Réseaux 1

Emmanuel Jeandel

13/03/2023

Table des matières

Préliminaires

Pour en savoir plus sur le cours, on pourra consulter les ouvrages de
- James F. Kurose et Keith W. Ross, *Analyse Structurée des Réseaux*, Pearson Education.
- Andrew Tanenbaum, *Réseaux*, Éditions Pearson.
- Douglas E. Comer, *Réseaux et Internet*, Campus Press Référence
- Richard Stevens, *TCP/IP illustré*, Vuibert Informatique

Tous ces livres sont disponibles à la bibliothèque universitaire. Le premier est particulièrement conseillé.

Outre les livres, on peut avoir besoin, pour si on veut bien comprendre un protocole, et en particulier tous les cas particuliers qu'il n'est pas possible de décrire dans un cours, d'aller voir les standards et les normes. Il existe un nombre important d'organismes de normalisation :

- L'ISO (organisation nationale pour la standardisation) et l'ITU-T (Union Internationale des Telecommunications - secteur Telecommunication). Il s'agit de deux organismes différents, mais, dans le domaine qui nous intéresse, tous les standards importants ISO nécessaires sont également disponibles sous la forme de recommendations de l'IUT-T. Dans le cadre de ce cours, on s'intéressera principalement à la description du modèle par couche (voir chapitre suivant) qui correspond aux recommandations X.2 ?? disponibles ici et à différents standards ISO. A noter que les recommendations sont disponibles en français.
- L'IETF (Internet Engineering Task Force) , qui publie régulièrement les RFC, ces documents qui décrivent en général très précisément tous les protocoles utilisés sur Internet. On peut trouver grâce à un moteur de recherche une traduction en français des documents les plus importants. Dans le cadre de ce cours, les RFCs seront utiles pour bien comprendre le fonctionnement de certains protocoles dans les couches hautes (mais il y a aussi beaucoup d'autres RFCs)
- L'IEEE (Institute of Electrical and Electronic Engineers). La plupart des normes qui nous intéressent sont disponibles gratuitement, sous réserve de créer un compte sur le site, à la page suivante. Les documents qu'on pourrait être amené à lire sur ce site contiennent des descriptifs très techniques de différentes technologies (Ethernet, Wifi, courant porteur, etc), et peuvent être un utile complément pour les couches basses.

Il n'est bien entendu pas demandé de consulter ces sites pour bien comprendre le cours, il s'agit essentiellement de compléments d'information qui peuvent permettre de comprendre en détails certains choix techniques. A la fin de chaque chapitre sera indiqué les différents documents qu'il peut être intéressant de lire pour en savoir plus.

1 Modèles par couche

1.1 Les modèles

Le modèle par couche est une modélisation d'un réseau, qui peut être une modélisation abstraite (le modèle OSI) ou une modélisation concrète (le modèle TCP/IP).

L'idée d'un modèle de couche est de représenter le fonctionnement d'un réseau, et les différentes tâches à effectuer, par un certain nombre de couches, numérotées de 1 à n, de sorte que :

— Chaque couche ajoute des fonctionnalités à la couche précédente ;
— Une couche offre des services à la couche supérieure, et utilise les services de la couche inférieure ; Ce sont d'ailleurs les seules couches avec lesquelles elle peut communiquer ;
— Une couche s'abstrait au maximum des couches inférieures : il n'est pas nécessaire de savoir comment la couche inférieure est implémentée, sauf quelques exceptions ;
— La couche de niveau n d'une machine dialogue avec une couche du même niveau n d'une autre machine : elles s'échangent des *unités de données* et ce par l'intermédiaire de ce qu'on appelle un *protocole*.

Le modèle OSI (Open Systems Interconnection) est un modèle proposé par l'association ISO (International Organization for Standardization). Il s'agit d'un modèle théorique du fonctionnement du réseau, mais ce n'est pas une norme : il est très possible qu'un réseau ne suive pas ce modèle.

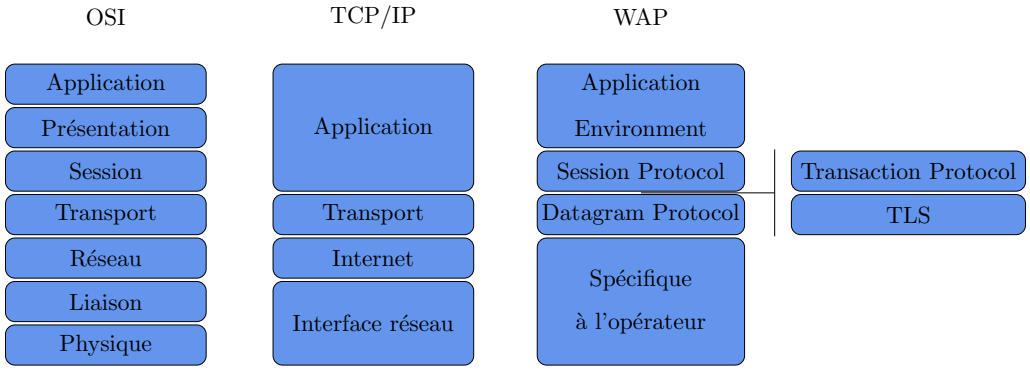


FIGURE 1.1 – Le modèle théorique OSI et deux modèles concrets.

La Figure 1.1 donne ainsi l'exemple du modèle OSI et de deux modèles concrets : le modèle TCP/IP (RFC 1122) qui est le modèle standard qu'on pourrait appeler le modèle d'Internet, et le modèle WAP, l'ancêtre de la 3G sur les téléphones portables. C'est la dernière fois qu'on parlera du modèle WAP : Dans la suite du document, on se concentrera principalement sur certaines couches du modèle OSI, en expliquant plus particulièrement comment ces couches sont mises en oeuvre dans le modèle TCP/IP.

1.2 Les couches du modèle OSI

Donnons ici une très brève description des couches du modèle OSI. Le modèle réel utilisé actuellement est le modèle TCP/IP, qui a tendance à confondre certaines couches. On se concentrera dans ce cours

sur les couches communes avec le modèle TCP/IP, et on ne parlera quasiment pas des couches Session et Présentation.

1.2.1 Couche 1 - Couche Physique (Physical Layer)

La couche physique permet la transmission d'une suite de bits d'un point A à un point B relié par un medium, qui peut être un cable, ou l'atmosphère (par exemple dans le cas d'un réseau radio, Bluetooth ou Wifi).

A ce niveau, il y a deux types de transmissions possible :

— Liaison point-à-point entre deux extrêmités : tout ce qui part d'une extrêmité arrive à l'autre extrêmité, et réciproquement

— Liaison Multipoint entre plusieurs machines : De la même façon que précédemment, tout ce qui part d'un machine arrivera à tous les machines

Cette couche n'offre aucune garantie de fiabilité ou d'efficacité : on peut avoir des erreurs de transmission (liées à des parasites par exemple, ou lorsque deux machines essaient de parler simultanément), les messages peuvent se dupliquer ou se perdre. La seule garantie en théorie est que les bits sont restitués dans l'ordre où ils ont été envoyés.

L'unité de donnée de la couche physique est donc le bit (ou une suite de bits).

1.2.2 Couche 2 - Couche Liaison (Data Link Layer)

La couche liaison ajoute deux fonctionnalités à la couche physique :

— La fiabilité. Les messages sont transmis sans erreurs, sans pertes et sans duplication

— L'adressage des liaisons de données : Chacun des protagonistes a un identifiant, qui permet de le repérer de façon unique dans le medium : on peut donc s'assurer pour qu'un message n'arrive qu'à la personne qui doit le recevoir, alors que ce n'est pas le cas dans la couche physique [1].

L'unité de donnée de la couche liaison est la *trame*.

Les deux couches 1 et 2 sont rassemblées en une seule couche dans le modèle TCP/IP.

1.2.3 Couche 3 - Couche Réseau (Network Layer)

La couche 2 (couche liaison) ne permet les communications qu'entre deux machines partageant le même medium physique, par exemple sur le même réseau filaire. On parle d'un *réseau local*. Les couches 3 et 4 offrent des fonctionnalités similaires aux couches 1 et 2, mais non pas au niveau d'un réseau, mais au niveau de l'*interconnexion de réseaux*.

Plus particulièrement, la couche réseau permet la transmission de données d'un point A à un point B en utilisant plusieurs points intermédiaires comme relai.

Commme pour la couche 2, on a maintenant une notion d'adresse réseau, qui est comprise par tous les points intermédiaires et qui permet d'identifier un point de façon unique dans toute l'interconnexion de réseaux (concrètement, dans tout Internet).

Chacun des points relais va transmettre les messages à d'autres points relais. On parle de *routage*, et ces points relais sont souvent appelés *routeurs*.

Même si le service sur chacun des media formant l'interconnexion des réseaux a été rendu fiable par la couche 2, le service n'est plus fiable au niveau de la couche 3 : Les messages peuvent s'égarer, et divers bouts d'un message peuvent arriver dans un ordre différent.

L'unité de donnée de la couche réseau est le *paquet*.

1. A noter que, dans certaines spécifications, la couche liaison est séparée en 2 : la couche MAC et la couche LLC. La séparation que nous faisons ici n'a *rien à voir* avec la distinction MAC/LLC. L'intégralité, ou presque, de ce que nous présentons correspond à des tâches realisées par la sous-couche MAC. En effet, la plupart des fonctionnalités de LLC ne sont pas utilisées par Ethernet II, qui est le modèle principal utilisé en pratique.

1.2.4 Couche 4 - Couche Transport (Transport Layer)

Intuitivement, la couche 4 offre les mêmes services au dessus de la couche 3, que la couche 2 offrait au dessus de la couche 1.

La couche transport permet en particulier la transmission fiable de données d'un point A à un point B.

Elle permet également le multiplexage, ce qui permet à deux processus sur la même machine de communiquer sur le même réseau (par exemple avec le même destinataire)

L'unité de donnée de la couche transport est le *segment*[2].

1.2.5 Couche 5 - Couche Session (Session Layer)

La couche session est la première des couches applicatives. Elle permet d'établir une relation entre deux applications distantes souhaitant coopérer. Elle permet en particulier la reprise en cas d'erreur (typiquement la coupure du réseau pendant un certain temps), en réinitialisant si nécessaire la connexion à un état précédent, souvent appelé point de contrôle (checkpoint).

1.2.6 Couche 6 - Couche Présentation (Presentation Layer)

La couche présentation permet la représentation des données (entier, flottants, etc) de façon universelle. Elle affranchit la couche application des problématiques de représentation des données, de compression et de chiffrement.

1.2.7 Couche 7 - Couche Application

La couche application est la dernière couche. C'est celle dans laquelle vivent les processus applicatifs, et dans laquelle ils collaborent avec l'environnemetn réseau.

Les 3 couches sont réunies en une seule couche, la couche Applications dans le modèle TCP/IP. Dit autrement, chaque protocole de la couche application (telle que HTTP (Web), SMTP (email), etc) doit se charger lui même de définir comment les données sont codées, comment gérer la situation en cas de coupure réseau, etc.

1.3 L'encapsulation de protocoles

Dans la définition d'un modèle de couches, une couche n ne peut communiquer qu'avec les couches $n-1$ et $n+1$.

Considérons donc maintenant ce qui se passe lorsque deux protocoles de couche 4 (par exemple) situés sur deux machines A et B différentes veulent dialoguer :

- La couche 4 de la machine A prévient la couche 3 de la machine A qu'elle veut dialoguer avec la machine B
- La couche 3 en prend note, et transmet les informations nécessaires à la couche 2
- La couche 2 en prend note, et transmet les informations nécessaires à la couche 1
- La couche 1 envoie le message sur le support physique
- Le message transite sur plusieurs medias physiques, passe éventuellement par plusieurs points relais et arrive sur couche 1 de la machine B.
- La couche 1 voit qu'il s'agit d'un message pour une couche supérieure et passe les informations utiles à la couche 2
- La couche 2 voit qu'il s'agit d'un message pour une couche supérieure et passe les informations utiles à la couche 3
- La couche 3 voit qu'il s'agit d'un message pour une couche supérieure et passe les informations utiles à la couche 4

2. On utilise quelquefois également le mot *datagramme*.

— La couche 4 découvre que la machine A veut lui parler et agit en conséquence.

Cette règle du jeu se manifeste dans les protocoles et dans l'encapulsation de protocoles : Un message envoyé par la couche n contient en général deux choses : des informations la concernant, et un message qu'elle doit transmettre en général à la couche supérieure.

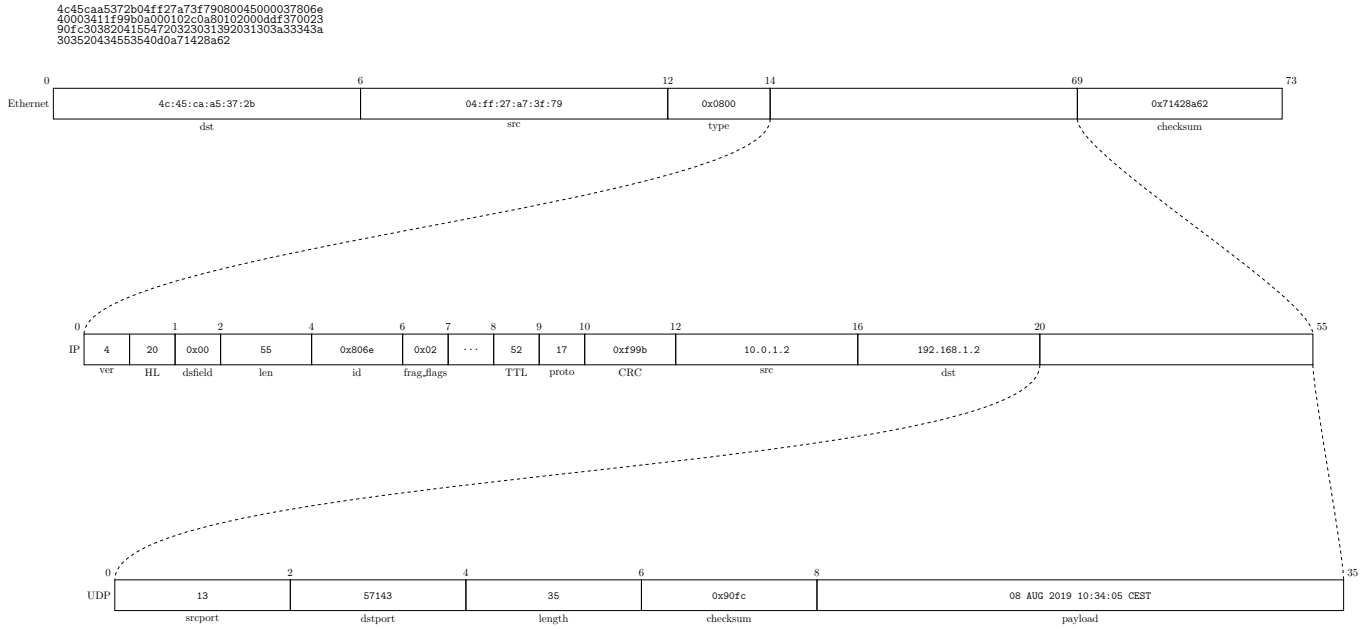


FIGURE 1.2 : Un exemple d'encapsulation

Un exemple d'un tel message est donné à la Figure 1.2. Il s'agit d'un exemple réel, basé sur le modèle TCP/IP et non pas sur le modèle théorique OSI.

On trouve tout au haut la série de bits qui est envoyé par une machine, codé en hexadécimal pour plus de simplicité. Du point de vue de la couche 1, il ne s'agit que d'une suite de 71 octets.

