4TAH506U Réseau

Samuel Thibault

Contact

— `samuel.thibault@labri.fr`
— `http://dept-info.labri.fr/~thibault/enseignements#Reseaulpro`

Références

(voir site web ci-dessus pour les liens)
— Disponible sur l'ENT : Le réseau Internet, Des services aux infrastructures
`https://univ-scholarvox-com.docelec.u-bordeaux.fr/reader/docid/88800772`
— *Réseaux*, Andrew Tanenbaum (encyclopédique)
— *Internetworking with TCP/IP Volume One*, Douglas Comer (beaucoup plus accessible)
— *Qu'est-ce qu'Internet*, 3 conférences de Benjamin Bayart.
— *What happens when you type google.com into your browser and press enter ?*

Plan du cours

— Parcours rapide de la pile
  — Introduction, piles de protocoles, principe d'encapsulation, calculs
— Parcours de la pile
  — Couche physique, addresse IP, routage
  — Principe de protocole, exemples d'en-têtes, fonctionnement de TCP
  — HTTP, SMTP, DNS
  — Programmation UDP/TCP, fonctionnement des buffers
— Approfondissement
  — Utilisation de select, options, getaddrinfo
  — Distribution des adresses IP, Firewalls, masquerading
  — Protocoles centralisés/décentralisés/acentrés
  — Technologies web
  — Couche présentation : chiffrement, encodage, heure, accessibilité
  — Flexibilité L2
  — Diffusion à grande échelle
  — Structure d'Internet à grande échelle

Pile OSI

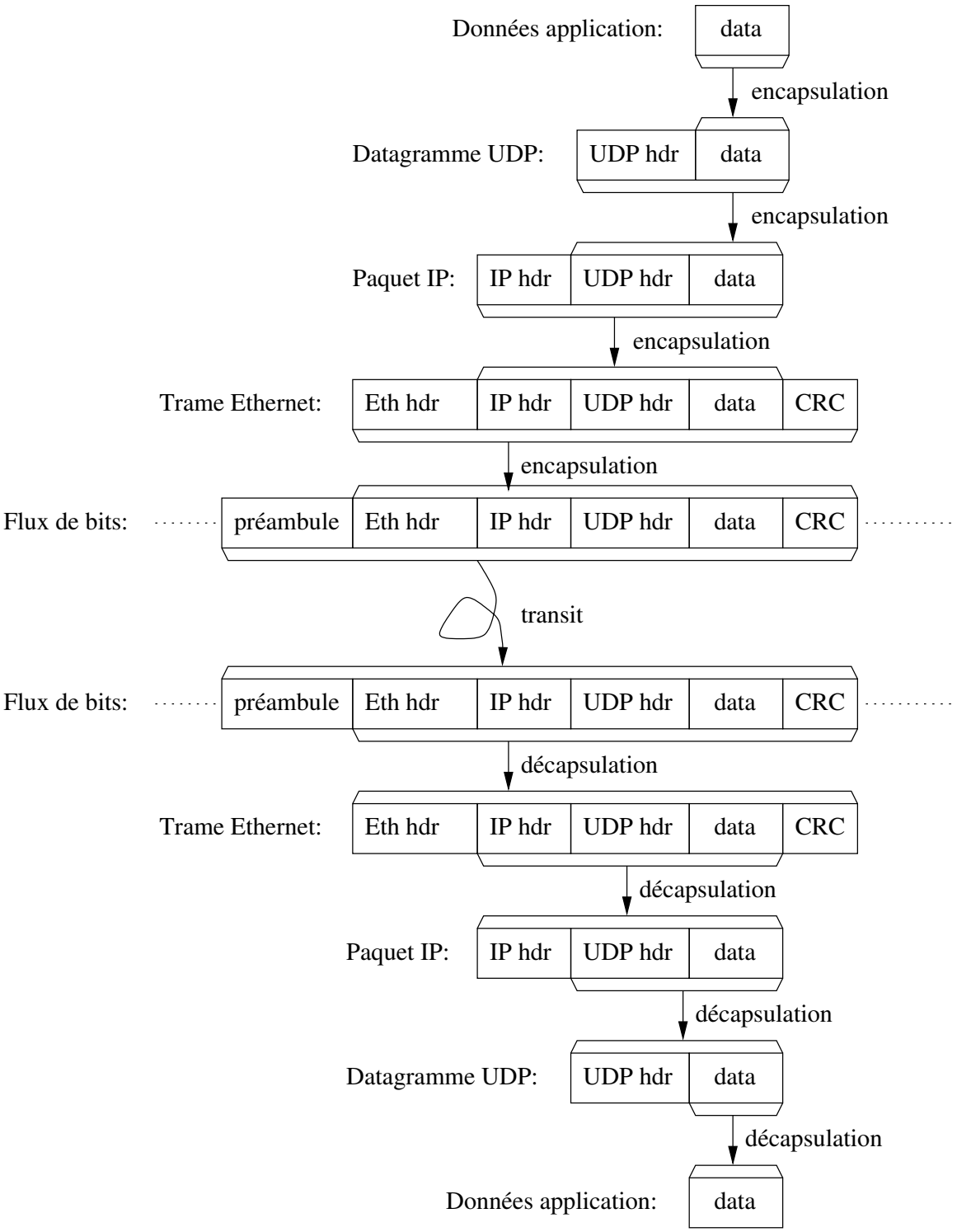
7. Application
6. Présentation
5. Session
4. Transport
3. Réseau
2. Liaison
1. Physique

Pile TCP/IP

<table>
<tr><td>7. Application</td><td rowspan="3">HTTP</td><td rowspan="3">SMTP</td><td>DNS</td><td rowspan="3">NTP</td><td>ping</td></tr>
<tr><td>6. Présentation</td><td rowspan="2"></td><td rowspan="2"></td></tr>
<tr><td>5. Session</td></tr>
<tr><td>4. Transport</td><td colspan="2">TCP</td><td colspan="2">UDP</td><td>ICMP</td></tr>
<tr><td>3. Réseau</td><td colspan="5">IP</td></tr>
<tr><td>2. Liaison</td><td colspan="2">Ethernet</td><td colspan="2">Wifi</td><td>...</td></tr>
<tr><td>1. Physique</td><td colspan="2">Câble
Ethernet</td><td colspan="2">Ondes
radio</td><td>...</td></tr>
</table>

