Chapitre 4

Le protocole IP

Dans ce chapitre, nous présentons IP, le protocole qui est à la base de tout le réseau Internet. Alors que le réseau Éthernet, décrit dans le chapitre précédent, peut parfaitement être remplacé par n'importe quelle autre technologie équivalente, c'est par IP que passent tous les échanges de données d'Internet. Il ne peut pas être facilement remplacé.

L'information présentée ici est cruciale pour acquérir la maîtrise des mécanismes sous-jacents du fonctionnement du réseau Internet ; c'est le plus gros chapitre avec celui sur le protocole TCP.

Le chapitre présente la version couramment utilisée d'IP, qu'on appelle IP version 4, souvent abrégée en IPv4. Il y a une autre version du protocole IP appelée IPv6 (IP version 6) qui remplacera peut-être IPv4 un jour et dont nous traiterons dans un autre chapitre.

4.1 Le but du protocole IP

Le protocole IP permet de connecter entre eux des réseaux. On a alors un *réseau de réseaux*. Le résultat est Internet. J'explique ici pourquoi on ne peut pas se contenter d'utiliser les adresses Éthernet pour y identifier les ordinateurs quoique chaque carte Éthernet dispose d'une adresse propre. Cela pose deux problèmes : le *plan d'adressage* et les technologies différentes.

Le réseau Internet utilise aussi des technologies différentes d'Éthernet, qui ont des formats d'adresses divers, pas nécessairement compatibles avec les adresses Éthernet. Il faudrait donc, pour utiliser les MAC address au niveau d'Internet, avoir d'une part une adresse et d'autre part un type qui précise le format dans lequel elle doit être interprétée. Cela impliquerait de transporter un agrégat de données (c'est toujours plus compliqué qu'un type de donnée simple) ; en plus il serait nécessaire d'avoir du code pour interpréter tous les types d'adresse qu'on peut rencontrer dans tous les endroits où l'on a à manipuler une adresse Internet

et ils sont nombreux. Pour introduire une nouvelle technologie de transmission, il faudrait modifier ce code sur toutes les machines pour permettre de la prendre en compte.

Les adresses Éthernet sont attribuées à une carte lors de sa fabrication ; cela signifie qu'il ne suffit pas d'examiner l'adresse pour déterminer *où* dans le réseau se trouve la machine qui la porte. Il faudrait donc, pour l'utiliser, que chaque ordinateur dispose dans l'idéal d'un plan complet du réseau Internet, avec l'adresse et la liste des connexions de chaque ordinateur. Au minimum d'une indication de la *direction* dans laquelle se trouve la machine qui répond à une adresse donnée. Ce plan (cette *table de routage*) serait beaucoup trop gros si il devait énumérer tous les ordinateurs d'Internet.

Pour résoudre ces problèmes, IP affecte à chaque interface réseau une adresse propre structurée, que nous appellerons *adresse IPv4*. Nous en détaillons la structure dans la section suivante.

4.2 Les adresses IPv4

Chaque interface réseau sur Internet dispose d'une adresse IPv4 qui est un nombre sur 32 bits (elle utilise donc quatre octets). Cette adresse est divisée en un *numéro de réseau* et un *numéro de machine sur le réseau*.

4.2.1 Écriture des adresses IPv4

Habituellement on écrit cette adresse sous la forme de quatre nombres qui contiennent la valeur de chaque octet, séparés par des points. (On peut considérer cela comme de la numération en base 256.) Ainsi des adresses IPv4 pourront être 1.2.3.4 ou 197.196.195.194. On appelle cette façon d'écrire des adresses IPv4 la *notation décimale pointée* (en anglais *dotted-decimal notation*). (On verra plus loin que des nomsde machines plus familiers comme `www.univ-paris8.fr` sont traduits en adresses IPv4.)

Sauf exceptions, l'adresse IPv4 d'une interface est unique sur le réseau Internet : il n'y a qu'une seule interface qui utilise cette adresse.

4.2.2 Numéro de réseau et numéro de machine

Les 32 bits de l'adresse IPv4 sont divisés en un numéro de réseau et un numéro de machine sur le réseau ; le numéro de réseau occupe les bits de poids forts et le numéro de machine ceux de poids faibles. Il existe deux manières équivalentes de noter la répartition entre réseau et machine dans l'adresse.

Masque de réseau

La première manière utilise un *masque* qui contient des bits à 1 dans les positions où l'adresse contient des bits du numéro de réseau.

Par exemple l'adresse 192.33.156.203 associée au masque 255.255.255.0 indique que le numéro de réseau occupe les 24 bits de poids forts (c'est le réseau 192.33.156.0) et le numéro de machine les 8 bits de poids faibles (c'est la machine 203 sur ce réseau).

Un autre exemple est l'adresse 193.54.153.250 avec le masque 255.255.255.128 : ici le numéro de réseau utilise les 25 bits de poids forts (c'est le réseau 193.54.153.128) et le numéro de machine les 7 bits de poids faibles (c'est la machine $250 - 128 = 122$ sur ce réseau).

Un dernier exemple avec l'adresse 193.54.153.122 avec le masque 255.255.255.128 : ici aussi le numéro de réseau utilise les 25 bits de poids forts (c'est le réseau 193.54.153.0) et le numéro de machine les 7 bits de poids faibles (c'est la machine 122 sur ce réseau).

Notation CIDR

La seconde manière de noter la répartition entre numéro de réseau et numéro de machine fait suivre l'adresse IP d'une oblique et du nombre de bits (de poids fort) qu'utilise le numéro de réseau. Les trois exemples du paragraphe précédent s'écriront respectivement `192.33.156.203/24`, `193.54.153.250/25` et `193.54.153.122/25`.

Adresse machine, réseau, broadcast

Les adresses dont le numéro de machine ne contiennent que des bits à 0 sont utilisées pour noter le réseau ; celles dont ce numéro ne contient que des 1 servent pour le *broadcast*, pour s'adresser à toutes les machines du réseau.

Avec des exemples :

— La machine `192.33.156.203/24` appartient au réseau numéro `192.33.156.0` et peut s'adresser à toutes les machines de son réseau local avec l'adresse `192.33.156.255`.

— La machine `193.54.153.250/25` appartient au réseau numéro `193.54.153.128` et peut s'adresser à toutes les machines de son réseau local avec l'adresse `193.54.153.255`.

— La machine `193.54.153.122/25` appartient au réseau numéro `193.54.153.0` et peut s'adresser à toutes les machines de son réseau local avec l'adresse `193.54.153.127`.

Classes d'adresses, CIDR et subnet

On parlera plus loin des *classes* d'adresses et du CIDR qui les a remplacées ; on trouve encore parfois des mentions d'une notion de *subnet* (*sous-réseau*) qui n'a de sens qu'avec les classes d'adresses qui ne sont (presque) plus utilisées.

4.3 Le datagramme

L'unité de transport des données du protocole IP s'appelle un *datagramme* (en anglais : *datagram*), comme un télégramme qui transporterait des données.

Chaque datagramme est transporté indépendamment des autres : c'est une unité élémentaire pour le protocole IP. En pratique, (presque) toutes les données d'Internet sont transportées dans des datagrammes.

Comme presque toujours dans le réseau, un datagramme comprend un en-tête et des données. L'en-tête ordinaire occupe 20 octets ; nous verrons sa structure en détail plus loin dans le chapitre. Les données peuvent occuper de 0 à 65535 octets.

Usuellement, un datagramme, en-tête et données en vrac, sera transporté comme des données dans une trame de la couche liaison (ordinairement une trame Éthernet).

Comme nous le verrons plus loin, l'en-tête du datagramme contient entre autres l'adresse IPv4 de la machine destinataire des données, de manière à pouvoir l'amener à bon port.

4.4 L'émission locale d'un datagramme, ARP

Le protocole IP va décider qu'une adresse est locale en comparant son numéro de réseau avec ceux des interfaces locales. Si les deux numéros de réseaux sont égaux, alors l'adresse est locale et le datagramme peut être envoyé directement à destination en utilisant la couche 2 (Éthernet en général).

Un morceau (simplifié) de l'émission de datagrammes IP correspond donc au pseudo-code suivant :

```
x ← @IPv4 destinataire de l'en-tête du datagramme
y ← mon @IPv4
si numéro de réseau(x) == numéro de réseau(y)
  prochain ← x
sinon
  il va falloir router
envoyer le datagramme à prochain
```

Le code est simplifié principalement parce qu'un ordinateur peut avoir plusieurs adresses locales ; il va falloir comparer le numéro de réseau de l'adresse du destinataire avec celle de chacune des interfaces. Nous verrons plus loin ce

qui se passe quand l'adresse n'est pas locale et qu'il est nécessaire de *router* le datagramme.

