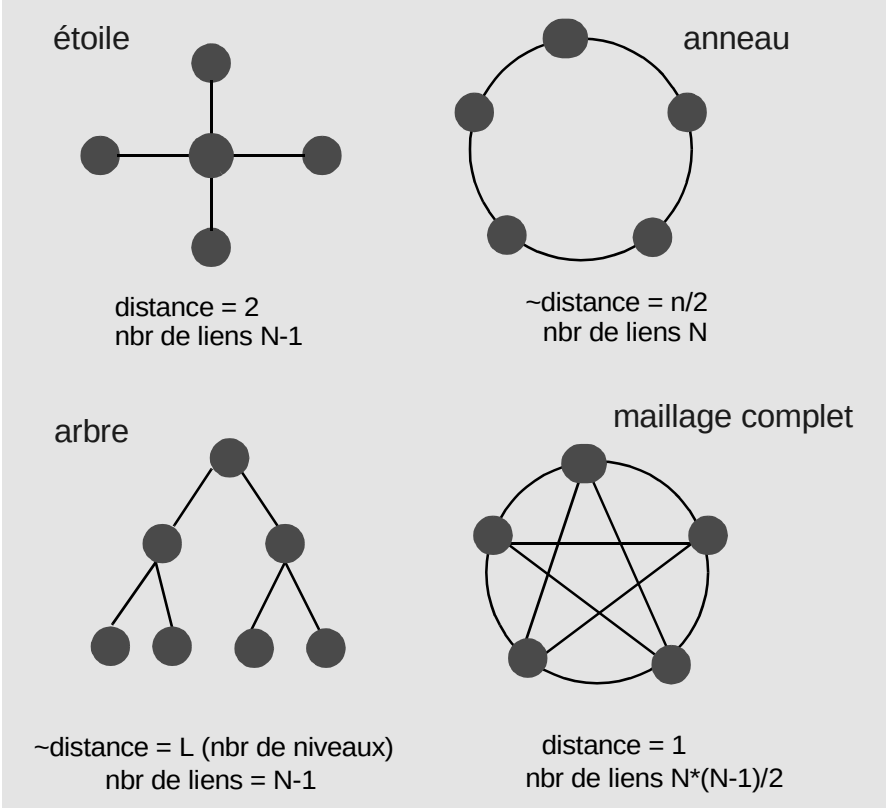
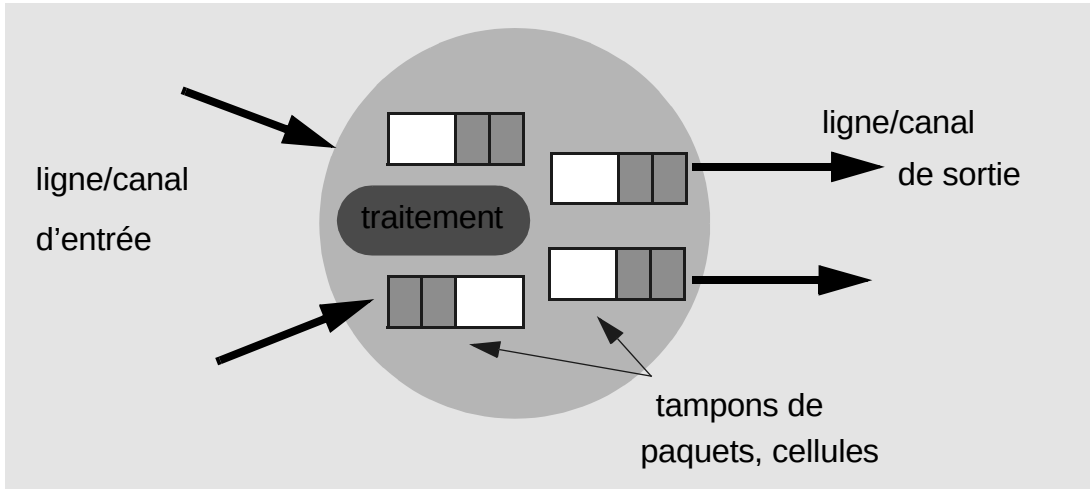


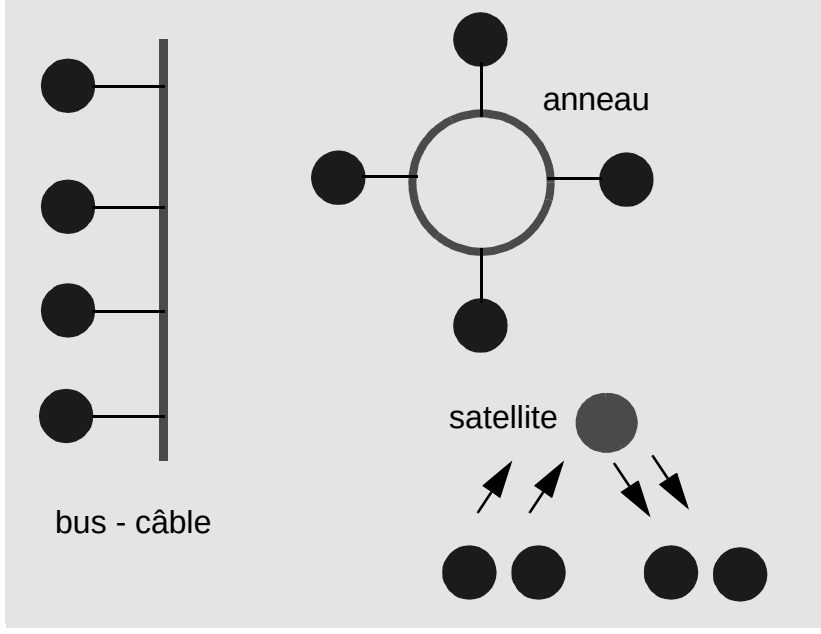
FIGURE 6. **Noeud de communication (commutateur, routeur)**



Réseaux de diffusion

Les réseaux de diffusion, essentiellement les réseaux locaux, sont basés sur un support commun de communication. Ce support peut être matériel ou hertzien. Ci-dessous nous présentons quelques topologies de base pour les réseaux locaux communiquant sur un segment du support matériel.

FIGURE 7. **Topologies des réseaux fonctionnant en mode diffusion**



Standards

Dans le domaine des communications, la grande majorité des technologies est standardisée. Dans ce cadre, deux principaux modèles de référence sont utilisés, celui de l'OSI et celui du monde internet (TCP/IP).

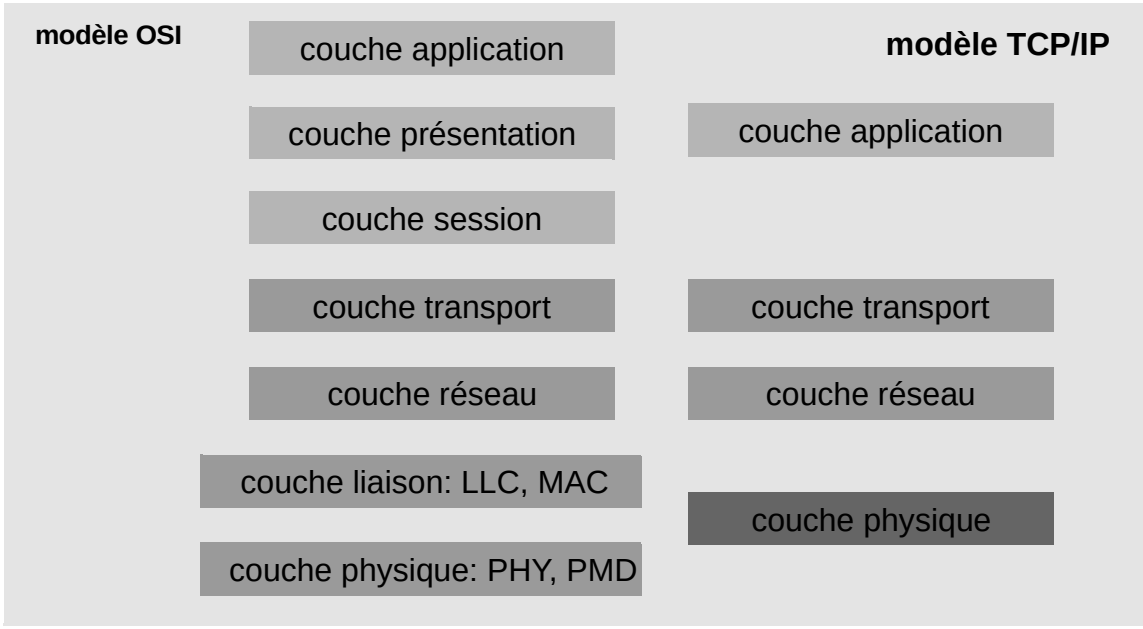
Le modèle OSI est découpé en sept couches, dont les quatre plus basses concernent l'implémentation du réseau. Les couches supérieures sont orientées vers l'interface de programmation et les applications utilisateur.

- la couche physique (PHY) spécifie les caractéristiques physiques propres à la transmission du signal (codage, détection du signal, synchronisation, détection de collisions, ..); la couche PHY est souvent décomposée en sous-couche haute PHY (codage, synchronisation,..) et en sous-couche basse PMD (*Physical Medium Dependent*)
- la couche liaison de données assure le transfert de données sur le support physique sous la forme de trames avec contrôle des erreurs et gestion de flux; dans les technologies LAN cette couche est subdivisée en deux niveaux: MAC (*Medium Access Control*) et LLC (*Logical Link Control*)
- la couche réseau unifie les couches sous-jacentes par introduction des datagrammes qui véhiculent les adresses réseau et les données utilisateur.
- la couche transport peut fournir seulement un aiguillage vers les niveaux d'application ou vers un service de transfert fiable, avec détection et correction d'erreur et gestion des flux.

Le modèle TCP/IP est constitué de quatre couches:

- la couche matérielle qui intègre les réseaux physiques LAN et MAN (Ethernet, ATM,..),
- la couche internet permettant d'interconnecter plusieurs réseaux physiques différents,
- la couche transport qui fournit les services de transport fiable ou non-fiable à travers le réseau internet,
- la couche application intégrant différents services applicatifs (e.g. services WEB).

FIGURE 8. Modèle OSI et modèle TCP/IP



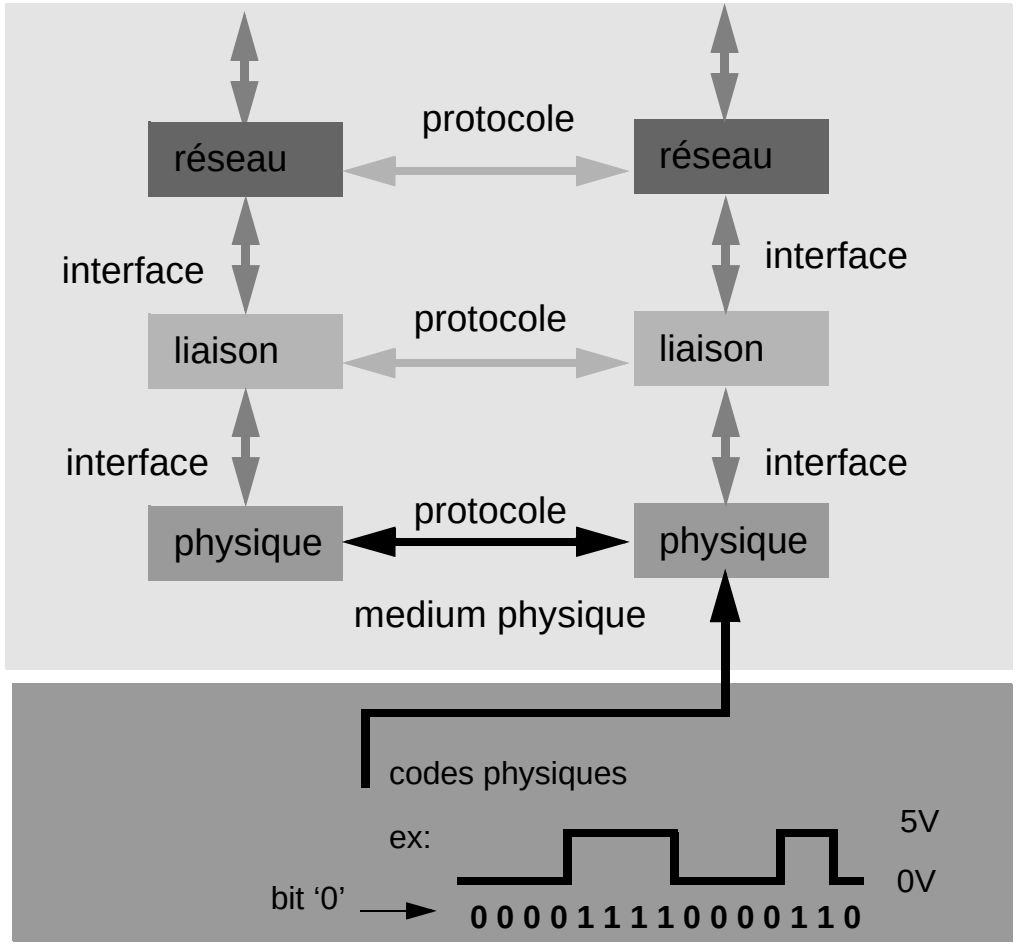
Protocoles et interfaces

Dans les modèles de référence, la couche de niveau N d'une station gère la communication avec la couche de niveau N d'une autre station. A chaque niveau ou couche correspond un **protocole de communication**. Les unités actives de niveau N sont appelées **processus pairs.** Entre chaque paire de couches adjacentes, on trouve une **interface**.

Couche Physique

La couche physique s'occupe de la transmission des **bits** sur un support de communication. Les bits sont codés sur 2 ou plus niveaux physiques du signal électrique, électro-magnétique, optique,...

FIGURE 9. Protocoles et interfaces: couche physique et code physique

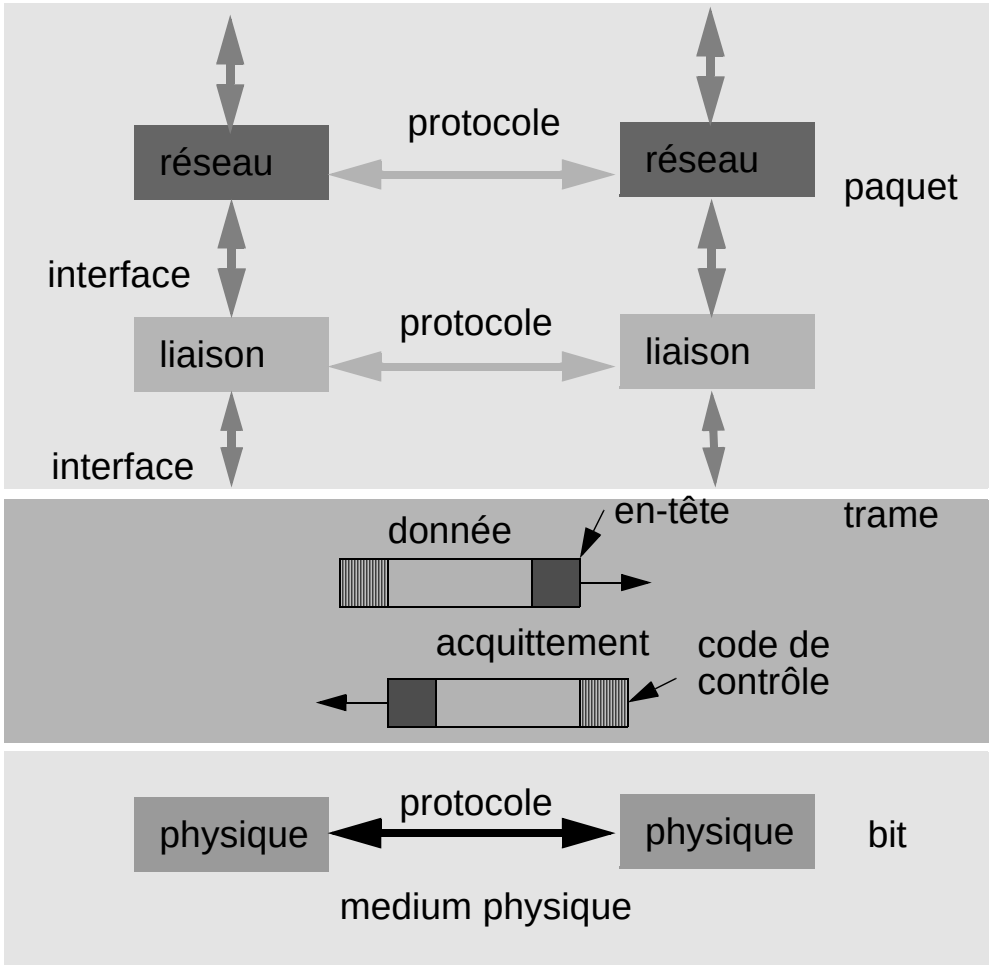


Couche Liaison

La couche liaison s'occupe de la transmission des **trames** (suites de bits) sur le support de communication. Les trames envoyées par l'émetteur s'appellent "trames de données". Les trames renvoyées par le récepteur s'appel-

lent “trames d’acquittement”. La couche liaison rend la communication fiable en employant trois opérations: **acquittement**, **temporisation**, et **retransmission**.

FIGURE 10. Protocoles et interfaces: couche de liaison et trames

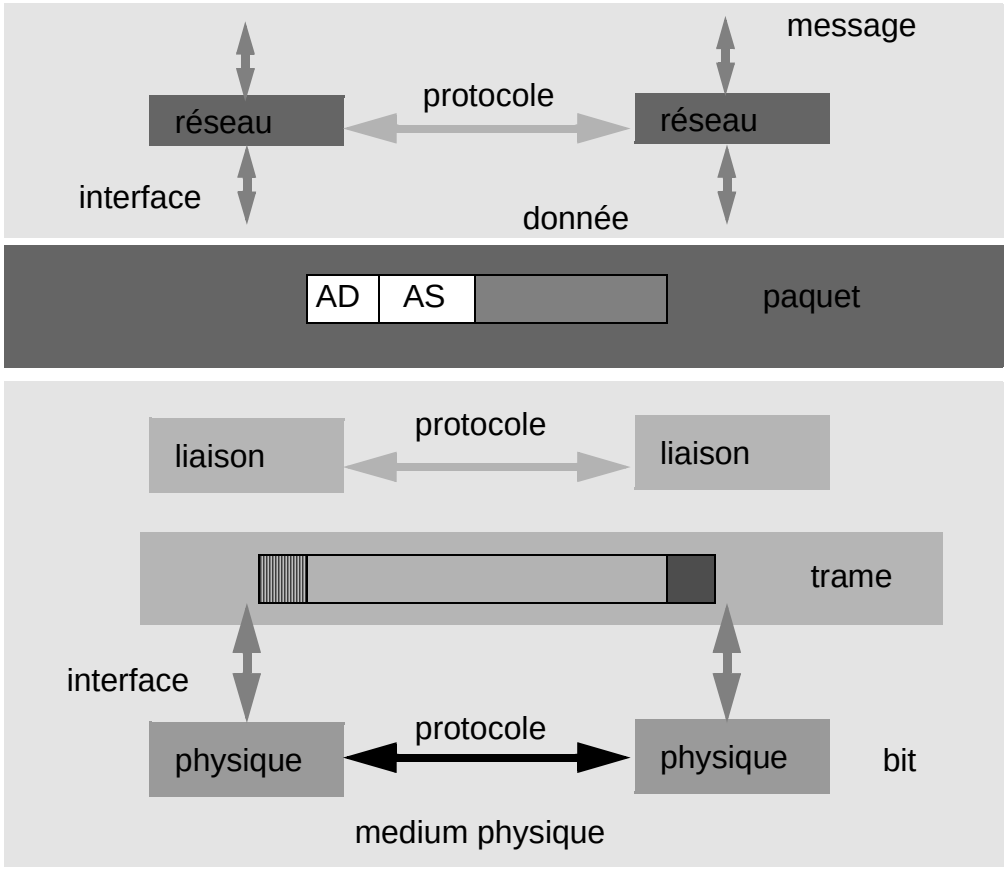


Couche Réseau

La couche réseau permet de gérer le sous-réseau et intègre les mécanismes permettant d’acheminer un **paquet** de la source au destinataire.

Les routes peuvent être déterminées statiquement ou dynamiquement pendant le fonctionnement du réseau. La même route peut être utilisée pour l’ensemble des paquets d’un message; alternativement plusieurs routes peuvent être utilisées pour la transmission d’un message décomposé en plusieurs paquets.

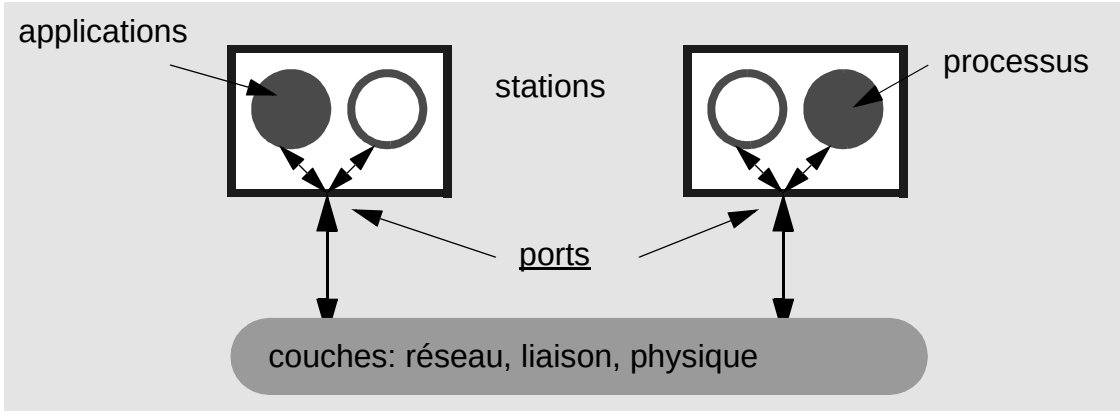
FIGURE 11. **Protocoles et interfaces: couche réseau et paquets**



Couche Transport

La couche transport implémente des protocoles permettant d'associer et de rendre fiable les services du sous-réseau aux applications qui s'exécutent sur les **équipements terminaux**.

FIGURE 12. **Couche de transport: ports et applications**



Synchronisme

Une des notions les plus importantes dans la compréhension des réseaux informatiques est le synchronisme. Les termes synchrone, asynchrone, isochrone et plésiochrone sont souvent utilisés dans le monde des télécommunications. Ils peuvent revêtir plusieurs significations selon que l'on s'intéresse au niveau physique (bit, octet) ou au niveau de réseau (paquet).

Au niveau physique:

- les communications sont **synchrones** si l'horloge ayant servi au codage du signal est continuellement véhiculée avec lui (soit dans le signal, soit en parallèle par un signal associé),
- les communications sont **asynchrones** si l'horloge du codage n'est pas transmise avec le signal et n'est donc pas connue du récepteur - dans ce cas les données émises sont encadrées par des bits de *start* et de *stop*,
- les communications sont **isochrones** si l'émetteur et le récepteur ont des horloges identiques, assujetties à une troisième horloge,
- les communications sont **plésiochrones** si l'émetteur et le récepteur ont des horloges indépendantes, mais suffisamment proches et précises pour permettre des échanges avec un minimum d'erreur.

Au niveau réseau:

- les communications sont **synchrones** si l'équipement émetteur dispose d'une bande passante et d'un délai garanti; de son côté le récepteur sait précisément à quel instant un paquet peut lui parvenir,
- les communications sont **asynchrones** si l'équipement récepteur n'a aucune connaissance a priori de l'instant précis où un paquet peut lui parvenir,
- les communications sont **isochrones** si elles sont synchrones et présentent un délai de transit d'acheminement fixe,
- les communications sont **plésiochrones** si elles sont asynchrones, mais que le délai de transit est borné et qu'une partie du débit est garantie.

Résumé

Dans cette introduction nous avons présenté quelques notions et concepts de base. Les chapitres suivants seront dédies à la présentation plus détaillée des couches fonctionnelles. Dans les derniers chapitres nous parlerons des grandes familles de réseaux: Ethernet, ATM, DSL,..

Dans la deuxième partie de cet ouvrage (en anglais), nous présentons les protocoles et la technologie Internet.

CHAPITRE 2

Couche physique

Le rôle de la couche physique est de transmettre les valeurs binaires sur le support physique. Cette transmission nécessite un support matériel - câbles et fibres ou support hertzien. Dans le monde des réseaux locaux, certains types de câbles se sont imposés. Il s'agit principalement du câble à paire torsadée non blindée de catégorie 5 (UTP5 - *unshielded twisted-pair*) et de la fibre optique de silice multimode à gradient d'indice 62,5/125µm (MMF - *multi mode fiber*). La fibre optique monomode (9-10/125µm) est nécessaire pour les débits avoisinants un gigabit par seconde sur les distances de plusieurs dizaines, voire centaines de kilometres.

Les paires torsadées sont utilisées sur les distances limitées (100 m) et permettent de supporter la communication en full-duplex avec un débit de 155 Mbit/s sur deux paires UTP3 ou de 1000 Mbit/s sur quatre paires UMT5.

FIGURE 13. Fibres optiques

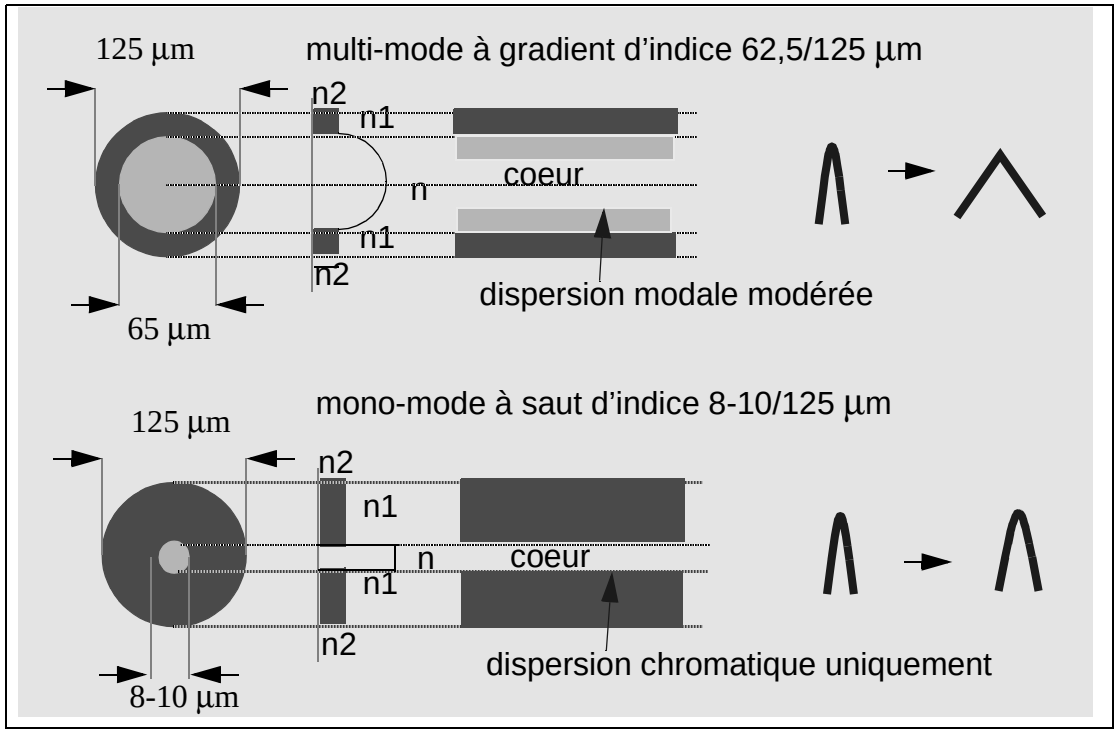
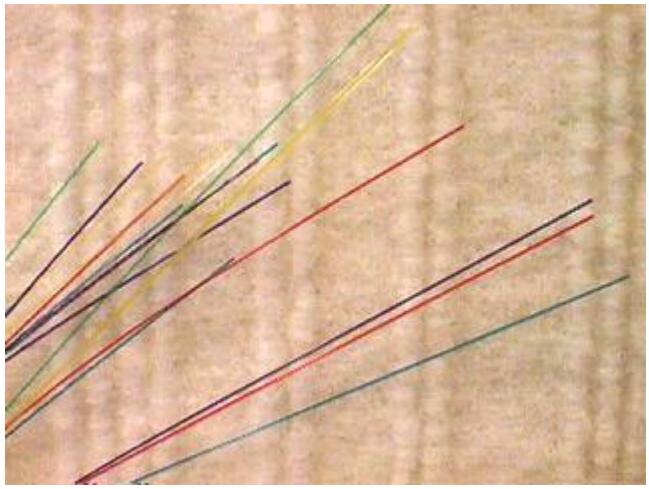


FIGURE 14. **Brins d'une fibre optique**

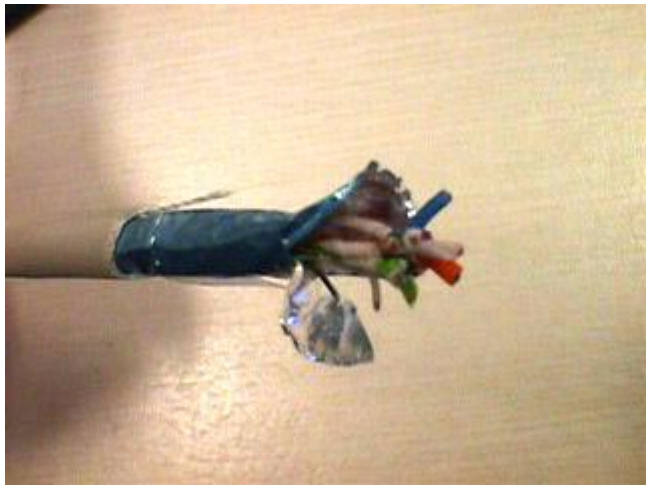


Un des paramètres les plus importants dans le choix du support est l'atténuation. Le tableau ci-dessous montre l'atténuation (en dB/100 m) du câble UTP5 sur la distance de 100 m en fonction de la fréquence (débit) utilisée.

TABLEAU 1. **Atténuation (dB/100m) sur un câble UMT5 en fonction des fréquences**

10 MHz	100 MHz	150 MHz	200 MHz	300 MHz	500 MHz
6,6	22	27	32	39	56

FIGURE 15. **Câble catégorie UMT5**



Remarque:

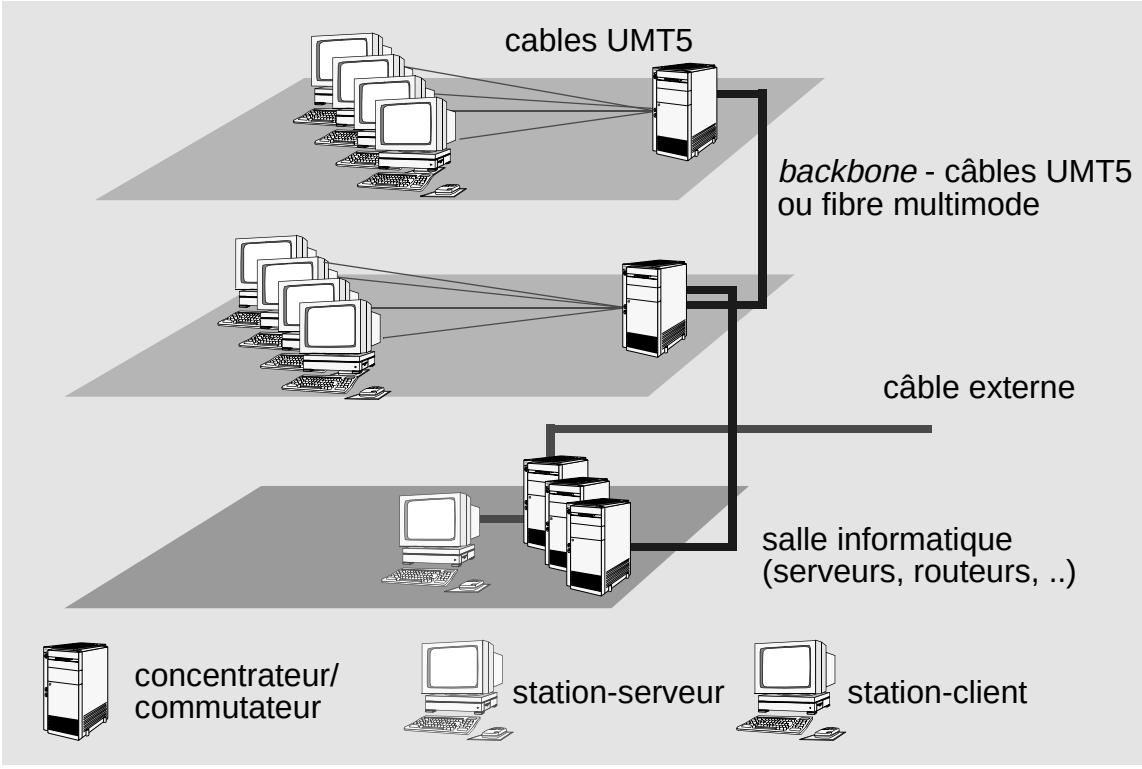
La vitesse de propagation d'un signal électrique sur la paire torsadée UMT5 est **0,6 c** (180 000 Km/s)

Pré-câblage d'un bâtiment

Basée sur le câble UTP et la fibre multimode, la structure traditionnelle d'un pré-câblage est la suivante:

- au coeur du réseau, un local technique - centre informatique va accueillir le commutateur ou concentrateur central, auquel sont reliés d'une part les concentrateurs d'étage par les rocades (*backbone*), et d'autre part les serveurs, les routeurs et les passerelles centralisées,
- un cablâge fédérateur vertical partant du centre informatique vers les locaux techniques d'étage, composé de fibres optiques si les distances entre le centre informatique et les locaux techniques sont supérieures à 100 m ou de câbles UTP multi-paires,
- des locaux techniques d'étage recevant les câbles reliés aux prises murales dans les bureaux, un ou deux câbles de rocades et le matériel actif - concentrateur (*hub*) ou commutateur
- une desserte capillaire horizontale en paire torsadée non blindée (UMT5) vers les prises situées à moins de 100m du local technique

FIGURE 16. Eléments du câblage de bâtiment

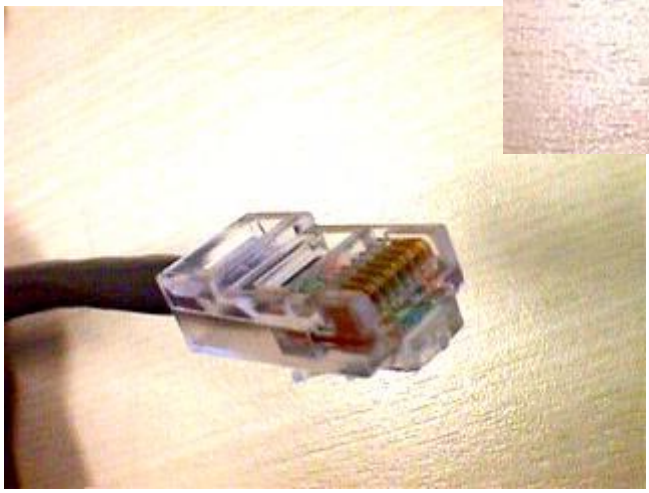


Connecteurs

Les câbles véhiculent les signaux qui doivent être communiqués aux équipements par le biais des connecteurs. Selon le type de câble métallique, deux sortes de connecteurs sont utilisés: les connecteurs type T (câble coaxial) et les connecteurs type RJ45 pour les paires torsadées.

FIGURE 17. Connecteurs réseau

Connecteur type T pour le câble coax fin (50 Ohms)



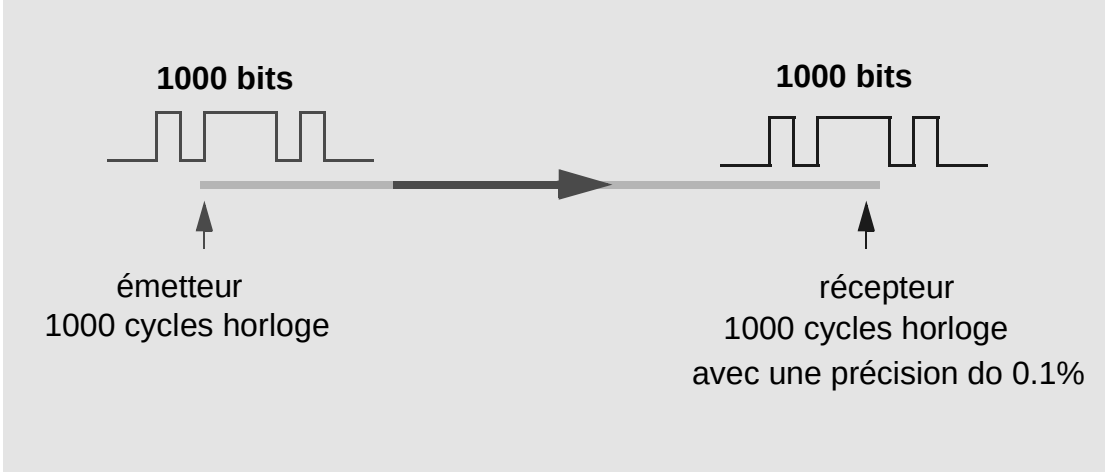
Connecteur type RJ45 pour le câble à huit fils (quatre paires torsadées)

FIGURE 18. Prises murales pour les connecteurs RJ45

Codage physique

Les codes physiques permettent de réduire la bande passante globale du signal et d'apporter une capacité de vérification d'intégrité du signal à la réception. Pour les paires torsadées à débit élevé de l'ordre de 100 Mbit/s, nous avons besoin de codes sur plusieurs niveaux. Les codes physiques permettent également de porter le signal de l'horloge.

FIGURE 19. Problème de précision des horloges



Les codes les plus simples sont à deux niveaux avec des fronts calés sur le signal horloge. Le choix **(-1, 1)** permet de produire un signal éléctrique équilibré avec une composante continue nulle. Les codes sur deux niveaux peuvent mettre en oeuvre un mode de transfert sur des mots de **n** bits. Pour le code avec controle d'intégrité, plus le mot est long, plus grande est l'efficacité du code.

FIGURE 20. Codage simple NRZ et NRZI

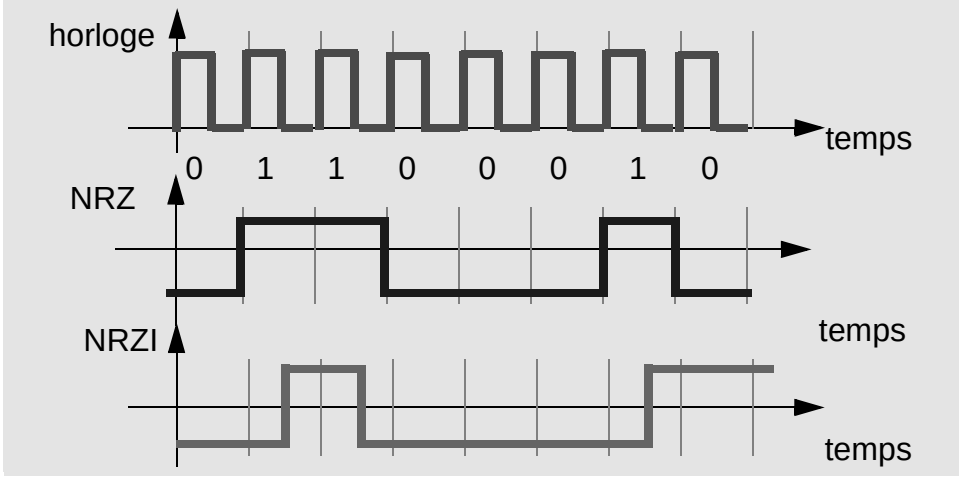


TABLEAU 2. Codes physiques à deux niveaux sur un bit

nom du code	présentation du code	réseaux
NRZ - *Non-Return to Zero*	le bit 0 : ‹-1›, le bit 1: (1); pas de composante constante, ne véhicule pas l'horloge	
NRZ I- *Non-Return to Zero Inverted*	le bit 0 ne change pas de niveau, le bit 1 fait basculer le niveau; permet de se proteger contre les suites de bit 1	
Manchester	intégre une variation de niveau au cours de chaque bit transmis; le bit 0 est codé comme transition 1 vers -1, le bit 1 comme transition -1 vers 1; le code véhicule l›horloge; les bits sans transition sont des violations du code utilisés pour indiquer un événement spécifique (plus 100 % de la bande passante)	Ethernet
Manchester différentiel	les bits intégrent une variation en leur milieu, mais le 1 présente une continuité par rapport au niveau précédent, le 0 est un changement; le bit 0 possède deux fronts un au début et un au milieu, alors que le bit 1 n'a qu'un front au milieu	*Token Ring*

TABLEAU 3. Codes physiques à deux niveaux sur un mot

nom du code	présentation du code	réseaux
4B/5B	4 bits de données sont transmis par un symbole de 5 bits; le code est défini par une table de codage où les 16 séquences de 4 bits de données sont mises en correspondance avec les symboles 5 bits. Ces symboles garantissent un front tous les 3 bits au plus et minimisent la composante continue; les 16 codes restants sont employés pour coder des symboles spécifiques; l'avantage du code 4B/5B est le contrôle d'intégrité une bonne exploitation de la bande passante (plus 25%)	FDDI
8B/10B	8 bits de données (1 octet) sont traités à la fois; le 256 combiniasons sont codées sur 1024 symboles; le décodage nécessite des tables de conversion importantes; la redondance de symboles permet d'obtenir le contrôle d'intégrité et un bon équilibre pour la réduction de la composante continue (plus de 25% de bande passante)	*Fiber Channel*

TABLEAU 4. Codes physiques à trois niveaux

nom du code	présentation du code	réseaux
MLT-3 - *Multi-Line Transmission*	ce code sur trois niveaux (-1,0,1) fonctionne comme NRZI, change d'état pour la transmission d'un '1', reste au même niveau pour un 0; la bande passante consommée est 50%.	Ethernet 100 Mbit/s - 100Base-TX
8B/6T	ce code convertit un octet en six codes à trois niveaux (256 combinaisons sont codées par 729 possibilités); l'intégrité du signal est contrôlée; la bande passante consommée est reduite de 60% par rapport à NRZ.	Ethernet 100Base-T4

FIGURE 21. Codage Manchester - code autosynchrone

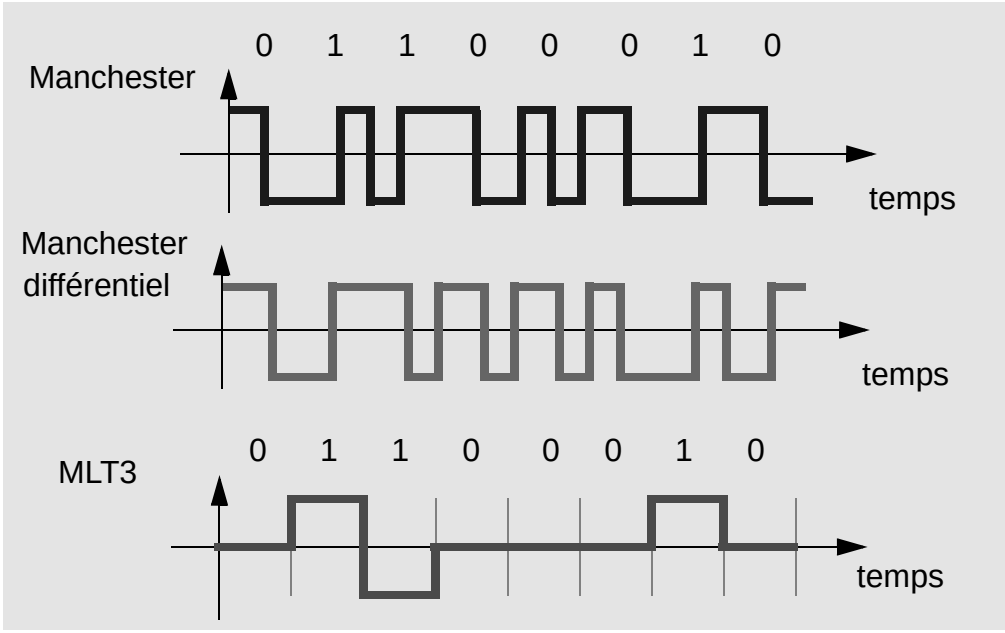


TABLEAU 5. Codes physiques à plusieurs niveaux avec modulation de phase

nom du code	présentation du code	réseaux
64-CAP - *Carrierless Amplitude/Phase modulation with 64 constellation points*	une suite de 6 bits en un couple de bits dénote le quadrant, et les quatre autres bits identifient l'une des 16 combinaisons; chaque point de constellation a pour abscisse et ordonnée une des quatres valeurs (1,3,5,7) en positif ou négatif; le débit de 155 Mbit/s correspond donc à 26 Mbauds	ATM 155 Mbit/s sur UTP3
PAM5 - *5 level Pulse Amplitude Modulation*	ce code emploie l'émission d'impulsions à cinq niveaux (-2,-1,0,1,2); dans le cas d'Ethernet à 1Gbit/s ce code est exploité avec un codage à treillis à quatre états	Ethernet 100Base-T4, Ethernet 1Gbit/s, ADSL

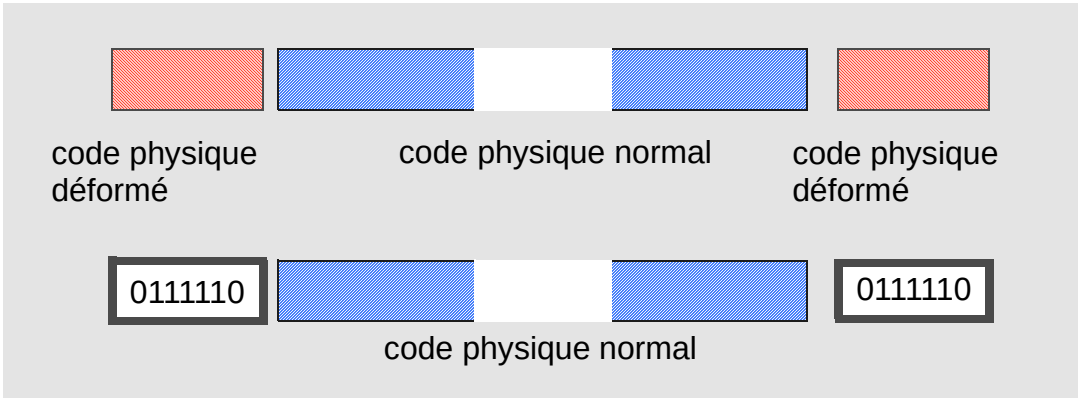
En général, pour les codes avec une faible sensibilité aux perturbations, une faible bande passante consommée et une redondance intrinsèque, nous avons besoin de traitements plus complexes et plus coûteux surtout pour les haut débits.

Marquage des trames

Le niveau physique permet de transmettre les bits d'information. Dans les réseaux informatiques, les bits d'information sont groupés en trames physiques. Une trame physique débute par un marqueur de début de trame; elle se termine par un marqueur de fin.

Un marqueur peut être construit à partir d'un code physique déformé ou partir d'une suite binaire spécifique - appelée fanion. Dans le cas d'utilisation des fanions, la séquence de plus de 5 bits '1' est interdite dans la partie interne de la trame. La technique appelée: insertion de zéro (*bit stuffing*) permet de transformer ces séquences en séquences à cinq '1' consécutifs suivies par un zéro: 111111 => 1111101. Les bits '0' ajoutés à l'émission sont enlevés à la réception.

FIGURE 22. Techniques de marquage des trames



Résumé

La couche physique est composée du matériel dans lequel circulent les signaux électriques et/ou optiques. Ses signaux portent l'information binaire. Différents codes ont été inventés pour augmenter l'efficacité et la qualité de transmission au niveau physique. Un groupement de ces codes permettant d'envoyer une suite de valeurs binaires est appelé **trame**.

Dans le chapitre suivant nous étudierons les principes de communication sur une ligne physique partagée. Ces principes et les mécanismes correspondants sont implémentés par une couche fonctionnelle appelée MAC pour *Multiple Access Control*. La couche MAC à été formalisée par le biais des standards IEEE 803.X concernant les réseaux locaux type *Token Ring*, Ethernet, FDDI, et beaucoup d'autres.

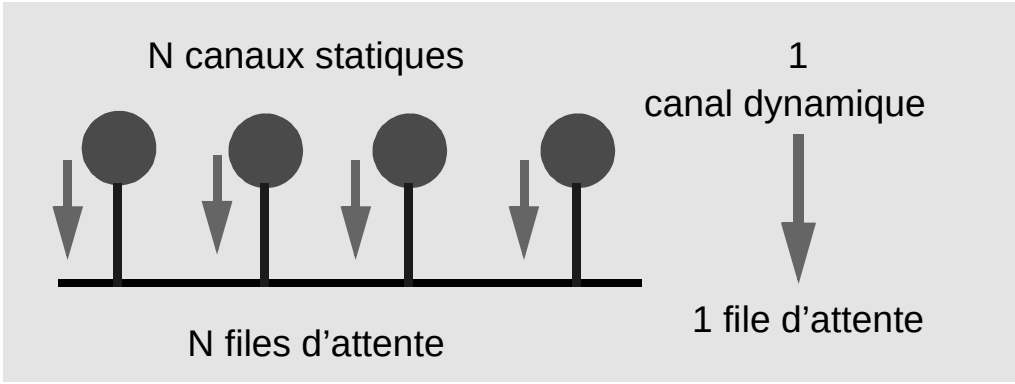
CHAPITRE 3

Couche MAC

Dans les réseaux locaux traditionnels, un support physique est partagé entre plusieurs utilisateurs. Dans cette situation, nous avons besoin de définir les règles d'accès au media pour l'émission des trames. En principe il y a deux sortes de solutions à ce problème: les solutions statiques (e.g. TDMA - *Time Division Multiple Access*) et les solutions dynamiques.

Les solutions statiques sont simples à introduire mais souffrent d'un rendement très faible. Prenons comme exemple un réseau à quatre postes interconnectés par le même support; le faible rendement de l'allocation statique peut être mise en évidence par une application simple de la théorie des files d'attente.

FIGURE 23. Partage de capacité statique et partage dynamique de la capacité de transmission



Commençons par le délai moyen d'attente **T**, pour un canal ayant une capacité **C** en bits par seconde avec une cadence d'arrivée des trames de **M** trames par seconde, chaque trame ayant une longueur moyenne **L** (calculée à partir d'une probabilité exponentielle).

$$\mathbf{T = 1/(C/L - M)}$$

Ce qui nous donne pour les valeurs C= 64 Kbits, M=10, L=4 Kbits:

$$T= 1/(64/4 - 10)= 1/(16-10)= 1/6 \text{ s}$$

Divisons alors ce canal de manière statique en *N* canaux, chacun d'eux ayant ainsi une capacité de **C/N** bits par seconde et une charge moyenne de **M/N** trames par seconde.

Ce qui nous donne la formule:

$$\mathbf{T_{statique} = 1/(C/L*N - M/N)= N/(C/L-M)=T*N}$$

Prenons comme valeurs C= 64 Kbits, M=10, L=4 Kbits, N=4

Ce qui nous donne:

$$T_{statique}= 4/(64/4 - 10)= 4/(16-10)= 4/6 \text{ s}$$

Conclusion:

Pour le multiplexage statique, le délai moyen d'attente T est N fois plus important que si toutes les trames avaient été ordonnées, comme par magie, dans une seule file d'attente utilisant globalement le canal unique.

Techniques d'accès dynamique

Dans les réseaux locaux existent plusieurs techniques permettant de gérer le droit à l'émission sur un lien physique unique. Parmi ces techniques nous pouvons en évoquer quatre:

- le *polling,*
- l'accès aléatoire,
- l'accès par le passage du jeton,
- l'accès par insertion de registre.

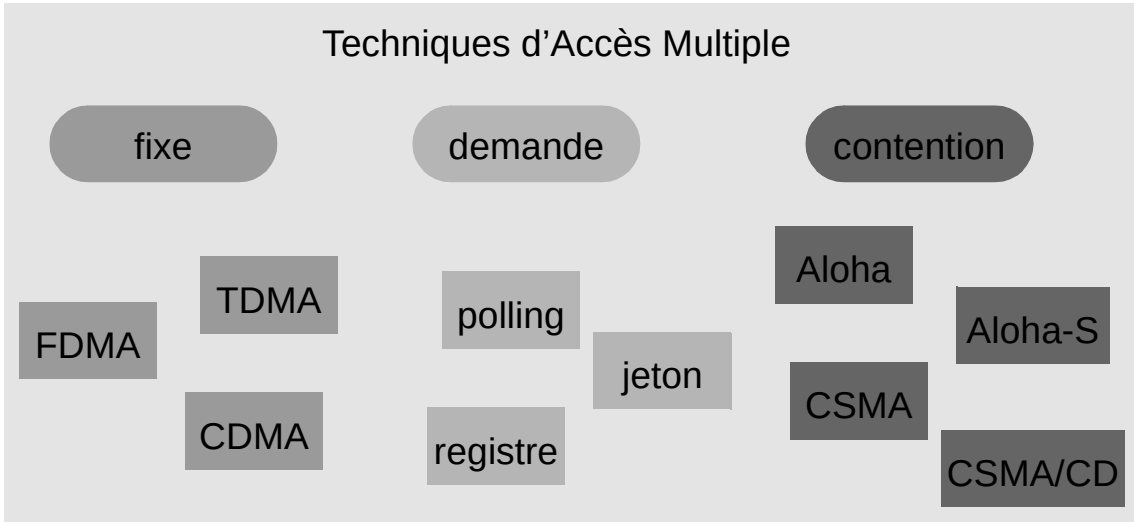
Le *polling,* ou scrutation, est basé sur une interrogation régulière et périodique de tous les équipements susceptibles de vouloir émettre. Cette solution est simple mais elle nécessite un gestionnaire central du réseau qui porte la responsabilité du *polling*.

L'accès aléatoire et multiple (*Multiple Access*) est une technique d'accès très simple qui ne prend pas en compte la présence d'autres stations. Cette technique est efficace seulement en cas de faible utilisation du support. Pour accroitre son utilité, elle était enrichie par l'insertion d'une surveillance de l'occupation du medium (*Carrier Sense*). L'introduction de détection de collision (*Collision Detection*) a permis d'augmenter d'avantage l'efficacité de cette technique d'accès. (*MACS/CD*)

L'accès par le passage du jeton (*Token Passing*) permet de régir les émissions sur un réseau circulaire. La station qui devient le jeton (une trame spéciale) a le **droit d'émettre** pendant un laps de temps borné. Après l'émission, le jeton est liberé et peut être récupéré par une station suivante. Le passage du jeton peut être associé à des niveaux de priorité.

L'insertion de registre correspond à la définition de tranches périodiques qui circulent devant les points d'accès. Si le registre est libre, une trame de taille fixe peut y être insérée au vol. Cette méthode a l'avantage de pouvoir être déterministe et synchrone, et permet de réserver une bande passante fixe pour les applications isosynchrones.

FIGURE 24. Techniques d'accès multiple



Standards IEEE 802.x

L'IEEE a défini plusieurs normes pour les réseaux locaux. Ces normes, connues sous le vocable IEEE 802, concernent le protocole CSMA/CD, le protocole bus à jeton (*Token Bus*), et le protocole anneau à jeton (*Token Ring*). L'ensemble des normes est divisé en sections.

- La norme IEEE 802.1 décrit le contexte des normes, traite de l'architecture et définit les primitives utilisées.
- La norme IEEE 802.2 concerne la sous-couche supérieure de la couche de liaison de données et en particulier le protocole LLC (*Logical Link Control*).

Les normes 802.3, 802.4, et 802.5 décrivent les trois types de réseaux locaux retenus par l'IEEE qui sont respectivement:

- 802.3 - le CSMA/CD,
- 802.4 - le protocole bus à jeton (Token Bus),
- 802.5 - le protocole anneau à jeton (Token Ring),
- ..
- 802.11 - réseaux sans fil.

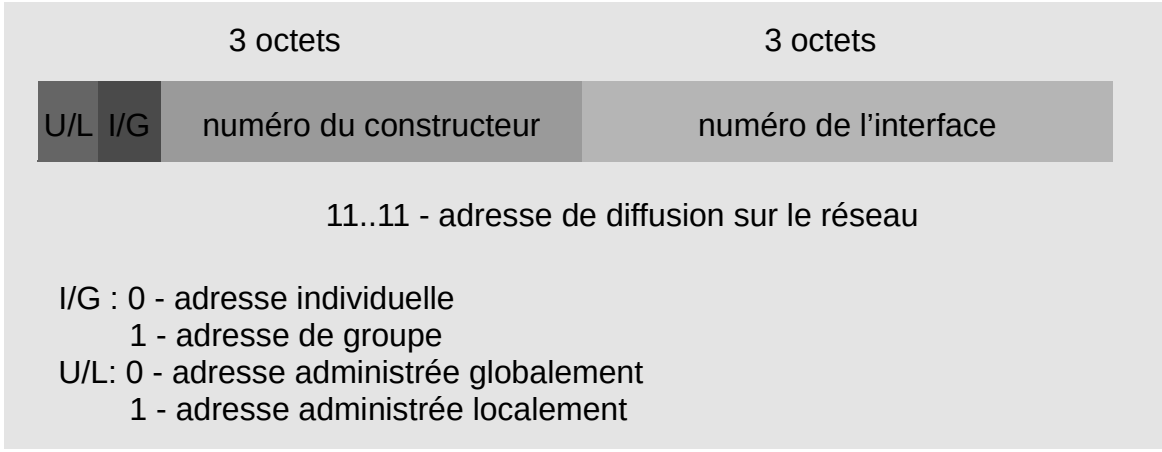
Chacune des ces normes définit la couche physique qu'elle utilise ainsi que le protocole MAC (*Medium Access Control*) qui l'accompagne.