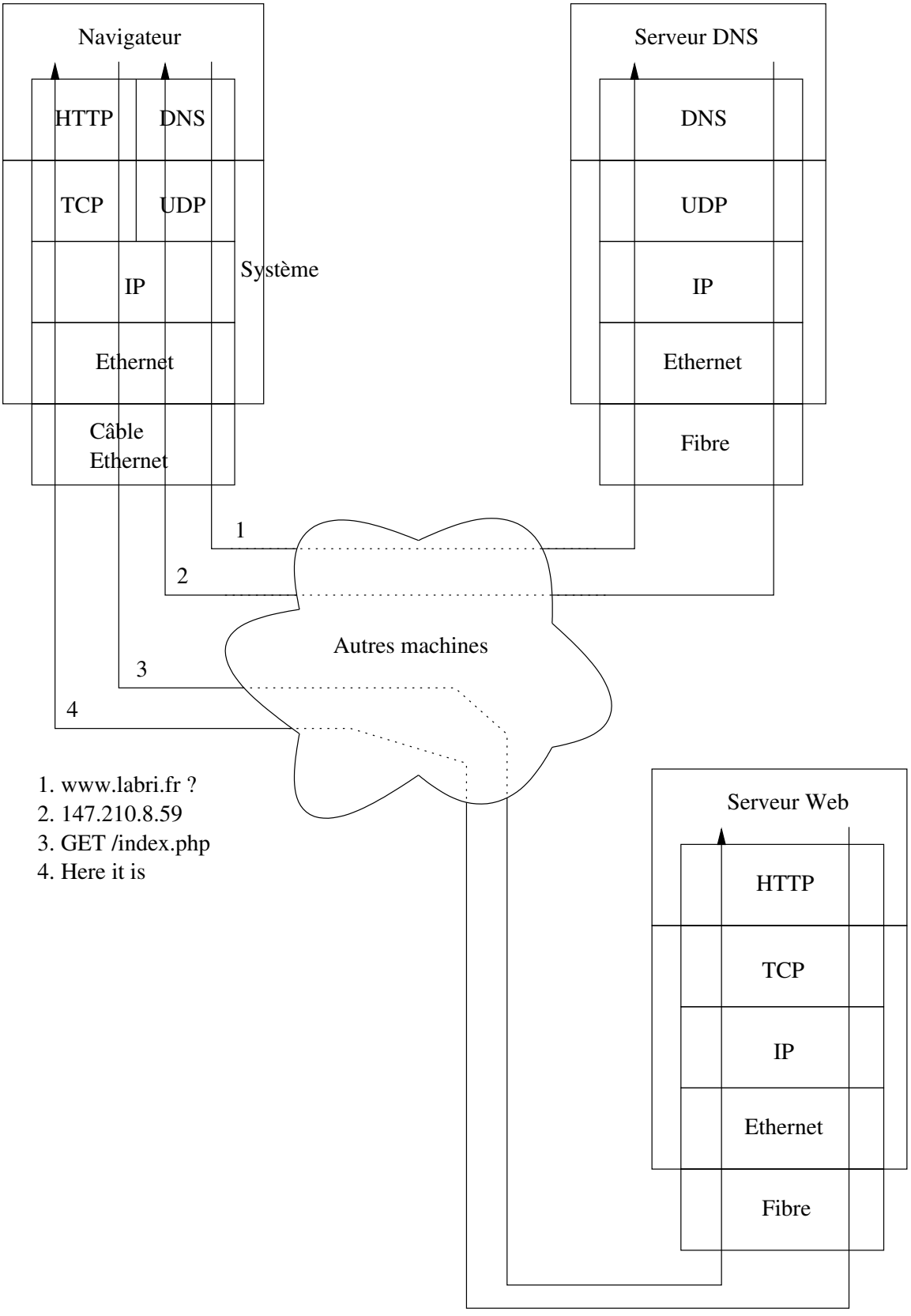
Encapsulation

$hdr = header =$ en-tête

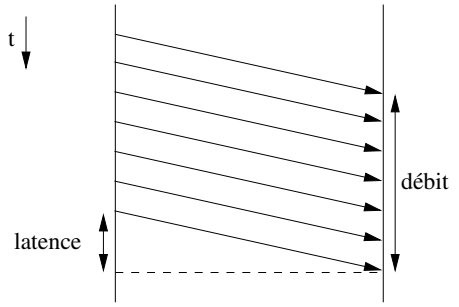


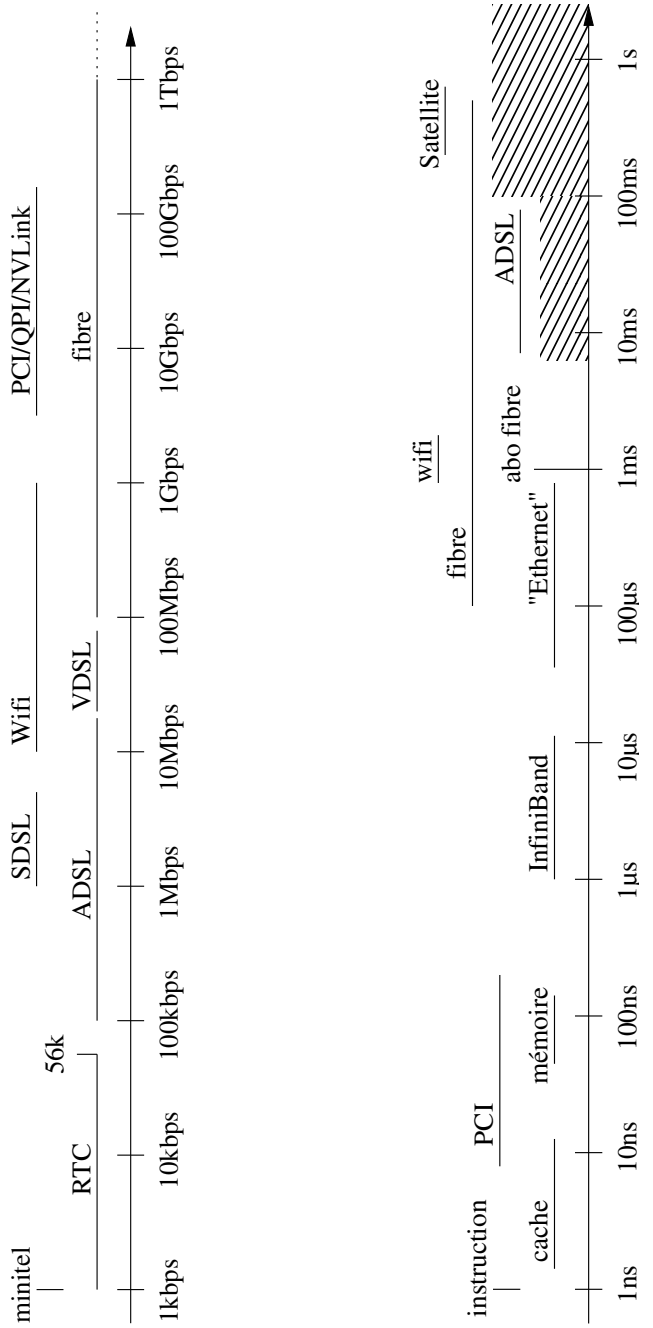
Pile TCP/IP en action

`http://www.labri.fr/index.php`



Débit / latence





Adresses

Adresses MAC : 48 bits en hexadécimal, e.g. : `00:21:70:b4:36:49`

Adresses IPv4 : 32 bits en décimal, e.g. : `193.50.110.76`
Adresses IPv6 : 128 bits en hexadécimal, e.g. : `2001:660:6101:800:252::4`
Plage d'adresses IPv4 : `u.v.w.x/n` avec `n` le nombre de bits de la partie réseau. e.g.

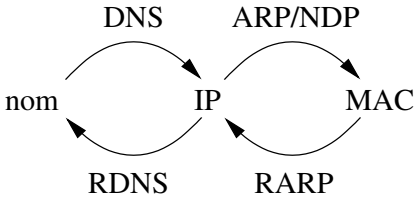
$$\texttt{10.0.0.0/9} \equiv \texttt{10.0.0.0} \text{ avec masque réseau } \texttt{255.128.0.0}$$
$$\equiv \texttt{10.0.0.0} - \texttt{10.127.255.255}.$$

La plage a donc 2^{32-n} adresses.
De même en IPv6, avec 2^{128-n} adresses.
Utiliser `ipcalc` pour calculer les réseaux.

Adresses privées : `10.0.0.0/8`, `172.16.0.0/12`, `192.168.0.0/16`, `fc00::/7`,
Adresses locales : `127.0.0.0/8`, `::1/128`
Adresses automatiques locales au lien : `169.254.0.0/16`, `fe80::/64`,

Port UDP/TCP : 16 bits en décimal, e.g. : `80`, voir `/etc/services` pour une liste.
Les ports 0 à 1023 sont réservés pour l'administrateur de la machine.

Traduction :



Configuration et Fichiers Linux

Configuration IP

```
$ /sbin/ifconfig
eth0      flags=5187<UP,BROADCAST,RUNNING,MASTER,MULTICAST>  mtu 9000
          inet 10.0.230.9  netmask 255.255.255.0  broadcast 10.0.230.255
   inet6 fe80::16fe:b5ff:fecb:8b61  prefixlen 64  scopeid 0x20<link>
   inet6 2001:660:6101:800:230::9  prefixlen 80  scopeid 0x0<global>
   ether 14:fe:b5:cb:8b:61  txqueuelen 1000  (Ethernet)
   RX packets 57757239  bytes 46658975385 (43.4 GiB)
   RX errors 0  dropped 25  overruns 0  frame 0
   TX packets 63931769  bytes 26268002555 (24.4 GiB)
   TX errors 0  dropped 3 overruns 0  carrier 0  collisions 0

$ ip addr ls
10: eth0: <BROADCAST,MULTICAST,MASTER,UP,LOWER_UP> mtu 9000 qdisc noqueue state UP group default qle
    link/ether 14:fe:b5:cb:8b:61 brd ff:ff:ff:ff:ff:ff
    inet 10.0.230.9/24 brd 10.0.230.255 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 2001:660:6101:800:230::9/80 scope global
       valid_lft forever preferred_lft forever
    inet6 fe80::16fe:b5ff:fecb:8b61/64 scope link
       valid_lft forever preferred_lft forever
```

Cache de résolution IP/MAC

```
$ /usr/sbin/arp -n
Adresse                  TypeMap AdresseMat          Indicateurs           Iface
10.0.230.5               ether   00:26:b9:75:d0:55   C                     eth0
10.0.230.254             ether   58:20:b1:b1:23:00   C                     eth0

$ ip neigh ls
10.0.230.5 dev eth0 lladdr 00:26:b9:75:d0:55 DELAY
10.0.230.254 dev eth0 lladdr 58:20:b1:b1:23:00 REACHABLE
```

Configuration du routage

```
$ /sbin/route -n
Table de routage IP du noyau
Destination     Passerelle      Genmask         Indic Metric Ref    Use Iface
0.0.0.0         10.0.230.254    0.0.0.0         UG    0      0        0 eth0
10.0.230.0      0.0.0.0         255.255.255.0   U     0      0        0 eth0

$ ip route ls
default via 10.0.230.254 dev eth0 onlink
10.0.230.0/24 dev eth0 proto kernel scope link src 10.0.230.9
```

Configuration DNS

```
$ cat /etc/resolv.conf
nameserver 10.0.220.13
nameserver 10.0.220.14
search emi.u-bordeaux.fr
```

Les différents numéros de protocoles référencés dans un en-tête IP.