Cependant, du point de vue de la couche 2, et plus exactement de protocole Ethernet, cette série de bits est une *trame*, et est constituée de 4 paramètres (dst, src, type, checksum) propres au protocole Ethernet et d'une donnée de 55 octets (les octets `4500…0d0a`). Grâce aux 4 paramètres, la couche 2 sait que le message (plus exactement le paquet) encapsulé utilise le protocole IP, protocole de couche 3. Les 55 octets sont donc transférés au module IP de la couche 3.

Du point de vue du protocole IP de la couche 3, ces 55 octets contiennent plusieurs paramètres et contient un message de 35 octets. Les paramètres permettent au module IP de savoir que le message (plus exactement le segment) encapsulé utilise le protocole UDP, protocole de couche 4. Les 35 octets sont donc transférés au module UDP de la couche 4.

Du point de vue du protocole UDP, ces 35 octets contiennent plusieurs paramètres et un message de 27 octets. Ce message de 27 octets sera donné directement à la couche application.

Notons que les deux derniers octets, circulant sur le fil et faisant intégralement partie du protocole Ethernet, ne sont en général pas visualisables par les outils classiques comme `wireshark` ou `tcpdump`. C'est pour cela que toutes les autres trames qui seront montrées dans ce cours ne contiendront pas ces deux octets.

Il est possible, dans de rares circonstances qu'un protocole de niveau n encapsule des données qui ne soient pas de niveau $n + 1$. Un des exemples importants, qu'on peut voir en TP, est l'exemple du

protocole ICMP qui est un protocole de couche 3 encapsulé dans un protocole de couche 3.

Un autre exemple important est la situation lors de l'usage d'un VPN. Un logiciel de type VPN permet de créer un réseau virtuel. On encapsule dans ce cas les données d'un réseau à l'intérieur d'un autre réseau. Cette mise en abyme est illustrée à la Figure 1.3.

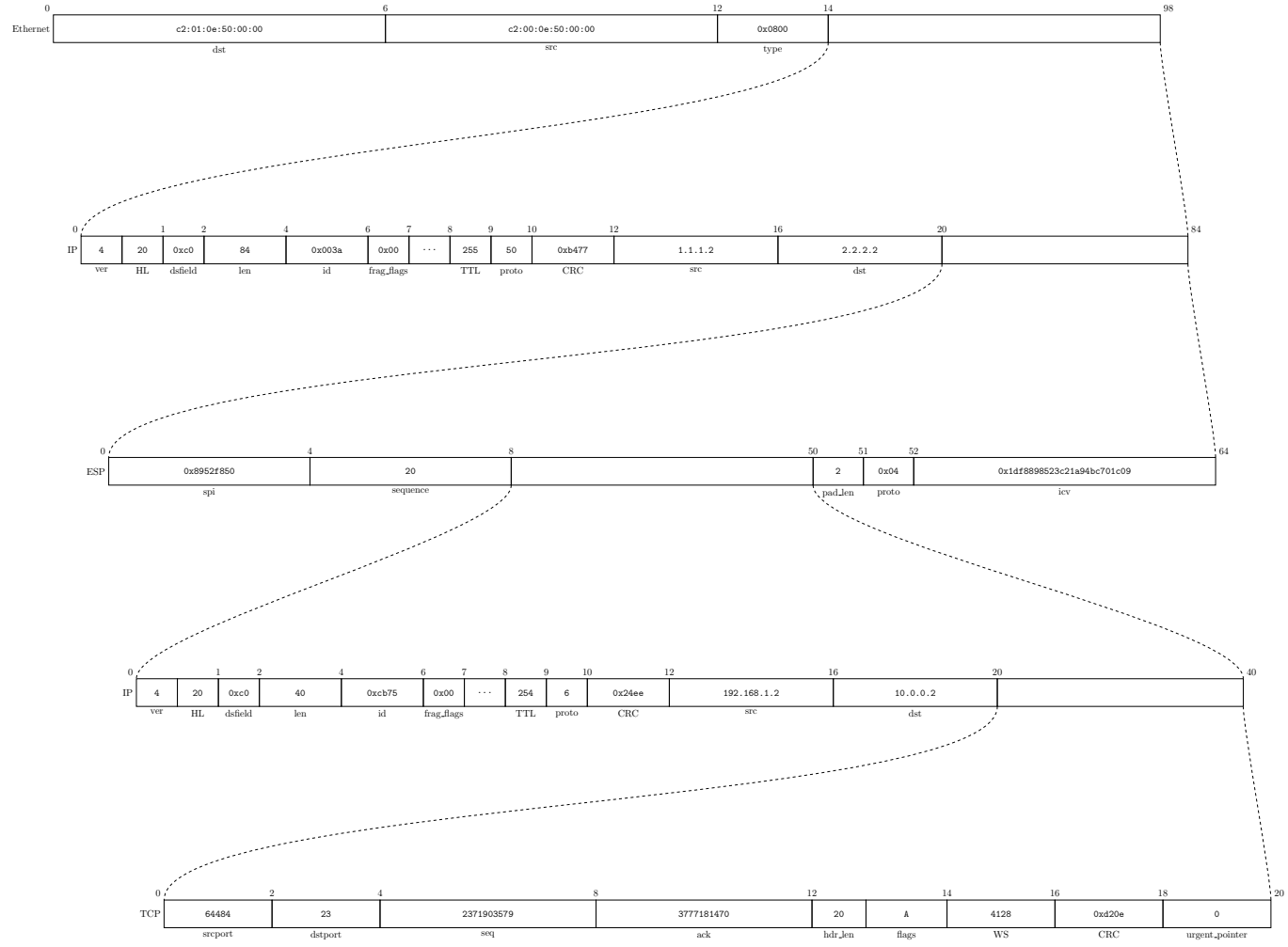


FIGURE 1.3 : Un exemple d'encapsulation avec mise en abyme. Lorsque le paquet a été envoyé, il s'agissait d'un paquet de la machine `192.168.1.2` vers la machine `10.0.0.2`. Ce paquet est acheminé à travers un VPN et, de l'extérieur, apparaît comme un paquet provenant de `1.1.1.2` et à destination de `2.2.2.2`.

Notes

A lire pour en savoir plus :

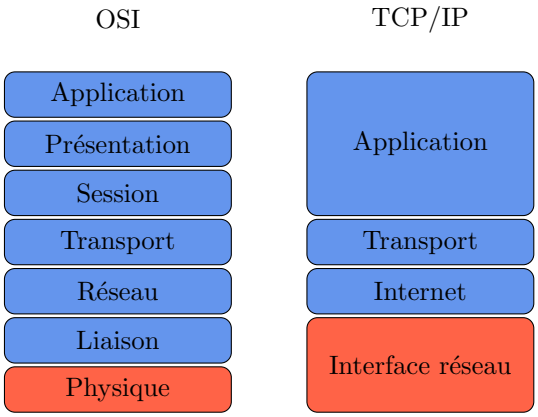
— La recommendation X.200 de l'UIT-T (qui explique le modèle OSI)

— Les RFC1122 et RFC1123 qui décrivent le modèle TCP/IP.

— Pour comparaison avec d'autres modèles :

— le document WAP-210-WAPArch-20010712-a qui décrit le modèle WAP ;
— Le document de Novell qui décrit IPX/SPX (voir en particulier la page 12) ;
— Le livre *Inside Macintosh : Networking* et plus particulièrement la page 18 de l'introduction, qui explique le modèle Appletalk.

2 Couche physique



Introduction

On commence naturellement par la première couche du modèle OSI, la couche physique. Du point de vue de TCP/IP, cette couche est cachée à l'intérieur de la couche interface réseau, qui correspond à ce cours, et au cours suivant.

Comme dit dans l'introduction, le but de cette couche est la transmission de bits dans un medium. La problématique qu'il faut régler est donc décider comment on "code" un 0 et un 1 dans un medium. On verra que la réponse dépend du type exact du medium (physique dans le cas des cables, ou air dans le cas du wifi)

Attention, la couche physique se contente d'envoyer des bits d'un bout à l'autre d'un medium mais ne vérifie pas que les bits sont arrivés : Comme on le verra, c'est un des rôles de la partie liaison.

Il est à noter que ce chapitre s'adresse à un public d'informaticiens, et non pas à un public d'étudiants en science pour l'ingénieur. Il se concentre donc uniquement sur la partie "codage", et laisse de côté quasi entièrement toute la partie traitement du signal, nécessaire dès qu'on veut rentrer un peu dans les détails.

2.1 Caractéristiques

Avant d'expliquer comment coder, expliquons les différentes caractéristiques d'une transmission sur un medium.

Spécifions d'abord les media dont on parle. Pour Internet, on s'intéresse principalement à la transmission dans des cables (en particulier des cables Ethernet) mais aussi des fibres optiques, ou dans l'air, lorsqu'on s'intéresse aux ondes wifi ou radio.

On peut diviser ces différents media en deux. Les premiers sont ce qu'on appelle des média *guidés* : le signal qui part d'un bout du medium est dirigé vers l'autre bout du medium, car il doit suivre le cable, ou la fibre. Ce n'est bien entendu pas le cas dans l'air.

Topologie

Cette différence impose en particulier des différences dans la teneur des échanges : Dans le cas d'un cable, on parle d'une transmission *point-à-point* : il y a un émetteur et un récepteur. Dans le cas d'une transmission dans l'air, on parle d'une transmission *multipoint*. Une telle transmission peut-être *centralisée* (un des interlocuteurs sur le medium est privilégié : les autres ne peuvent pas parler entre eux, mais uniquement avec l'interlocuteur privilégié. C'est ce qui se passe dans une salle de cours normalement, ou avec les téléphones) ou *décentralisée* (tout le monde peut parler avec tout le monde. C'est le cas de Bluetooth).

Organisation des échanges

Certains des media permettent des communications bidirectionnelles : les deux interlocuteurs peuvent parler en même temps. On parle de medium *duplex*. C'est le cas des cables Ethernet. D'autres media permettent uniquement un mode *half duplex*, ce qui signifie que chacun des interlocuteurs peut parler à l'autre, mais pas en même temps. C'est le cas du Wifi. Enfin, dans un medium *simplex*, seul l'un des deux interlocuteurs peut parler à l'autre, mais la communication dans l'autre sens n'est pas possible. C'est le cas par exemple des signaux pour la TNT, ou de la communication entre un clavier et un ordinateur.

Notons que techniquement un cable ethernet contient deux (en fait même plus) paires torsadées : une des paires permet à A de parler à B et l'autre permet simultanément à B de parler à A. Donc on peut voir un cable ethernet comme deux media simplex qui ensemble forment un medium duplex.

Autres caractéristiques

Parmi les autres caractéristiques, on peut noter les transmissions *parallèles* (plusieurs bits envoyés à la fois) vs *séquentielles* (un seul bit envoyé à la fois) et les transmissions *synchrones* (les deux protagonistes partagent une horloge) ou *asynchrones*. On reviendra sur cette dernière différence un peu plus tard.

Débit vs Latence

Deux données importantes d'une transmission sont le débit et la latence. Souvent, les deux sont confondus alors qu'ils jouent des rôles très différents.

La *latence*, souvent exprimée en secondes, représente le délai entre l'envoi d'un message et la réception du message. Le *débit*, souvent exprimés en octets/seconde, représente le nombre de bits envoyés (ou reçus) par seconde.

Intuitivement (et si la latence et le débit restent constants) la latence représente le temps qu'il faut pour recevoir le premier octet et le débit est la vitesse à laquelle les suivants arrivent.

> **Exemple**
>
> Prenons l'exemple d'une connexion à 25 Mbps, c'est à dire 25 mégabits par secondes, entre Paris et Nancy, et une latence de 6 ms (millisecondes).
> Supposons télécharger 5ko, c'est à dire 40kb. Le premier bit arrivera au bout de 6ms, et les suivants à la vitesse de 25 mégabits par seconde, donc arriveront en $\frac{40000}{25000000} = 0.0016$ secondes. Le temps total écoulé entre le début de l'émission du fichier chez l'envoyer et la fin de la réception du fichier chez le destinaire sera donc de 7.6ms.
> Supposons télécharger 1Mo, c'est à dire 8Mb. Le premier bit arrivera au bout de 6ms, et les suivants à la vitesse de 25 mégabits par seconde, donc arriveront en $\frac{8000000}{25000000} = 0.32$ secondes. Le temps total écoulé sera donc 327ms.

Les deux paramètres sont critiques, mais jouent des rôles différents. Quand on télécharge un fichier, le paramètre important est le débit, car c'est celui qui identifie clairement le temps avant réception du fichier (la latence est négligeable). Quand on joue à un jeu en ligne, ou qu'on fait une visioconférence, la latence

(aussi appelé “lag” en anglais) devient importante : avec une latence d'une seconde, votre interlocuteur entend ce que vous dites une seconde après que vous l'ayez prononcé, ce qui rend la communication très difficile.

2.2 Transmission en bande de base

Nous allons maintenant examiner les différentes façons de transférer des bits dans un cable, ou dans l'air.

La transmission en bande de base correspond au cas le plus simple, où on convertir directement des bits en tension sur un cable.

2.2.1 NRZ

Le premier codage facile qu'on peut obtenir est le codage NRZ (pour “No Return to Zero”) qui est défini de la façon suivante :

— +5V pour 1
— -5V pour 0
— 0V quand il n'y a pas de signal

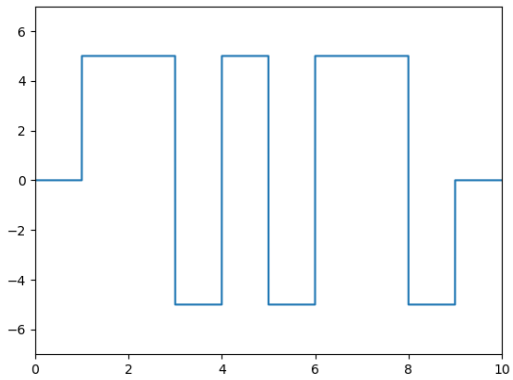


FIGURE 2.1 – Codage NRZ de la séquence 11010110. En abscisse le temps(en millisecondes par exemple), en ordonnée la tension electrique (en V)

La Figure 2.1 montre l'exemple du codage du mot 11010110. Pour pouvoir décoder un tel signal, il faut être précis sur comment fonctionne NRZ et dire, par exemple, que les bits sont envoyés toutes les millisecondes. (plus exactement : Quand on voit un 0 on émet -5V pendant une milliseconde. Quand on voit un 1 on émet 5V). Pour décoder, il suffit alors de regarder la tension sur le cable toutes les millisecondes.

Avant de passer à des codages différents, on va expliquer pourquoi ce codage très simple n'est pas toujours suffisant.

D'abord il faut être précis sur le décompte du temps : si on mesure la tension toutes les 0.9999 millisecondes, ou toutes les 1.0001 millisecondes, on va avoir un décalage entre ce qui a été envoyé et ce qui a été lu.

La Figure 2.2 montre les conséquences de ce décalage. Dans le premier cas (lecture du bit toutes les millisecondes), on retrouve bien le mot 11010110 qui a été codé. Dans le deuxième cas (lecture un tout petit peu trop rapide), on lit le mot 110101110. Il y a donc une erreur de transmission.