Adressage MAC

L'adressage du niveau MAC permet d'identifier chaque carte interface connectée au réseau. Ces adresses sont utilisées dans l'en-tête de la trame pour préciser le destinataire de l'information mais aussi son émetteur. Normalement, une adresse MAC est implémentée de façon statique et elle est figée dans une ROM sur la carte adaptateur (*Burnt In Address*). Chaque carte adaptateur est donc identifiée par son adresse MAC.

L'adressage MAC est uniforme pour les réseaux du standard 802.x. La longueur des adresses MAC est de 6 octets; la premiere partie sur 3 octets (*Organizationally Unique Identifier*) identifie le numéro du constructeur , la deuxième le numero de série de l'interface. Les valeurs des adresses MAC sont présentées en code héxadécimal, par exemple: `35:d6:12:a1`

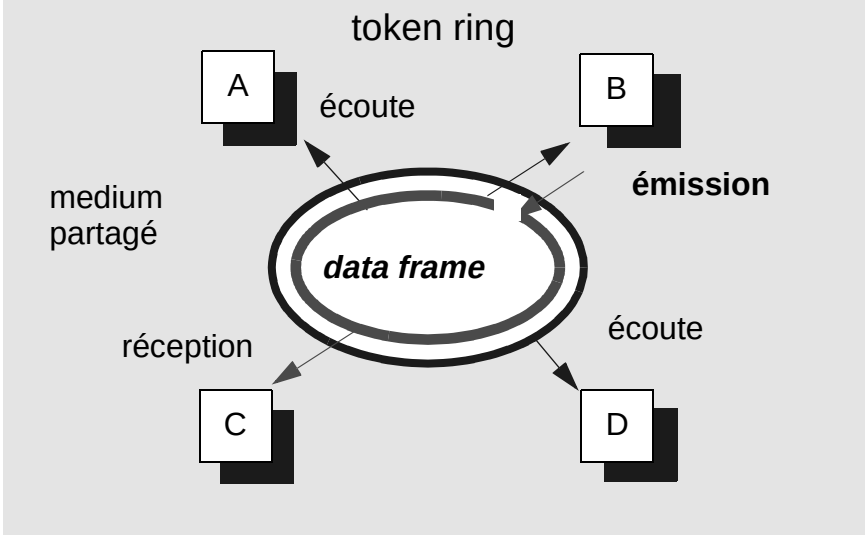
FIGURE 25. Structure interne d'une adresse MAC



Techniques de jeton - vers standard 802.5 et Token Ring

Le principe de cette technique, introduite en 1969 par Newhall, a donné naissance à de nombreuses variantes. Elle est utilisée sur des réseaux locaux comme le "Token Ring". D'une façon générale, cette technique consiste à faire circuler sur le réseau un **permis à émettre** ou "jeton", et seule la station qui le possède est effectivement autorisée à transmettre.

FIGURE 26. Emission d'une trame sur Token Ring



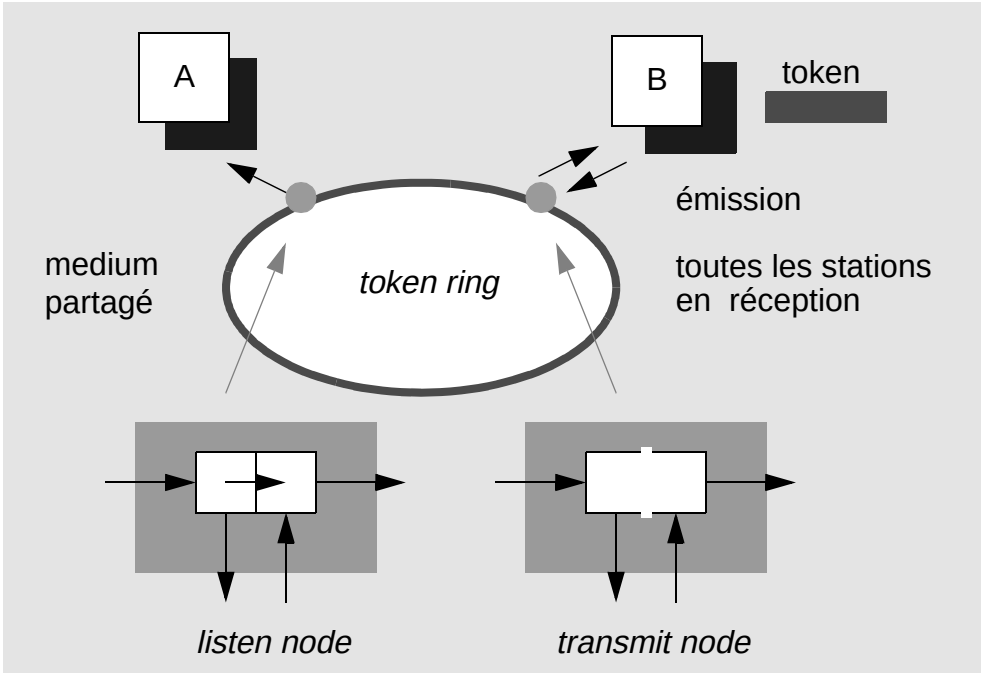
Fonctionnement:

- le jeton circulant peut être capté par toute station prête à émettre,
- une fois en possession du jeton, la station peut émettre ses données.

L'émission terminée, plusieurs cas de figure sont possibles quant au renvoi du jeton:

- La station ne rend le jeton que lorsque sa trame est entièrement revenue; il y a une seule trame sur le réseau. La trame est obligatoirement prélevée par l'émetteur, celui-ci réinsérant le jeton sur l'anneau.
- La station rend le jeton dès qu'elle a reçu l'en-tête de sa propre trame; cette approche permet un gain de temps par rapport à la précédente.
- La station rend le jeton immédiatement; dans cette approche on autorise la propagation simultanée de plusieurs trames différentes. Il n'y a toujours qu'une seule station autorisée à émettre à un instant donné; mais dès qu'elle a fini, elle redonne le contrôle du réseau.

FIGURE 27. Interface du Token Ring



Exercice:

Prenons un ensemble de valeurs - paramètres:

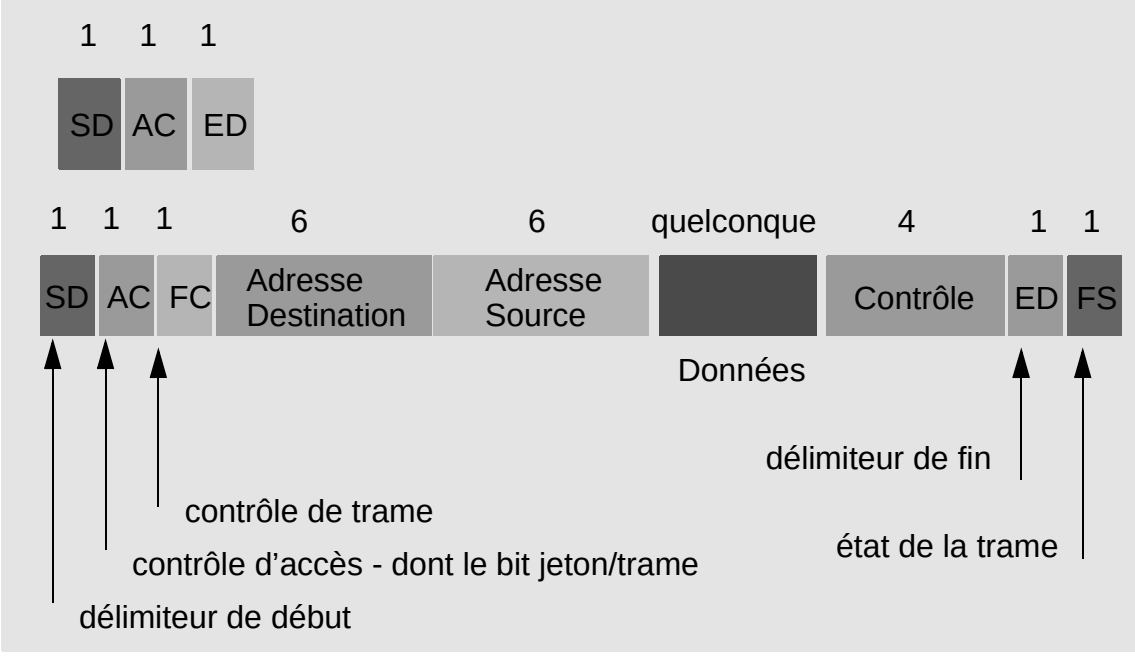
- longueur physique de l'anneau: 500 m
- débit binaire: 16 Mbit/s
- vitesse de propagation: 200 000 Km/s -> 1 Km - 5 µs
- nombre de stations connectées: 24

Et calculons le nombre de bits étalés sur le support physique du réseau.

Norme MAC IEEE 802.5 - Token Ring

Le fonctionnement courant du protocole 802.5 est très simple. Quand il n'y a aucun trafic sur l'anneau, un jeton comportant 3 octets circule en permanence sur l'anneau. La station désirant transmettre une trame le capture en positionnant l'un des bits du deuxième octet à 1. Cette action transforme ces deux premiers octets en une nouvelle séquence dite "début de trame".

FIGURE 28. Format d'un jeton/trame Token Ring



Dans des conditions normales, les premiers octets de la trame sont déjà de retour à la station émettrice alors qu'une bonne partie de la trame reste encore à émettre. La trame *Token Ring* est encadrée par **délimiteur de début** et un **délimiteur de fin**. Chacun d'eux contient des séquences de l'encodage Manchester différentiel invalides dénommées **HH** et **LL**.

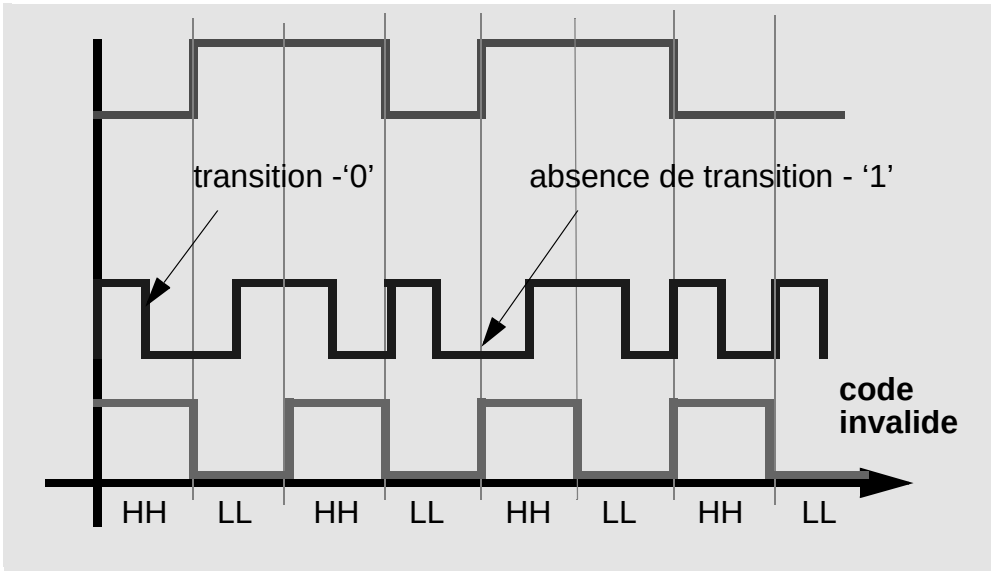
L'octet de contrôle contient le bit jeton/trame, le bit moniteur (positionné à 1 par la station moniteur au moment du premier passage), 3 bits de priorité et 3 bits de réservation de priorité.

L'octet **statut** de la trame contient deux paires de bits positionnés par la station destinataire et dénommés:

- bit A - destinataire actif,
- bit C - trame copiée.

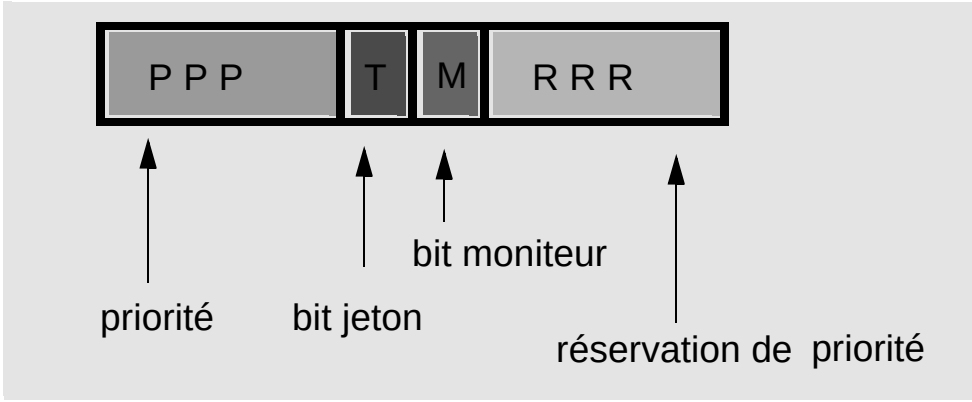
Quand l'octet statut retourne à la station émettrice, celle-ci peut déduire le sort de la trame (accusé de réception). Le marqueur de fin contient un bit dénommé E, pour erreur. Il sera positionné au moment du contrôle d'erreur par le récepteur. Le mécanisme d'accès prioritaire permet de prendre en compte différentes trames prioritaires.

FIGURE 29. **Codage Manchester modifié**



Quand une station veut transmettre une trame dont le niveau de priorité est **n**, elle doit attendre le passage d’un jeton dont les trois bits affectés au niveau de priorité courant expriment un niveau de priorité inférieur à **n**. La station qui veut transmettre une trame prioritaire peut effectuer une réservation en inscrivant le niveau de priorité dans la trame de données. Toutefois, si une réservation de rang supérieur au sien a déjà été formulée elle ne peut exprimer la sienne. Lors de la génération du nouveau jeton la priorité réservée sera inscrite dans le champ de priorité du jeton.

FIGURE 30. **Octet contrôle d’accès**



Animation du Token Ring

Chaque réseau Token Ring est doté d’une station moniteur qui supervise la circulation du jeton et des trames sur l’anneau physique. Si la station ayant pris en charge la fonction du moniteur vient à défaillir, elle est remplacée immédiatement par une autre station. Au moment de l’activation de la première station de l’anneau, ou quand l’une de stations constate qu’il n’y a pas de moniteur, elle peut transmettre la trame de contrôle - **recherche jeton** . Si cette trame effectue une rotation sur l’anneau, celle-ci devient le moniteur.

Les fonctions du moniteur:

- vérifier la perte du jeton: régénération du jeton,
- vérifier la structure du jeton: régénération du jeton,
- détecter une trame orpheline (bit Moniteur): l'extraction de la trame,
- maintenir de la contenance du réseau (24 bits): ajout d'une mémoire de correction.

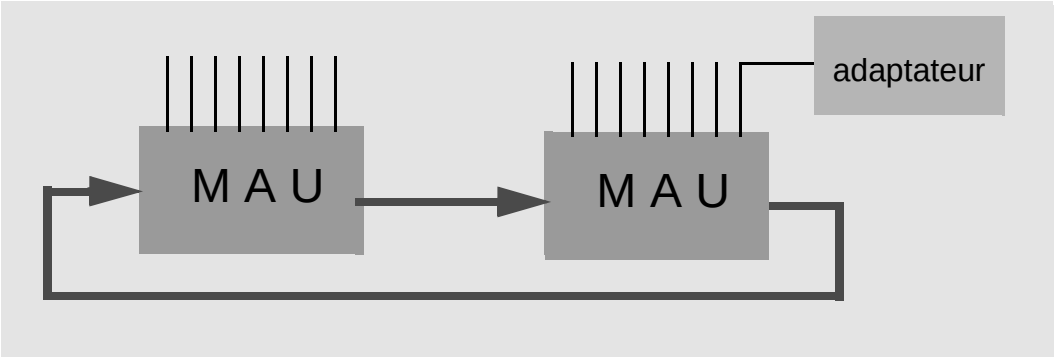
Configuration du Token Ring

Pour installer le réseau Token Ring, il faut insérer une carte adaptateur de réseau Token Ring dans chaque ordinateur. Chaque adaptateur est relié à une boîte appelée MAU (*Medium Access Unit*) par deux paires de câbles, l'une servant à la transmission et l'autre à la réception. Huit stations peuvent être raccordées à un seul MAU ; plusieurs MAU (33 au maximum) peuvent être connectés entre eux, ce qui porte à 260 le nombre maximum de stations dans un réseau.

Les unités de raccordement MAU peuvent être séparées de:

- 200 mètres si elles sont raccordés par du câble téléphonique normal (UMT3),
- 750 mètres si elles le sont avec du câble blindé (STP),
- 2 km si elles le sont avec des fibres optiques.

FIGURE 31. Configuration Token Ring à 16 postes maximum



Techniques d'accès aléatoire vers standard MAC IEEE 802.3 - Ethernet

Ethernet se définit au niveau d'accès comme une technologie de base en bus. Physiquement, un réseau Ethernet peut être constitué par un câblage en bus ou par un câblage arborescent. Le code physique utilisé est dans la majorité des cas de type Manchester. Le débit standard initial est de 10 Mbit/s. Ce débit est également porté à 100 Mbit/s et plus récemment à 1 Gbit/s.

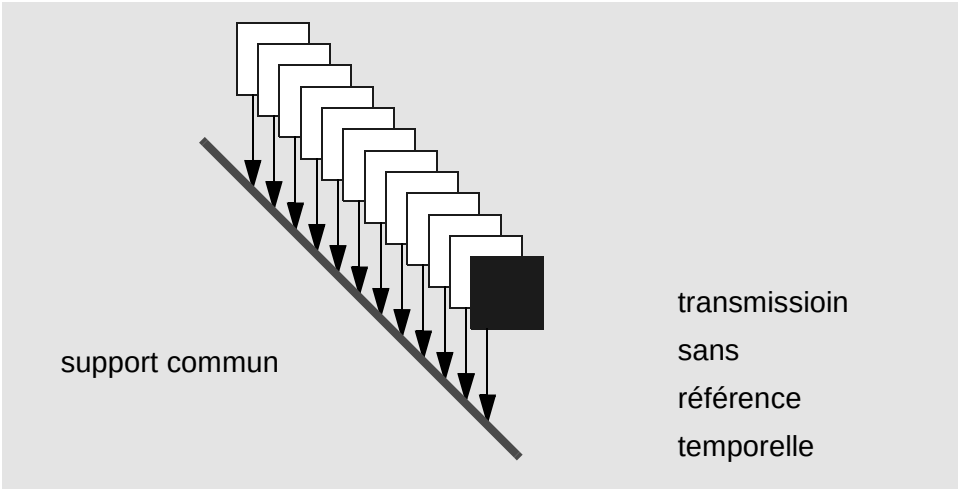
Fonctionnement

Un port de communication Ethernet est partagé entre tous les coupleurs raccordés. Le débit nominal est donc partagé selon des besoins dans le temps entre tous les coupleurs raccordés. Le protocole d'accès d'Ethernet repose fondamentalement sur l'accès aléatoire dérivé du protocole Aloha. Ce mode d'accès est totalement autonome, a priori sans connaissance des autres coupleurs reliés au réseau.

Principe du protocole Aloha

En 1970, Norman Abramson et ses collègues de l'université de Hawaï (situé sur l'île ALOHA) ont inventé une nouvelle méthode permettant de résoudre le problème de l'allocation du droit à émettre. L'idée de base est applicable à tout système dépourvu d'unité de coordination et dont les stations sont en compétition pour utiliser le canal unique qu'elles doivent se partager. Dans un système Aloha pur, les trames sont transmises sans aucune référence temporelle.

FIGURE 32. Accès aléatoire sans référence temporelle



Appelons **durée d'une trame** le temps moyen nécessaire à la transmission d'une trame. Durée d'une trame est définie par: **longueur d'une trame (en bits) / débit binaire nominal (en bits/seconde)**.

Considérons une population infinie d'utilisateurs générant des trames selon une distribution de Poisson où **S** est le nombre moyen de trames par **durée d'une trame**. **G** représente le pourcentage moyen de trames transmises (trames initiales plus trames retransmises).

A faible charge, avec **S** voisin de 0, il y aura très peu de collisions, donc très peu de retransmissions, et **G** sera très peu différent de **S**. A forte charge, il y aura beaucoup de collisions, et **G** sera grand par rapport à **S**.

Quelle que soit la charge, le débit utile est égal au débit offert **G**, multiplié par le coefficient de probabilité pour qu'une trame transmise le soit avec succès $\mathbf{P_0}$; il en résulte que:

$$\mathbf{S = G*P_0}.$$

Remarque:

Une trame passe sans collision si aucune autre trame n'est transmise pendant la période comprise entre son début et sa fin d'émission.

FIGURE 33. Trames effectivement envoyées (S) et charge du lien (G)

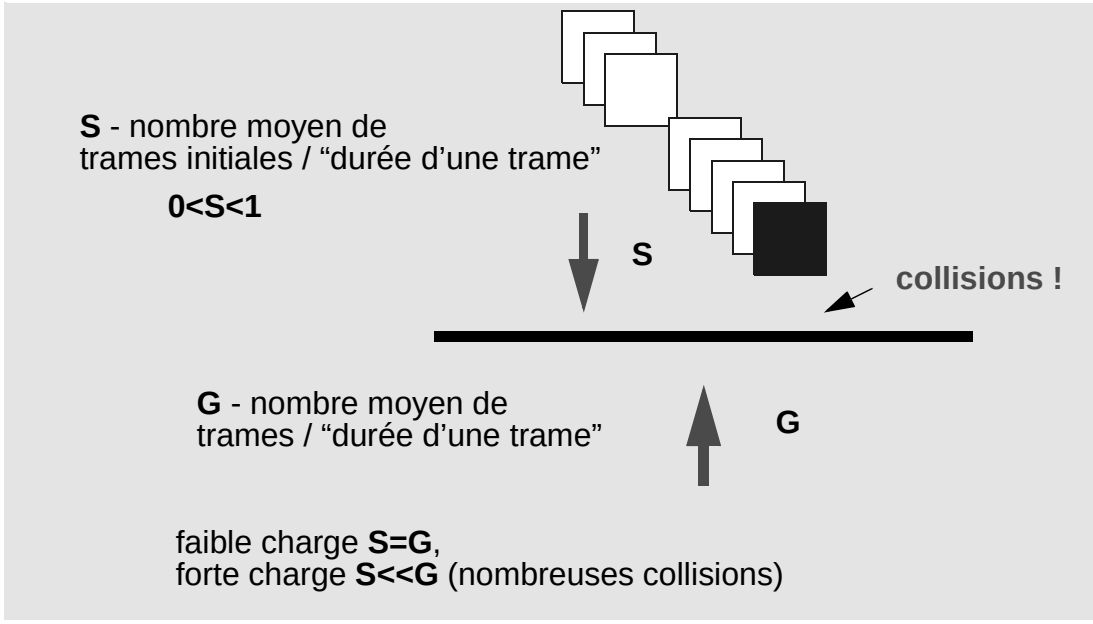
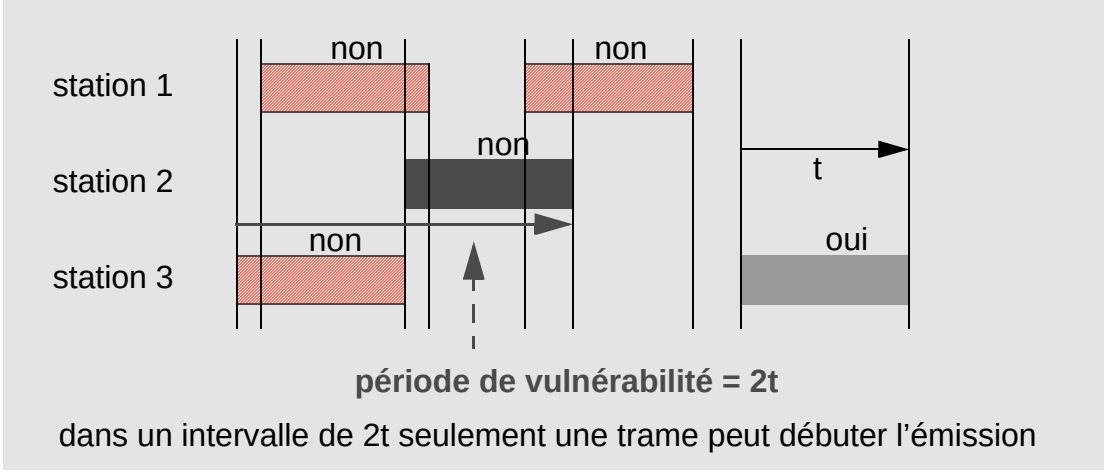


FIGURE 34. Période de vulnérabilité dans le protocole Aloha pur



La probabilité pour que **k** trames soient générées pendant un temps égal à la durée d'une trame est donnée par la formule de la distribution de Poisson:

$$\textbf{probabilité} = G^k * e^{-G}/k!$$

aussi la probabilité d'avoir 0 trame est de la forme e^{-G}.

Dans un intervalle de durée égale à 2 "durées de trame", le nombre moyen de trames générées est égal à **2G**. La probabilité qu'aucun trafic supplémentaire ne soit généré pendant la totalité de cette période de vulnérabilité est donnée par $P_0=e^{-2G}$.

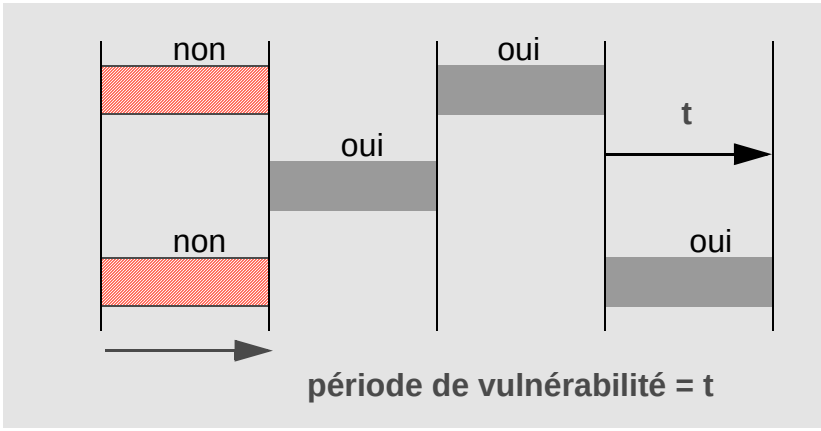
En posant $S=G*P_0$ nous obtenons:

$$S=G*e^{-2G}$$

La relation entre le trafic offert (charge) et le trafic effectivement acheminé est représentée ci-dessous.

Le trafic maximum est obtenu pour **G=0.5** avec $S_{max}=1(2e)$ dont la valeur environ 0,184 (18%). L'introduction des intervalles répétitifs de durée constante permet de doubler la capacité de transmission $S_{max}=1/e$. Comme on peut le voir sur la figure ci-dessus l'Aloha discrétisé présente un maximum pour **G=1**, avec un trafic écoulé de $S_{max}=1/e$.

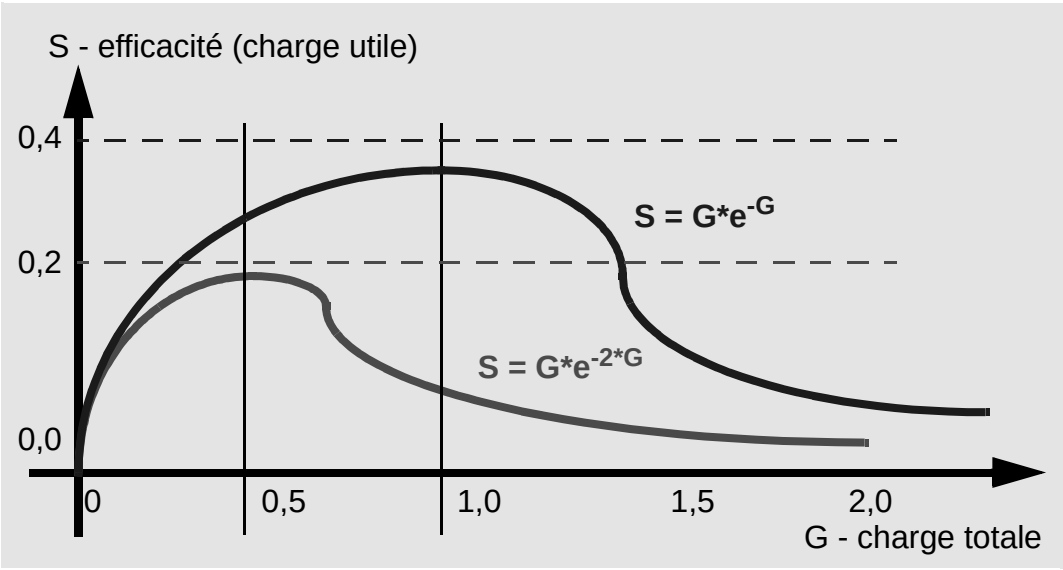
FIGURE 35. Période de vulnérabilité pour le protocole Aloha discrétisé



Dans cette méthode, qui est devenue le *slotted* Aloha ou Aloha discrétisée, une station doit attendre le début d'un intervalle de temps pour commencer à émettre. La période de vulnérabilité est limitée à la durée d'une trame; ce qui nous conduit à:

$$S=G*e^{-G}$$

FIGURE 36. Efficacité des protocoles Aloha pur et aloha dicrétisé



Pour prendre conscience de la rapidité de la croissance des collisions lorsque **G** augmente, remarquons que la probabilité qu'une trame évite la collision est égale à $\mathbf{e^{-G}}$; c'est en fait la probabilité que les autres stations ne transmettent pas pendant la période d'émission de notre trame.

La probabilité qu'il y ait une collision est égale à $\mathbf{1-e^{-G}}$.

La probabilité pour que la transmission nécessite exactement **k** tentatives, c'est-à-dire **k-1** collisions suivies d'une transmission réussie, est:

$$\mathbf{P_k = e^{-G} * (1-e^{-G})^{k-1}}$$

Le nombre de transmissions E, pour chaque tentative d'émission, est alors:

$$\mathbf{E = \Sigma k * P_k = \Sigma\, k * e^{-G} * (1-e^{-G})^{k-1} = e^{G}}$$

Exemple:

Slotted Aloha avec **G=1** pour le trafic maximal,

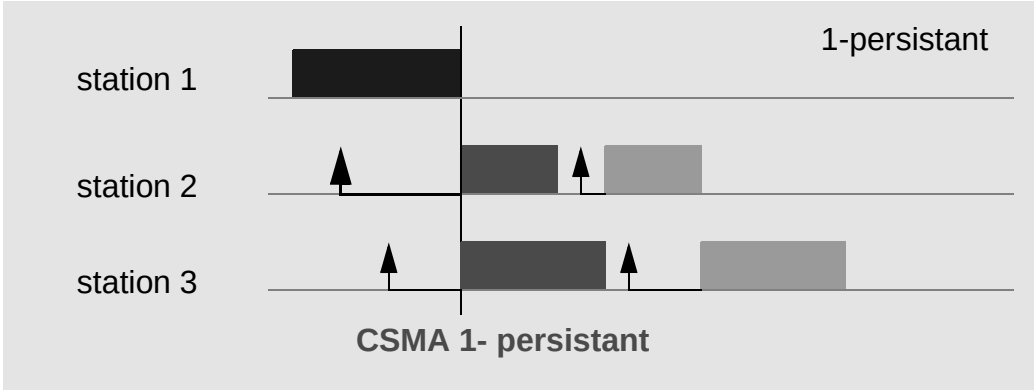
$$\mathbf{E = e^{G} = e^{1} = e = 2,7..}$$

La dépendance exponentielle de **E** par rapport à **G** est telle que de faibles accroissements de la charge du canal réduisent énormément ses performances.

Protocole avec l'écoute de la porteuse - CSMA persistant

Le premier protocole CSMA (*Carrier Sense Multiple Access* - accès multiple par l'écoute de la porteuse) est appelé "1-persistant" car la station qui veut émettre maintient l'écoute du canal et émet dès qu'elle constate qu'il est devenu libre (sans porteuse).

FIGURE 37. Protocole CSMA 1-persistant

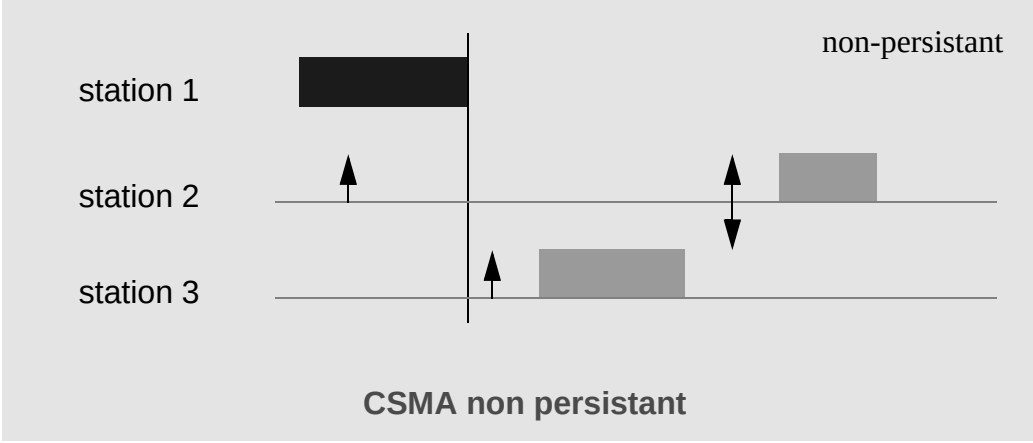


Le délai de propagation a une incidence importante sur les performances de ce protocole. Ils est possible qu'immédiatement après qu'une station a commencé à émettre, une autre station en écoute du canal soit prête à émettre sa trame.

Protocole CSMA non-persistant

Si les deux stations prêtes à émettre étaient plus patientes il y aurait moins de collisions. Le protocole CSMA non-persistant accepte délibérément d'être moins pressé d'émettre sa trame que le protocole 1-persistant. Avant d'émettre sa trame la station écoute le canal; s'il est disponible, elle émet la trame, mais s'il n'est pas disponible, **elle ne reste pas en écoute permanente**. Après une période d'attente, la station reprend le scénario; elle écoute à nouveau la porteuse et émet si le canal est libre.

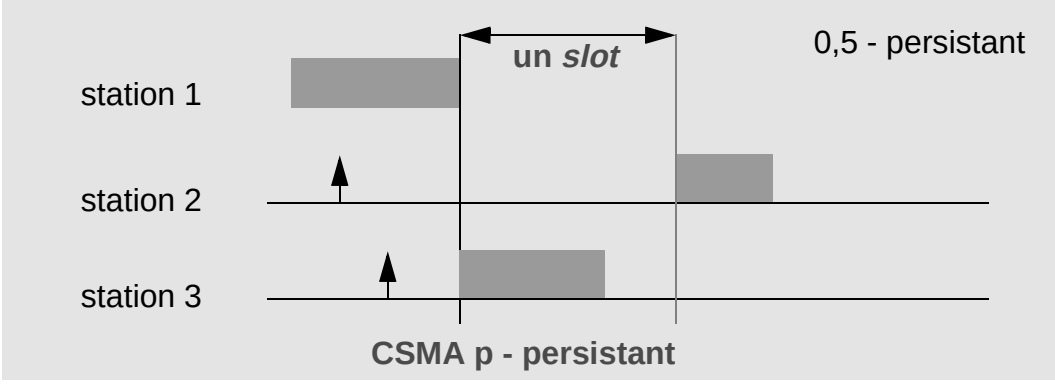
FIGURE 38. Protocole CSMA non-persistant



Protocole CSMA p-persistant

Le protocole CSMA p-persistant s'applique aux canaux utilisant des intervalles de temps pour le contrôle d'accès et fonctionne comme suit. La station qui veut émettre écoute le canal. Si le canal est disponible, elle émet avec la probabilité égale à **p**. La probabilité qu'elle attende les intervalles de temps suivants est **q=1-p**; Si, lors du début du deuxième intervalle de temps le canal est disponible les probabilités d'émettre et de non-émettre sont les mêmes (**p** et **q=1-p**).

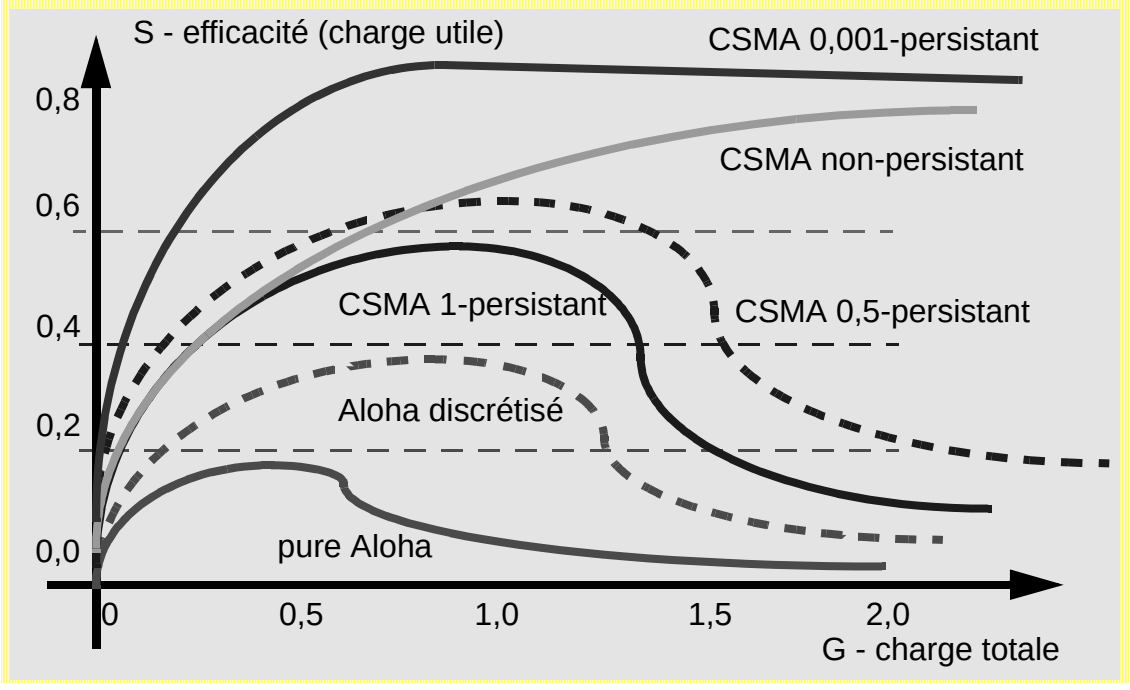
FIGURE 39. Protocole CSMA 0,5-persistant



Comparaison de l'efficacité des protocoles CSMA

La figure ci-dessous représente pour chacun de protocoles Aloha/CSMA, le débit utile par rapport au trafic offert.

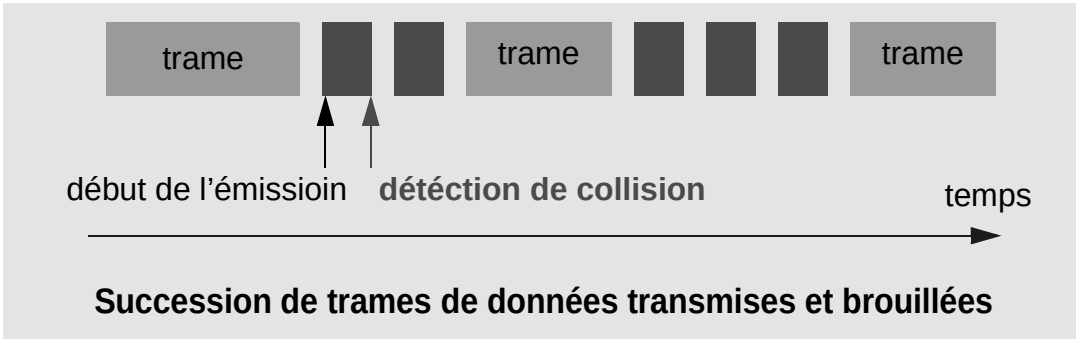
FIGURE 40. Comparaison de l'efficacité des protocoles Aloha et CSMA



Protocole CSMA/CD

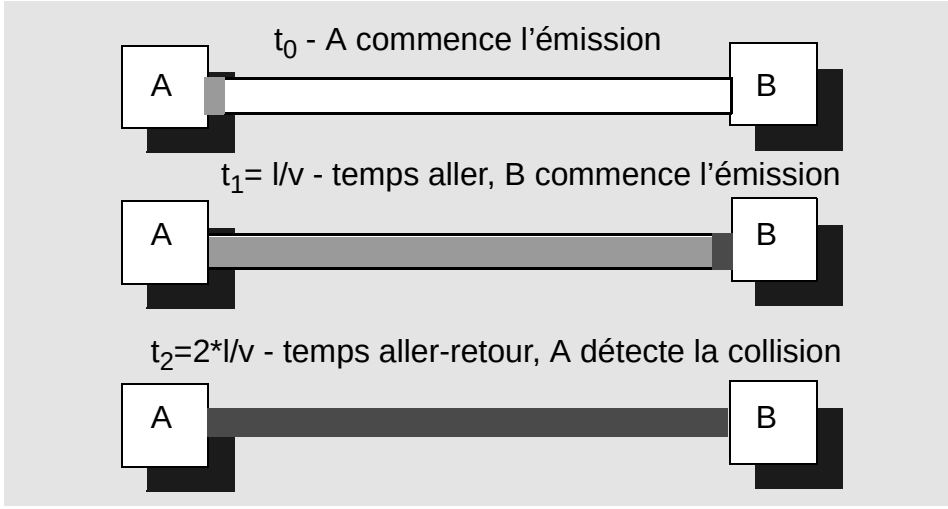
Les protocoles CSMA permettent d'améliorer le rendement du protocole ALOHA car ils apportent la certitude que les stations se garderont d'émettre si elles constatent qu'une autre station est en train d'émettre. Une autre amélioration consiste à introduire un mécanisme permettant, en cas de collision, d'arrêter immédiatement la suite de transmission d'une trame. En d'autres termes, les stations émettrices voyant que la trame entre en collision cessent d'émettre. Elles vont recommencer la transmission (si le support est libre) après une durée aléatoire. L'application du protocole CSMA/CD implique une succession de périodes de transmission et de détection de collision entrecoupées de périodes de transmission.

FIGURE 41. Protocole CSMA/CD - détection de collisions



Le temps minimum pour détecter une collision est égal au temps nécessaire pour que le signal émis par une station parvienne à l'autre station. Considérons le cas le plus défavorable avec **t** représentant le temps nécessaire pour joindre les deux stations les plus éloignées l'une de l'autre. A l'instant $\mathbf{t_0}$ la station **A** commence à émettre. La station **B** commence également à émettre en moment $\mathbf{t_1}$, juste avant que le signal de la station **A** lui arrive. La perturbation provoquée par l'émission de la station **B** ne parviendra à la première station qu'après un délai de **2t**; où **t** est le temps de propagation du signal entre deux stations.

FIGURE 42. Détection d'une collision - temps aller-retour



Pour un câble coaxial mesurant 1 kilomètre, **le temps de propagation est de l'ordre de 5 μs**. Il en résulte que pour une ligne de **L** km le temps maximum nécessaire pour détecter une collision est:

$$\mathbf{T_{col/max} = 2 * L * 5\mu s}$$

Si nous utilisons un support débitant **D** bits par seconde, la longueur minimale d'une trame **F** pour que la collision puisse être détectée est:

$$\mathbf{F = D * T_{col/max}}$$

Exemple:

L = 2 km, D = 5 Mbit/s

F = 5 Mbit/s * 2* 2 km * 5 μs = 100 bits

Norme MAC IEEE 802.3 -ETHERNET

La norme IEEE 802.3 appartient à la famille des réseaux locaux de type CSMA/CD. Elle a pour origine le système Aloha. Les fonctions “écoute de la porteuse” et “détection de collision” furent ajoutées ensuite par Xerox qui construisit un système CSMA/CD à 2,94 Mbit/s fonctionnant sur un câble de 1000 mètres de long et permettant de connecter 100 stations de travail. Ce système a été appelé ETHERNET.

Le système Xerox ETHERNET a été la base d’un nouveau standard défini par DEC, Xerox, et Intel pour un réseau au débit nominal de 10 Mbit/s. Le standard IEEE 802.3 diffère des spécifications ETHERNET dans ce sens qu’il spécifie toute une famille de protocoles CSMA/CD avec différents types de supports physiques: câbles coaxiaux, fibre optique, paires torsadées,.. (les câbles coaxiaux sont de moins en moins utilisés).

Ethernet se définit comme une technologie employant une topologie en bus, mais il est de plus en plus souvent constitué par un câblage arborescent. Le débit standard de 10 Mbit/s est porté à 100 Mbit/s et plus récemment à 1 Gbit/s.

Le réseau Ethernet fonctionne (en principe) en mode *half-duplex*. Le signal émis par une station se propage à partir de son *transceiver* sur l’ensemble du réseau. La longueur d’un réseau Ethernet ne peut pas dépasser 4 km pour 10 Mbit/s et 412 m pour 100 Mbit/s.

Un réseau Ethernet est composé de cartes interfaces coupleur, de câbles, des *transceivers*, de répéteurs, de *hubs* et de commutateurs.

- le coupleur - (NIC - *Network Interface Card*) est une interface presque toujours dotée d’un simple port RJ45, l’interface coupleur est exploitée par un *driver* logiciel; le driver fait la liaison entre le système informatique et le transfert des octets en sortie et en entrée,
- le *transceiver* - (*transmitter-receiver*) est l’équipement chargé de la réception et de la transmission sur le media. Il peut être intégré à la carte (circuit) coupleur. Il possède un ou plusieurs accès (DB 15, RJ45, ..) selon le media employé,
- le répéteur - permet d’étendre le réseau au-delà d’un seul segment. Il recopie les signaux amplifiés entre deux (ou plus) points de connexion simulant le comportement d’un bus. Etant le centre d’une topologie étoilée dans le cas de 10Base-T, ils sont appelés *hubs* (moyeux).

FIGURE 43. Circuit coupleur-*transceiver* 10BASE2

FIGURE 44. Répéteurs du signal logique

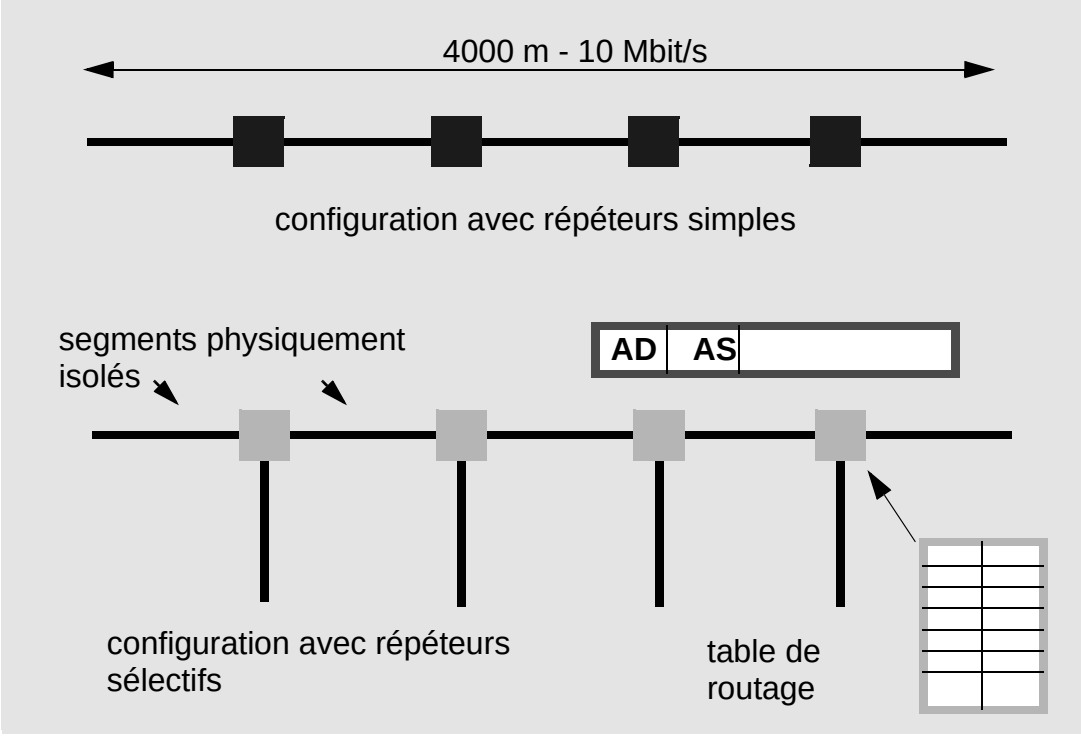


FIGURE 45. Un répéteur Ethernet 10BASE-T

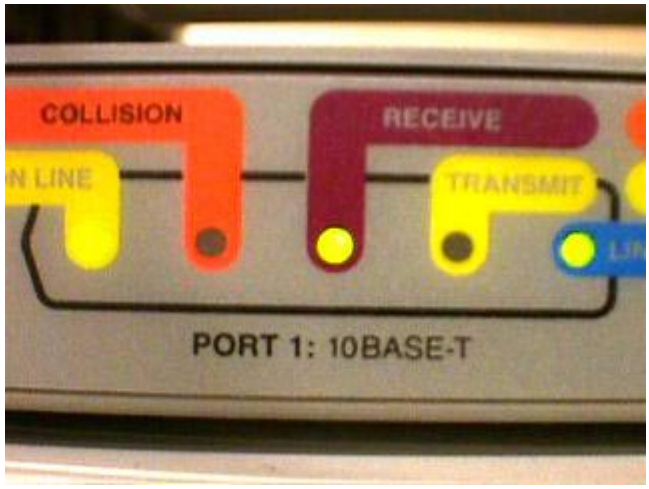


FIGURE 46. **Configuration réseau basée sur un concentrateur et un *hub***

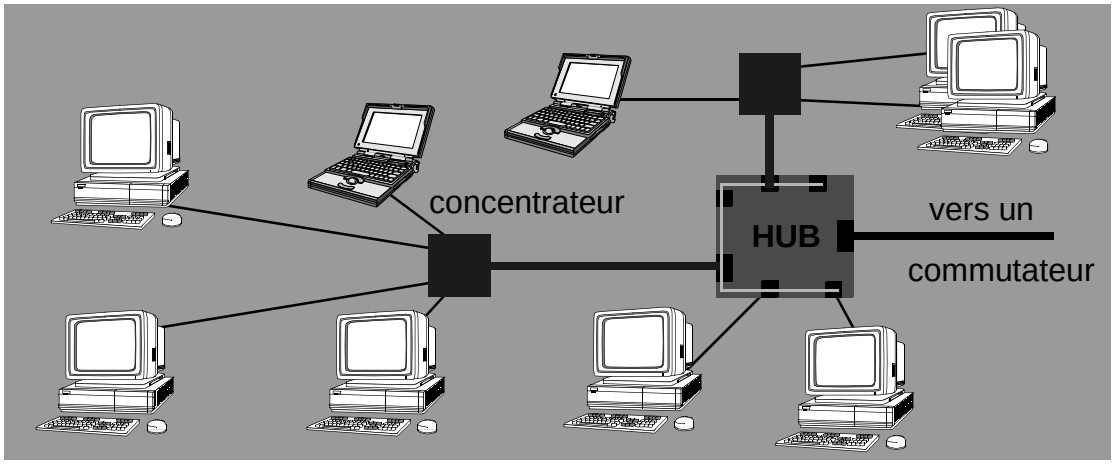
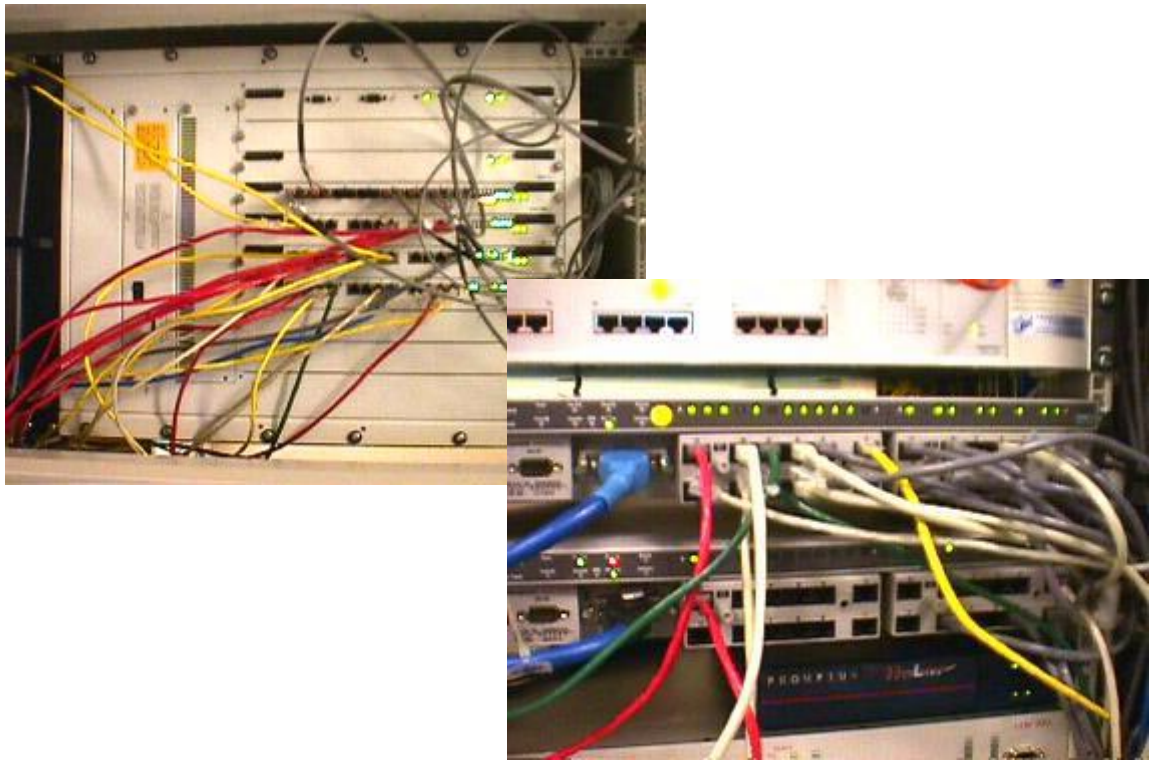


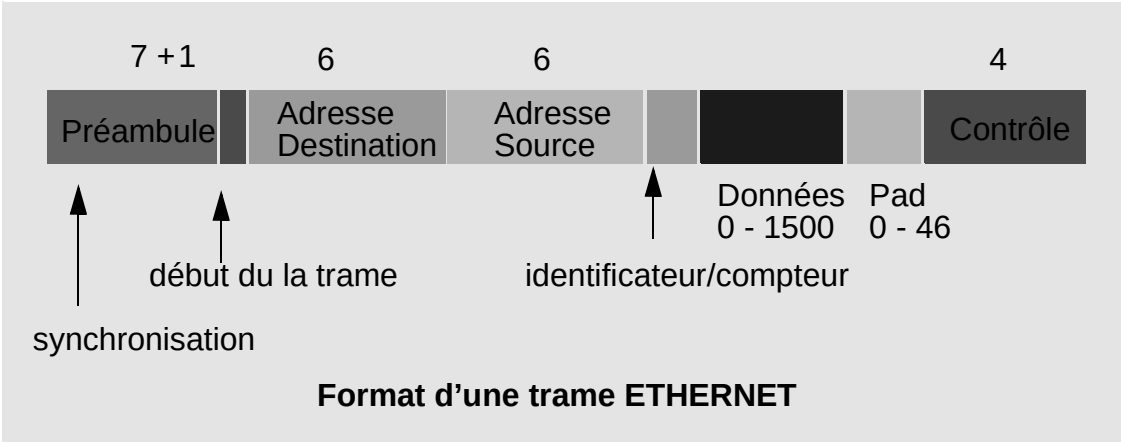
FIGURE 47. **Un *hub* et un *switch* pour réseau Ethernet**



Fonctionnement

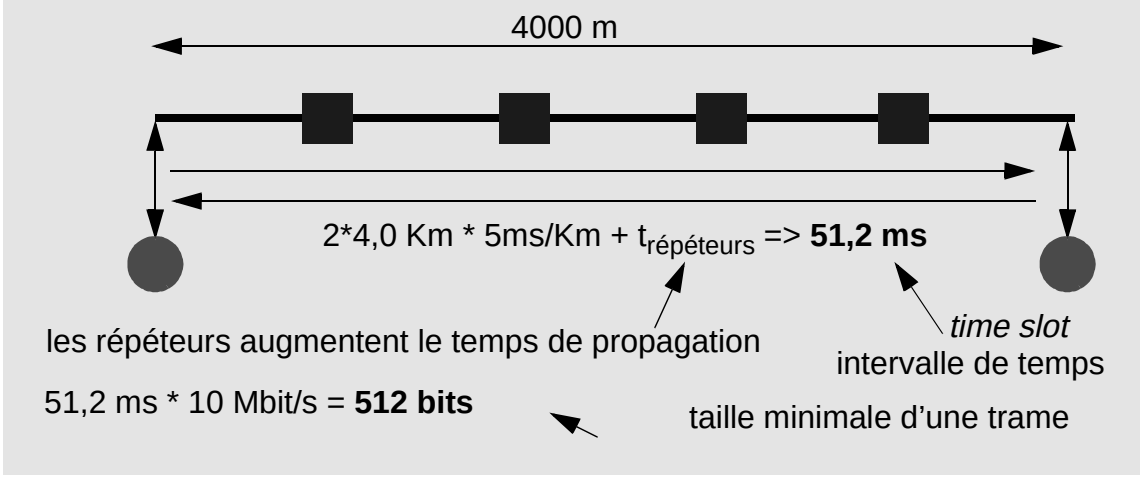
Chaque trame ETHERNET débute par une amorce, préambule de 7 octets, chacun d'eux contenant `10101010`. Après l'encodage en code Manchester direct, cette amorce fournit une onde rectangulaire d'une fréquence de 10 MHz permettant de synchroniser la réception des bits. L'octet suivant contient le marqueur de début - `10101011` de trame. Chaque trame contient deux adresses physiques - l'adresse destination et l'adresse source, un compteur/identificateur, le champ de données et une somme de contrôle.