Une fois déterminée l'adresse IP de la prochaine machine, il faut transmettre le datagramme en utilisant la couche 2, par exemple Éthernet. Comment connaître la MAC address du destinataire alors qu'on n'a que son adresse IP ? Ceci se fait à l'aide d'un protocole spécifique appelé ARP, comme *Address Resolution Protocol* (en français : *protocole de résolution d'adresse*).

Le protocole ARP spécifie un petit format de message, embarqué dans une trame Éthernet, qui contient un type (*question* ou *réponse*), l'adresse IP et la MAC Address de celui qui pose la question et l'adresse IP qui constitue la question. Cette trame est envoyée sur le réseau en broadcast ; elle est donc reçue et traitée par tous les ordinateurs actifs du réseau. La machine qui utilise cette adresse IP va ajouter dans le message sa MAC Address, remplacer le type *question* par le type *réponse* et envoyer cette réponse dans une autre trame Éthernet à la machine qui a posé la question. (Elle connaît la MAC Address du questionneur puisqu'elle apparaît dans le message qui contient la question).

Le cache ARP et la commande `arp`

L'utilisation du protocole ARP signifie que pour transmettre un datagramme avec de l'information utile, il est nécessaire de transmettre trois trames : la première avec la requête ARP, la deuxième avec la réponse ARP et la troisième qui transporte (enfin) le datagramme.

En fait, ce sur-coût n'intervient qu'exceptionnellement : la réponse à la requête ARP, qui associe adresse IPv4 et MAC address, est placée à l'intérieur d'une table qui sert de cache. Lors de l'émission d'un autre datagramme vers la même adresse IP, ce sera la MAC address trouvée dans la table qui sera utilisée directement, sans qu'il soit nécessaire de reposer la question sur le réseau.

Les entrées dans la table `arp` y sont conservées pendant quelques temps (voir exercice), puis retirées. Cela permet par exemple d'arrêter un ordinateur, de changer sa carte réseau (qui aura une nouvelle MAC address) puis de le redémarrer avec cette nouvelle MAC address et d'utiliser le réseau sans avoir besoin d'intervenir sur les autres ordinateurs pour leur faire mettre leur table à jour.

Il existe une *commande* accessible à l'utilisateur qui permet de consulter la table. Cette commande porte aussi le nom `arp`. Quand on la lance sans argument, elle se contente d'imprimer le contenu de la table. L'administrateur (root) peut aussi ajouter ou détruire des entrées dans la table.

Attention, le nom *arp* désigne plusieurs choses qu'il est important de distinguer : premièrement, ARP est un *protocole*, qui spécifie des formats de messages et des séquences d'échanges sur le réseau ; deuxièmement il désigne le *code* dans la pile TCP/IP qui implémente ce protocole (sous Unix, ce code sera dans le noyau) ; troisièmement, on l'utilise aussi comme nom pour le *cache* dans lequel sont conservées les réponses récentes aux interrogations ARP ; finalement, c'est

aussi la *commande* qui permet de manipuler cette table. Il faut souvent déduire du contexte duquel de ces quatre ARP on parle et il est important de ne pas les mélanger.

Des détails inutiles sur le protocole ARP

La section présente les détails du format des messages du protocole ARP. Ces détails sont inutiles en soi pour comprendre comment fonctionne le protocole, ou le réseau Internet. En revanche, ils sont l'occasion de montrer comment analyser des informations disponibles sur un protocole.

Une manière d'obtenir des détails sur un protocole d'Internet consiste à lire en grand détail tous les RFC qui y font référence ; souvent ce n'est pas très pratique parce que les RFC ne sont pas d'abord faites pour être comprises mais pour être précises, pour lever toute ambigüité sur la définition du protocole. (Il faut lire et comprendre les RFC pour *implémenter* un protocole.)

Une autre façon consiste à examiner comment le protocole est implémenté sur sa machine : l'exactitude de la méthode n'est pas totalement garantie puisqu'on peut propager ainsi des erreurs faites par l'implémenteur du protocole sur notre machine mais au moins on a des informations complètes et exactes sur comment ça marche dans notre ordinateur (c'est une des raisons pour lesquelles il est important de disposer des sources d'un logiciel).

Pour trouver le format des messages ARP, on peut aller le chercher dans les fichiers C à inclure du système, qui sont stockés en dessous du répertoire `/usr/include` ; on y cherche tous les fichiers où apparaît le mot `arp` :

```
$ cd /usr/include
$ grep -i arp *
curses.h:#define TRACE_CHARPUT  0x0010  /* trace all character outputs */
ncurses.h:#define TRACE_CHARPUT 0x0010  /* trace all character outputs */
resolv.h:# include <arpa/nameser.h>
```

On n'a rien obtenu de très utile ; les choses sont sans doute décrites dans un fichier stocké dans un sous-répertoire :

```
$ grep -i arp */* | wc
    418    2487   24553
```

En cherchant dans les sous-répertoires, on trouve 418 lignes qui font référence à la chaîne de caractère `arp` ; c'est beaucoup mais un survol rapide permet de voir qu'un des fichiers où le nom apparaît souvent s'appelle `linux/if_arp.h`, ce qui est prometteur.

Ce fichier `/usr/include/linux/if_arp.h` commence par la définition , avec `#define`, de constantes qui servent à identifier les différents types de matériels qui supportent le protocole ; les constantes ont un nom qui commence toujours par `ARPHRD` (comme *ARP HaRDware*). La liste débute avec

```
#define ARPHRD_NETROM   0   /* from KA9Q: NET/ROM pseudo */
#define ARPHRD_ETHER    1   /* Ethernet 10Mbps           */
```

et se termine avec

```
#define ARPHRD_IEEE80211_RADIOTAP 803 /* IEEE 802.11 + radiotap header */
#define ARPHRD_VOID       0xFFFF      /* Void type, nothing is known */
#define ARPHRD_NONE       0xFFFE      /* zero header length */
```

On trouve ensuite la définition des valeurs qui codent les opérations du protocole : ce sont les deux premières valeurs qui nous intéressent, celles qui permettent de poser une question et de fournir une réponse :

```
/* ARP protocol opcodes. */
#define ARPOP_REQUEST   1     /* ARP request   */
#define ARPOP_REPLY     2     /* ARP reply     */
#define ARPOP_RREQUEST  3     /* RARP request  */
#define ARPOP_RREPLY    4     /* RARP reply    */
#define ARPOP_InREQUEST 8     /* InARP request */
#define ARPOP_InREPLY   9     /* InARP reply   */
#define ARPOP_NAK       10    /* (ATM)ARP NAK  */
```

Suit le prototype d'une structure `arpreq`, qui est utilisée pour les *requêtes ioctl*, c'est à dire pour les appels systèmes avec lesquels un processus pourra consulter ou modifier la table ARP ; il y a aussi le prototype d'une structure `arpreq_old`, probablement le résidu d'une version antérieure du système. Cette structure nous indique des choses sur comment modifier la table ARP du système mais pas sur le protocole lui-même.

Suivent la définition de constantes qui semblent servir à qualifier les entrées dans le cache :

```
/* ARP Flag values. */
#define ATF_COM         0x02   /* completed entry (ha valid)   */
#define ATF_PERM        0x04    /* permanent entry              */
#define ATF_PUBL        0x08   /* publish entry                */
#define ATF_USETRAILERS 0x10   /* has requested trailers       */
#define ATF_NETMASK     0x20   /* want to use a netmask (only
                                  for proxy entries) */
#define ATF_DONTPUB     0x40   /* don't answer this addresses  */
```

On trouve finalement la définition de la structure qui décrit le message ARP tel qu'il transite sur le réseau ; noter que la structure porte le nom `arphdr`, comme *ARP header* (en français *en-tête ARP*) : les messages ARP se composent d'un en-tête sans données.

```
/*
 *      This structure defines an ethernet arp header.
```

```
 */

struct arphdr
{
        __be16          ar_hrd;         /* format of hardware address   */
        __be16          ar_pro;         /* format of protocol address   */
        unsigned char   ar_hln;         /* length of hardware address   */
        unsigned char   ar_pln;         /* length of protocol address   */
        __be16          ar_op;          /* ARP opcode (command)         */
#if 0
         /*
          *      Ethernet looks like this :
          *      This bit is variable sized however...
          */
        unsigned char   ar_sha[ETH_ALEN]; /* sender hardware address */
        unsigned char   ar_sip[4];        /* sender IP address       */
        unsigned char   ar_tha[ETH_ALEN]; /* target hardware address */
        unsigned char   ar_tip[4];        /* target IP address       */
#endif
};
```

La structure utilise les types `unsigned char` et `__be16` ; on devine sans peine que `unsigned char` désigne un octet et que `__be16` une valeur sur deux octets.

