

# 4TIN401U / 4TAH501U Réseau

Samuel Thibault

## Contact

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

## Références

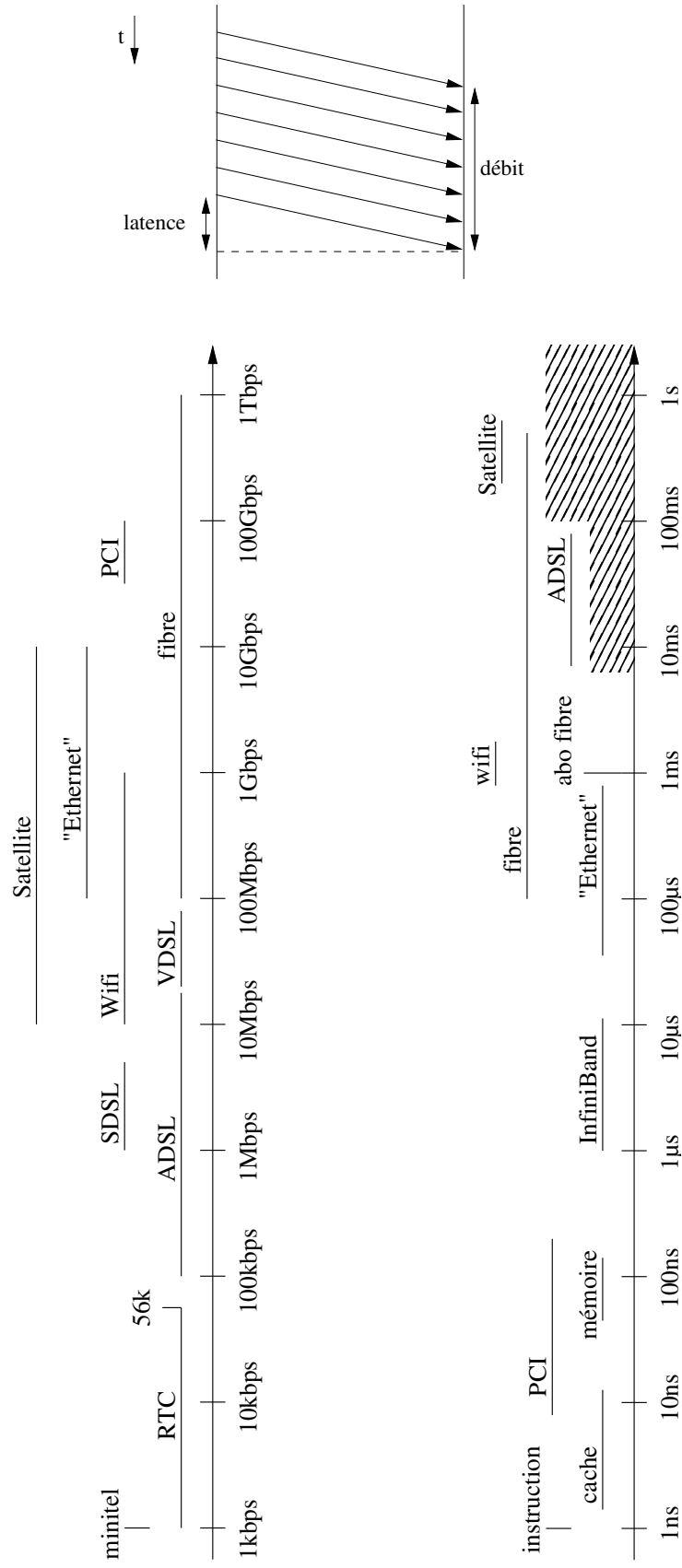
- *Réseaux*, Andrew Tanenbaum (encyclopédique)
- *TCP/IP*, Douglas Comer (beaucoup plus accessible)
- *Qu'est-ce qu'Internet*, 3 conférences de Benjamin Bayart.
- *What happens when you type google.com into your browser and press enter?*

## Plan du cours

- Parcours rapide de la pile
  - Introduction, piles de protocoles, principe d'encapsulation, calculs
- Parcours de la pile
  - Couche physique, adresse 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, heuristique, accessibilité
  - Flexibilité L2
  - Diffusion à grande échelle
  - Structure d'Internet à grande échelle