FIGURE 48. Structure d'une trame Ethernet



Le champ suivant contient le compteur/identificateur du champs de données. Depuis peu, ce champ est exploité comme indicateur du protocole porté par la trame (e.g. protocole IP). La taille des données est exprimée en octets et comprise entre 0 et 1500. Le champ *Pad* sert à compléter la taille d'une trame à 72 (64) octets minimum. Le protocole MAC de l'ETHERNET est du type CSMA/CD. Dans ce protocole, toute station détectant une collision des trames arrête immédiatement sa transmission et génère une séquence de brouillage pour avertir les autres stations et attend pendant un temps aléatoire avant de recommencer la nouvelle émission.

FIGURE 49. Calcul de la taille minimale de trame Ethernet



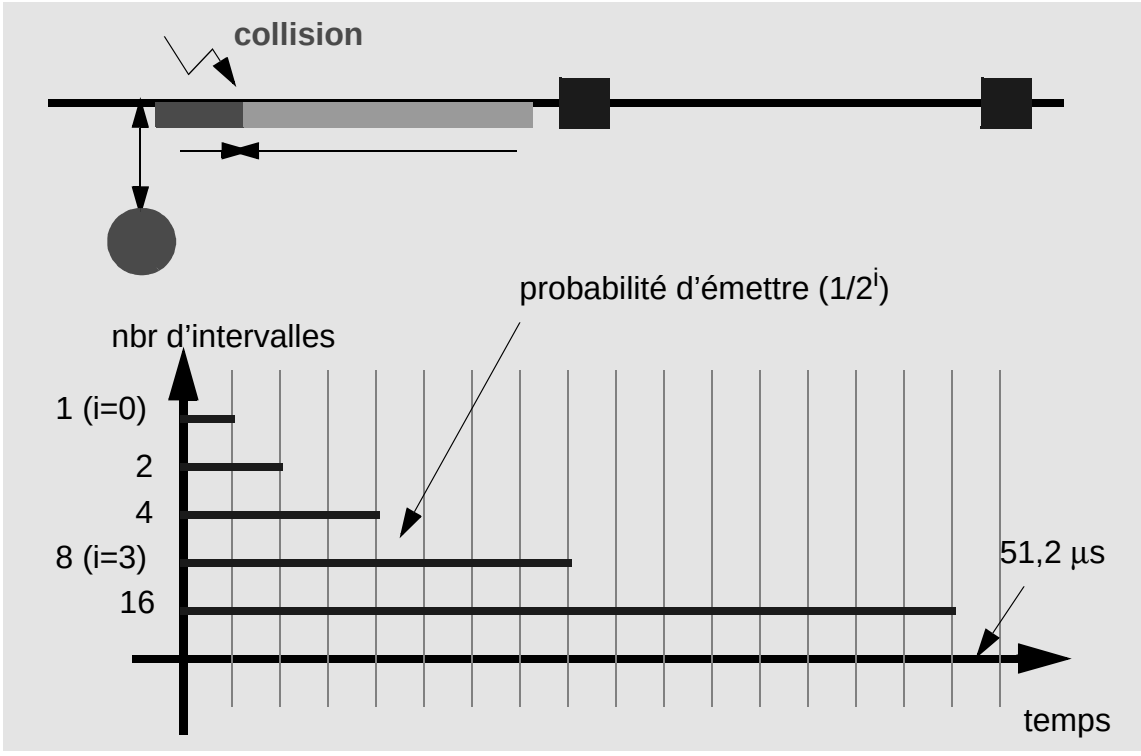
Après une collision, le temps qui s'écoule est divisé en intervalles de temps dont la durée est égale au temps aller-retour entre deux stations éloignées le plus, c'est-à-dire 4 Km.

La gestion des collisions

Un intervalle de temps ou *time slot* ETHERNET est de 51,2 µs. Une collision entre deux trames ETHERNET peut être détectée à coup sûr pendant ce temps. Après la première collision, chaque station attend 0 ou 1 intervalle de temps pour émettre. Après la deuxième collision, elle attendra de 0 à 3 intervalles. En général, après **i**

collisions elle attendra 2^i-1 intervalles de temps, avec un maximum de 1023 intervalles. Après la seizième collision consécutive, le coupleur de réseau cesse d'émettre et signale la situation à la station de travail.

FIGURE 50. Protocole CSMA/CD de l'Ethernet



Différentes versions de la couche physique Ethernet

Pour répondre aux besoins variés en termes de débit, de longueur et de nombre de stations raccordées, la norme 802.3 offre trois versions de communication en bande de base et une en bande large : 10BASE5, 10BASE2, 10BASE-T, 100BASE-T, 100BASE-T4, 100BASE-TX, 1000BASE-T, ..

Les noms des versions se lisent comme suit:

<débit><type_de_medium><longueur_max_de_segment>

Le débit est exprimé en Mbit/s. Le type désigne soit un médium en bande de base, soit un médium en large bande.

TABLEAU 6. Différentes versions de la couche physique Ethernet

PARAMETRE	10BASE2	10BASE-T	100BASE-T	1000BASE-T
type de médium	coax 75 ohms	paire torsadée		paire torsadée
codage	Manchester	Manchester	Manchester (4B/ 5B)	m-phase (NRZ)
débit en Mbit/s	10	10	100	1000

TABLEAU 6. Différentes versions de la couche physique Ethernet

PARAMETRE	10BASE2	10BASE-T	100BASE-T	1000BASE-T
longueur maximum de segment	185	100		1800
étendue max. du réseau en metres	925	200		3600
diamètre du cable en millimetres	coax fin (50 Ohms)	paire torsadée UTP 3/4	paire torsadée UTP 3/4/5	inconnu

GigaEthernet

La volonté de pouvoir constituer des réseaux fonctionnant en mode *half-duplex* sur un bus logique impose à l'Ethernet sur 1Gbit/s des contraintes très fortes. En multipliant le débit par 10 (100 => 1000 Mbit/s) on divise le temps d'émission d'une trame par 10. En conséquence, une trame de 64 octets est émise en 0,51 µs. Le temps de propagation aller-retour devient trop court car il correspond à 40 m. La solution consiste à modifier la taille minimale du *slot-time*, en la faisant passer **de 512 bits à 512 octets**. Il en résulte qu'un réseau partagé constitué d'un lien en fibre optique peut atteindre 316 m.

Le code physique employé en technologie 1000BASE-X est de type **8B**/10B. Une horloge d'émission des **octets** est cadencée à 125 MHz.

Le câblage est effectué en fibre optique **multi-mode**, en paire torsadée blindée (STP) ou en quatre paire torsadées (UTP5). La longueur d'un segment est limitée à 25 m pour la solution STP et à 100 m pour les paires UTP5.

Performance du protocole Ethernet

Nous allons maintenant examiner brièvement les performances du protocole employé par l'Ethernet. Si chaque station de travail transmet durant un intervalle de temps avec la probabilité **p**, la probabilité **A** pour qu'une station prenne possession du support de transmission est:

$$A = k*p*(1-p)^{k-1}$$

avec un maximum tendant vers **1/e** lorsque **p=1/k.**

La probabilité pour que la période de contention comporte exactement **j** intervalles de temps est de la forme:

$$A*(1-A)^{j-1}$$

aussi, le nombre moyen d'intervalles de temps par période de contention est-il donné par la formule:

$$\Sigma j*A*(1-A)^{j-1} = 1/A$$

Comme chaque intervalle de temps a une durée de 2τ, la valeur moyenne de la période de contention est $2\tau/A$. Pour la valeur optimum de **p**, le nombre moyen d'intervalles de temps n'est jamais supérieur à **e**; soit très peu

différent de **5,4**τ. Si la durée moyenne de transmission d'une trame est de **P** secondes, l'efficacité **E** du canal est:

$$E=P/(P+2\tau/A)$$

Il est instructif de réformuler la dernière équation en fonction de la longueur des trames **F**, du débit nominal **B**, de la longueur du câble **L**, et de la vitesse de propagation des signaux **V**. Pour le cas optimum où **e** intervalles de temps séparent deux trames transmises, avec **P=F/B** nous obtenons:

$$E = 1/(1+2*B*L*e/V*F)$$

Lorsque le deuxième terme du numérateur est grand, l'efficacité du protocole est faible. Afin d'améliorer les performances, il faudrait augmenter la longueur de la trame (**F**), ou diminuer la longueur du câble (**L**) ou le débit (**B**). Malheureusement, la plupart des extensions du réseau Ethernet conduisent à augmenter la valeur du produit **B*L**.

Commutation et réseaux commutés Ethernet

Dans le développement des réseaux Ethernet il est possible d'introduire trois sortes d'équipements de type commutateur: les commutateurs *stand-alone*, les commutateurs modulaires et les commutateurs empilables.

Les **commutateurs *stand-alone*** comportent une ou deux dizaines de ports 10 Mbit/s et un ou deux ports haut débit.

Les **commutateurs modulaires** incorporent un coeur de commutation à très haut débit (plusieurs Gbits par seconde). Ces équipements peuvent recevoir, via l'insertion de modules appropriés, des ports à débit courant et à haut débit. Il supportent paires torsadées, câble coaxial et fibres optiques. Ils peuvent être adaptés à une seule technologie (Ethernet) ou à plusieurs (Ethernet, Token Ring, FDDI, ATM) en intégrant des capacités de pontage à translation et conversion de trames.

Les **commutateurs empilables** reprennent la philosophie modulaire sous la forme de boîtiers autonomes. Un commutateur empilable offre un étage de commutation quel que soit le nombre d'éléments de la pile. Il dispose d'un coeur de commutation accessible à toutes les unités de la pile. Avec cette technologie, des débits à quelques gigabits par seconde doivent circuler à l'extérieur des unités.

Topologie d'un réseau commuté

Une architecture commutée peut être composée d'un commutateur central connecté en périphérie à des concentrateurs. Les serveurs sont liés directement au commutateur central. Les *hubs* sont reliés au commutateur par des liens traditionnels et communiquent avec l'ensemble des stations réparties dans le bâtiment.

Certains concentrateurs peuvent être remplacés par des commutateurs périphériques; il peuvent ainsi alimenter les stations de travail avec bande passante dédiée en full-duplex.

Les débits de coeurs des commutateurs LAN sont des multiples du gigabit par seconde. Les délais de traversée des commutateurs sont notablement plus importants que les délais imposés par les répéteurs sur un support partagé. Heureusement, ce délai dû au stockage des trames à l'intérieur d'un commutateur est inversement proportionnel au débit.

L’architecture d’un commutateur Ethernet

Il existe deux techniques d’implémentation pour les commutateurs Ethernet:

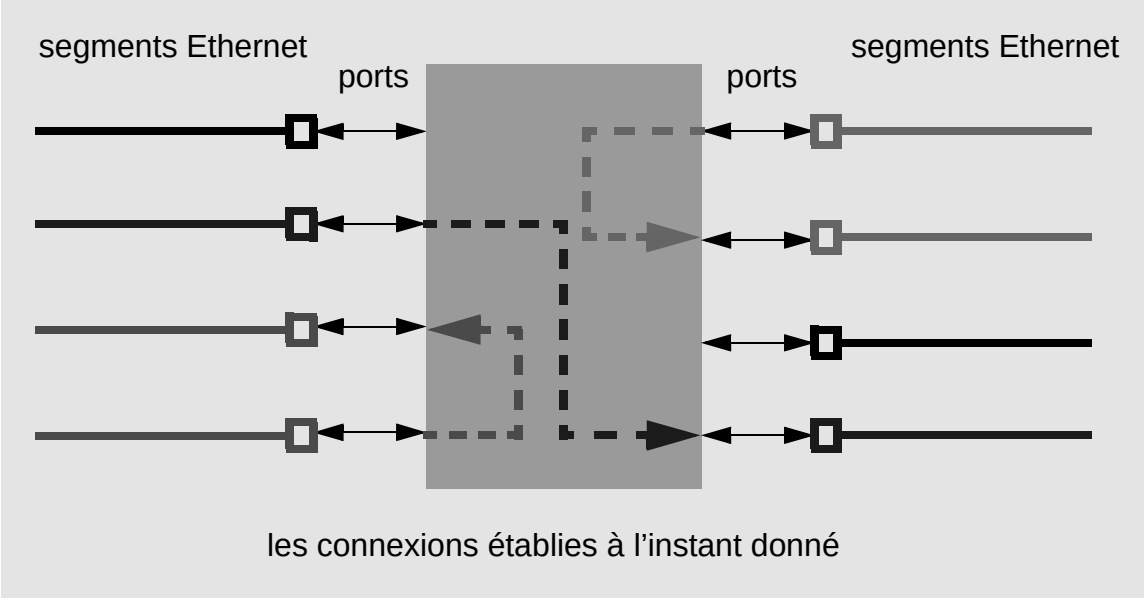
- les implémentations basées sur les processeurs RISC,
- les implémentations basées sur les circuits ASIC.

Parmi les avantages de la solution RISC nous trouvons:

- le prix peu élevé - les processeurs RISC sont disponibles sur le marché à grande échelle,
- le traitement complet incluant l’analyse des adresses - les fonctions des routeurs peuvent être facilement intégrées,
- la modification facile des paramètres et des protocoles exécutés.

La solution ASIC permet d’obtenir une plus grande performance de traitement.

FIGURE 51. Architecture simplifiée d’un commutateur Ethernet



Architectures *Store-and-Forward* et *Cut-through*

Il y a deux types d’architectures de commutation importants pour les réseaux Ethernet:

- architecture *cut-through* (remise immédiate),
- architecture *store-and-forward* (mémoriser et faire suivre).

Dans l’architecture type *cut-through,* le commutateur émet des paquets dès que l’adresse de destination est lue (20 à 30 premiers octets de la trame); les trames sont donc retransmises très rapidement mais le commutateur ne peux pas arrêter les trames incorrectes.

L’architecture type *store-and-forward* contient des tampons pour enregistrer les trames avant leur retransmission sur le réseau suivant. La présence de ces tampons permet de vérifier le code CRC et d’abandonner les trames erronées. Par contre, le délai de retransmission est important, au moins égal à la «longueur» d’une trame.

Remarque: Les commutateurs qui lient des réseaux Ethernet aux débits différents (e.g. 10 Mbit/s et 100 Mbit/s) fonctionnent toujours en mode *store-and-forward.*

Performance de commutation

Débit

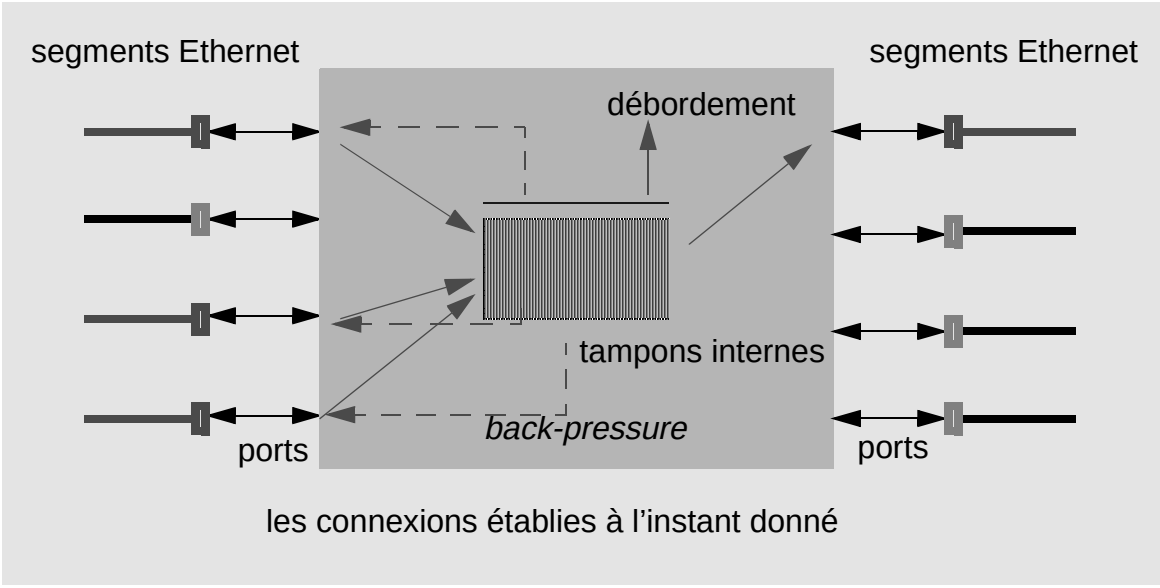
Le débit global interne du commutateur doit être égal ou supérieur à la somme des débits sur l'ensemble des liens d'entrée. Dans le cas d'un commutateur 10BASE-T à N ports, ce débit doit être supérieur à N/2*10 Mbit/s en termes de bits et 14 880 *pps* (*packets per second*) en termes de paquets.

Congestion et Latence

Plusieurs flux de paquets/trames envoyés sur le même port de sortie peuvent provoquer une congestion. Afin d'éviter ce type de problèmes, les architectures de commutations intègrent les mécanismes de stockage (tampons FIFO) et utilisent les schémas de priorité.

Parfois, on utilise la technique de *back pressure* pour informer les émetteurs du problème de congestion. Par exemple, le commutateur peut mettre en état actif son propre émetteur sur le port congestionné.

FIGURE 52. Congestion et *back-pressure* dans un commutateur Ethernet



Réseaux locaux sans fil (réseaux locaux hertziens)

Avec l'encombrement des locaux et les difficultés de tirer des câbles dans les salles utilisées occasionnellement, il est important de pouvoir déployer des réseaux locaux sans fil - WLAN (*Wireless* LAN)

Un tel réseau est immédiatement opérationnel du moment qu'une station de base et des coupleurs à haute fréquence sont disponibles et activés.

Les principales caractéristiques d'un réseau local sans fil sont les suivantes:

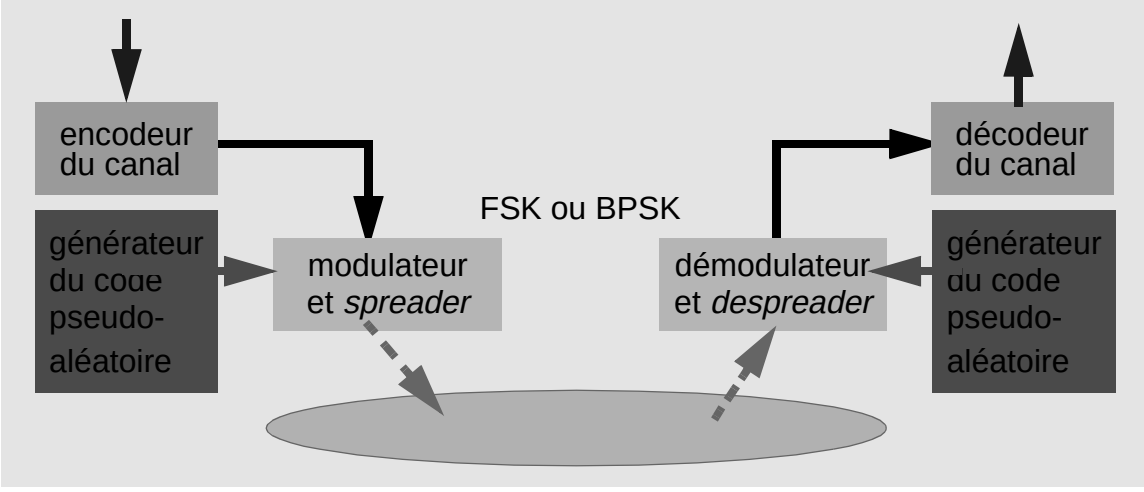
- le débit - le protocole d'accès doit être efficace pour maximiser l'utilisation de la capacité de transmission d'un canal hertzien,
- la couverture - le diamètre d'une zone desservie par un WLAN doit être de l'ordre de 100 à 300 m,
- le nombre de noeuds - un WLAN doit supporter plusieurs dizaines de noeuds,
- connexion au *backbone* - un réseau WLAN doit être connecté à une magistrale de communication (*backbone*)
- la qualité et la sécurité - le protocole d'accès du WLAN doit être robuste et fiable,
- la configuration dynamique - la gestion des adresses MAC doit être dynamique et automatique.

La technologie la plus usuelle pour le déploiement des réseaux WLAN est basée sur le spectre étalé (*spread spectrum*).

Communication en mode *spread spectrum*

La technique du *spread spectrum* est basée sur un étalement du signal sur un large spectre des fréquences disponibles sur le canal de communication. Dans ce contexte, deux techniques sont exploitées, le **saut de fréquence** et le **séquencement direct**.

FIGURE 53. Modèle général d'un système de communication basé sur *spread spectrum*



Le signal originel est porté par une bande de fréquences relativement limitée autour d'une fréquence centrale. Ce signal est ensuite modulé moyennant une séquence de valeurs pseudo-aléatoires. Le résultat de la modulation est un signal étalé (*spread*) sur la totalité de la bande passante disponible.

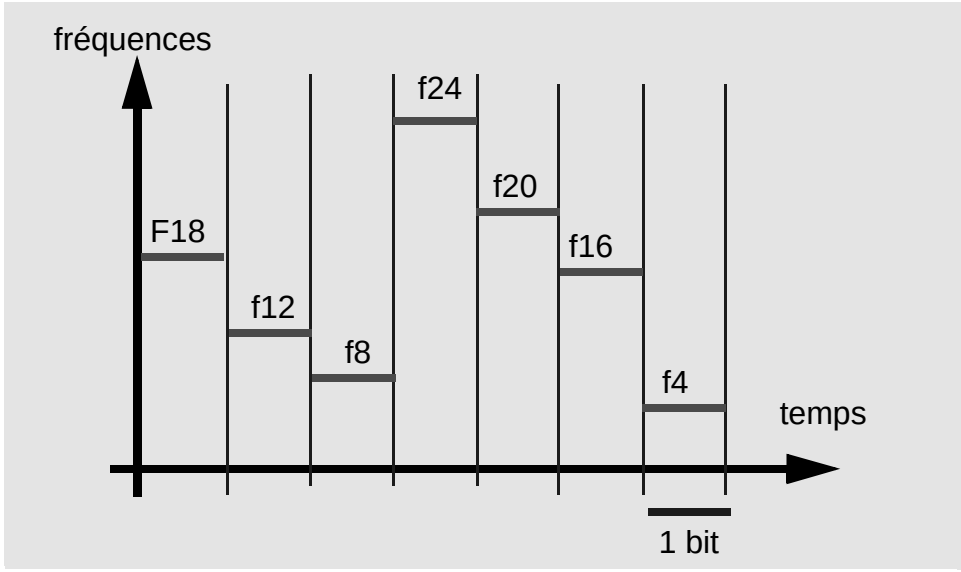
A la réception, la même séquence pseudo-aléatoire est appliquée pour démoduler le signal originel. L'algorithme qui génère la séquence pseudo-aléatoire est basé sur une valeur initiale appelée *seed* (graine). Seulement les postes qui connaissent le *seed* et l'algorithme de modulation peuvent écouter le signal émis par leur(s) correspondant(s).

Saut de fréquence

Pour la transmission des données binaires on utilise un modulateur avec un schéma d'encodage type FSK (*frequency-shift keying*) ou BPSK (*binary phase-shift keying*). Le signal résultant est centré (déplacé) sur une fréquence de base séléctionnée par le générateur de nombres pseudo-aléatoires.

Par exemple, dans le cas de la modulation FSK, le modulateur sélectionne la fréquence **f0** pour la valeur '0' et la fréquence **f1** pour la valeur '1'. Ensuite la fréquence **f0** ou **f1** est augmentée par une fréquence déterminée par le générateur des valeurs aléatoires à l'instant (bit) donné - **fi**. Nous obtenons ainsi deux fréquences qui peuvent être émises: **f0+fi** pour le bit '0' et **f1+fi** pour le bit '1'.

FIGURE 54. Saut de fréquence : à chaque bit correspond une fréquence



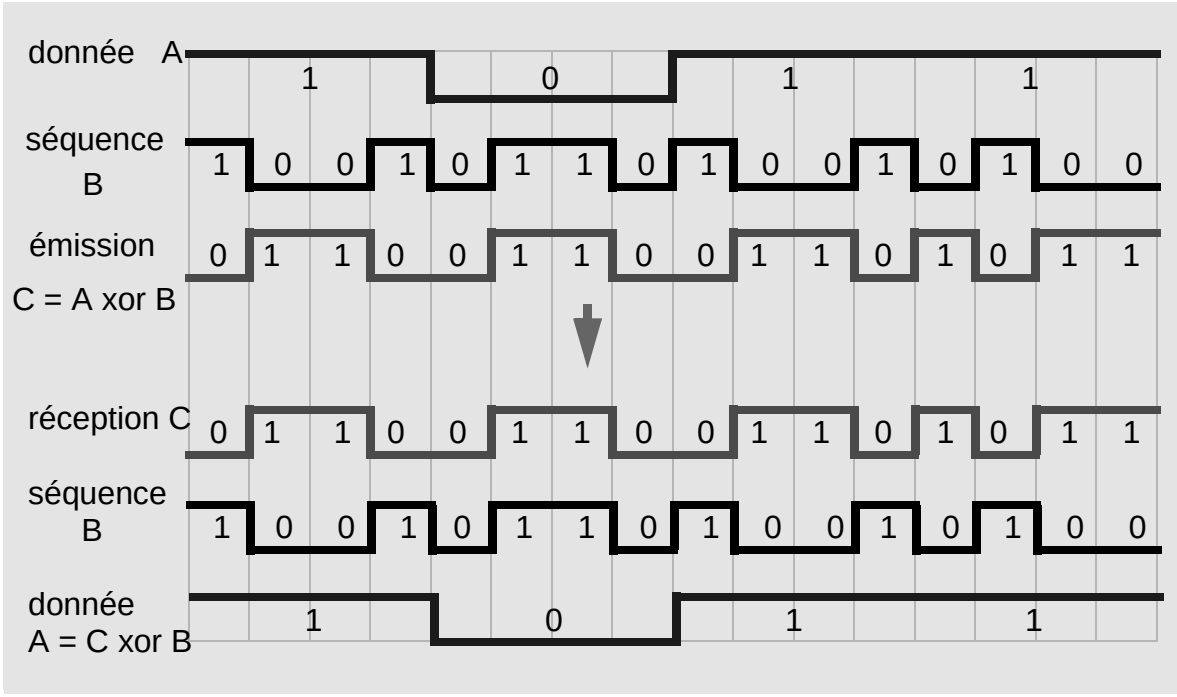
Séquence directe

Dans ce schéma, à chaque bit du signal originel correspondent plusieurs bits de transmission, ces bits portent le nom de *chipping code* (*chips*). Le *chipping code* étale le signal sur une bande de fréquence plus large, en proportion directe au nombre de bits dans le *chipping code*. Un *chipping code* à 10 bits étale le signal sur une bande de fréquences 10 fois plus importante qu'un *chipping code* à 1 bit.

Une des techniques pour la séquence directe est de combiner l'information binaire avec une séquence binaire pseudo-aléatoire moyennant l'opérateur exclusif-OU. La présence d'un '1' inverse la séquence binaire, tandis qu'un '0' provoque la transmission directe de la séquence pseuo-aléatoire.

Dans l'exemple ci-dessous, les séquences pseudo-aléatoires sont composées de quatre bits. Le débit binaire après la modulation est donc quatre fois plus important que le débit initial.

FIGURE 55. Exemple d'une séquence directe pour *spread spectrum*

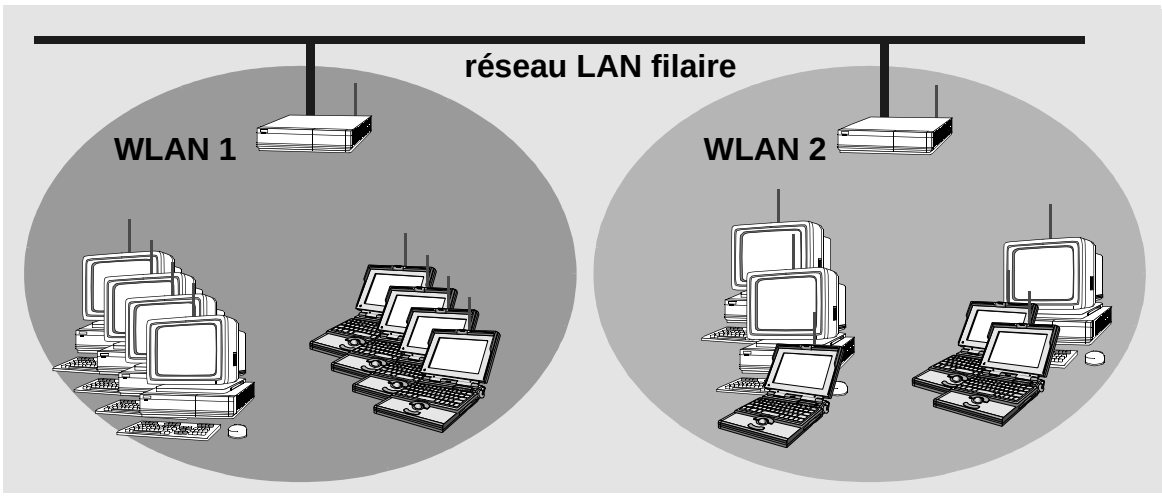


Dans les implémentations typiques, les données et les séquences aleatoires sont combinées après la conversion en signal analogique.

Configurations de WLAN

Les WLANs basés sur la technique du spectre étalé sont organisés sous la forme de cellules. Les cellules adjacentes utilisent différentes fréquences centrales. A l'intérieur d'une cellule, la topologie est du type égal-à-égal ou du type station-*hub*. Le *hub* peut être installé au plafond de la pièce et connecté au *backbone*.

FIGURE 56. Une configuration simple d'un réseau multi LAN



Le *hub* peut également fonctionner comme un répéteur *multiport* dans les réseaux 10/100BASE-T.

Les réseaux *spread-spectrum* fonctionnent sur différentes fréquences. Parmi les fréquences qui peuvent être utilisées sans une licence préalable on trouve les fréquences dans la bande 2,4 GHz (la même que celle utilisée pour les fours micro-ondes). Sur cette bande de fréquences les équipements réseau peuvent communiquer avec un débit binaire maximal de l'ordre de 5 Mbit/s.

Le standard IEEE 802.11

Le groupe de travail IEEE 802 a développé un ensemble de spécifications pour les réseaux WLAN sous la forme du standard IEEE 802.11. Le standard IEEE 802.11 défini un certain nombre de services spécifiques d'un réseau WLAN:

- **l'association** - détermine l'identité et l'adresse d'une station sur WLAN, cette identification établit l'association entre le point d'accès (e.g. station de base) et la station à l'intérieur du BSS (*basic service set*); la station de base peut communiquer cette information aux autres stations de base dans ESS (*extended service set*),
- **la réassociation** - effectue le transfert d'une association vers un autre point d'accès (station de base), permettant à la station mobile de se déplacer entre BSS,
- **la disassociation** - annulation d'une association avant l'arrêt de travail,
- **l'authentification** - ce service permet d'identifier une station (mot de passe),
- **la sécurité** - ce service prévoit l'encryptage des messages.

Couche physique

Le standard 802.11 prévoit l'utilisation du spectre étalé dans la bande 2,4 GHz avec deux versions possibles, celle du saut de fréquence et celle du séquencement direct.

Couche MAC

Le protocole retenu pour la couche MAC est nommé DFWMAC (*distributed foundation wireless* MAC). Ce protocole prévoit un mécanisme de contrôle distribué de type Ethernet avec une option du contrôle centralisé. La sous-couche basse du protocole MAC s'appelle DFC (*distributed coordination function*); la couche supérieure porte le nom de PCF (*point coordination function*).

La couche DCF n'intègre pas le mécanisme de détection de collision ! Le DCF gère un ensemble de retards permettant de construire un schéma de priorité.

Prenons au début un seul retard (délai) dénoté comme IFS (*interframe space*); moyennant IFS, les règles du CSMA sont les suivantes:

- la station qui veut transmettre une trame teste le support,
- si le support est libre, elle attend le temps de IFS; si pendant cet intervalle le support est toujours libre, elle commence à émettre,
- si le support est occupé, la station diffère l'émission et continue à scruter le support,
- dès que le support est libéré, la station attend un temps de IFS; si le support est toujours libre, la station utilise le protocole de *back-off* (attente aléatoire contrôlée), puis elle scrute le medium, s'il est toujours libre elle commence à émettre.

Notons que le mécanisme de *back-off* est le même que celui utilisé dans l'Ethernet. Il permet d'éviter le blocage du réseau par une charge importante.

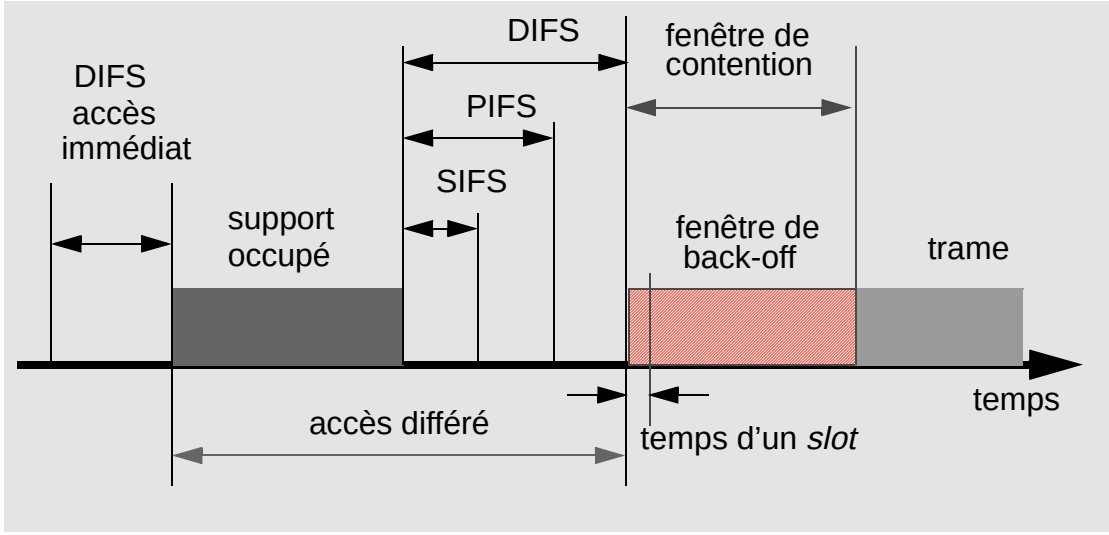
Le simple schéma ci-dessus doit être complété par l'introduction des nouvelles valeurs pour IFS:

- SIFS (*short* IFS) - utilisé pour les réponses immédiates,
- PIFS (*point coordination* IFS) - utilisé par le contrôleur central dans le schéma PCF pour envoyer les requêtes de scrutation,
- DIFS (**distributed** *coordination function*) - le plus long ISF utilisé comme délai minimum pour les trames asynchrones demandant l'accès au support.

La figure ci-dessous montre l'utilisation de ces valeurs. Une station qui utilise SIFS obtient l'accès de la façon prioritaire. Le SIFS peut être utilisé dans les cas suivants:

- **l'acquittement** (ACK) - quand une station reçoit une trame sélective elle répond après avoir attendu un laps de SIFS; ce mécanisme, en l'absence de détection de collisions, permet de confirmer une réception correcte; il permet également d'implémenter facilement le protocole de lien (LLC),
- ***clear-to-send*** - une station peut entamer une communication par l'envoi d'une courte trame RTS (*request to send*); le destinataire de cette trame doit répondre immédiatement par une trame CTS (*clear to send*); toutes les stations qui reçoivent la trame RTS doivent différer leurs émissions,
- ***poll response*** (réponse à la scrutation) - mécanisme utilisé dans la fonction PCF.

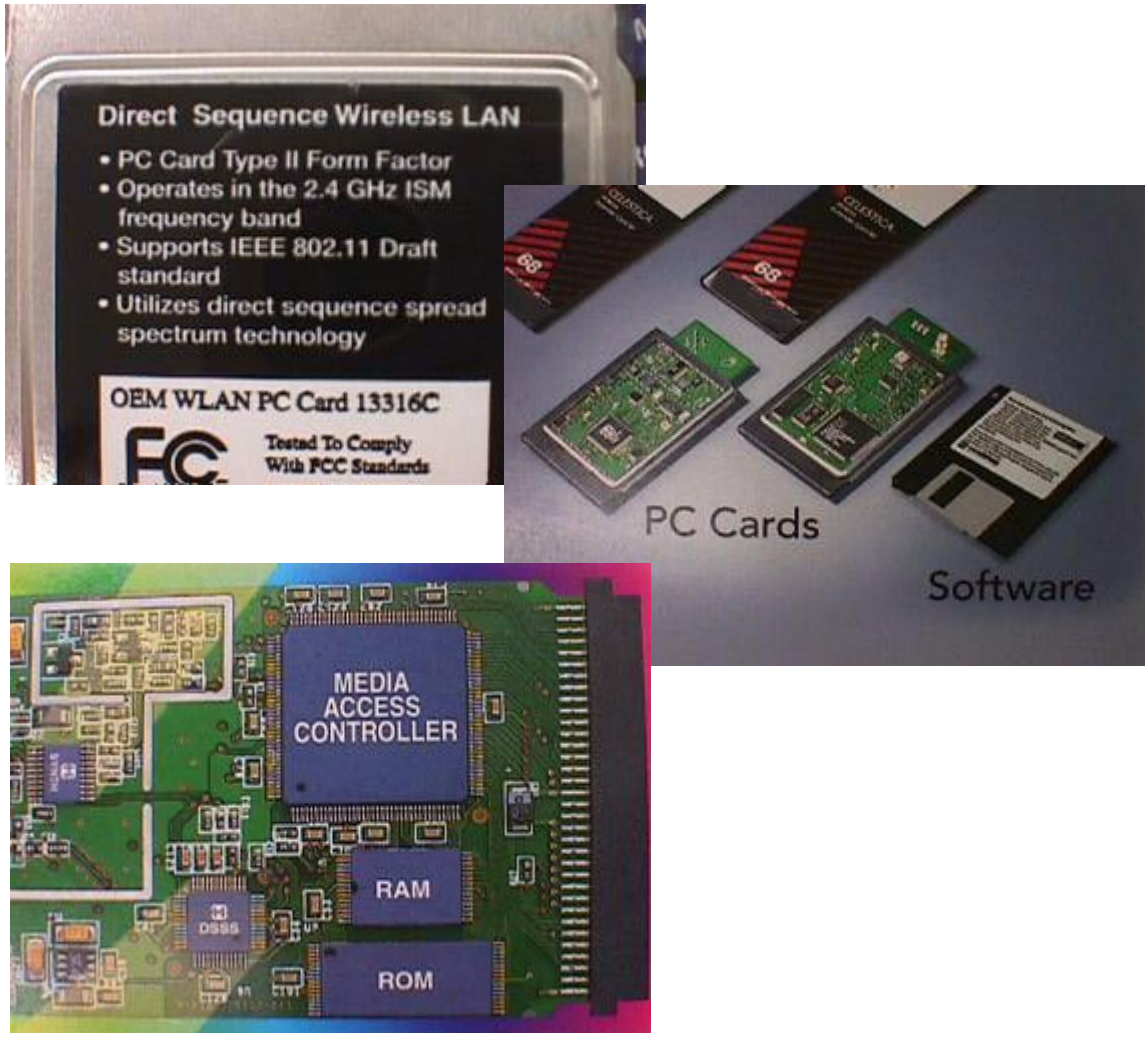
FIGURE 57. Accès de base pour le MAC de IEEE 802.11



La fonction PCF est implémentée au-dessus de la fonction DCF. Le rôle de PCF est de coordonner les fonctions de la base par le biais de la scrutation (*polling*). PCF exploite le délai PIPS pour l'envoi des requêtes de scrutation. Etant donné que le PIFS est plus court que DIFS, la station de base peut accaparer le support et bloquer le trafic asynchrone pendant l'envoi des *polls* et la réception de réponses.

L'utilisation prolongée des PIFS, due par exemple à une panne de la station scrutée, peut provoquer le blocage du réseau. Afin d'éviter ce type de blocage, le standard prévoit un intervalle défini par la **super-trame** (*super-frame*). Après l'écoulement de la période correspondant à la super-trame, la station de base permet le trafic asynchrone basé sur la contention.

FIGURE 58. Carte WLAN 802.11 en format PCMCIA pour ordinateurs portables



Résumé

La couche fonctionnelle MAC joue un rôle essentiel dans la conception et dans l'implémentation des réseaux locaux filaires et hertziens. La principale technologie développée est celle basée sur l'accès aléatoire, la technologie qui mène aux réseaux de type Ethernet. Les réseaux Ethernet permettent d'obtenir un large spectre de solutions allant d'un simple réseau à 10 Mbit/s jusqu'aux configurations complexes intégrant les segments à 10, 100 voir 1000 Mbit/s. Ces réseaux complexes nécessitent l'utilisation des répéteurs, *hubs*, et des commutateurs.

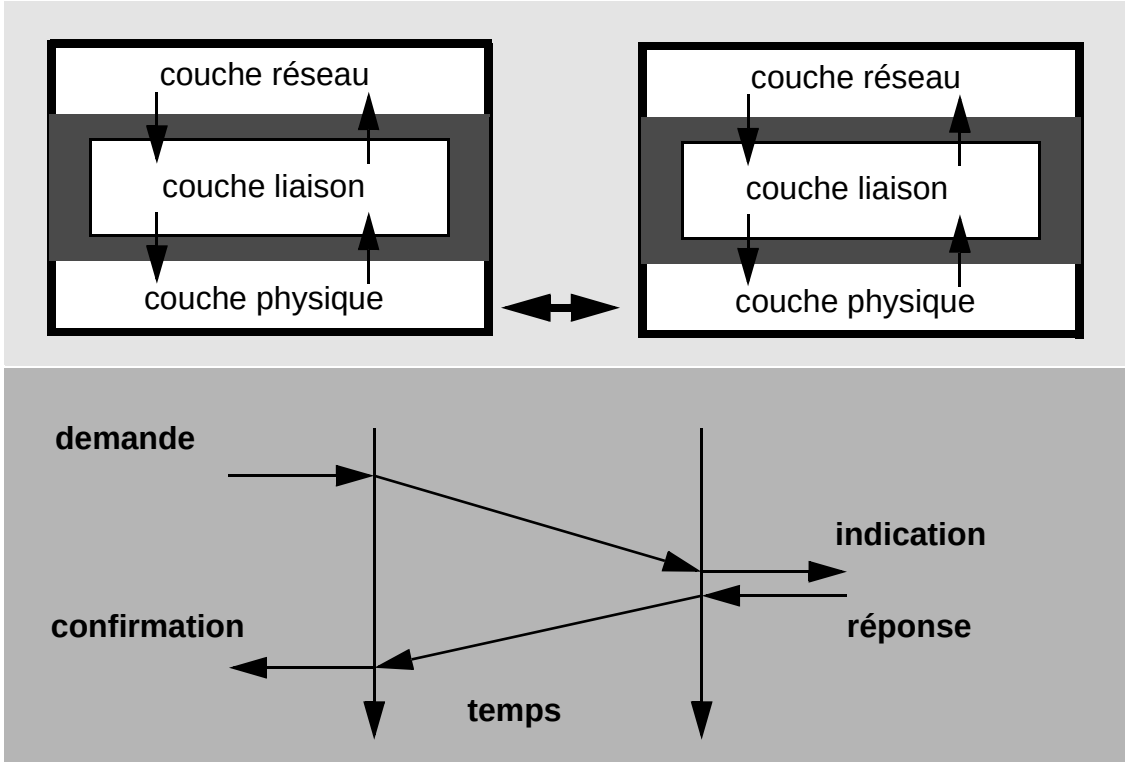
Depuis peu on voit l'apparition des réseaux locaux sans fil (WLAN) basés sur la norme IEEE 802.11. Les réseaux WLAN peuvent jouer un rôle très important car ils permettent plus de mobilité aux utilisateurs et simplifient l'installation du réseau.

CHAPITRE 4

Couche liaison

Les protocoles de la couche de **liaison de données** permettent à deux stations ou noeuds de communication de communiquer de façon fiable sur un lien physique. Le rôle des protocoles de lien est d'assurer la fiabilité de communication sur les liens dont le débit binaire est limité, le délai de propagation non nul, et des erreurs de transmission peuvant survenir... Ces facteurs, auxquels s'ajoute le temps de traitement, et qui influent fortement sur le transfert de données, sont pris en compte dans l'élaboration des protocoles.

FIGURE 59. Couche liaison et les primitives de service



La notion de trame

La couche physique assure le transport de trains de bits sur le support déployé entre deux stations ou noeuds de communication. Ces trains de bits peuvent comporter des erreurs. La couche liaison doit détecter ces erreurs et les corriger si nécessaire. A cette fin, la couche liaison découpe le train de bits en trames et calcule une somme de contrôle d'erreurs pour chaque trame.

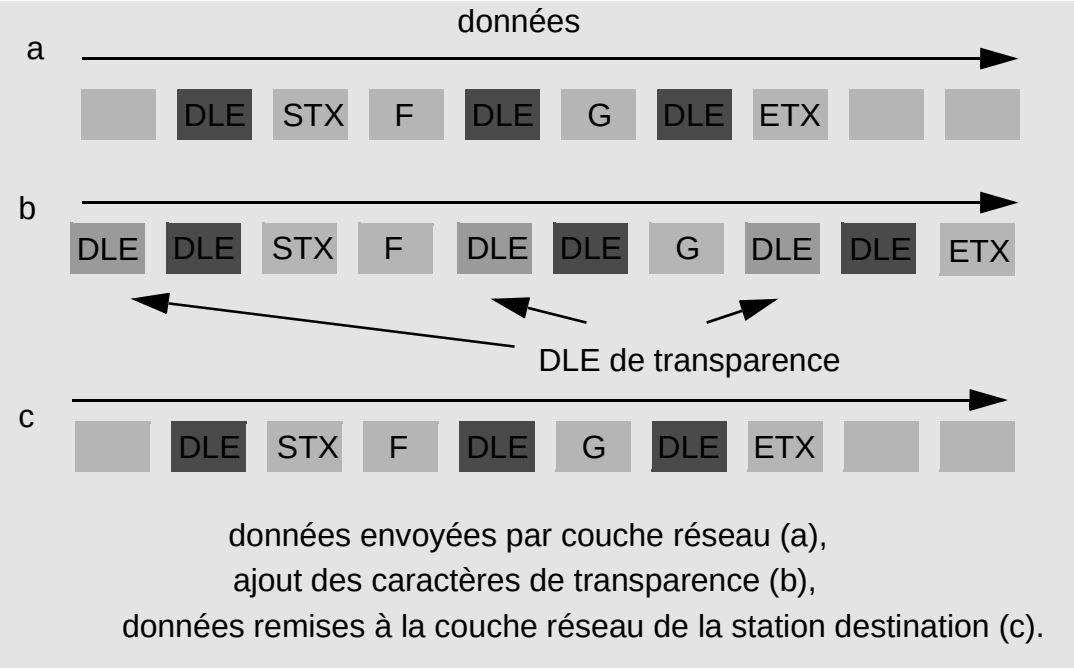
Le découpage en trames peut être effectué selon trois techniques qui impliquant:

- l'utilisation des caractères de début et de fin de trame avec caractères de transparence,
- l'utilisation des fanions binaires de début et de fin de trame avec des bits de transparence,
- la violation du codage physique.

Trame de caractères

La première technique exploite les caractères de contrôle pour délimiter une trame. Les séquences de caractères DLE STX (*Data Link Escape*, *Start of TeXt*) et DLE ETX (*Data Link Escape*, *End of TeXt*) sont placées respectivement en début et en fin de trame.

FIGURE 60. Caractères de contrôle et de transparence

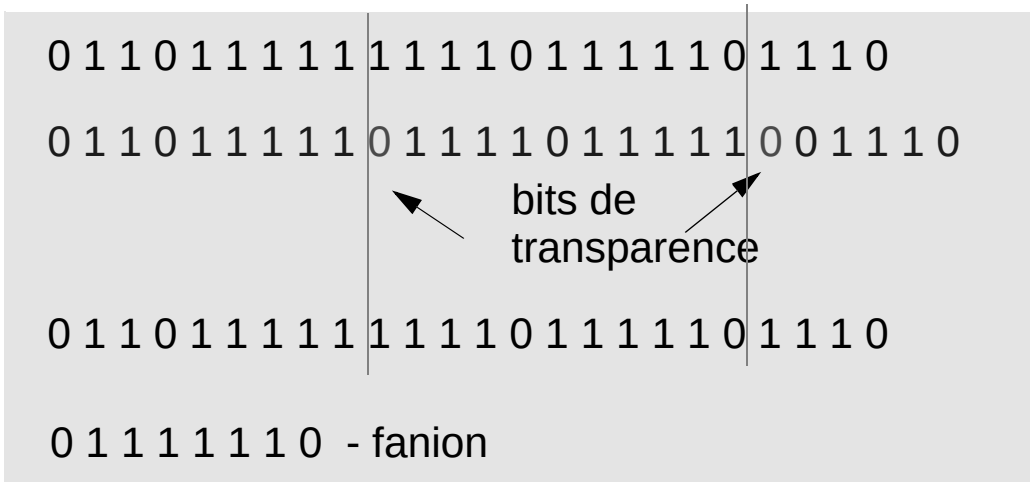


Lorsque dans les données à transmettre apparaissent les séquences DLE STX et DLE STX, le transmetteur ajoute un autre caractère **DLE** devant tout le caractère **DLE**. La couche liaison du récepteur enlève les caractères **DLE** ajoutés par l'émetteur.

Trame "binaire"

Les trames "binaires" permettent de contenir un nombre arbitraire de bits et autorisent l'utilisation de codages dans lesquels chaque caractère est représenté par un nombre arbitraire de bits. Chaque trame commence par une séquence particulière de bits **`01111110`** appelée **fanion** (en héxadécimal **7E**). Lorsque la couche liaison détecte cinq '1' consécutifs dans les données à transmettre, elle ajoute à leur suite un bit '0' avant d'envoyer le train de bits sur la ligne. Quand le récepteur reçoit cinq bits '1' consécutifs suivis d'un '0', il enlève automatiquement ce dernier.

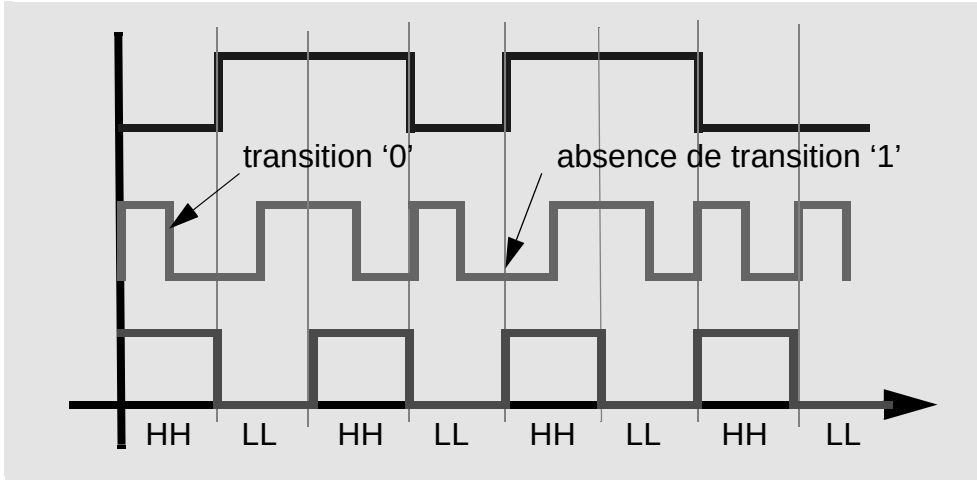
FIGURE 61. Technique de *bit stuffing* - bits de transparence



Violation du code physique

Lorsque le code physique contient de redondances, il est possible d'introduire des états anormaux pour marquer le début ou la fin d'une trame. Par exemple le code Manchester représente le bit '0' par une impulsion positive (un demi-bit) suivie d'une impulsion négative. Les combinaisons positive-positive (HH) et négative-négative (LL) ne sont pas utilisées pour la représentation de données.

FIGURE 62. Violation du code physique pour marquage des trames

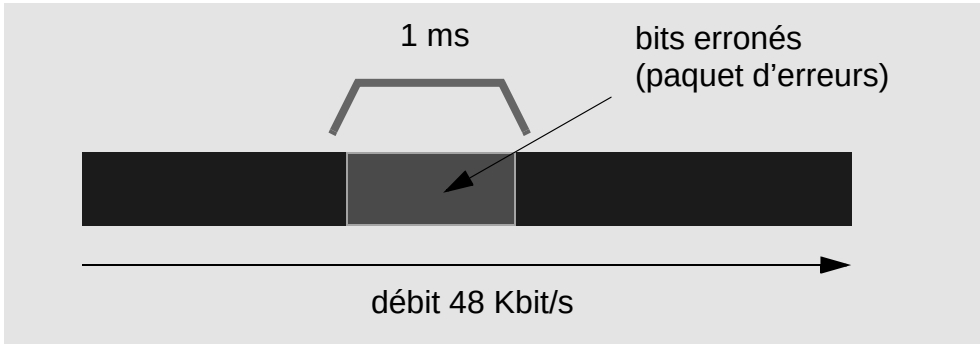


Détection et correction des erreurs

Les erreurs de transmission sont dues à différents phénomènes physiques. Le bruit thermique est toujours présent. Il est provoqué par l'agitation des électrons dans les câbles métalliques. Rappelons-nous comment le rapport signal à bruit intervient dans le théorème de Shannon.

Le bruit impulsif est un autre phénomène non négligeable. Il provoque des impulsions parasitaires avec des périodes de 1 ms. Par exemple, sur une ligne à 48 Kbit/s, elles causent la perte de 48 bits.

FIGURE 63. Bruit impulsif et paquet d'erreurs



Une autre source physique d'erreurs est due à la différence de la vitesse de propagation de signaux à fréquences différentes. D'autres erreurs, telles que la perte d'une trame, peuvent être impliqués par les limites structurelles des architectures de réseaux. En général toutes ces intrusions ont tendance à créer des paquets d'erreurs plutôt que des erreurs simples.

Deux stratégies du contrôle d'erreurs

Les concepteurs de réseaux ont développé deux stratégies dans le domaine du contrôle des erreurs de transmission: la correction et la détection/retransmission.

- la **correction** consiste à inclure dans les blocs de données suffisamment de redondances pour que le récepteur puisse restituer les données originales à partir des données reçues,
- la **détection/retransmission** consiste à ajouter juste assez de redondances dans les données à transmettre afin que le récepteur puisse détecter les erreurs sans pouvoir les corriger.

FIGURE 64. Correction d'erreurs - applications temps-réel

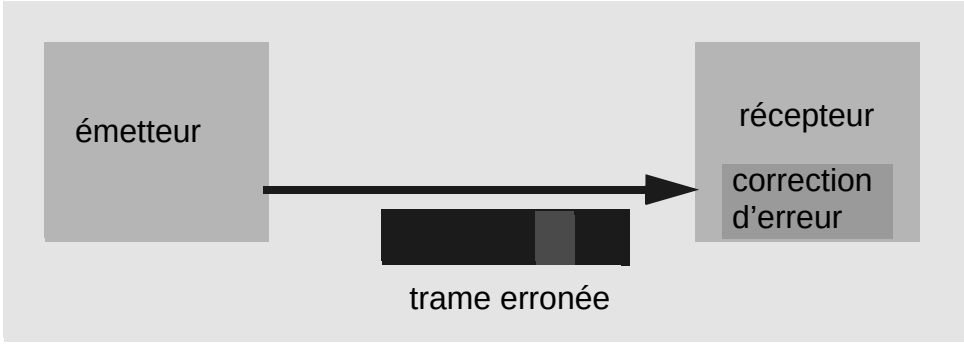
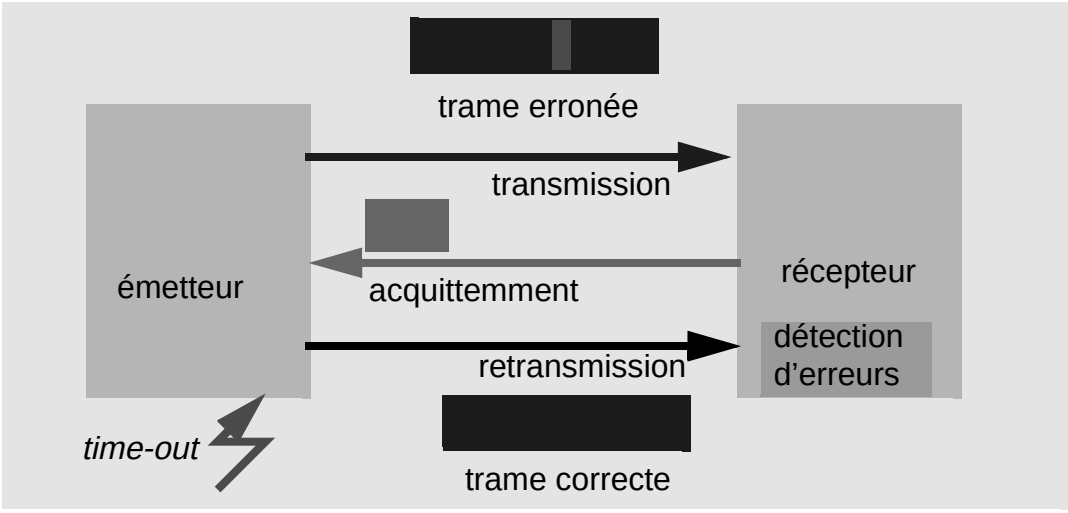


FIGURE 65. **Détection et retransmission - applications temps-différé**



Les codes correcteurs

Afin de pouvoir corriger les erreurs, il est nécessaire de connaître la définition exacte d'une erreur. Supposons qu'une trame est formée de **m** bits de données et de **r** bits de contrôle. Si l'on note **n** la longueur de la trame (**n=m+r**), l'ensemble de **n** bits sera appelé **mot de code**. Etant donné deux mots de code, il est possible de déterminer de combien de bits ils diffèrent.