La seule solution à ce problème est de s'arranger pour que les deux machines (émetteur et récepteur) synchronisent les horloges. On peut le faire en transmettant, en plus du signal contenant les données,

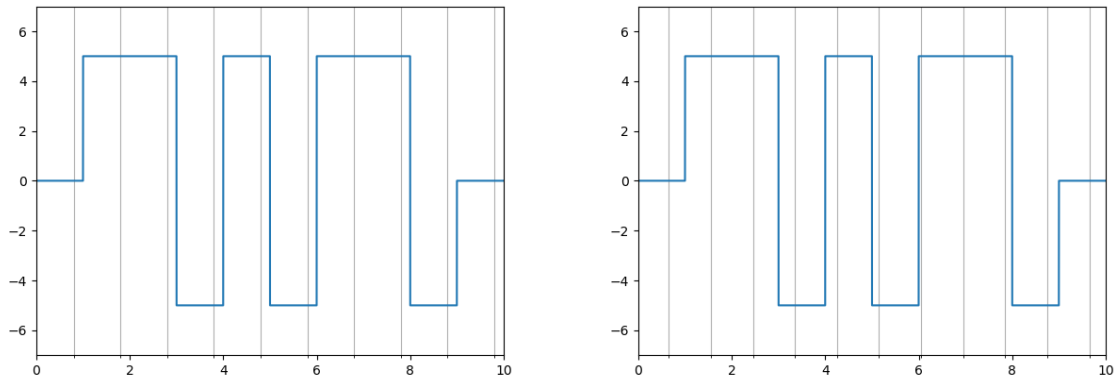


FIGURE 2.2 – Décodage d'un signal NRZ. Si les horloges ne sont pas parfaitement synchronisées (si l'une est plus rapide que l'autre par exemple), on voit qu'on peut mal décoder le signal.

un signal d'horloge, ou on peut s'arranger pour qu'on puisse retrouver le signal d'horloge dans le signal envoyé.

Un deuxième problème est illustré en Figure 2.3. Les différentes atténuations du signal (et la façon dont les composants d'un circuit électronique fonctionnent) ont pour conséquence que le signal reçu peut avoir une amplitude différente et que la tension 0V peut changer de position : Quand on mesure -0.5V, est-ce qu'il faut supposer qu'il s'agit d'une tension de 0V qui a légèrement été modifiée, ou d'une tension de -5V qui a subi une modification d'amplitude et un décalage ? Ce problème doit être évité, et conduit à utiliser des codages qui garantissent qu'il est n'est pas nécessaire de connaître exactement les tensions, mais juste de savoir si des tensions sont suffisament différentes. Dit autrement, il faut qu'on puisse retrouver les bits codés sans connaître l'échelle sur l'axe des ordonnées.

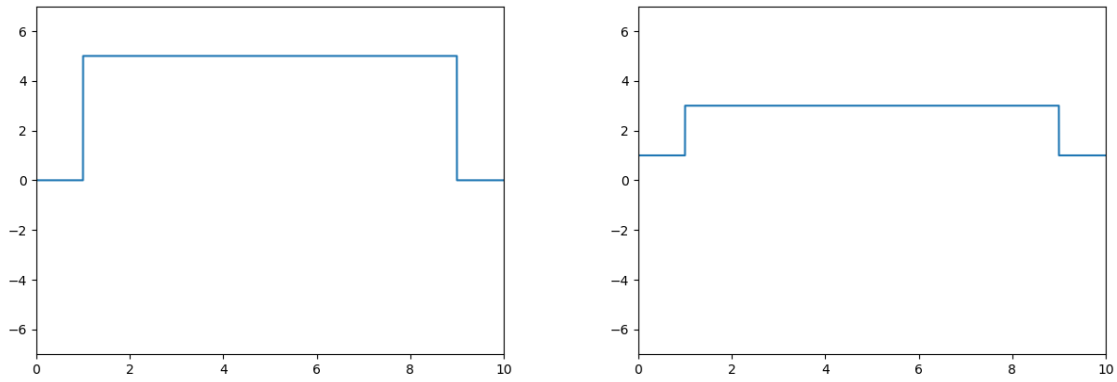


FIGURE 2.3 – Décalage d'amplitude et du zéro entre ce qu'on envoie et ce qui est reçu

En mettant ensemble les deux problématiques, on s'aperçoit que le codage NRZ n'est pas suffisant et qu'il nous faut un codage des 0 et 1 qui a les propriétés suivantes

— On peut déduire de ce qu'on reçoit le signal d'horloge
— On doit s'arranger pour que la partie positive du signal compense *à peu près* la partie négative (de façon à retrouver le 0)

A noter qu'il existe d'autres solutions pour retrouver le 0, qu'on verra plus loin.

2.2.2 Codage Manchester

Le codage Manchester est une solution possible aux problèmes. Le codage fonctionne de la façon suivante :

— Transition -5V/+5V pour 0
— Transition +5V/-5V pour 1

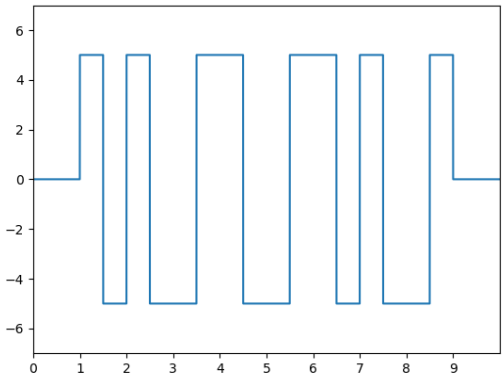


FIGURE 2.4 – Exemple du codage Manchester de 11010110

Un exemple est illustré Figure 2.4. Vérifions que les deux problèmes sont résolus. D'abord, on peut retrouver l'horloge : ce codage force des transitions, quel que soit les bits de départ, ce qui fait qu'on peut toujours ce resynchroniser en repérant les fronts montants et descendants.

De plus, ce codage vérifie qu'on a toujours autant de +5V que de -5V, ce qui garantit que la position du 0V n'est pas essentielle, et que les atténuations du signal n'ont pas d'importance.

2.2.3 Un peu de théorie du signal

Pour comprendre les restrictions supplémentaires qu'il est nécessaire d'apporter aux codages précédents, il est nécessaire de faire un peu de théorie du signal. En effet, avec ce qu'on a dit jusqu'à présent, rien n'empêche d'obtenir un codage avec un débit infini (plus exactement aussi grand qu'on veut), car il y a deux critères sur lesquels on pourrait jouer : on pourrait envoyer 1 bit toutes les millièmes de millièmes de millièmes de secondes, plutôt qu'un par milliseconde, et on pourrait aussi utiliser toute la panoplie de voltage plutôt qu'uniquement deux voltages différents.

Cependant, quand on regarde ce qui arrive typiquement chez le récepteur, on s'aperçoit dans la Figure 2.5 que le signal est très modifié lorsqu'il passe dans le medium.

Pour comprendre précisément ce qui se passe, il faut faire de la théorie du signal, ce qui dépasse un cours de L2 Informatique. On va uniquement fixer quelques bases ici.

L'idée essentielle est la suivante : tout signal sur un cable peut s'écrire comme un somme de sinusoïdes. Un exemple est donné dans la Table 2.1. Le comportement d'un medium sur un signal donné se déduit du comportement qu'il a sur les sinusoïdes : chaque sinusoïde (de différente fréquence) se retrouve atténuée (compression horizontale), et potentiellement retardée (décalage vertical)

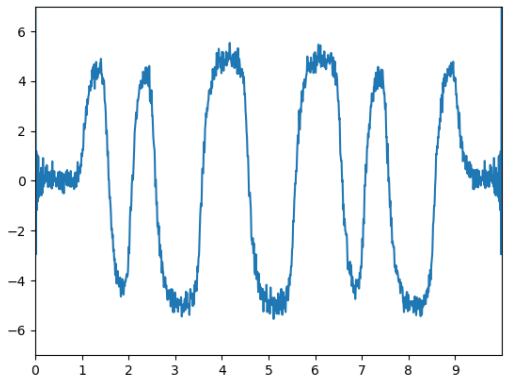
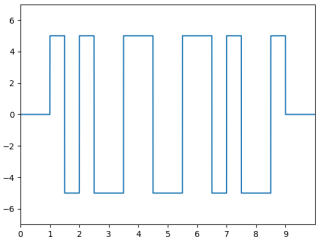
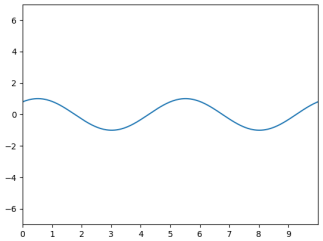
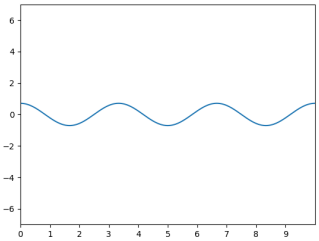
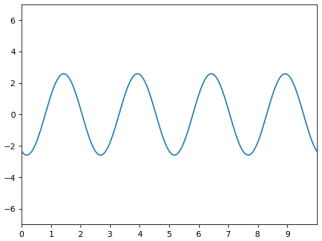
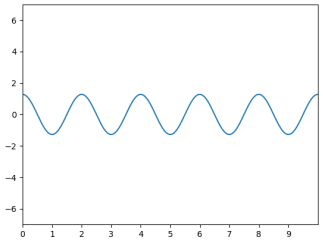
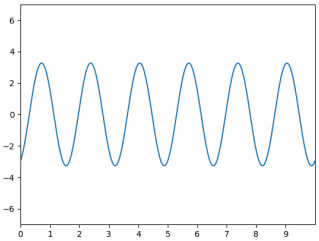
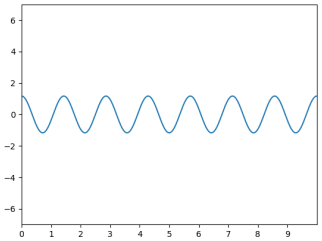


FIGURE 2.5 – Ce que pourrait recevoir le destinaire si on transmet le signal présent à la Figure 2.4.

TABLE 2.1 – Comment un signal se décompose en sinuoïdes

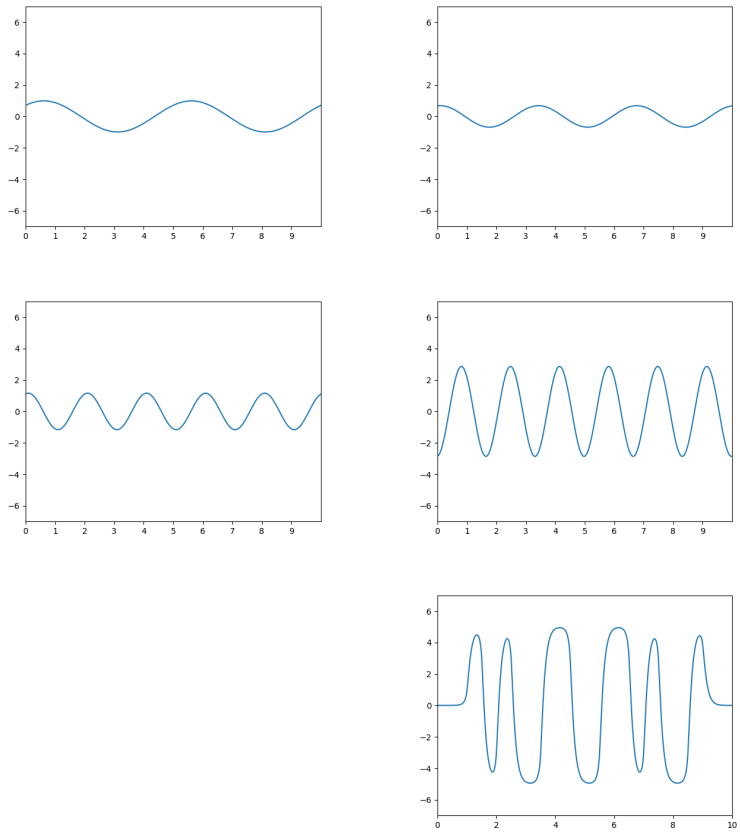
		
		
		

La Table 2.1 représente le signal initial, décomposé en sinusoïdes. En pratique, il y a bien plus que les 6 sinusoïdes dessinées.

La Table 2.2 représente le résultat : Chaque sinusoïde a été transformée. Pour trouver à quoi ressemble

le signal final, il suffit de recomposer les nouvelles sinusoïdes ensemble. On voit que le résultat est très différent de ce qui a été envoyé, même si chaque sinusoïde pris séparément n'a été que peu modifiée.

TABLE 2.2 – Comment chaque sinusoïde est modifiée, et le signal reconstitué.



Il est à noter en particulier que certaines sinusoïdes sont tellement atténuées que, en pratique, c'est comme si elles avaient disparu.

Dans un medium, on appelle *bande passante* la plus grande fréquence F tel que toutes les sinusoïdes de fréquence inférieure à F ne sont pas trop atténuées.

La bande passante réduit considérablement l'étendue des codages possibles : En effet, ça ne sert à rien d'émettre à une fréquence très supérieure à F puisque ces fréquences vont disparaître dans le medium.

Le théorème de Nyquist donne une borne sur le nombre maxium de valeurs qu'on peut coder :

Théorème 2.1 (Nyquist)**.** *Un signal de bande passante F est entièrement déterminé par la connaissance de $2F$ valeurs du signal.*

Dit autrement : on peut coder maximum $2F$ valeurs à la seconde dans un signal de bande passante F.

Techniquement ce théorème à lui seul n'interdit pas d'avoir des transmission de données à une vitesse très très grande : on ne peut pas dépasser une certaine fréquence dans l'envoi des données, mais rien n'empêche d'utiliser toute la gamme des voltages pour ce faire.

Ce n'est cependant évidemment par raisonnable, mais pour une autre raison, le bruit qui se produit dans les media. Ce bruit est reproduit et illustré à la Figure 2.5 vu précédemment : Sans le bruit, le signal ressemblerait au dessin en bas de la Table 2.2

La quantité de bruit est mesurée par le rapport *signal/bruit*, rapport entre la puissance du signal S et la puissance du bruit N.

Le théorème de Shannon donne une mesure précise des limites d'un canal :

Théorème 2.2 (Shannon)**.** *La capacité maximale (en bit/s) d'un canal de bande passante H et de rapport signal bruit S/N est*

$$H\log_2(1+\frac{S}{N})$$

Si le rapport S/N est exprimé en dB alors on obtient

$$0.332 \times H \times SNR$$

La problématique de transmission de bits dans un medium peut désormais être précisée : Il faut essayer de transmettre le maximum de bits à la seconde sachant que :

— On a une limite théorique sur la fréquence d'envoi
— On a une limite théorique sur le nombre de valeurs à chaque pas de temps
— Le codage doit être "balancé"

2.2.4 Codage MLT-3

Le codage MLT-3 est le codage utiliseé dans Ethernet 100 Mbps, donc c'est souvent celui utilisé lorsque vous connectez deux interfaces réseaux par un cable.

Le principe du codage est le suivant :

— On encode d'abord 4 bits en 5 bits en s'arrangeant pour ne pas avoir plus que trois symboles 0 consécutifs grâce à la table suivante :

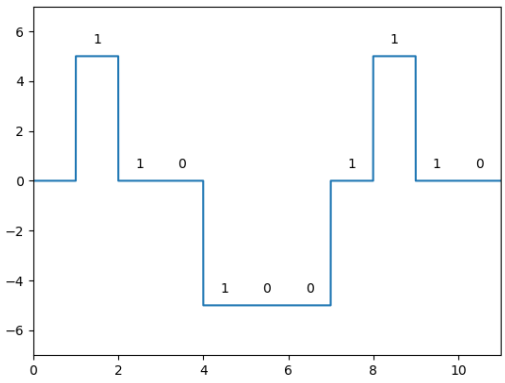
0000	0001	0010	0011	0100	0101	0110	0111
11110	01001	10100	10101	01010	01011	01110	01111

1000	1001	1010	1011	1100	1101	1110	1111
10010	10011	10110	10111	11010	11011	11100	11101

— On applique ensuite l'opération suivante :
  — 0 : on ne change pas la tension
  — 1 : on passe de -5V à 0 puis à 5V puis à 0, etc

Prenons l'exemple du mot 11000110.