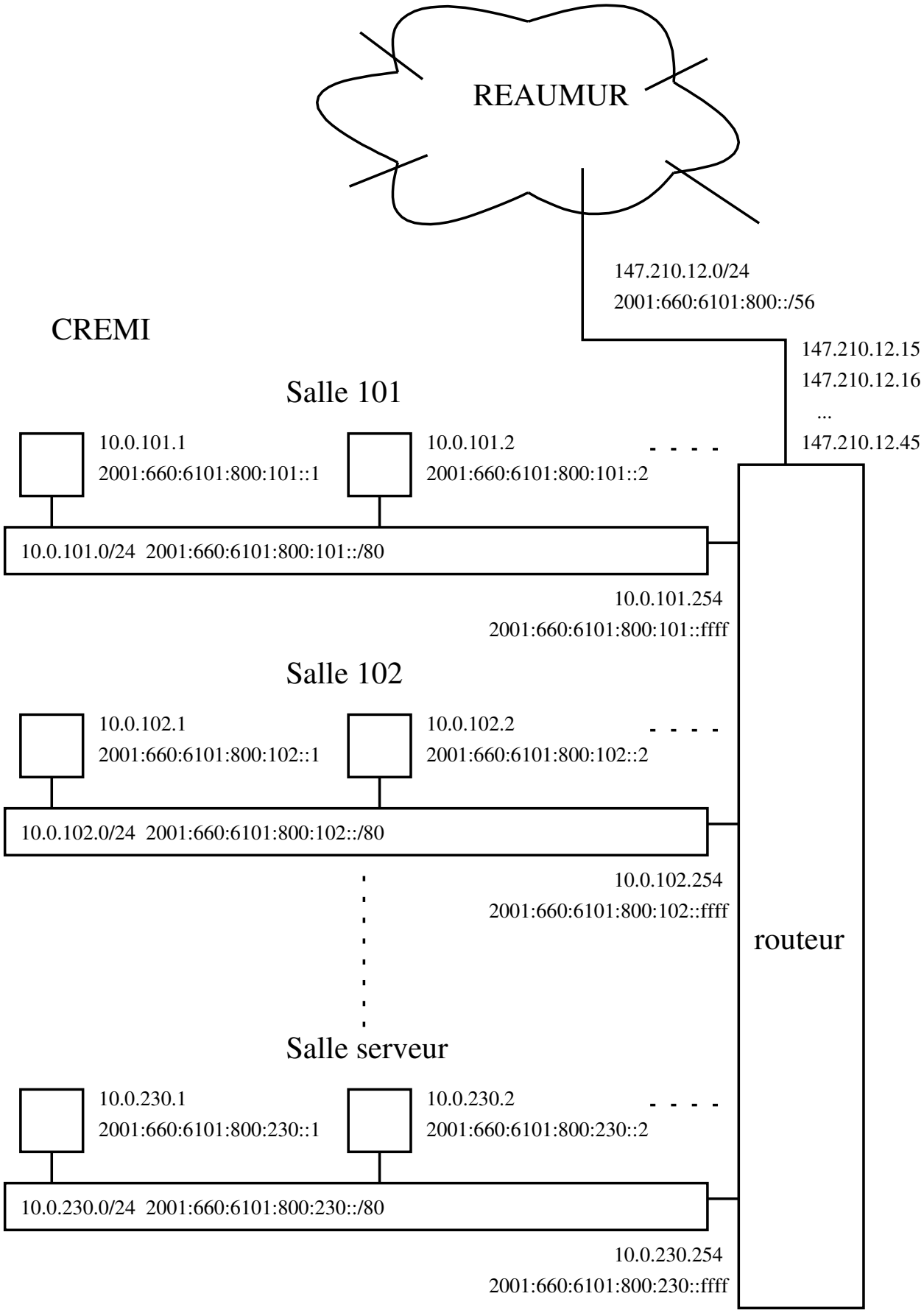
```
$ cat /etc/protocols
...
icmp 1 ICMP
tcp 6 TCP
udp 17 UDP
...
```

Les différents numéros de ports utilisés par UDP et TCP.

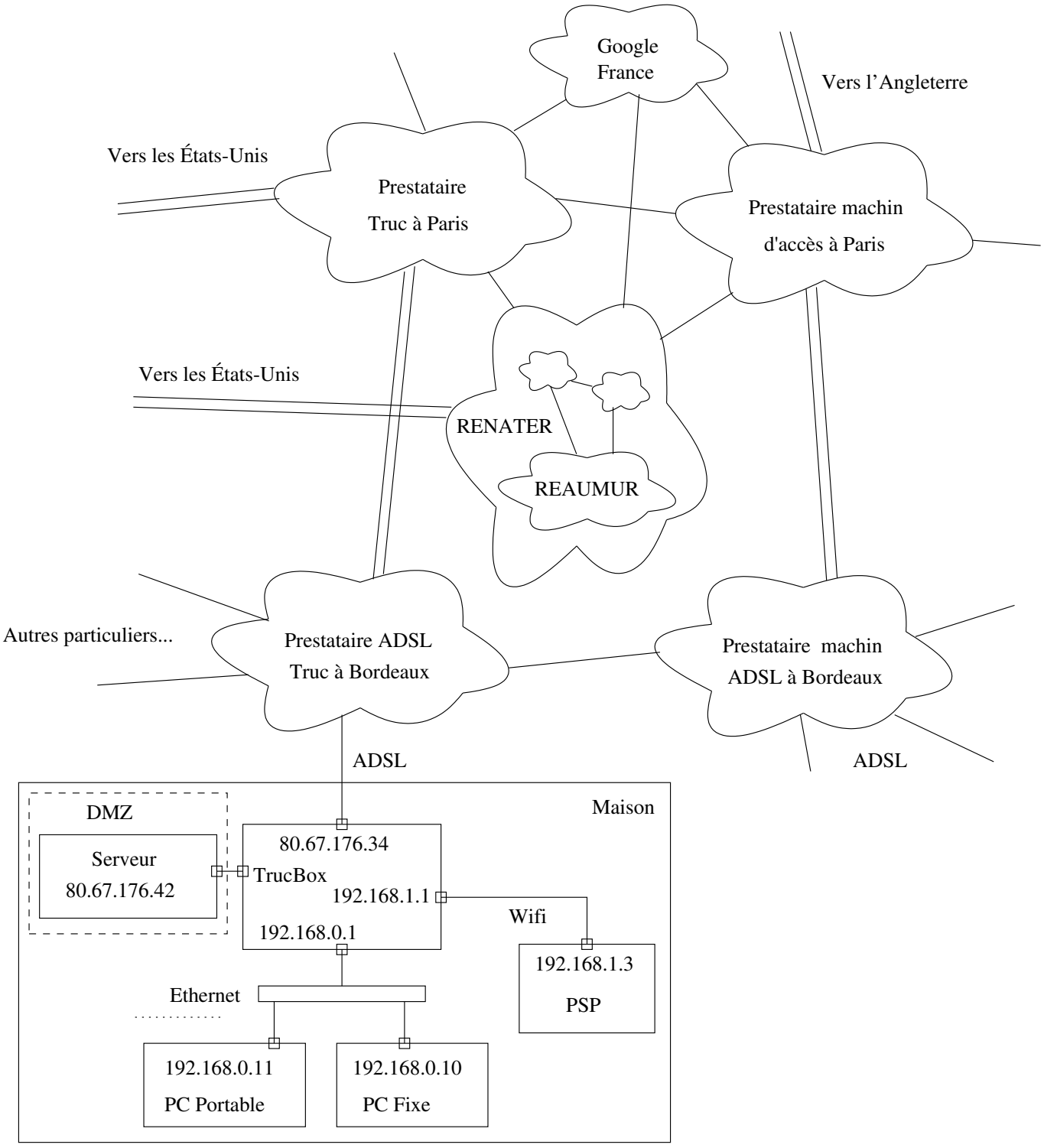
```
$ cat /etc/services
...
echo  7/udp
echo  7/tcp
ssh  22/tcp
telnet  23/tcp
smtp  25/tcp
domain  53/udp
http  80/tcp
...
```