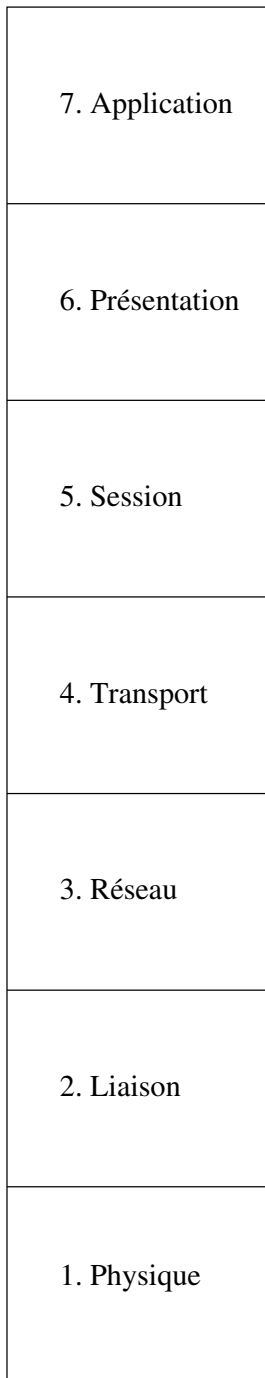


# Débit / latence





## Pile OSI

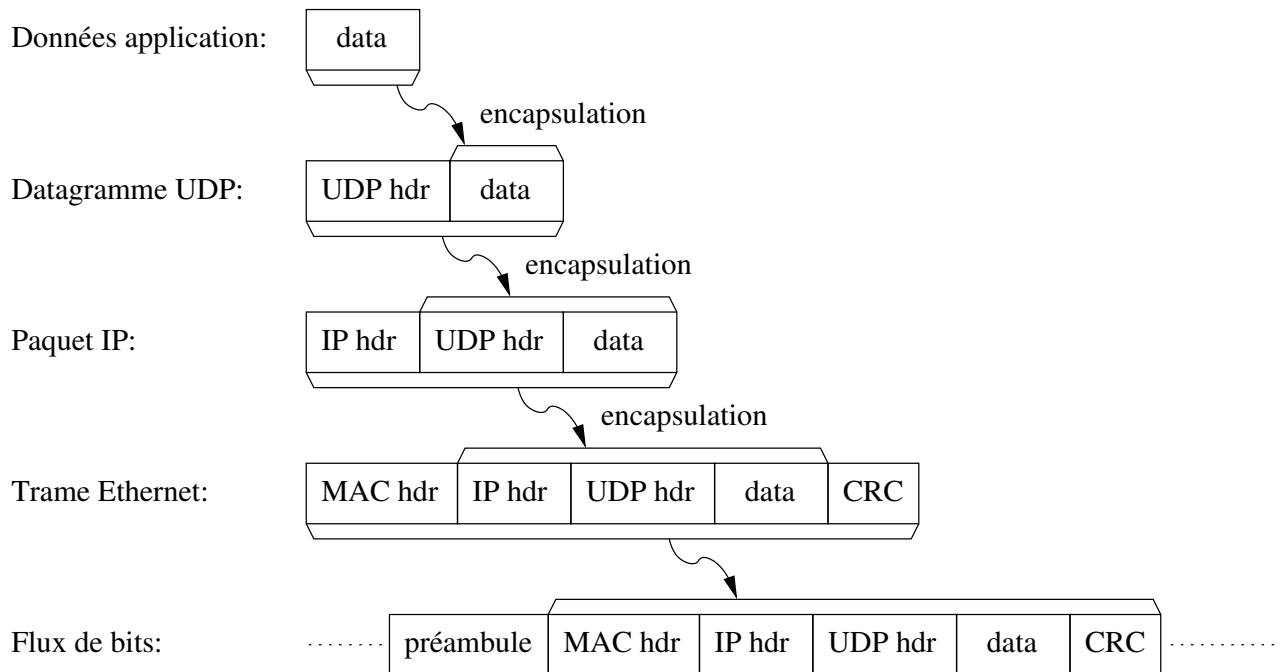


## File TCP/IP

7. Application			DNS		ping
6. Présentation	HTTP	SMTP		NTP	
5. Session					
4. Transport	TCP		UDP		ICMP
3. Réseau	IP				
2. Liaison	Ethernet		Wifi		...
1. Physique	Câble Ethernet		Ondes radio		...

# Encapsulation

*hdr = header = en-tête*



# 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.

$$\begin{aligned}
 10.0.0.0/9 &\equiv 10.0.0.0 \text{ avec masque réseau } 255.128.0.0 \\
 &\equiv 10.0.0.0 - 10.127.255.255.
 \end{aligned}$$

La plage a donc  $2^{32-n}$  adresses.

De même en IPv6, avec  $2^{128-n}$  adresses.

Utiliser `ipcalc` pour calculer les réseaux.

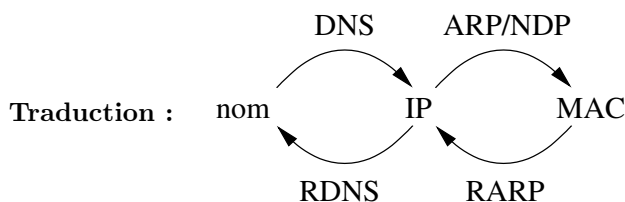
**Adresses privées :** 10.0.0.0/8, 172.16.0.0/12, 192.168.0.0/16, fc00::/7,

**Adresses locales :** 127.0.0.0/8, ::1/128

**Adresses automatiques locales au lien :** 169.254.0.0, 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.



# Configuration et Fichiers Linux

## Configuration IP

```
$ /sbin/ifconfig
eth0      Link encap:Ethernet  HWaddr 00:21:70:b4:36:49
          inet adr:193.50.110.76  Bcast:193.50.110.255  Masque:255.255.255.0
          adr inet6: fe80::221:70ff:feb4:3649/64 Scope:Lien
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:78693 errors:0 dropped:0 overruns:0 frame:0
          TX packets:51449 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 lg file transmission:1000
          RX bytes:44994757 (42.9 MiB)  TX bytes:14060107 (13.4 MiB)
          Interruption:18
```

## Cache de résolution IP/MAC

```
$ /usr/sbin/arp
Address          HWtype  HWaddress          Flags Mask          Iface
193.50.110.254   ether    00:1e:bd:5d:6b:00  C                   eth0
193.50.110.5     ether    00:16:3e:55:83:18  C                   eth0
```

## Configuration du routage

```
$ /sbin/route
Table de routage IP du noyau
Destination      Passerelle      Genmask          Indic Metric Ref       Use Iface
193.50.110.0    *                255.255.255.0   U        0      0        0 eth0
default         193.50.110.254  0.0.0.0         UG       0      0        0 eth0
```