En utilisant la table, on commence par convertir 1100 en 11010 et 0110 en 01110. Il faut donc maintenant convertir 1101001110 en signal, ce qui donne le résultat suivant :



Le codage MLT-3 est meilleur que le codage Manchester. En effet, pour un envoi du même nombre d'octets, la fréquence du signal est bien plus petite avec MLT-3 qu'avec Manchester : la fréquence représente intutivement le temps qu'il faut pour faire un cycle complet 0V+5V0V-5V0V. Dans MLT-3, on code 4 faux bits dans ce cyle, donc 3.2 vrais bits. Dans Manchester, on code un seul bit par cycle.

A fréquence donnée, on code donc 3.2 plus de bits avec MLT-3 qu'avec Manchester.

2.3 Transmission en large bande

Evoquons maintenant rapidement la transmission en large bande, qui est celle utilisée en autres par le Wifi et les ondes radios.

L'idée principale est de *moduler* un signal sinusoidal à une fréquence donnée f, appelée *porteuse*. Contrairement à la transmission précédente, on reste donc très proche d'une fréquence donnée, sans s'en éloigner. C'est très important pour la transmission dans l'air, ou certaines plages de fréquences sont utilisées pour autre chose (dont la voix humaine par exemple), ce qui garantit qu'on peut communiquer sans être dérangés par les autres transmissions, sous réserve qu'elles utilisent des fréquences différentes.

Il existe plusieurs techniques de modulation, représentées à la figure Figure 2.6.

— Coder 0 et 1 par deux amplitudes différentes (Modulation d'amplitude)
— Coder 0 et 1 par deux fréquences différentes (Modulation de fréquence)
— Coder 0 et 1 par deux phases différentes (Modulation de phase)

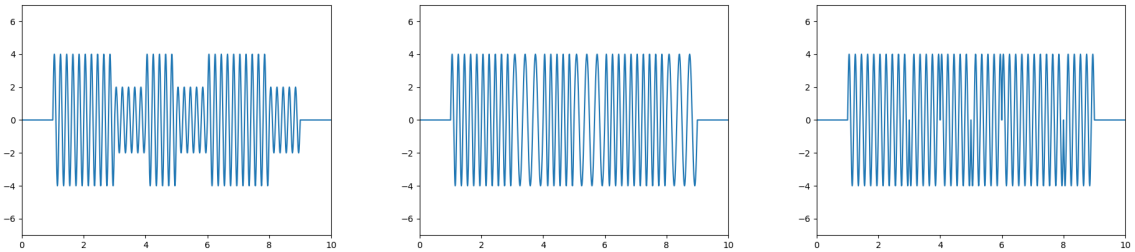


FIGURE 2.6 – Modulation d'amplitude, de fréquence, et de phase. Dans les 3 cas, le mot codé est le mot 11010110.

2.4 Equipement

Pour terminer, précisons les différents équipements fonctionnant dans la couche physique.

Les cables les plus utilisés actuellement sont des cables Ethernet. Ils sont composés de 4 paires torsadées. Une paire permet d'établir une communication simplex entre les deux extrêmités, ce qui permet de faire une communication duplex en utilisant deux paires. Dans les variantes sophistiquées d'Ethernet, les 4 paires sont utilisées.

Les cables ne permettent de relier que deux équipements donnés. Pour en relier plus, on peut utiliser des répéteurs et des concentrateurs.

Un répéteur, comme son nom l'indique, permet de répeter un signal. Intuitivement, il permet de connecter ensemble deux cables. Un concentrateur, ou *hub*, permet de connecter plusieurs cables ensembles, et de relayer ce qui vient d'un cable sur les autres cables.

Les hubs peuvent être passifs (se contentent de relayer) ou actifs (amplifient également le signal).

Intuitivement, un hub avec, par exemple, 4 ports a un comportement très simple : chaque cable contenant une paire torsadée qui émet, et l'autre qui reçoit, il suffit que le hub écoute sur les 4 paires torsadées qui émettent, et qu'il transmet tout ce qu'il écoute sur les 4 paires torsadées qui reçoivent (en veillant à ne pas retransmettre à l'émetteur évidemment).

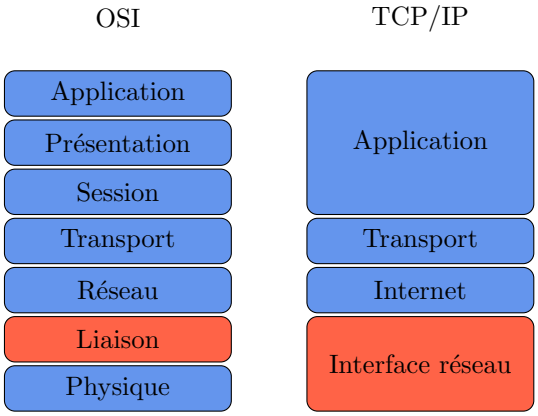
Que se passe-t-il si deux personnes émettent en même temps ? Il y a un problème, que le hub n'essaie pas de résoudre : ce sera entre autres le rôle de la couche liaison.

Notes

A lire pour en savoir plus :

— Les chapitres correspondant des livres mentionnés supra ;
— La recommandation X.211 de l'UIT-T (qui expliquent le rôle de la couche physique) ;
— La norme IEEE qui décrit Ethernet. Dans ce document de 5600 pages (!), on pourra regarder plus particulièrement la table 24-1 (page 823) qui décrit le codage 4B/5B utilisé dans MLT-3. Le codage en lui même n'est pas dans ce document, mais présent dans une autre norme.

3 Couche liaison



Introduction

Grâce à la couche physique expliquée dans le chapitre précédent, nous savons comment transférer une suite de bits entre deux machines.

Transférer des bits ne suffit pas, il faut les transformer en *messages*, ce que, au niveau de la couche liaison, on appelle des trames. C'est la première problématique que nous étudierons dans ce chapitre.

Ensuite, la couche physique n'offre que peu de garantie : elle essaie d'envoyer des bits d'un point A à un point B, mais ne garantit pas que les bits sont arrivés, ou qu'ils sont arrivés correctement. Il faut donc un ou plusieurs mécanismes pour vérifier que les bits sont arrivés (comment ?) et savoir quoi faire si ce n'est pas le cas (quoi ?) C'est la deuxième problématique que nous étudierons dans ce chapitre.

Enfin, une troisième problématique se présente, lorsqu'il y a plus de deux machines sur le medium (et même lorsqu'il n'y en a que deux pour certains media) : Si on prend l'exemple de la voix humaine dans une salle de classe, si tout le monde parle en même temps, c'est la cacophonie. Il faut un mécanisme pour gérer le temps de parole, et pour que les machines disent à qui elles s'adressent. Ces mécanismes seront étudiés dans le chapitre suivant, qui constituera la deuxième partie de la couche liaison [1].

3.1 Découpage en trames

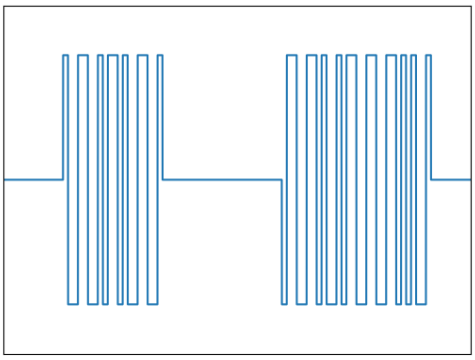
La couche physique nous permet une transmission brute de bits par un medium reliant une machine A à une machine B.

On veut convertir ces suites de bits en *trames*, d'une certaine taille. La première problématique qui se pose est de savoir comment délimiter ces trames, autrement dit, comment séparer les différents messages transmis.

1. A noter que, dans certaines spécifications, la couche liaison est séparée en 2 : la couche MAC et la couche LLC. La séparation que nous faisons ici n'a *rien à voir* avec la distinction MAC/LLC. L'intégralité, ou presque, de ce que nous présentons correspond à des tâches realisées par la sous-couche MAC. En effet, la plupart des fonctionnalités de LLC ne sont pas utilisées par Ethernet II, qui est le modèle principal utilisé en pratique.

3.1.1 Ethernet

Cette problématique ne se pose que si la couche physique ne résout pas déjà le problème. Dans le cas du codage Manchester, les 0 et les 1 sont codés par des transitions entre -5V et +5V. Autrement dit, une tension de 0V témoigne d'une absence de signal : le début d'une trame est donc quand la tension n'est plus à 0V, et la fin de la trame lorsqu'elle revient à 0V, comme illustré ci-dessous :



C'est ce mécanisme qui est utilisé dans Ethernet, et dans la norme IEEE 802.3 : Le récepteur se réveille pour traiter une nouvelle trame quand il voit un signal apparaître. Bien entendu, un tel mécanisme ne fonctionne pas complètement : il est possible de rater légèrement le début.

La solution choisie est de préfixer la suite de bits qu'on veut envoyer par 8 octets (64 bits) :

— 7 octets `10101010`

— 1 octet `10101011`

La présence de deux symboles `1` consécutifs permet donc de repérer le vrai début du message. L'intérêt d'une séquence aussi longue est qu'elle permet aussi de synchroniser l'horloge entre émetteur et récepteur.

3.1.2 Codage de début et de fin

Supposons n'avoir aucune aide de la part de la couche physique, et qu'elle nous fournisse une suite ininterrompue de 0 et de 1… Comment séparer cette suite infinie en trames ?

Pour expliquer un des codages un peu classique, on va d'abord raisonner en termes d'octets plutôt que de bits : la problématique est la même, mais elle est plus simple à expliquer.

Une solution simple dans ce cas serait d'indiquer le début et la fin du message par un caractère spécial, par exemple le caractère #.

Par exemple le message `Platypus` pourrait être envoyé de la façon suivante :

#	P	l	a	t	y	p	u	s	#

Mais que se passe-t-il si le message contient déjà un caractère # ? Par exemple, le message `Room #1349` deviendrait :

#	R	o	o	m	:	#	1	3	4	9	#

Qu'on interpreterait comme deux messages, et non pas un seul.

Une solution possible est de faire la chose suivante : les caractères `#` à l'intérieur du message sont préfixés par un autre caractère spécial, par exemple le caractère `!`, avec la convention suivante : tout caractère présent après le caractère `!` est repris tel quel, et non interprété.

Ainsi, le message `Room #1349` devient :

#	R	o	o	m	:	!	#	1	3	4	9	#

et le message `Yes! I Won` (qui contient un point d'exclamation) deviendrait :

#	Y	E	S	!	!		I		W	o	n	#

Un codage du même type, dans un format bits et non pas octets, est utilisé dans le protocole HDLC/PPP (RFC 1662).

— On code le début et la fin par le fanion `01111110`
  — Quand il y a deux messages consécutifs, on a un seul `01111110` au milieu
— Dans tout le texte, une suite de 5 caractères `1` consécutifs est remplacée par `111110`

Ainsi, pour envoyer `10110111010111111011011` on enverra plutôt `01111110`1011011101011111`0`1011011`01111110`

Réciproquement, essayons de décoder `01111110111001111101111101001011111101000011111110`.

— On commence par chercher les occurrences de 01111110 :

`01111110`11100111110111101001`01111110`1000`01111110`

— On en déduit qu'il y a deux messages
— Pour obtenir le premier message, on prend la chaîne `111001111101111101001` et on enlève le 0 après chaque bloc de 5 symboles 1, pour obtenir `11100111111111101001`
— Pour obtenir le deuxième message, on prend la chaîne `1000` et on enlève le 0 après chaque bloc de 5 symboles 1, pour obtenir `1000`.

3.2 Codes correcteurs d'erreur

Nous avons résolu une première problématique avec le paragraphe précédent : nous avons expliqué comment passer d'une couche physique qui transmet une suite de bits, en une couche liaison qui transmet des messages.

Le second problème que nous cherchons à résoudre maintenant est de savoir si le message a bien été transmis.

En effet, la couche physique n'offre aucune garantie que les bits envoyés arrivent bien. Des erreurs de transmission peuvent en effet se produire. Par exemple :

— Sur un cable Ethernet, la probabilité d'erreur est de l'ordre de 10^{-6} : un bit sur un million sera faux.
— Sur une fibre optique, cette erreur est de l'ordre de grandeur de 10^{-14}.
— Sur le WiFi, l'ordre de grandeur est de 10^{-5}.
— Si on transmet des bits par pigeon voyageur, la probabilité d'erreur est de 0.55.

En d'autres termes, si on transmet un fichier de 1 Mo (c'est à dire 8 millions de bits) sur un cable Ethernet, le fichier à l'arrivée aura en moyenne 8 bits faux (et donc sans doute 8 octets faux, si les 8 bits faux ne sont pas au même endroit).

Il faut donc un mécanisme pour :

— repérer les erreurs
— corriger les erreurs si possible
— redemander le message si jamais on ne peut pas le corriger.

Les mécanismes pour redemander les messages ne sont pas aussi simples qu'il n'y paraît, et seront examinés ultérieurement dans le cours sur la couche transport (ou une problématique similaire se pose.)

On se concentrera donc ici sur les deux premiers mécanismes, à travers le concept de codes correcteur d'erreur.

Les *codes correcteurs d'erreur* sont utilisés partout pour éviter les problèmes lors d'une transmission : les messages qui se perdent, les chiffres qu'on n'arrive plus à lire sur un code-barre, etc. Le principe est d'insérer de l'information redondante dans le message pour être sûr de réussir à le reconstituer même si on perd quelques bits, et même si quelques bits ont été mal transmis (1 à la place de 0, etc).

On va voir ici des codes très simples, qui permettent de corriger une erreur lors d'une transmission.

3.2.1 Premiers exemples

Code qui double

Un exemple de code très simple consiste à doubler chaque bit. Ainsi, au lieu d'écrire :

$$0 \cdot 0 \cdot 1 \cdot 0 \cdot 0 \cdot 1 \cdot 0 \cdot 1 \cdot 0$$

on écrira

$$00 \cdot 00 \cdot 11 \cdot 00 \cdot 00 \cdot 11 \cdot 00 \cdot 11 \cdot 00$$

On remarque plusieurs propriétés sur ce code :

— Si lors de la transmission, un (seul) bit est erroné, on peut s'en rendre compte. Par exemple, si on reçoit :

$$00 \cdot 01 \cdot 00 \cdot 11 \cdot 00 \cdot 11 \cdot 00 \cdot 00 \cdot 11$$

on s'aperçoit que le 3ème ou le 4ème bit est faux. Cependant, nous ne sommes pas en mesure de savoir lequel des deux est faux, ni de corriger l'erreur.

— Si lors de la transmission, deux bit sont erronés, on ne peut pas toujours s'en rendre compte. Ainsi dans l'exemple suivant, on s'en rend compte :

$$00 \cdot 01 \cdot 00 \cdot 11 \cdot 01 \cdot 11 \cdot 00 \cdot 00 \cdot 11$$

mais pas dans l'exemple suivant

$$00 \cdot 00 \cdot 00 \cdot 11 \cdot 11 \cdot 11 \cdot 00 \cdot 00 \cdot 11$$

il se peut en effet tout à fait que les bits 3 et 4 soient erronés, mais il n'y a aucune manière de s'en rendre compte.