Plan partiel de Renater et l'Université de Bordeaux I

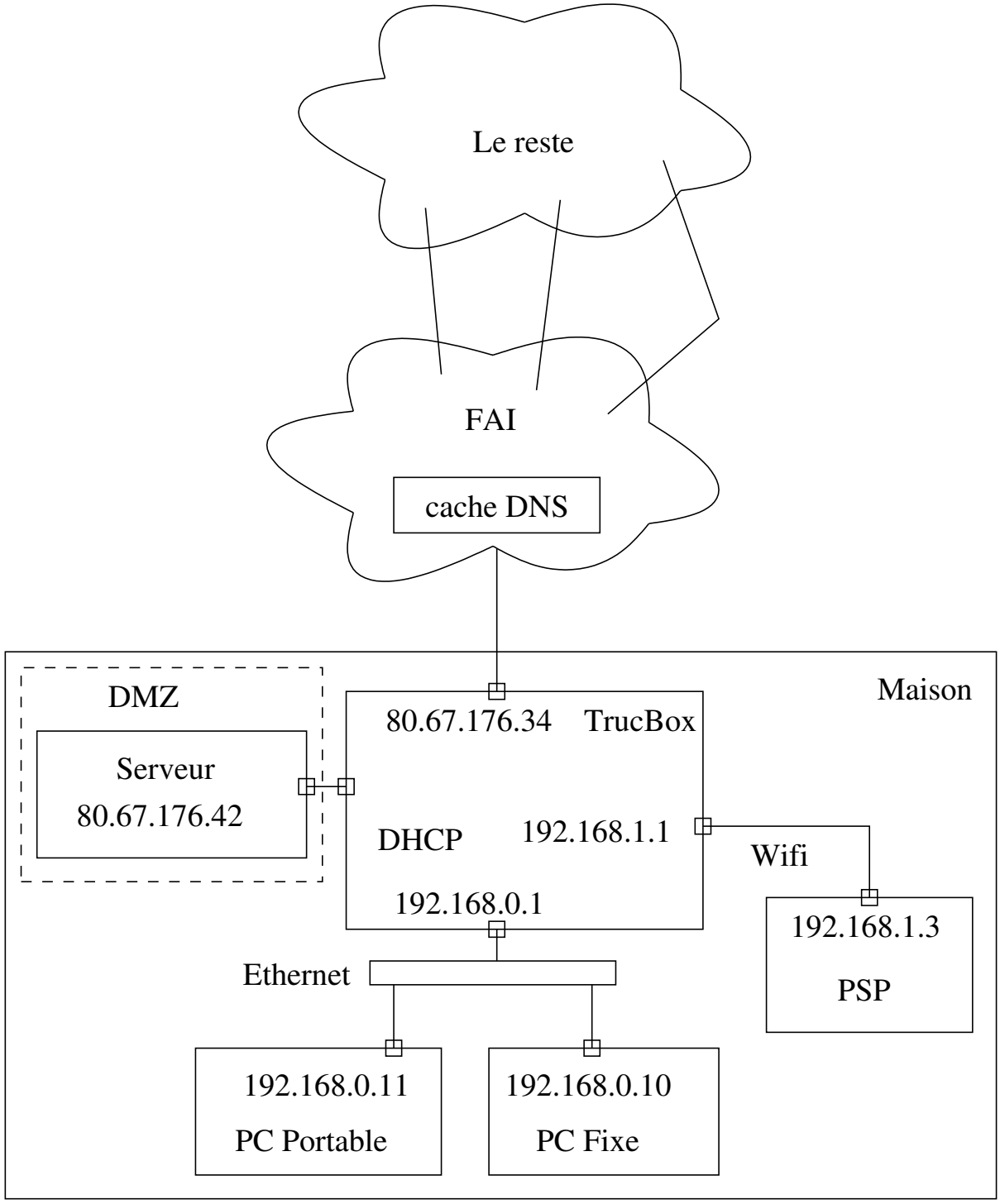
Zoom sur le réseau du CREMI



Plan typique « Maison » et prestataires Internet

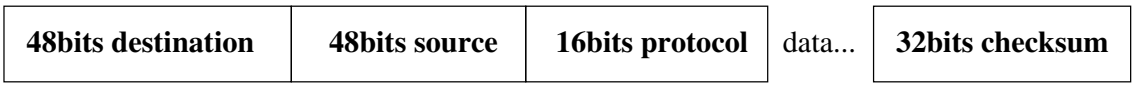


Adresses / Firewall / Masquerading / NAT



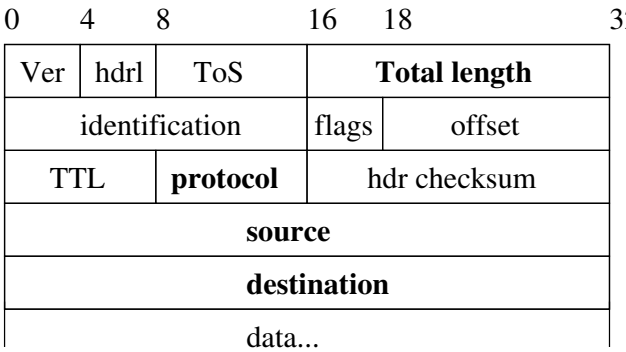
Exemples d'en-têtes

Ethernet, 14 octets d'en-tête et 2 octets de fin

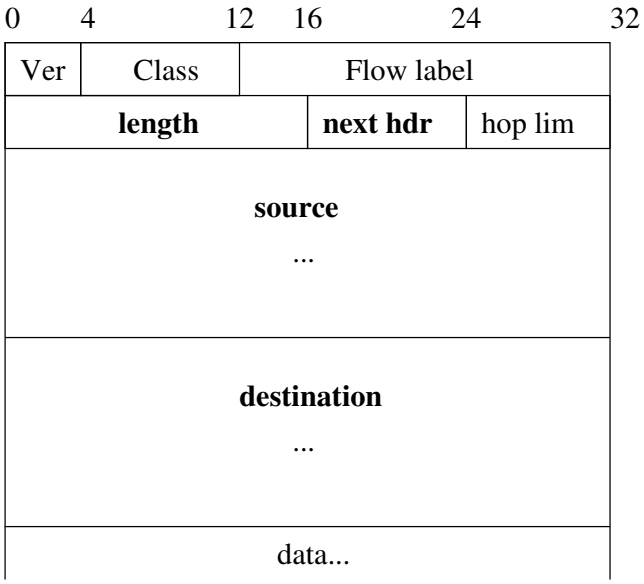


IP

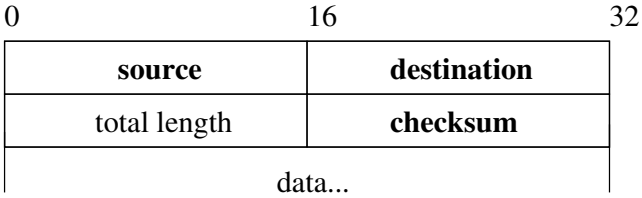
Version 4, 20 octets d'en-tête



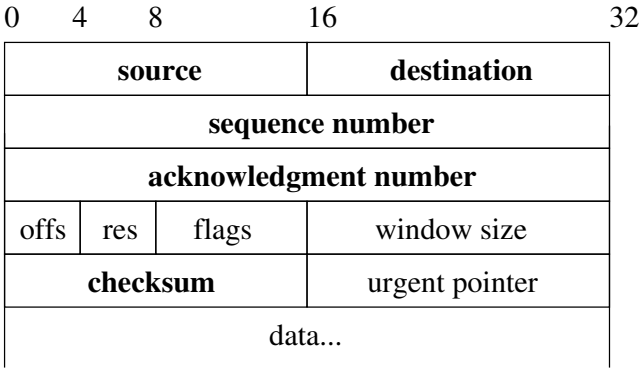
Version 6, 40 octets d'en-tête



UDP, 8 octets d'en-têtes

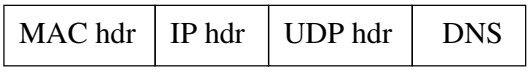


TCP, 20 octets d'en-tête



Correspondance données brutes / en-têtes

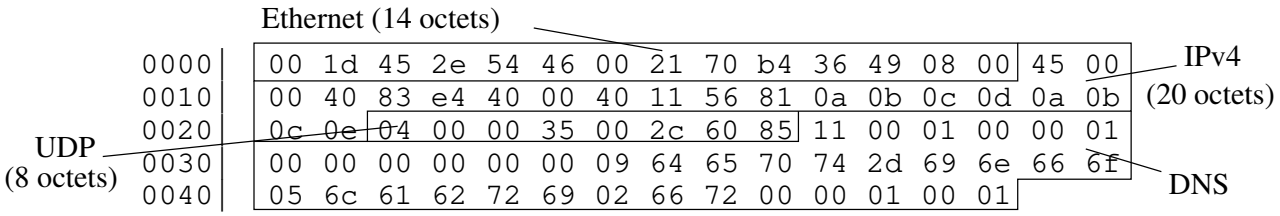
Exemple de trame Ethernet contenant un paquet IPv4, contenant un datagrame UDP, contenant une requête DNS, c'est-à-dire l'encapsulation :