## Configuration DNS

```
$ cat /etc/resolv.conf
nameserver 193.50.111.150
search bordeaux.inria.fr
```

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

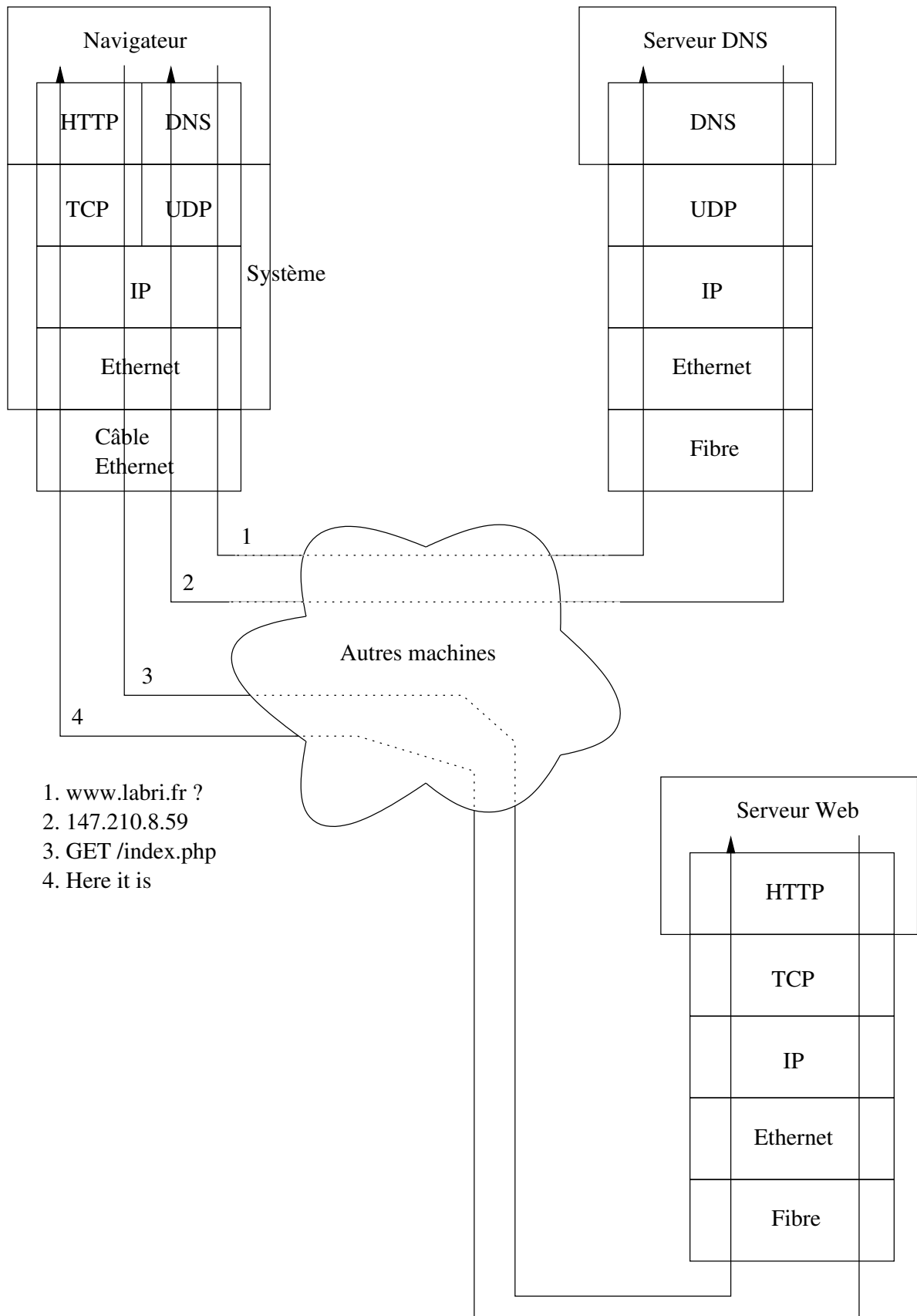
```
$ cat /etc/protocols
...
icmp 1 ICMP
tcp 6 TCP
udp 17 UDP
...
```

Les différents numéros de ports utilisés par UDP et TCP.

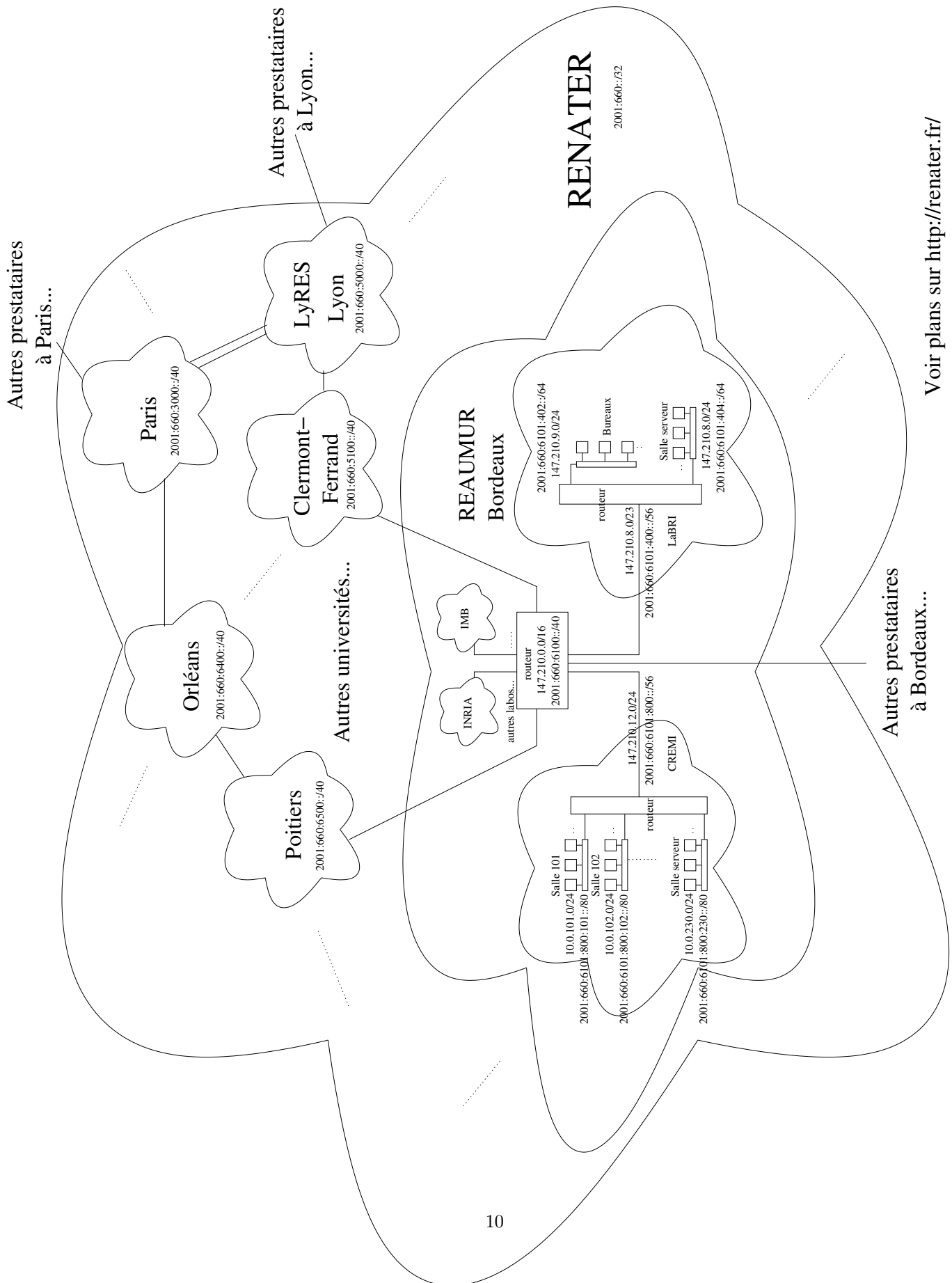
```
$ cat /etc/services
...
echo 7/udp
echo 7/tcp
ssh 22/tcp
telnet 23/tcp
smtp 25/tcp
domain 53/udp
http 80/tcp
...
```