Exemple:

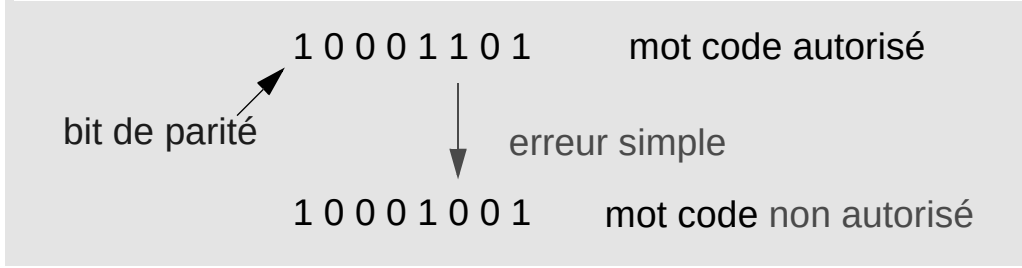
1 0 *0* 0 1 *1* 0 *1*

1 0 *1* 0 1 *0* 1 *0*

Pour cet exemple le nombre de différences appelé **distance de Hamming** est égal 4. Dans un mot de code il est possible d'utiliser 2^m combinaisons des bits de données mais seule une partie des 2^n combinaisons est autorisée ($2^m<2^n$). Cela est dû à la façon dont sont calculés les bits de contrôle.

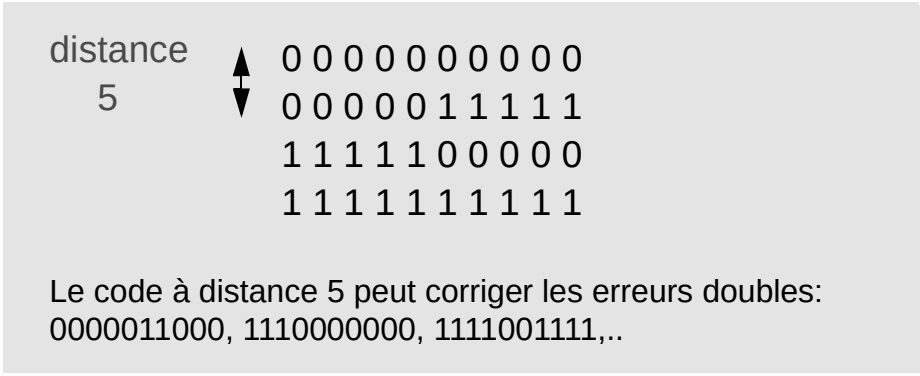
Il est possible d'obtenir la liste de tous les mots de code afin de retrouver la distance minimale entre deux mots du **code autorisé**. Pour détecter **d** erreurs, il faut que le code ait une distance de Hamming de **d+1**. En effet il est impossible que **d** erreurs changent un mot de code en un autre mot de code autorisé. Le code de parité constitue un exemple simple de code détecteur. Ce code a une distance de 2, donc toute erreur simple produit un **mot non autorisé**. Pour corriger **d** erreurs, il faut que le code ait une distance de Hamming de **2*d+1**.

FIGURE 66. **Code autorisé et non-autorisé**



Dans ce cas, la distance entre chaque mot de code est telle que, même si **d** erreurs simples se produisent, le mot de code original reste plus proche du mot transmis: on peut donc le retrouver.

FIGURE 67. Distance Hammingmaximale pour la correction



Supposons que nous voulions construire un code qui permette de corriger les erreurs simples, avec **m** bits de données et **r** bits de contrôle. Pour chacun de 2^m des mots codes possibles, il existe **n** mots de code non-autorisés situés à une unité du mot de code autorisé. Ils sont obtenus à partir du mot de code autorisé en inversant un de ses **n** bits. A chacune des 2^m combinaisons de bits de données possibles, on associe **n+1** mots de **n** bits.

Comme le nombre total de mots de **n** bits est 2^n, la relation suivante doit être vérifiée:

$$(n+1)*2^m < 2^n$$

En utilisant l'égalité **n=m+r**, l'inégalité devient:

$$(m+r+1)<2^r$$

Connaissant **m**, cette inégalité permet de déterminer le nombre minimal de bits de contrôle nécessaire pour que toutes les erreurs simples soient corrigées.

Question:

Combien de bits de contrôle (**r**) faut-il ajouter à un mot de données de 10 bits pour pouvoir corriger une erreur simple?

Réponse: $(10 + r + 1) < 2^r$ => r= 4

Les codes détecteurs

Les codes correcteurs sont utilisés pour les transmissions de données dans les cas où il est impossible de demander une retransmission. Le plus souvent on utilise une technique de détection avec retransmission pour des raisons d'efficacité.

Exemple: Prenons comme exemple un canal avec des erreurs simples et un taux d'erreurs de 10^{-6}. Considérons des blocs de données de 1000 bits. Si l'on désire corriger les erreurs simples, il faut ajouter 10 bits de contrôle par bloc (1010 **bit/bloc**). Pour un mégabit de données, on aura **10 000** bits de contrôle. Pour détecter simplement un mauvais bloc, il suffit d'avoir un bit de parité (1001 bit/bloc). Avec la méthode de détection et retransmission le nombre de bits supplémentaires est seulement **2001** (1000*1 + 1001), alors qu'il en faut **10 000** avec un code de correction.

Bien que les bits de parité puissent être exploités pour la détection des erreurs multiples, en pratique on utilise les codes polynômiaux, appelés aussi **codes CRC** (*Cyclic Redundancy Code*). Dans les codes CRC, on considère que les bits d'une chaîne de caractères sont les coefficients d'un polynôme. Ces coefficients ne prennent que deux valeurs: 0 ou 1. Un bloc de **m** bits est vu comme une série de coefficients d'un polynôme comprenant **m** termes allant de $\mathbf{x^{m-1}}$ à $\mathbf{x^0}$. Un tel polynôme est dit de degré **m-1**.

Par exemple, la chaîne 10101 comprend 5 bits: elle est représentée par un polynôme de 5 termes dont les coefficients sont 1,0,1,0 et 1: $\mathbf{x^4+x^2+x^0}$.

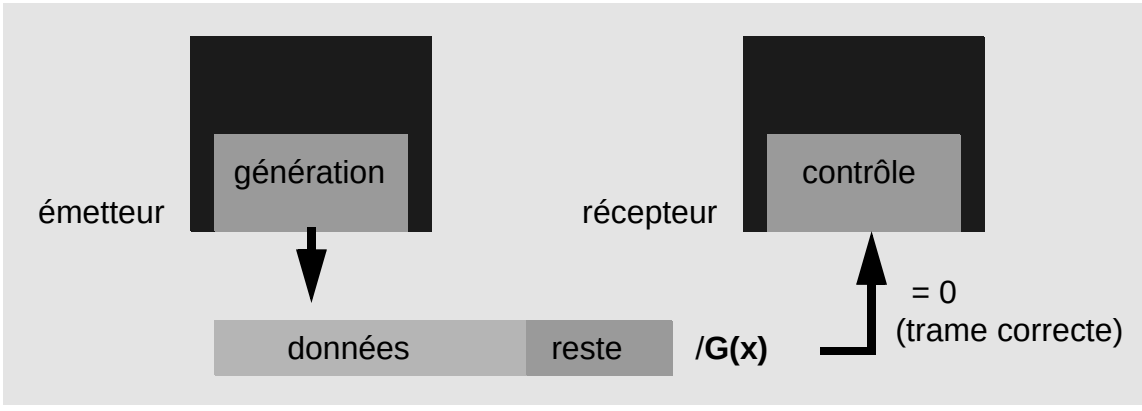
Etant donné que l'arithmétique polynômiale est faite **modulo 2,** les opérations d'addition/soustraction donnent le même résultat. Notons également que l'arithmétique modulo 2 ne prend pas en compte les retenues.

Exemple:

```
+1001          +1001
+1010          -1010
---------      ---------
=0011          =0011
```

Pour utiliser un code polynômial, l'émetteur et le récepteur doivent se mettre d'accord sur le choix d'un polynôme générateur **G(x)**. Le générateur doit avoir son bit de poids fort et son bit de poids faible égaux à 1. Pour calculer la somme de contrôle d'un bloc de **m** bits, il faut qu'il soit plus long que le polynôme générateur. L'idée de base consiste à coller, à la fin du bloc, des bits de contrôle de façon que la trame (**bloc+bits de contrôle**) soit **divisible par G(x)**. Quand le récepteur reçoit la trame, il la divise par **G(x)**. Si le reste est non nul, une erreur de transmission a eu lieu.

FIGURE 68. Emission et réception des trames avec le code CRC

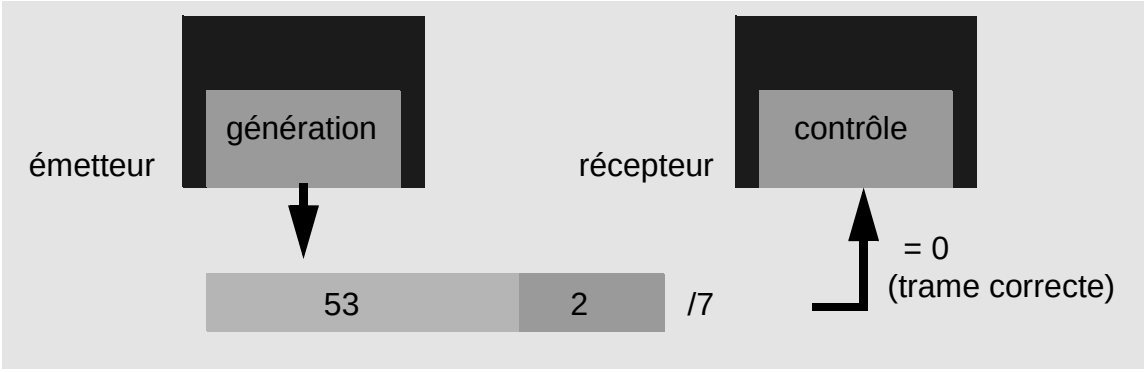


Exemple (décimal):

- donnée - 53
- générateur - 7

```
530: 7          75  - le quotient
49
 40
 35
  5  complément de 7 est 2 ; donc la valeur à générer est  532
```

FIGURE 69. Génération et test d'une trame avec le code CRC (version décimale)



Exemple (binaire): trame: `1101011011`, générateur: `10011`; après ajout de 4 bits à `0`: `11010110110000`

```
11010110110000
10011
 10011
 10011
   00001
   00000
   00010
   00000
    00101
    00000
     01011
     00000
      10110
      10011
       01010
       00000
        10100
        10011
         01110
         00000
          1110      - le reste
```

Trame émise: `11010110111`**`1110`**

Les trois polynômes générateurs qui ont été normalisés sont:

- CRC-16 = $x^{16}+x^{15}+x^2+1$
- CRC-CCITT= $x^{16}+x^{12}+x^5+1$
- CRC (802.3/802.5) = $x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^8+x^7+x^5+x^4+x^2+x+1$

Tous ces polynômes contiennent **x+1** comme facteur premier. Les polynômes CRC-16 et CRC-CCITT sont utilisés pour des caractères codés sur 8 bits. Ils permettent de détecter toutes les erreurs simples et doubles, toutes les erreurs comportant un nombre impair de bits et tous les paquets d'erreurs de longueur inférieure ou égale à 16. Ils détectent avec une probabilité de 99,997% les paquets d'erreurs de 17 bits et avec une probabilité de 99,998% les paquets d'erreurs de longueur supérieure ou égale à 18 bits.

Important:

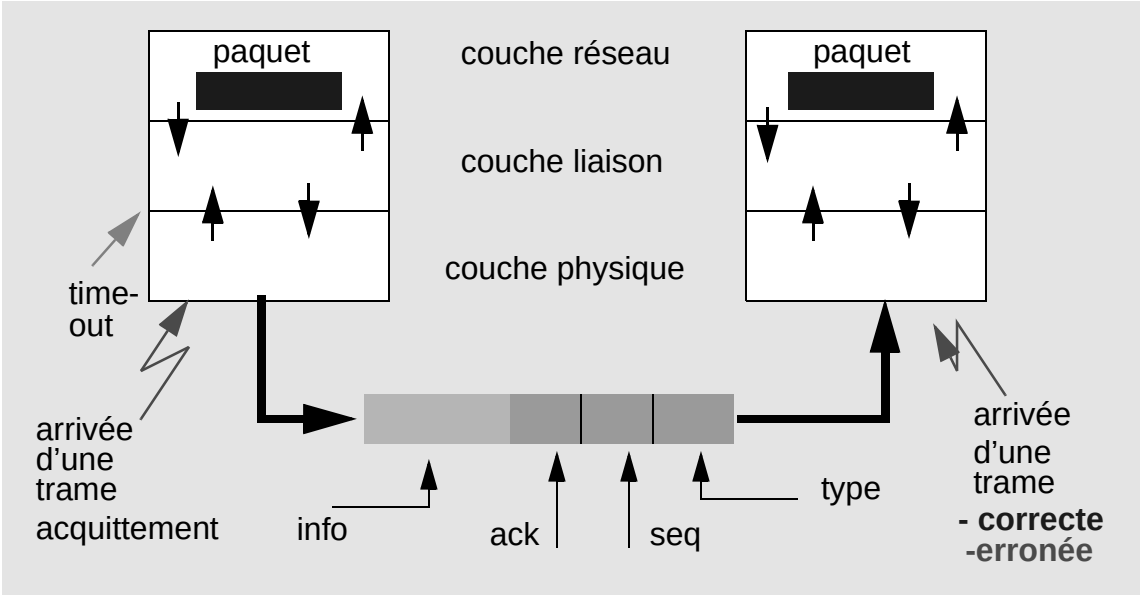
En pratique, un simple registre à décalage suffit pour obtenir la somme de contrôle. On utilise presque toujours des circuits électroniques pour les codages et décodages polynômiaux.

Protocoles élémentaires de ligne

Avant de commencer l'étude des protocoles simples de ligne, nous supposons que les couches physique, liaison, et réseau sont des processus indépendants qui communiquent en échangeant des messages. Sur le schéma ci-dessous nous avons illustré les éléments essentiels nécessaires pour la construction des protocoles de ligne. Ils sont:

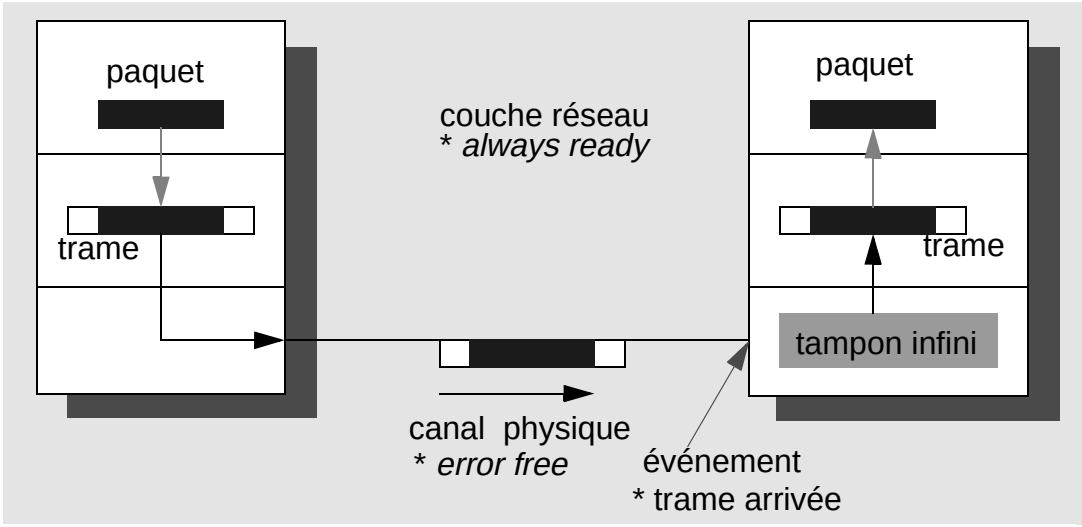
- les opération de transfert entre couches (flèches),
- les événements à gérer par protocoles,
- le format d'une trame envoyée sur ligne.

FIGURE 70. Eléments essentiels des protocoles de ligne



Protocole 1: protocole "utopique"

FIGURE 71. Protocole 1: protocole élémentaire avec un canal idéal




```
-- sender
loop
From_Net_L(packet);
frame.info<=packet;
To_Phys_L(frame);
end loop;
-- until doomsday
```

```
-- receiver
loop
wait on frame_arrived
From_Phys_L(frame);
packet<=frame.info;
To_Net_L(packet);
end loop;
-- until doomsday
```

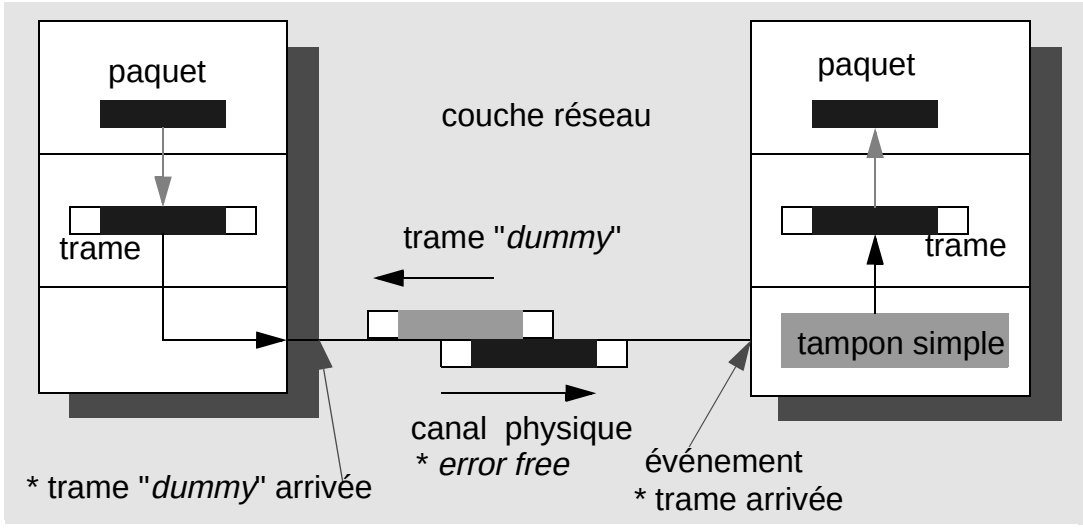
Dans le protocole 1, on ne transmet les données que dans un sens. Les couches réseau côté émetteur et récepteur sont toujours prêtes respectivement à émettre et à recevoir des paquets. On ignore le temps de calcul. Le récepteur dispose d'autant de mémoires tampon que nécessaire. Le canal physique est parfait: aucune trame n'est erronée ni perdue.

Protocole 2: protocole de type "envoyer et attendre"

Dans le protocole 2 nous allons supprimer l'hypothèse que la couche réseau est apte à traiter les données arrivant de façon instantanée. Nous supposons toujours que le canal est parfait et que les données ne circulent que dans un sens. Le problème que nous allons à résoudre est de trouver une manière d'empêcher l'émetteur d'envoyer des données plus rapidement que le récepteur ne peut les traiter. De plus, nous supposons que le récepteur n'est pas capable de stocker et de mettre en file d'attente les trames reçues.

La solution la plus générale pour résoudre ce problème consiste à obliger le récepteur à informer l'émetteur de son état. Après avoir transmis un paquet à la couche réseau, le récepteur envoie à l'émetteur une petite trame ne contenant aucune information mais donnant la permission d'émettre la trame suivante. Les protocoles dans lesquels l'émetteur envoie une trame et attend ensuite un acquittement avant d'envoyer la suivante sont appelés protocoles de type "envoyer et attendre".

FIGURE 72. Protocole 2: protocole *simplex* de type “envoyer et attendre”




```
-- sender
loop
From_Net_L(packet);
frame.info<=packet;
To_Phys_L(frame);
wait on dummy_frame;
end loop;
-- until doomsday
```

```
-- receiver
loop
wait on frame
From_Phys_L(frame);
packet<=frame.info;
To_Phys_L(dummy_frame);
To_Net_L(packet);
end loop;
-- until doomsday
```

Protocole 3: protocole simple pour canal bruité

Dans un canal physique réaliste les trames peuvent être erronées ou perdues. Nous supposons que l’on peut détecter les trames endommagées au cours de la transmission lors du calcul de la somme de contrôle. Dans le protocole étudié les données ne sont transmises que dans un seul sens. Ce protocole prévoit la perte éventuelle des trames, grâce à l’utilisation du temporisateur dont la durée doit être suffisamment longue pour permettre l’arrivée des acquittements. Si le temporisateur expire avant l’arrivée de l’acquittement, l’émetteur envoie une trame dupliquée. Dans chaque trame l’émetteur insère un numéro alternatif de séquence (`0,1,0,1,.`). Dans une trame dupliquée il va insérer le même numéro de séquence que celui inséré dans la trame originale.

FIGURE 73. Insertion du numéro de séquence

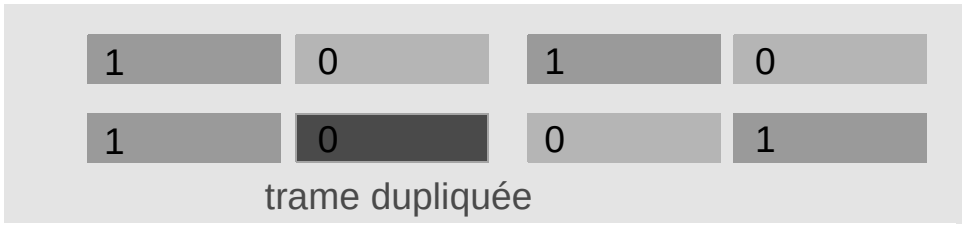


FIGURE 74. **Protocole 3: protocole simple pour canal bruité: *positive acknowledge/retransmission***

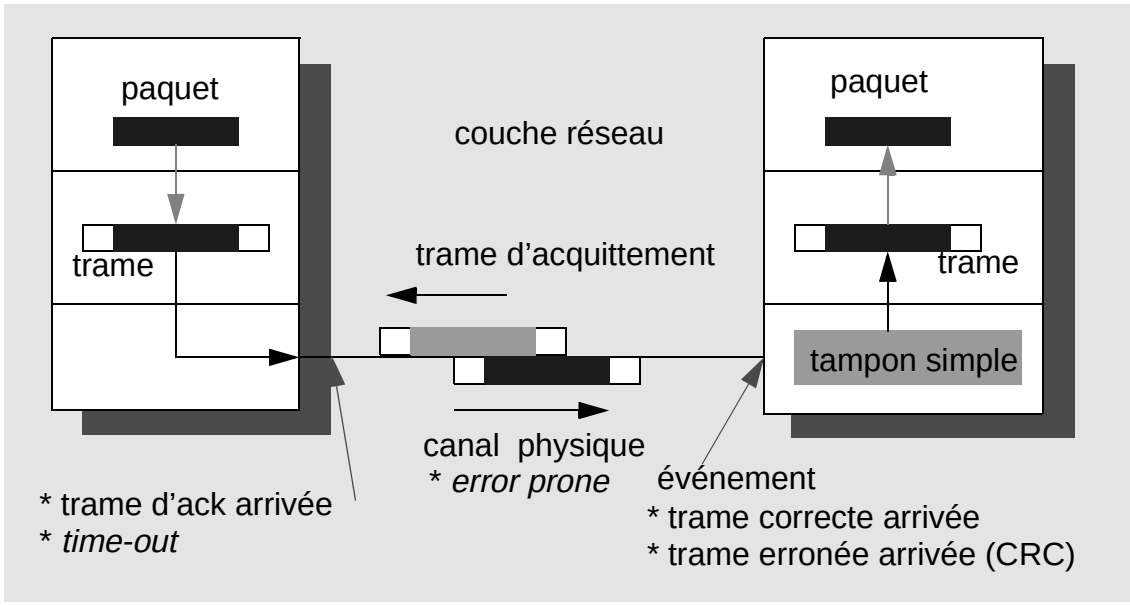


FIGURE 75. **Protocole 3: protocole simple pour canal bruité - algorithme**

```
-- sender
SeqNum<=0;
From_Net_L(packet);
loop
frame.info<=packet;
frame.sn<= SeqNum;
To_Phys_L(frame);
Start_T(frame.sn);
wait on any_event;
if any_event=ack_frame
  then
  From_Net_L(packet);
  SeqNum<=SeqNum+1;
  endif;
end loop;
-- until doomsday
```

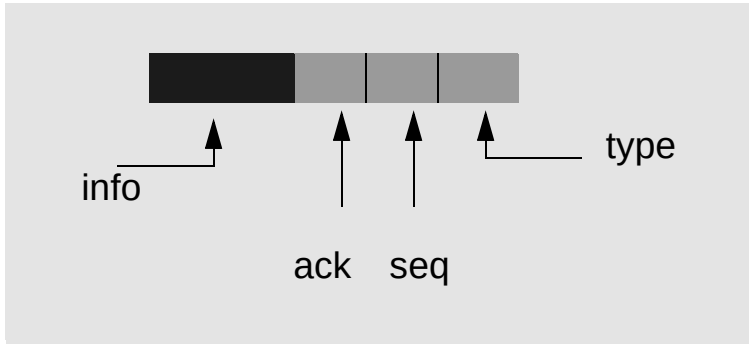
```
-- receiver
SeqExp<=0;
loop
wait on any_event;
if any_event=correct_frame
  then
  From_Phys_L(frame);
  if SeqExp=frame.sn
    then
    To_Net_L(frame.info);
    SeqExp<=SeqExp+1;
    endif;
  To_Phys_L(ack_frame);
  endif;
end loop;
-- until doomsday
```

Protocole 4 : protocole *duplex*

Dans les trois protocoles élémentaires les données ne circulaient que dans un sens. Or, dans de nombreuses situations il est nécessaire de transmettre des données dans les deux sens. On peut ainsi obtenir une liaison en

duplex. Dans ce modèle, lorsqu'une trame de données arrive, le récepteur n'envoie plus de trame d'acquittement séparée, mais attend que la couche réseau lui transmette le paquet suivant. L'acquittement est joint à la trame de données (en utilisant le champ *ack* de la trame).

FIGURE 76. Trame composée donnée-acquittement: *piggybacking*



Cette technique retarde légèrement l'envoi d'un acquittement afin d'attendre la trame de données suivante; elle est connue sous le nom de *piggybacking*.

FIGURE 77. Protocole 4: protocole duplex - architecture

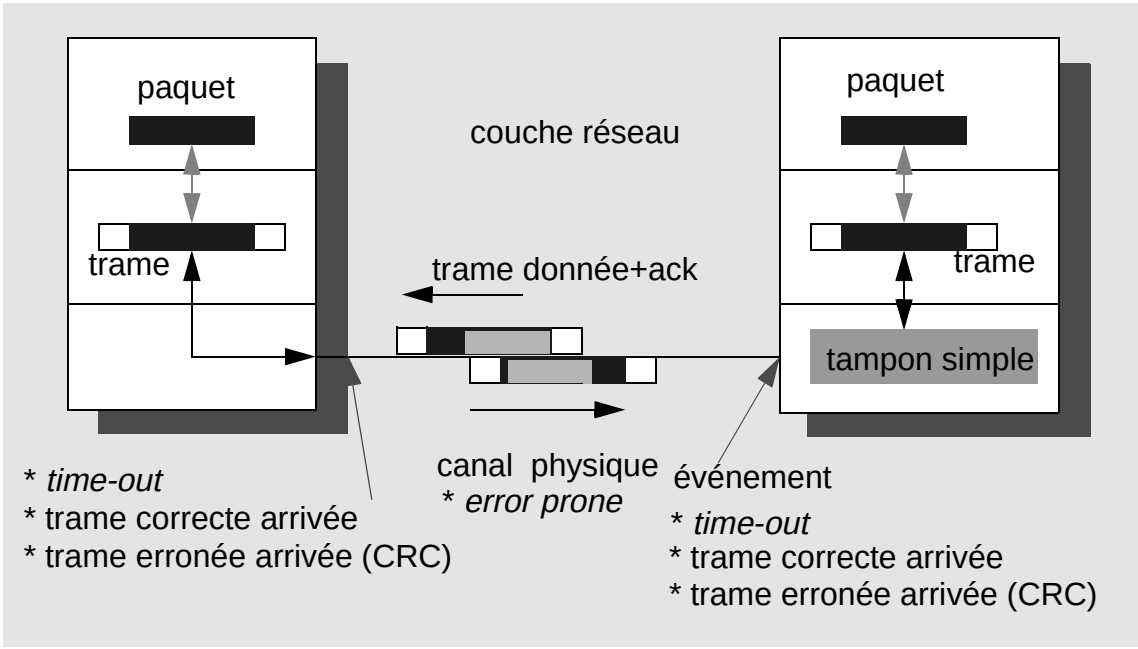


FIGURE 78. **Protocole 4: protocole duplex - algorithme**

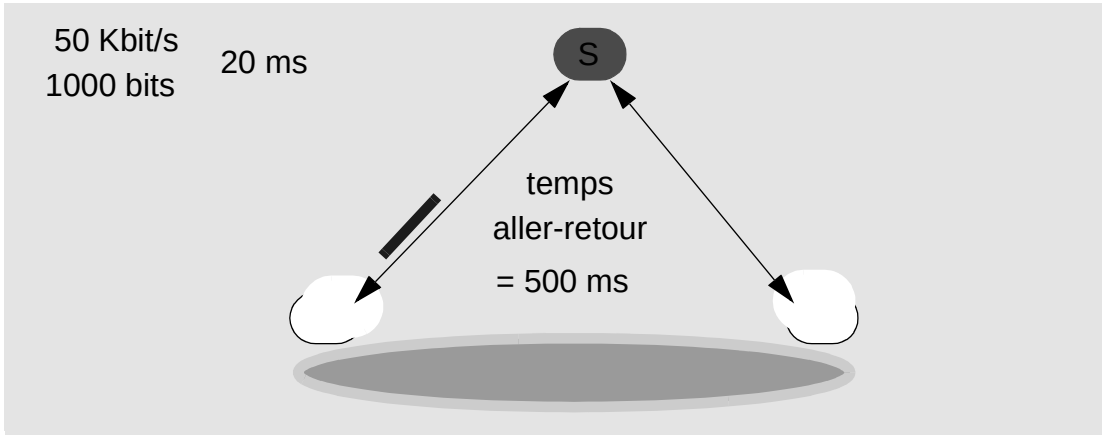
```
-- sender/receiver
SeqNum<=0;
SeqExp<=0;
From_Net_L(packet);
frame.info<=packet;
frame.sn<= SeqNum;
frame.an<= SeqExp+1;
To_Phys_L(frame);
Start_T(frame.sn);
loop
wait on any_event;
if any_event=correct_frame
  then
  From_Phys_L(frame);
  if SeqNum=frame.sn
    then
    To_Net_L(frame.info);
    SeqExp<=SeqExp+1;
    endif;
  if any_event=ack.frame
  then
  From_Net_L(packet);
  SeqNum<=SeqNum+1;
  endif;
frame.info<=packet;
frame.sn<= SeqNum;
frame.an<= SeqExp+1;
To_Phys_L(frame);
Start_T(frame.sn);
end loop;
-- until doomsday
```

Protocoles à fenêtre d'anticipation de largeur n

Avant d'étudier les protocoles à fenêtres d'anticipation, considérons l'exemple d'un canal satellite à 50 Kbit/s avec un délai de propagation aller-retour de 500 ms. Essayons d'imaginer ce que pourrait donner l'emploi du protocole 4 avec des trames de 1000 bits:

- au temps t=0, l'émetteur commence à envoyer la première trame,
- à t=20 ms, la trame est complètement émise; la trame parvient au récepteur à t = 270 ms (250+20) et l'acquittement est reçu au mieux par l'émetteur à t = 520 ms.

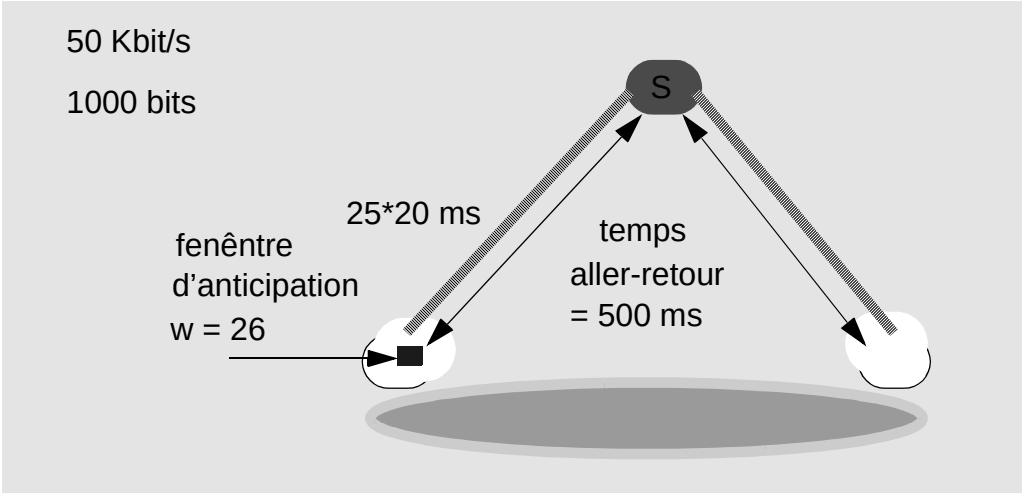
FIGURE 79. **Temps aller-retour (fenêtre d'une trame)**



Cela signifie que l'émetteur reste bloqué pendant 500 ms /520 ms, soit 96% du temps; la bande passante est utilisée à 4%. Pour résoudre ce problème, nous allons autoriser l'émetteur à envoyer **w** trames sans attendre un acquittement. Avec une valeur **w** appropriée, l'émetteur pourra émettre de façon continue pendant un temps égal aller-retour (500 ms). Dans le cas de l'exemple ci-dessus, **w** devrait être égal au moins à 26.

Notez qu'à chaque instant, il existe 25 ou 26 trames non acquittées.

FIGURE 80. Temps aller-retour (fenêtre de 26 trames)



Cette technique est connue sous le nom de *pipelining*. Si la capacité du canal est de **B** bit/s, la taille de trames **L** bits et le temps aller-retour de R secondes, il faut **L/B** secondes pour transmettre une trame. Dans le protocole "envoyer et attendre", la ligne est occupée pendant **L/B** secondes et non utilisée pendant **R** secondes, ce qui donne le taux d'utilisation de **L/(L+B*R)**.

Problème:

Que se passe t-il si une trame, située au milieu de la série de trames envoyées, est erronée ou perdue? Quand la trame erronée arrive, le récepteur n'en tient pas compte. Mais que doit-il faire des autres trames correctes qui arrivent ensuite?

Deux techniques sont possibles pour résoudre ce problème:

- le rejet de toutes les trames qui suivent la trame erronée, cette technique correspond à un récepteur ayant un tampon - fenêtre de taille 1,
- la réception et le stockage de toutes les trames correctes arrivées après la trame erronée dans un tampon de taille **w**.

FIGURE 81. Conséquence d'une erreur de transmission avec une taille de fenêtre égale à 1.

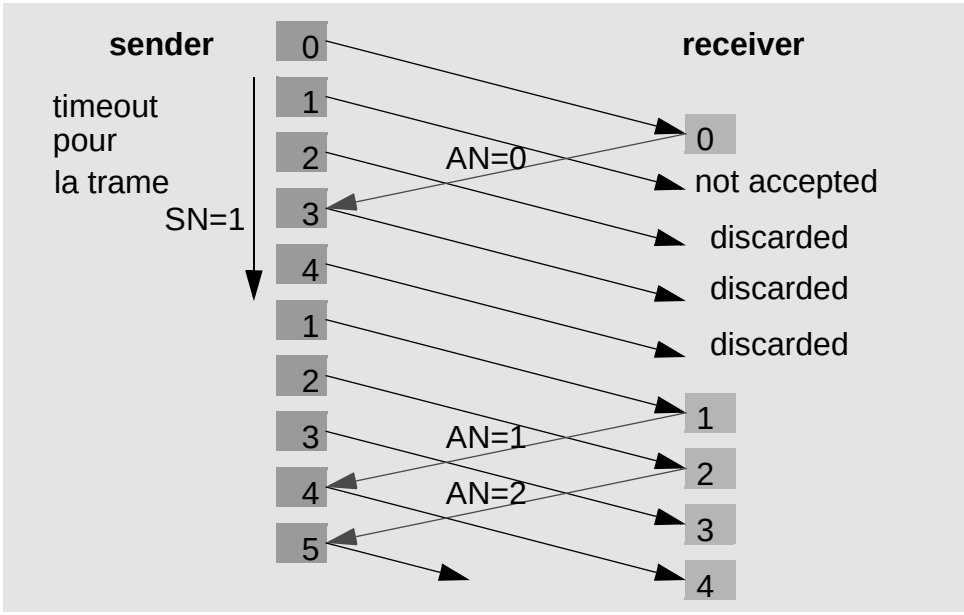
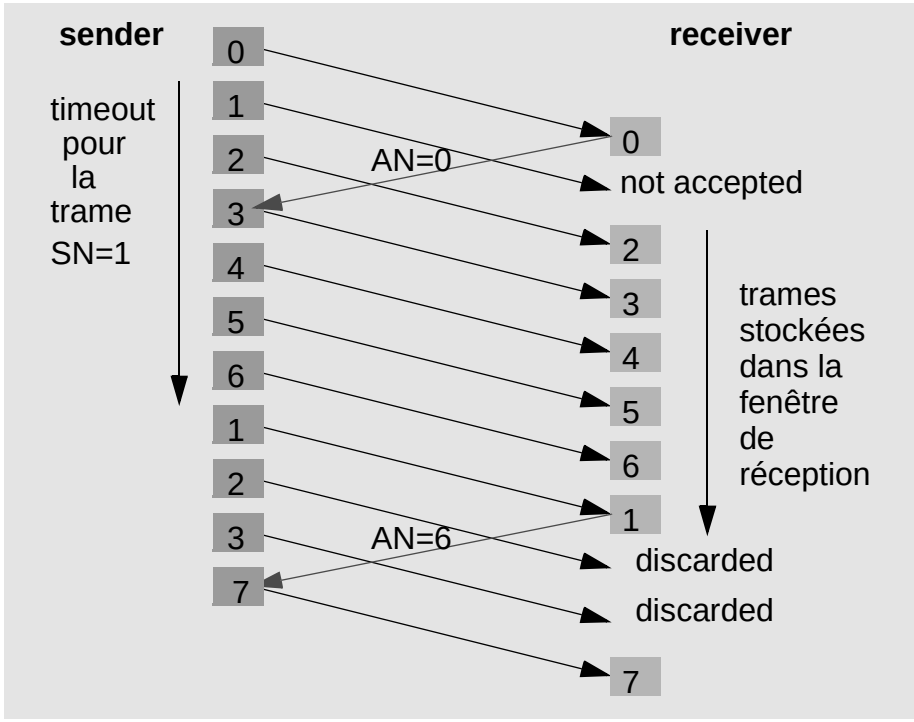


FIGURE 82. Conséquence d'une erreur de transmission avec une taille de fenêtre supérieure à 1.



Performances

Le modèle sur la figure ci-dessous permet d'introduire quelques paramètres essentiels pour l'analyse des performances. Dans cette analyse nous évaluons le taux d'utilisation de la capacité de transmission.

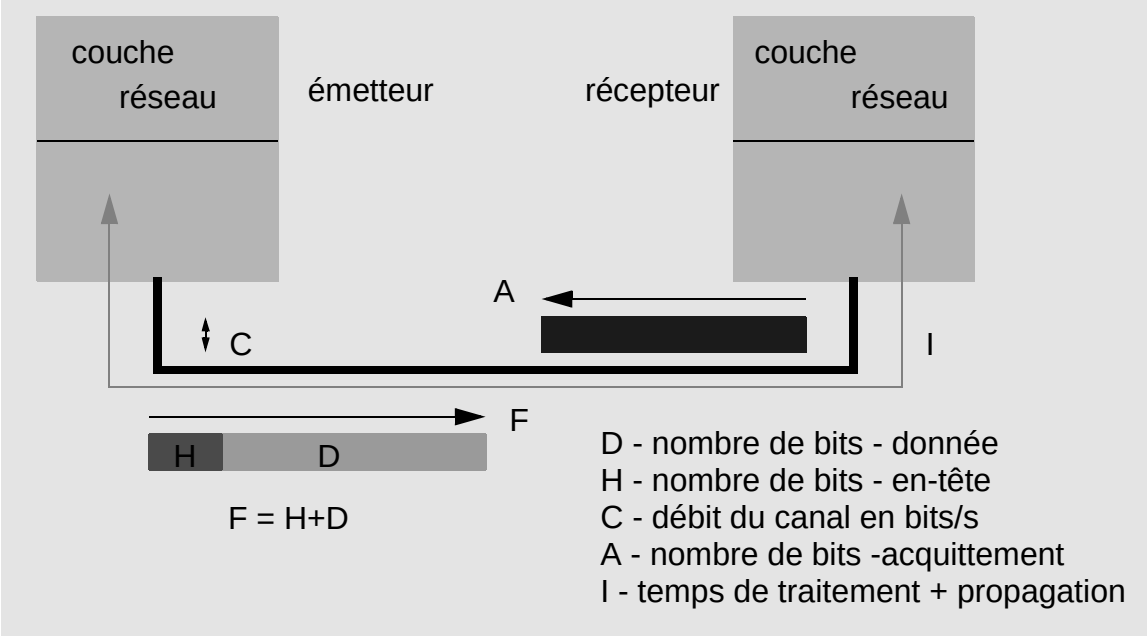
Protocole simple

Pour le protocole simple sur un canal fiable la performance (ou plus précisément le taux d'utilisation de la capacité de transmission) peut être exprimée par:

$$U = D/H+D = D/F$$

où: **U** - le taux d'utilisation (<100%), **D** - le nombre de bits de données par trame, **H** - le nombre de bits d'un en-tête de trame, **F - (D+H)** la longueur totale de la trame.

FIGURE 83. Modèle fonctionnel pour l'évaluation des performances



Protocole "envoyer et attendre" - canal fiable

$$U = D/ (F+A+ 2*C*I)$$

où: **A** - le nombre de bits contenus dans une trame d'acquittement, **C** - le débit du canal en bit/s, **I** - le temps d'interruption et de service plus le *délai de propagation.*

Interprétation:

- prenons comme origine de l'axe des temps le moment où l'émetteur commence à envoyer une trame,
- à l'instant **(F/C + I)**, le dernier bit de la trame de données est parvenu au récepteur, il est prêt à envoyer la trame d'acquittement,

- à l'instant (**F/C + I + A/C + I**), l'émetteur a traité la trame de l'acquittement et est prêt à envoyer la prochaine trame de données,
- pour calculer la "bande passante", ou la capacité de transmission prise pendant cette échange, il faut multiplier (**F/C + A/C + 2*I**) (le temps d'utilisation) par **C** - le débit du canal, soit **F+A + 2*C*I**,
- le nombre de bits de données porté par une trame est **D**.

Le taux d'utilisation du canal (ou l'efficacité du protocole) est donc: **D/(F+A + 2*I*)**

Protocole "envoyer et attendre" - canal non fiable

Dans ce protocole si la trame de données est perdue ou erronée l'émetteur attendra **T** secondes (l'expiration du temporisateur - *timeout*) après l'envoi du dernier bit. Ainsi un échec de transmission d'une trame se solde par la perte de **F+C*T** bits de la capacité du canal. Si le nombre moyen de retransmissions est **R,** la capacité perdue est **R*(F+C*T)**.

La probabilité d'une transmission avec succès peut être calculée par:

$$(1- Pd)*(1-Pa)$$

Et la probabilité d'une transmission avec échec peut être calculée par:

$$L=1-(1-Pd)*(1-Pa)$$

où: **Pd** - la probabilité d'erreur dans la trame de données, **Pa** - la probabilité d'erreur dans la trame d'acquittement.

La probabilité d'effectuer exactement i tentatives (**i-1** retransmissions) est:

$$(1-L)*L^{i-1}$$

A partir de cette évaluation ,on obtient le nombre moyen de transmissions par trame de:

$$1/(1-L)$$

et un nombre moyen de retransmissions:

$$R = 1/(1-L) -1 = L/(L-1)$$

En utilisant la valeur de **R** que nous venons de déterminer, nous arrivons à un taux d'utilisation du canal:

$$U= D/((L/(L-1))(F+C*T) + (F+A+2*I*))$$

Si la **variance** du temps de service du récepteur est faible, l'émetteur peut régler la durée du *timeout* de son temporisateur de façon qu'il expire juste après l'arrivée de la trame d'acquittement:

$$T=A/C + 2*I$$

Dans ce cas, l'efficacité du protocole devient:

$$U = (D/(H+D))*(1-Pd)*(1-Pa)*(1/(1+ C*T/(H+D)))$$

Exemple:

Une ligne de 100 Km (1Km-> 5μs) fiable est gérée par un protocole "envoyer et attendre". Le temps de traitement (donnée et acquittement) est deux fois plus important que le temps de transit; donc **I** =500 μs + 2*500 μs. La trame de 10^4 bits avec une en-tête de taille négligeable (**H**=0, **D**=10^4) est débitée avec la vitesse de 1 Mbit/s (**C**=10^6). Le premier bit de l'acquittement confirme la réception de la trame de données (**A**=0).

L'efficacité de ce protocole:

$$\mathbf{D/(D+H+A+2*C*I) = D/(D+2*C*I)} = 10^4/(10^4 + 2*10^6*1500*10^{-6})=$$

$$= 10^4/(10^4 + 3000)= 1/1,3= 0,769= 77\%$$

Performances du protocole à anticipation

En premier lieu, nous supposons un canal sans erreur. Si la fenêtre est suffisamment large, l'émetteur peut émettre comme il veut; les acquittements arrivent avant que la limite de la fenêtre soit atteinte. Le temps de transmission d'une trame est **F/C**; donc l'émetteur peut émettre des trames jusqu'à l'instant **W*F/C**. Le premier acquittement est envoyé **2*I** secondes après que la première trame a été transmise, et arrive à l'instant **F/C +2*I**.

Conclusions:

Cas 1: l'émetteur pourra transmettre sans interruption si

$$\mathbf{W*F/C > F/C +2*I.}$$

L'efficacité dans ce cas (grande fenêtre) est:

$$\mathbf{U= D/(H+D)}$$

Cas 2: Si le **W*F<F/C+2*I** (petite fenêtre), l'efficacité est:

$$\mathbf{U= D/(H+D) * W/(1+2*C*I/(H+D))}$$

Supposons maintenant un canal avec erreurs.

Dans le cas d'une grande fenêtre d'anticipation, les transmissions se font en continu, sauf si l'on doit envoyer des trames supplémentaires pour corriger les trames erronées. Nous avons vu que le nombre moyen de transmissions par trame est **1/(1-L)**. Pour recevoir **W** trames sans erreur, il faut donc en transmettre **W/(1-L)**.

Nous obtenons:

Cas 3: (grande fenêtre)

$$\mathbf{U = D/(H+D) * (1-L)}$$

Cas 4: (petite fenêtre)

$$\mathbf{U= D/(H+D) * (1-L) * W/(1+2*C*I/(H+D))}$$

Conclusion

Les figures ci-dessous montrent :

- le taux d'utilisation du canal en fonction de la longueur du câble.
- le taux d'utilisation du canal en fonction de la taille de la fenêtre

Dans les deux cas, l'en-tête (**H**) représente 20% de la trame.

FIGURE 84. Taux d'utilisation du canal en fonction de la longueur du câble

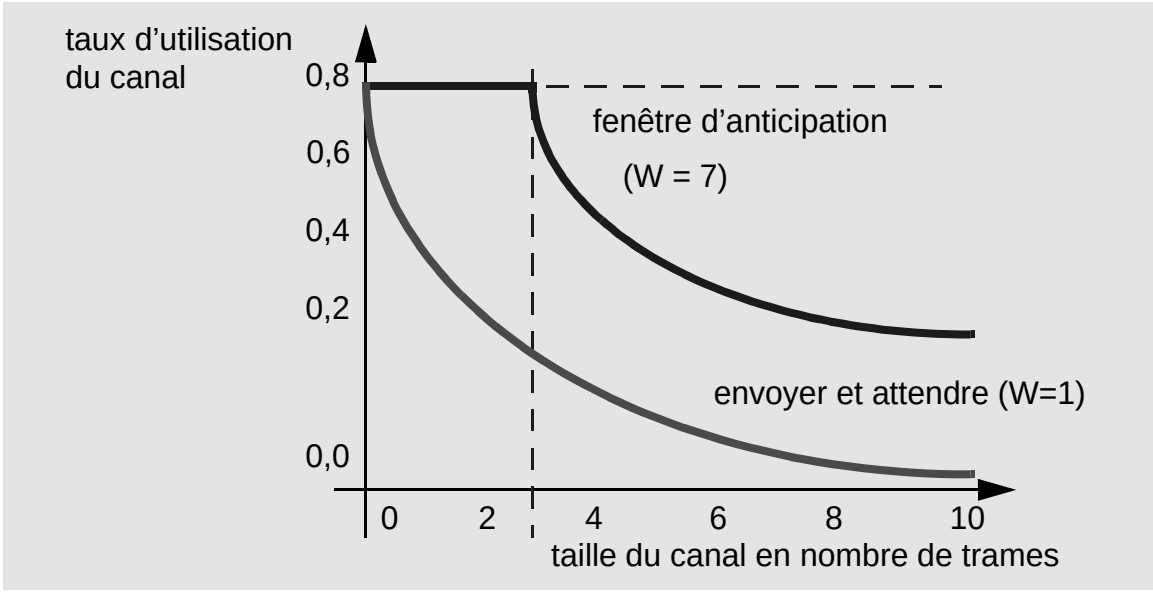
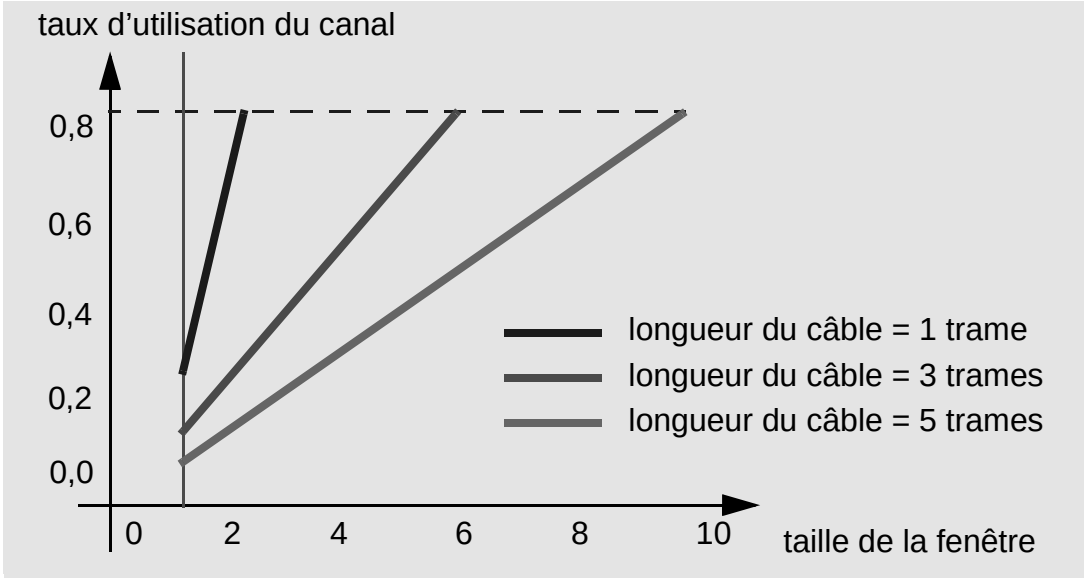


FIGURE 85. Taux d'utilisation du canal en fonction de la taille de la fenêtre (W)



Couche liaison dans réseaux locaux

La couche liaison LLC (*Logical*L*ink Control*) du standard IEEE 802 est utilisée au-dessus de la couche MAC. Le rôle principal de cette couche est de pouvoir échanger les données entre les utilisateurs d'un ou plusieurs réseaux contrôlés par le protocole MAC.

Le standard 802.2 spécifie différentes formes de services:

- service sans connexion et sans acquittement,
- service sans connexion avec acquittement,
- service en mode connecté.

Le service sans connexion et sans acquittement permet simplement d'envoyer et de recevoir les datagrammes au nivau du lien - *link level control protocol data unit* (LLC PDU). Dans chaque trame on envoie l'adresse source et l'adresse destination.

Dans le service sans connexion avec acquittement, l'utilisateur peut envoyer les données et recevoir un acquittement sur cette donnée.

Le service connecté permet d'établir une connexion logique. L'utilisateur peut être notifié si une connexion est établie ou libérée. Une connexion contient les fonctions du contrôle de flux, du séquencement et de la retransmission.

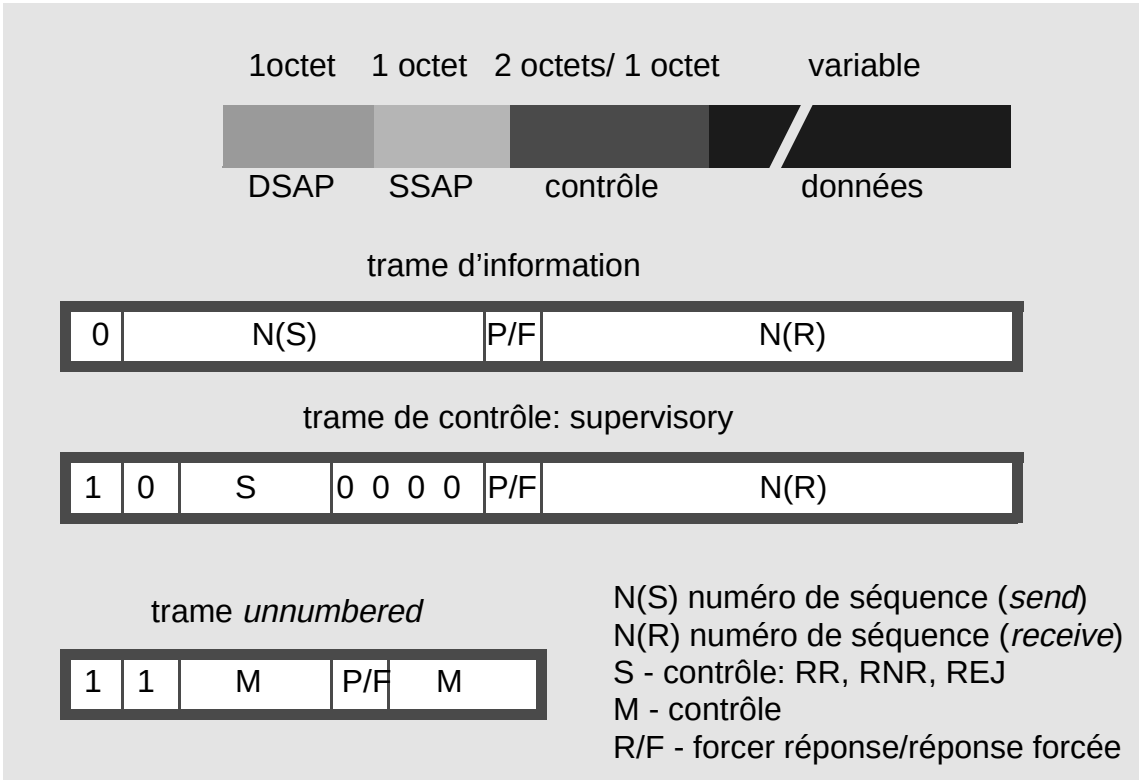
Dans ces trois types de communication, on utilise le même format de trames composé de quatre champs:

- **DSAP** - *destination service attachment point*: l'adresse de destination ou le numéro du lien logique,
- **SSAP** - *source service attachment point*: l'adresse de destination ou le numéro du lien logique,
- **contrôle** - numéros de séquence et d'acquittement,
- **donnée** - données utilisateur.

Selon la nature de la communication, il y a trois sortes des champs de contrôle:

- **information** - la trame transporte les données utilisateur, le champ contient les numéros de séquence en émission et en réception (acquittement) et indicateur **P/F** - *poll/final*,
- ***supervisory*** - la trame est utilisée par le contrôle de flux et de gestion d'erreurs; le champ contient un indicateur S sur deux bits pour indiquer trois situations: *receive ready* - **RR**, *receive not ready* - **RNR** et *reject* (**REJ**),
- ***un-numbered*** - trame de contrôle.

FIGURE 86. Différentes trames du protocole LLC (IEEE 802.2)



Résumé

Les protocoles de lien ou de liaison sont exploités au niveau de la couche logique. Néanmoins, les mécanismes très semblables sont implémentés dans la couche de transport permettant de fiabiliser la transmission entre deux points finaux de communication (*end-to-end communication*). Selon la nature de la communication, l'utilisateur peut choisir entre un mode de communication sans ou avec acquittement et sans ou avec une connexion. Les communications avec les contraintes du temps réel nécessitent l'utilisation du code correcteur pour retrouver l'information correcte immédiatement après la réception. Les communications sans contraintes temporelles utiliseront de préférence un protocole avec détection et retransmission.