— Si un bit est perdu dans la transmission, on peut retrouver lequel, ce qui se voit sur l'exemple suivant :

$$00 \cdot 00 \cdot 0x \cdot 11 \cdot 11 \cdot 11 \cdot 00 \cdot 00 \cdot 11$$

Code qui triple

On peut également tripler chaque bit, et on transmettra

$$0 \cdot 0 \cdot 1 \cdot 0 \cdot 0 \cdot 1 \cdot 0 \cdot 1 \cdot 0$$

par le message

$$000 \cdot 000 \cdot 111 \cdot 000 \cdot 000 \cdot 111 \cdot 000 \cdot 111 \cdot 000$$

Ce second code a alors des propriétés très différentes

— Si lors de la transmission, un (seul) bit est erroné, on peut s'en rendre compte

$$000 \cdot 110 \cdot 111 \cdot 000 \cdot 111 \cdot 000 \cdot 000 \cdot 111 \cdot 111$$

et même savoir lequel.

— Si deux bits sont erronés, on s'en rend compte. On peut en effet tomber sur les deux situations suivantes :

$$000 \cdot 110 \cdot 111 \cdot 000 \cdot 111 \cdot 001 \cdot 000 \cdot 111 \cdot 111$$

$$000 \cdot 111 \cdot 111 \cdot 000 \cdot 111 \cdot 011 \cdot 000 \cdot 111 \cdot 111$$

Si on sait que deux bits ou moins sont erronés, on peut dans le premier cas corriger ces deux erreurs, mais pas dans le deuxième. Est-ce que seul le 0 de 011 est erroné, ou s'agit-il des deux autres bits ?

— Enfin, si deux bits sont perdus dans la transmission, on peut les retrouver, comme on s'en rend compte sur les deux situations suivantes

$$000 \cdot 111 \cdot 111 \cdot 000 \cdot 11x \cdot 00x \cdot 000 \cdot 111 \cdot 111$$

$$000 \cdot 111 \cdot 111 \cdot 000 \cdot 1xx \cdot 111 \cdot 000 \cdot 111 \cdot 111$$

3.2.2 Définitions et premières propriétés

On appelle *code* de paramètre (n, m) un dispositif qui transforme n bits en m bits. On notera $f(x)$ la transformation. On appelle *mot de code* tout mot de m bits obtenu lors de la transformation, c'est donc l'ensemble des mots de m bits qui sont de la forme $f(x)$ pour un certain x.

On dit qu'un mot est transmis avec k erreurs si le résultat de la transmission contient k bits différents de leur valeur initiale

Définition 3.1 (Détection)**.** Un code *détecte* k erreurs si lorsqu'un mot $f(x)$ est transmis avec k erreurs ou moins, on peut se rendre compte que le message est erroné

Définition 3.2 (Correction)**.** Un code *corrige* k erreurs si lorsqu'un mot $f(x)$ est transmis avec k erreurs ou moins, on peut se rendre compte que le message est erroné et retrouver $f(x)$.

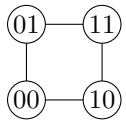
> **Exemple**
>
> Le code qui double tous les bits est un code de paramètre $(n, 2n)$ qui détecte une erreur mais n'en corrige pas.
> Le code qui triple les bits est un code de paramètre $(n, 3n)$ qui détecte deux erreurs et en corrige une.

3.2.3 Graphe d'un code

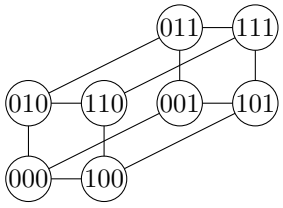
Une bonne façon de comprendre et d'analyser un code est de le représenter par un graphe.

On va regarder le graphe dont les sommets sont tous les mots possibles de n bits, et où on relie deux sommets s'ils n'ont qu'un bit de différent.

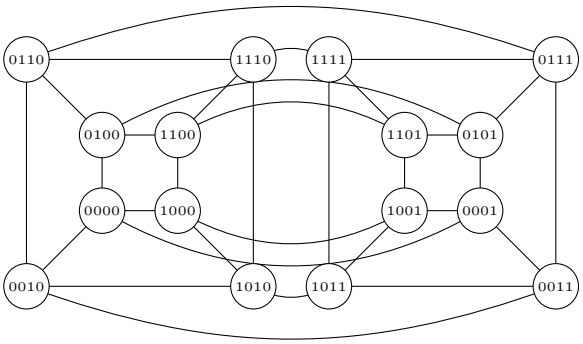
Voici le graphe pour $n = 2$:



Voici le graphe pour $n = 3$

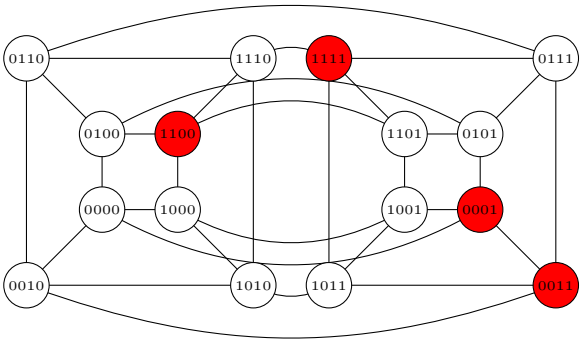


Et enfin pour $n = 4$:



Si on décide de représenter 2 bits parmi 4 (donc si on parle d'un code (2,4)), cela revient à dire qu'on choisit un code constitué de 4 bits pour représenter 00, 01, 10 et 11.

Cela revient donc à choisir 4 sommets sur le graphe pour $n = 4$:



Sur le graphe, faire une erreur revient à se déplacer d'une arête. Un bon code est donc un code qui "espace" le plus possible les mots choisis.

> **!** Important
>
> Un code détecte une erreur si, lorsqu'on se place sur un mot de code et qu'on se déplace d'une arête, on ne retombe pas sur un mot de code. Dit autrement, un code détecte une erreur s'il n'y a pas deux mots de code reliés par une arête.

> **i** **Exemple**
>
> Le code représenté sur le dessin précédent ne détecte pas une erreur. En effet, deux des mots qui ont été choisis (0001 et 0011) sont reliés par une arête.

Plus généralement :

> **!** Important
>
> Un code détecte k erreurs si, lorsqu'on se place sur un mot de code et qu'on se déplace de k arêtes ou moins, on ne retombe pas sur un mot de code.
> Dit autrement, un code détecte k erreurs si deux mots de code ont au moins $k + 1$ bits différents.

Tester si un code *corrige* une erreur est un peu plus subtil.

Un code corrige 1 erreurs s'il est capable de corriger tout mot de code transmis avec 1 erreurs ou

moins. Prenons un mot x transmis avec 1 erreur, ce qui donne le mot z. Pour que le code soit capable de corriger z, il faut qu'il n'ait qu'une seule possibilité pour le corriger, autrement dit qu'il n'existe pas de mot y qui a également 1 bits bits de différence avec z (sans quoi on ne sait pas s'il faut corriger z en x ou en y). On obtient donc :

> **!** Important
>
> Un code corrige k erreurs si et seulement si deux mots de code ont au moins $2k+1$ bits différents.

Attention, le raisonnement précédent ne permet pas de prouver qu'une des deux directions (si un code corrige k erreurs, alors deux mots de code ont au moins $2k+1$ bits différents), mais pas l'autre direction (si tous les mots de code ont au moins $2k+1$ bits différents, alors le code corrige k erreurs).

3.2.4 Exemples importants

3.2.4.1 Bit de parité

Donnons maintenant quelques exemples. L'exemple le plus simple consiste à ajouter un bit de parité. On transforme chaque mot x en un mot dans lequel on a ajouté un bit qui vaut 1 si et seulement si le nombre de 1 dans x est impair, et 0 sinon.

Ainsi, le mot $0 \cdot 0 \cdot 1 \cdot 1$ devient $0 \cdot 0 \cdot 1 \cdot 1 \cdot 0$ et le mot $0 \cdot 0 \cdot 1 \cdot 0$ devient $0 \cdot 0 \cdot 1 \cdot 0 \cdot 1$.

Dit autrement, on s'aperçoit que le bit ajouté est tel que le nombre de 1 du mot obtenu est pair, et que les mots de code sont donc tous les mots dont le nombre de 1 est pair.

On en déduit que ce code, de paramètre $(n, n+1)$ détecte une erreur. En effet, si on change un bit dans un mot de code, le nombre de bits égaux à 1 deviendra impair. Plus généralement, si on change 3 bits dans un mot de code, ou 5 bits, ou 7 bits, etc, on obtiendra un mot qui a au moins une erreur, mais pas si on change 2, 4, 6 bits.

3.2.4.2 ISBN

Un code détecteur d'erreur utilisé régulièrement est celui utilisé dans les ISBN. L'ISBN d'un document est constitué de dix chiffres. Le dernier chiffre est calculé de la manière suivante : Si on note x_i le $i$ème chiffre, on a $x_{10} = 1 * x_1 + 2 * x_2 + 3 * x_3 + 4 * x_4 + \cdots + 9 * x_9 \mod 11$. Par exemple, on trouve les ISBN : 2-070-36621-9, ou encore 2-226-10936-6 ou 2-754-00257-X (le X est présent lorsqu'on tombe sur $x_{10} = 10$)

On vérifie aisément que ce code détecte une erreur. Si l'erreur porte sur x_{10}, c'est évident. Sinon, supposons que x_i soit transmis avec une erreur, et qu'on transmette à la place y_i. Pour que l'erreur ne se repère par sur x_{10}, il faut que la contribution de x_i dans la définition de x_{10} soit la même que la contribution de y_i. Autrement dit, il faut $ix_i = iy_i \mod 11$, ce qui implique $x_i = y_i$.

Notons que la démonstration explique pourquoi il faut prendre 11 et non 10 pour calculer le modulo. Si on change par exemple le 8ème chiffre de 2 vers 7, on voit que x_{10} ne changerait pas (puisque $2*8 = 7*8 \mod 10$) Comme 11 est premier, cette situation n'arrive pas.

Notons aussi qu'on aurait pu prendre pour faire l'ISBN une définition bien plus simple en prenant $x_{10} = x_1 + \cdots + x_9$. Cependant l'ISBN comme il est défini a d'autres avantages. Supposons par exemple qu'on échange 2 chiffres consécutifs, les chiffres en position i et $i+1$. Si la valeur de x_{10} est toujours la même, cela signifie que $ix_i + (i+1)x_{i+1} = ix_{i+1} + (i+1)x_i$, c'est à dire $x_i = x_{i+1}$. Autrement dit, on peut aussi se rendre compte que deux chiffres ont été échangés.

3.2.5 Code rectangle

Donnons ici un exemple simple de code $(nm, nm+n+m)$ qui corrige une erreur.

L'idée est la suivante : On organise les nm bits dans un rectangle $n \times m$ (ici n=3, m=2)

$$\begin{array}{ccc} 1 & 1 & 1 \\ 0 & 0 & 1 \end{array}$$

puis on calcule la parité de chaque ligne et de chaque colonne (dit autrement, on ajoute des bits de façon à ce que le nombre de bits à 1 sur chaque ligne et chaque colonne soit pair)

$$\begin{array}{ccc|c} 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ \hline 1 & 1 & 0 & \end{array}$$

Grâce à ce dispositif, on peut détecter une erreur et la corriger. On a en effet deux cas lorsqu'une erreur s'est produite : Le premier cas est le suivant :

$$\begin{array}{ccc|c} 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ \hline 1 & 1 & 1 & \end{array}$$

Dans ce cas, on constate deux incohérences : la première ligne n'est pas de somme paire, pas plus que la deuxième colonne. On en déduit que l'erreur (puisqu'on la suppose unique) se situe sur l'intersection de la première ligne et de la deuxième colonne, il faut donc changer le 1 qui s'y trouve en 0.

Le deuxième cas est le suivant

$$\begin{array}{ccc|c} 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 \\ \hline 0 & 0 & 0 & \end{array}$$

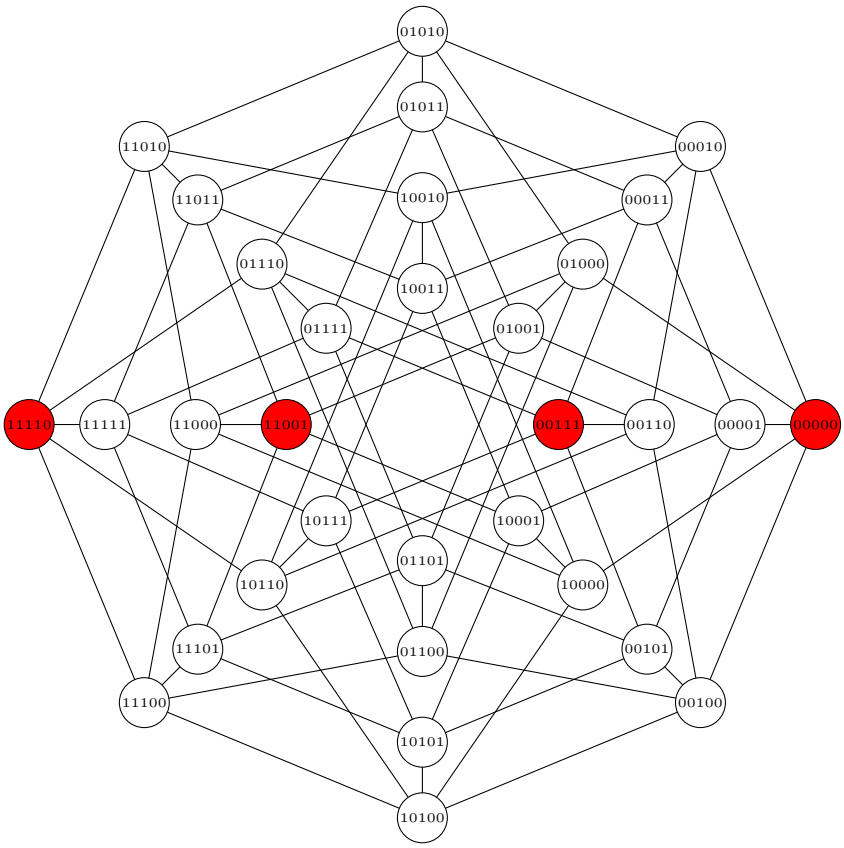
Dans ce cas, on constate une seule incohérence : la deuxième colonne n'est pas de somme paire. L'erreur provient donc uniquement du bit de parité ajouté, que l'on change donc de 0 à 1.

Un cas particulier important est obtenu lorsqu'on part d'un rectangle 1 par 2 . On obtient alors le code suivant :

$$\begin{array}{c|c} x & x \\ y & y \\ \hline x+y & \end{array}$$

Par construction, on transforme donc 2 bits en 5 bits : Il s'agit donc d'un code $(2, 5)$ et celui-ci corrige une erreur. Il est constitué des mots (quand on lit de gauche à droite et de haut en bas) 00000, 00111, 11001 et 11110.

Avec un peu de motivation, on peut représenter les 4 mots de code sur le graphe de tous les mots de taille 5 :



On peut vérifier sur le dessin que les 4 points sont bien à une distance de 3 les uns des autres.

3.2.6 CRC (Cyclic Redundancy Check)

Le CRC est une technique très standard pour obtenir un code correcteur d'erreur.