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

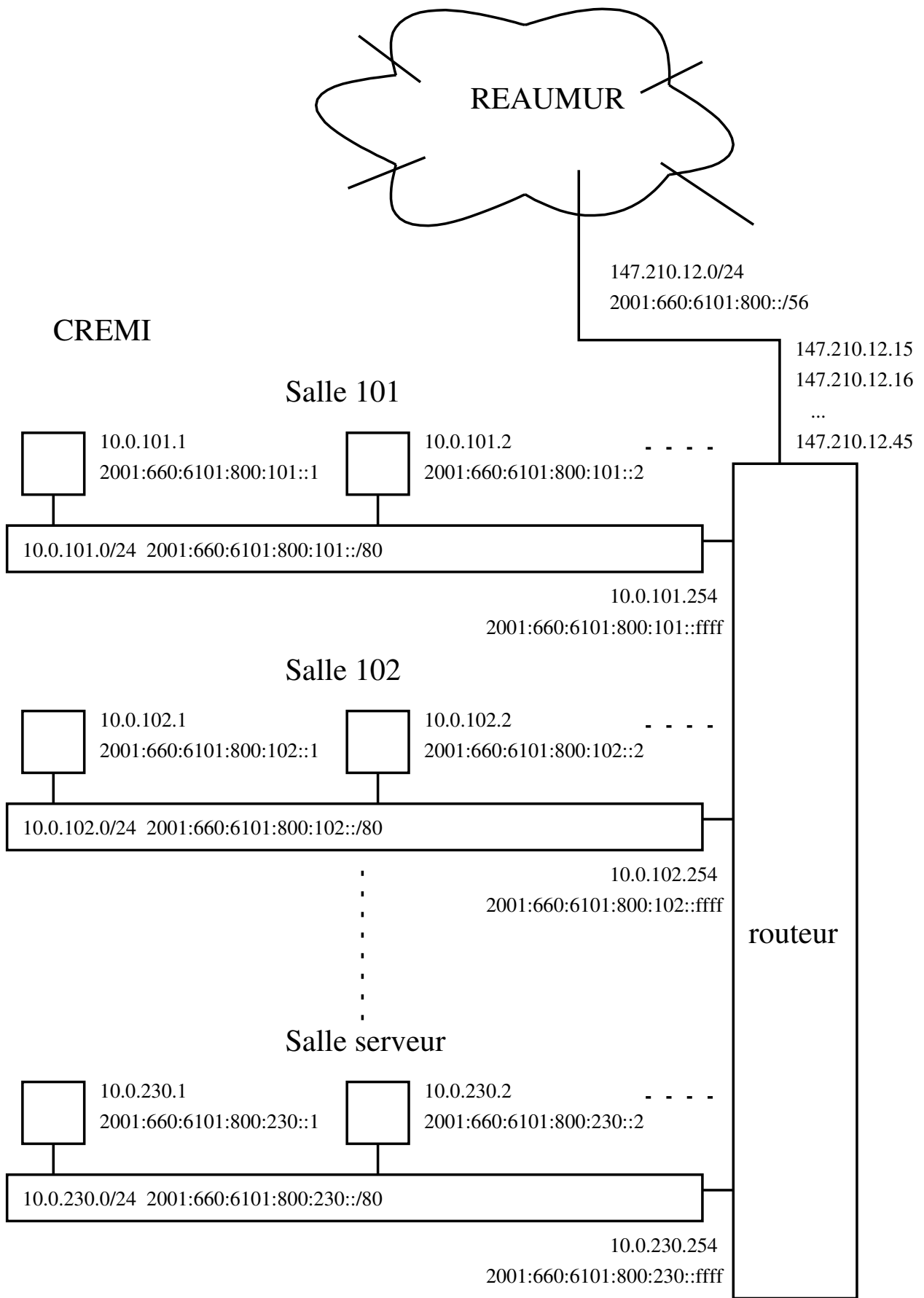


# Plan partiel de Renater et l'Université de Bordeaux I

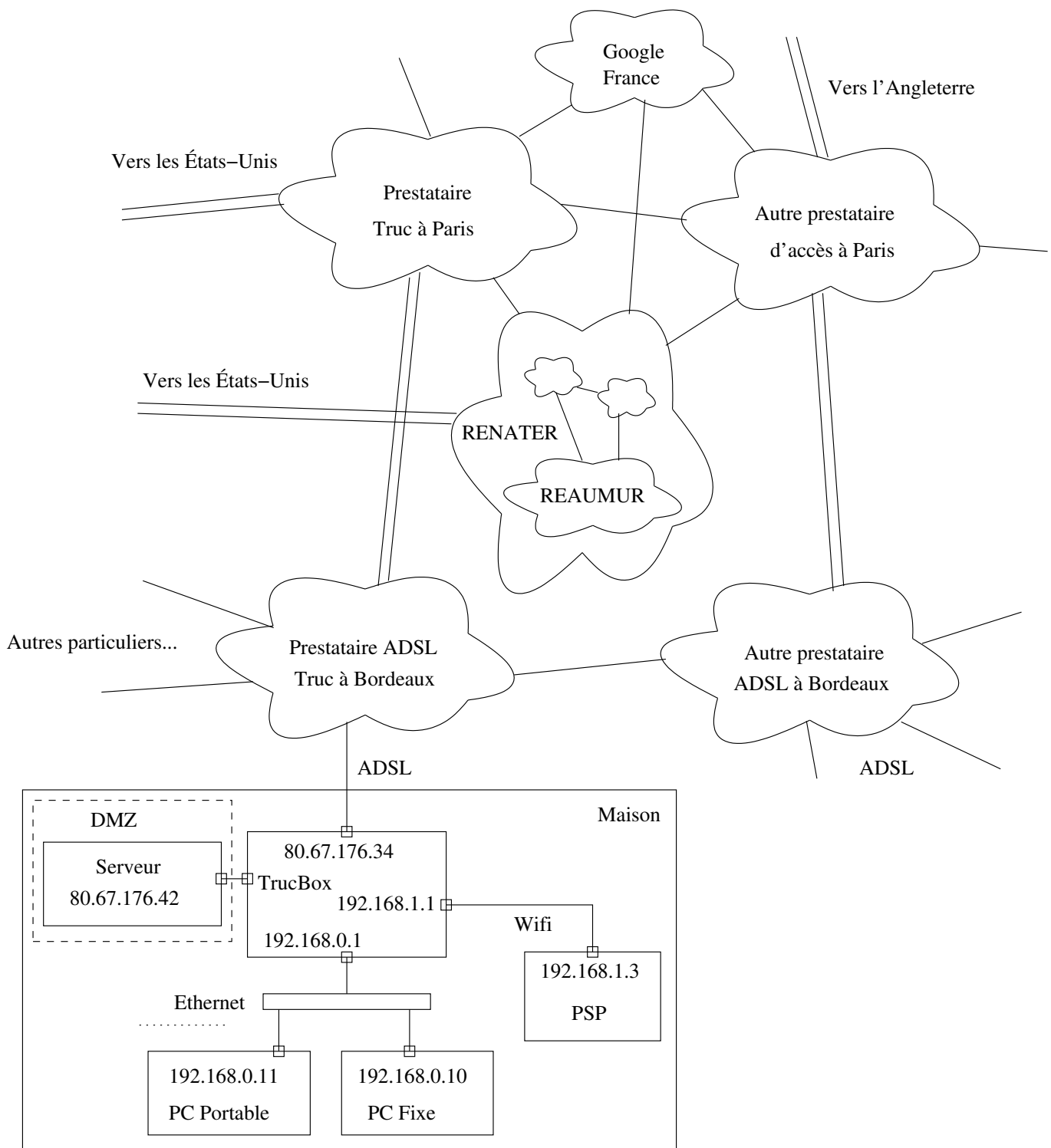


Voir plans sur <http://renater.fr/>

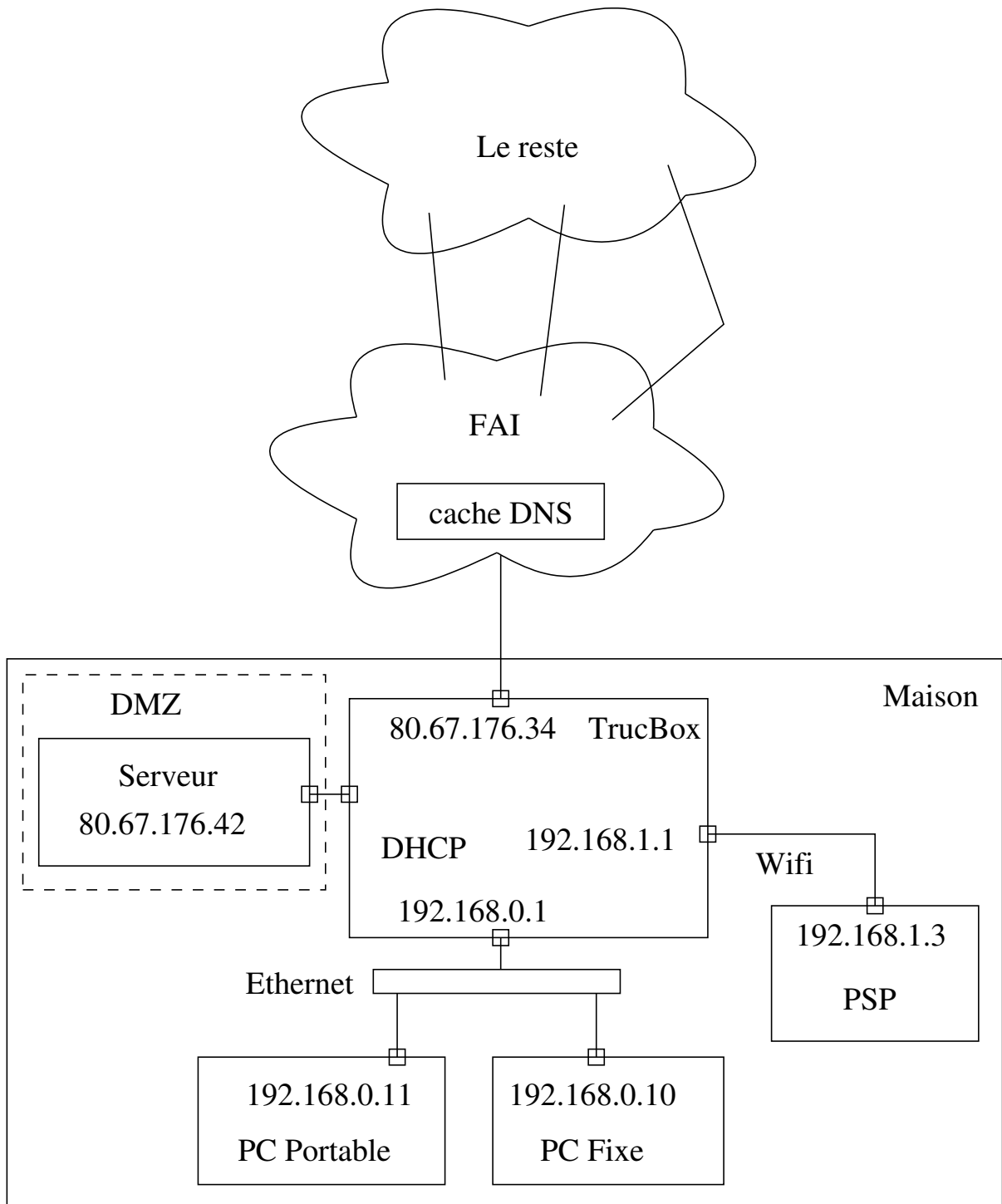
# Zoom sur le réseau du CREMI



# Plan typique « Maison » et prestataires Internet

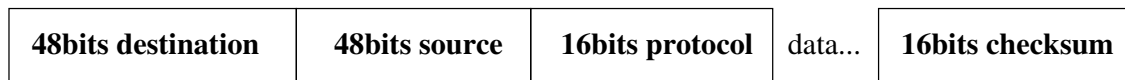


# Adresses / Firewall / Masquerading / NAT

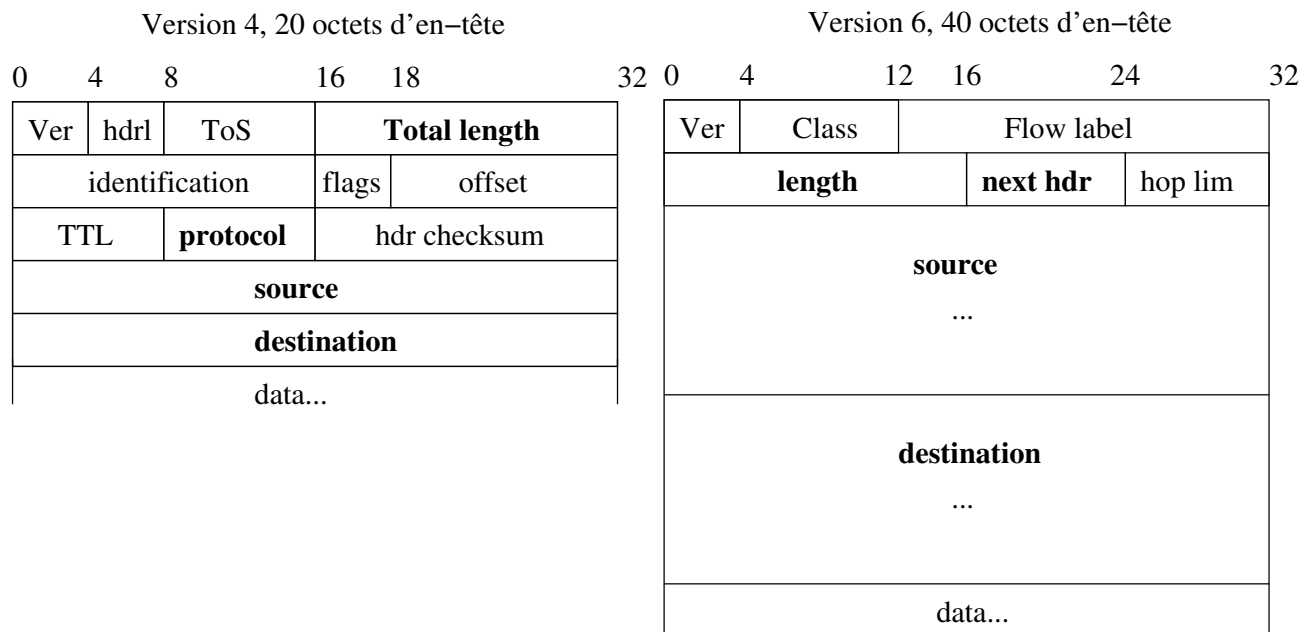


## Exemples d'en-têtes

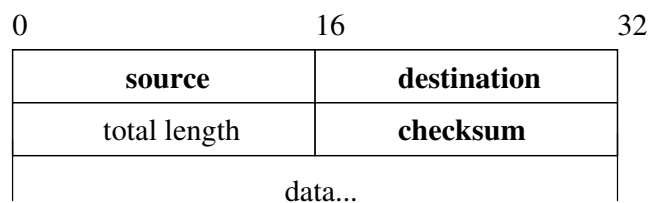
Ethernet, 14 octets d'en-tête et 2 octets de fin



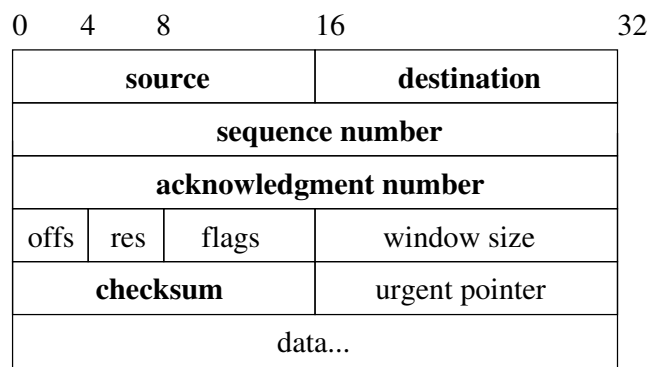