Comme le protocole ARP est défini pour différentes couches 2, avec des adresses de tailles diverses, la partie qui contient les adresses est de taille variable : il n'y a donc pas de moyen pratique de représenter cela avec une structure. Le source contient néanmoins la description des champs pour le cas le plus courant où on utilise une adresse Éthernet. Cependant, le code est encadré entre un `#if 0` et un `#endif` : ces lignes seront ignorées par le compilateur.

Le format du message ARP peut être représenté sous la forme de la figure 4.1 ; il permet de montrer l'importance respective des différents champs, parce que la représentation de chaque champs est proportionnelle au nombre d'octets qu'il utilise. Un autre format souvent utilisé est celui dans lequel les champs sont organisés de manière à faire apparaître les mots de 32 bits, comme dans la figure 4.2.

En regardant dans les sources d'un autre système, par exemple Plan 9, on va trouver (dans un fichier `arp.h`) une autre définition équivalente :

```
/* Format of ethernet arp request */
struct Arppkt {
        uchar   d[6];
        uchar   s[6];
        uchar   type[2];
        uchar   hrd[2];
        uchar   pro[2];
        uchar   hln;
```

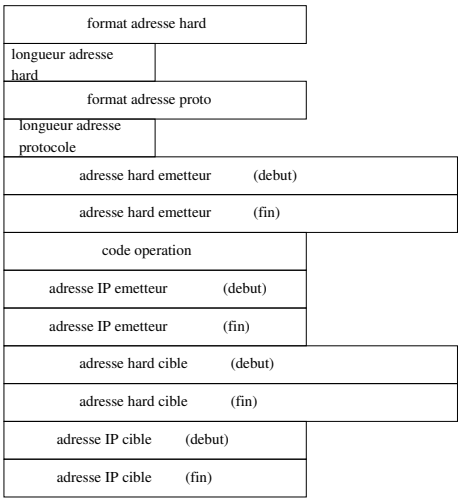

FIGURE 4.1 – Une représentation d'un message ARP.

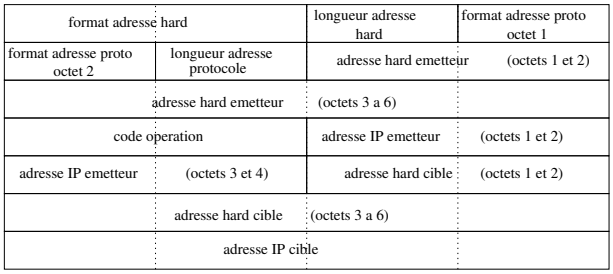

FIGURE 4.2 – Une autre représentation d'un message ARP.

```
        uchar   pln;
        uchar   op[2];
        uchar   sha[6];
        uchar   spa[4];
        uchar   tha[6];
        uchar   tpa[4];
};
```

Ici, la structure décrit le format de la trame Éthernet elle-même, avec les adresses destination et source et le type dans les trois premiers champs puis la structure du message arp dans les lignes suivantes. Tous les champs utilisent le type `uchar`, une abbréviation pour `unsigned char`, qui correspond à un octet. Les champs qui s'étendent sur plusieurs octets sont définis comme des tableaux d'`uchar`, ce qui fait sens : puisqu'on n'a pas l'intention de faire d'arithmétique dessus, il n'est pas bien utile de les définir comme des entiers sur 16 ou 32 bits.

Pour finir, voici la description du format des messages arp dans la RFC 826 :

```
    Ethernet transmission layer (not necessarily accessible to
         the user):
        48.bit: Ethernet address of destination
        48.bit: Ethernet address of sender
        16.bit: Protocol type = ether_type$ADDRESS_RESOLUTION
    Ethernet packet data:
        16.bit: (ar$hrd) Hardware address space (e.g., Ethernet,
                         Packet Radio Net.)
        16.bit: (ar$pro) Protocol address space.  For Ethernet
                         hardware, this is from the set of type
                         fields ether_typ$<protocol>.
         8.bit: (ar$hln) byte length of each hardware address
         8.bit: (ar$pln) byte length of each protocol address
        16.bit: (ar$op)  opcode (ares_op$REQUEST | ares_op$REPLY)
        nbytes: (ar$sha) Hardware address of sender of this
                         packet, n from the ar$hln field.
        mbytes: (ar$spa) Protocol address of sender of this
                         packet, m from the ar$pln field.
        nbytes: (ar$tha) Hardware address of target of this
                         packet (if known).
        mbytes: (ar$tpa) Protocol address of target.
```

Comme indiqué précédemment, les RFC ne sont pas faites pour être faciles à lire.

4.5 Le routage d'un datagramme

Lorsque le numéro de réseau de l'adresse destination ne correspond à aucun des numéros de réseau des interfaces de l'ordinateur local, il est nécessaire de *router* le datagramme : l'envoyer à un intermédiaire qui se chargera de le faire parvenir à destination.

Sans surprise, les machines qui routent sont appelées des *routeurs* (en anglais, *router* ou *gateway*). A priori, tous les ordinateurs sous Unix sont susceptibles d'être utilisés comme routeurs ; cependant, dans la plupart des cas, on confie ce rôle à une machine spécialisée qui sera plus facile à mettre en place et à maintenir, pour un coût moindre et avec des performances supérieures à celle d'un ordinateur courant. À l'heure actuelle, plus de la moitié des routeurs vendus sont issus du même constructeur appelé CISCO, qui est en position de définir des normes de fait.

La manière de router un datagramme est élémentaire : il existe dans la pile TCP/IP une *table de routage*, qui contient les routeurs auxquels on peut envoyer des messages et des informations sur les routeurs à utiliser suivant les adresses de destinations. Cette table de routage sera remplie par des commandes ou *par des protocoles de plus haut niveau* qui utiliseront IP pour échanger des données.

La vision (à peine simplifiée) du traitement d'un datagramme devient :

```
x ← @IPv4 destinataire de l'en-tête du datagramme
y ← mon @IPv4
si numéro de réseau(x) == numéro de réseau(y)
  prochain ← x
sinon
 prochain ← route(x)
envoyer le datagramme à prochain
```

Tout le travail est fait par la fonction `route` qui va choisir l'adresse du prochain routeur qui devra traiter le datagramme, sur la base de la table de routage et de l'adresse destination.

Pour les ordinateurs sur le réseau, il y a grosso modo trois sortes de machines : celles qui se trouvent dans une feuille du réseau (comme un ordinateur personnel ou le boîtier de connexion fourni par un fournisseur d'accès) ; celles qui jouent le rôle de routeur interne dans une organisation et finalement celles qui se trouvent à la limite d'une organisation. Nous illustrons ceci avec une vision simplifiée du de l'ancien réseau de l'université Paris 8 sur le site de Saint Denis (figure 4.3).

Pour une machine dans une feuille, la table de routage n'aura besoin de contenir qu'une seule entrée : la route par défaut qui conduit au routeur qui la relie au reste du réseau.

Les routeurs de l'université ont une table de routage qui contient tous les numéros de réseau utilisés sur le site et une route par défaut vers le routeur qui nous relie au reste du réseau, à travers notre fournisseur d'accès internet. Les tables de routages sont mises à jour soit à travers des commandes de configura-

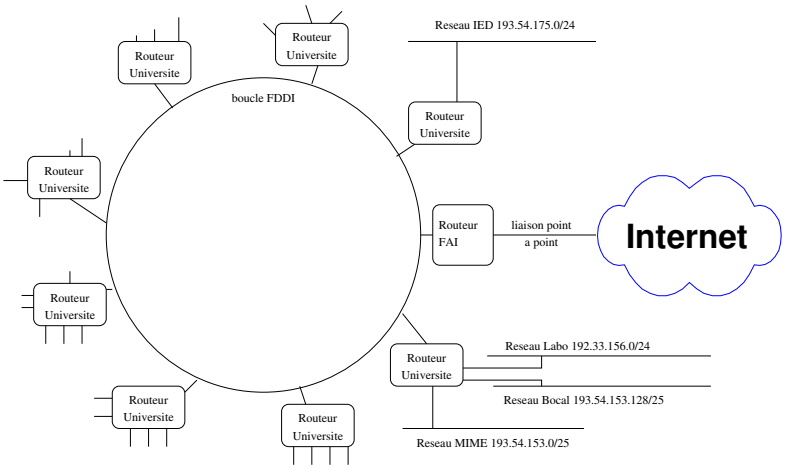

FIGURE 4.3 – Une vision simplifiée de l'ancienne organisation du réseau de l'université de Paris 8 : les départements connectent leurs machines via un réseau Éthernet relié au réseau de l'université par un routeur unique : c'est leur route par défaut. Les routeurs départementaux sont reliés entre eux par une boucle FDDI ; ils échangent leurs tables de routage, ce qui leur permet de connaître tous les numéros de réseau de l'université. Les datagrammes destinés à d'autres adresses sont envoyés au routeur qui nous relie à notre fournisseur d'accès.

tion entrées manuellement, soit en laissant chaque routeur échanger le contenu de sa table de routage avec ses routeurs voisins ; les deux protocoles les plus fréquemment utilisés pour ce faire sont RIP (*Routing Internet Protocol*, qui commence à dater) et OSPF (*Open Shortest Path First*).