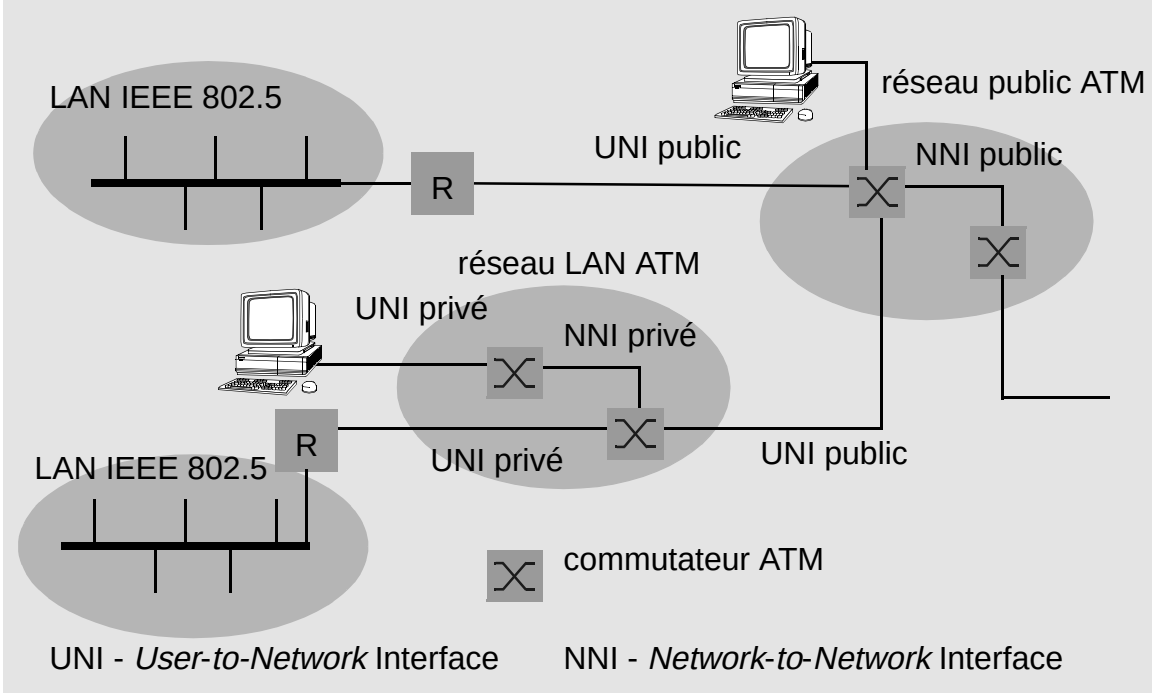
CHAPITRE 5

Réseaux ATM

L'architecture et les fonctions des réseaux ATM (*Asynchronous Transfer Mode*) sont suffisament complètes et complexes pour être traitées dans un chapitre indépendant. La technologie ATM propose d'accéder aux haut débits et d'unifier differents types du trafic - données, voix, vidéo ,..

L'ATM peut s'appliquer aux domaines des réseaux locaux (LAN) et des réseaux longue distance (WAN).

FIGURE 87. Topologie et interfaces de réseau ATM



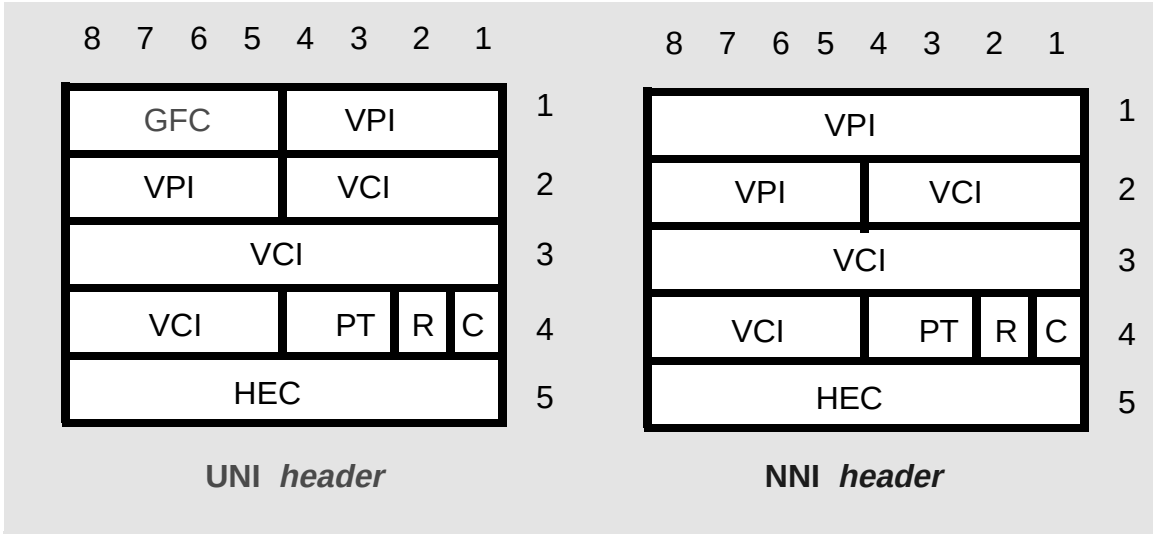
L'ATM est une technique de transmission par commutation et multiplexage. Il constitue une variante de la commutation par paquets basée sur des paquets courts de longueur fixe appelés cellules. Le traitement d'une cellule par un commutateur se limite à l'analyse d'une en-tête dans laquelle est enregistré le numéro de voie (circuit logique). Les fonctions plus complexes, telles que le contrôle de flux et le traitement des erreurs ne sont pas effectués dans le réseau ATM. Cette simplification permet de supporter des contraintes temporelles associées au trafic en temps-réel de la voix et des images.

Structure d'une cellule ATM

Une cellule ATM de longueur de 53 octets contient deux champs principaux:

- l'entête (*header*) de 5 octets dont le rôle est d'identifier les cellules appartenant à une même connexion et d'en permettre l'acheminement; l' en-tête est protégée par un octet du code de correction HEC (*header error correction*),
- le champ d'information contient la charge utile - *payload.*

FIGURE 88. Structure d'une cellule ATM: interface utilisateur, interface réseau



Couches fonctionnelles ATM

L'architecture fonctionnelle du réseau ATM est composée de trois couches:

- la couche physique qui assure l'adaptation du protocole ATM au support de transmission,
- la couche ATM en charge du multiplexage et de la commutation des cellules,
- la couche AAL (*ATM Adptation Layer*) qui adapte les flux d'information à la structure des cellules.

Couche physique

Le flux de cellules généré par la couche ATM peut être en pratique transporté dans la charge utile d'un quelconque système de transmission numérique. Cette caractéristique permet au système ATM de s'adapter aux systèmes de transmission actuels ou à venir: SONET/SDH/E1/E3 (WAN), *Fiber Channel*, VDSL, ..

La couche physique fournit deux sous-couches fonctionnelles:

- la sous-couche de media physique, qui prend en charge codage, décodage, embrouillage et adaptation au support,
- la sous-couche de convergence qui s'occupe de l'adaptation du débit, de la protection de l'en-tête, et de la délimitation des cellules.

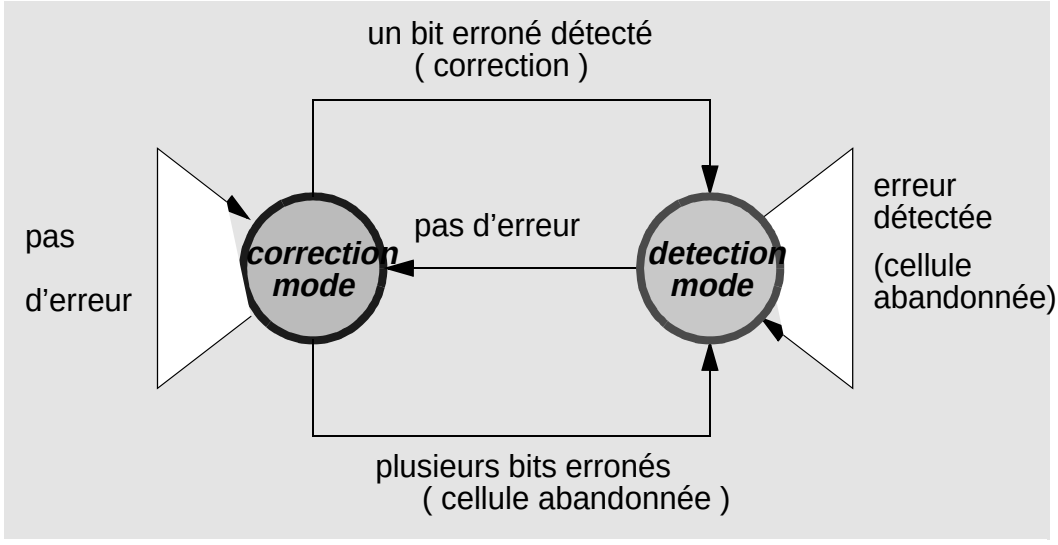
L'adaptation de débit est nécessaire pour pouvoir projeter les flux de cellules ATM sur le débit utile d'un lien physique. Pour un système de transmission à débit constant cette adaptation peut être effectuée par l'insertion de cellules vides.

La protection des cellules est assurée par le champ HEC. Ce champ permet de détecter des erreurs multiples et de corriger une erreur isolée dans l'en-tête de la cellule.

Le récepteur dispose d'un mode de correction et d'un mode de détection:

- en **mode correction** les cellules correctes sont passées à la couche ATM; les cellules dont le HEC présente un syndrome d'erreur simple sont passées après la correction de l'en-tête à la couche ATM; les cellules avec erreurs multiples sont détruites et le récepteur passe en mode détection,
- en **mode détection** toutes les cellules avec une ou plusieurs erreurs sont détruites; la détection d'une cellule avec le HEC correct provoque le retour au mode correction.

FIGURE 89. Mode correction et mode détection basés sur le code HEC



La valeur du HEC est calculée de la manière suivante:

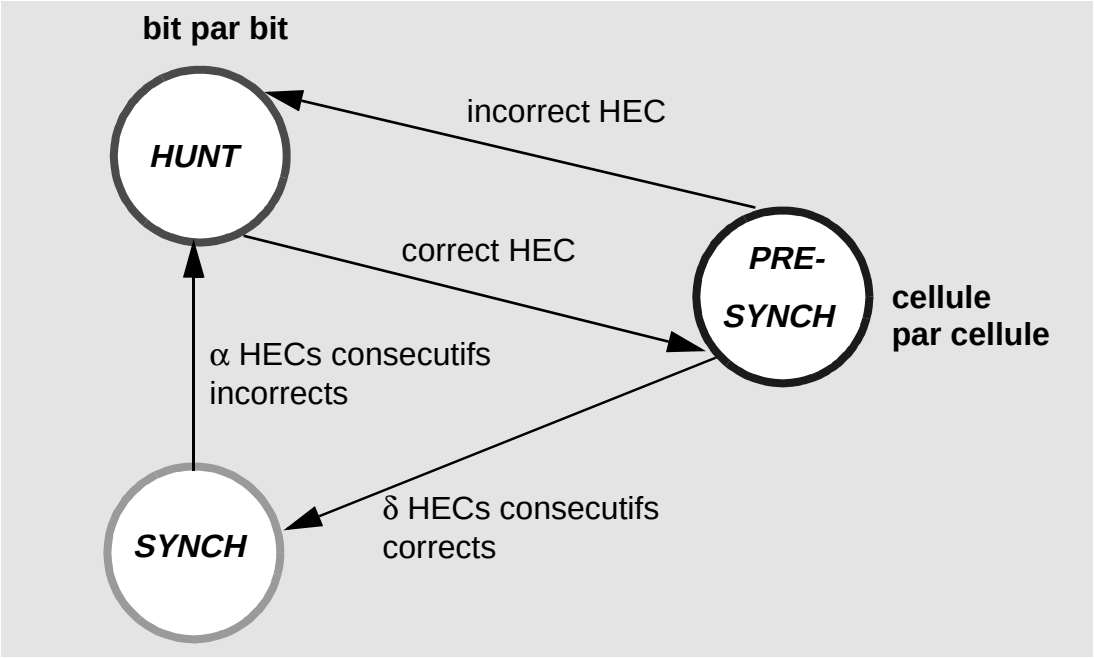
- les 4 premiers octets (32 bits) sont pris comme coefficients d'un polynome **M(x)** de degré 31,
- le polynome **M(x)** est multiplié par x^8, puis divisé par un polynome générateur: $G(x)=x^8 + x^2 + x + 1$;
- le polynome $C(x)= x^6 + x^4 + x^2 + 1$ est additionné modulo 2 au reste de la division $M(x)\%G(x)= R(x)$;
- le polynome **R(x)** forme la séquence de 8 bits du HEC.

L'octet HEC ainsi genéré créé un code sur 36 bits d'une distance de Hamming de quatre, propriété qui autorise la correction d'une erreur simple et la détection des erreurs multiples.

Délimitation des cellules

La délimitation des cellules est basée sur la détection du champ HEC correct de l'en-tête de cellule. La détection d'une cellule sur un flux binaire continu est obtenue en déterminant l'emplacement de l'octet HEC où des règles de codage se trouvent vérifiées.

FIGURE 90. **Mécanisme de cadrage des cellules ATM**



Dans l'état de *hunt* (recherche) des limites de cellules, on vérifie si les règles de codage du HEC s'appliquent à l'en-tête présumée. Après la détection d'une cellule validée par un HEC valide, le mécanisme passe dans l'état de *pre-synch* (pré-synchronisation). La détection de delta (δ) HEC consécutifs valides autorise le passage en état de *synch* (synchronisation).

Dans l'état *synch* le mécanisme continue la vérification des HEC; en cas de détection de alpha (α) HEC consécutifs et incorrects le mécanisme retourne en état *hunt*.

Deux ensembles de valeurs alpha et beta ont été proposés:

- alpha = 7 , beta = 8 pour les systèmes de transmission par paquets (fiber channel, ..)
- alpha = 7 , beta = 6 pour les systèmes de transmission synchrone: (SONET/SDH/HDLS)

Le temps entre deux pertes de cadrage dépend du taux d'erreurs dans le système de transmission. Pour une valeur BER=10^{-6} ce temps est supérieur à 10^{30} cellules.

Couche ATM

Les fonctions remplies par la couche ATM correspondent aux valeurs contenues dans les différents champs de l'en-tête de la cellule:

- **champ GFC**: le contrôle de flux est exercé par les 4 bits du champ GFC; si un noeud réceptionne plus de 10 cellules comportant un champ GFC différent de zéro, sur 30 000 cellules reçues, alors il passe en mode contrôle; ce mécanisme intervient en cas de surcharge de trafic à l'interface,

- **champs VPI/VCI**: le routage des cellules repose sur l'existence de couple *Virtual Path Identifier/Virtual Circuit Identifier*; dans l'interface UNI, le couple d'identificateurs VPI/VCI vaut 8+16=24 bits, tandis que dans l'interface NNI il vaut 12+16=28 bits; la concaténation de plusieurs VC détermine une connexion de conduits virtuels (VPC); une connexion avec un circuit virtuel VC peut être semi-permanente ou établie à la demande au moment de la négociation entre abonné et réseau.
- **champ PTI**: *payload type indicator* identifie le type d'informations véhiculé par la cellule; la cellule peut acheminer des données utilisateur ou des données de service qui sont utilisées pour la gestion et la maintenance du réseau (OAM - *Operation Administration Maintenance*); le champ PTI peut discerner les cellules d'information utilisateur associées au flux F5 sur une même connexion; les flux F3/F4 nécessitent des connections propres au contrôle.

TABLEAU 7. Codage du champ PTI: *payload type indicator*

C1	C2	C3	caractèristiques
0	0	0	données utilisateur: fragment SAR AAL5
0	0	1	données utilisateur: dernier fragment SAR AAL5
0	1	0	données utilisateur en congestion: fragment SAR AAL5
0	1	1	données utilisateur en congestion: dernier fragment SAR AAL5
1	0	0	flux F5 : local
1	0	1	flix F5: bout en bout
1	1	0	gestion de ressources: cellules RM - resource management
1	1	1	réservé

- **bit CLP**: cet indicateur est positionné à 1 si la cellule possède une faible priorité; dans le cas d'un débit constant toutes les cellule possèdent une haute priorité (CLP=0)

Couche adaptation

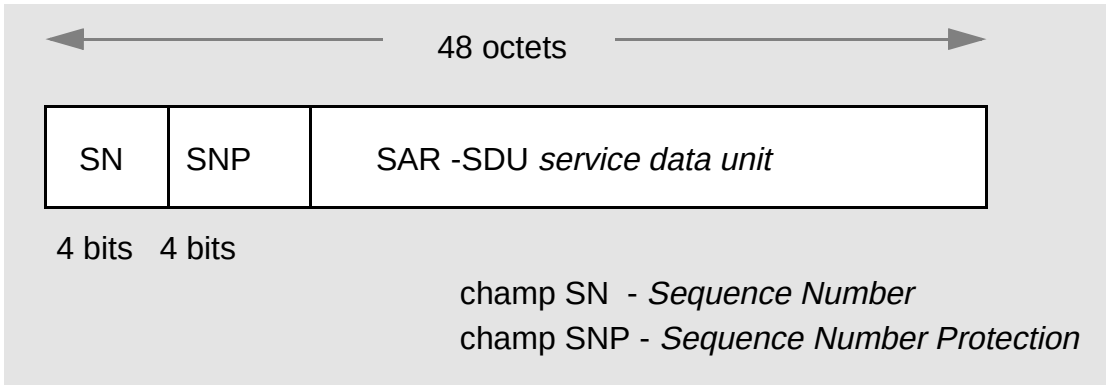
Le rôle de la couche d'adaptation AAL consiste à offrir des services pour les différentes classes d'application. Elle découpe et rassemble (SAR - *segmentation and reassembly*) les données applicatives en blocs de 48 octes. La couche AAL contribue également à la qualité de services (QoS - *quality of service*). Quatre classes de service ont été définies: A, B, C, et D.

TABLEAU 8. Classes et modes de fonctionnement au niveau des applications

classe	synchronisation	débit	mode
A	oui	constant	connecté : CBR
B	oui	variable	connecté : VBR
C	non	variable	connécté
D	non	variable	non connecté : ABR, UBR

Dans la couche AAL 1 le transfert d'information est synchronisé de bout-en-bout. La sous-couche de segmentation et assemblage fonctionne avec un PDU de 48 octets. Les informations de contrôle sont placées dans le premier octet. Elles contiennent les numéros de séquence (SN), les bits de protection (SNP) et un code détecteur d'erreur. Le bit indicateur CSI permet d'effectuer la synchronisation au niveau AL.

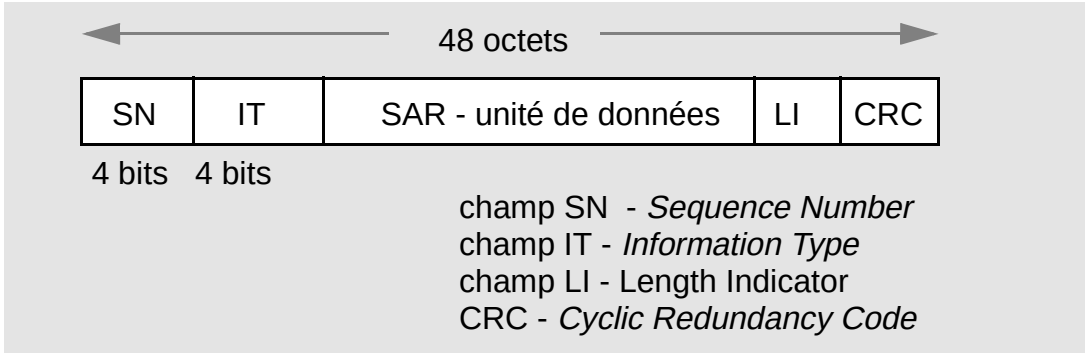
FIGURE 91. Unité PDU pour AAL1 - débit constant



La couche d'adaptation de type AAL2 gère un service à débit variable (voix ou vidéo compressée). Elle traite les fonctions de segmentation et de réassemblage des données applicatives, la gestion des cellules perdues ou incomplètes, la recherche des erreurs et l'adaptation du débit. Une donnée utilisateur est encadrée par quatre champs de contrôle:

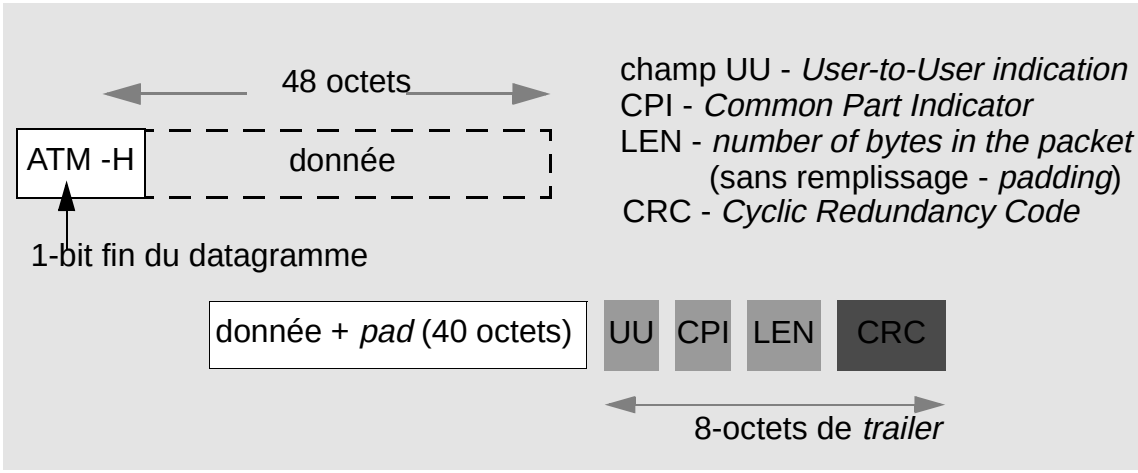
- numérotation de cellule (SN),
- type d'information (TI),
- indicateur de remplissage,
- code CRC de détection d'erreur.

FIGURE 92. Unité PDU pour AAL2 - débit variable



La couche AAL5 ne supporte pas de multiplexage; les PDUs sont directement segmentés et forment, sans en-tête supplémentaire, la sous-couche SAR. Pour discerner la position du fragment dans la séquence de SAR on utilise le champ PT inclus dans l'en-tête ATM. Dans la sous-couche de convergence, le champ PAD contient des octets de bourrage (de 0 à 47) dont le rôle consiste à rendre la longueur du champ d'information multiple de 48 octets. Le champ *length* caractérise et délimite la taille d'un segment à 2^{16} octets. Le code CRC est disposé sur 4 octets.

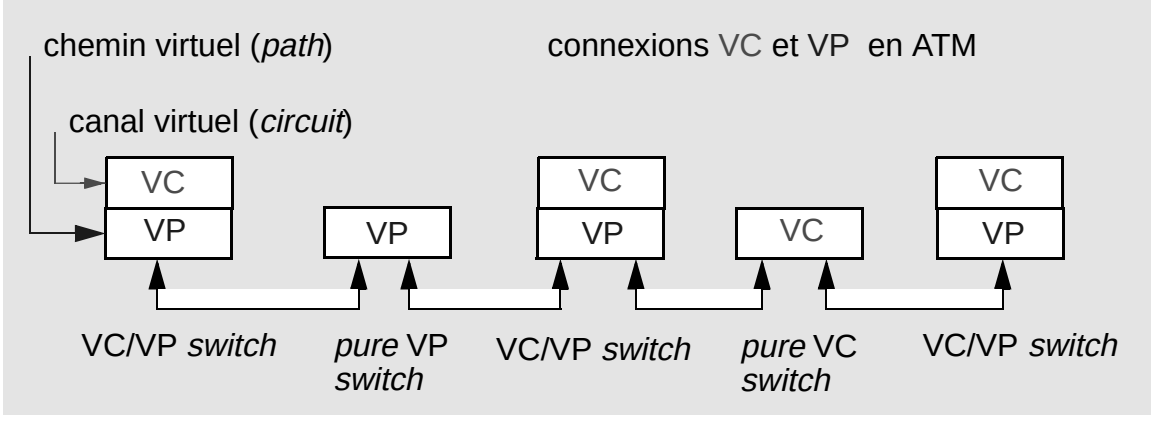
FIGURE 93. Unité PDU pour AAL5 - plusieurs cellules ATM.



Commutateurs ATM

Les noeuds de communication ATM sont des commutateurs qui assurent la mise en correspondance des couples d'identificateurs VCI/VPI. Le routage et l'acheminement des cellules est effectué par l'association d'un ou plusieurs ports de sortie à un port d'entrée.

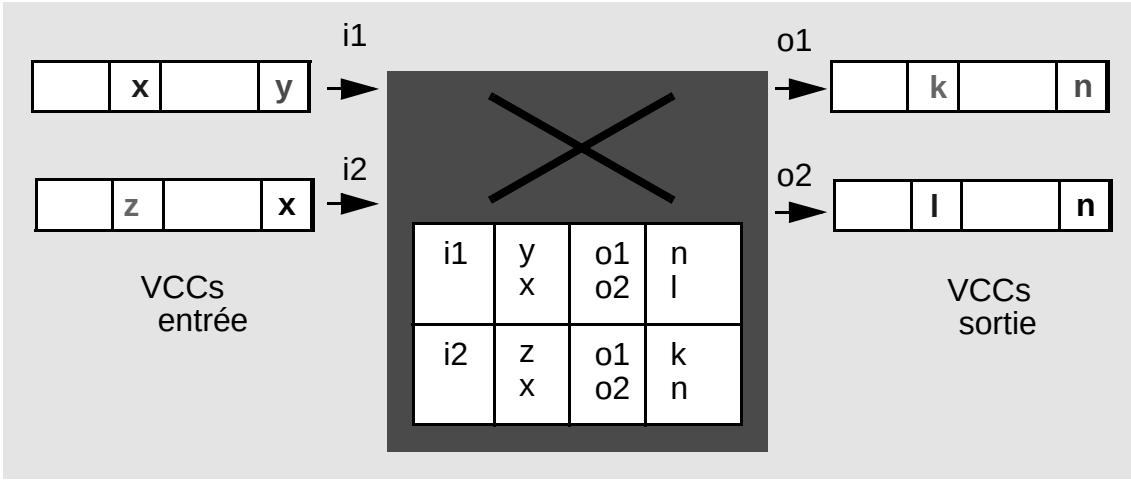
FIGURE 94. Commutation des circuits et des voies (VC/VP)



Les débits élevés imposent l'utilisation des architectures matérielles pour la réalisation des commutateurs ATM. Les commutateurs peuvent être classifiés selon l'architecture de commutation utilisée. Les techniques les plus utilisées sont:

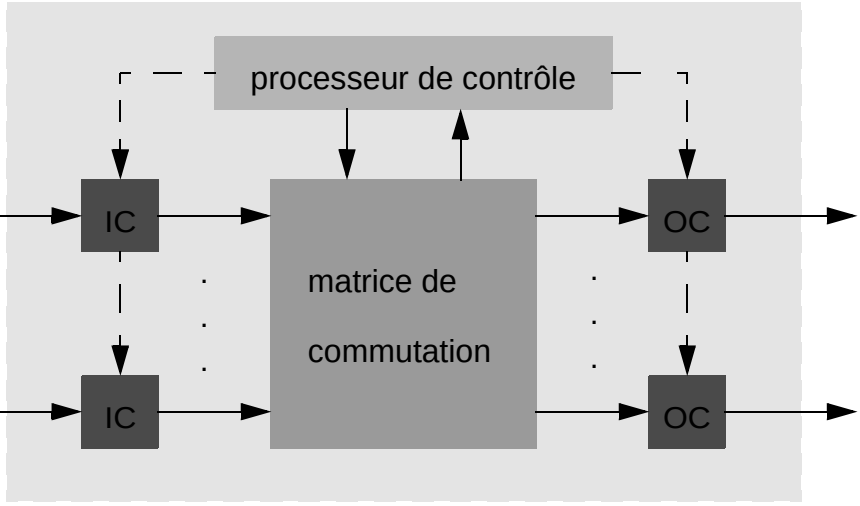
- commutation temporelle,
- commutation spatiale,
- commutation spatiale multi-étages.

FIGURE 95. Principe de commutation ATM



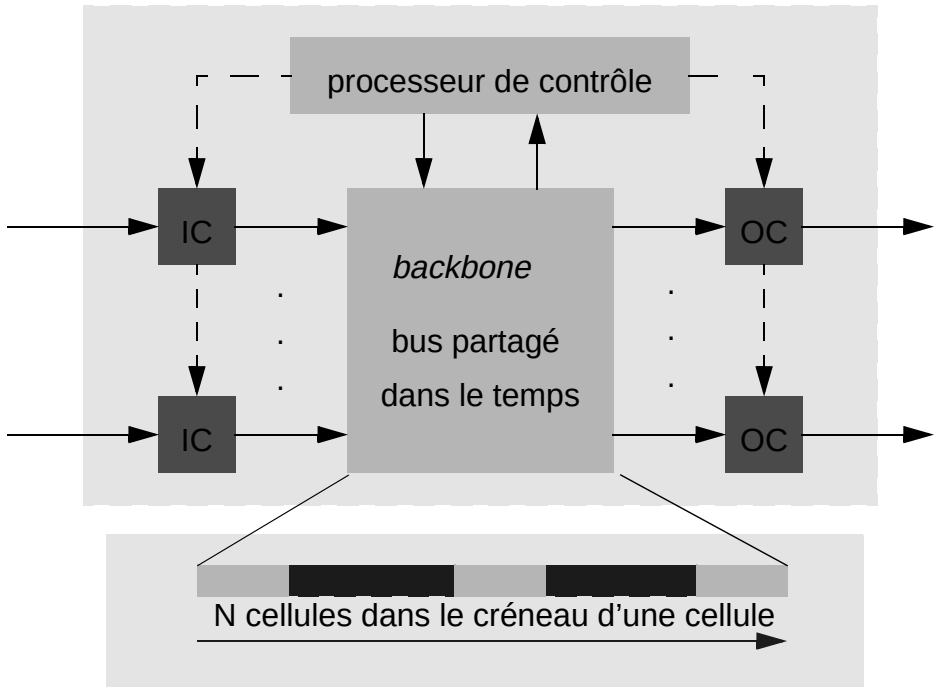
Un commutateur est composé d'un processeur et d'une matrice de commutation. Selon l'architecture interne la matrice de commutation peut réaliser la commutation de la façon spatiale ou temporelle

FIGURE 96. Commutateur ATM: le processeur - contrôleur et la matrice de commutation



Dans la commutation temporelle le circuit est capable d'«absorber» N cellules (N - nombre d'entrées) dans l'intervalle de temps correspondant à une cellule (cycle cellule). Le bus système est de taille 53*8 bits et il est capable de transmettre une cellule dans un cycle horloge. Le contrôleur d'entrée ajoute à chaque cellule le numéro du port de sortie; cette information sert à sélectionner le contrôleur de sortie qui doit lire et stocker la cellule dans un tampon. S'il y a plusieurs cellules destinées au même port de sortie, elles doivent être stockées dans un tampon associé à cette sortie.

FIGURE 97. Architecture d'un commutateur temporel

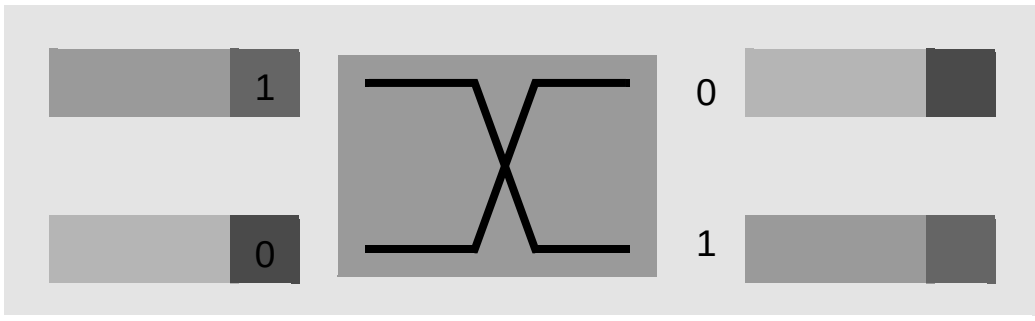


Le nombre d'entrées dans un commutateur temporel dépend de la vitesse du bus système. Par exemple un bus à 2,5 Gbit/s peut supporter 4 entrées à 622 Mbit/s ou 16 entrées à 155Mbit/s chacune.

La commutation spatiale est une solution avec une matrice de commutation **N sur N**. Cette architecture offre plusieurs chemins entrée/sortie indépendants. Afin d'éviter la congestion, le commutateur à **N** entrées doit fonctionner avec cadence **N** fois plus importante que la cadence d'arrivée d'une cellule. Les contrôleurs de sortie nécessitent toujours des tampons de sortie. La taille d'un commutateur spatial augmente avec la proportion N^2 par rapport au nombre d'entrées.

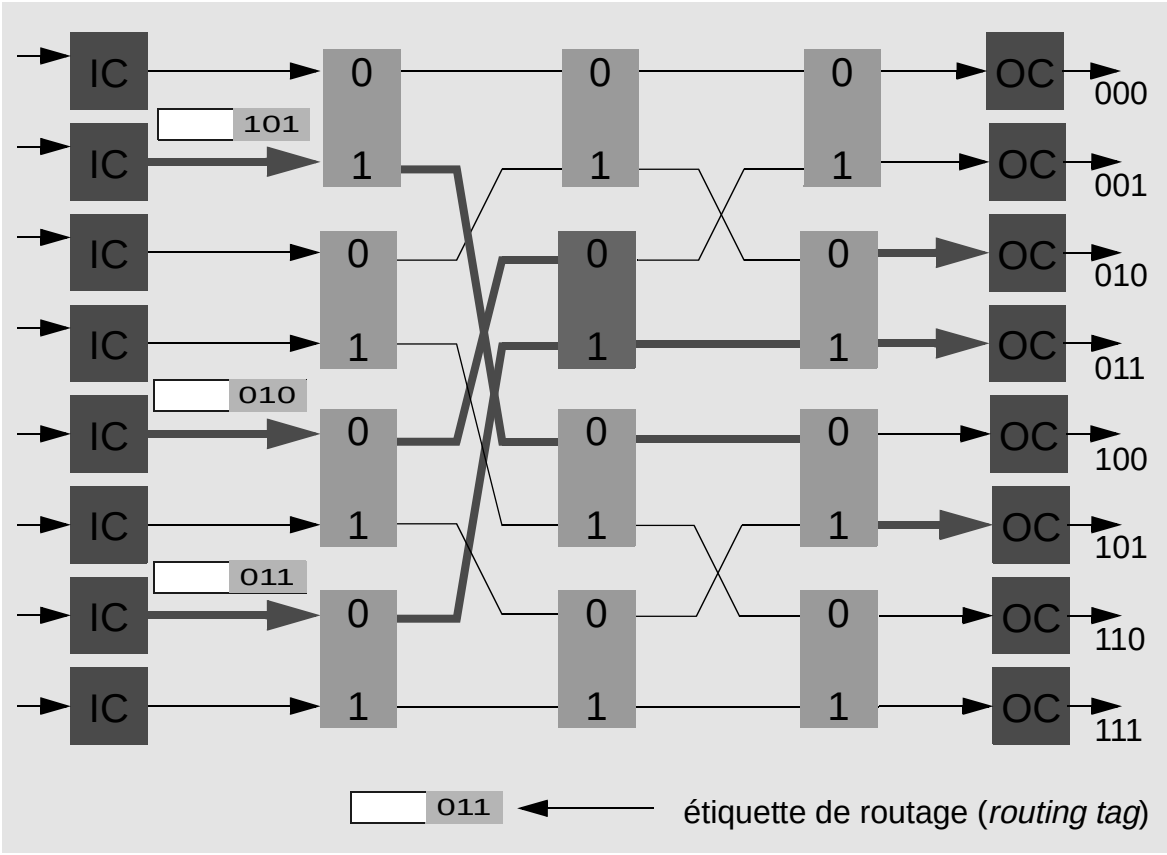
Les solutions pratiques impliquent l'utilisation de plusieurs étages de commutation. Les éléments de commutation sont de taille 2*2 ou 4*4. Le nombre d'éléments de commutation est déterminé par l'expression: **X = N/M**; où **N** est le nombre total des entrées et **M** le nombre d'entrées d'un élément de commutation. Le nombre **Y** d'étages de commutation est déterminé par: $\mathbf{M^Y = N}$.

FIGURE 98. Elément de commutation type delta



Un élément de commutation sélectionne la sortie selon l'état du bit de routage (*routing tag*). Le problème avec les matrices simples de commutation réside dans la possibilité des blocages. L'exemple ci-dessous montre l'apparition du blocage pour les valeurs 010 et 011 de bits de routage.

FIGURE 99. Matrice de commutation simple - commutation delta



Pour éviter les blocages, une solution consiste à utiliser des mémoires tampons, ce qui augmente la durée de transit dans le commutateur. Une autre technique consiste à augmenter le nombre d'étages de commutation en introduisant la redondance au niveau du nombre de chemins.

Résumé

L'ATM est une technologie permettant de transporter tous les types de trafic (données, voix, vidéo) et de couvrir toutes les distances envisageables de quelques dizaines de mètres à plusieurs centaines de kilomètres. Les réseaux ATM s'appuient sur des systèmes de transmission à haut débit de quelques mégabits par seconde à plusieurs dizaines de gigabits. Avec des installations récentes dans les systèmes de télécommunication, il semble que la technologie ATM est incontournable pour le déploiement de toutes sortes de services incluant le système Internet.

CHAPITRE 6

Technologie xDSL et ADSL

Le développement du système INTERNET et l'accès à ce système par une ligne téléphonique nécessite des nouvelles technologies permettant de transmettre des données multimedia avec un débit convenable. Ce débit convenable se situe au niveau de plusieurs centaines de kilobits par seconde. Le débit des modems actuels est limité par la bande passante d'une ligne téléphonique. Les 33 Kbit/s et 56 Kbit/s offerts par les modems le plus rapides ne suffisent pas pour la transmission des données audio (e.g. fichiers MP3) ou vidéo (e.g. fichiers MPEG2).

Un autre problème est associé au mode de fonctionnement des réseaux téléphoniques où chaque communication nécessite l'établissement d'une connexion. Une connexion, ou un circuit virtuel, bloque de façon continue la bande passante du réseau même si les données sont envoyées en rafales avec de longues périodes de silence.

Pour s'assurer un lien performant il est possible de louer une ligne (un accès) primaire au système téléphonique. Une telle ligne (e.g. E1) intègre 32 canaux de 64 Kbits/s et offre le débit de 2,048 Mbit/s. Néanmoins, elle nécessite l'installation de deux paires du câble avec amplificateurs. Pour un utilisateur individuel, le problème est le prix qui dépasse 1K*Euro* par mois. D'autre part une ligne E1 est totalement numérique et ne permet pas de supporter directement l'équipement analogique d'un abonné.

On cherchera donc une solution technique permettant de transmettre des données analogiques et numériques sur une paire de câbles (boucle locale d'abonné) avec un débit relativement important de l'ordre de 500 Kbit/s ou plus.

La solution est apportée par la technologie xDSL.

Famille xDSL

ADLS est beaucoup plus qu'une technologie d'accès pour les abonnés sur la boucle locale. ADSL est une des nombreuses techniques d'accès qui peuvent être utilisées pour transformer une ligne ordinaire en un lien numérique à haut débit et éviter la surcharge d'un réseau téléphonique commuté. Cette technologie forme une famille appelée xDSL (*x-type Digital Subscriber Line*). Certains membres de cette famille utilisent les signaux analogiques de transmission du contenu analogique ou numérique via des lignes analogiques de la boucle locale. D'autres membres de la famille xDLS utilisent les vrais signaux numériques pour le transfert des données numériques. Ils sont proches des techniques utilisées pour la communication numérique sur les réseaux de télécommunication: RNIS ou E1,E3.

Les origines du système DSL se trouvent dans le développement du réseau RNIS dont l'objectif était de transformer la totalité du réseau téléphonique en signalisation numérique. La solution RNIS offre un accès de base

qui opère sur deux canaux duplex de 64 Kbits/s (B *channels*) et sur un canal de contrôle de 16 Kbits/s (*channel* C). Les deux canaux B peuvent être combinés et fournir le débit de 128 Kbit/s. A l'heure actuelle, ces débits ne sont pas impressionnants et diffèrent de peu par rapport aux modems de haute qualité (56 Kbit/s).

La solution xDSL permet de déployer les canaux dont le débit est au moins de 1,5 Mbit/s avec le mode de fonctionnement symétrique (*duplex*) ou asymétrique où le débit descendant (*downstream*) est plus important que le débit ascendant (*upstream*). Remarquons que les services de haut débit offerts par les fournisseurs d'accès à l'Internet ont un caractère asymétrique. La majorité du trafic est en mode descendant vers l'utilisateur (client). Dans ce cas ,il est intéressant de supporter le plus grand débit vers l'utilisateur terminal, tout en assurant un débit plus faible vers le fournisseur d'accès.

En effet, les nouvelles versions du xDSL (ADSL, RADSL, et VDSL) sont essentiellement asymétriques. Le VDSL est une technologie prometteuse pour les applications à très haut débit. Cette technologie nécessite l'utilisation de fibre optique au moins sur la moitié de la distance entre le centre de commutation et l'utilisateur. Les débits proposés sont 13 à 52 Mbit/s dans le sens descendant et de 1,5 à 6 Mbit/s dans le sens ascendant.

Les caractéristiques techniques de la famille xDSL sont présentées dans la liste ci-dessous:

- HDLS/HDLS2 - *High-bit-rate DSL*. Technologie de transmission symétrique utilisée pour l'implémentation des lignes numériques T1 et E1 (1, 544 Mbits/s et 2, 048 Mbits/s). La version HDLS2 a besoin d'une seule paire torsadée. La longueur du câble doit être inférieure à 4,5 Km.
- SDLS - *Symetric DSL* utilise une seule paire torsadée sur une distance de 3 Km maximum. Le débit offert par cette technologie est 768 Kbit/s.
- ADSL - *Asymetric DSL* a les mêmes caractéristiques que SDLS mais supporte un trafic asymétrique et est capable de fonctionner sur une paire torsadée de 5,4 Km maximum.
- RADSL - Rate-*adaptive* DSL est une technologie ADSL avec possibilité d'adaptation du débit à l'état de la ligne. Cette technique est actuellement disponible en ADSL implémenté avec le codage DMT (*discrete multitone*)
- IDSL - ISDN DSL (DSL pour RNIS) est employé pour l'implémentation des canaux RNIS (2B+C) sur l'équipement DSL
- VDSL - *High-speed* DSL est une nouvelle technologie permettant d'obtenir de très haut débits sur une paire torsadée longue de 1 500 m. La principale application de cette version DSL est le portage des cellules ATM.

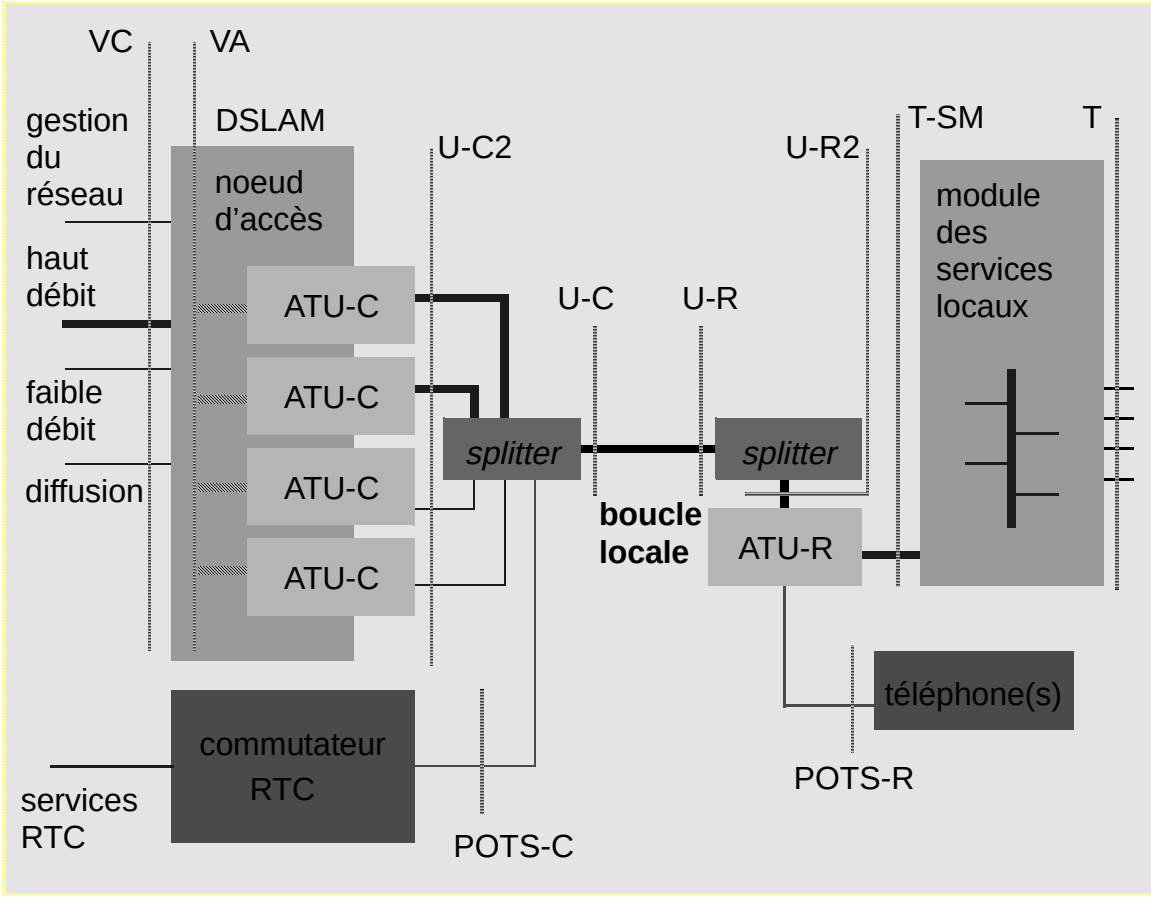
Architecture ADSL

Malgré la confusion sur les rapports entre DSLS, xDSLs, et ADSL, une chose est sure: ADSL est le plus normalisé de tous, ens termes de documentation disponible, évaluations de service, et spécifications ouvertes. La figure ci-dessous montre la structure élémentaire d'un système ADSL. Bien que le schéma puisse sembler un peu complexe, l'arrangement général des composants ADSL est direct.

Entre les interfaces, divers blocs fonctionnels sont définis. Ils peuvent être rassemblés par des vendeurs d'équipement ADSL en des produits qui fournissent les fonctions nécessaires. Les fabricants sont libres d'apporter les options ou les améliorations qui sont nécessaires.

Le fonctionnement interne de ces composants est à définir par les fabricants.

FIGURE 100. **Architecture ADSL**



Les acronymes signifient:

- ATU-C (*ADSL Termination Unit - Central office*): unité de transmission ADSL, côté réseau
- ATU-R (*ADSL Termination Unit - Remote*): unité de transmission ADSL, côté abonné
- DSLAM DSL: multiplexeur d'accès au DSL
- POTS-C: interface entre RTC et *splitter*, coté réseau (POTS - *plain old telephone service*)
- POTS-R: interface entre RTC et *splitter*, côté abonné
- T-SM: T-interface terminale de l'abonné
- U-C: interface U sur la boucle locale côté réseau
- U-C2: interface U sur le *splitter* côté réseau
- U-R: interface U sur la boucle locale côté abonné
- U-R2: interface U sur le *splitter* côté abonné
- VA: interface sur ATU-C, côté noeud d'accès de ATU - C
- VC: interface du fournisseur des services côté noeud d'accès

Une des caractéristiques importantes de l'ADSL est le fait qu'il supporte le service analogique de voix (*plain old telephone service*, ou POTS). Un dispositif spécial appelé *splitter* impose la voie analogique de 4KHz entre le commutateur et l'équipement analogique de l'abonné.

Beaucoup de services sont imaginés pour le système ADSL, incluant la transmission large bande et la diffusion numérique (vidéo et accès Internet), ainsi que la gestion du réseau. Tous ces services sont accédés en dehors du commutateur RTC, résolvant ainsi le problème d'encombrement du commutateur. Beaucoup de liaisons ADSL sont gérées par un noeud d'accès aux services installé dans le central de télécommunications. Ce noeud d'accès est parfois appelé DSLAM (*DSL module d'accè)*. Bien qu'un DSLAM puisse certainement alimenter l'accès de service aux lignes ADSL, une architecture complète d'un DSLAM est beaucoup plus complexe que celle illustrée sur le schéma.

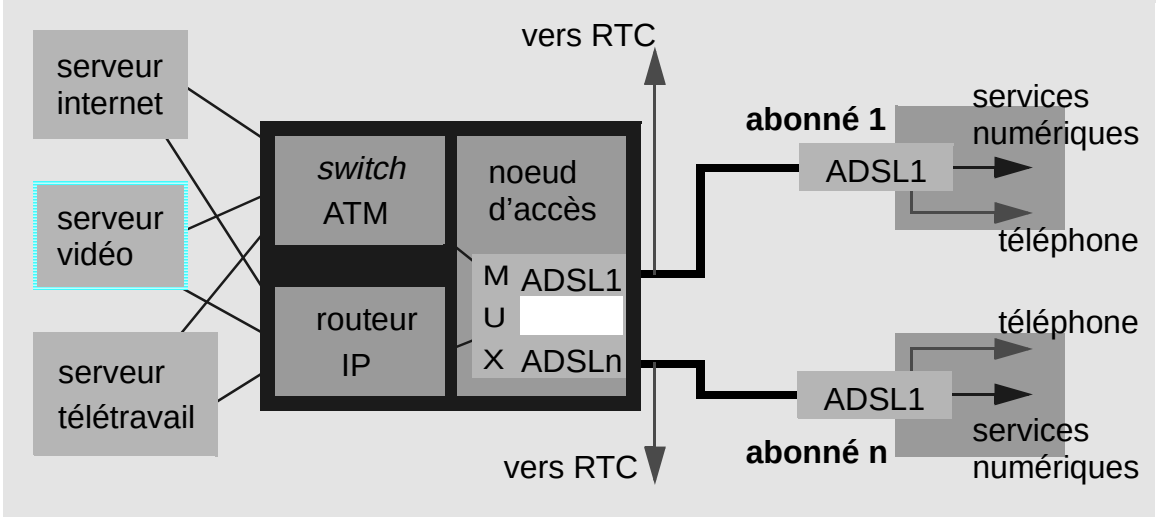
Certaines des interfaces présentées sur le schéma sont optionnelles. L'interface T-SM entre le ATU-R et le module de service pourrait dans certains cas être le même que l'interface T, surtout si le module de service est intégré dans l'ATU-R. Si l'interface T-SM existe, on peut desservir plusieurs types de périphériques incluant les réseaux locaux. Par exemple, un ATU-R pourrait avoir les deux connecteurs: 10Base-T Ethernet et V.35. Les diverses interfaces U ne pourraient pas exister si le *splitter* était une partie intégrante du ATU-C, ou si le *splitter* disparaîtrait entièrement. C'est la nouvelle tendance parmi les vendeurs d'équipement, mais cela empêche l'utilisation des téléphones analogiques sur la même ligne.

Les interfaces V pourraient être logiques plutôt que physiques, ce qui est surtout vrai de l'interface VA si le DSLAM où accède l'ADSL exécute certaines tâches de commutation. Enfin, les installations de ADSL dans les locaux de client peuvent prendre diverses formes. Cela peut être aussi simple qu'une paire de fils raccordée aux périphériques, tels qu'un poste de TV ou un ordinateur personnel, et aussi complexe qu'un réseau local Ethernet.

Un réseau ADSL

La technologie ADSL est beaucoup plus qu'une solution plus rapide pour télécharger des pages Web à la maison. ADSL fait partie d'une architecture réseau qui a le potentiel d'alimenter les sites résidentiels et les petites entreprises avec tous types de nouveaux services en large bande. Dans ce contexte, - «large bande»- signifie les services ayant besoin de connexions à 2 Mbit/s et plus.

FIGURE 101. Configuration ADSL pour les services à haut débit



La figure ci-dessus montre ce à quoi ressemblerait un réseau de large bande basé sur ADSL. Dans la version la plus simple de cette architecture, les clients requerraient essentiellement un nouveau modem ADSL. Ce périphérique aurait une prise ordinaire RJ11 pour le téléphone analogique dans le logement. Autres ports, peut-être 10BASE - T Ethernet, se connecteraient à un ensemble de télévision et PC pour une variété de services, tels que accès rapides à l'Internet ou vidéo sur demande. La fonction de *splitter* sépare le service téléphonique analogique des services numériques. Dans beaucoup de cas, un câblage supplémentaire des locaux peut être nécessaire, mais ceci est au-delà du cadre du réseau ADSL.

Dans l'office central de télécommunications (OC), le service analogique de voix passe au commutateur de voix par le biais d'un *splitter.* Les lignes locales ADSL se terminent maintenant dans un noeud d'accès ADSL au lieu d'aller directement au commutateur central. Le noeud d'accès est un multiplexeur d'accès DSL ou DSLAM. La magistrale (bus système) du noeud d'accès possède des raccordements pour les unités de liaisons aux routeurs IP/TCP et aux commutateurs ATM.

Ces commutateurs et routeurs permettent aux utilisateurs d'accéder aux services de leur choix. Remarquons que ces services peuvent aussi être logés dans l'OC. Dans beaucoup de cas, les serveurs peuvent être localisés à travers la rue et connectés par un câble court à l'OC.

Si le fournisseur de service est aussi le fournisseur du liaison ADSL, il est alors possible que tous les services soient logés directement dans l'OC.

Le noeud d'accès est un point essentiel dans la norme ADSL. Pour l'instant, les noeuds d'accès exécutent une simple agrégation du trafic. Tous les bits et paquets en dehors du noeud d'accès sont reportés aux services par des circuits simples. Par exemple, s'il y a 10 clients ADSL communiquant à 2 Mbit/s dans le sens descendant et 64 Kbit/s dans le sens montant (une connexion typique), alors la liaison entre le noeud d'accès et le réseau des services internet, doit être au moins de 20 Mbit/s (10x2 Mbit/s) dans les deux sens afin d'éviter la congestion et le rejet des paquets.

Les débits entre le noeud d'accès et les serveurs en montant doivent être les mêmes (20 Mbit/s) bien que le débit binaire d'agrégat soit ici seulement de 640 Kbit/s (10X64 Kbit/s). Ceci est dû à la nature duplex des liaisons E de la télécommunication. Une amélioration de base possible pour le système ADSL serait l'introduction d'un **multiplexeur statistique** dans le noeud d'accès ADSL lui-même.

Dans le cas d'un multiplexeur statistique, basé sur la nature *bursty* (rafale) des paquets eux-mêmes, une baisse de vitesse des liaisons est possible parce que tous les utilisateurs ne seront pas actifs en même temps. D'une manière ou d'une autre, les mêmes 10 clients pourraient être servis par une liaison plus modeste que 20 Mbit/s, peut-être seulement de 1,5 Mbit/s.

ADSL et les normes

Comme tout autre technologie de communication, l'ADSL a besoin des normes. Toutes les technologies évoluent par une phase d'exploration et d'expérimentation (à leurs débuts les voitures et les avions prenaient beaucoup de tailles et de formes bizarres). Avant que des consommateurs acceptent une nouvelle technologie, celle-cidoit devenir suffisament normalisée pour satisfaire tout le monde. Les gens veulent que la technologie et les produits soient homogènes, indépendants d'un fabricant particulier, et compatibles avec d'autres produits dans la même catégorie.

Aux Etats-Unis, la norme pour l'exploitation de la couche physique ADSL a tout d'abord été décrite par l'organisme de normalisation américain (ANSI) T1.413-1995. Ce document spécifie exactement comment un équipe-

ment ADSL communique sur une ligne locale auparavant analogique. Le document ne écrit pas l'architecture du réseau et des services, ni même le fonctionnement interne d'un noeud d'accès ADSL. Il définie seulement le code de ligne (comment les bits sont envoyés sur le support physique) et la structure des trames (comment les bits sont organisés) sur la ligne.

Dans les produits ADSL on utilise plusieurs techniques de codage de ligne:

- la modulation d'amplitude et de phase sans porteuse - CAP (*carrierless amplitude/phase modulation*),
- la modulation d'amplitude en quadrature (MAQ),
- la technologie de multi-tonalité discrète - DMT (*discrete multitone*).

La gamme de fréquences doit être divisée en bandes en amont et en aval (multiplexage en fréquence). Si les mêmes fréquences sont utilisées en aval et en amont l'annulation d'écho doit être utilisée. L'annulation d'écho élimine la possibilité qu'un signal reflété dans une direction soit interprété comme un «émetteur» dans le sens opposé.

Dans ADSL, les deux techniques d'annulation d'écho et de multiplexage en fréquence peuvent être combinées. Dues à la nature asymétrique de largeurs de bande ADSL, les gammes de fréquences peuvent se chevaucher, mais elles ne coïncident pas.

En tout cas, la norme ADSL doit appliquer la technique DMT avec l'annulation d'écho ou de multiplexage en fréquence. Notons que le multiplexage en fréquence est l'approche plus simple à implémenter. L'annulation d'écho est toujours vulnérable aux effets de diaphonie, où un récepteur capte des signaux qui sont transmis sur la bande adjacente.

Le multiplexage en fréquence évite la diaphonie en permettant au récepteur d'ignorer totalement la gamme de fréquences que son propre émetteur envoie sur la ligne. En contre partie, le multiplexage en fréquence diminue la totalité de la bande disponible dans l'un ou l'autre sens de communication. L'annulation d'écho valide les fréquences possibles à utiliser, ce qui maximalise la performance.