## IP



UDP, 8 octets d'en-têtes

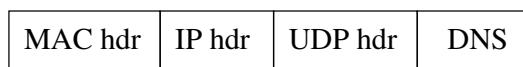


TCP, 20 octets d'en-tête



## Correspondance données brutes / en-têtes

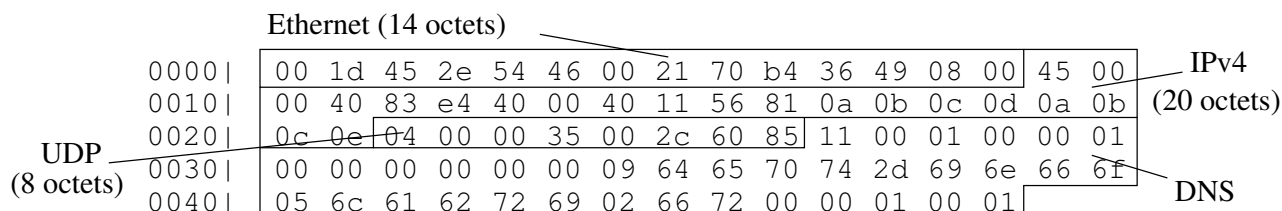
Exemple de trame Ethernet contenant un paquet IPv4, contenant un datagramme UDP, contenant une requête DNS, c'est-à-dire l'encapsulation :