Entre les routeurs d'un fournisseur d'accès, le mécanisme est identique à ce qui passe dans les routeurs de Paris 8, à ceci près qu'il existe (normalement) plusieurs routes différentes qui permettent d'accéder au reste d'Internet.

Finalement, les échanges entre les différents fournisseurs d'accès à Internet (FAI) se passent dans des locaux spécialisés qui servent de plate-forme d'échange : chaque fournisseur d'accès connecte une de ses machines dans ce local à son réseau (probablement via une ligne spécialisée) et tous les ordinateurs de tous les FAI présents sont connectés ensemble par un réseau local rapide. Il y a une quinzaine de plates-formes de ce type en France, dont une dizaine à Paris.

(L'une des plates-formes d'échange s'appelle MAE-Paris et pour une fois le MAE n'est l'acronyme de rien du tout : dans les débuts d'Internet, il y avait deux plates-formes aux États Unis, l'une sur la cote Est et l'autre sur la cote Ouest ; celle de la cote est à reçu le nom de MAE-East pour pouvoir donner un nom d'actrice célèbre à celle de la cote Ouest. Depuis le nom MAE à été réutilisé dans de nombreuses autres plates-formes. Beaucoup d'autres plates-formes ont un nom qui se termine en IX comme *Internet eXchange*.)

Il y a des accords entre les différents fournisseurs d'accès qui vont accepter ou pas de router les datagrammes d'un autre FAI en fonction de considérations politico-commerciales : en effet, un FAI doit bien évidemment accepter les datagrammes qui sont destinés aux ordinateurs de ses clients mais il n'accepte pas nécessairement de router sans compensation les datagrammes d'un client d'un autre FAI destiné à un troisième FAI. Le protocole utilisé dans ces routeurs à la frontière des FAI s'appelle BGP comme *Border Gateway Protocol*.

4.6 Les détails de l'en-tête d'un datagramme IP

Dans cette section, nous allons examiner le contenu des 20 octets de l'en-tête ordinaire d'un datagramme IP. Ce sera l'occasion d'apporter quelques précisions sur le fonctionnement du protocole IP.

4.6.1 L'en-tête lui-même

Dans le fichier `/usr/include/linux/in.h`, on trouve la définition d'une structure pour décrire une adresse IPv4 :

```
struct in_addr {
        __be32  s_addr;
};
```

Elle nous indique bien qu'une adresse IPv4 est en fait un mot de 32 bits.

Dans le fichier `/usr/include/netinet/ip.h`, on trouve la définition de la structure de l'en-tête IP ordinaire :

```
struct ip
  {
    unsigned int ip_v:4;           /* version */
    unsigned int ip_hl:4;          /* header length */
    u_int8_t ip_tos;               /* type of service */
    u_short ip_len;                /* total length */
    u_short ip_id;                 /* identification */
    u_short ip_off;                /* fragment offset field */
#define IP_RF 0x8000               /* reserved fragment flag */
#define IP_DF 0x4000               /* dont fragment flag */
#define IP_MF 0x2000               /* more fragments flag */
#define IP_OFFMASK 0x1fff          /* mask for fragmenting bits */
    u_int8_t ip_ttl;               /* time to live */
    u_int8_t ip_p;                 /* protocol */
    u_short ip_sum;                /* checksum */
    struct in_addr ip_src, ip_dst; /* source and dest address */
  };
```

Nous allons examiner chacun des champs de cette structure.

4.6.2 Version et longueur de l'en-tête

```
    unsigned int ip_v:4;           /* version */
    unsigned int ip_hl:4;          /* header length */
```

Le premier octet de l'en-tête est divisé en deux morceaux de quatre bits : les quatre premiers bits contiennent le numéro de version du protocole IP auquel appartient l'en-tête : dans notre cas, la valeur 4. Les quatre autres bits du même octet contiennent la longueur de l'en-tête, exprimée comme un nombre de mots de 32 bits (quatre octets) : un en-tête IP utilise normalement 20 octets et ce champs contiendra donc la valeur 5 mais il existe des options qui peuvent faire grossir l'en-tête ; puisqu'on n'a que 4 bits pour exprimer cette longueur, le champs ne peut prendre qu'une valeur entre 0 et 15 ; cela signifie qu'un en-tête ne pourra pas faire plus de $15 \times 4 = 60$ octets.

Nous regarderons les autres champs en commençant par la fin de la structure.

4.6.3 Adresses source et destination

```
    struct in_addr ip_src, ip_dst; /* source and dest address */
```

Le dernier champs contient les adresses IP source et destination du datagramme : la source est le premier ordinateur qui a construit et émis le datagramme, la destination indique l'adresse finale. Si un datagramme envoyé par A à la machine B est routé par les routeurs X et Y, la trame Éthernet dans laquelle X envoie le datagramme à Y aura bien les MAC address de X et Y comme source et destination de la trame ; le datagramme (transporté comme des données dans la trame) contiendra lui les adresses IP de A et B comme adresses source et destination du datagramme.

Ces adresses contiennent donc les adresses IP de la machine d'origine du datagramme et celle de la machine destination, sans modification suivant les liens entre les routeurs que le datagramme traversera.

4.6.4 Le *checksum*

```
u_short ip_sum;                 /* checksum */
```

Le checksum (*somme de contrôle* en français) est utilisé pour détecter les erreurs de transmission de l'en-tête du datagramme. IP n'essaye pas de détecter les erreurs de transmission des données (peut-être que les données utilisent un code auto-correcteur qui permettra de corriger l'erreur). En revanche, s'il y a une erreur sur l'en-tête du datagramme, il n'y a plus moyen de le router d'une façon fiable et le récepteur doit le jeter.

4.6.5 Le protocole

```
u_int8_t ip_p;                  /* protocol */
```

L'octet `p` (comme *protocole*) indique de quelle manière les données transportées doivent être interprétées. On y trouve une valeur numérique qui code un protocole de niveau supérieur comme TCP, UDP ou ICMP.

4.6.6 Le *time to live*

```
u_int8_t ip_ttl;                /* time to live */
```

L'octet `ttl`, comme *time to live*, est utilisé pour éviter que des datagrammes se promènent indéfiniment de routeur en routeur sur le réseau. Chaque fois qu'un routeur reçoit puis retransmet un datagramme, il décrémente ce champs ; quand le champs atteint la valeur 0, le routeur doit jeter le datagramme.

Puisque le champs est dans un octet, on ne peut y mettre qu'une valeur entre 0 et 255 ; cela signifie qu'un datagramme ne peut pas traverser plus de 255 routeurs. Par défaut, ce champs est initialisé à 64 lors de la création d'un nouveau datagramme.

Un inconvénient du champs ttl est que chaque routeur le modifie et doit donc recalculer le checksum de l'en-tête.

4.6.7 L'identificateur

```
    u_short ip_id;                  /* identification */
```

Chaque datagramme reçoit à sa création un numéro probablement unique, qui sert à l'identifier dans la machine destinataire. Cela facilite la détection des duplications éventuelles et surtout la reconstruction des fragments dont nous parlons plus loin.

4.6.8 La longueur

```
    u_short ip_len;                 /* total length */
```

La longueur est stockée sur deux octets : elle décrit la taille en octet des données transportées dans le datagramme. Cela signifie qu'un datagramme ne pourra pas transporter plus de 64 Kilo-octets de données.

4.6.9 Le *type of service*

```
    u_int8_t ip_tos;                /* type of service */
```

Le champs `tos` code sur 8 bits le genre de service pour lequel le datagramme est utilisé : on a trois bits qui servent à indiquer l'importance du datagramme et quatre bits qui peuvent servir à demander soit une route qui arrive rapidement à destination, soit une route avec un gros débit, soit une route avec peu d'erreurs, soit une route bon marché.