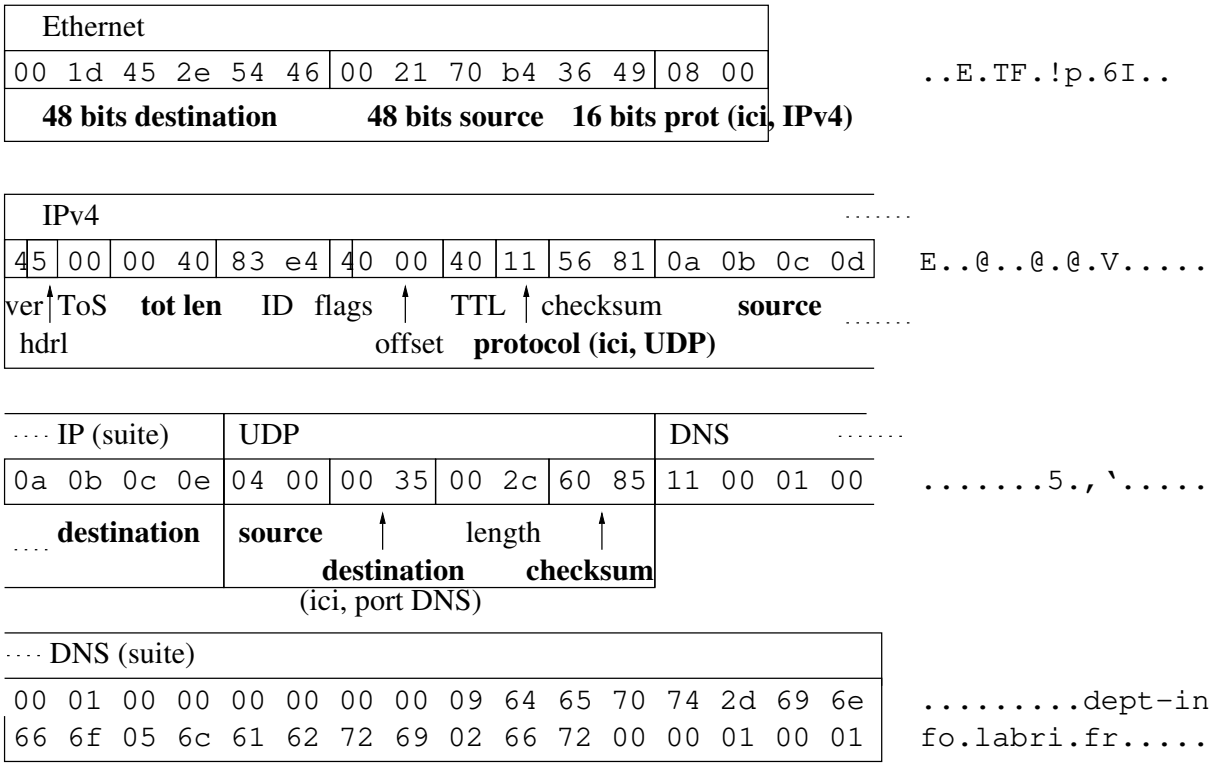
Données ethernet brutes (ce qui passe sur le câble)

```
pos      hexadécimal                                             ASCII
0x0000|  00 1d 45 2e 54 46 00 21 70 b4 36 49 08 00 45 00  ..E.TF.!p.6I..E.
0x0010|  00 40 83 e4 40 00 40 11 56 81 0a 0b 0c 0d 0a 0b  .@..@.@.V.......
0x0020|  0c 0e 04 00 00 35 00 2c 60 85 11 00 01 00 00 01  .....5.,`.......
0x0030|  00 00 00 00 00 00 09 64 65 70 74 2d 69 6e 66 6f  .......dept-info
0x0040|  05 6c 61 62 72 69 02 66 72 00 00 01 00 01        .labri.fr.....
```

Découpage des données en fonction de la taille des en-têtes

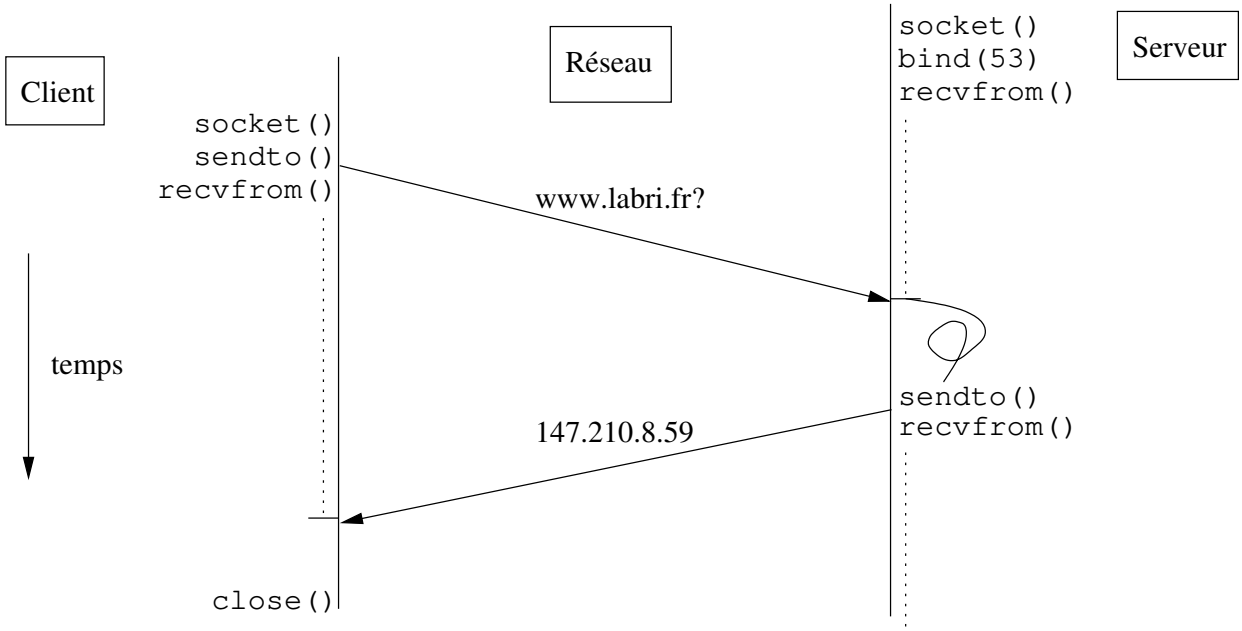


Analyse à l'intérieur de chaque en-tête

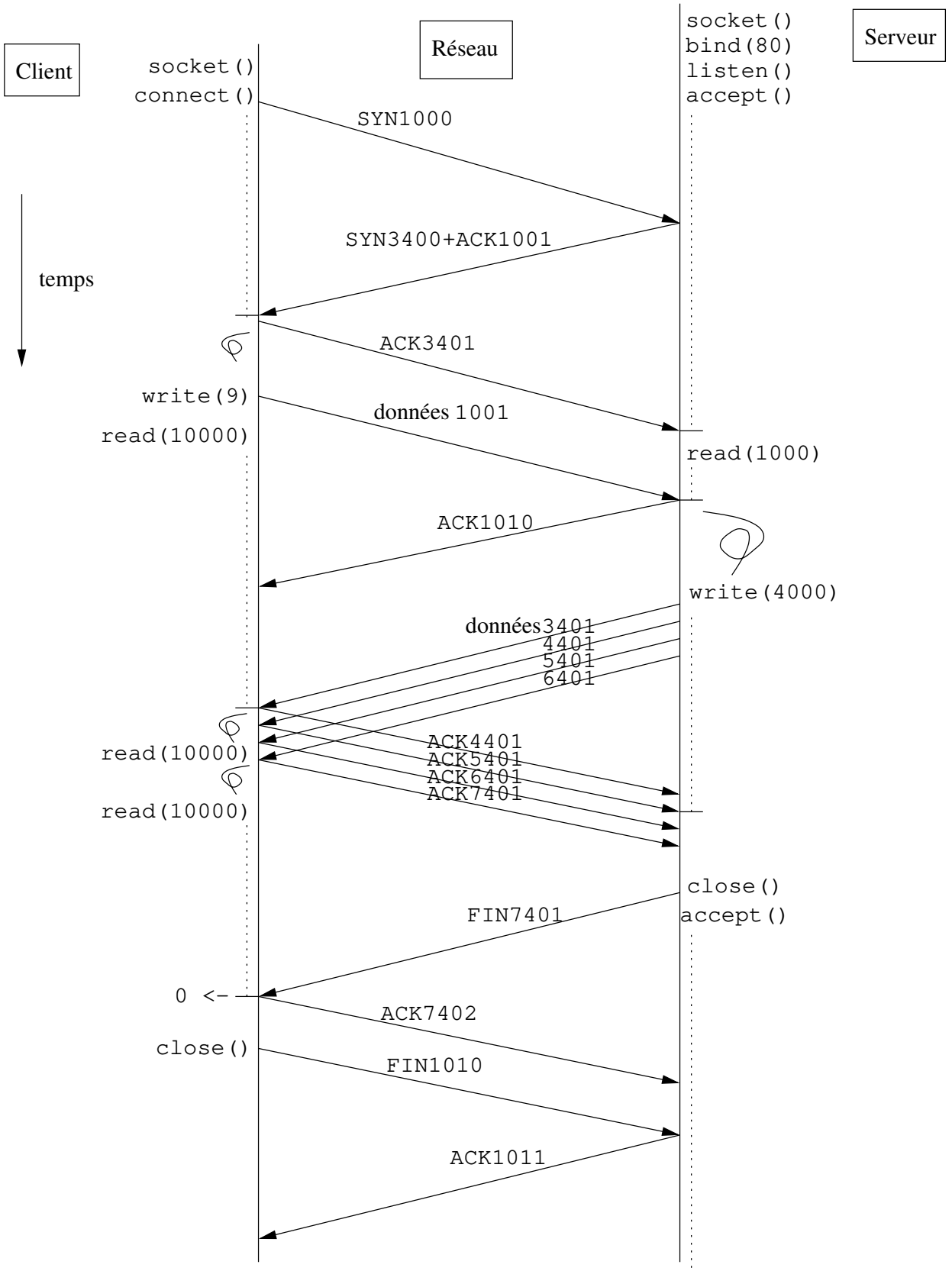


Exercice pour le lecteur : trouver le format de l'en-tête DNS sur Internet, et trouver le numéro du type de requête DNS utilisé ici.