FIGURE 102. Bandes passantes ADSL: multiplexage de fréquences

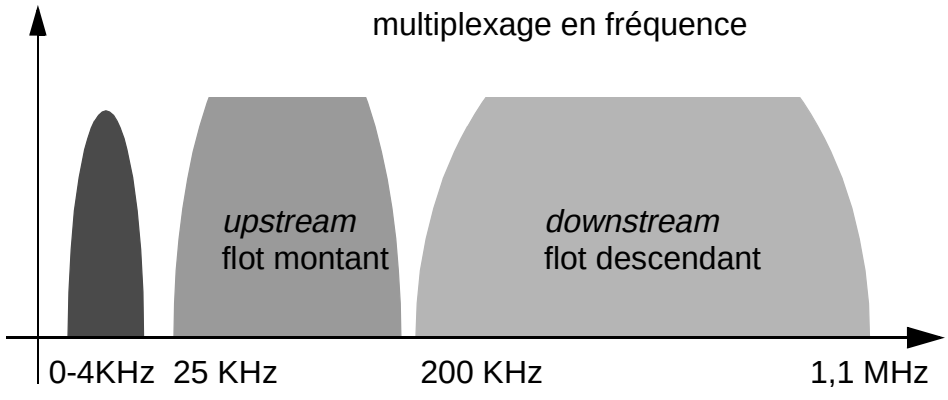
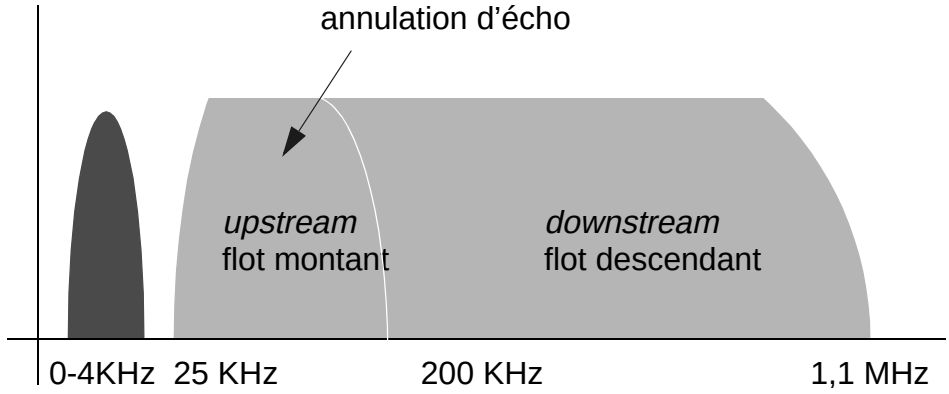


FIGURE 103. Bandes passantes ADSL avec superposition et l'annulation d'écho



CAP et DMT

CAP et DMT sont les codes de ligne les plus utilisés pour ADSL. CAP est une technique de modulation en amplitude et phase; DMT est une technique de modulation multi-tonalité. CAP est très proche de la technique QAM (*Qaudrature Amplitude Modulation*). La technique DMT permet une adaptation de la bande passante aux conditions de la boucle locale à l'instant donné; par contre la modulation CAP est plus simple à implémenter.

Fonctionnement du CAP

Nombre de vendeurs majeurs de l'équipement ADSL proposent le CAP comme le code de ligne. Le code de ligne détermine simplement comment les '0' et 'l' sont envoyés de l'unité ATU-R à l'unité ATU-C. Un code parfait de ligne devrait fonctionner correctement malgré des conditions réelles sur ligne, incluant la présence de bruit, diaphonie, et pertes.

Le CAP est un proche parent du codage connu comme **modulation d'amplitude en quadrature** (MAQ). En fait, mathématiquement ces codes sont presque identiques. La meilleure définition du CAP est probablement - «technique MAQ sans porteuse». On peut considérer le CAP comme une amélioration de la MAQ.

CAP utilise la totalité de la bande passante (sauf le 4KHz pour la voie analogique) de la boucle locale pour le transfert des données. L'opération en duplex est réalisée par le multiplexage en fréquence. Dans le codage de ligne, CAP il n'y a pas de la notion de *subcarriers* ou de sous-canaux.

Fonctionnement du MAQ

Cette section présente quelques détails sur les principes de fonctionnement MAQ. Tous les signaux porteurs analogiques sont caractérisés par l'amplitude, la fréquence, et la phase. Chacun de ces paramètres peut être utilisé pour signaler les '0's et 'l's qui font le contenu de l'information numérique. A titre d'exemple, considérons la phase. La phase peut être utilisée dans un décalage de phase différentiel (*differential phase shift*). Dans ce cas, la phase de l'onde est changée par un angle spécifique à la représentation de chaque baud.

Prenons un exemple simple avec le décalage à deux phases. Si l'émetteur désire émettre un '1', il change simplement la phase de la porteuse (sa phase actuelle) de 180°. Si'l émet un' 0', le décalage de phase introduit est 0° (la phase actuelle de la porteuse n'a pas changé).

A partir d'un simple codage DPSK (*differential phase shift keying*) le décalage de phase en quadrature - QPSK (*quadrature phase shift keying*) peut être compris à deux niveaux. Sur la surface, il est simplement DPSK avec la signalisation de deux bits par un baud. Les décalages de phase sont mesurés à partir de la phase actuelle de l'onde. Dans QPSK, le signal qui est réellement envoyé est la combinaison d'une onde sinusoïdale et d'une onde cosinuoïdale à la fréquence **f**, où **f** est la fréquence de l'onde porteuse.

Parce que les fonctions d'onde **cosinus** et **sinus** sont toujours déphasées de 90°, elles sont dites «dans la quadrature», car ces deux quantités sont perpendiculaires, ou différentes de 90° l'une de l'autre.

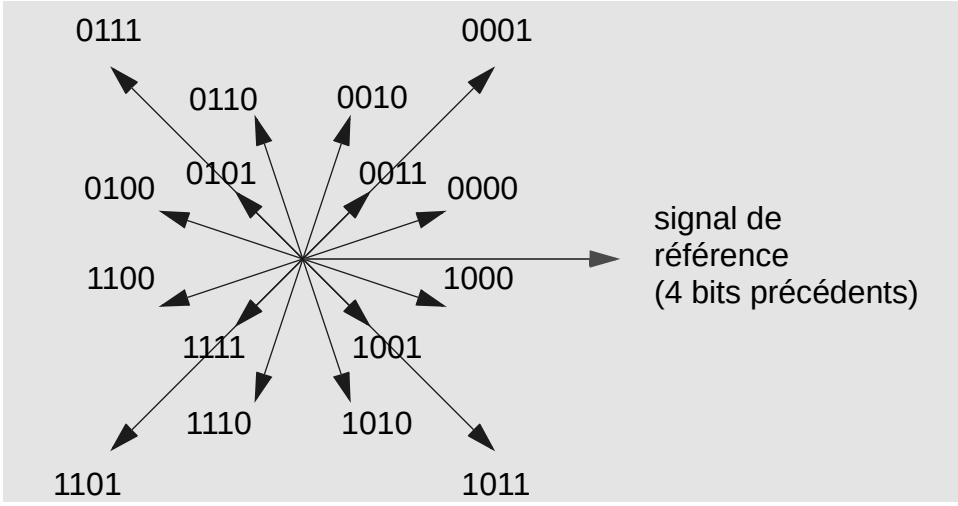
En fait, QPSK n'a réellement rien à faire avec le décalage de phase. Le décalage de phase est une conséquence de la **modulation des amplitudes de deux ondes dans la quadrature**. Si l'on comprend QPSK à ce niveau, le sujet de modulation d'amplitude en quadrature sera simple; mais une vue du QPSK comme «décaleur» des phases est également suffisante.

Le codage MAQ peut être, comme QPSK, visualisé à deux niveaux. Sur la surface, un modulateur MAQ crée 16 signaux différents en introduisant les deux phases et la modulation d'amplitude. En introduisant 12 décalages de phase et deux amplitudes, il est possible d'obtenir 16 types de signal. Ces 16 types du signal peuvent représenter 4 bits par un baud de signalisation.

A un niveau plus profond, MAQ module simplement les amplitudes de deux ondes dans la quadrature (décalées de 90°). Par exemple un MAQ peut appliquer quatre amplitudes différentes pour chacune des deux ondes. Si les quatre amplitudes sont étiquetées de A1 à A4, alors les 16 types différents du signal sont obtenus en utilisant toutes les paires possibles: amplitudes combinées avec des fonctions cosinus et sinus de l'onde.

Par exemple: **$A1*\sin(F_t) + A1*\cos(F_t)$, $A1*\sin(F_t) + A2*\cos(F_t)$,...).** L'ensemble de types crée la caractéristique du code MAQ dite «constellation». Une simple «constellation" 16MAQ est illustrée ci-dessous:

FIGURE 104. Modulation MAQ 4*4 ou 16MAQ - une simple constellation



En général, plus le nombre de niveaux d'amplitude est grand, plus est le nombre de points de constellation est grand et en conséquence plus grands le nombre de bits par le changement du signal (baud). Le codage MAQ est limité au nombre de niveaux qui peuvent être discernés par le récepteur face au bruit.

Par exemple le MAQ/CAP peut être utilisé à livrer l'information numérique de vidéo. Le flot binaire de sortie d'un «numériseur» est cassé en morceaux de 4 bits (*halfbyte* ou *nibble*). Ces morceaux alimentent le codeur qui sélectionne les amplitudes à appliquer sur les ondes dans la quadrature.

La sortie du codeur alimente un modulateur qui crée le signal de sortie. Le modulateur combine en fait les amplitudes appropriées du sinus et cosinus à la fréquence de l'onde porteuse, créant ainsi les décalages d'amplitude et phase associés aux points correspondants de la constellation.

DMT pour ADSL

Les périphériques ADSL (l'ATU C et ATU-R) utilisent les codes MAQ, CAP, et DMT comme codes de ligne. Le code officiel de ligne normalisé pour l'ADSL est DMT. Bien que le DMT soit plus nouveau et puissant que le CAP ou le MAQ, il n'était pas implémenté jusqu'à récemment parce que le MAQ/CAP était suffisant pour les besoins communs de la télécommunication.

La technique DMT divise la capacité entière de largeur de bande en un grand nombre de sous-canaux également espacés. Techniquement, ils sont appelés *subcarriers*, mais plus communément des **sous-canaux**. Au-dessus de la bande de base conservée pour le canal analogique, la totalité de la bande passante est prolongée habituellement à 1,1 MHz et divisée en 256 sous-canaux. Chaque sous-canal occupe 4,3125 KHz, ce qui donne une largeur totale de la bande de 1,104 MHz. Certains sous-canaux sont réservés pour des fonctions spécifiques, d'autres ne sont pas utilisés. Par exemple, le canal #64 à 276 KHz est affecté pour le signal de pilotage.

La majorité des systèmes DMT utilise seulement 249 ou 250 sous-canaux pour l'information. Dans la plupart des cas, les sous-canaux bas #1 à #6 sont affectés à la voix analogique. Parce que 6 multiplié par 4,3125 Hz donne la valeur 2,.875 KHz, il est commun de voir 25 KHZ comme point de départ des services ADSL.

Quand l'annulation d'écho est utilisée il y a 32 canaux montants, démarrant habituellement au canal #7, et 250 canaux descendants, c'est qui donne à ADSL sa bande asymétrique la plus large possible.

Quand on utilise seulement le multiplexage en fréquence, il y a typiquement 32 canaux montants et seulement 218 canaux descendants parce qu'ils ne se chevauchent plus. Les canaux montants occupent la partie basse du spectre pour deux raisons:

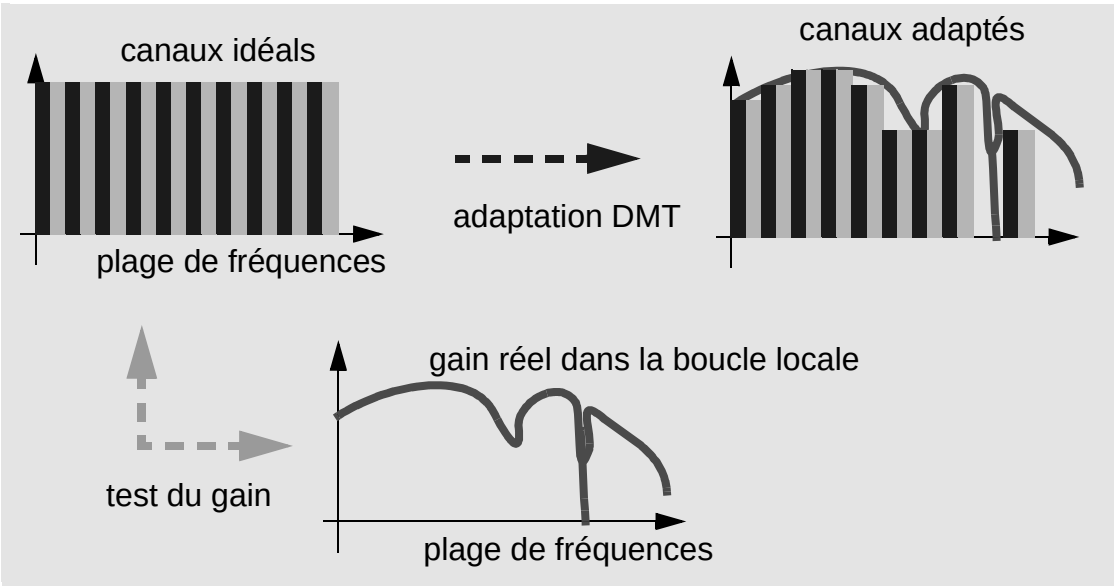
- l'atténuation de signal est moindre sur les fréquences moins élevées,
- les pertes du signal à la fin du spectre sont très importantes et inacceptables pour les signaux de contrôle.

Quand les périphériques ADSL qui emploient DMT sont activés, chacun des sous-canaux est «testé» pour l'atténuation. Dans la pratique, le test est une procédure d'échange de données où le paramètre utilisé est le gain (le contraire de l'atténuation).

Chacun des nombreux sous-canaux emploie le codage basé sur MAQ. La capacité totale de transfert est la somme de tous les débits binaires envoyés sur tous les sous-canaux actifs. De plus, tous les sous-canaux sont constamment contrôlés pour la performance et les erreurs.

Le groupement des sous-canaux individuels DMT donne une granularité de 32 Kbit/s. Autrement dit, un périphérique DMT pourrait fonctionner à 768 Kbit/s ou à 736 Kbit/s (soit 32 Kbit/s moins), dépendant des conditions ambiantes.

FIGURE 105. Fonctionnement du codage - adaptation DMT



Les experts concèdent généralement qu'une granularité plus fine est un bénéfice qui peut maximaliser la performnce de l'installation. Les périphériques DMT peuvent mesurer le gain dans chaque *subcarrier* et le régler au nombre réel de bits par seconde par le canal. Dans certaines conditions, les canaux peuvent être "mis à zéro."

De façon plus générale, le DMT est choisi comme le norme ADSL pour les raisons suivantes:

- l'optimisation incorporée du sous-canal (RADSL),
- le monitorage actif en cours,
- le haut niveau de souplesse pour l'adaptation du débit,
- l'immunité au bruit pour une grande capacité de transfert.

Canaux et trames ADSL

Le transfert des données entre le noeud d'accès ADSL et l'interface de l'abonné est effectué sous la forme de trames. Le flux binaire à l'intérieur des trames peut être décomposé au maximum en 7 canaux de transport (*bearer channels*). Les canaux de transport *simplex* descendants sont de deux types:

- les canaux descendants numérotés par AS0 .. AS3 (maximum 4),
- les canaux montants numérotés LS0 .. LS2 (maximum 3).

Chaque canal de transport peut être programmé pour transférer un nombre multiple de 32 Kbit/s (sous-canaux). Le nombre de multiples de 32 Kbit/s est normalisé via la notion de la **classe de transport**.

Le nombre maximal de sous-canaux et le nombre maximal de canaux de transport dans une configuration dépendent de la classe de transport:

- la classe 2M-1 peut être une combinaison d'un à trois canaux de transport descendants fonctionnant avec le débit de 2,048 Mbit/s: un canal 6,144 Mbit/s, un canal 4,096 Mbit/s et un canal 2,048 Mbit/s, trois canaux 2,048 Mbit/s.
- la classe 2M-2 peut être une combinaison d'un ou deux canaux de transport descendants fonctionnant avec le débit de 2,048 Mbit/s: un canal 4,096 Mbit/s, deux canaux 2,048 Mbit/s.
- la classe 2M-3 est un canal simple de 2,048 Mbit/s (AS0).

Les canaux duplex comprennent un canal obligatoire de contrôle qui porte le débit de 16 ou 64 Kbit/s (LS0) et 1 à 2 canaux de transport LS1 et LS2. Le canal LS1 fonctionne avec le débit de 160 Kbit/s et le canal LS2 avec le débit de 384 Kbit/s ou 576 Kbit/s.

Selon la classe de transport nous avons:

- la classe 2M-1: configuration 1 - LS1 et LS2 160 Kbit/s + 384 Kbit/s; configuration 2 - LS2 à 576 Kbit/s
- la classe 2M-2: configuration 1 - LS1 160 Kbit/s; configuration 2 - LS2 à 384 Kbit/s
- la classe 2M-3: seulement une configuration - LS1 160 Kbit/s

Les canaux bidirectionnels ont une option pour le transfert des cellules ATM dans un canal du type LS2. Le canal LS2 peut porter des cellules formatées pour le protocole d'adaptation AAL5 (VBR - *variable bit rate*) ainsi que les cellules de AAL1 (CBR - *constant bit rate*).

Le tableau ci-dessous montre différentes options de transport pour l'ensemble des canaux de transport.

TABLEAU 9. Options de canaux de transport

classe de transport	**2M-1**	**2M-2**	**2M-3**
canaux de transport descendants (*downstream*)			
capacité maximale en Mbit/s	6,144	4,096	2,048
options en Mbit/s	2,048	2,048	2,048
	4,096	4,096	
	6,144		
sous-canaux actifs (max)	trois: AS0,AS1,AS2	deux: AS0,AS1	un: AS0
canaux de transport montants (*upstream*)	640	608	176
capacité maximale en Kbit/s	576	-	-
options en Kbit/s	384	384	-
	160	160	160
	C(64)	C(64)	C(16)
sous-canaux actifs (max)	trois: LS0,LS1,LS2	deux: LS0,LS1 ou LS1, LS2	deux: LS0,LS1

Les données de contrôle de l'ADSL (*overhead*)

En plus des canaux de transport, les trames ADSL portent les données de contrôle pour diverses fonctions.

- le canal de synchronisation qui permet d'effectuer la configuration des canaux AS et LS,
- le canal des fonctions internes - EOC (*embedded operations channel*),
- le canal de reconfiguration - OCC (*operations control channel*),
- la correction d'erreurs - CRC (cyclical redundancy check),
- l'opération, administration, maintenance - AOM,
- la correction anticipée d'erreurs.

Ces informations circulent dans les deux sens en aval et en amont. Dans la plupart des cas, les bits d'*overhead* sont envoyés avec un débit de 32 Kbit/s. Dans les canaux de transport plus rapides, les bits de contrôle prennent la bande de 64 Kbit/s voire 128 Kbit/s.

Données lentes et données rapides

La spécification de l'ADSL caractérise deux catégories de données:

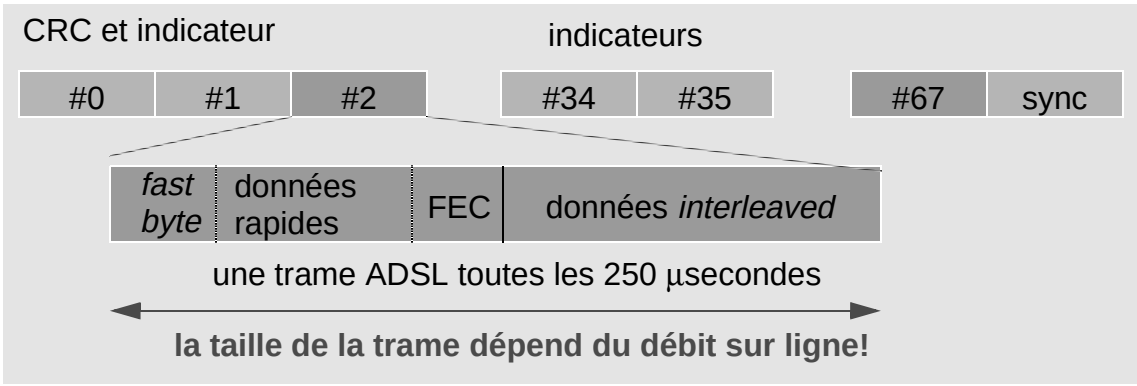
- les données rapides (*fast data*) qui transitent par un tampon de faible latence (*delay sensitive buffer*)
- les données lentes qui transitent par un tampon lent (*interleave data buffer*)

Cette organisation permet d'introduire un simple système de priorité. L'espace de transport dans les trames ADSL est assigné indépendamment pour chaque catégorie de données.

Super-trame ADSL

Les bits organisés en trames sont envoyés sur le lien ADSL sous la forme d'une super-trame. Une super-trame ADSL est composée d'une séquence de 68 trames ADSL. Certaines trames ADSL ont des fonctions spécifiques. Par exemple les trames #0 et #1 contiennent le contrôle d'erreur (CRC) et l'idicateur (IB). Autres indicateurs sont portés dans les trames #34 et #35. A la fin de la super-trame se trouve une trame de synchronisation. Une super-trame est envoyée toutes les 17 millisecondes. Une trame simple ADSL est donc envoyée à la cadence de 4000 par seconde ou une trame toutes les 250 µsecondes. Une trame simple est composée d'une partie «rapide» où se trouvent des données rapides et d'une partie «lente» où sont enregistrées des données non-rapides. La partie «rapide» est protégée par un code FEC *(forward error correction*).

FIGURE 106. Super trame ADSL



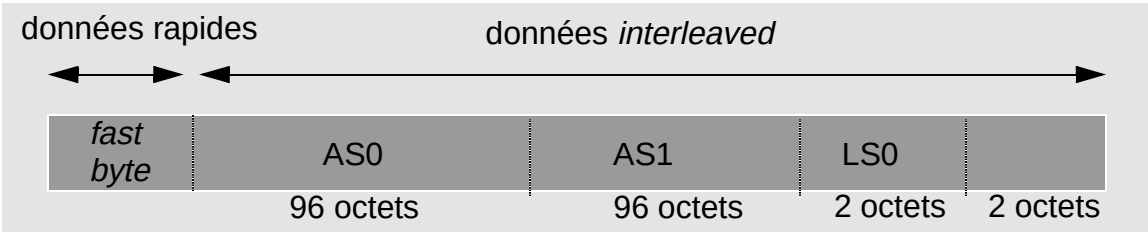
Les trames «ordinaires» portent également des indicateurs de contrôle pour les fonctions supplémentaires (*embedded operations*) et la synchronisation. Quatre fonctions supplémentaires ont été définies:

- indicateur de détection d'erreur,
- indicateurs OAM: opération, administration, maintenance.

Ces indicateurs sont insérés sous la forme d'octets dans l'en-tête des trames ADSL.

Pour terminer la présentation de la structure des trames ADSL prenons un exemple. Considérons le transport de classe 2M-1 avec les canaux de transport AS0, AS1, et AS2, chacun envoyant 64 octets dans chaque trame ADSL. Nous avons donc trois canaux de transport descendants avec le débit de 64 octets* 8 bits/octet * 4000/ seconde= 2,048 Mbit/s. Dans cette configuration, le canal LS0 montant transmet les données dans les deux sens avec un débit de 2 octets* 8 bits/octet * 4000/second = 64 Kbit/s.

FIGURE 107. Structure d'une trame ADSL pour la classe de transport 2M-1

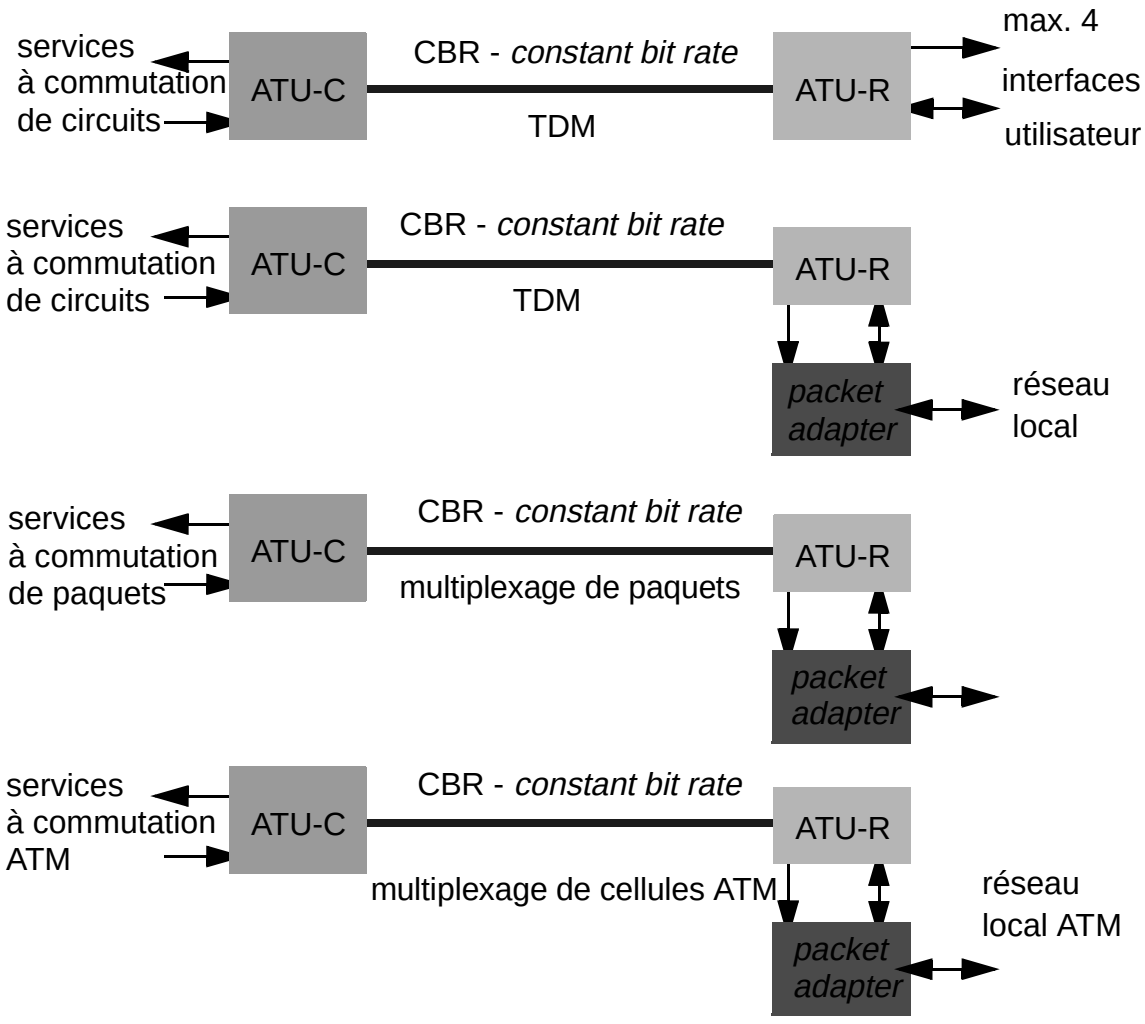


Les modes de distribution ADSL

Un lien ADSL ressemble à une connexion par modem à 56Kbit/s, sauf qu'il offre un débit allant à 6,144 Mbit/s. Un circuit ADSL offre un débit constant (CBR). Les super-trames ADSL sont envoyées toutes les 17 millisecondes; chacune contient 68 trames simples. Les trames portent des données rapides (audio, vidéo) avec un délai court et stable, et les données moins sensibles aux délais mais sensibles aux erreurs (e.g. pages WEB). Pour caractériser les catégories des données portées dans les trames ADSL, le **Forum ADSL** a défini quatre modes de distribution:

- le mode synchrone au niveau du bit (*bit synchronous mode*),
- le mode synchrone au niveau des paquets (*packet adapter mode*),
- le mode de multiplexage des paquets (*end-to-end packet mode*),
- le mode de multiplexage des cellules ATM (ATM mode).

FIGURE 108. Modes de distrubution ADSL



Dans le **mode synchrone** au niveau du bit, le noeud d'accès récupère les suites de bits qui arrivent dans les canaux de transport C ou LS. Le noeud d'accès envoie les bits descendants sur les canaux AS.

Dans le **mode paquet** le site utilisateur *(user premises*) contient un bloc d'adaptation qui permet aux périphériques utilisateur d'envoyer et recevoir les paquets à la place d'une chaîne binaire non structurée. Cette solution permet d'attacher à l'adaptateur un ensemble de dispositifs fonctionnant en mode paquet (p.e. réseau Ethernet).

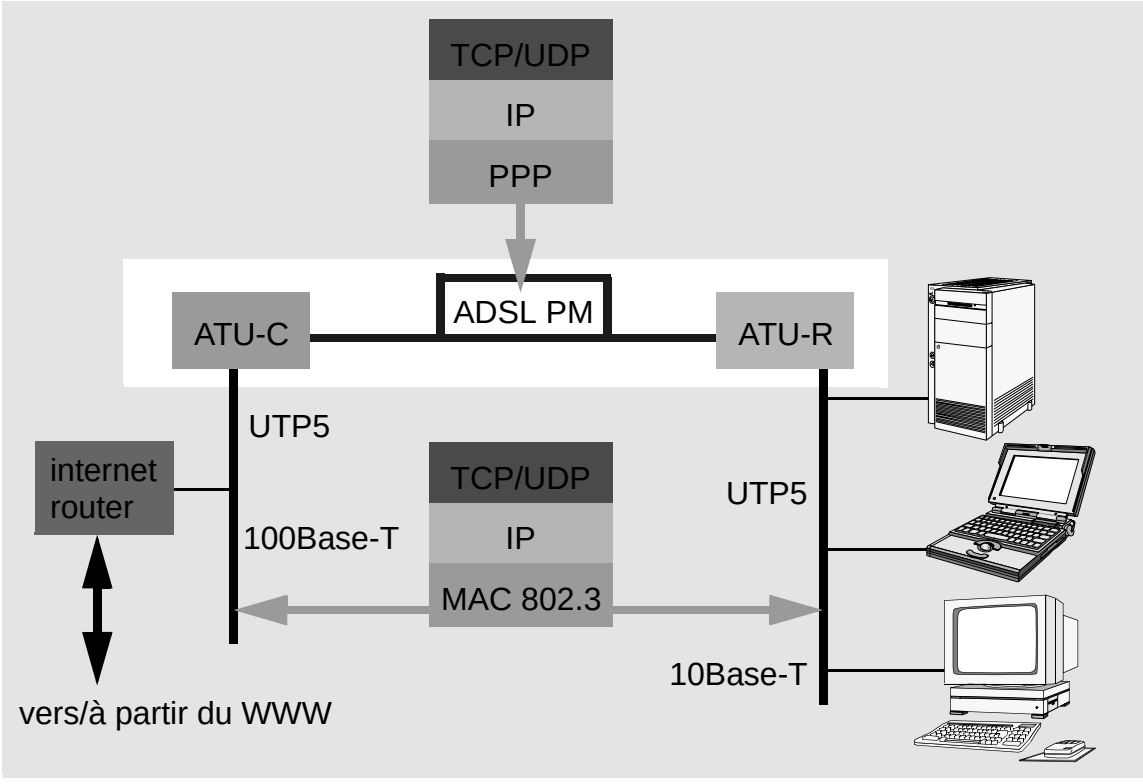
Le **mode paquet de bout-en-bout** (*end-to-end*) est une simple extension du mode précédent. Dans ce mode on suppose que le noeud d'accès gère les chaînes binaires également au niveau des paquets.

Finalement, le **mode ATM** est prévu pour un fonctionnement ATM de bout en bout. Le noeud d'accès passe les cellules ATM au réseau ATM. Les cellules ATM peuvent porter des datagrammes IP ou trames PPP.

Liens ADSL et Internet

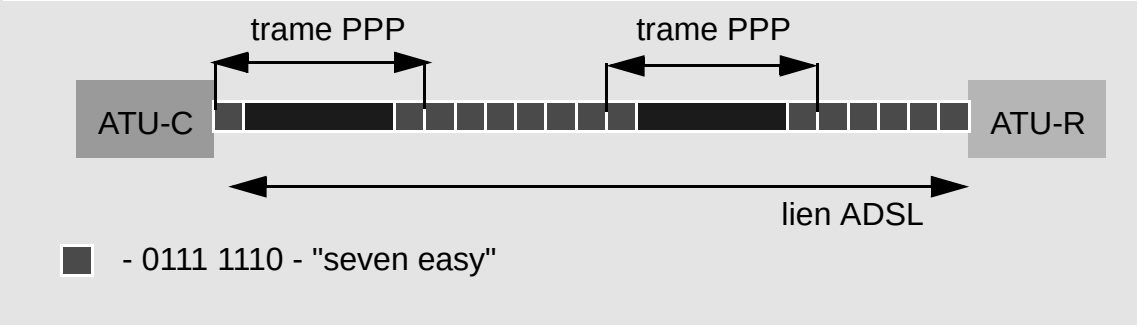
L'Internet et WEB semblent être les applications essentielles et privilégiées pour les connexions ADSL. Ces connexions peuvent fonctionner en mode paquet de bout-en-bout. Dans cette solution tous les services sont accessibles par le biais des protocoles TCP/IP. Sur le schéma ci-dessous, nous voyons une configuration ADSL conçue pour le portage des protocoles TCP/IP.

FIGURE 109. Mode paquet IP/TC de bout-en-bout



Les paquets IP sont portés par les trames PPP. Chaque trame PPP débute et se termine par un fanion (0111 1110 ou **7E** en hexadécimal). L'espace entre les trames PPP est donc rempli par les fanions. Grâce à cette solution, un lien ADSL fonctionnant en mode CBR peut porter les rafales de trames PPP et des datagrammes IP.

FIGURE 110. Insertion des datagrammes IP dans les trames PPP sur un lien ADSL



Lignes ADSL en pratique

Les débits offerts par un lien ADSL permettent d'envisager l'implémentation des services audio et vidéo et leurs accès à la maison. Ces services incluent vidéo à la demande, enseignement à distance (téléenseignement) , jeux vidéo, services d'information, achat à distance (téléachat), vidéo conférences, et beaucoup d'autres.

Selon la nature de la communication, les rapports entre le débit descendant et le débit montant peuvent varier. Le débit montant pour une communication numérique, qui intègre l'accès à l'Internet et l'accès aux réseaux locaux, nécessite autour de 10% du débit descendant (e.g. 150 Kbit/s montant et 1,5 Mbit/s descendant). Les applications basées sur la vidéo, comme le téléachat, nécessitent beaucoup moins de débit montant (e.g. 64 Kbit/s montant pour 1,5 Mbit/s descendant).

Notons que le service téléphonique analogique occupe seulement 4KHz de la bande passante. L'accès RNIS nécessite les débits de 160 Kbit/s en descendant et en montant.

Les distances sur lesquelles les débits ADSL sont accessibles évoluent avec les générations d'équipements disponibles sur le marche. Un équipement qui fournissait 1,5 Mbit/s sur la distance de 3,5 Km il y a trois ans est actuellement capable de fournir 3 Mbit/s sur la distance de 4,5 Km. La compétition est ouverte.

Installation d'une ligne

Pour installer une ligne ADSL à la maison il faut acquérir l'unité de raccordement (ATU-R). Dans la solution la plus simple l'ATU-R intègre le *splitter* et il suffit de le brancher à la place du téléphone analogique. Ensuite il faut installer les câbles et/ou un réseau pour communiquer avec les équipements numériques. Une autre solution consiste à approcher l'ATU-R à votre ordinateur et sans installation nécessaire du câblage pour les liens numériques.

FIGURE 111. Installation d'une ATU-R avec *splitter*; câblage d'un réseau local

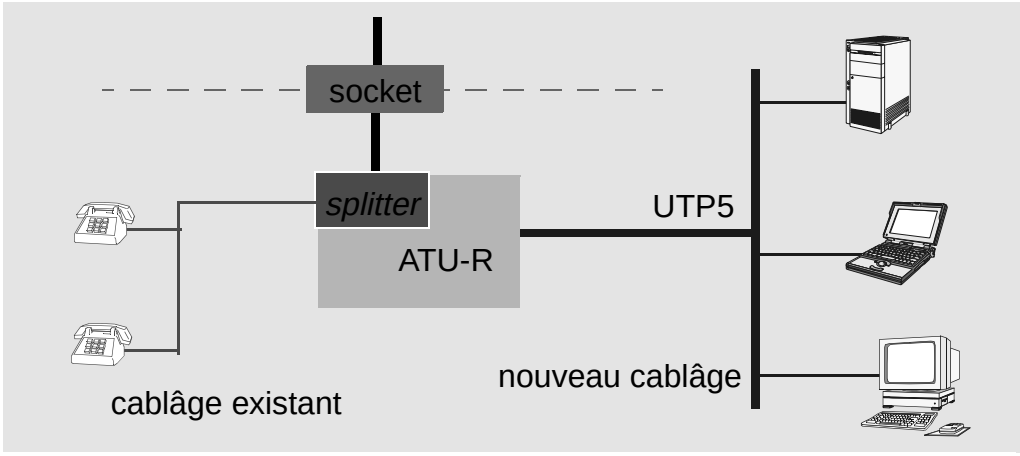
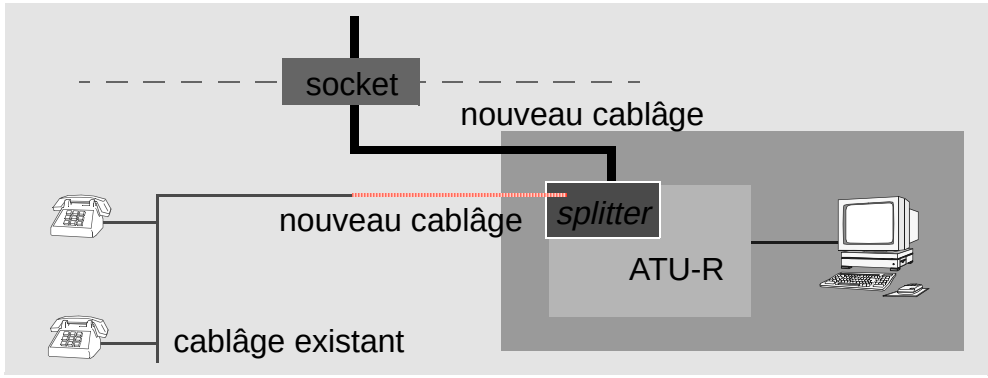
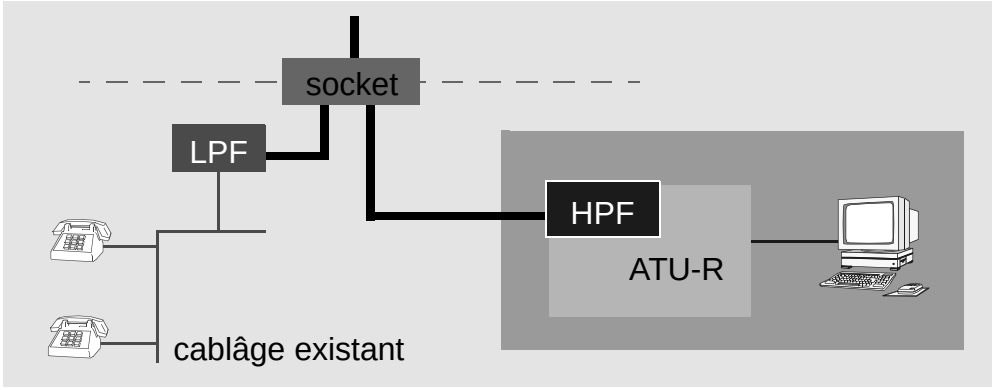


FIGURE 112. Installation d'une ATU-R avec *splitter*: modification du câblage analogique



Les solutions proposées ci-dessus combinent les liens analogiques avec les liens numériques. Pour avoir une installation de meilleure qualité, il est important d'introduire des filtres permettant de découpler les deux sortes de liens.

FIGURE 113. Installation avec des filtres (passe bas - LPF et passe haut - HPF)

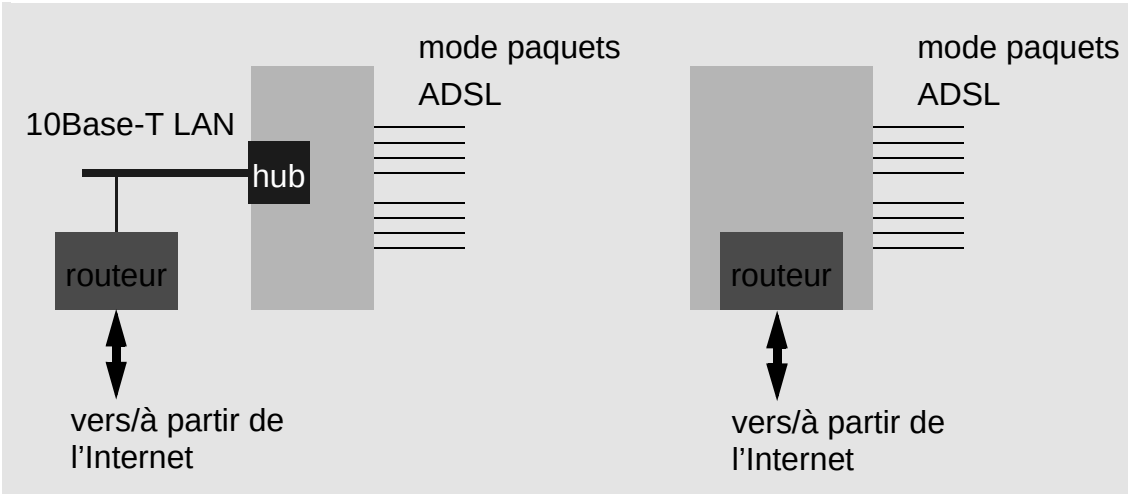


Architecture d'un point d'accès - multiplexeur - DSLAM

Le noeud d'accès - DSLAM (*Digital Subscriber Line Access Multiplexer*) est un élément central dans l'architecture ADSL. Tous les trafics entre les utilisateurs (abonnés) et les serveurs passent par le DSLAM. Un DSLAM est situé typiquement dans le même endroit que le noeud d'accès de la boucle locale - LEC (*Local Exchange Carrier*). Cette solution est possible si le fournisseur de service (e.g. Internet) est également propriétaire du noeud d'accès LEC. Sinon le DSLAM peut être logé dans un local indépendant proche du LEC (quelques centaines de mètres) et connecté par un lien court à haut débit (e.g. 155 Mbit/s).

Pour raison d'efficacité, le simple mode de fonctionnement en multiplexage temporel est remplacé par le fonctionnement en mode paquet avec le multiplexage statistique. Plusieurs équipements DSLAM contiennent les ports Ethernet 10Base-T ou 100Base-T. Ces DSLAMs peuvent être connectés par le biais d'un réseau local à un *hub* ou à un routeur. Une autre solution consiste à intégrer le routeur directement dans le DSLAM.

FIGURE 114. DSLAM avec un routeur internet externe et interne



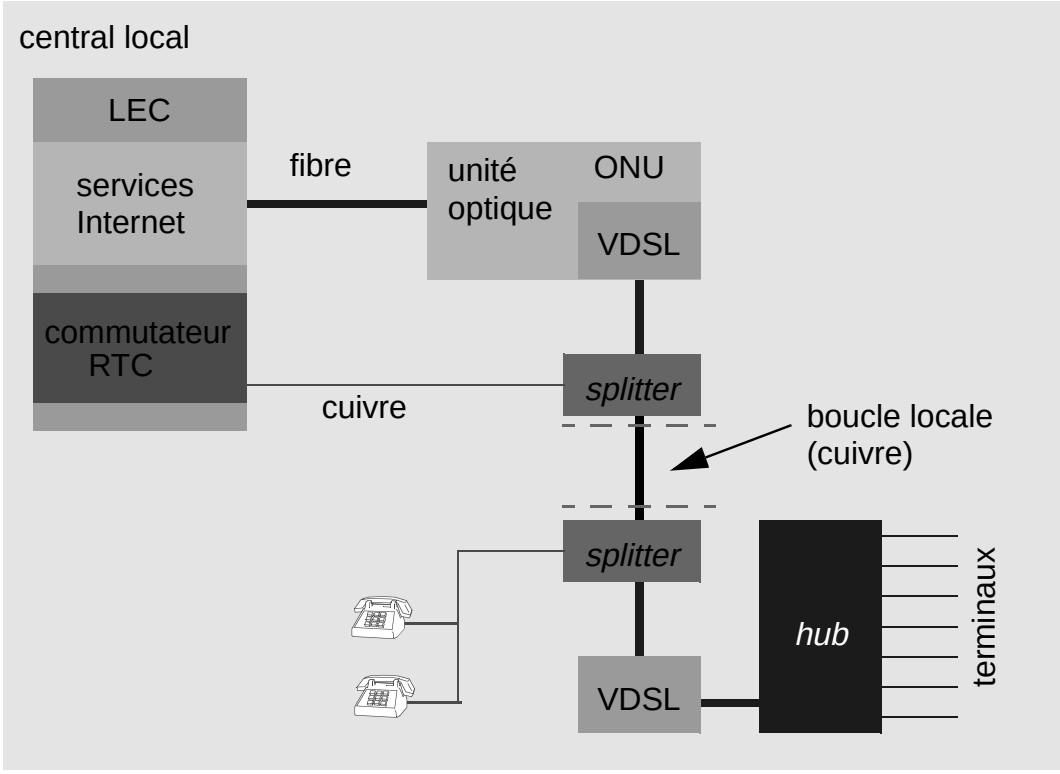
Les routeurs peuvent être connectés à l'ensemble du système Internet par un lien E1 avec le relais de trames. Le service de relais de trames permet de porter les datagrammes IP dans les trames de relais. Un routeur peut être également connecté au réseau ATM préparé pour le portage de datagrammes IP.

ADSL large bande - VDSL (Very High-speed Digital Subscriber Line)

Le débit maximal d'un lien ADSL s'arrête au niveau de 1,5 Mbit/s. Pour dépasser cette limite et arriver au débit de plusieurs dizaines de mégabits par seconde il faut introduire le schéma du VDSL.

L'utilisation du VDSL nécessite l' introduction d'une «rallonge» en fibre optique allant du noeud de raccordement (CO) vers le site d'abonné. Cette «rallonge» ou *fiber feeder* est souvent présente dans les installations modernes du réseau téléphonique.

FIGURE 115. Architecture VDSL



Les débits maximum de communication dépendent de la longueur de la boucle locale.

Le débit descendant sur la distance inférieure à:

- 300 m est 55,2 Mbit/s,
- 1 Km est 27,6 Mbit/s
- 1,5 Km est 13,8 Mbit/s.

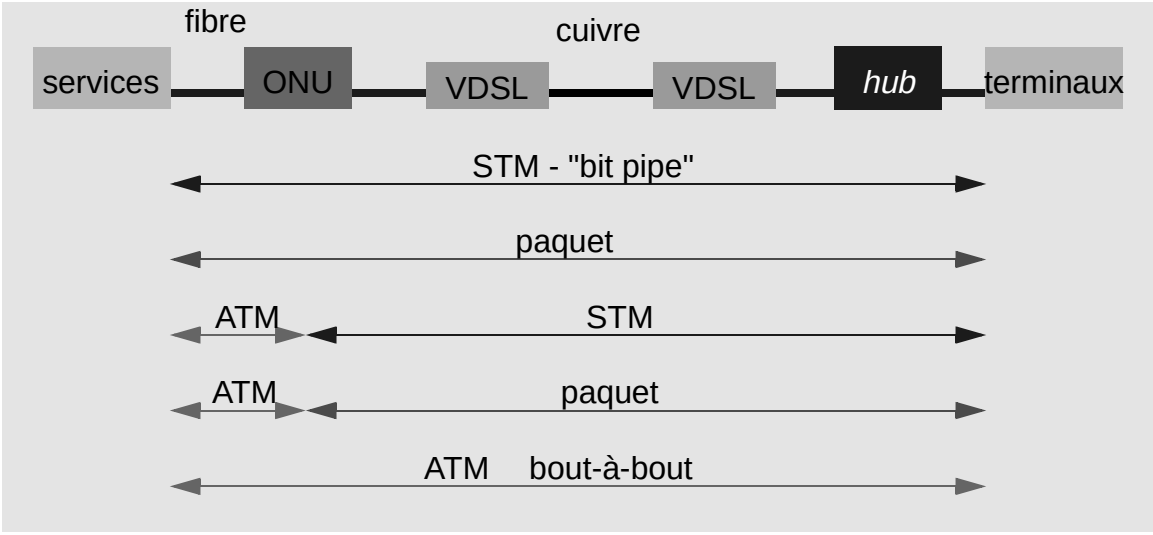
Le débit montant est à partir de 1,5 Mbit/s; selon le cas il peut égaler le débit descendant.

Notons que 55,2 Mbit/s est un débit minimal sur le réseau ATM. Les valeurs de débits proposées pour VDSL correspondent donc aux fractions des débits de communication sur les réseaux ATM.

Les modes de transport VDSL

Le Forum ADSL a défini un certain nombre de modes de transport pour le liens VDSL. La norme VSDL offre le mode bit-synchrone - STM. Dans le mode STM, le lien est une transmission du type CBR. Un lien VSDL peut également fonctionner en mode paquet. Finalement un mode de transfert de cellules ATM est également possible.

FIGURE 116. Modes de transport pour VDSL



Résumé

Dans ce chapitre nous avons étudié la technologie xDSL (*Digital Subscriber Loop*) tout en insistant sur sa principale variante ADSL. Les liens ADSL permettront aux utilisateurs de profiter pleinement des ressources et des services accessibles via l'Internet et *Wide World Web*. Les débits offerts sur un lien ADSL sont au moins dix fois supérieurs aux débits des modems traditionnels. Les liens ADSL, une fois disponibles pour tous les abonnés du réseau téléphonique, permettront de fournir aux utilisateurs un grand éventail des services incluant la vidéo à la demande et le téléenseignement.

CHAPITRE 7

Réseaux WAN

La majorité des réseaux WAN (*Wide Area Networks*) appartient aux organismes publics qui, le plus souvent, gèrent leurs installations au niveau national.

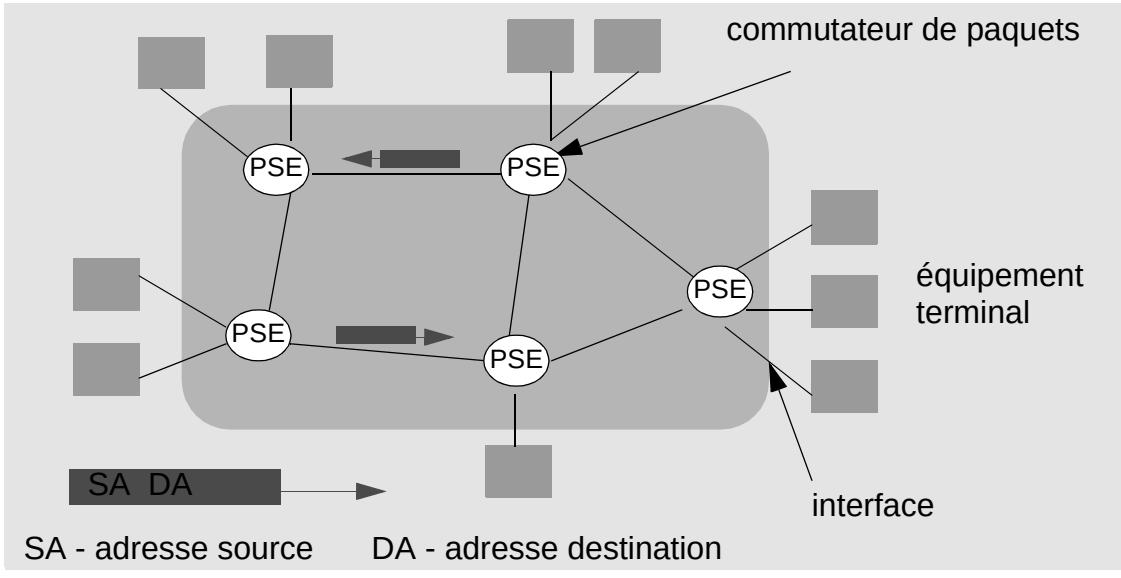
Il y a deux types des réseaux publics WAN:

- les réseaux à commutation de circuits,
- les réseaux à commutation de paquets

La communication par un réseau à commutation de circuits nécessite une phase d'établissement de connexion. La phase d'établissement de connexion prépare un canal physique de communication offrant un débit constant. Dans les réseaux à commutation de paquets, les données peuvent être transmises dès qu'un paquet complet est assemblé par l'émetteur. Chaque paquet contient l'adresse destination et l'adresse source.

Les noeuds de communication (commutation) de paquets stockent et retransmettent des paquets vers la destination demandée. Les paquets en propagation occupent de façon (dynamique/aléatoire) la capacité physique des liens entre les noeuds de communication et les terminaux d'abonnés. Afin d'éviter de longues transmissions avec des délais d'attente importants, la taille d'un paquet est limitée à quelques centaines, voire quelques milliers, d'octets.

FIGURE 117. Architecture simplifiée d'un réseau WAN

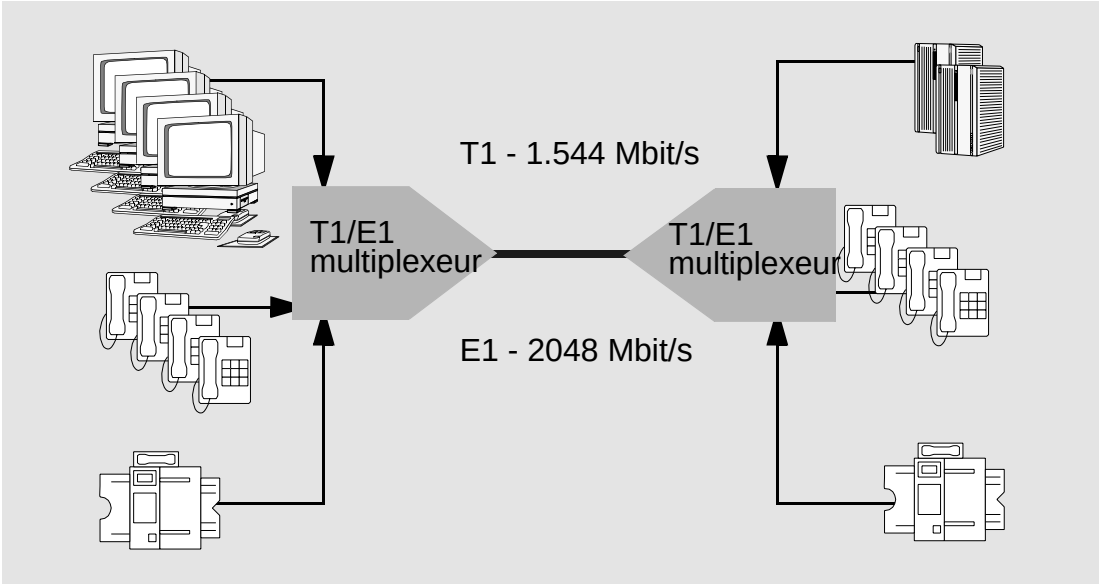


Réseaux - liens T1/E1

Les premiers liens numériques ont été développés dans les années soixante-dix (1977). Initialement un lien T1, utilisant deux paires torsadées permettait de transmettre dans deux sens 1,544 Mbit/s (24 canaux vocaux). Les liens à longue distance utilisent un câble coaxial avec des répéteurs disposés tous les 60 Km. En Europe, le même type de connexion porte un débit de 2,048 Mbit/s - E1 (32 canaux vocaux)

Les liens T1/E1 sont utilisés pour la connexion des réseaux LAN distants - voir figure ci-dessous.

FIGURE 118. Lien T1/E1: interconnexion de réseaux locaux



Réseau Numérique d'Intégration des Services - RNIS

RNIS est devenu un standard depuis 1984 quand CCITT a publié ses spécifications. L'extension des capacités de ce réseau est implémentée sous la forme du réseau SDH (*Synchronous Digital Hierarchy*) ou SONET (*Synchronous Optical NETwork*).

Il y a deux niveaux d'accès au réseau RNIS:

- l'accès de base: *basic rate interface - BRI* ,
- l'accès primaire: *primary rate interface - PRI*.

L'accès de base est composé de deux canaux B offrant chacun un débit de 64 Kbit/s et d'un canal D à 16 Kbit/s. Un flot de 128 Kbit/s peut être porté sur deux canaux de 64 Kbit/s ce qui permet d'interconnecter des réseaux LAN à distance (*remote bridges*).

Un accès primaire offre 30 canaux de 64 Kbit/s et un canal de contrôle à 64Kbit/s.

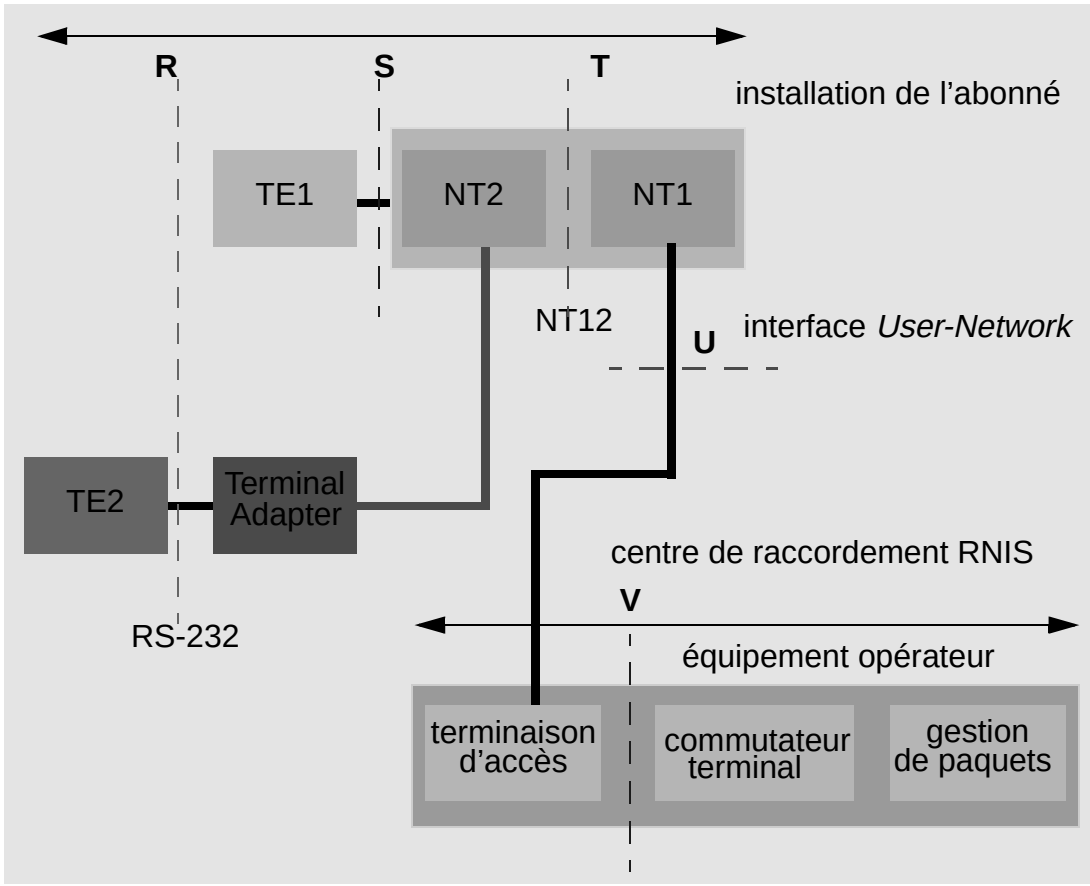
Interfaces RNIS

Le standard RNIS définit un ensemble de dispositifs et d'interfaces. Les dispositifs qui ne sont pas compatibles avec RNIS sont caractérisés comme l'équipement terminal TE2. Un terminal TE2 peut être connecté au système RNIS par le biais d'un adaptateur.