4.6.10 Le MTU, la fragmentation

```
    u_short ip_off;                 /* fragment offset field */
#define IP_RF 0x8000                /* reserved fragment flag */
#define IP_DF 0x4000                /* dont fragment flag */
#define IP_MF 0x2000                /* more fragments flag */
#define IP_OFFMASK 0x1fff           /* mask for fragmenting bits */
```

Chaque datagramme est embarqué en entier à l'intérieur d'une trame de la technologie de niveau 2, ce qui impose une taille maximum sur le volume de données transportées. Par exemple, on a vu que la taille maximum des données d'une trame Éthernet était de 1500 octets. D'autres techniques peuvent utiliser des tailles plus courtes ou plus grosses ; par exemple, les liaisons satellites permettent d'échanger des paquets de données dont la taille est de l'ordre du méga octets, alors que sur une liaison lente par ligne série, on souhaite la limiter pour améliorer l'interactivité. On appelle cette taille le MTU comme *Maximum Transmit Unit*.

Cela pose un problème parce que l'émetteur ne sait pas, lors de l'émission, par quelle route le datagramme va passer et donc il ignore également quelles technologies de niveau 2 seront employées pour le transporter : comment choisir la quantité de données maximum à placer dans chaque datagramme ?

La réponse originale a été d'utiliser la *fragmentation* ; quand un routeur recevait un datagramme trop gros pour être transporté par la technologie de niveau 2 de l'étape suivante, il était découpé en fragments : les données étaient découpées en morceaux de la taille adéquate, l'en-tête IP était dupliqué dans chacun d'eux avec le champs `off` qui indiquait la place des données qu'il transportait dans le datagramme initial et le bit MF (*More Fragment*) ajouté dans tous les fragments sauf le dernier. Ensuite, chacun des fragments était routé indépendamment comme un datagramme ordinaire jusqu'à la destination finale qui collectait les fragments et reconstruisait le datagramme original. Notez qu'un autre routeur sur le chemin disposait de toute l'information nécessaire pour re-découper un fragment en fragments plus petits si nécessaire.

A l'heure actuelle, la fragmentation n'est presque plus utilisée. En général, l'émetteur d'un datagramme IP positionne dans le champs offset le bit DF (*Don't fragment*) qui interdit au routeur de fragmenter. Quand un routeur se retrouve avec un paquet trop gros, il répond alors avec un message d'erreur (contenu dans un message du protocole ICMP dont nous parlons dans le prochain chapitre) et la machine initiale renvoie les données dans des datagrammes plus petits. On parle alors de *Path MTU discovery* (en français *découverte de l'unité de transmission maximale du chemin*).

Le Path MTU nécessite que l'ordinateur qui émet un datagramme en conserve une copie pour être en mesure de retransmettre les données pendant le temps maximum pendant lequel il peut recevoir le message d'erreur.

4.7 Les adresses IPv4 : le retour

Dans cette section, je complète les informations sur les adresses IP données dans le début du chapitre avec des choses bonnes à savoir mais qui me semblent moins importantes.

4.7.1 L'adresse locale

Toutes les implémentations de TCP/IP utilisent une adresse spéciale, dite *loopback* (en français *adresse de bouclage*) : cette adresse désigne la machine elle-même.

Toutes les adresses qui commencent par `127` sont des adresses de l'interface de loopback. On utilise généralement la première des adresses disponibles, c'est à dire 127.0.0.1.

4.7.2 Les adresses privées

Au début de l'attribution des adresses IP, un certain nombre d'adresses ont été mises de coté pour permettre à chacun de configurer des réseaux qui utilisent TCP/IP avec des numéros IP valides sans avoir besoin de demander des numéros de réseaux aux autorités qui distribuent les adresses : on les appelle des adresses *privées*.

Les adresses privées sont des adresses valides mais les routeurs situés à la limite d'une organisation (que ce soit le routeur vers Internet d'une entreprise ou le boîtier d'un fournisseur d'accès au domicile d'un particulier) ne va pas router les datagrammes qui utilisent ces adresses : leur usage est *privé*, cantonné à l'intérieur de l'organisation.

Il y a trois groupes d'adresses, qui correspondent à des adresses de classe A, B et C (voir la section suivante). Il s'agit des adresses `10.0.0.0/8`, `172.16.0.0/12` et `192.168.0.0/16`.

Nous reparlerons des adresses privées dans le chapitre qui traite du NAT et des Firewalls.

4.7.3 Les classes d'adresse

À l'origine d'Internet, il y avait une convention sur le nombre de bits du numéro de réseau qui conduisait à diviser les adresses en *classes*.

Les grands réseaux de classe A

Si le bit de poids fort du premier octet était égal à 0, on avait ce qu'on appelle une adresse de *classe A* : le numéro de réseau occupait les huit bits de poids forts et le numéro de machine les 24 bits de poids faibles. On pouvait donc avoir sur Internet 126 réseaux de classe A et chacun de ces réseaux pouvait contenir environ $2^{24} = 2^4 \times 2^{20} \sim 16 \times (2^{10})^2 \sim 16 \times (10^3)^2 \sim 16 \times 10^6 \sim 16$ millions de machines différentes. Puisque le bit de poids fort de l'octet de gauche est à zéro, le premier nombre d'une adresse IP de classe A était compris entre 1 et 126 ; les réseaux allaient de 1.0.0.0/8 à 126.0.0.0/8.

Les réseaux intermédiaires de classe B

Si le bit de poids fort du premier octet était à 1 et le bit suivant à 0, on avait une adresse de classe B : le numéro de réseau occupait les deux octets de poids forts et le numéro de machine les deux octets de poids faibles. Puisque deux bits sont fixés, il reste 14 bits dans le numéro de réseau et on avait donc $2^{14} \sim 16\,000$ numéros de réseaux de classe B, susceptibles de contenir chacun 65534 machines. Les réseaux allaient de 128.1.0.0/16 à 191.255.0.0/16.

Les petits réseaux de classe C

Si les deux bits de poids forts étaient à 1 et le bit suivant à 0, alors on avait un réseau de classe C, dans lequel le numéro de réseau occupait trois octets et le numéro de machine l'octet suivant. Puisque les trois bits de poids forts sont fixés, il reste 21 bits dans les trois octets de poids fort pour numéroter le réseau ; on avait donc $2^{21} \sim 2$ millions de réseaux de classe C, susceptibles de contenir chacun 254 machines. Les réseaux allaient de 192.0.0.0/24 jusqu'à 223.255.255.0/24.

La classe D pour le *multicast*

Avec les trois bits de poids forts à 1 et le bit suivant à 0, on a une adresse de *multicast*, utilisée par un émetteur pour envoyer en une fois des données à plusieurs destinataires. Les adresses de multicast vont de 224.0.0.0 jusqu'à 239.255.255.255.

La classe E

Avec les quatre bits de poids forts à 1, les adresses vont de 240.0.0.0 jusqu'à 255.255.255.255 : ce sont des adresses de classe E, qui ne sont pas utilisées à l'heure actuelle.

Le masque de sous-réseau

A l'intérieur d'un réseau (de classe A, B ou C), il peut être souhaitable de diviser les adresses disponibles en sous-réseaux indépendants. En plus du masque de réseau, on accompagnait dans ce cas l'adresse IPv4 d'un masque de *sous-réseau* qui indiquait quels bits du numéro de machine devaient en fait être considérés comme faisant partie du numéro de réseau. A l'heure actuelle, on intègre presque toujours ce masque de sous-réseau dans le masque de réseau.

Attribution des adresses

Aux débuts d'Internet, il était facile d'obtenir une adresse de classe C : il suffisait de télécharger (par le réseau) un formulaire, de le remplir et de le renvoyer par mail à l'organisme chargé de gérer les adresses sur Internet. Cela a changé aux alentours de 1994, quand on a commencé à entrevoir une pénurie d'adresse IPv4 et que le mécanisme des classes à été remplacé par le CIDR, présenté dans la section suivante. A l'heure actuelle, on obtient des numéros de réseau de son fournisseur d'accès. (Un utilisateur ordinaire ne reçoit qu'une seule adresse IP de son fournisseur d'accès mais peut combiner les adresses privées présentées plus haut et la translation d'adresse (NAT) qui sera présentée plus loin pour obtenir un résultat à peu près équivalent.)