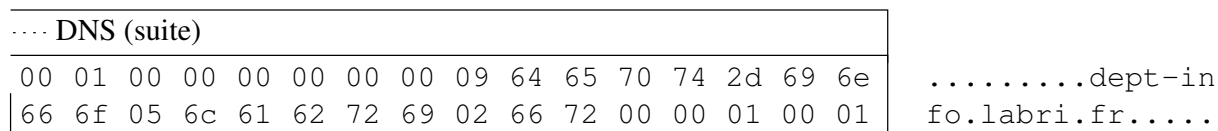
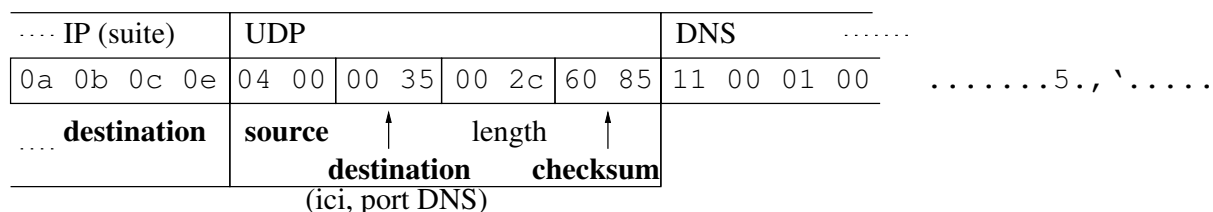
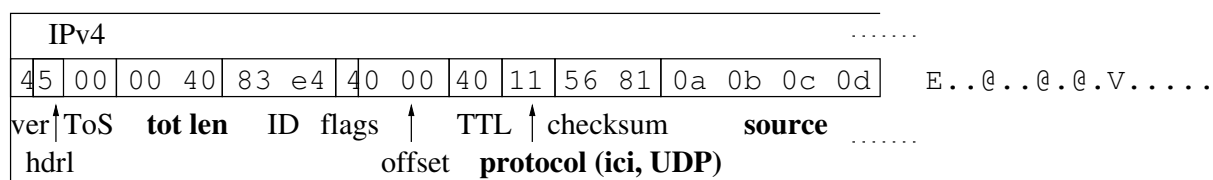
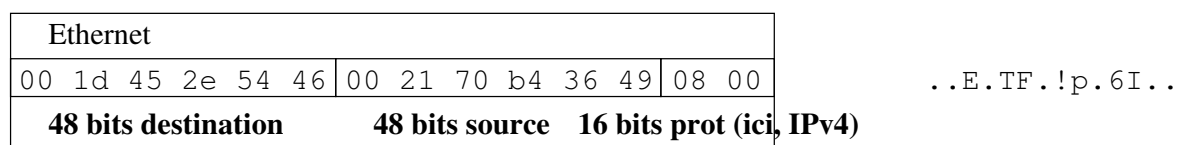
### Données ethernet brutes (ce qui passe sur le câble)