Les terminaux du type TE1 sont compatibles avec le système RNIS (terminaux numériques) et peuvent être connectés au réseau par l'interface S. L'interface NT2 - (*Network Termination* 2) offre les fonctions de commutation, multiplexage de concentration/distribution des données sur le site de l'abonné. L'interface NT1 réalise les fonctions de contrôle, de gestion d'alimentation, de synchronisation. Ces fonctions peuvent être implémentées sur une carte à intégrer dans une station de travail ou dans un PBX.

Le schéma ci-dessous illustre la position de ces interfaces dans le système RNIS.

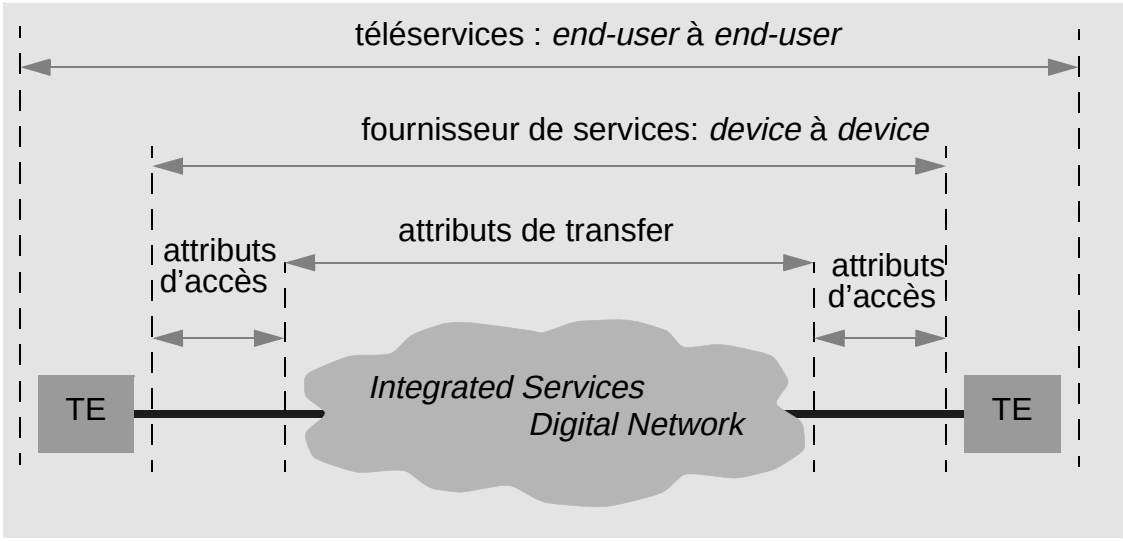
FIGURE 119. Dispositifs et interfaces RNIS



Services RNIS

Différents services RNIS, incluant les services utilisateur, les services à distance et les services de contrôle sont définis en termes d'attributs. Les services utilisateur sont caractérisés par les attributs d'accès, d'information et par les attributs supplémentaires

FIGURE 120. Classification et étendue des services RNIS



Les attributs d'accès spécifient les méthodes d'accès aux fonctions réseau. Les attributs d'information caractérisent les capacités de transmission. Les attributs supplémentaires de service spécifient les délais et les taux d'erreur acceptables pour le service donné. Le tableau ci-dessous donne un ensemble d'attributs de service.

TABLEAU 10. RNIS: les attributs de service

type de service	attribut	description
information transfer attributes	*information transfer mode*	*circuit switched, packet switched*
	information transfer rate	*bit rates in Kbits: 64, 2*64, 384, 1536 (T1), 1920 (E1)*
	information transfer capability	*unrestricted digital, audio in KHz: 3.1, 7, 15; speech, video*
	structure	*service data unit integrity, unstructured 8 KHz integrity*
	establishement of communication	*demand, reserved, permanent*
	symmetry	*unidirectional, bidirectional symmetric, bidirectional asymmetric*
	communication configuration	*point-to-point, multi-point, broadcast*
access attributes	*access channel and rate*	*D (16 Kbit/s), D (64 Kbit/s), B (64 Kbit/s), H0 (384 Kbit/s), H11 (1536 Kbit/s) and H12 (1920 Kbit/s)*
	signalling access and information access	*HDLC, LAPB, LAPD and others*
general attributes	*supplementary services*	*calling line ID, call transfer and others*
	quality of service	*to be defined*
	internetworking	*to be defined*
	operational and commercial	*to be defined*

Couche physique RNIS

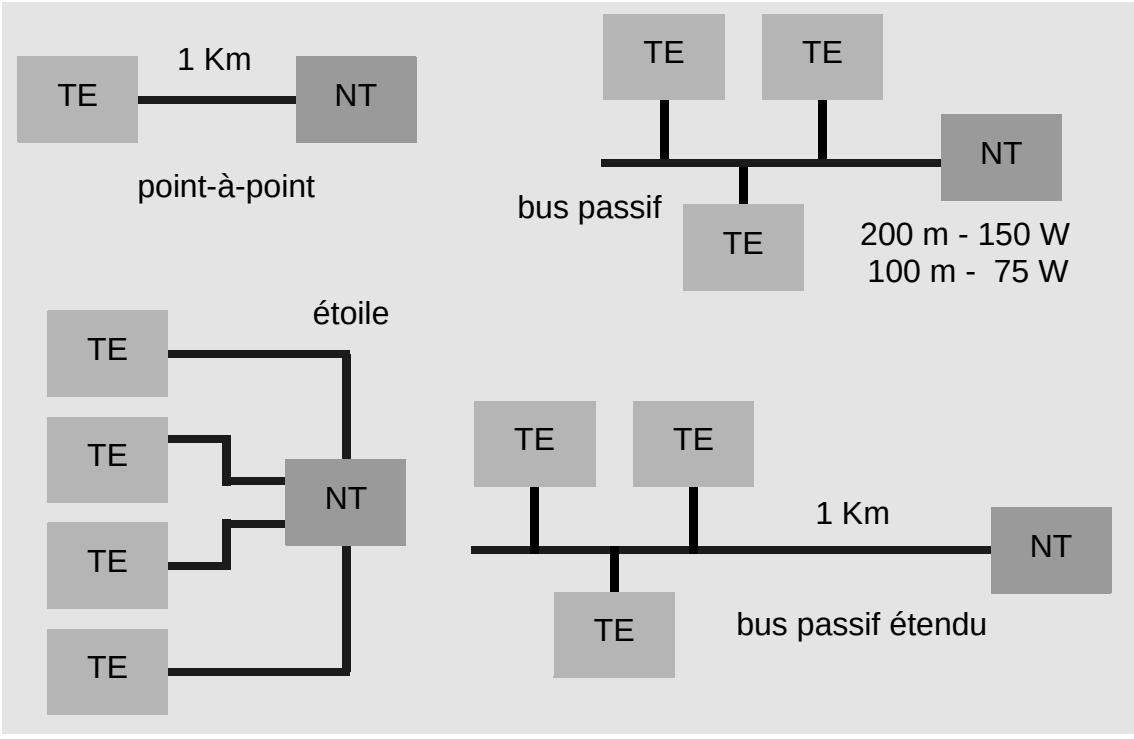
La couche physique fournit les capacités de transfert pour les canaux B et D. Les canaux **2B+D** peuvent être partagés par plusieurs dispositifs. Chaque canal B peut se voir assigné un numéro téléphonique.

Configurations terminales

Le schéma ci-dessous montre quatre configurations possibles pour les équipements terminaux. La première configuration est une simple topologie point-à-point. La distance maximale entre TE (*terminal equipment*) et NE (*network equipment*) est de 1 Km.

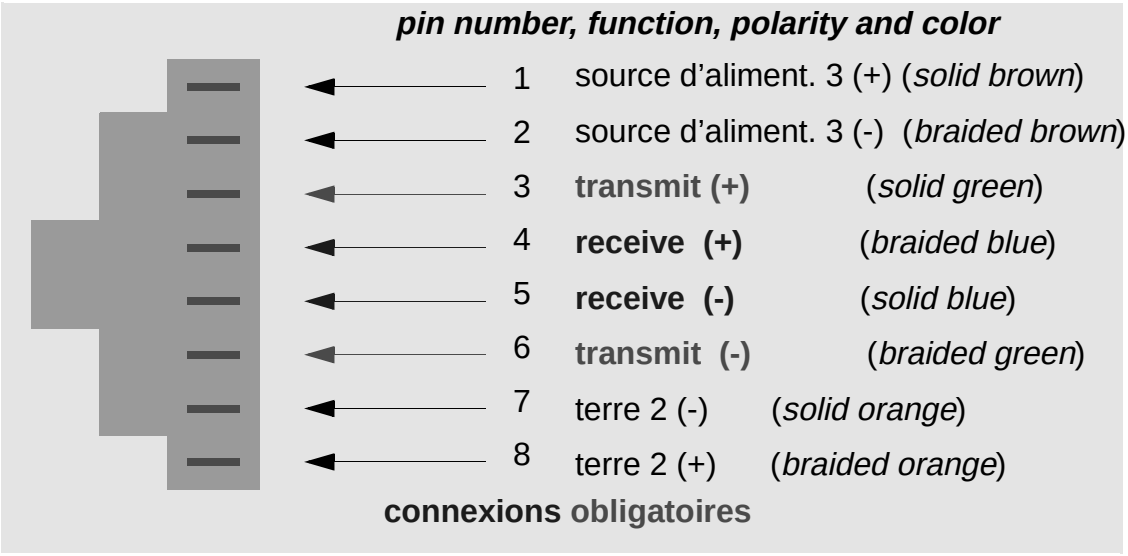
Les configurations suivantes sont des topologies multi-points dans lesquelles peuvent participer au maximum 8 équipement terminaux (e.g. 8 postes téléphoniques). Le contrôle d'accès au bus (2*B) est organisé par le biais du canal D. L'interface S/T utilise deux paires torsadées connectées en parallèle en multi-points.

FIGURE 121. Configurations locales/terminales d'un abonné RNIS



Indépendamment de la configuration, on utilise le même type de connecteur RJ-45. La disposition des fils dans ce connecteur est illustrée dans la figure ci-dessous. Notons que le même connecteur peut être utilisé pour l'accès de base et l'accès primaire.

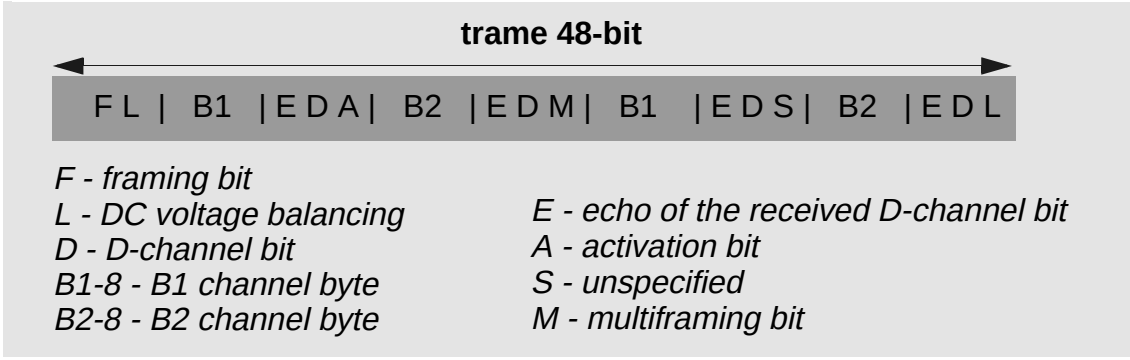
FIGURE 122. **RNIS configuration des fils dans un connecteur RJ-45**



Trames RNIS sur l'interface S/T

Les canaux B et D sont portés par des trames de 48 bits. Chaque trame est composée de deux groupes de 8 bits du canal B1, de deux groupes de 8 bits du canal B2 et de 4 bits du canal D. Une trame est envoyée toutes les 250 microsecondes.

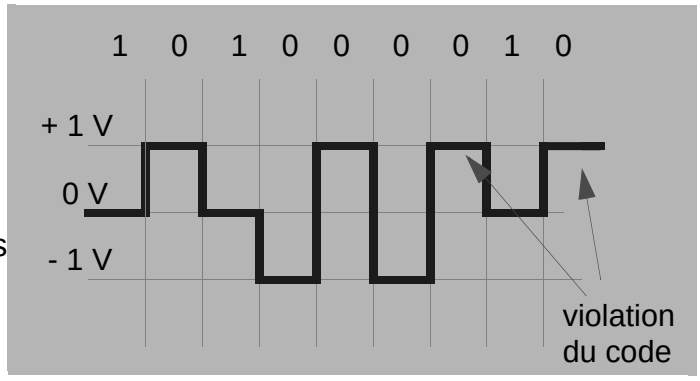
FIGURE 123. **Trame RNIS et son code physique de ligne**



codage *pseudo-ternary*

la polarité change à chaque bit 0,

une violation du code est détectée si 2 bits 0 consécutifs sont portés par le même niveau



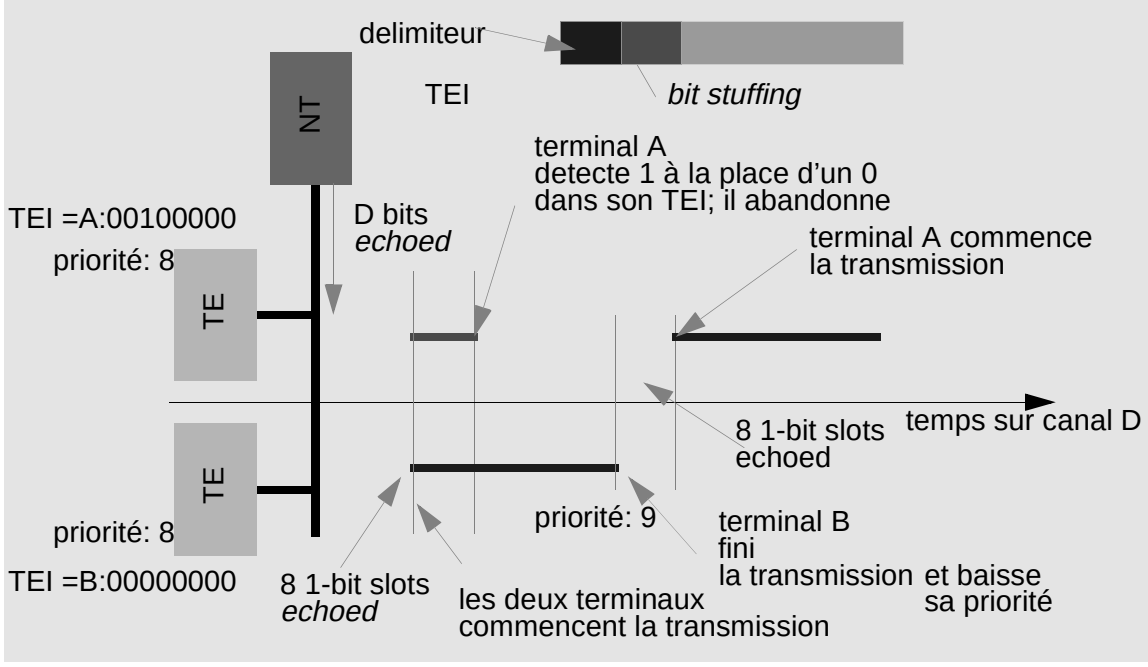
Accès multiple CSMA/CR avec résolution des conflits

Le canal D est utilisé pour la signalisation, les canaux B pour les données. Les terminaux sont connectés sur un bus, le problème est donc comment résoudre le conflit d'accès entre plusieurs utilisateurs du canal D et comment déterminer quel terminal a le droit de transmettre à un instant donné. Les bits D sont envoyés sur le canal D par le terminal TE et par le terminal du réseau NT. Les bits E sont renvoyés (*echoed*) avec le contenu des derniers bits D envoyés par le terminal TE. Cette information permet au terminal TE de savoir à qui appartient la dernière trame envoyée. Si un terminal émet un '0' pour un bit de D, et un autre terminal envoie le bit '1' à cet instant, le terminal réseau NT reçoit le bit '0' car un'1' ne porte pas d'énergie dans le code ternaire.

Le canal D est utilisé pour signaler deux niveaux de priorité. Les valeurs 8 (*normal*) et 9 (*lower*) sont attribuées à l'information de signalisation, les valeurs 10 (*normal*) et 11(*lower*) sont attribuées aux données. Un terminal qui commence la transmission obtient la priorité normale (*normal*). Selon la classe et le niveau de la priorité, le terminal doit compter le nombre de '1' reçus de la part du NT dans le canal avant de commencer la transmission. Après avoir fini la transmission, le terminal baisse son niveau de priorité. Dans une demande suivante, il sera en mode bas (*lower*). S'il arrive à compter jusqu'à la valeur de sa priorité, il peut à nouveau augmenter la priorité et émettre.

L'exemple suivant montre le fonctionnement du protocole CSMA/CR (*carrier sense multiple access with collision resolution*) appliqué à deux terminaux. Les terminaux sont identifiés par TEIs (*terminal end-point identifier*) A:00100000 et B:00000000 (32 et 0), respectivement pour le terminal A et le terminal B. Les deux terminaux envoient les trames avec priorité 8 (données de contrôle). Les deux terminaux transmettent le'1' (0 V) sur le canal D, et le NT répond par'1' sur le canal E (écho). Les terminaux comptent 8 '1's successifs et commencent à émettre. Les champs - drapeaux sont les mêmes, donc ils reçoivent la même réponse sur le canal E. Mais après les drapeaux il y a un bit différent et son écho est égal à '0'. Le terminal A constate donc que son bit '1' n'est pas confirmé et il arrête l'émission. Le terminal B continue la transmission. Le terminal B termine l'émission et baisse sa priorité à 9. Ceci permet au terminal A de compter à 8 et de recommencer la transmission.

FIGURE 124. Protocole CSMA/CR pour deux terminaux



Interface primaire PRI (*Primary Rate Interface*)

L'interface primaire RNIS permet d'obtenir le débit de 2,048 Mbit/s - E1 en Europe et de 1,544 Mbit/s - T1 (Etats Unis).Elle sert essentiellement à interconnecter des réseaux LAN distants.

Adressage RNIS

L'adressage RNIS est composé de trois parties de tailles variables:

- le code du pays -max 3 chiffres,
- code national -max 17 chiffres,
- code local - max 40 chiffres.

SDH et SONET

SDH - *Synchronous Digital Hierarchy* est un réseau de télécommunication implémenté pour le transfert des signaux numériques par le biais de la commutation des circuits et multiplexage synchrone. SDH est le seul standard pour la télécommunication à haut débit sur fibre optique.

Les réseaux standard SDH sont utilisés pour le transfert de données ATM et RNIS large bande.

SONET (*Synchronous Optical NETwork*) est un réseau SDH développé et implémenté aux Etats Unis.

Le tableau ci-dessous montre les interfaces et les débits standard offerts sur le réseau SDH/SONET.

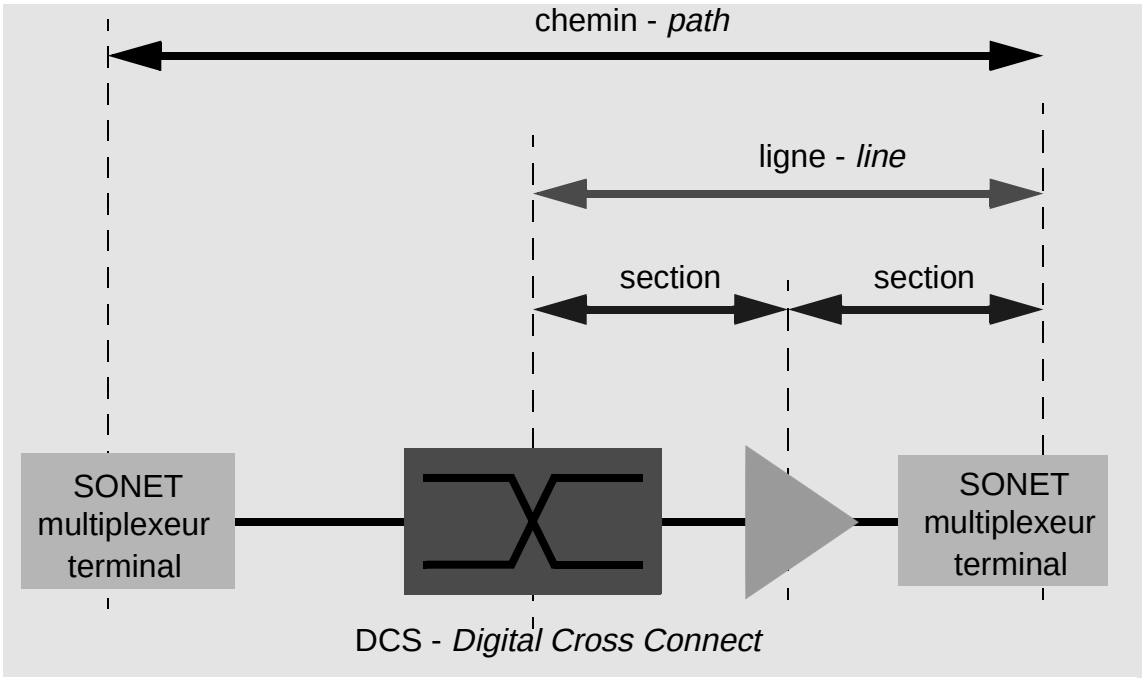
TABLEAU 11. SDH ou SONET: débit/interfaces standard

Synchronous Transport Signal	Optical Carrier designations	débit de ligne
STS-1	OC-1	51.84
STS-3	OC-3	155.52
STS-12	OC-12	622.08
STS-48	OC-48	2488.32

Dans un système SDH il y a trois types de dispositifs:

- les multiplexeurs terminaux - points d'accès terminaux,
- les dispositifs de commutation DCS (*Digital Cross-connect System*), lesquels réalisent la commutation directe synchrone,
- les répéteurs qui permettent de régénérer les signaux optiques et d'effectuer la détection d'erreurs; les répéteurs peuvent être disposés tous les 50 Km.

FIGURE 125. **SDH: niveaux de transport - section, lien et chemin**



Selon le type de dispositifs intégrés dans une topologie il y a trois sortes d'interconnexions:

- **une section** dans le système SDH implémente trois fonctions: création de trames (*framing*), brouillage (*scrambling*) et localisation d'erreurs.
- **une ligne** implémente multiplexage, synchronisation, commutation et interconnexion des signaux SDH.
- **un chemin** (*path*) représente les liens de bout-en-bout entre les applications utilisateur.

SDH - support du transport

Dans une trame STS-1 il y a 90 colonnes et 9 rangées - au total 810 octets. Une telle trame est décomposée en une en-tête et l'enveloppe du contenu SPE (*synchronous payload envelope*). L'en-tête et le SPE sont composés respectivement de 3 et de 87 colonnes.

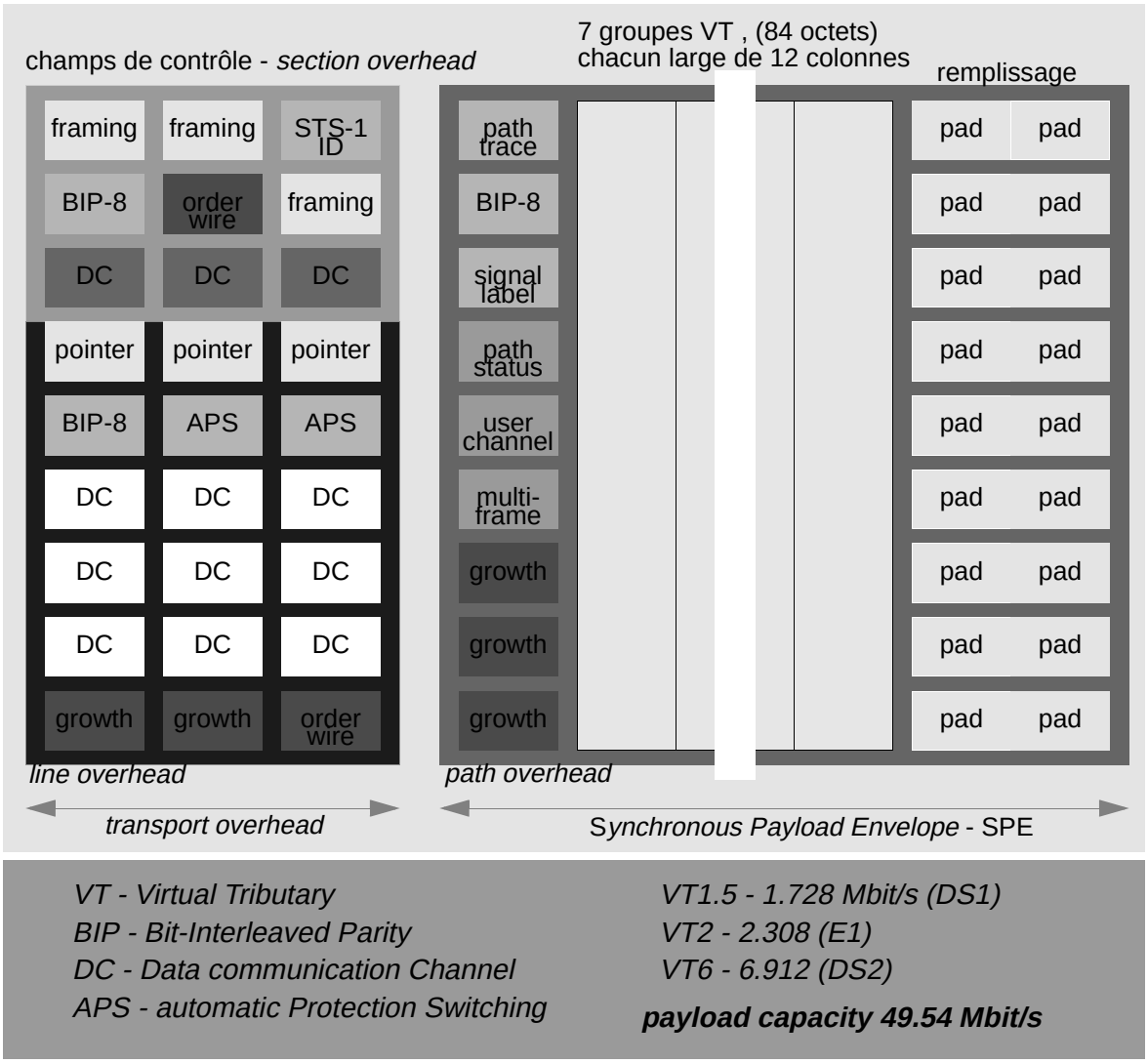
Le transfert d'une trame nécessite 125 microsecondes; 8000 trames sont envoyées dans chaque seconde.

L'en-tête est subdivisée en partie section et partie ligne. L'enveloppe SPE est décomposée en champs indicateur du chemin (*path overhead*) et charge utile (*payload*).

Le champ DC (*Data communication Channel*) est utilisé pour la gestion du réseau. La charge utile du SPE est décomposée en sept groupes VT (*Virtual Tributary*) et 18 octets d'emballage. Chaque groupe VT est étalé sur 12 colonnes et peut contenir un ou deux VT du même type.

Un type VT représente une donnée de taille fixe correspondant au débit alloué à l'utilisateur. Par exemple, un VT2 est utilisé pour créer un canal E1 (2,048 Mbit/s). La totalité d'un SPE peut être exploitée pour l'obtention d'un canal DS3 (51.84 Mbit/s), dans ce cas les VTs ne sont pas utilisés.

FIGURE 126. SDH: format d'une trame STS-1



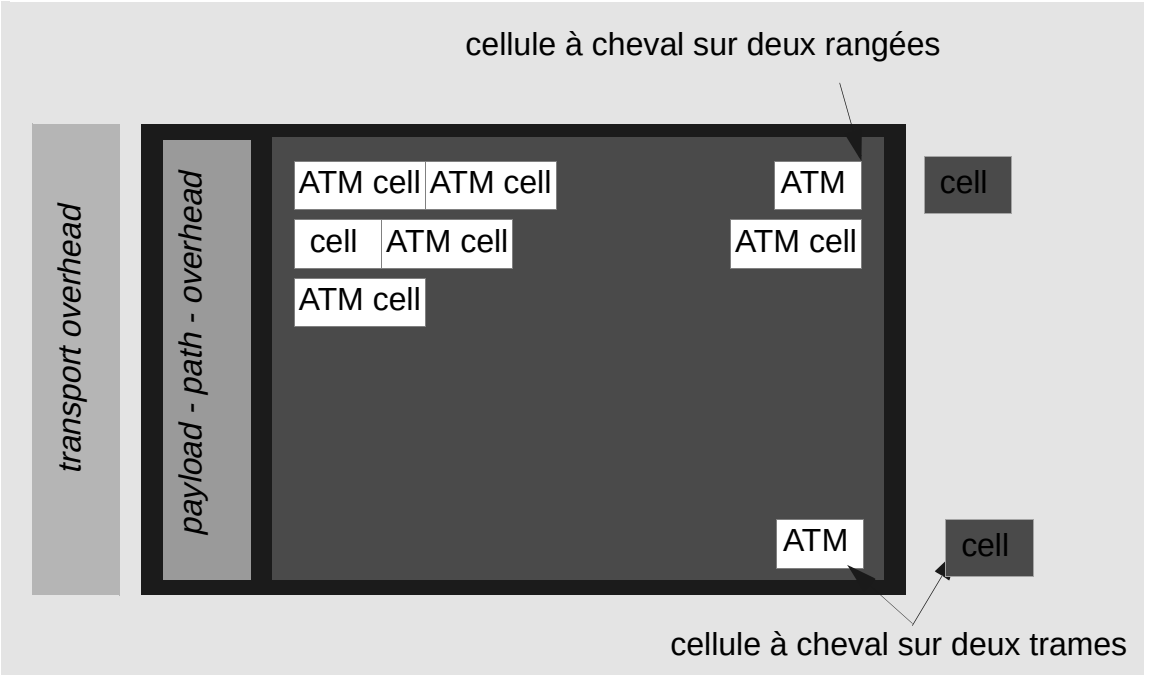
SDH et ATM

La hiérarchie digitale synchrone a été conçue pour les systèmes téléphoniques à longue distance mais elle peut être facilement exploitée par les réseaux de transfert de données. Une projection spécifique a été développée pour le transfert des cellules ATM avec un débit de 155,52 Mbit/s.

Dans le schéma d'encapsulation, chaque trame SDH contient une colonne POH (*payload overhead*), suivie par les cellules ATM logées dans les rangées de la charge utile.

Etant donné que la taille d'une trame SDH n'est pas un multiple de 53 octets, un mécanisme d'alignement a été ajouté afin d'aligner les cellules sur le cadre d'une trame SDH. Les cellules individuelles sont reconnues dans une suite binaire grâce au mécanisme du cadrage et à la vérification des champs HEC dans les en-têtes des cellules ATM.

FIGURE 127. Cellules ATM dans une trame (e.g. OC-3)



X.25 et relais de trames

La majorité des sociétés de télécommunications offre des services à commutation de paquets. Le service du type X.25 appelé **Transpack** est disponible en France depuis des années 70. Le réseau X.25 utilise la commutation des paquets et les circuits virtuels.

En réalité X.25 est un ensemble de protocoles qui couvre trois niveaux:

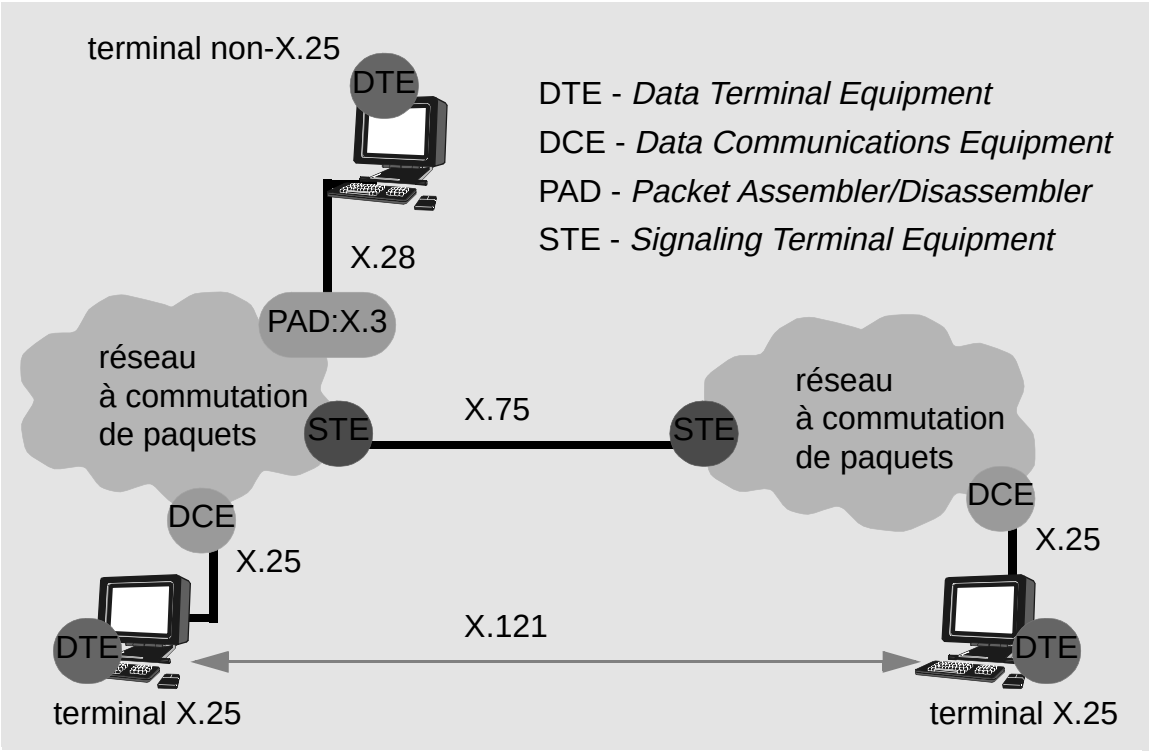
- niveaux physique,
- niveau logique (*link layer*),
- niveau de réseau (*packet network layer*).

Dans la plupart des cas, un équipement spécifique X.25 doit être utilisé pour connecter l'équipement terminal au réseau X.25 - PAD (*packet assembler/dissasembler*).

Les réseaux X.25 peuvent être interconnectés avec d'autres types des réseaux par le biais de passerelles (*gateways*).

Un plan d'adressage X.25 peut impliquer au maximum 14 chiffres plus un '0' ou un '1' optionnel.

FIGURE 128. **Réseau X.25 et ses protocoles**



Protocole de lien de X.25

Le protocole de lien SDLC (*Synchronous Data Link Control)* permet d'établir un lien logique sur le support physique. Le protocole SDLC gère trois sortes de trames:

- les trames de données,
- les trames de contrôle,
- les trames non numérotées.

Le lien fonctionne en trois modes opérationnels:

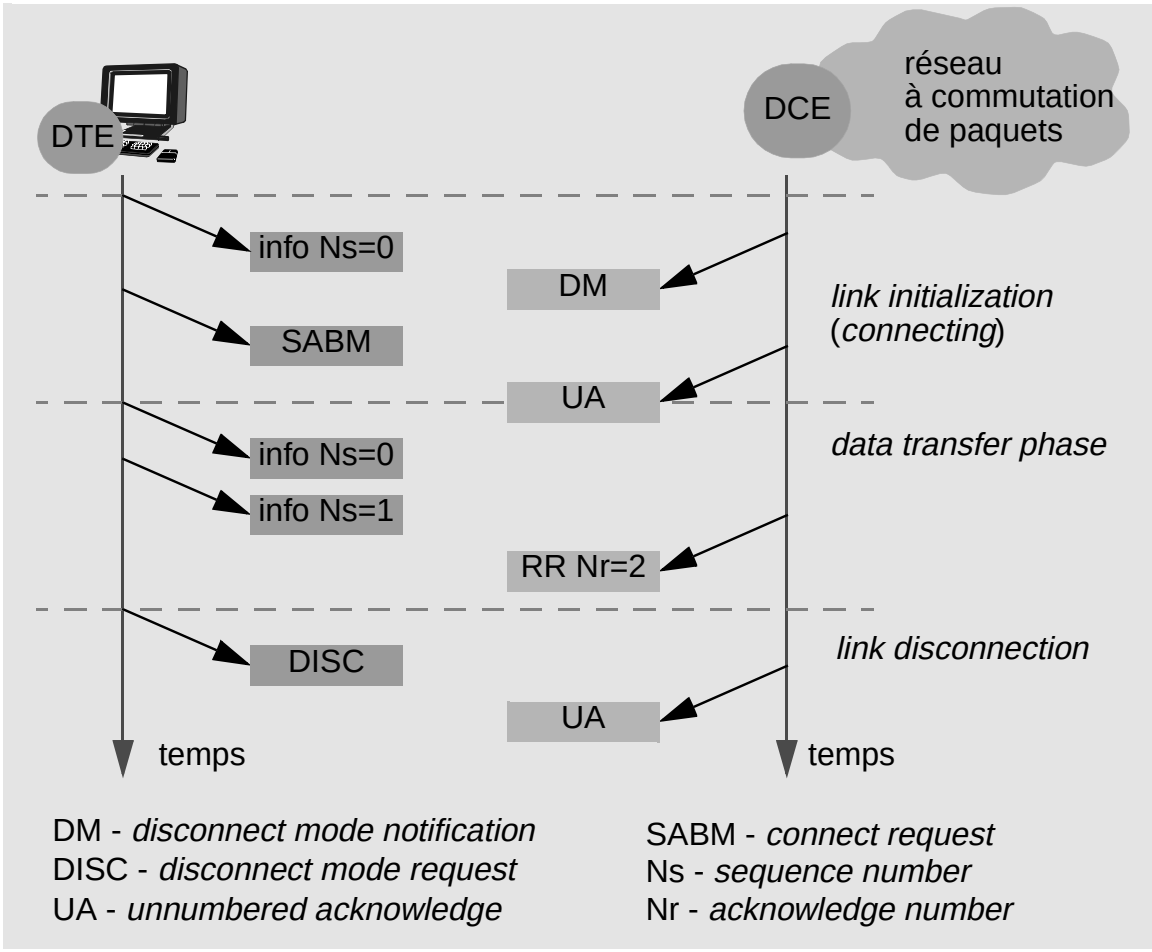
- mode normal - SNRM (*normal response mode*), avec la scrutation et la sélection (*polling* et *selecting*),
- mode asynchrone - SARM (*asynchronous response mode*), fonctionnant en *half-duplex* et point-à-point,
- mode asynchrone équilibré - SABM (*asynchronous balance mode*) fonctionnant en f*ull-duplex* et point-à-point.

Le mode SABM est utilisé pour établir/libérer une connexion.

Le schéma ci-dessous illustre le fonctionnent d'une connexion X.25 et les trois phases opérationnelles:

- la phase de connexion,
- la phase de transfert des données,
- la phase de libération de la connexion.

FIGURE 129. **Trois phases de fonctionnement d'une connexion X.25**



Les trames/paquets X.25

Les trames X.25 sont très typiques des réseaux numériques. Une trame X.25 commence par un fanion (7E) et se termine par le même fanion (7E). Les données du niveau réseau sont encapsulées dans les paquets X.25 et insérées dans les trames physiques. Un paquet X.25 contient:

- le numéro du lien logique LCI (*logical channel identifier*) - max 4096,
- l'octet de contrôle,
- le champ de données,
- la somme de test (FCS - *frame check sequence*),

Chaque noeud de communication maintient une table de correspondances entre les adresses de destination et des numéros logiques des liens. La valeur du LCI peut être différente sur les liens physiques consécutifs.

FIGURE 130. X.25: couches: réseau, lien logique, et trame physique

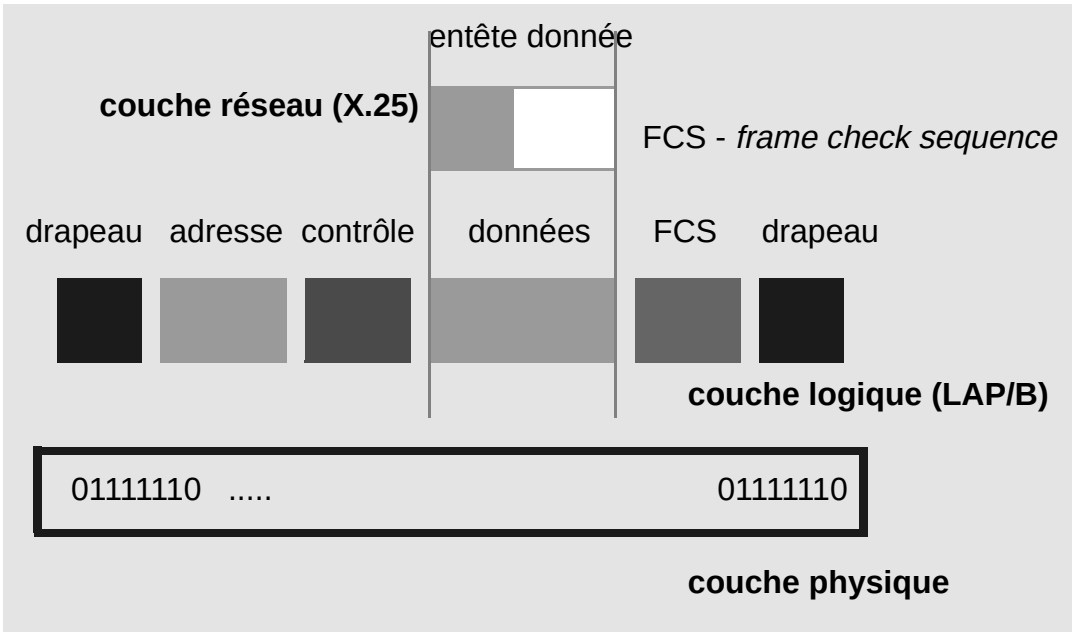
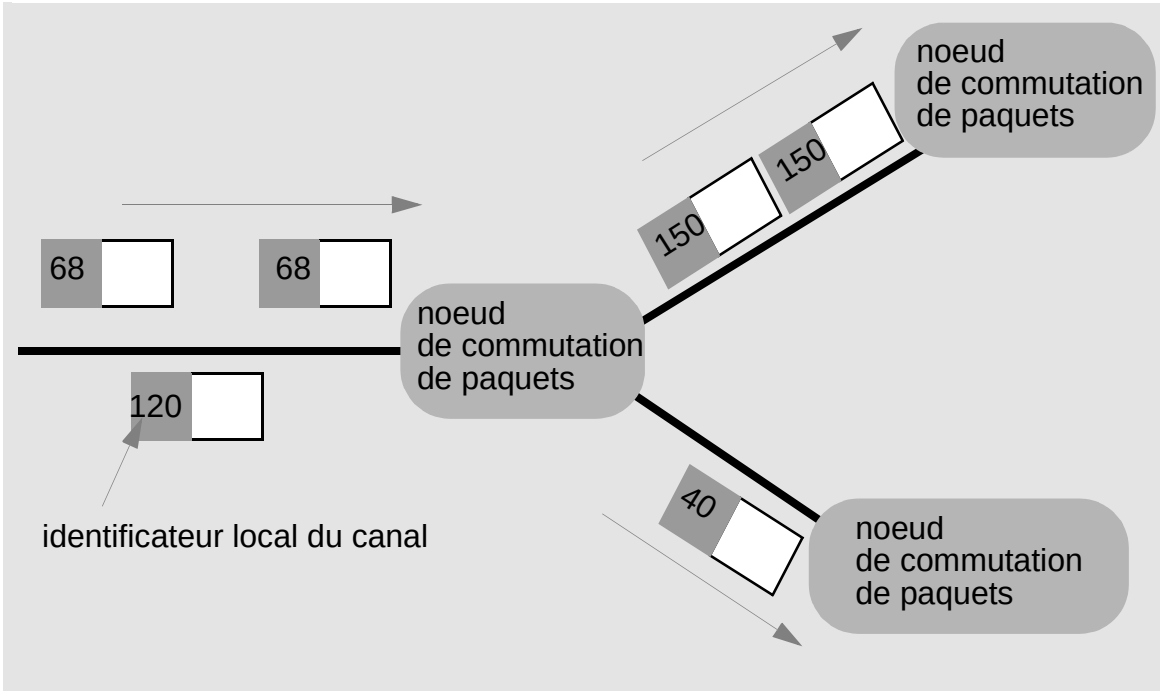


FIGURE 131. X.25: routage des paquets

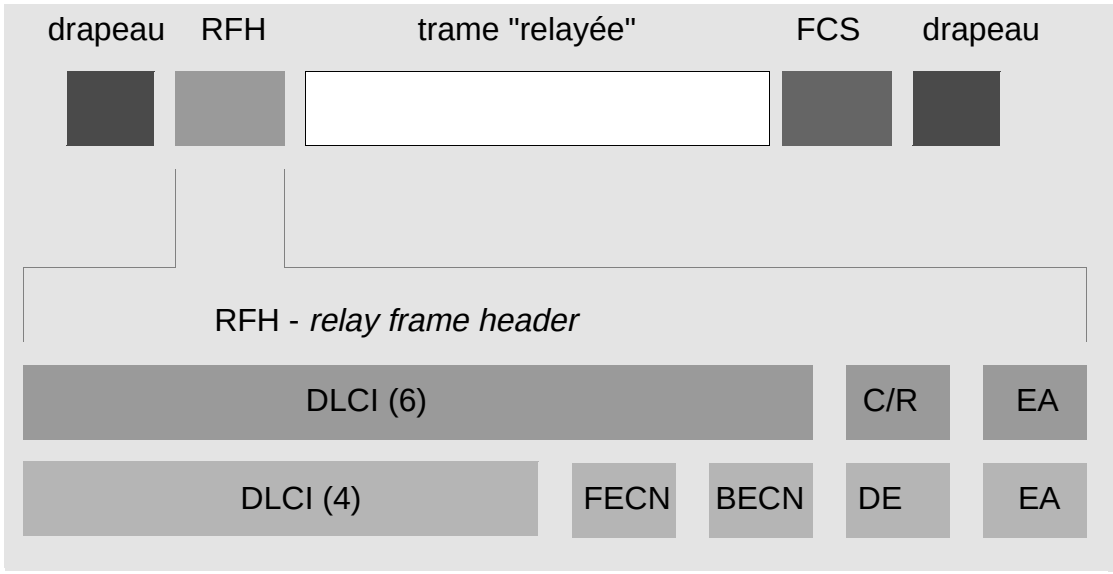


Relais de trames - "frame relay"

Le réseau «relais de trames» est une extension du protocole de lien de X.25. Le protocole «f*rame relay*» permet de transmettre à l'interieur de ces propres trames les trames provenant d'un réseau externe (*alien frames*). Le protocole «*frame relay*» fonctionne avec des trames plus longues que les trames X.25. Les trames externes ne sont ni analysées ni modifiées par le «*frame relay*».

Sur le schéma ci-dessous nous présentons la structure d'une trame «*frame relay*».

FIGURE 132. Une trame de "*frame relay*"



DLCI - 10 bits, data-link connection identifier -
identifies uniquely the virtual circuit to which the frame belongs

C/R - 1 bit, a command/response bit

EA - 2 bits, the extended address markers,
0 in the first byte and 1 in the second to indicate the end of the header

FECN - 1 bit, forward explicit congestion notification,

BECN - 1 bit, backward explicit congestion notification

DE - 1 bit, discard eligibility

Une trame «relais de trames» commence et se termine par un fanion (0x7E). Le contenu de la trame est protégé par un code FCS sur 16 bits. Le champ DLCI (10 bits) identifie un des 1024 circuits virtuels; il est utilisé pour le routage.

Le réseau «*frame relay*» offre seulement des circuits virtuels permanents PVC (*Permanent Virtual Circuits*).

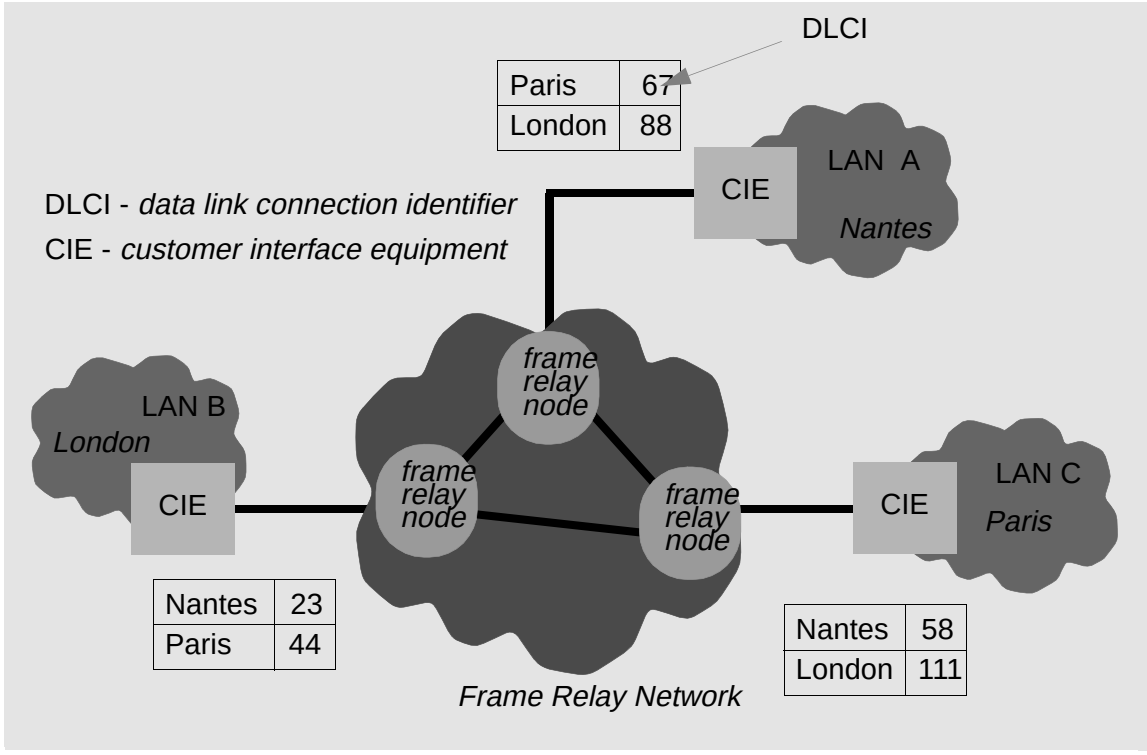
Services "frame relay"

Le service «*frame relay*» permet de relayer les trames utilisateur dont la taille ne dépasse pas 8192 octets (e.g. une trame Ethernet contient 1520 octets). L'acheminement des trames est très rapide; les commutateurs de «frame relay» n'effectuent ni la détection des erreurs ni la retransmission. Les trames qui provoquent une congestion sont abandonnées. Les débits offerts par les «frame relays» varient entre 128 Kbit/s et 34 Mbit/s. Néanmoins, la présence de plusieurs noeuds de communications peut introduire des délais difficilement acceptables pour les applications temps-réel. La congestion est un autre problème à résoudre. Les réseaux de «frame relay» réduisent ces difficultés par l'emploi des plusieurs indicateurs tels que:

- FECN - *Forward Explicit Congestion Notification*,
- BECN - *Backward Explicit Congestion Notification*,
- DE - *Discard Eligible*.

Les indicateurs FECN et BECN sont initialisés par le réseau pour informer les noeuds de communication du problème de congestion. Le bit DE=1 sert à indiquer que la trame en question est moins prioritaire et peut être abandonnée.

FIGURE 133. Réseau «frame relay»: connexions et circuits logiques



Un utilisateur qui a négocié les services avec un débit garanti peut être sûr que toutes les trames qui respectent le débit négocié soient acheminées sans problème.

Résumé

Les réseau xet les liens à longue distance offrent une multitude de services permettant d'interconnecter les réseaux locaux et les postes des utilisateurs individuels. Selon les besoins les services WAN permettent de communiquer avec des débits importants et des délais relativement courts. Une partie non négligeable des ressources de communication à longue distance est sollicitée par les fournisseurs de services Internet. Une autre partie est exploitée par les établissements pour leur réseaux virtuels. D'autres services, tels que la téléconférence et le télétravail nécessitent également des ressources de télécommunication à haut débit.

Dans les années à venir, les services WAN seront de plus en plus sollicités par les applications multimedia offertes sur l'Internet et seront accessibles par le biais des connexions xDSL.

Sockets

1.0 *BSD* Unix Interface to Internet Protocols

The Unix input and output operations follow a paradigm *sometimes referred to as* open-read/write-close. This scheme includes *character oriented* I/O devices like teletypes (ttys), and *block oriented* devices like disks and data files. However, the network based devices need more complex interaction between user processes. In particular, *an interface to protocols must allow users* to create both server process that *awaits* connections *passively* and client process that forms connections actively. *Furthermore*, user processes *may wish to* communicate using both connectionless as well as connection based operations.

Given all these cases the BSD Unix Interface to Internet Protocols proposes several additional system calls *coping with* the increased complexity of network based I/O functionalities. This interface is provided *by means of an abstraction known as the socket*. *We can still think of* a socket as a generalization of the initial *open-read/write-close paradigm*.

The essential difference between file descriptors or IPC identifiers and sockets is that the creation of a socket does not *bind* it to a specific destination address or device. To create a socket the `socket` call is used (`sid=socket(af,type,protocol)`). The `af` parameter *stands for* address family and indicates how to interpret network addresses passed *through* the network calls. The `type` parameter specifies the type of communication desired, and finally the `protocol` parameter gives the protocol number to be used. Notice that in the `socket` call there is nothing *to attach* the socket to a specific network address or device.

The following table gives us a comparison between several inter process communication calls based on files/ FIFOs, messages, and sockets. These call are grouped according to the server and client

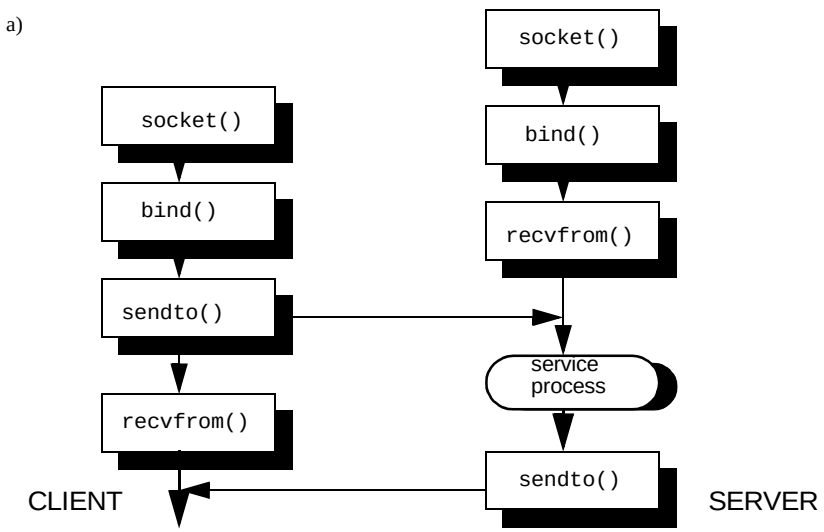
functionalities.

The underlying differences between two basic communication modes, connection-oriented and c*onnectionless*, significantly influence *the choice* of the network calls necessary to provide an effective communication between two processes. Figure 4.21 illustrates *the suites of network calls used respectively for connectionless and connection-oriented protocols*.