4.7.4 CIDR

Vers 1995, le mécanisme des classes d'adresses décrit dans le paragraphe précédent a explosé : il n'y avait plus d'adresses de classe B disponibles, parce que de nombreuses institutions qui avaient besoin de plus des 254 adresses d'un réseau de classe C demandaient (et obtenaient) un numéro de classe B. D'autre part, le nombre de réseau de classe C augmentait vertigineusement et on commençait à se trouver dans une situation où les tables de routage deviendraient trop grosses pour permettre aux routeurs de travailler suffisamment vite.

La solution adoptée a été d'abandonner le mécanisme des classes d'adresses et de mettre en route un nouveau plan d'adressage, nommé *Classless Internet Domain Routing* (en français : *routage de domaine d'adresses sans classe*).

Les adresses de classe C qui n'étaient pas encore attribuées ont été divisées en grands blocs d'adresses par continent, puis chaque continent a divisé son espace d'adressage en sous-classes suivant les pays, chaque pays suivant les fournisseurs d'accès et ainsi de suite.

Les adresses attribuées avant la mise en place du CIDR n'ont pas bougées. C'est la raison pour laquelle il y a encore un réseau de classe C à l'université dont le numéro est 192.33.156.0/24, attribué au laboratoire d'informatique avant la mise en place du CIDR.

On trouve une représentation condensée et plaisante de l'utilisation des numéros de réseau IP sur une des pages de `xkcd`, une excellente collection de dessins, à l'adresse `http://xkcd.com/195/`.

4.8 En pratique

Toutes les informations qui précèdent peuvent s'illustrer par des commandes présentées dans cette section.

4.8.1 La commande `ifconfig`

L'association de la pile IP avec une interface se fait à l'aide de la commande `ifconfig`. Quand on la lance sans argument, elle donne l'état de toutes les interfaces. Chaque interface est décrite par une demie douzaine de lignes. Elle porte un nom qui commence par quelques caractères alphabétiques qui indiquent son type, suivi d'un chiffre qui sert à identifier les interfaces du même type. Si la pile TCP/IP est lancée, `ifconfig` sans arguments doit au moins donner le nom de l'interface de *loopback*, en général `lo0` et probablement celle de l'Éthernet filaire qui sous Linux s'appelle presque toujours `eth0`. Il peut y avoir d'autres interfaces configurées sur votre ordinateur. A partir d'ici, je supposerai que votre interface filaire porte le nom `eth0`.

On peut demander l'état d'une interface particulière en donnant son nom comme argument de la commande :

```
$ ifconfig lo0
lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:82 errors:0 dropped:0 overruns:0 frame:0
          TX packets:82 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:8056 (7.8 KB)  TX bytes:8056 (7.8 KB)
```

La deuxième ligne nous indique l'adresse IPv4 et son masque associés à l'interface. La suivante nous indique un numéro IPv6 dont nous parlerons plus loin. La troisième ligne contient les caractéristique de l'interface, notamment son MTU. On a ensuite des compteurs pour le nombre de paquets reçus et émis, avec un compteur de collisions et une indication du nombre de paquets en attente et finalement des compteurs pour le nombre d'octets traités.

On peut facilement dissocier une interface de la pile TCP-IP avec la commande (pour root) :

```
# ifconfig eth0 down
```

Vous pouvez constater qu'ensuite le réseau ne fonctionne plus. Il est utile de connaître cette commande car il est nécessaire d'arrêter le réseau filaire si on veut utiliser le Wifi. (Quand la machine a le choix entre le Wifi et le filaire, elle choisit le filaire qui est plus rapide et surtout plus fiable.) On peut redémarrer l'interface avec

```
# ifconfig eth0 up
```

mais il n'y a plus de route par défaut.

4.8.2 La commande `ping`

Une façon élémentaire d'utiliser le réseau est la commande `ping` : elle boucle sur l'envoi d'un datagramme (à chaque seconde) et l'affichage du datagramme reçu en retour. Nous rentrerons plus en détail dans le protocole utilisé par `ping` dans le prochain chapitre. On peut l'interrompre avec un CTRL-C, comme un processus ordinaire, ou bien lui donner avec l'option `-c` le nombre de paquets à envoyer.

```
# ping 127.0.0.1
PING 127.0.0.1 (127.0.0.1) 56(84) bytes of data.
64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.029 ms
64 bytes from 127.0.0.1: icmp_seq=2 ttl=64 time=0.016 ms

--- 127.0.0.1 ping statistics ---
```

```
2 packets transmitted, 2 received, 0% packet loss, time 999ms
rtt min/avg/max/mdev = 0.016/0.022/0.029/0.008 ms
```

Le temps affiché sur chaque ligne est celui d'un aller-retour, en anglais *round trip time* abrégé en `rtt`. Pour transporter du trafic téléphonique dans de bonnes conditions, il doit être de l'ordre de 10 milli-secondes.

Dans les exemples d'utilisation de `ping` qui suivent, je ne recopierai pas dans le support les trois lignes de statistiques qu'il affiche avant de sortir.

Ping est une commande élémentaire, qu'il est utile d'avoir sous la main pour tester si un réseau fonctionne ou pas, plutôt que d'utiliser d'autres commandes plus complexes qui peuvent ne pas fonctionner pour de nombreuses autres raisons. À l'oral, on utilise le verbe *pinger* (prononcer *pineguer*) pour dire qu'on l'utilise pour tester la connexion avec une machine, comme dans *« J'ai rebooté le serveur, essaye de le pinger, s'il te plaît. »*.

4.8.3 La commande `arp`

Il existe une commande `arp` pour examiner le contenu de la table ARP. (Root peut aussi utiliser cette commande pour retirer ou ajouter des entrées permanentes dans cette table.)

Au boot, si le noyau a correctement configuré le réseau, il y a toutes probabilités pour que la table contienne une entrée, celle qui décrit le routeur qui vous connecte à Internet. A mon domicile, j'obtiens ce résultat (je suis connecté à Internet par une FreeBox) :

```
$ arp -n
Address       HWtype  HWaddress         Flags Mask Iface
192.168.0.254 ether   00:07:CB:0D:13:EA C          eth0
```

J'accède maintenant à une autre machine locale, connectée elle aussi à ma FreeBox (j'ai un petit hub branché sur la FreeBox et les deux machines sont branchées sur le hub).

```
$ ping 192.168.0.98
PING 192.168.0.98 (192.168.0.98) 56(84) bytes of data.
64 bytes from 192.168.0.98: icmp_seq=1 ttl=64 time=2.59 ms
64 bytes from 192.168.0.98: icmp_seq=2 ttl=64 time=0.387 ms
```

Une nouvelle entrée apparaît dans la table ARP pour décrire cette machine :

```
$ arp -n
Address            HWtype  HWaddress           Flags Mask Iface
192.168.0.254      ether   00:07:CB:0D:13:EA   C          eth0
192.168.0.98       ether   00:04:75:D7:CE:DE   C          eth0
```

Si j'essaye d'accéder à une machine locale qui n'existe pas, il y a une entrée incomplète qui apparaît dans la table ARP. Tentative d'accès à la machine :

```
$ ping 192.168.0.253
PING 192.168.0.253 (192.168.0.253) 56(84) bytes of data.
From 192.168.0.10 icmp_seq=1 Destination Host Unreachable
From 192.168.0.10 icmp_seq=2 Destination Host Unreachable
From 192.168.0.10 icmp_seq=3 Destination Host Unreachable
--- 192.168.0.253 ping statistics ---
6 packets transmitted, 0 received, +3 errors, 100% packet loss, time 5016ms
```

Nouvel état de la table :

```
$ arp -n
Address          HWtype  HWaddress           Flags Mask  Iface
192.168.0.254    ether   00:07:CB:0D:13:EA   C           eth0
192.168.0.253            (incomplete)                    eth0
192.168.0.98     ether   00:04:75:D7:CE:DE   C           eth0
```

Si j'attends suffisamment, les entrées inutilisées vont disparaître de la table.

Si j'accède à une machine distante, le datagramme sera routé vers la FreeBox et il n'y aura donc pas de nouvelle entrée dans la table pour cette machine. Accès à la machine :

```
$ ping 192.33.156.10
PING 192.33.156.10 (192.33.156.10) 56(84) bytes of data.
64 bytes from 192.33.156.10: icmp_seq=1 ttl=54 time=32.2 ms
64 bytes from 192.33.156.10: icmp_seq=2 ttl=54 time=32.1 ms
```

Nouvel état de la table :

```
$ arp -n
Address          HWtype  HWaddress           Flags Mask  Iface
192.168.0.254    ether   00:07:CB:0D:13:EA   C           eth0
192.168.0.253            (incomplete)                    eth0
192.168.0.98     ether   00:04:75:D7:CE:DE   C           eth0
```

4.8.4 La commande `tcpdump`

Il existe des commandes qui placent une interface en mode promiscuous puis affichent dans un format lisible le contenu de chacune des trames Éthernet reçue. Les deux commandes les plus utilisées sont `etherreal` (que nous n'utiliserons pas) et `tcpdump`.