Avant d'expliquer comment elle fonctionne, on va s'attarder sur un exemple un peu plus concret qui va nous guider

Exemple

Les deux derniers chiffres du numéro de sécurité sociale, souvent appelés “clé” sont obtenus de façon à ce que le numéro entier, une fois qu'on a ajouté ces deux derniers chiffres, soit divisible par 97.
Par exemple, si on part du numéro de sécu (fictif) 1 80 06 57 150 282, on doit ajouter deux chiffres à la fin de ce numéro de façon à ce que le résultat soit divisible par 97. Les deux chiffres à ajouter sont 67,comme on pourra le vérifier avec une calculatrice ou un téléphone intelligent.
Comment trouver ces deux chiffres ? On procède ainsi : On commence par ajouter deux 0 au nombre en question, pour obtenir 180065715028200, et on divise le résultat par 97, ce qui fait, après calcul, 30.
Si on ajoute 97-30 au résultat, c'est à dire 67, le reste passe donc à 30+97-30 = 97, ce qui est la même chose que 0.
On obtient donc l'algorithme suivant pour trouver les deux chiffres à ajouter : Ajouter deux 0 au numéro de sécu, diviser le résultat par 97. Si x est le reste de la division, alors il faut remplacer les deux 0 par $97 - x$.
En exercice, il est conseillé d'essayer de vérifier que la clé correspondant au numéro de sécu 1 79 01 88 158 117 est 51.
Pourquoi ce mécanisme permet-il de corriger les erreurs ?
Supposons se tromper dans le numéro de sécu, et avoir écrit 1 80 06 51 150 282 67 plutôt que 1 80 06 57 150 282 67. On a donc commis une erreur d'exactement 600000000. Or, 600000000 n'est pas divisible par 97, donc on calculant le reste de la divison par 97, on verra qu'on a fait une erreur. Le raisonnement est le même si on se trompe sur un autre chiffre.

Les CRC fonctionnent exactement de la même façon, sauf qu'on raisonne non pas en terme de nombres, mais en terme de polynôme, et qu'on raisonne modulo 2. (Retenir donc que 1 + 1 = 0, et que -1=1 dans les calculs qui suivent).

Important

Pour calculer le CRC correspondant au polynôme G(x) pour un mot w, on procède de la façon suivante :

- On ajoute n chiffres 0 à la fin du mot w, où n est le degré du polynôme
- On convertit le mot obtenu en un polynôme (le chiffre des unités correspond au terme constant, le chiffre suivant au terme en X, le chiffre suivant au terme en X^2, etc), qu'on note P
- On divise P par $G(x)$ et on retient le reste R
- On reconvertit le reste R en un mot binaire, et on remplace les 0 qu'on avait ajouté par R.

Intuitivement le polynôme $G(x)$ joue le rôle de 97. Dans la version avec les entiers, il faut ajouter $97 - R$. Ici on ajoute R, car on est en base 2 et donc addition = soustraction.

Exemple

On va calculer le CRC pour la suite de bits 10101011 et le polynôme $G(X) = X^5 + X + 1$.
Le polynôme générateur G est de degré 5 donc on ajoute 5 bits, qu'on fixe initialement à 0, pour obtenir

1010101100000

On le traduit en polynôme, ce qui donne

$$X^{12} + X^{10} + X^8 + X^6 + X^5$$

On effectue ensuite la division du polynôme $X^{12} + X^{10} + X^8 + X^6 + X^5$ par $X^5 + X + 1$:

$$\begin{array}{lllllll|l}
X^{12} & +X^{10} & +X^8 & & +X^6 & +X^5 & & X^5 + X + 1 \\
\cline{8-8}
X^{12} & & +X^8 & +X^7 & & & & X^7 + X^5 + X^2 \\
\cline{1-5}
 & X^{10} & & +X^7 & +X^6 & +X^5 & & \\
 & X^{10} & & & +X^6 & +X^5 & & \\
\cline{2-6}
 & & & X^7 & & & & \\
 & & & X^7 & & & +X^3 & +X^2 \\
\cline{4-8}
 & & & & & & X^3 & +X^2
\end{array}$$

Le résultat est le polynôme $X^3 + X^2$, qui se convertit en la suite de bits 01100. Le CRC cherché est donc 01100.
Dans une transmission qui utiliserait le CRC, on enverrait donc la chaîne de bits initiale, suivi du CRC, c'est à dire 1010101101100.

Au lieu de raisonner en termes de polynôme, il existe une deuxième façon de faire le calcul. Les deux raisonnements donnent exactement les mêmes calculs, il faut donc préferer celui qu'on trouve le plus simple.

Important

Pour calculer le CRC correspondant au polynôme G(x) pour un mot w, on procède de la façon suivante :

- On ajoute n chiffres 0 à la fin du mot w, où n est le degré du polynôme
- On convertit $G(x)$ en mot (le chiffre des unités correspond au terme constant, le chiffre suivant au terme en X, le chiffre suivant au terme en X^2, etc), qu'on note g
- On essaie de soustraire g, en partant de la gauche du mot de départ, de façon à éliminer tous les 1. Attention ! On est en base 2, donc il faut faire des xor bit à bit et non pas une vraie soustraction.
- Lorsqu'on ne peut plus soustraire g, on a terminé, et on obtient le CRC.

Exemple

On reste sur le même exemple.On va calculer le CRC pour la suite de bits 10101011 et le polynôme $G(X) = X^5 + X + 1$. On commence comme précédemment par ajouter 5 symboles 0 à la suite de bits, pour obtenir 1010101100000. Cette fois-ci, c'est G qu'on convertit en une suite de bits, pour obtenir 100011. On essaie ensuite d'éliminer tous les 1 de notre suite de bits en utilisant g :

$$\begin{array}{r} 1\,0\,1\,0\,1\,0\,1\,1\,0\,0\,0\,0\,0 \\ 1\,0\,0\,0\,1\,1 \\ \hline 0\,0\,1\,0\,0\,1\,1\,1\,0\,0\,0\,0\,0 \\ \quad 1\,0\,0\,0\,1\,1 \\ \hline 0\,0\,0\,1\,0\,0\,0\,0\,0\,0\,0 \\ \quad\quad 1\,0\,0\,0\,1\,1 \\ \hline 0\,0\,0\,0\,1\,1\,0\,0 \end{array}$$

Il faut bien se souvenir que,à chaque étape, on fait des xor bit à bit : il n'y a pas de retenue.
Lorsqu'on a terminé le calcul, on note les 5 derniers bits, et on retrouve 01100. Le CRC cherché est donc 01100.
Dans une transmission qui utiliserait le CRC, on enverrait donc la chaîne de bits initiale, suivi du CRC, c'est à dire 1010101101100.

Le lecteur est encouragé à refaire le calcul précédent des deux manières différentes, et à se convaincre qu'on fait vraiment la même chose présentée d'une façon différente.

3.3 Partage du medium

Le mécanisme de découpage en trames permet d'utiliser la couche physique pour transmettre des messages, et l'ajout de codes correcteurs d'erreur permet de savoir si le message a été transmis correctement ou non.

La dernière problématique qu'il reste à gérer est la problématique d'un medium partagé, c'est à dire lorsque deux machines ou plus essaient de communiquer en même temps sur le même medium, cable ou air.

La difficulté qui se rajoute est que les deux machines peuvent essayer de parler en même temps. Dans ce cas, suivant la technologie de la couche physique, il est très probable que les messages envoyés par les deux machines interfèrent, et qu'ils soient donc bruités voire illisibles.

La problématique qu'il faut régler est donc une problématique de *gestion du temps de parole*.

3.3.1 Problématique

D'un point de vue théorique, le problème se pose ainsi : nous avons N machines qui essaient de communiquer par un unique medium, de sorte que, si deux trames sont transmises simultanément sur le medium, il y a *collision*.

Cette collision peut se produire dans plusieurs situations :

— Dans un réseau sans fil, si deux machines parlent en même temps
— Entre deux machines reliées par un cable, si les deux machines utilisent le même fil pour communiquer (ce n'est PAS le cas dans Ethernet, où on utilise un paire torsadée pour les communications de A vers B et une autre pour les communications de B vers A, les deux dans le même cable).
— Dans un réseau filaire constitué de plusieurs machines, et connectées par un hub, si le hub reçoit par exemple deux messages provenant de deux machines à peu près en même temps.

Les protocoles qui permettent de résoudre, au moins partiellement, ce problème dépendent de la possibilité ou non de détecter des collisions, ce qui va dépendre de l'implémentation de la couche physique.

— Dans un réseau cablé, on peut écouter et parler en même temps. Quand on envoie un message, on peut donc, en écoutant sur le cable, voir si quelqu'un est en train de parler en même temps.
— Dans un réseau wifi, on ne peut pas en général émettre et parler en même temps. Autrement dit, on peut, avant de parler, vérifier que personne n'est déjà en train de parler, mais on ne peut pas voir que quelqu'un est en train de parler si on est nous même en train de parler.

Dans le cas d'un réseau sans fil, un autre problème se pose : même s'il était possible d'émettre et écouter en même temps, on ne pourrait pas pour autant détecter les collisions ! En effet, considérons la situation suivante :

— Nous avons 3 machines qui communiquent par le wifi, A, B, et C.
— A et C essaient de parler à B
— L'émetteur wifi de A permettent d'atteindre B, mais pas C.
— L'émetteur wifi de C permettent d'atteindre B, mais pas A.

Dans ce cas, si A et C parlent en même temps, A et C ne peuvent pas le savoir puisque, même en écoutant ce qui se passe à leur niveau, ils ne voient pas que quelqu'un d'autre parle. Et B ne peut pas non plus leur signaler qu'il y a un problème, car il recevra les deux signaux mélangés, et ne comprendre pas qui est en train de parler.

3.3.2 Solution en couche physique

Une première solution au problème est de supposer qu'il n'y a pas de problème. C'est ce qui arrive si la couche physique résout le problème pour nous. Par exemple :

— Deux machines reliées par un cable *full-duplex* (par exemple dans le cas d'Ethernet, qui utilisent une paire pour la communication de A à B et une paire pour la communication de B à A) . Dans ce cas, aucune collision ne peut se produire.
— Plusieurs machines sur un réseau wifi, qui utilisent des *fréquences* différentes.

3.3.3 Solution Token Ring.

Le token ring est une solution au problème, utilisée en autre par IBM, qui suppose une topologie particulière du réseau : les machines sont organisées en un anneau (cercle).

L'idée est qu'il y a un jeton dans l'anneau, et que seule la machine qui a le jeton a le droit de parler.

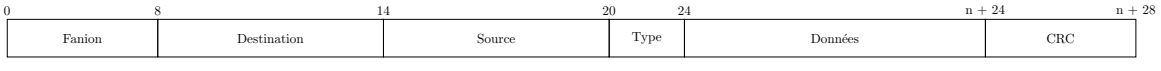
Si la machine veut parler, elle envoie un message qui va aller vers sa voisine, puis la suivante, jusqu'à faire tout le tour du cercle et le revenir. Quand le message lui revient, elle passe son jeton à sa voisine pour que celle-ci puisse parler.

Pour qu'un tel système soit performant, il faut bien entendu des mécanismes pour garder le jeton si on a beaucoup de données à envoyer et que les autres n'ont rien à dire

3.3.4 Aloha

Aloha est un protocole très simple.
TODO

3.4 Ethernet



Le fanion est constitué des bits 10101010 répétés 7 fois puis des bits 10101011. Le dernier octet du fanion est quelquefois appelé SFD, pour *start frame delimiter*.

Le but du fanion est de réveiller l'interface réseau qui reçoit le message. Il se peut qu'elle rate donc le début du fanion.

Les 4 derniers octets de la trame correspondent à une CRC calculé sur tous les autres champs (sans compter le fanion), pour le polynôme $G(x) = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$.

> **Note**
>
> Il y a deux différences entre la théorie des CRCs expliquée précédemment et ce qui se passe vraiment dans Ethernet :
> — Avant de calculer la CRC d'un message, on complémente (on remplace les 0 par des 1 et réciproquement) les 32 premiers bits du message (donc les 4 premiers octets de l'adresse destination)
> — On insère dans le champ CRC non pas la CRC, mais son complément.
>
> La raison de ces deux modifications est laissée en exercice au lecteur. (Considérer ce qui se passe quand on ajoute un 0).

A noter que la plupart des cartes réseaux s'occupent de vérifier que la CRC est correcte, et n'envoient pas le paquet au reste de la pile réseau si ce n'est pas le cas. En pratique, dans un analyseur réseau (type tcpdump, wireshark, etc), on obtient donc des trames ethernet sans le fanion et sans le CRC.

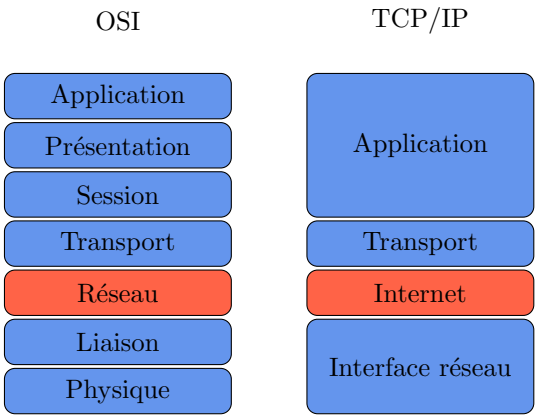
3.4.1 Ethernet commuté

TODO

Notes

A lire pour en savoir plus :
— Les chapitres correspondant des livres mentionnés supra ;
— La recommandation X.212 de l'UIT-T (qui expliquent le rôle de la couche liaison) ;
— La norme IEEE 802.3 qui décrit Ethernet. On trouvera page 118 et suivantes (chapitre *Media Access Control (MAC) frame and packet specifications*) le descriptif d'une trame Ethernet.
— La norme IEEE 802.1D qui décrit les commutateurs (appelés "bridge'' dans ce document). On trouvera dans le chapitre 7 quelques explications sur l'Ethernet commuté.

4 Couche réseau, IP



Introduction

La couche liaison, vu dans le cours précédent, permet d'établir des communications fiables entre deux interlocuteurs situés sur le même medium, ou sur le même réseau *local*.

Mais Internet n'est pas un seul réseau, mais une interconnexion de réseaux.

Le rôle de la couche réseau est donc de permettre à deux interlocuteurs situés à différents endroits d'une interconnexion de réseau de dialoguer.

Plus précisément, la couche réseau offre plusieurs fonctionnalités :

— Le transfert de données entre deux utilisateurs de la couche supérieure (couche transport) sans que ceux-ci aient à connaître les détails de l'interconnexion de réseau ;

— Une notion d'*adresse réseau* associée à une machine (plus exactement à la couche réseau d'une interface réseau), qui l'identifie de façon unique dans toute l'interconnexion. Cette adresse dépend de l'endroit exact où se trouve la machine.

Pour opérer le transfert de données, le service de réseau doit typiquement acheminer les données à travers plusieurs réseaux intermédiaires. On parle de *routage*.

Dans le cas de TCP/IP, les adresses utilisées sont des adresses IP, et le protocole utilisé pour l'envoi des données est le protocole IP.

Pour être exact, il existe plusieurs versions du protocole (et des adresses) IP : Actuellement, les versions majoritairement utilisées sont les versions IPv4 (RFC 791) et IPv6 (RFC 2460 et 8200).

Dans ce cours, nous ne parlerons que de la version IPv4.

4.1 Adresses Réseau dans IPv4

4.1.1 Un peu de bureaucratie

Les adresses réseau qu'on donne aux machines (plus précisément aux interfaces réseaux) ne peuvent pas être arbitraires : L'adresse permettant d'identifier l'interface de façon *unique* sur toute l'interconnexion de réseaux, il faut éviter que deux interfaces puissent avoir la même adresse. On ne peut pas donc laisser chacun choisir son adresse librement, et il est nécessaire d'avoir des *autorités* en charge des adresses [1].

Un ensemble d'adresses réseau gérée par une même autorité s'appelle un *domaine d'adressage*.