Protocole DNS sur UDP (donc sans connexion)



Protocole TCP (donc avec connexion)



Protocole HTTP 1.x

http://www.google.fr/index.html

```
GET /index.html HTTP/1.0
User-Agent: w3m/0.5.2
Accept: text/*, image/*, audio/*, application/*, message/*, video/*
Accept-Language: fr-fr,en
Accept-Encoding: gzip, compress
Host: www.google.fr


HTTP/1.1 200 OK
Server: Apache
Date: Tue, 29 Sep 2009 15:55:34 GMT
Last-Modified: Fri, 04 Sep 2009 09:02:59 GMT
Content-Type: text/html; charset=utf-8
Content-Language: fr

<html><head><title>
...
```

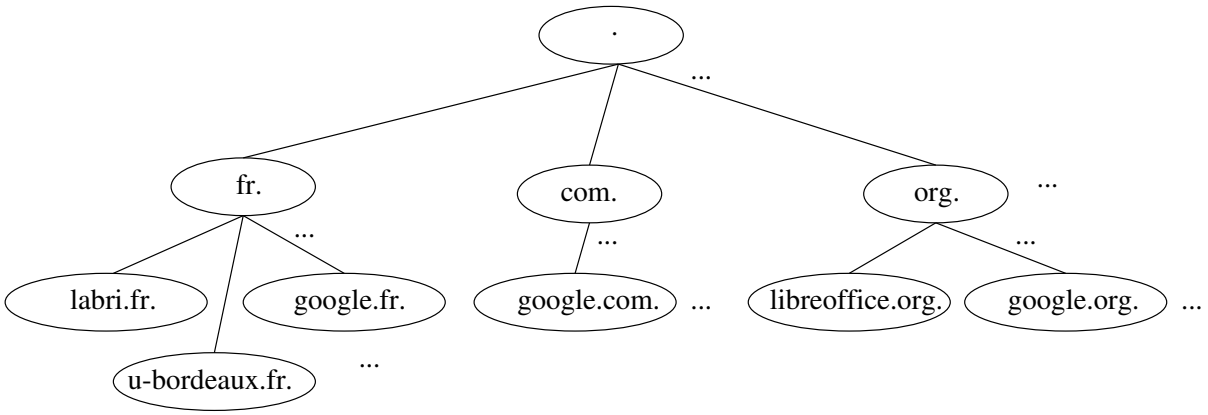
Protocole SMTP

mailto:rms@gnu.org

```
220 smtp.fdn.fr ESMTP Postfix (Debian/GNU)
helo mon.chezmoi.com
250 smtp.fdn.fr
mail from:<samuel.thibault@labri.fr>
250 2.1.0 Ok
rcpt to:<rms@gnu.org>
250 2.1.5 Ok
data
354 End data with <CR><LF>.<CR><LF>
Hi!
.
250 2.0.0 Ok: queued as 4F94C116E9A
quit
221 2.0.0 Bye
```

Hiérarchie DNS

```
$ dig any +trace www.labri.fr
; <<>> DiG 9.9.5-12-Debian <<>> +trace www.labri.fr
;; global options: +cmd
.   500493 IN NS a.root-servers.net.
.   500493 IN NS b.root-servers.net.
.   500493 IN NS c.root-servers.net.
...
.   500493 IN NS m.root-servers.net.
;; Received 913 bytes from 193.50.111.150#53(193.50.111.150) in 154 ms

fr.   172800 IN NS d.ext.nic.fr.
fr.   172800 IN NS d.nic.fr.
fr.   172800 IN NS e.ext.nic.fr.
fr.   172800 IN NS f.ext.nic.fr.
fr.   172800 IN NS g.ext.nic.fr.
;; Received 604 bytes from 199.7.91.13#53(d.root-servers.net) in 198 ms

labri.fr.  172800 IN NS ns.univ-bordeaux.fr.
labri.fr.  172800 IN NS edwood.emi.u-bordeaux1.fr.
labri.fr.  172800 IN NS donaser.labri.fr.
labri.fr.  172800 IN NS neouvielle.enseirb-matmeca.fr.
;; Received 695 bytes from 194.0.9.1#53(d.nic.fr) in 26 ms

www.labri.fr.  28800 IN CNAME www3.labri.fr.
www3.labri.fr.  28800 IN A 147.210.8.59
www3.labri.fr.  28800 IN AAAA 2001:660:6101:404::80
labri.fr.  28800 IN NS donaser.labri.fr.
labri.fr.  28800 IN NS ns.univ-bordeaux.fr.
labri.fr.  28800 IN NS neouvielle.enseirb-matmeca.fr.
labri.fr.  28800 IN NS edwood.emi.u-bordeaux1.fr.
;; Received 311 bytes from 147.210.8.187#53(donaser.labri.fr) in 3 ms
```



API Python TCP/UDP/IP

La documentation est disponible sur `https://docs.python.org/library/socket.html` et dans votre terminal `pydoc3 socket`

```
import socket
```

En UDP :

```
# Créer une socket
s = socket.socket(socket.AF_INET6, socket.SOCK_DGRAM, 0)

# Choisir l'adresse locale (notamment le port)
s.bind(('', 12345))

# Choisir l'adresse distante (IP et port), optionnel, pour utiliser
# send/write/recv/read plutôt que sendto/recvfrom
s.connect(('2a0c:e300::12', 7))

# Envoyer / recevoir
s.send(b'bonjour')
data = s.recv(1500)

# Envoyer un datagramme UDP à un service d'une machine donnée
s.sendto(b'bonjour', ('2a0c:e300::15', 9))

# Recevoir un datagramme UDP depuis n'importe quelle machine
data, addr = s.recvfrom(1500)

# Fermer la socket
s.close()
```

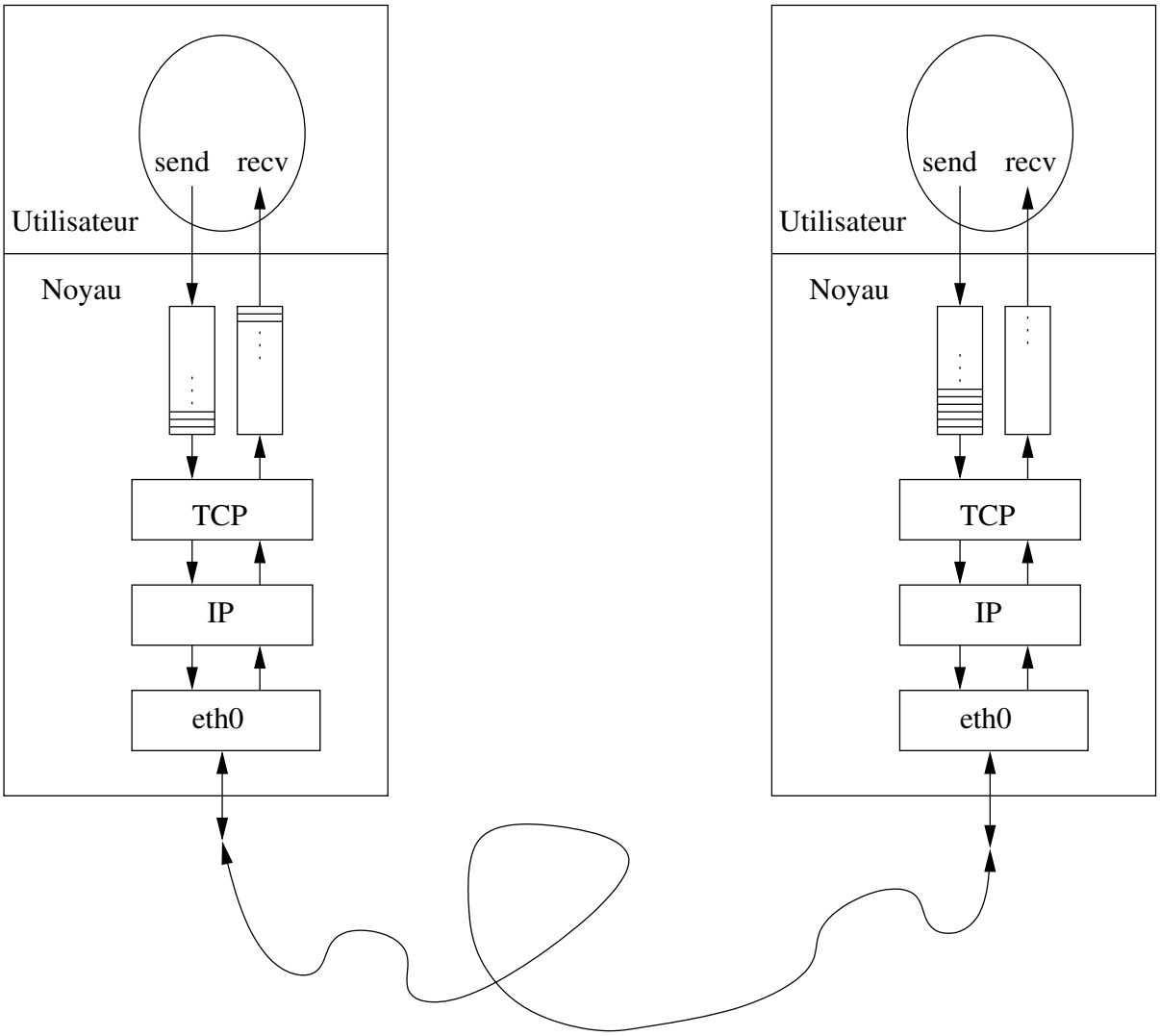
En TCP, côté client :

```
# Créer une socket
s = socket.socket(socket.AF_INET6, socket.SOCK_STREAM, 0)

# Se connecter à un service d'une machine donnée
s.connect(('2a0c:e300::12', 7))

# ... s.read() / s.write()
```