La commande `tcpdump` se lance dans un terminal. Il faut être super-utilisateur pour avoir le droit de s'en servir. On lui donne en option l'interface sur laquelle écouter, puis une expression qui indique les trames qui nous intéressent. (L'expression utilisée pour choisir les trames intéressantes peut devenir redoutablement complexe.)

En n'écoutant que les trames ARP sur l'interface `eth0` :

```
# tcpdump -i eth0 -n arp
listening on eth0, link-type EN10MB (Ethernet), capture size 96 bytes
16:13:55.247579 arp who-has 192.168.0.254 tell 192.168.0.10
16:13:55.247957 arp reply 192.168.0.254 is-at 00:07:cb:0d:13:ea
16:14:05.428905 arp who-has 192.168.0.10 tell 192.168.0.254
16:14:05.428921 arp reply 192.168.0.10 is-at 00:1e:8c:5c:e1:13
```

On voit passer les requêtes ARP (étiquetées avec *who-has*) et les réponses (avec *reply*) ; le contenu de chaque trame est imprimé d'une façon compréhensible.

4.8.5 La commande `route`

On peut regarder la table de routage avec la commande `route` appelée avec les options `-e -n`. (On obtient la même chose avec `netstat -rn`.) Sur mon ordinateur sans Wifi, j'obtiens :

```
$ route -e -n
Kernel IP routing table
Destination  Gateway       Genmask       Flags MSS Window irtt Iface
192.168.0.0  0.0.0.0       255.255.255.0 U       0 0         0 eth0
0.0.0.0      192.168.0.254 0.0.0.0       UG      0 0         0 eth0
```

Le contenu de la table indique que les paquets destinés au réseau 192.168.0.0/24 sont à envoyer directement à travers l'interface `eth0` et tous les autres au routeur 192.168.0.254 qui se chargera de les acheminer.

La commande `route` permet aussi de manipuler la table de routage. Dans l'exemple suivant, je supprime une route et constate qu'il n'y a plus moyen d'accéder aux machines extérieures.

État initial de la table de routage :

```
# route -en
Kernel IP routing table
Destination  Gateway       Genmask       Flags MSS Window irtt Iface
192.168.0.0  0.0.0.0       255.255.255.0 U       0 0         0 eth0
0.0.0.0      192.168.0.254 0.0.0.0       UG      0 0         0 eth0
```

J'acquiers une adresse IP et je teste qu'on peut l'atteindre. (Nous parlerons de la commande `host` dans un chapitre ultérieur.)

```
# host www.iedparis8.net
www.iedparis8.net has address 195.154.244.137
# ping -c 1 195.154.244.137
PING 195.154.244.137 (195.154.244.137) 56(84) bytes of data.
64 bytes from 195.154.244.137: icmp_seq=1 ttl=56 time=32.3 ms
```

Je supprime la route par défaut de la table de routage et je regarde le nouvel état de la table : il ne reste que la route vers les machines voisines.

```
# route del default
# route -en
Kernel IP routing table
Destination Gateway Genmask       Flags MSS Window irtt Iface
192.168.0.0 0.0.0.0 255.255.255.0 U       0 0         0 eth0
```

Il n'y a pas moyen d'atteindre une machine extérieure :

```
# ping -c 1 195.154.244.137
connect: Network is unreachable
```

En revanche, on peut atteindre une machine voisine (par exemple le routeur) :

```
# ping -c 1 192.168.0.254
PING 192.168.0.254 (192.168.0.254) 56(84) bytes of data.
64 bytes from 192.168.0.254: icmp_seq=1 ttl=64 time=7.41 ms
```

Je remet la route par défaut en place et j'imprime la table de routage :

```
# route add default gw 192.168.0.254
# route -en
Kernel IP routing table
Destination Gateway       Genmask       Flags MSS Window irtt Iface
192.168.0.0 0.0.0.0       255.255.255.0 U       0 0         0 eth0
0.0.0.0     192.168.0.254 0.0.0.0       UG      0 0         0 eth0
```

Je peux de nouveau accéder aux machines distantes :

```
$ ping -c 1 195.154.244.137
PING 195.154.244.137 (195.154.244.137) 56(84) bytes of data.
64 bytes from 195.154.244.137: icmp_seq=1 ttl=55 time=1.16 ms
```

4.8.6 La commande `traceroute`

La commande `traceroute` permet de connaître le chemin suivi par des datagrammes pour atteindre une machine distante, en manipulant le ttl de l'en-tête IP. Elle commence par émettre trois datagrammes avec un ttl à 0, reçoit normalement trois messages d'erreur qui lui permettent de connaître l'identité du premier routeur ; ensuite elle recommence avec un ttl de 1, puis 2, puis 3 et ainsi de suite jusqu'à ce que le datagramme arrive à destination. Exemple d'utilisation vers une machine japonaise :

```
$ traceroute -n www.yahoo.jp
traceroute to www.yahoo.jp (124.83.139.192), 30 hops max, 40 byte packets
 1  192.168.0.254  1.122 ms  1.438 ms  1.843 ms
 2  82.224.154.254  33.725 ms  34.552 ms  35.578 ms
 3  * * *
```

```
 4  212.27.50.13  39.780 ms  40.960 ms  42.430 ms
 5  212.27.57.9  42.761 ms * *
 6  212.27.58.26  126.453 ms * *
 7  206.223.115.23  127.531 ms  114.257 ms  114.431 ms
 8  216.98.96.174  181.238 ms  180.562 ms  181.344 ms
 9  216.98.96.102  295.530 ms  295.624 ms  318.746 ms
10  58.138.81.174  306.257 ms 58.138.81.90  305.818 ms  305.008 ms
11  58.138.81.130  304.498 ms  303.923 ms  304.257 ms
12  58.138.106.38  303.531 ms 58.138.106.34  303.569 ms
                                          58.138.106.38  303.403 ms
13  210.130.135.142  298.187 ms  299.030 ms  299.339 ms
14  124.83.128.30  304.750 ms  306.364 ms  304.547 ms
15  124.83.128.50  301.026 ms  301.365 ms  302.050 ms
16  124.83.128.50  304.436 ms !X * *
```

Les étoiles dans la sortie correspondent à des réponses qui ne sont pas revenues.

4.8.7 Kit de survie Internet

Cette section reprend les commandes vues précédemment du point de vue d'un utilisateur qui cherche à connecter son ordinateur à Internet dans un environnement inconnu.

- Confirmer que le périphérique réseau est bien reconnu par le système, en vérifiant qu'il apparaît dans ceux listés par `ifconfig`.
- Il faut demander un mode d'emploi à l'administrateur et le suivre. Les recettes qui suivent ne s'appliquent que si on n'a pas de contact avec l'administrateur.
- Trouver une prise Éthernet et se brancher dessus. Il y a souvent des leds proches des prises rj45 sur les ordinateurs et les hubs qui peuvent permettre de vérifier qu'on est connecté.
- L'ordinateur est sans doute configuré pour tenter de se connecter automatiquement. Si la configuration automatique réussit (quelques secondes peuvent être nécessaires), le travail est fait.
- Si la configuration automatique échoue, repérer les numéros de réseaux utilisés avec `tcpdump`. (Même si on est sur un switch, on va recevoir les requêtes ARP qui sont envoyées en broadcast et donc propagées par le switch à toutes les machines branchées.)
- Choisir une adresse IP inutilisée sur ce réseau. *Attention*, c'est la seule étape un peu dangereuse : si on devine mal et qu'on réemploie une adresse déjà attribuée à une autre machine, cette autre machine n'aura plus d'accès au réseau. Supposons que nous choisissons l'adresse 193.231.122.123. Nous l'associons à l'interface avec
  ```
  ifconfig eth0 193.231.122.123
  ```
- Tester que la configuration a fonctionné en vérifiant qu'on peut pinguer une adresse locale (dont on a repéré l'adresse avec tcpdump) ; par exemple :

```
ping -c 1 193.231.122.111
```
On peut aussi utiliser la commande (beaucoup plus complexe)
```
nmap -sP 193.231.122.0/24
```
pour tester toutes les machines locales.

— Si le ping ne fonctionne pas, c'est peut-être que les ordinateurs locaux ont été configurés pour ne pas répondre à ses requêtes. Vérifier si les MAC address ont été résolues en consultant la table ARP : dans ce cas c'est bon aussi.