Il est important que l'adresse d'une machine permette de savoir à quelle autorité elle "appartient". En effet, le transfert d'un paquet réseau d'une machine à une autre va éventuellement traverser plusieurs autres réseaux, et ce n'est pas gratuit : il faut savoir qui va "payer" pour ce transport.

D'un point de vue pratique, cela signifie qu'une partie de l'adresse réseau doit permettre d'identifier facilement l'autorité en charge de l'adresse. Pour y arriver, on utilise un codage préfixe : le "début" d'une adresse réseau permet d'identifier l'autorité, le reste de l'adresse réseau permet d'identifier la machine au sein de toutes les machines administrées par l'autorité.

D'un point de vue théorique, l'adresse d'une interface réseau est donc a priori uniquement bureaucratique, et complètement indépendante des interconnexions réseaux. Mais séparer complètement l'adressage (chaque machine a une adresse différente) du routage (on doit être capable d'acheminer des paquets entre deux machines, quelque soit l'interconnexion de réseau qui les relie) est contreproductif, d'autant qu'en pratique il y a souvent unité de lieu : un même sous-réseau (comme un réseau local) appartient en général au même sous-domaine.

Dans la pratique, l'adresse réseau d'une machine doit également permettre de repérer un tout petit peu la position de la machine dans l'interconnexion de réseaux, c'est à dire l'ensemble des sous-réseaux auxquels la machine appartient (une machine appartient à un réseau local A, lui-même inclus dans un réseau un peu plus grand B, lui-même inclus dans un réseau plus grand C…).

Tout ceci implique que les différents sous-réseaux auxquels une machine appartient doivent pouvoir être lisible également facilement dans l'adresse réseau, et en particulier également que le concept d'adresse réseau ne va pas s'appliquer uniquement aux machines (interfaces réseaux) mais aussi aux réseaux eux-même : Les réseaux vont également avoir des adresses réseaux.

C'est ce que nous allons voir dans le cas d'IPv4.

4.1.2 Adresses IPv4 - Généralités

Les adresses réseau utilisées dans le modèle TCP/IP sont les adresses IPv4 [2].

Les adresses IPv4 sont constituées de 32 bits. Il est d'usage de regrouper ces 32 bits en 4 blocs de 8 bits, chaque bloc étant représenté sous forme d'un entier binaire, comme on peut le constater sur l'exemple suivant correspondant à l'adresse 00000010001000000100000000100101 :

00000010		00010000		01000000		00100101
2	.	16	.	64	.	37

Comme expliqué ci-dessus, chaque adresse IP peut se découper en deux parties : l'adresse du réseau dans laquelle se trouve l'interface réseau, et l'adresse de l'interface à l'intérieur du réseau.

Pour savoir comment découper, il faut savoir de quel réseau on parle : une même IP est dans plusieurs réseaux à la fois : le réseau local, un réseau un peu plus grand qui contient le réseau local, un réseau encore plus grand, et enfin l'Internet tout entier. Du point de vue de l'interface réseau, seul le réseau

1. Le modèle OSI (voir X.213) prévoit même une hiérarchie d'autorités
2. Les adresses IPv1,IPv2,IPv3 n'ont jamais vraiment été utilisées. Le premier concept d'adresse IP (nommé IPv0) date de la note IEN2 parue août 1977, les adresses IPv4 apparaissent en juin 1978 (IEN 41). Il faut noter qu'à l'aube d'Internet, TCP et IP formaient un seul protocole (nommé TCP) jusqu'au jour (IEN2) où un des concepteurs de l'Internet, Jon Postel, s'est rendu compte qu'il fallait séparer les deux (comme quoi le modèle OSI c'est pas inutile).

local compte, et c'est donc ce dernier qui est important lorsqu'on parle du réseau auquel appartient une adresse

Connaissant une adresse IP, un réseau est identifié par un *préfixe*, représentant le nombre de bits de l'adresse qui constituent le réseau (les autres bits représentant l'adresse de l'interface dans le réseau).

Il y a plusieurs façons de donner ce préfixe :

— D'abord en donnant explicitement le nombre n de bits (entre 0 et 32) qui désigne le réseau, les autres bits étant destinés à identifier la machine dans le réseau

— Soit en donnant un *masque* qui est la donnée d'un autre entier 32 bits composé de n bits à 1 suivi de $32 - n$ bits à 0

On verra l'intérêt de la deuxième représentation un peu plus loin.

! Adresse Réseau

En IPv4, l'adresse d'une machine (plus exactement d'une interface réseau) est la donnée :

— D'une adresse IP, c'est à dire d'un entier 32 bits souvent donné sous la forme de 4 entiers 8 bits $xxx.yyy.zzz.www$.

— D'un préfixe, permettant d'identifier combien de bits de l'adresse IP désignent le réseau et combien désignent la machine à l'intérieur du réseau. Ce préfixe n peut être donné

  — Par le nombre de bits n de l'adresse qui désigne le réseau. Dans ce cas on écrit l'adresse de la machine sous la forme $xxx.yyy.zzz.www/n$.

  — Par un entier 32 bits formés de n bits à 1 suivi de $32 - n$ bits à 0. On appelle ce nombre le *masque* du réseau.

i **Exemple**

Prenons une machine d'adresse 62.23.12.15/21.
L'adresse IP de la machine est donc 62.23.12.15. Le réseau de la machine correspond aux 21 premiers bits de l'adresse.
L'adresse 62.23.12.15 s'écrit en binaire 00111110.00010111.00001100.00001111. L'adresse du réseau est donc 001111100001011100001 (les 21 premiers bits) et l'adresse de la machine dans le réseau est donc 10000001111 (les 11 derniers bits).
Les adresses possibles dans ce réseau sont donc toutes les adresses qui partagent ces 21 premiers bits. La première d'entre elles est donc 00111110.00010111.00001000.00000000, soit encore 62.23.8.0 et la dernière est 00111110.00010111.00001111.11111111, soit encore 62.23.15.255.
Si on voulait donner le préfixe par un masque, il faut donner le nombre dont les 21 premiers bits sont à 1, et les 11 derniers à 0. C'est le nombre 11111111.11111111.11111000.00000000, qu'on peut écrire si l'on préfère 255.255.240.0.

Intérêt du masque

Le masque est une façon a priori un peu barbare de désigner le préfixe d'un réseau. Son intérêt majeur est technique : Soit A une adresse IP dans un réseau de masque M. Alors B est dans le même réseau que A si et seulement si faire le ET binaire de A et du masque M donne le même résultat que faire le ET binaire de B et du masque M.

(A noter que faire le ET binaire de A et du masque correspond à calculer la première adresse possible dans le réseau)

Un peu d'histoire

Historiquement, on pouvait "lire" le préfixe directement dans l'adresse IP, et les réseaux étaient divisés en classes :

- Si l'adresse IP est xxx.yyy.zzz.www avec xxx 127 alors c'est un réseau de préfixe 8 (donc constitué de toutes les machines dont l'adresse IP est xxx. ???. ???. ???), appelé réseau de classe A
- Si l'adresse IP est xxx.yyy.zzz.www avec 128 xxx 191 alors c'est un réseau de préfixe 16 (donc constitué de toutes les machines dont l'adresse IP est xxx.yyy. ???. ???), appelé réseau de classe B
- Si l'adresse IP est xxx.yyy.zzz.www avec 192 xxx 223 alors c'est un réseau de préfixe 24 (donc constitué de toutes les machines dont l'adresse IP est xxx.yyy.zzz. ???), appelé réseau de classe C
- Les autres adresses IPs avaient des significations particulières

Avec la pénurie des adresses IPs, une division aussi simple posaient beaucoup de problèmes : Si on a 100 000 machines, il faut un réseau de classe A (un réseau de classe B est trop petit), dont on n'utilisera qu'une portion infirme (moins de 1%).

Le fait de séparer les préfixes de l'adresse IP, et de permettre des préfixes de taille variable (et pas uniquement 8,16,24) est connu sous le nom de CIDR (Classless Inter-Domain Routing), et expliqué dans les RFCs 1518 et 1519.

De cette histoire est resté la nomenclature "classe A", "classe B", "classe C" : On appellera ainsi abusivement réseau "classe C" tout réseau dont le préfixe est /24 (même si il n'est pas de la forme indiquée précédemment).

4.1.3 Adresses IPv4 particulières dans un réseau

Les nombreuses normes qui régissent Internet ont conduit à particulariser certaines adresses dans un réseau :

- La toute première adresse possible désigne l'adresse du *réseau*. Cette adresse ne peut pas être utilisée pour une machine ou une interface réseau. On ne verra jamais l'adresse d'un réseau circuler sur Internet (historiquement ces adresses étaient utilisées pour une forme particulière de broadcast appelé directed broadcast)
- La toute dernière adresse possible désigne l'adresse de *broadcast* du réseau. Envoyer un message sur cette adresse revient à envoyer vers toutes les machines du réseau.

! Adresses IPv4 particulières

La première adresse d'un réseau (obtenue en mettant tous les bits de la partie machine à 0) est appelée l'adresse du réseau.
La dernière adresse d'un réseau (obtenue en mettant tous les bits de la partie machine à 1) est appelée l'adresse de *broadcast* du réseau.
Ces deux adresses ne peuvent pas être utilisées pour des interfaces de réseau. Un réseau de préfixe $/n$ contient donc 2^{32-n} adresses différentes, dont seulement $2^{32-n} - 2$ peuvent être utilisées pour des interfaces réseaux.

i **Exemple**

Reprenons l'exemple du réseau contenant l'interface dont l'IP est 62.23.12.15/21.
Le réseau a donc l'adresse IP 62.23.8.0 et l'adresse de broadcast est l'adresse 62.23.15.255.
Il y a donc 2046 adresses disponibles pour une machine, allant de 62.23.8.1 à 62.23.15.254.

A noter qu'un réseau /32 n'existe techniquement pas. De la même facon un réseau /31 représente un réseau avec 0 machines. Si on veut parler d'un réseau avec 2 machines (typiquement un réseau point à point), il faut donc théoriquement utiliser un préfixe /30 (depuis la RFC 3021, une exception a été faite

pour pouvoir utiliser un préfixe /31, sans adresse de réseau ni adresse de broadcast, pour désigner une connexion point à point).

4.1.4 Réseaux IP particuliers

Certaines réseaux et adresses réseaux ont été définis pour des cas particuliers.
Commençons par les adresses réseaux :
— 0.0.0.0 désigne l'interface réseau dans le réseau dans lequel on se trouve ;
— 255.255.255.255 désigne l'adresse de broadcast du réseau dans lequel on se trouve.
On peut globalement partager les réseaux en deux :
— Les réseaux publics : Chaque adresse dans un réseau public peut être utilisée une et une seule fois dans l'Internet : elle désigne l'interface réseau d'une machine spécifique au niveau de tout l'Internet
— Les réseaux privés : Un réseau privé n'existe pas dans Internet, mais juste localement : un routeur par défaut ne connaît pas ces réseaux. Plusieurs machines peuvent avoir la même adresse privée sans conflit, tant qu'elles sont sur des réseaux physiquement différents.
Les réseaux privés peuvent par exemple être utilisés si on veut concevoir un réseau non relié à Internet.
Il existe plusieurs réseaux privés (RFC 8190), dont voici les principaux
— 0.0.0.0/8 est un réseau particulier, qui représente le réseau local. On peut uniquement l'utiliser comme une adresse source
— 127.0.0.0/8 est un réseau particulier, appelé boucle locale, qui est un réseau fictif qui ne contient que la machine (qui sera très souvent d'adresse 127.0.0.1)
— 10.0.0.0/8 est un réseau privé
— 172.16.0.0/12 est un autre réseau privé
— 192.168.0.0/16 est un autre réseau privé
— 100.64.0.0/10 est un réseau privé, appelé espace partagé, qui est conçu pour connecter ensemble d'autres réseaux privés.

4.2 Routage - machine

Expliquons maintenant comment le routage fonctionne du point de vue d'une machine, puis d'un routeur.

Il existe deux façons principales de router plusieurs messages entre deux points A et B :
— On choisit le chemin que va prendre chaque message séparément
— On se met d'accord sur le chemin que vont prendre tous les messages de A vers B

Dans le deuxième cas, on parle d'un service réseau *connecté* : A chaque fois qu'un hôte A veut envoyer un message à un hôte B, on crée un *circuit virtuel* (la liste des routeurs à traverser) par lequel vont passer tous les paquets de A vers B. C'est le service qui est mis en place en particulier en téléphonie.

Dans ce cours, on se place principalement dans le cas de TCP/IP, pour lequel le service réseau n'est *pas* connecté : chaque message peut donc en théorie prendre un chemin différent.

Comme expliqué précédemment, le routage concerne les paquets qui ne sont pas à destination du réseau. Il faut donc commencer par expliquer ce qu'on fait des paquets sur le réseau local

4.2.1 Destination : réseau local - ARP

Sur le réseau local, on sait déjà comment envoyer un paquet vers une machine, si tant est qu'on dispose de son adresse *liaison* (dans le cas d'Ethernet, il s'agit de l'adresse MAC).

Il reste à expliquer comment obtenir l'adresse liaison à partir de l'adresse réseau. Le protocole à utiliser dépend évidemment de l'implémentation des adresses liaisons et réseau. Contentons-nous du cas le plus important : Ethernet et IP. Il faut donc convertir une adresse IP **locale** en une adresse MAC.

Le protocole remplissant cette fonctionnalité s'appelle ARP (Address Resolution Protocol) (RFC826).

Le principe est très simple : quand une machine A veut connaître l'adresse MAC associée à une adresse IP :

— Elle envoie un message ARP sur le réseau local en broadcast (pour que toutes les interfaces réseau du réseau local le reçoive) en précisant :
  — Son adresse MAC (pour qu'on sache où envoyer la réponse)
  — Son adresse IP (Elle fournit l'information "au cas où")
  — L'adresse IP de la machine qu'elle sert à joindre
— La machine concernée répond alors directement à la machine A en lui fournissant son adresse IP [3].

Le protocole est donné à la figure Figure 4.1. Le champ type précise quel est le type d'adresses qu'on veut convertir (dans le cas qui nous intéresse, on veut convertir de IPv4 à MAC), et la longueur de ces adresses (6 pour MAC et 4 pour IPv4). Le champ Opération précise si on est en train de faire une requête ou une réponse.

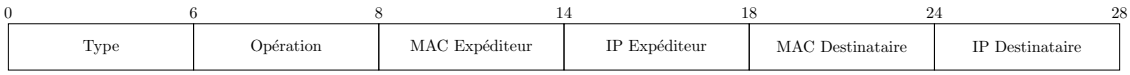


FIGURE 4.1 – Structure d'un message ARP

Donnons l'exemple de la machine A d'adresse IP 134.32.12.24/24 et d'adresse MAC `aa:bb:01:02:cc:de` qui veut parler à la machine B d'adresse IP 134.32.12.49/24 (qui est donc sur son réseau local). A ne connaît pas l'adresse MAC de B et va donc utiliser ARP pour l'obtenir :
— A envoie donc un message ARP en broadcast (adresse MAC destination `ff:ff:ff:ff:ff:ff`) en demandant qui a l'adresse IP 134.32.12.49.
— B répond directement à A (adresse MAC destination `aa:bb:01:02:cc:de`) en donnant son adresse MAC

Les trames correspondantes sont données à la Figure 4.2.

A noter que chaque machine garde les correspondances obtenues dans une table, sobrement appelée table ARP. Sur une machine linux, la correspondance est gardée par défaut pendant une durée de 60 secondes

La requête étant envoyée en broadcast, toutes les interfaces du réseau la reçoivent et peuvent donc en théorie stocker la correspondance IP/MAC de l'expéditeur.