TABLEAU 1.

	files/FIFOs	**messages**	**sockets**
Server	`open(), mknod()`	`msgget()`	`socket()`
create endpoint			`bind()`
bind address			`listen()`
specify queue			`accept()`
wait for connection			
Client			
create endpoint	`open(), mknod()`	`msgget()`	`socket()` `bind()` `connect()`
transfer data	`read(),write()`	`msgrcv()`	`read(),write(), recv(),send()`
transfer datagrams			`recvfrom(), sendto()`
terminate	`close(),unlink()`	`msgctl()`	`close(), shutdown()`

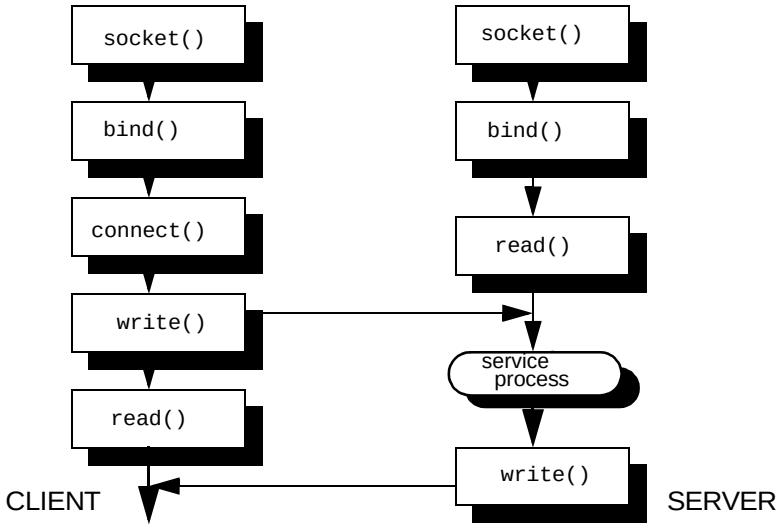
FIGURE 1. Connectionless communication with DATAGRAM sockets: without server address preparation,



Connectionless communication with DATAGRAM sockets
a) without server address preparation,
b) with server address prepared by connect call (which does not mean connection establishment)

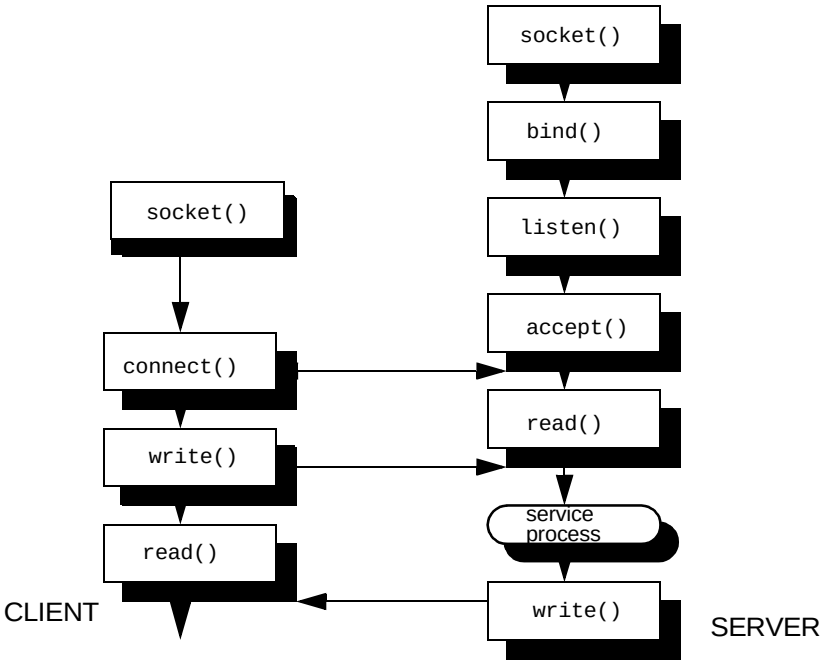
FIGURE 2. Connectionless communication with DATAGRAM sockets: with server address prepared

b)



Connectionless communication with DATAGRAM sockets
a) without server address preparation,
b) with server address prepared by connect call (which does not mean connection establishment)

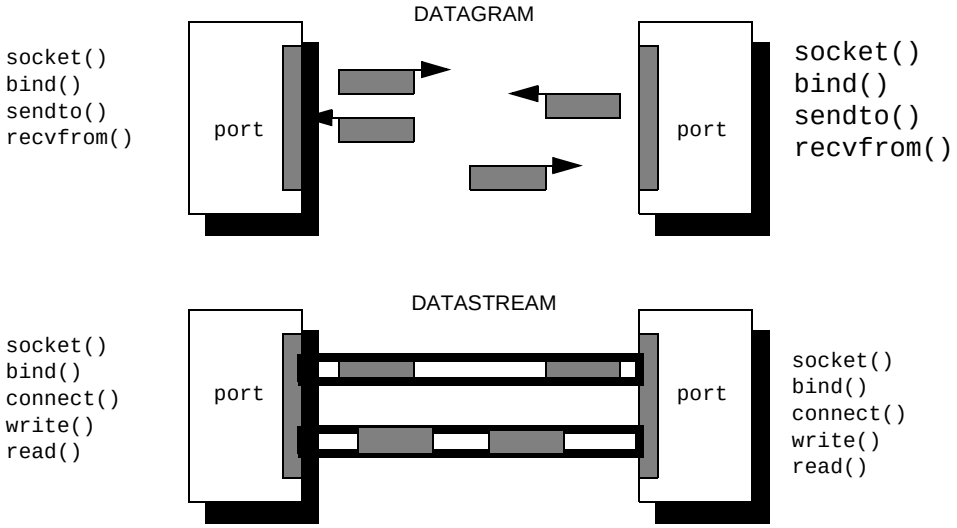
FIGURE 3. Connection-based communication with STREAM sockets.



Connection-based communication with STREAM sockets.

The connect call in the client process establishes connection.

FIGURE 4. Datagram and Datastream communication



Datagram and Datastream communication and the corresponding socket based operations.

2.0 Socket Addresses

One of the essential parameters of socket mechanism is address family. *There are more than ten different address families* or domains, however in our *future examples* only two of them will be considered Internet addresses - the Internet address domain, and local addresses - the Unix domain. The definition of the corresponding address structures are given in `<sys/socket.h>` and `<sys/un.h>` files.

```
#define  AF_UNSPEC  0  /* unspecified */
#define  AF_UNIX  1    /* local to host  */
#define  AF_INET  2  /  * internetwork: UDP, TCP, etc. */
/* Structure used by kernel to store most addresses.*/
struct sockaddr {
u_short  sa_family;  /* address family */
char  sa_data[14];  /* up to 14 bytes of direct add.*/
};
/ * Definitions for UNIX IPC domain. */
struct  sockaddr_un {
short   sun_family;    /* AF_UNIX */
char    sun_path[108];  /* path name (gag) */
};
```

The content of the `sa_data[14]` field for the Internet family is as follows :

```
struct sockaddr_in {
short    sin_family;  /* AF_INET */
u_short  sin_port;    /* 16 bit port number - nbo */
struct   in_addr sin_addr; /* 32 bit netid/hostid */
char     sin_zero[8]; /* unused */
};
```

Where :

```
struct in_addr {
u_long  s_addr;   /* 32-netid/hostid */
                  /* nbo - network byte ordered */
};
```

3.0 Socket Based System Calls

Any application process doing network operations must start by `socket` system call.

`int sd=socket(int af,int type,int prot);`

The `af`- address family is (in our applications) one of :

- `AF_UNIX` - Unix domain and internal protocols,
- `AF_INET` - Internet domain and Internet protocols.

The socket type is one of the following :

```
#define  SOCK_STREAM  1    /* stream socket */
#define  SOCK_DGRAM  2     /* datagram socket */
#define  SOCK_RAW  3      /* raw-protocol interface */
#define  SOCK_RDM  4  /* reliably-delivered message */
```

Only few combinations of the above presented address' families and type parameters are valid:

	AF_INET	**AF_UNIX**
SOCK_STREAM	TCP	yes
SOCK_DGRAM	UDP	yes
SOCK_RAW	IP	

Typically, the `prot` argument is *set to zero*.

The `sd` value returned by the `socket` call is a small integer similar to file descriptor. We call it socket descriptor. Notice that after the `socket` call, which specifies only the address domain and protocol to be used, *we still need to bind it to a real host's addresses and port*s.

To close a socket we use `close` call.

`int close(int sd);`

In the Unix domain (`AF_UNIX`) a specific socket call - `sockpair` creates *a pair* of socket descriptors. The `socketpair` call is, *to some extent*, a*nalogous to* the `pipe` call.

An example of two *allowable* `socketpair` call versions is given below.

```
int s[2];
/* socket input-s[1]/output descriptors s[0]*/

socketpair(AF_UNIX,SOCK_STREAM,0,s);
```

or

```
socketpair(AF_UNIX,SOCK_DGRAM,0,s);
```

Remark: To be used effectively the `s[0]`and `s[1]` descriptors *must be bound* to the internal Unix addresses - pathnames.

`int bind(int sd,struct sockaddr *ma,int al);`

The `sd` argument is the socket descriptor obtained after the `socket` call, the second

argument `*ma` is a pointer to Internet (Unix) protocol specific address, and the third argument is the size of the address structure - `sizeof(sockaddr)` or `sizeof(sockaddr_un)`.

There are three uses of `bind` call:

- to register the well-known addresses of the server for both connection-oriented and connectionless protocols;
- to register specific addresses of the client for itself;
- to specify some unique address for a client *in order to* pass it on to the server.

When binding a socket to the service port we dont need to know the *underlying* Internet address. The following fragment of program illustrates this case:

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

#define  SERV_TCP_PORT  7000

main(argc,argv)
int argc;
char *argv[];
{
int  sockfd;
struct  sockaddr_in sa;

if((sockfd=socket(AF_INET,SOCK_STREAM,0))<0)
  {
   printf("server: can't open stream socket");
   exit(1);}

bzero((char *)&sa,sizeof(sa));
sa.sin_family  = AF_INET;
sa.sin_addr.s_addr = htonl(INADDR_ANY);
sa.sin_port =  htons(SERV_TCP_PORT);

if(bind(sockfd,(struct sockaddr *)&sa,sizeof(sa))<0)
  {
   printf("bind: can't connect local address");
   exit(2);}
/* rest of the server program */
```

The binding of sockets in the Unix domain *is much simpler*. All we need is *an ASCII string* representing the *datapath association.*

```
#include <sys/un.h>
..
struct  sockaddr_un sa;
...
sockfd=socket(AF_INET,SOCK_STREAM,0))
sa.sin_family  = AF_UNIX;
strcpy("/tmp/toto",sa.sun_path);
bind(AF_UNIX,(struct sockaddr_un *)&sa, sizeof(sa))
```

connect(int sd,struct sockaddr *sa,int al.

The sd argument is the socket descriptor *obtained* after the socket call, the second argument *sa is a pointer to Internet (Unix) *protocol specific and remote server address*, and the third argument al is the size of the address structure.

The connect call *results in* the establishment of a connection between local and remote Unix kernel. The client process connecting to the remote server does not need to bind the socket before calling connect. In this case, the connect call associates a local address (port number and Internet address) to the local process.

Notice that for the datagram based communication the bind call is obligatory in the client process. The client can also evoke the connect call just to prepare the remote server address to be used in the consecutive datagrams. The "connected" datagrams can be sent and received by using simple write/read calls.

```
#include  <netinet/in.h>
#include  <arpa/inet.h>
#define  SERV_TCP_PORT  7000
#define  SERV_HOST_ADDR "191.8.220.44"

main(argc,argv)
int argc;
char *argv[];
{
int  sockfd;
struct  sockaddr_in sa;
bzero((char *)&sa,sizeof(sa));
sa.sin_family  = AF_INET;
sa.sin_addr.s_addr = inet_addr(SERV_HOST_ADDR);
sa.sin_port =  htons(SERV_TCP_PORT);
if((sockfd=socket(AF_INET,SOCK_STREAM,0))<0)
  { printf("client: can't open stream socket");
    exit(1);}
if(connect(sockfd,(struct sockaddr *)&sa,
   sizeof(sa))<0)
  { printf("client: can't connect to server address");
    exit(2); }
```

int listen(int sd,int backlog);

The listen call is executed after the bind call. The *backlog* argument specifies the size of connection queue, *it means the number of consecutive* connection *calls to be allowed by* the server process. Recall the client-server model with multiple servers (Figure 4.20), where the initial server process *forks* at each new connection request call.

Normally, the listen call is immediately followed by the accept call.

accept(int sd,struct sockaddr *ca,int *al);

The accept call *takes the first connection request on the queue* and creates a new soc-

ket which has the same properties as the initial one. The `*ca` argument receives the address of the *newly connected* client process. For the Internet the value of `al` argument is the actual number of bytes in the `ca` argument.

The following fragment shows the use of `accept` call in the context of a *concurrent server*.

```
if(bind(sd,(struct sockaddr *)&sa,sizeof(sa))<0)
  printe("bind: can't connect local address");
listen(sd,5);
for(;;)      {
   cl=sizeof(ca);
   newsd=accept(sd,(struct sockaddr *)&ca,&cl);
   if(newsd<0) printe("server: accept error");
   if((childpid=fork())<0)
     printe("server: fork error");
   else if(childpid==0)
     {
     close(sd);
     do_something_with(newsd);
     exit(0);
     }
   close(newsd);   /* parent process */
   }
}
```

And the corresponding "sequential" server *spawning* one process at a time:

```
if(bind(sd,(struct sockaddr *)&sa,sizeof(sa))<0)
  printe("bind: can't connect local address");

listen(sd,5);

for(;;)      {
   cl=sizeof(ca);
   newsd=accept(sd,(struct sockaddr *)&ca,&cl);

   if(newsd<0) printe("server: accept error");

   do_something_with(newsd);
   close(newsd);   /* parent process */
   }
```

4.0 Data Transfer Calls

The data transfer calls are similar to `read/write` system calls, but additional parameters are required.

- connection-based transfers

```
int send(int sd,char *buff,int nb,int flags);
int recv(int sd,char *buff,int nb,int flags);
```

- datagram transfers

```
int sendto(int sd,char *buff,int nb,int flags,
                  struct sockaddr *to,int al);
int recvfrom(int sd,char *buff,int nb,int flags,
                  struct sockaddr *from,int al);
```

All four transfers return the number of bytes sent/received effectively by the function.

The possible *flags* arguments are the following :

`MSG_OOB`	send or receive *out-of-band data*
`MSG_PEEK`	*peek* the incoming message (not destructive receive)
`MSG_DONTROUTE`	*bypass routing*

5.0 Utility Functions

To complete our introduction to socket based functions we need some additional functionalities for specific *data moves* and conversions.

Data conversions

The conversion of internal to network data formats and vice versa are done by the following functions:

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

Remark: See the big-endian and little-endian architectures (p. 59).

5.1 String manipulation functions

The following routines allow us to copy and to compare the byte strings *which are not necessarily ended* by `NULL` character.

```
bzero(char *dest, in nb);
/* load nb NULL characters starting with dest address */
bcopy(char *source, char *dest, in nb);
/* move nb characters from source to dest address */
int bcmp(char *source1, char *source2, int nb);
/* compare nb characters starting from source1 and source2  adresses */
```

5.2 Address conversions

The address conversion routines allow the user to convert decimal Internet addresses into internal format and vice versa.

Dotted Internet address (e.g. `128.10.2.3`) to long integer:

```
long inet_addr((char *)ta);
```

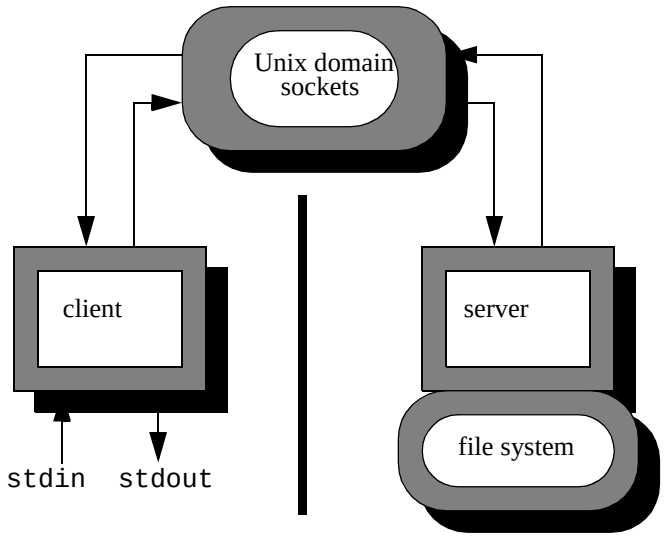
Long integer to dotted Internet addres :

```
char *inet_ntoa((long) inta);
```

6.0 A Simple Example - Unix Domain

Our first example is based on the client-server model and on socket mechanism applied to the communication in the Unix domain. Recall our client-server model used in the previous chapter.

FIGURE 5. Simple file server



In this model the client sends the server the name of the file *he wants to read*. The server receives the name, opens the file (if it exists), and transmits the content of the file to the client. The *IPC device* used now in our example is socket mechanism. In the first three versions of the model, *the client and the server reside at the same Unix kernel*. It means that the address domain used is the Unix domain.

The first and the second version are based on datagram communication, so the socket creation call, both in the client and in the server process is as follows:

```
socket(AF_UNIX,SOCK_DGRAM,0);
```

This socket is *bound explicitly* by `bind` call or implicitly by `connect` call to the local name, then the communication *proceeds* through `sendto` and `recvfrom` calls.

6.1 Client process

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/un.h>
#include "dg_un.h"
main()
{
int sdc;
struct sockaddr_un client_s,server_s;

if((sdc=socket(AF_UNIX,SOCK_DGRAM,0))<0)
  printe("client: can't socket_c");

bzero((char *)&client_s,sizeof(client_s));
client_s.sun_family=AF_UNIX;
strcpy(client_s.sun_path,"socket_c");

if((bind(sdc,(struct sockaddr *)&client_s,
         sizeof(client_s)))<0)
  printe("client: can't bind sdc");

bzero((char *)&server_s,sizeof(server_s));
server_s.sun_family=AF_UNIX;
strcpy(server_s.sun_path,"socket_s");
client_dg(sdc,(struct sockaddr_un *)&server_s,
          sizeof(server_s));
close(sdc);
unlink("socket_c");
exit(0);
}
```

The server process creates `sdc` socket (`sdc` is socket's descriptor), and it binds the socket with local name `"socket_c"` *previously loaded into* `client_s` *structure*. Then it prepares `server_s` structure describing the server socket and passes it to `client_dg` function. The `client_dg` function reads the name of the file from standard input and sends it the server process (`sendto`); then the

client process *suspends and awaits* the result at `recvfrom` call. The received data, error message or file content, *are displayed* at the standard output.

6.2 Server process

```
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/un.h>
#include "dg_un.h"

main()
{
int sds;
struct sockaddr_un server_s,client_s;
```

```
if((sds=socket(AF_UNIX,SOCK_DGRAM,0))<0)
  printe("server: can't socket s");
bzero((char *)&server_s,sizeof(server_s));
server_s.sun_family=AF_UNIX;
strcpy(server_s.sun_path,"socket_s");

if((bind(sds,(struct sockaddr *)&server_s,
         sizeof(server_s)))<0)
  printe("server: can't bind sds");
printf("server started \n");

client_s.sun_family=AF_UNIX;
strcpy(client_s.sun_path,"socket_c");
server_dg(sds,(struct sockaddr *)&client_s,
          sizeof(client_s));
close(sds);
write(1,"\n end of server \n",21);
unlink("socket_s");

exit(0);
}
```

The server process creates a "datagram" socket and binds it with a local name ("`socket_s`"). Then it prepares the client socket address structure and passes it to `server_dg` function *along with* the local socket descriptor (`sds`). The `server_dg` function uses these arguments to receive (`recvfrom`) and to send the data (`sendto`). *After the data has been sent, the server ends execution and exits.*

6.3 `server_dg` and `client_dg` functions

```
#include <stdio.h>
#include "printe.h"
#define MAXBUFF124

client_dg(sd,sad,adl)
int sd;
struct sockaddr *sad;
int adl;
{
char  buff[MAXBUFF];
int  n;
int slen=adl;
printf("give the file name to read by server : ");
if(fgets(buff,MAXBUFF,stdin)==NULL)
  printe("client: file name read error \n");
n=strlen(buff);
if(buff[n-1]=='\n') n--;
if(sendto(sd,buff,n,0,sad,adl)!=n)
  printe("client: file name write error \n");
while((n=recvfrom(sd,buff,MAXBUFF,0,sad,&slen))>0)
  write(1,buff,n);
if(n<0) printe("client: data read error \n");
}
```

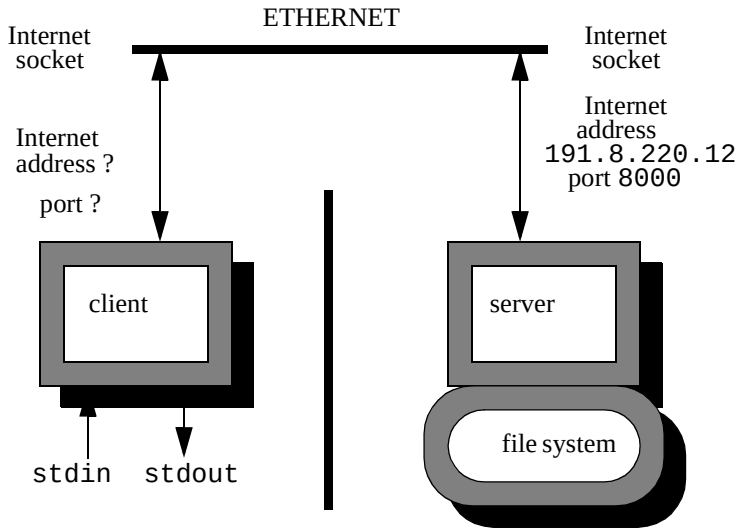
```
server_dg(sd,sad,adl)
int sd;struct sockaddr *sad;int adl;
{
char  buff[MAXBUFF], errmesg[256];
int  n,fd;
int clen=adl;
n=recvfrom(sd,buff,MAXBUFF,0,sad,&clen);
if(n<0)
  printe("server: can't recvfrom \n");
buff[n]='\0';
if((fd=open(buff,0))<0)
  {
  sprintf(errmesg," \ncan't open file %s\n",buff);
  strcat(buff,errmesg);
  n=strlen(buff);n++;
if(sendto(sd,buff,n,0,sad,clen)<0)
    printe("server: errmesg write error \n");
  }
else
  {
  printf("server sending a file \n");
          while((n=read(fd,buff,MAXBUFF))>0)
  if(sendto(sd,buff,n,0,sad,clen)<0)
    printe("server: file write error \n");
  if(n==0)return 0;
  }
}
```

7.0 A Simple Example - Internet domain

The next two versions of our example are based on *Internet addressing*. To communicate, the Internet address of the server host *must be known to the client*. In our case the server's host address is `191.8.220.12`. If you wish to use the following program you must modify this value by giving the Internet address of your server host. The port number has been set at `8000`. *Anyway* it must be greater than `1023` (reserved port numbers) and *should be greater than* `5000`.

The following example represents the UDP based client and server programs communicating through an Internet socket. The client calls `bind` after creating its socket and asking the system to assign automatically any local address (Internet address and an available port number). *Setting the port to zero causes the system* to assign the client some *unused* port in the range `1024` through `5000`. When the `sendto` call to server is activated the client sends a datagram with its automatically assigned local Internet address and port number.

FIGURE 6. Internet domain communication with sockets: a simple file server



7.1 client/server programs for UDP communication

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include "dg_un.h"
#define UDP_port  8000           /* server port */
#define HOST_addr "191.8.220.12" /* server address */

main()
{
int sdc;
struct sockaddr_in client_s,server_s;
if((sdc=socket(AF_INET,SOCK_DGRAM,0))<0)
  printe("client: can't socket_c");
bzero((char *)&client_s,sizeof(client_s));
client_s.sin_family=AF_INET;
client_s.sin_addr.s_addr=htonl(INADDR_ANY);
client_s.sin_port=htons(0);
if((bind(sdc,(struct sockaddr *)&client_s,
         sizeof(client_s)))<0)
  printe("client: can't bind sdc");
bzero((char *)&server_s,sizeof(server_s));
server_s.sin_family=AF_INET;
server_s.sin_addr.s_addr=inet_addr(HOST_addr);
server_s.sin_port=htons(UDP_port);
client_dg(sdc,(struct sockaddr_un *)&server_s,
          sizeof(server_s));
close(sdc);
exit(0);
}
```

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include "dg_un.h"
#define UDP_port  8000              /* server port */

main()
{
int sds;
struct sockaddr_in server_s,client_s;
if((sds=socket(AF_INET,SOCK_DGRAM,0))<0)
  printe("server: can't socket s");
bzero((char *)&server_s,sizeof(server_s));
server_s.sin_family=AF_INET;
server_s.sin_addr.s_addr=htonl(INADD_ANY);
server_s.sin_port=htons(UDP_port);
if((bind(sds,(struct sockaddr *)&server_s,
         sizeof(server_s)))<0)
  printe("server: can't bind sds");
server_dg(sds,(struct sockaddr *)&client_s,
          sizeof(client_s));
close(sds);
exit(0);
}
```

7.2 Client program for TCP based communication

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include "str_un.h"

#define TCP_port 8000
#define HOST_addr "191.8.220.12"

main(c,v)
int c;
char **v;
{
int sdc;
struct sockaddr_in server_s;
if((sdc=socket(AF_INET,SOCK_STREAM,0))<0)
  printe("client: can't socket_c");
bzero((char *)&server_s,sizeof(server_s));
server_s.sin_family=AF_INET;
server_s.sin_addr.s_addr = inet_addr(HOST_addr);
if(c==1) server_s.sin_port=htons(TCP_port);
else server_s.sin_port=htons(atoi(v[1]));
connect(sdc,(struct sockaddr *)&server_s,sizeof(server_s));
client_str(sdc);
close(sdc);
exit(0);
}
```

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include "str_un.h"
#define TCP_port 8000

main(c,v)
int c;
char **v;
{
int sds,new_sds;
int clen;
int cpid;
struct sockaddr_in server_s,client_s;

if((sds=socket(AF_INET,SOCK_STREAM,0))<0)
  printe("server: can't socket s");
bzero((char *)&server_s,sizeof(server_s));
server_s.sin_family=AF_INET;
server_s.sin_addr.s_addr = htonl(INADDR_ANY);
if(c==1)server_s.sin_port=htons(TCP_port);
else server_s.sin_port=htons(atoi(v[1]));
if((bind(sds,(struct sockaddr *)&server_s,sizeof(server_s)))<0)
  printe("server: can't bind sds");
printf("server started \n");
listen(sds,4);
for(;;)
  {
  clen=sizeof(client_s);
  if((new_sds=accept(sds,
              (struct sockaddr *)&client_s,&clen))<0)
    printe("server: can't accept");
  if((cpid=fork())<0)
    printe("server: can't fork");
  else
    if(cpid==0)
      {
      close(sds);
      server_str(new_sds);
      exit(0);
      }
  close(new_sds);
  }
}
```

8.0 Network Utility Functions

The Internet protocols used by Unix *support several library routines* which make it possible *to look for* host names and addresses, service names and numbers and so on. *Let us consider a case* when we know the name of the host and the service (registered) and we need to find the corresponding Internet address and port number. To do so we will use `gethostbyname` and `getservbyname` routines, both defined in `<netdb.h>` file.

```
struct hostent  *gethostbyname(char *hostname);
```

The `hostent` and `servent` structures are defined as follows :

```
struct hostent {
char  *h_name;             /* official name of host */
char  **h_aliases;         /* alias list */
int  h_addrtype;           /* host address type */
int   h_length;            /* length of address */
char  **h_addr_list;
/* list of addresses from name server */
};
```

```
struct servent *getservbyname(char *servname, char *pname);
struct servent {
char  *s_name;             /* official service name */
char  **s_aliases;         /* alias list */
int   s_port;              /* port # */
char  *s_proto;            /* protocol to use */
};
```

The following program takes the name of the host and the name of the service and displays the corresponding Internet address (dotted format) and port number.

```
/* getport.c */

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

main(c,v)
int c;
char **v;
{
struct hostent *host;
struct servent *service;
struct in_addr *addr;
if(c<3)
  {
  printf("usage: getport host_name
                 service_name [protocol_name]\n");
  exit(1);
  }
host = gethostbyname(v[1]);
addr = (struct in_addr *) *(host->h_addr_list);
printf("Internet address: %s  ",inet_ntoa(*addr));
if(c==4)service = getservbyname(v[2],v[3]);
else service = getservbyname(v[2],NULL);
printf("port number: %lu  \n",ntohl(service->s_port));
}
```

Execution's example :

```
$getport samara ftp tcp
Internet address: 191.8.220.31  port number: 21
```

The steps required by a client to start a dialogue with a server process are standard. For the connectionless protocol (UDP based) we go through `socket`, `bind`, and optionally `connect` call to prepare the server address for the *future datagrams*. In this context we would like not to use *hard-coded* network addresses but those supplied by network utility routines. *Below we provide some utility functions that handle* a large number of operations required by a typical client/server model application.

```
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/un.h>
#include <stdio.h>
#include <netinet/in.h>
#include <netdb.h>
typedef int INsct ; /*  Internet socket*/

INsct crs  (/*int type*/) ;
void    gpn (/*INsct s*/) ;
void    gpn_p (/*INsct s, int nport*/) ;
void    connet    (/*INsct s, char *rhost, int nport*/) ;

INsct mkc     (/*int type, char *rhost, int nport*/) ;
INsct mks    (/*int type*/) ;
INsct mks_p    (/*int type, int nport*/) ;

INsct crs(type)
int type ;
{int sock ;
   if ((sock = socket(AF_INET, type, 0)) < 0) {
     perror ("crs") ; exit (1) ;
   }  return sock ;
}

void     gpn(s)
INsct s ;
{struct sockaddr_in pub_name ;
 int lg = sizeof pub_name ;
   bzero((char *)&pub_name,lg);
   pub_name.sin_family = AF_INET ;
   pub_name.sin_addr.s_addr = INADDR_ANY ;
   pub_name.sin_port = htons(0);
   if (bind (s, (struct sockaddr *) & pub_name,
             sizeof pub_name) < 0) {
     perror ("gpn:bind") ; exit (1) ;
   }
   if (getsockname (s, (struct sockaddr *) &pub_name, &lg) < 0) {
      perror ("gpn:getsockname") ; exit (1) ;
   }
   printf ("port number allocated: %d\n", ntohs (pub_name.sin_port)) ;
}
```

```
void     gpn_p(s,nport)
INsct s ;int nport ;
{struct sockaddr_in pub_name ;
 int lg = sizeof pub_name ;

   bzero((char *)&pub_name,lg);
   pub_name.sin_family = AF_INET ;
   pub_name.sin_addr.s_addr = INADDR_ANY ;
   pub_name.sin_port = htons(nport);
   if (bind (s, (struct sockaddr *) & pub_name,sizeof pub_name) < 0)
                        {perror ("gpn:bind") ; exit (1) ; }

   if (getsockname (s,(struct sockaddr *) &pub_name, &lg) < 0)
      { perror ("gpn:getsockname") ; exit (1) ;}
   printf ("port number allocated: %d\n", ntohs (pub_name.sin_port)) ;
}

void     connet(s,rhost,nport)
INsct s ; char *rhost ; int nport ;
{struct sockaddr_in pub_name ;
 struct hostent *he ;

   pub_name.sin_family = AF_INET ;
   if ((he = gethostbyname (rhost)) == NULL) {
     fprintf (stderr, "connet:gethostbyname%s: unknown host\n",rhost);
      exit (1) ;
   }
  bcopy((char *) he->h_addr,(char *)&pub_name.sin_addr, he->h_length);
  pub_name.sin_port=htons(nport);
   if (connect(s,(struct sockaddr *) & pub_name,sizeof pub_name) < 0)
                      {perror ("connet:connect") ; exit (1) ;}
}

INsct mkc(type,rhost,nport)
int type ; char *rhost ; int nport ;

{INsct s ;
   s = crs(type) ;
   connet (s,rhost,nport);
   return s ;
}

INsct mks(type) int type ;
{INsct s ;
   s = crs(type) ;
   gpn(s) ;
   return s ;
}

INsct mks_p(type,nport) int type ; int nport;
{INsct s ;
   s = crs(type) ;
   gpn_p(s ,nport) ;
   return s ;
}
```

Now we can use our library file to simplify the previously introduced versions of client and server programs.

For example the TCP version of the client program can be expressed as:

```
#include "lib_util.c"
#include "str_un.h"

main(c,v)
int c;
char **v;
{
INsct sdc;
if(c!=3)
  { printf("usage: c31 host_name port_number\n"); exit(1);}
sdc=mkc(SOCK_STREAM,v[1],atoi(v[2]));
if(sdc<0)printe("client: can't socket_c");
client_str(sdc);
close(sdc);
exit(0);
}
```

Notice, that the user must deliver the server's host name (ASCII string) and the port number.

The corresponding server program may be presented in two forms; the first one expresses "concurrent" server, the second "sequential" server.

```
#include "lib_util.c"
#include "str_un.h"
main(c,v)
int c;
char **v;
{
INsct sds,new_sds;
int cpid;
if((sds=mks(SOCK_STREAM))<0)
  printe("server: can't socket s");
listen(sds,4);
for(;;)
  {
  if((new_sds=accept(sds,NULL,NULL))<0)
                        printe("server: can't accept");
  if((cpid=fork())<0)
                        printe("server: can't fork");
  else
    if(cpid==0)
      {
      close(sds);
      server_str(new_sds);
      exit(0);
      }
  close(new_sds);
  }
}
```

```
#include "lib_util.c"
#include "str_un.h"
main(c,v)
int c;
char **v;
{
INsct sds,new_sds;
if(c!=2)
  {printf("usage: s312 port_number\n");exit(1);}
if((sds=mks_p(SOCK_STREAM,atoi(v[1])))<0)
  printe("server: can't socket ");
listen(sds,4);
for(;;)
  {
  if((new_sds=accept(sds,NULL,NULL))<0)
    printe("server: can't accept");
  server_str(new_sds);
  close(new_sds);
  }
}
```

Finally, *we shall try to find* automatically the server host and process by looking through the network database. In the following client example the parent process *looks for* the existing hosts by using `gethostent` call. Then it creates one child process per host address. Each child process uses this address trying to connect with a server (the service port number is fixed). After the connect *failure* or `time_out` fixed in the program the child process exits, *otherwise* it connects. The server program is the same as in the previous example.

```
#include "lib_util.c"
#include "str_un.h"
#include <signal.h>
#define tempo 3
int connection;
ch_inter(n) /* child interrupt procedure */
int n;
{
signal(n,ch_inter);
printf("to long delay\n");
kill(getppid(),SIGUSR2);
exit(1);
}

inter(n)  /* parent interrupt procedure */
int n;
{
signal(n,inter);
if (n==SIGUSR1)  {
if (connection==0) printf("successful connection \n");
connection=connection+1;  }
if (n==SIGUSR2) printf("no connection\n");
}
```

```
main (c,v)
int c;
char **v;
{
INsct sock ;
struct hostent *hp;
struct sockaddr_in pub_name;
int n,recep_signal;
signal(SIGALRM,ch_inter);
signal(SIGUSR1,inter);
signal(SIGUSR2,inter);
connection=0;
if(c!=2)
  { printf("usage: c312rech port_number\n");exit(1);}
sethostent(1);
while ((hp=gethostent())!=NULL)
{
if ((n=fork())==0)
{printf("\ntried host  : %s\n",hp->h_name);
sock = crs(SOCK_STREAM);
pub_name.sin_family = AF_INET;
bcopy ((char *) hp->h_addr,(char *) & pub_name.sin_addr,hp->h_length);
pub_name.sin_port = htons (atoi(v[1]));
alarm(tempo);
if (connect (sock,(struct sockaddr *) & pub_name, sizeof pub_name) <0)
{
signal(SIGALRM,SIG_IGN);
kill(getppid(),SIGUSR2);
close(sock);
exit(1);
}
else
{
signal(SIGALRM,SIG_IGN);
kill(getppid(),SIGUSR1);
printf("\nyou are connected with %s\n",hp->h_name);
client_str(sock);
close (sock) ;
kill(getppid(),SIGUSR1);
exit(0);
}
}
else
{
/* parent process */
pause();
if (connection==1) break;

}
}
endhostent();
if (connection==0)
printf("\nno connection found\n");
else while (connection != 2 );
exit(0);
}
```

9.0 Broadcasting

One of the interesting possibilities of datagram communication is broadcasting. As you recall, *to broadcast* a datagram we need to construct an Internet address in which all the bits indicating host computers *must be set to 1*. For example all hosts connected to a class B network `128.5.X.X` will be addressed by `128.5.255.255` value.

The following example contains two programs; the first one broadcasts messages (server), the second receives messages (client).

```
#include "lib_util.c"
#include "printe.h"
#define portnum 5555
main(c,v) /* emitter */
int c;
char **v;
{
INsct sock;
struct netent *netbuf;
u_long netaddr;
int i_val;
struct sockaddr_in destaddr;
if (c!=3) {printf("usage : broadcast network_name message\n");
    exit(1);}
sock=mks_p(SOCK_DGRAM,portnum+1);

netbuf = getnetbyname(v[1]);

if(netbuf==NULL)printe("net name unknown");
netaddr=netbuf->n_net;
do {
netaddr = netaddr << 8;
netaddr = netaddr | 0xFF;
}
while (netaddr < inet_addr("0.255.255.255"));
destaddr.sin_family = AF_INET;
destaddr.sin_addr.s_addr = netaddr;
destaddr.sin_port = portnum;
sendto(sock,v[2],strlen(v[2]),0,&destaddr,sizeof destaddr);
}
```

```
#include "lib_util.c"
#define portnum 5555
main () /* receiver */
{
INsct sock ;
char buf [1024] ;
   sock = mks_p (SOCK_DGRAM,portnum) ;
   while(1)
   {
   read (sock, buf, 1024) ;
   if(*buf=='.') break;
   printf ("the received message: %s\n", buf) ;
   }
   close (sock) ;}
```

9.1 Connection to an unknown server

The second example gives us another solution to the problem of connection to an unknown server host. *This time* we use broadcast messages to send a query requesting the name of the server. The server process *sends* a broadcast message *as a reply*, and inserts its host name in it. After the reception of this datagram the client establishes connection with the server. Notice, that the server *must bind its* socket to *an externally known* port number.

The following program represents the client process which *starts with* the creation of a datagram socket and the sending of a broadcast message. To do it correctly the client process must get the local net name using `local_name` function prepared in `"local_name.h"` file.

```
char *local_netname()
{
struct netent *np;
struct hostent *hp;
char host[128];
unsigned long *addr,netaddr;
gethostname(host,128);
hp=gethostbyname(host);
addr=(u_long *) *(hp->h_addr_list);
netaddr=inet_netof(*addr);
np=getnetbyaddr(netaddr,AF_INET);
return(np->n_name);
}
```

9.1.1 Client program :

```
#include "lib_util.c"
#include "str_un.h"
#include "local_name.h"
main(c,v)
int c;
char **v;
{
INsct sock;
struct netent *netbuf;
u_long netaddr;
int i_val;
struct sockaddr_in destaddr;
char *message="client ready";
char recep[1024];
if (c!=2) { printf("usage : c32 port_number\n"); exit(1); }
sock=mks_p(SOCK_DGRAM,atoi(v[1])+1);
netbuf = getnetbyname(local_netname());
if(netbuf==NULL) printe("can't net_name ");
netaddr=netbuf->n_net;
do {
netaddr = netaddr << 8;
netaddr = netaddr | 0xFF;
}
while (netaddr < inet_addr("0.255.255.255"));
```

```
destaddr.sin_family = AF_INET;
destaddr.sin_addr.s_addr = netaddr;
destaddr.sin_port = atoi(v[1]);
sendto(sock,message,strlen(message)+1,0,&destaddr,sizeof destaddr);
read(sock,recep,1024);
printf("client: received server name %s\n",recep);
close(sock);
sleep(10);
sock=mkc(SOCK_STREAM,recep,atoi(v[1])+2);
if(sock<0) printe("client: can't mkc");
client_str(sock);
close(sock);
exit(0);
}
```

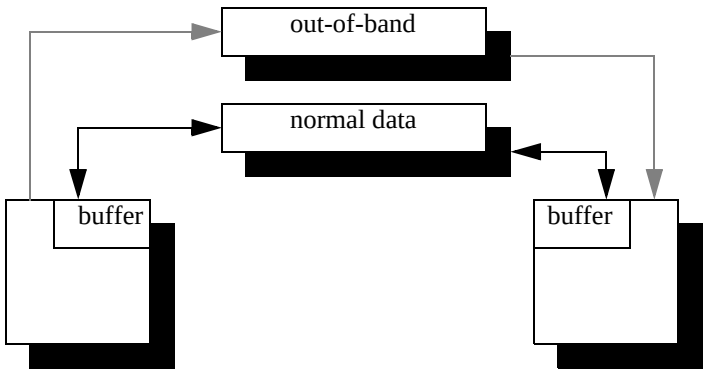
9.1.2 server program :

```
#include "lib_util.c"
#include "str_un.h"
#include "local_name.h"
main(c,v)
int c;
char **v;
{
INsct sock,new_sock;
struct netent *netbuf;
u_long netaddr;
int i_val;
struct sockaddr_in destaddr;
char *message="client ready";
char recep[128];
char host[128];
int size;
if (c!=2) {  printf("usage : s32 port_number\n"); exit(1);}
sock=mks_p(SOCK_DGRAM,atoi(v[1])); /* bound to port_number+1 */
read(sock,recep,128);
printf("server: broadcast message received: %s\n",recep);
netbuf = getnetbyname(local_netname());
if(netbuf==NULL) printe("can't net_name ");
netaddr=netbuf->n_net;
do {
netaddr = netaddr << 8;
netaddr = netaddr | 0xFF;
}
while (netaddr < inet_addr("0.255.255.255"));
destaddr.sin_family = AF_INET;
destaddr.sin_addr.s_addr = netaddr;
destaddr.sin_port = atoi(v[1])+1;
gethostname(host,size);
printf("host %s size %d\n",host,size);
sendto(sock,host,strlen(host)+1,0,&destaddr,sizeof destaddr);
close(sock);
sock=mks_p(SOCK_STREAM,atoi(v[1])+2);
if(sock<0) printe("server: can't mks_p");listen(sock,6);
if((new_sock=accept(sock,NULL,NULL))<0)printe("server:can't accept");
server_str(new_sock);
close(new_sock);close(sock);}
```

10.0 Out-of-Band Data

In a basic *byte-stream service*, the user process can read or write any number of bytes at a time, since there are no message *boundaries*. What happens is that TCP buffers this data internally and *then passes it to the next lower layer* for transmission to the other end. Sometimes a user process *wants to disable* this buffering to send *urgent data* to the server process, for example, an interruption signal generated at client's *keyboard*. The Unix interrupt keys (typically `Control-C` or `Delete` key) are examples of this, as are the *terminal flow control characters* (typically `Control-S` and `Control-Q`). This type of information *is termed* out-of-band data or *expedited* data. The *out-of-band* data are sent by TCP before any data that is buffered. Similarly, at the *receiving end* we want this data to pass before any data that *might have been buffered*. A Unix `signal` is used for this notification. The term of out-of-band is used because it *appears as* a separate communication *channel* (*band*), in addition to the normal data channel.

FIGURE 7. Out-of-band data: urgent data



The TCP protocol allows the use of single-byte (of-line) as well as unlimited-size (in-line) out-of band data.

The following example shows the implementation of a *single-byte* out-of-band data. In the sending program we use a `send` call with `MSG_OOB` flag to indicate that the one-byte `oob` value is to be sent as out-of-band data.

```
#include "lib_util.c"
#include "printe.h"
#include <sys/stat.h>
#define port_number 6000

main(c,v)
int c;
char **v;
{
INsct sock;
char oob='x';
if(c!=3)
  {printf("usage: oob_s1 dest_name normal_message");exit(1);}
```

```
sock=mkc(SOCK_STREAM,v[1],port_number);
if(sock<0) printe("client: can't mkc");
if(send(sock,v[2],strlen(v[2])+1,0)<0)printe("client can't send");
if(send(sock,&oob,1,MSG_OOB)<0)
             printe("client can't send out of band");
printf("out of band message sent\n");
close(sock);
}
```

Notice, that *because of the buffering operations at the receiving end*, the out-of-band data will be received before the normal message.

The receiving process *is slightly more complicated*. To prepare the reception of out-of-band data, two operations must be carried out. First, the `SIGURG` signal is redirected towards a user defined function; then the socket driver is informed that the `SIGURG` signal handling is carried out by the user process. This can be established by calling `fcntl` with a command of `F_SETOWN`, or by calling `ioctl` with a command of either `FIOSETOWN` or `SIOCSPGRP`.

```
#include "lib_util.c"
#include "printe.h"
#include <sys/stat.h>
#include <sys/fcntl.h>
#include <signal.h>

#define port_number 6000
INsct sock,sock_conn;

oob()
{
char mark;
if(recv(sock,&mark,1,MSG_OOB)<0) printe("recv: can't receive");
printf("oob character: %c\n",mark);
close(sock);
close(sock_conn);
exit(0);
}

main()
{
int pid;
signal(SIGURG,oob);
sock_conn=mks_p(SOCK_STREAM,port_number);
listen(sock_conn,4);
sock=accept(sock_conn,NULL,NULL);
if(fcntl(sock,F_SETOWN,-getpid())<0) printe("recv: can't fcntl ");
for(;;);
}
```

The next example implements, in a simplified way, the capture of `Control-C` and `Delete` signals by the client process and the urgent transmission of these signals towards the server process. The server process accepts this data (1 byte specifying the signal number) and *kills itself by sending the corresponding signal*.

```
#include "lib_util.c"
#include "printe.h"
#include <sys/stat.h>
#include <signal.h>
#define port_number 6000
INsct sock;
s_oob(sn)
int sn;
{
char ascii_s;
ascii_s=sn +'0';
if(send(sock,&ascii_s,1,MSG_OOB)<0)
                    printe("send:can't send out of band");
close(sock);
exit(sn);
}

main(c,v)
int c;
char **v;
{
if(c!=3){printf("usage: oob_s2 dest_name normal_message");exit(1);}
signal(SIGINT,s_oob);
signal(SIGQUIT,s_oob);
sock=mkc(SOCK_STREAM,v[1],port_number);
if(sock<0) printe("send: can't mkc");
for(;;)
if(send(sock,v[2],strlen(v[2])+1,0)<0) printe("client can't send");
}
```

The server program :

```
#include "lib_util.c"
#include "printe.h"
#include <sys/stat.h>
#include <sys/fcntl.h>
#include <signal.h>
#define port_number 6000

INsct sock,sock_conn;
oob()
{
char mark;
if(recv(sock,&mark,1,MSG_OOB)<0)printe("recv: can't receive");
printf("\07\nreceived oob character: %c\n",mark);
close(sock);
close(sock_conn);
kill(getpid(),mark-'0');
}
```

```
main()
{
char buff[128];
int nm;
signal(SIGURG,oob);
sock_conn=mks_p(SOCK_STREAM,port_number);
listen(sock_conn,4);
sock=accept(sock_conn,NULL,NULL);
if(fcntl(sock,F_SETOWN,-getpid())<0)
  printe("recv: can’t fcntl ");
for(;;)
  {
  nm=read(sock,buff,128);
  write(1,buff,nm);
  }
}
```

The example above shows how to capture an out-of-band byte independently from its *main stream*. If *we do not need the urgent data to be immediately signalled* we may keep it in the main data stream. To extract the out-of-band data we use the `ioctl` call with `SIOCATMARK` operation and the corresponding `recv` call.

The following example illustrates a receiver program to which the sender transmits urgent data.

```
#include "lib_util.c"
#include "printe.h"
#include <sys/stat.h>
#include <sys/ioctl.h>
#define port_number 6000

INsct sock,sock_conn;
char *mark(sd)
int sd;
{int m;
char oob;
static char buf[1024];
while(1)
  {
  if(ioctl(sd,SIOCATMARK,&m)==-1) printe("can’t ioctl");
  if(m==1)break;
  recv(sd,buf,1024,0);
  }
if(recv(sd,&oob,1,MSG_OOB)<0) printe("can’t recv oob");
return &oob;
}
main()
{
int pid;
int on=1;
sock_conn=mks_p(SOCK_STREAM,port_number);
listen(sock_conn,4);
sock=accept(sock_conn,NULL,NULL);
printf("received oob : %c \n", *mark(sock));
}
```

11.0 Asynchronous reception

The use of SIGIO signal allows the data transmitted by sockets to be received in asynchronous manner. A process, or process group, may *be signalled* the arrival of data at the given socket. Below we recall the necessary phases to establish asynchronous control of an I/O device.

- definition of handler for SIGIO signal;`signal(SIGIO,handler);`
- assignment of the process or process group to the socket device;
  `fcntl(sd,F_SETOWN,-getpid());  /* process */`
- request to send the SIGIO signal at each data arrival. `fcntl(sd,FASYNC,&on);`
  `/* int on=1; */`

```
#include "lib_util.c"
#include "printe.h"
#include <sys/stat.h>
#include <sys/fcntl.h>
#include <signal.h>
#define port_number 6000
INsct sock,sock_conn;

asyn()
{
char mes[1024];
if(recv(sock,mes,1024)<0)
  printe("recv: can't receive");
printf("received async message: %s\n",mes);
close(sock);
close(sock_conn);
exit(0);
}

main()
{
int pid;
int on=1;
signal(SIGIO,asyn);
sock_conn=mks_p(SOCK_STREAM,port_number);
listen(sock_conn,4);
sock=accept(sock_conn,NULL,NULL);
if(fcntl(sock,F_SETOWN,-getpid())<0)
  printe("asyn_recv: can't fcntl ");
fcntl(sock,FASYNC,&on);
for(;;);
}
```

12.0 Socket Options

Several basic option are available to specify the special behaviour of a socket. The options are listed in the <sys/socket.h> file as follows:

```
/* turn on debugging info recording */
#define SO_DEBUG  0x0001
/* socket has had listen() */
#define SO_ACCEPTCONN  0x0002
/* allow local address reuse */
#define SO_REUSEADDR  0x0004
/* keep connections alive */
#define SO_KEEPALIVE  0x0008
/* just use interface addresses */
#define SO_DONTROUTE  0x0010
/* permit sending of broadcast msgs */
#define SO_BROADCAST  0x0020
/* bypass hardware when possible */
#define SO_USELOOPBACK  0x0040
/* linger on close if data present */
#define SO_LINGER  0x0080
/* leave received OOB data in line */
#define SO_OOBINLINE  0x0100
/*
 * Structure used for manipulating linger option.
 */
structlinger {
intl_onoff;/* option on/off */
intl_linger;/* linger time */
};
/*
 * Level number for (get/set)sockopt() to apply to socket itself.
 */
#defineSOL_SOCKET0xffff/* options for socket level */
```

Two system calls,`set/get sockopt` allow us to get and set the above listed options.

```
> int getsockopt(int sd,int level,int option,
                 char *arg_p,int al)
> int setsockopt(int sd,int level,int option,
                 char *arg_p,int al)
```

```
/* Additional options, not kept in so_options. */
#define SO_SNDBUF 0x1001/* send buffer size */
#define SO_RCVBUF 0x1002/* receive buffer size */
#define SO_SNDLOWAT 0x1003/* send low-water mark */
#define SO_RCVLOWAT 0x1004/* receive low-water mark */
#define SO_SNDTIME 0x1005/* send timeout */
#define SO_RCVTIME 0x1006/* receive timeout */
#define SO_ERROR 0x1007/* get error status and clear */
#define SO_TYPE  0x1008/* get socket type */
```

The following example shows the use of some of the above listed socket options:

```
#include "lib_util.c"
#include "printe.h"
#include <sys/stat.h>
#include <sys/fcntl.h>
main(c,v)
int c;
char **v;
{
INsct sock;
int st,n;
struct linger l;
if(c!=3)
  {  printf("usage: op1 dest_name port");  exit(1); }
sock=mkc(SOCK_STREAM,v[1],atoi(v[2]));
if(sock<0) printe("client: can't mkc");
st=4096;
setsockopt(sock,SOL_SOCKET,SO_SNDBUF,&st,4);

l.l_linger=1;
l.l_onoff=1;
setsockopt(sock,SOL_SOCKET,SO_LINGER,&l,sizeof(l));

fcntl(sock,F_SETFL,FNDELAY|fcntl(sock,F_GETFL,0));
st=0;
for(;;)
  {
  n=write(sock,"0123456789",10);
  if(n<0)
    {
    printf("can't write\n");
    break;
    }
  st+=n;
  }
fcntl(sock,F_SETFL,FNDELAY^fcntl(sock,F_GETFL,0));
printf("the size of output+input buffers =%d\n",st);
printf("close sock demanded\n");
close(sock);
printf("close sock realized");
}
```

13.0 Summary

In this chapter we have presented *essential internetwork protocols* and socket mechanisms available to system programmers operating under BSD and Sun Unix systems. Only basic network protocols have been presented; among them: IP (Internetwork Protocol), UDP (User Datagram Protocol), and TCP (Transport Control Protocol). All of them may be used through socket mechanism which makes it possible to elaborate the applications which require connectionless as well as the connection oriented communication. This communication may be realized internally in the Unix domain or externally between *separate Unix kernels* in the Internet domain.

We have used the well-known client/server model *introduced previously* to illustrate various possible applications *built on sockets*. Several additional examples have been used to explain the functioning of out-of-band communication and asynchronous reception.

Additionally, we have presented a simple program involving the use of socket options *in order to control the operations related to buffers' size*.

14.0 Words & Phrases

BSD** - **Berkeley Software Distribution,
sometimes - de temps en temps,
***to refer to a**s* - faire mention de,
character oriented - manipulant caractères,
block oriented - manipulant blocs (de caractères),
awaits .. passively - attend passivement,
furthermore - plus encore,
may wish to - peut vouloir de, il se peut que les processus utilisateurs souhaitent,
to cope with - s'occuper de, faire face à,
open-read/write-close paradigm - le paradigme ouvrir-lire/écrire-fermer,
to bind - lier,
to stand for - représenter,
through - par le biais de, via,
to attach - lier, attacher,
an interface to protocols must allow users - un interface aux protocoles doit permettre aux utilisateurs,
by means of an abstraction known as the socket - par une abstraction connue comme le "socket",
we can still think of - nous pouvons toujours penser à , nous pouvons continuer à nous représenter..,
connectionless - sans connexion,
the choice - le choix,
the underlying differences relative to two basic communication mode**s - les différences sous-jacentes relatives à deux modes principaux de communication,
the suites of network calls used respectively for connectionless and connection-oriented protocols - les suites des appels système utilisées respectivement pour les protocoles sans et avec connexion,
there are more than ten different address families - il y a plus de dix familles d'adresses différentes,
raw-protocol - protocol rudimentaire,
set to zero - initialisé à zéro,
any application process doing network operations - toutes les applications faisant des opérations réseau,
we still need to bind it to a real host's addresses and port**s - nous avons encore besoin de le lier aux adresses et aux ports réels
a pair - une paire,
to some extent - dans une certaine mesure,
analogous to - analogue à,
allowable - acceptable, possible,
must be bound - doivent être liés,
in order to - afin de,
underlying - sous-jacent,
there are three uses of - il y a trois utilisations de,
when binding a socket to - lorsqu'on lie un socket à,
is much simpler - est beaucoup plus simple,
an ASCII string - une chaine de caractères ASCII,
datapath association - association au chemin d'accès au répertoire,

to obtain - obtenir,
to result in - mener à, entraîner,
protocol specific and remote server address - l'adresse du serveur distant spécifique de protocole,
backlog - compteur de demandes de connexions,
to fork - bifurquer, se réproduire,
newly connected - nouvellement connecté,
it means the number of consecutive connection calls to be allowed by - il signifie le nombre acceptable d'appels consécutifs de connect,
takes the first connection request on the queue - prend la première demande de connexion dans la file d'attente,
to spawn - étaler, (ici) générer,
all four - tous les quatres,
flag - drapeau,
out-of-band data - donnée hors piste (hors canal de base)
to peek - jeter un coup d'oeil furtif ,
bypass routing - contourner (le protocole) de routage,
data move - déplacement de données,
host short - un entier court (16 bits),
host long - un entier long (32 bits),
dest address - adresse destination,
dotted Internet address - adresse Internet avec les codes des octets séparés par points,
which are not necessarily ended - lesquelles (chaines) ne sont pas nécessairement términées,
IPC device - InterProcess Communication device - dispositif de communication entre processus,
he wants to read - (lequel) il veut lire,
the client and the server reside at the same Unix kernel - les processus client et serveur résident sous le contrôle du même noyau UNIX,
previously loaded into client_s structure - chargé précédemment dans la structure client_s,
along with - avec,
after the data has been sent , the server ends execution and exits - après avoir envoyé des données, le serveur termine l'exécution et sort,

Internet addressing - *adressage type Internet,*

anyway - en tous cas,
unused - non utilisé,
must be known to the client - doit être connue du client,
should be greater than - devrait être plus grand que,
setting the port to zero causes the system - l'initialisation du numéro de port à zéro amène le système à attribuer,
to look for - chercher,
support several library routines - fournissent plusieurs routines de bibliothèque,
let us consider a case - considérons un cas,
future datagrams - datagrammes à venir,
hard-coded - (ici) sous forme de constantes,
the steps required by a client to start a dialogue - les pas dont le client a besoin pour commencer le dialogue,
below we provide some utility functions that handle - ci-dessous nous apportons quelques fonctions utilitaires qui gérent,
we shall try to find out - nous allons essayer de rétrouver,
to look for - chercher ,
failure - échec,
otherwise - autrement, sinon,
to broadcast - diffuser,
must be set to 1 - doivent être positionnés à 1,
this time - cette fois-ci,
sends as the reply - envoie comme réponse,

must bind its - doit lier son,
an externally known - connue à l'extérieur,
to start with - commencer par,
out-of-band - hors piste, hors canal principal,
byte-stream service - service à flot d'octets,
boundary - borne, limite,
wants to disable - veut invalider, empêcher,
urgent data - donnée urgente,
keyboard - clavier,
is termed - est appelé,est nommé
expedited - expressée,
receiving end - du coté du récepteur,
to appear as - apparaître comme,
channel - canal,
band - bande, piste,
then passes it to next lower layer - ensuite il le transmet au niveau immédiatement inférieur,
terminal flow control characters - caractères de contrôle de flot de données à partir d'un clavier,
might have been buffered - aurait pu être bufférisée,
single-byte - octet simple,
because of the buffering operations at the receiving end - à cause des opérations de bufferisation sur le terminal de réception,
is slightly more complicated - est un peu plus compliqué,
kills itself by sending the corresponding signal - se tue par l'envoi du signal correspondant,
main stream - flot principal,
we do not need the urgent data to be immediately signalled - nous n'avons pas besoin que la donnée urgente soit immédiatement signalée,
be signalled - être signalé,
separate Unix kernels - noyaux Unix séparés,
introduced previously - qui a été introduit précédemment,
built on sockets - construites sur les sockets,
combined usage - utilisation combinée,
essential internetwork protocols - protocoles essentiels de la communication inter-réseaux,
in order to control the operations related to buffers' size - afin de contrôler les opérations concernant la taille de tampons,