— Deviner le numéro IP du routeur par défaut du réseau. Souvent les routeurs utilisent un numéro de machine qui vaut 1 ou 254. Vérifier qu'on peut l'atteindre avec ping.
```
ping -c 1 193.231.122.254
```

— Ajouter une route par défaut vers le routeur avec par exemple :
```
route add default gw 193.231.122.254
```

— Vérifier qu'on peut atteindre une machine distante (c'est pratique pour cela de connaître par cœur au moins une adresse IP d'un ordinateur qui reste allumé en permanence) :
```
ping -c 1 195.154.244.137
```

— S'il n'y a pas moyen d'atteindre la machine, essayer de comprendre avec traceroute quel est le routeur qui pose problème. Si c'est le routeur local, c'est peut-être parce qu'il est configuré pour ne laisser passer que les adresses IP affectées par l'administrateur, ou bien les MAC address de ses machines. On peut essayer de débrancher une machine locale et/ou de modifier la MAC address de notre machine pour réutiliser l'adresse IP et/ou la MAC address de la machine débranchée. Attention c'est franchement impoli. (Un administrateur réseau considérera sans doute cela comme un acte carrément hostile ; un avocat bien informé pourra peut-être vous faire condamner pour tentative d'intrusion sur le système informatique ; la peine maximale est de 5 ans de prison.)

— Il reste à placer dans le fichier `/etc/resolv.conf` l'adresse IP d'un serveur qui sait traduire entre numéros IP et noms de machines. Nous verrons cela plus loin dans le cours ; en attendant, vous pouvez toujours essayer
```
echo nameserver 8.8.8.8 > /etc/resolv.conf
```

4.9 Le réseau Wifi

Pour un réseau Wifi, les étapes précédentes doivent être précédées d'une *association* entre l'ordinateur et la borne à laquelle on souhaite se raccorder. Une des manières de procéder est d'utiliser la commande `iwconfig`, par exemple avec

```
iwconfig ath0 essid any
```

Ensuite, on se retrouve dans une situation équivalente à celle où on est raccordé au même réseau éthernet que la borne.

Il y a de nombreux outils pour tester la présence de bornes wifi à proximité et tenter de passer à travers les étages d'encryption. La commande `kismet` est semble-t-il celle qui est à la mode en ce moment.

4.10 Le reste

Le protocole IP version 4 possède des options, peu utilisées, que le cours passe sous silence.

Avant la mise en place du CIDR, il y a eu un autre mécanisme mis en place pour compacter les tables de routage, fondé sur un dispositif supplémentaire nommé les *Autonomous Systems* (en français *systèmes autonomes*). Chaque réseau appartient à un système autonome et le routage par BGP se fait entre systèmes autonomes. Ordinairement, un système autonome correspond à un fournisseur d'accès à Internet. Les systèmes autonomes sont numérotés sur deux octets, il ne peut donc y en avoir qu'environ 65 000 au maximum.

Il y a un protocole symétrique du protocole ARP utilisé par les ordinateurs sans disque, donc sans fichier de configuration, pour déterminer quelle adresse IP ils devaient utiliser. Ce protocole s'appelle RARP comme *Reverse ARP* (en français *ARP inversé*). Il n'est quasiment plus plus utilisé à l'heure actuelle : il a été remplacé par le protocole DHCP.

4.11 Résumé

Chaque interface vers Internet dispose d'une adresse IPv4 différente. Sur Internet, les machines échangent les données dans des *datagrammes IP*, paquets de données d'environ 1Koctet maximum qui sont transmis indépendamment les uns des autres. Si le destinataire est un voisin, le paquet est envoyé directement à destination ; sinon chaque machine fait suivre le datagramme vers un routeur en fonction de sa table de routage.

4.12 Exercices

Exercice 4.1 — (facile) Parmi les adresses IP suivantes, lesquelles sont valides et lesquelles invalides

adresse	valide	invalide	pourquoi
1.1.1.1	o	o	
10.11.12.13.14	o	o	
123.321.321.213	o	o	
192.33.156.10	o	o	
222.222.222.222	o	o	
1.2.3.4.5	o	o	

Exercice 4.2 — Finir de remplir le tableau suivant

Adresse	Masque	Notation CDIR
10.20.30.40	255.255.128.0	10.20.30.40/17
193.33.156.10	255.255.255.0	
		193.35.156.10/26
10.0.0.1	255.255.192.0	
		192.168.0.1/22

Exercice 4.3 — Traduire le nom de machine `www.cs.univ-paris8.fr` avec la commande

```
$ host www.cs.univ-paris8.fr
www.cs.univ-paris8.fr is an alias for inferno.cs.univ-paris8.fr.
inferno.cs.univ-paris8.fr has address 193.54.153.250
```

(1) constater qu'on peut accéder indifféremment avec un browser web à
`http://www.cs.univ-paris8.fr`
et à
`http://193.54.153.250`
Tombe-t-on sur la même page ?
(2) traduire le numéro IP en notation décimale pointée en un entier avec $((193\times 256+54)\times 256+153)\times 256+250$ et tenter d'accéder à l'adresse `http://3241581050`. Tombe-t-on encore sur la même page ?

Exercice 4.4 — (l'explication viendra beaucoup plus tard) Même exercice avec la machine `www.univ-paris8.fr`.

Exercice 4.5 — En supposant un réseau Éthernet à 100 mégas bits par seconde, huit mille machines sur le réseau qui envoient chacune une requête ARP toute les 10 secondes, compter le nombre de bits d'une requête et d'une réponse ARP ; en déduire la fraction de la bande passante utilisée par ARP. (Se souvenir qu'une trame Éthernet est précédée de 64 bits pour signaler l'occupation du réseau.)

Exercice 4.6 — (difficile et inutile) Est-ce qu'un ordinateur doit jeter un datagramme quand il le reçoit avec un ttl à 0, ou bien ne pas l'émettre si le ttl vaut 0 après l'avoir décrémenté ? (plus facile et plus utile) Qu'est-ce qui change dans le comportement du réseau suivant qu'on fait l'un ou l'autre ?

Exercice 4.7 — (pour faire la différence entre débit et temps de réponse)

Imaginer un réseau constitué d'un semi-remorque rempli de DVD qui fait l'aller et retour entre Paris et Marseille. Le temps de réponse sera de l'ordre de 24 heures (le temps pour le camion de faire l'aller-retour, avec quelques pauses). Évaluer la bande passante de ce réseau (en bits par seconde).

Exercice 4.8 — Quelle est le numéro des premières RFC qui décrivent le CIDR ? De quand sont-elles datées ?

Exercice 4.9 — Dans CIDR, quelles adresses correspondent à quels continents ?

Exercice 4.10 — (pratique, facile) Copier la sortie de la commande `ifconfig -a` sur votre machine.

Exercice 4.11 — (pratique, facile) Quelle est la MAC address de votre liaison filaire ? Quelle est son MTU ?

Exercice 4.12 — (pratique) Constater que le `rtt` est identique quand on fait un ping sur l'interface locale `lo0` et sur l'adresse IP de l'interface Éthernet `eth0`. Quel est-il ?

Exercice 4.13 — (pratique) Constater qu'on peut utiliser n'importe quelle adresse IP valide qui commence par 127 pour nommer l'interface locale.

Exercice 4.14 — Combien de temps une entrée reste-t-elle dans la table ARP ? (Ça pourrait être une bonne idée d'utiliser la commande `tcpdump` pour trouver la réponse facilement. N'oubliez pas de mentionner la façon dont vous avez obtenu la réponse.)

Exercice 4.15 — (A faire) Lancer `tcpdump` pour écouter les trames ARP et tenter d'accéder à une machine locale qui n'existe pas (par exemple avec `ping`). Constater ce qui se passe, avec le moment où les choses se produisent. Expliquer alors pourquoi le `ping` vers une machine qui ne répond pas est resté silencieux pendant trois secondes puis a donné trois messages d'erreurs d'un coup.

Exercice 4.16 — (A faire) Si on lance plusieurs commandes `ping` vers la même machine locale inexistante (dans des fenêtres différentes), chacune va tenter d'envoyer une interrogation par seconde. En écoutant les messages ARP, on va savoir si la traduction entre numéro IP et MAC address est faite dans la commande (dans ce cas, il y aura autant de requêtes ARP par seconde que de commandes `ping` lancées en même temps) ou bien dans la pile TCP/IP (dans ce cas, le nombre de requêtes ARP ne changera pas). Quelle est la réponse ?