En TCP, côté serveur :

```
# Créer une socket
s = socket.socket(socket.AF_INET6, socket.SOCK_STREAM, 0)

# Attacher à un port local donné
s.bind(('', 12345))

# Passer en mode serveur
s.listen(1)

# Accepter une nouvelle connexion, s2 est la socket connectée au client
s2, addr = s.accept()

# ... s2.read() / s2.write()

s2.close();
```

Tampons noyau




```
$ netstat
Connexions Internet actives (sans serveurs)
Proto Recv-Q Send-Q Adresse locale          Adresse distante        Etat
tcp        0    672 193.50.110.253:50740    foo.bar.com:ssh         ESTABLISHED
```

Adresses

```
# Récupérer l'adresse locale
addr = s.getsockname()

# Récupérer l'adresse distante
addr = s.getpeername()
```

Options de socket

```
s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)

i = s.getsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR)

# Notamment:
#
# SOL_SOCKET, SO_REUSEADDR, voir man 7 socket
#
# IP_TOS, voir man 7 ip
#
# SOL_TCP, TCP_NODELAY, TCP_CORK, voir man 7 tcp
# voir aussi man 7 udp
```

Résolution de nom de domaine

```
l = socket.getaddrinfo("www.u-bordeaux.fr", "http", 0, socket.SOCK_STREAM,
    0, 0)
for choix in l:
  (family, type, proto, name, addr) = choix
  s = socket.socket(family, type, proto)
  # ...
  s.connect(addr)
```

ou inversement

```
(name, port) = socket.getnameinfo(('185.233.100.14',7), 0)
```

Select

https://docs.python.org/library/select.html
pydoc3 select

Permet d'attendre sur plusieurs sockets à la fois :

```
import select

readl, writel, errorl = select.select([s1,s2], [], [])
for s in readl:
  traite(s)
```

Résumé Python

```
import socket
```

Client UDP :

```
l = socket.getaddrinfo("neutral.aquilenet.fr", "echo", 0, socket.SOCK_DGRAM
    , 0, 0)
(family, type, proto, name, addr) = l[0]
s = socket.socket(family, type, proto)
s.sendto(b"abcd", addr)
r,addr = s.recvfrom(1024)
```

Serveur UDP :

```
s = socket.socket(socket.AF_INET6, socket.SOCK_DGRAM, 0)
# Pouvoir réutiliser le port:
s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
# Oui, il faut les doubles parenthèses, pour ne passer qu'un argument qui
    est un tuple
s.bind(('', 1234))
r,addr = s.recvfrom(1024)
s.sendto(r, addr)
```

Client TCP :

```
l = socket.getaddrinfo("neutral.aquilenet.fr", "echo", 0, socket.
    SOCK_STREAM, 0, 0)
(family, type, proto, name, addr) = l[0]
s = socket.socket(family, type, proto)
s.connect(addr)
s.send(b"abcd")
r = r.recv(1024)
```

Serveur TCP :

```
import socket
s = socket.socket(socket.AF_INET6, socket.SOCK_STREAM, 0)
# Pouvoir réutiliser le port:
s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
# Désactiver l'algorithme de Nagle:
s.setsockopt(socket.SOL_TCP, socket.TCP_NODELAY, 1)

s.bind(('', 1234))
s.listen(1)

while True:
  s2,addr = s.accept()
  while True:
    r = s2.recv(1024)
    if r == b'':
      break
    s2.send(r)
  s2.close()
```

API C IP

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

/* Adresse réseau */
struct sockaddr {
    /* ... */
    sa_family_t sa_family;
};

/* Adresse IPv4 */
struct in_addr {
    __be32 s_addr;
};

/* Adresse réseau IPv4 */
struct sockaddr_in {
    /* ... */
    sa_family_t    sin_family; /* AF_INET */
    struct in_addr sin_addr;
    __be16         sin_port;
};

struct in6_addr {
    /* ... */
    u8 s6_addr[16];
}

/* Adresse réseau IPv6 */
struct sockaddr_in6 {
    /* ... */
    sa_family_t     sin6_family; /* AF_INET6 */
    struct in6_addr sin6_addr;
    __be16          sin6_port;
};

/* Adresse réseau inconnue */
struct sockaddr_storage {
    /* ... */
    sa_family_t     ss_family;
};

/* Conversion vers/depuis ordre réseau */
uint16_t htons(uint16_t hostshort);
uint16_t ntohs(uint16_t netshort);
uint32_t htonl(uint32_t hostlong);
uint32_t ntohl(uint32_t netlong);

/* Conversion ascii/binaire */
int inet_aton(const char *ascii, struct in_addr *binaire);
char *inet_ntoa(struct in_addr binaire);

/* De même, mais portables v4/v6 */
int inet_pton(int af, const char *ascii, void *bin);
const char *inet_ntop(int af, const void *bin, char *ascii, socklen_t size)
```

API C UDP

```c
/* Créer une socket */
int socket(int domain, int type, int protocol);
/* e.g. fd = socket(AF_INET, SOCK_DGRAM, 0); */

/* Choisir l'adresse distante (IP et port), optionnel, pour utiliser
   send/write/recv/read plutôt que sendto/recvfrom */
int connect(int sockfd, const struct sockaddr *addr, socklen_t addrlen);

/* Choisir l'adresse locale (notamment le port) */
int bind(int sockfd, const struct sockaddr *addr, socklen_t addrlen);

/* Envoyer un datagramme UDP à un service d'une machine donnée */
ssize_t sendto(int sockfd, const void *buf, size_t len, int flags,
               const struct sockaddr *dest_addr, socklen_t addrlen);
/* Recevoir un datagramme UDP depuis n'importe quelle machine */
ssize_t recvfrom(int sockfd, void *buf, size_t len, int flags,
                 struct sockaddr *src_addr, socklen_t *addrlen);

/* Fermer la socket */
int close(int fd);
```

API C TCP

```c
/* Créer une socket */
int socket(int domain, int type, int protocol);
/* e.g. fd = socket(AF_INET, SOCK_STREAM, 0); */

/*** Partie client ***/

  /* Se connecter à un service d'une machine donnée */
  int connect(int sockfd, const struct sockaddr *addr, socklen_t addrlen);

/*** Partie serveur ***/

  /* Attacher à un port local donné */
  int bind(int sockfd, const struct sockaddr *addr, socklen_t addrlen);
  /* Passer en mode serveur */
  int listen(int sockfd, int backlog);
  /* Accepter une nouvelle connexion, donc le fd est retourné */
  int accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen);

/* Fermer la socket */
int close(int fd);

/* Récupérer l'adresse locale */
int getsockname(int sockfd, struct sockaddr *addr, socklen_t *addrlen);
/* Récupérer l'adresse distante */
int getpeername(int sockfd, struct sockaddr *addr, socklen_t *addrlen);
```

Options de socket

```
int setsockopt(int sockfd, int level, int optname, const void *optval,
    socklen_t optlen);

int getsockopt(int sockfd, int level, int optname, void *optval, socklen_t
    *optlen);

/* SOL_TCP, TCP_NODELAY, TCP_CORK, voir man 7 tcp
 * voir aussi man 7 udp
 * voir aussi man 7 ip
 * SOL_SOCKET, SO_REUSEADDR, voir man 7 socket */
```

select

```
/* voir man 2 select */

void FD_ZERO(fd_set *set);
void FD_CLR(int fd, fd_set *set);
void FD_SET(int fd, fd_set *set);
int  FD_ISSET(int fd, fd_set *set);

int select(int nfds, fd_set *readfds, fd_set *writefds, fd_set *exceptfds,
           struct timeval *timeout);

/* Attend des données lisibles sur fd1 ou fd2, traite en conséquence */
int attend(int fd1, int fd2) {
    fd_set set;
    int max;
    FD_ZERO(&set);
    FD_SET(fd1, &set);
    FD_SET(fd2, &set);
    max = MAX(fd1, fd2);

    if (select(max+1, &set, NULL, NULL, NULL) == -1)
        return -1;

    if (FD_ISSET(fd1, &set))
        traite(fd1);
    if (FD_ISSET(fd2, &set))
        traite(fd2);

    return 0;
}

int poll(struct pollfd *fds, nfds_t nfds, int timeout);
```

getaddrinfo

```c
/* Client */

const char *host = "jaguar.emi.u-bordeaux1.fr";
const char *port = "ssh";

int sockfd;
struct addrinfo *res,*cur;
struct addrinfo hints;