En pratique ce n'est pas le cas : il est inutile d'encombrer sa table ARP avec des correspondances IP/MAC de machines avec lesquelles on ne communiquera jamais. Seule donc la machine réellement destinataire de la trame ARP (B dans l'exemple) stockera la correspondance IP/MAC de l'expéditeur (A dans l'exemple). La seule exception intervient lorsque la correspondance que l'on observe sur le réseau correspond à une adresse IP que l'on connaît.

A noter que la réponse ARP est *généralement* envoyée uniquement à la machine demandeuse, les autres interfaces ne la voient donc pas et ne peuvent stocker cette deuxième correspondance.

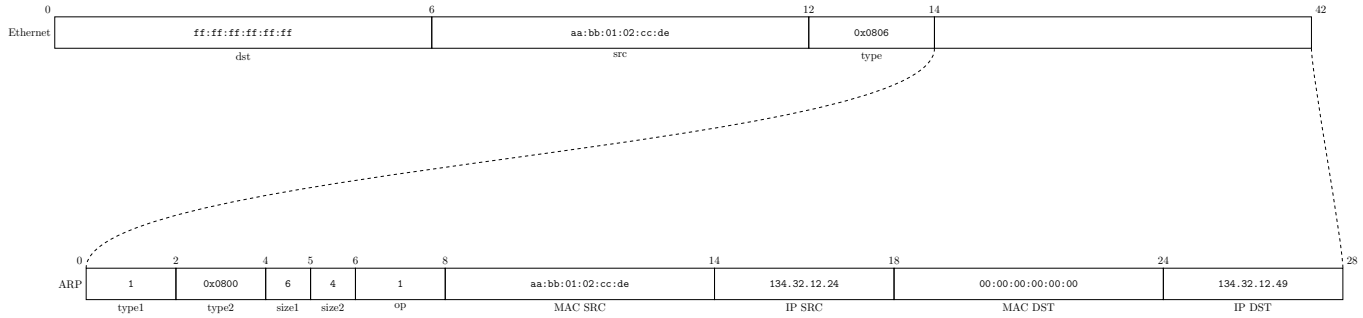
4.2.2 Vers l'infini et au delà - Tables de routage

Regardons maintenant comment fonctionne le routage lorsqu'on souhaite envoyer un message à une machine qui n'est pas sur le réseau local.

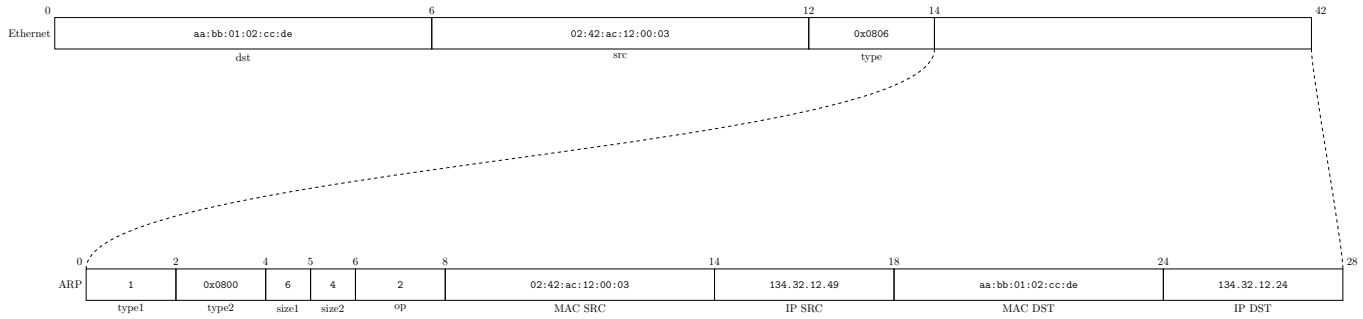
Si la machine n'est pas sur le réseau local, la seule question à se poser est de savoir *à qui* envoyer le paquet, puisque la machine ne peut le transmetter elle-même.

Pour cela, la machine dispose d'*une table de routage*. Une table de routage spécifie, pour chaque adresse réseau possible, à quel routeur, s'il existe, envoyer le message pour acheminer le paquet. On l'appelle également en anglais *forwarding table* puisqu'elle explique à qui on va transférer le message.

3. Techniquement, rien n'interdit dans la RFC que la machine réponde en broadcast plutôt que directement à A. Lire en particulier la RFC5227 à cet égard.



(a) Première trame



(b) Deuxième trame

FIGURE 4.2 : Un échange ARP

Dès qu'on parle d'un vrai réseau (typiquement Internet), il n'est pas possible (ou désirable) de lister toutes les adresses, et comment les joindre. En pratique, la table de routage contiendra donc des *réseaux* et non pas des adresses réseaux.

Une table de routage est donc une liste de *réseaux* et de routeurs à contacter pour atteindre ce réseau.

Table 4.2 – Une table de routage

Réseau	Routeur
192.168.0.0/24	123.42.1.14
14.17.0.0/24	123.42.1.15
10.0.0.0/8	123.42.1.14
10.0.10.0/24	123.42.1.15
18.21.16.64/32	123.42.1.15

Un exemple de table de routage fictif, pour le protocole IPv4, est donné Table 4.2.

Si une machine possède une telle table de routage, on peut en déduire qu'on peut transférer le message au routeur 123.42.1.14 si on souhaite atteindre une adresse du réseau 192.168.0.0/24, c'est à dire toute adresse entre 192.168.0.0 et 192.168.0.255. En particulier la dernière ligne de la table n'est pas une erreur (même si un réseau /32 ne contient a priori aucune machine), mais donne le routeur à contacter si on veut accéder à l'adresse exacte 18.21.16.64

A noter que la table de routage donne deux façons d'accéder à l'adresse 10.0.10.2 :

— En utilisant la troisième entrée de la table de routage (10.0.0.0/8) et en contactant donc le routeur 123.42.1.14.

— En utilisant la quatrième entrée de la table de routage (10.0.10.0/24) et en contactant donc le routeur 123.42.1.15.

Dans ce cas, il est précisé (RFC1519) qu'on doit utiliser la route *la plus spécifique*, c'est à dire celle qui correspond au plus petit nombre de machines (et donc au préfixe le plus grand).

Nous avons déjà vu ci-dessus qu'on pouvait trouver une entrée dans une table de routage correspondant à un réseau /32, c'est à dire constitué d'une seule machine. L'extrême inverse consiste à trouver une entrée 0.0.0.0/0. Une telle entrée représente toutes les adresses IP possibles (puisque le préfixe est de taille 0) : c'est une manière d'indiquer une route *par défaut* (qui, d'après l'explication précédente, ne sera prise que si une entrée plus spécifique n'existe pas).

A noter qu'une fois qu'on sait à quel routeur envoyer le paquet, il faut pouvoir contacter ce routeur, c'est à dire que ce routeur soit sur le réseau local, et connaître son adresse MAC. ARP strikes again !

On peut résumer :

! Algorithme de routage

Pour envoyer un message à l'adresse *xxx.yyy.zzz.www* :

— S'il s'agit d'une adresse d'un réseau local, on l'envoie directement sur le réseau local.
  — Si on connait l'adresse MAC de la machine, on l'envoie directement sinon on effectue une requête ARP pour la connaître.

— Sinon, on consulte la table de routage pour trouver le routeur à contacter.
  — Si plusieurs entrées de la table sont possibles, on choisit la plus spécifique ; L'entrée 0.0.0.0/0 est une entrée par défaut, toujours possible ;
  — Si aucune entrée ne correspond à notre adresse, on abandonne (No route to host) ;
  — On envoie ensuite le message au routeur correspondant à l'entrée de la table de routage.
    — Si l'adresse du routeur n'est pas dans le réseau local, on abandonne [a] ;
    — Si on connait l'adresse MAC du routeur, on l'envoie directement sinon on effectue une requête ARP pour la connaître.

a. Situation impossible en pratique.

Le schéma précédent est en fait un peu réducteur : pour savoir s'il s'agit d'une adresse d'un réseau local, il faut en fait aussi consulter la table de routage ! Par exemple, regardons la machine 10.4.0.1/16. Si, dans sa table de routage se trouve une ligne correspondant à 10.4.5.0/24, cela signifie que toutes ces adresses ne sont en fait pas sur son réseau local, mais qu'il s'agit d'un sous-réseau accessible à partir d'un routeur. En pratique, il faudrait donc considérer le réseau local comme une ligne particulière de la table de routage. Ce n'est pas ce qu'on fera dans ce cours par simplicité.

4.2.3 Le protocole IPv4

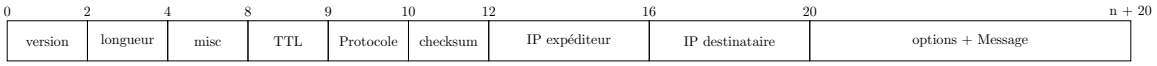


FIGURE 4.3 – Structure simplifiée d'un paquet IPv4

Nous avons maintenant assez d'informations pour comprendre la plupart des champs du protocole IPv4, présentés à la Figure 4.3.

Commençons par le protocole,

— Les deux premiers octets contiennent la version du protocole IP (4 pour IPv4), et d'autres paramètres importants (comme la longueur de l'entête IP) ;
— Les deux octets suivants contiennent la longueur totale du paquet IP ;
— Le TTL sera présenté à la section suivante ;
— Le protocole désigne le protocole encapsulé dans le paquet IP. Il s'agit en général d'un protocole transport (TCP, UDP, …) mais il peut aussi s'agit d'un autre protocole réseau, comme ICMP ;
— On trouve ensuite l'adresse IP de l'expéditeur et de la destination.

Les seules adresses IP mentionnées dans le protocole sont donc celles de l'expéditeur et de la destination. En particulier, nul part n'est précisé de quel routeur le paquet est parti, et à quel routeur il arrive : Un paquet IP, même s'il transite par de multiples routeurs, reste le même sur tout le chemin qu'il emprunte, à un détail important près, le TTL, que nous allons voir ultérieurement. Notons cependant que la trame (en général Ethernet) qui véhicule ce message changera, elle, durant le chemin.

4.3 Routage - routeur

Reste à expliquer comment fonctionne un routeur. On a déjà presque tout dit. *En fait, un routeur utilise des tables de routage, tout comme une machine.* La seule différence est dans ce que fait un routeur d'un paquet qui lui arrive et dont il n'est pas le destinataire : une machine le jette, un routeur le transférera au prochain routeur.

Dans le cas d'IP, traverser un routeur change très légèrement le paquet IP [4] : Le paramètre TTL (pour time to live) décremente en effet de 1 à chaque routeur et le paquet est détruit si le TTL atteint 0 (il commence usuellement à 64 ou à 128).

4.4 Commandes

Finissons cette partie par donner les différentes commandes qui permettent d'examiner (voire de modifier) l'état du réseau. Pour chaque utilisation, on va donner deux commandes : la commande `net-tools`, et la commande `iproute2`. `net-tools` était encore il y a peu le standard pour configurer/gérer un réseau, mais il est maintenant obsolète. `iproute2` est son remplaçant moderne.

4.4.1 Adressage

D'un point de vue pratique, on peut voir l'adresse IP d'une interface réseau avec les commandes `ifconfig/ip addr show` (Linux/BSD/OS X) ou `ipconfig` (Windows).

Voici un exemple de la sortie de la commande `ifconfig` sous Linux :

4. Il y a en pratique une autre modification, liée à la fragmentation. Par simplification, ce cours ne parle pas de fragmentation.

```
eth1: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        inet 172.17.0.1  netmask 255.255.0.0  broadcast 172.17.255.255
        ether 02:42:db:61:66:71  txqueuelen 0  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 152.81.9.143  netmask 255.255.240.0  broadcast 152.81.15.255
        inet6 fe80::4a4d:7eff:fee5:14ea  prefixlen 64  scopeid 0x20<link>
        ether 48:4d:7e:e5:14:ea  txqueuelen 1000  (Ethernet)
        RX packets 3488025  bytes 4068318682 (3.7 GiB)
        RX errors 0  dropped 151  overruns 0  frame 0
        TX packets 233765  bytes 33680748 (32.1 MiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
        device interrupt 16  memory 0xf7100000-f7120000

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Boucle locale)
        RX packets 251  bytes 24124 (23.5 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 251  bytes 24124 (23.5 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

On déduit de la sortie que la machine possède trois interfaces réseaux : `eth0`, `eth1` et `lo`.

Prenons l'exemple de l'interface `eth0`. La deuxième ligne précise que cette interface réseau a l'IP `152.81.9.143` et le masque `255.255.240.0`. Il s'agit donc du réseau `152.81.9.143/20`, qui contient toutes les adresses entre `152.81.0.0` et `152.81.15.255` (ce qu'on vérifie avec l'adresse de broadcast).

Si on veut donner une adresse précise à une interface réseau on peut utiliser une des deux syntaxes suivantes :

```
ifconfig INTF IP netmask MASQUE
ip addr add IP/PREFIX dev INTF
```

4.4.2 ARP

La table ARP peut être consultée sous Linux avec une des deux commandes suivantes

```
arp -n
ip neigh
```

Voici un exemple de table ARP :

```
Adresse         TypeMap AdresseMat          Indicateurs Iface
192.168.1.1     ether   a4:2b:b0:f6:a0:d6   C           eth0
192.168.1.4     ether   28:39:26:23:8b:7d   C           eth0
192.168.1.6     ether   48:b0:2d:2d:c7:9a   C           eth0
192.168.1.9     ether   00:11:32:93:06:79   C           eth0
192.168.1.15    ether   b8:27:eb:e4:36:82   CM          eth0
```

```
192.168.1.16            (incomplete)                        eth0
```

L'indicateur peut valoir `C`, s'il s'agit d'une entrée apprise (typiquement par une requête ARP) ou `CM` pour une entrée codée en dur. La ligne `incomplete` signifie que la tentative d'obtenir l'adresse MAC en utilisant ARP a échoué.

4.5 Routes

On peut accéder aux routes accessibles depuis une machine grâce aux commandes :

```
route -n
ip route
```

Ces commandes listent également les réseaux locaux auxquels la machine est connectée, c'est à dire pour lesquels aucun routage n'est stricto sensu nécessaire.

Pour ajouter une route à la table de routage, on peut utiliser l'une des commandes suivantes :

```
route add -net RESEAU netmask MASK gw PASSERELLE
ip route add RESEAU/PREFIX via PASSERELLE
```

Il suffit de remplacer le mot `add` par `del` pour enlever la route de la table de routage.

Enfin la commande suivante permet de savoir quelle route serait empruntée pour atteindre la destination

```
ip route get ADDRESSE
```

Notes

Les icônes utilisées dans cette partie du document sont les icônes proposées par Cisco et libres d'utilisation sans modification.

A lire pour en savoir plus :

— Les chapitres correspondant des livres mentionnés supra ;
— La recommandation X.213 de l'UIT-T (qui expliquent le rôle de la couche réseau) ;
— les RFCs suivantes :
  — La note IEN 2, qui explique pourquoi il faut séparer les couches réseau et transport, et qui donne la version 0 du protocole IP.
  — RFC791 qui explique le protocole IPv4
  — RFC826 qui explique le protocole ARP
  — RFC4632 qui explique le CIDR (la notation $xxx.yyy.zzz.www/nn$). La précédente version RFC1519 explique un peu plus précisément la notion de préfixe et de route la plus spécifique.
  — RFC8190 qui donne la liste des réseaux privés

Ressources

Voici une capture wireshark pour ARP