/* Type de socket voulue */
memset(&hints, 0, sizeof(hints));
hints.ai_socktype = SOCK_STREAM;

/* Requête DNS */
gaierrno = getaddrinfo(host, port, &hints, &res);
if (gaierrno) {
  fprintf(stderr,"getaddrinfo: %s\n", gai_strerror(gaierrno));
  return -1;
}

/* Parcours des résultats */
for(cur = res; cur; cur = cur->ai_next) {

  /* Essayer d'établir la socket */
  sockfd = socket(cur->ai_family, cur->ai_socktype, cur->ai_protocol);
  if (sockfd < 0) {
    if (errno != EAFNOSUPPORT) /* Éviter de râler si ipv6 n'est pas
    disponible */
      perror("socket");
    continue;
  }

  /* Essayer de se connecter */
  if (connect(sockfd, cur->ai_addr, cur->ai_addrlen) < 0) {
    perror("connect");
    close(sockfd);
    continue;
  }

  /* OK, on est connecté! */
  break;
}

freeaddrinfo(res);

if (!cur) {
  fprintf(stderr, "could not connect\n");
  return -1;
}
```

```c
/* Serveur */
const char *host = "";
const char *port = "http";

int sockfd;
struct addrinfo *res,*cur;
struct addrinfo hints;

/* Type de socket voulue */
memset(&hints, 0, sizeof(hints));
hints.ai_flags = AI_PASSIVE;
hints.ai_socktype = SOCK_STREAM;

/* Requête DNS */
gaierrno = getaddrinfo(host, port, &hints, &res);
if (gaierrno) {
  fprintf(stderr,"getaddrinfo: %s\n", gai_strerror(gaierrno));
  return -1;
}

/* Parcours des résultats */
for(cur = res; cur; cur = cur->ai_next) {

  /* Essayer d'établir une socket de ce type */
  sockfd = socket(cur->ai_family, cur->ai_socktype, cur->ai_protocol);
  if (sockfd < 0) {
    if (errno != EAFNOSUPPORT) /* Éviter de râler si ipv6 n'est pas
     disponible */
      perror("socket");
    continue;
  }

  /* Essayer d'établir une socket d'écoute pour cette adresse */
  if (bind(sockfd, ai->ai_addr, ai->ai_addrlen) < 0) {
    perror("bind");
    close(sockfd);
    continue;
  }

  /* OK, ajouter aux autres sockets d'écoute de l'application, et
   * continuer avec les autres adresses */

  ... sockfd ...
}

freeaddrinfo(res);
```

Voir aussi la réciproque de `getaddrinfo` :

```c
int getnameinfo(const struct sockaddr *sa, socklen_t salen,
                char *host, size_t hostlen,
                char *serv, size_t servlen, int flags);
```

API Java

Client UDP :

```
DatagramSocket s = new DatagramSocket();
InetAddress address = InetAddress.getByName("neutral.aquilenet.fr");
byte[] message = "abcd".getBytes();
DatagramPacket p = new DatagramPacket(message, message.length, address, 13)
    ;
s.send(p);
```

Serveur UDP :

```
DatagramSocket s = new DatagramSocket(1234);
byte[] buffer = new byte[1024];
DatagramPacket p = new DatagramPacket(buffer, buffer.length);
String str;

s.receive(p);
str = new String(buf, 0, p.getLength());
System.out.print(str);
s.send(p);
```

Client TCP :

```
try {
  Socket s = new Socket("serverName", 1234);
  OutputStream os = s.getOutputStream();
  InputStream is = s.getInputStream();
  os.write('a');
  System.out.println(is.read());
  s.close();
} catch(Exception e) {
  // Traitement d'erreur
}
```

Serveur TCP :

```
try {
  ServerSocket ecoute = new ServerSocket(1234);
  while(true) {
    Socket client = ecoute.accept();
    OutputStream os = client.getOutputStream();
    InputStream is = client.getInputStream();
    os.write('a');
    System.out.println(is.read());
    client.close();
  }
} catch(Exception e) {
  // Traitement d'erreur
}
```

Pour encore plus de facilités, utiliser les outils de Java :

```
BufferedReader br = new BufferedReader(new InputStreamReader(is, "utf-8"));
PrintStream ps = new PrintStream(os, false, "utf-8");

String line = br.readLine();
ps.println("hello " + line + "!\n");
ps.flush();
```

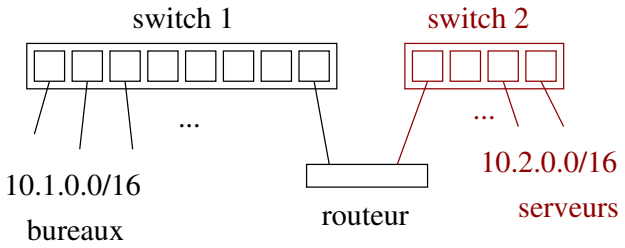
Pour désactiver l'algorithme de Nagle :

```
client.setTcpNoDelay(true)
```

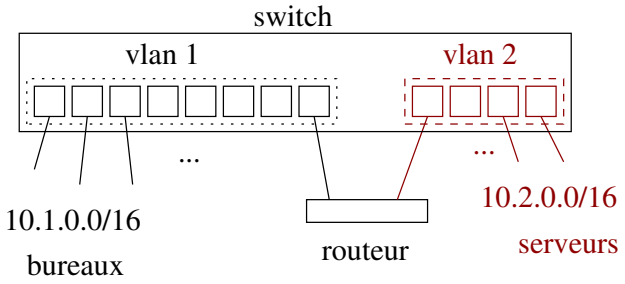
Pour effectuer des appels non bloquants, utiliser les classes `ServerSocketChannel` et `SocketChannel` de NIO. On peut aussi utiliser la méthode `setSoTimeout` des socket standard pour limiter la duré de bloquage

vlans

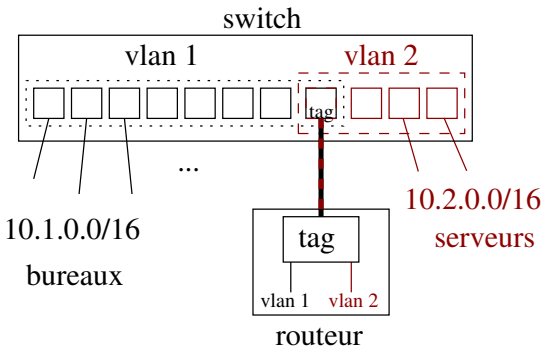
Deux switchs physiques



Un switch physique, deux vlans

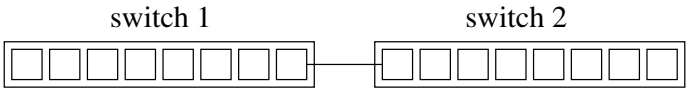


Un switch physique, deux vlans, un port "tagged" (ou "trunk")

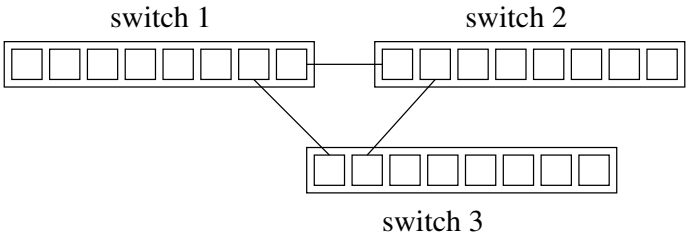


48bits destination	**48bits source**	**32bits 802.1Q**	**16bits protocol**	data...	**32bits checksum**

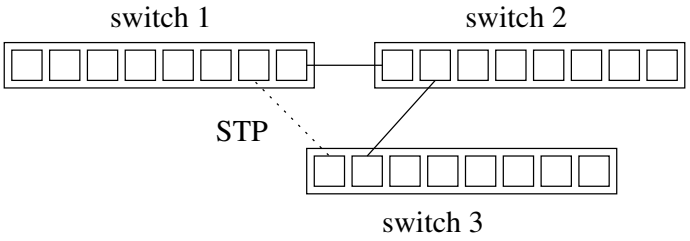
Deux switchs chaînés



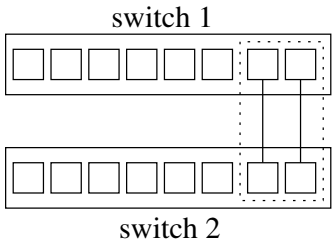
Trois switchs chaînés, ouille !



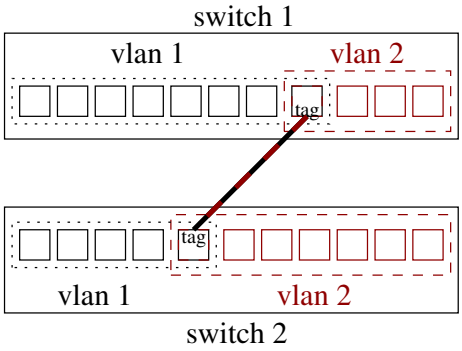
Trois switchs chaînés, avec STP



Deux switchs chaînés, avec link bundling



Deux switchs chaînés, avec vlan trunking



Un serveur de VMs