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

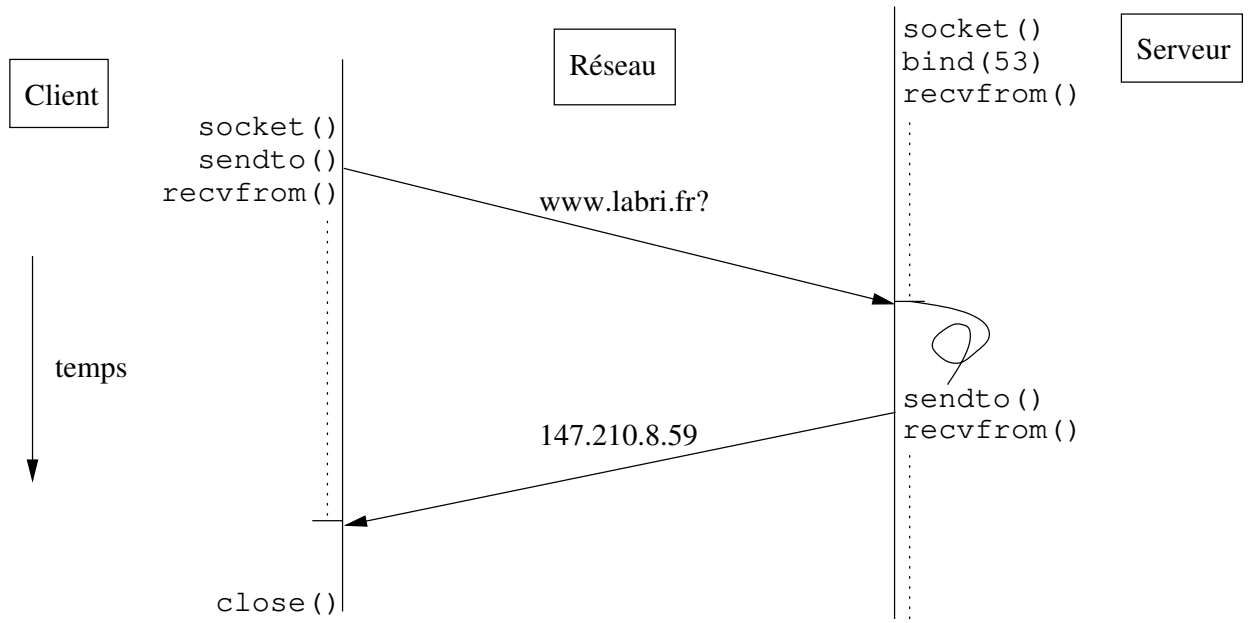


### Analyse à l'intérieur de chaque en-tête



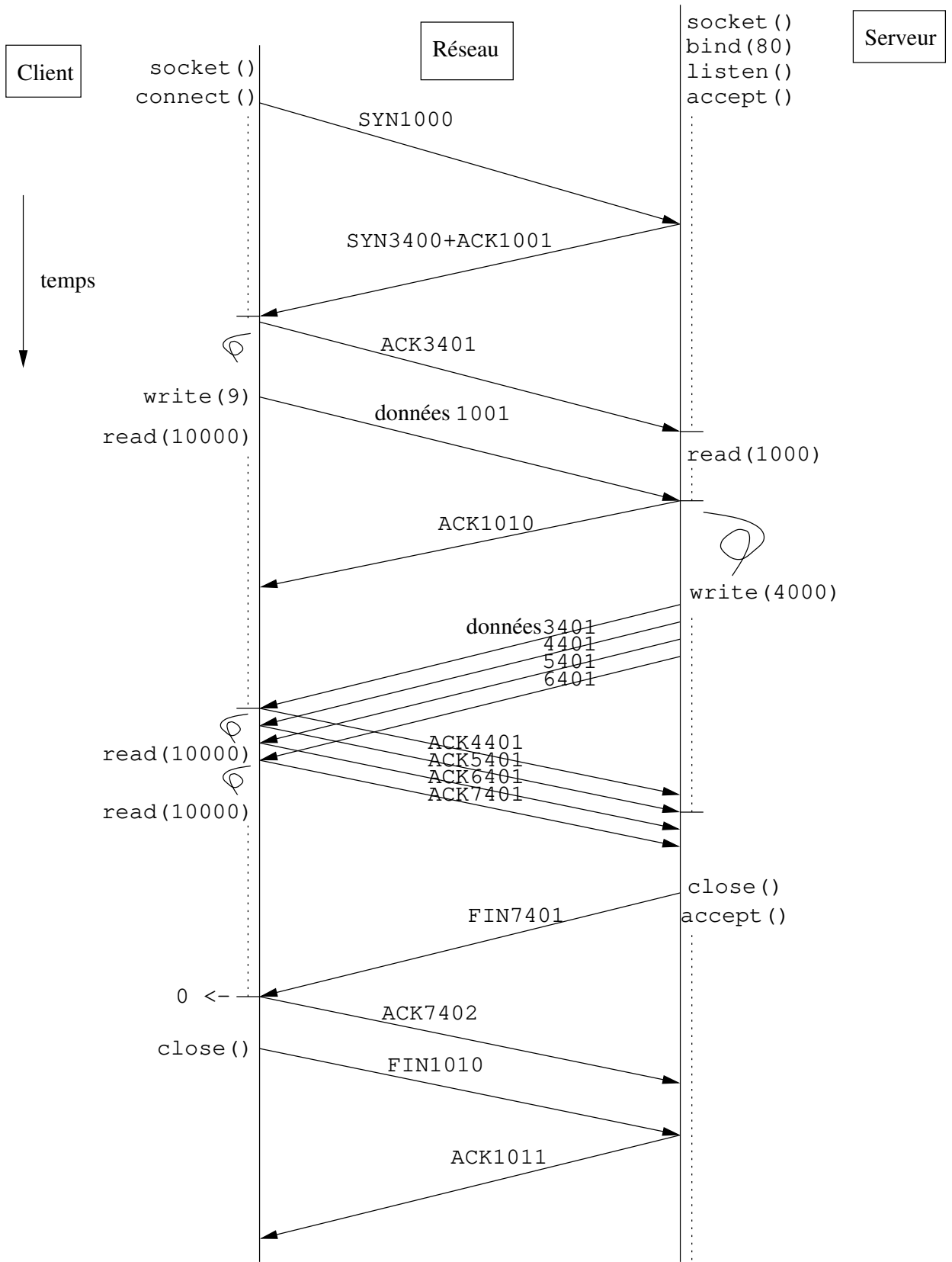
Exercice pour le lecteur : trouver le format de l'en-tête DNS sur Internet, et trouver le numéro du type de requête DNS utilisé ici.

# 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
```

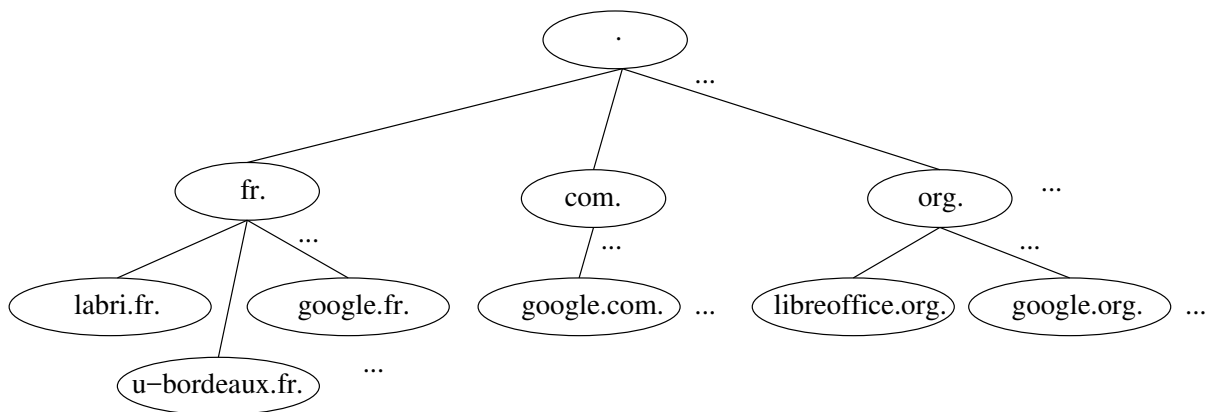
# Hiérarchie DNS

```
$ 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 C IP

```
#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(sa_family_t af, const char *ascii, void *binaire);
const char *inet_ntop(sa_family_t af, const void *binaire, char *ascii, socklen_t size);
```

## API C UDP

```
/* voir man 7 udp pour plus d'explications */

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

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

```
/* voir man 7 tcp pour plus d'explications */

/* 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);

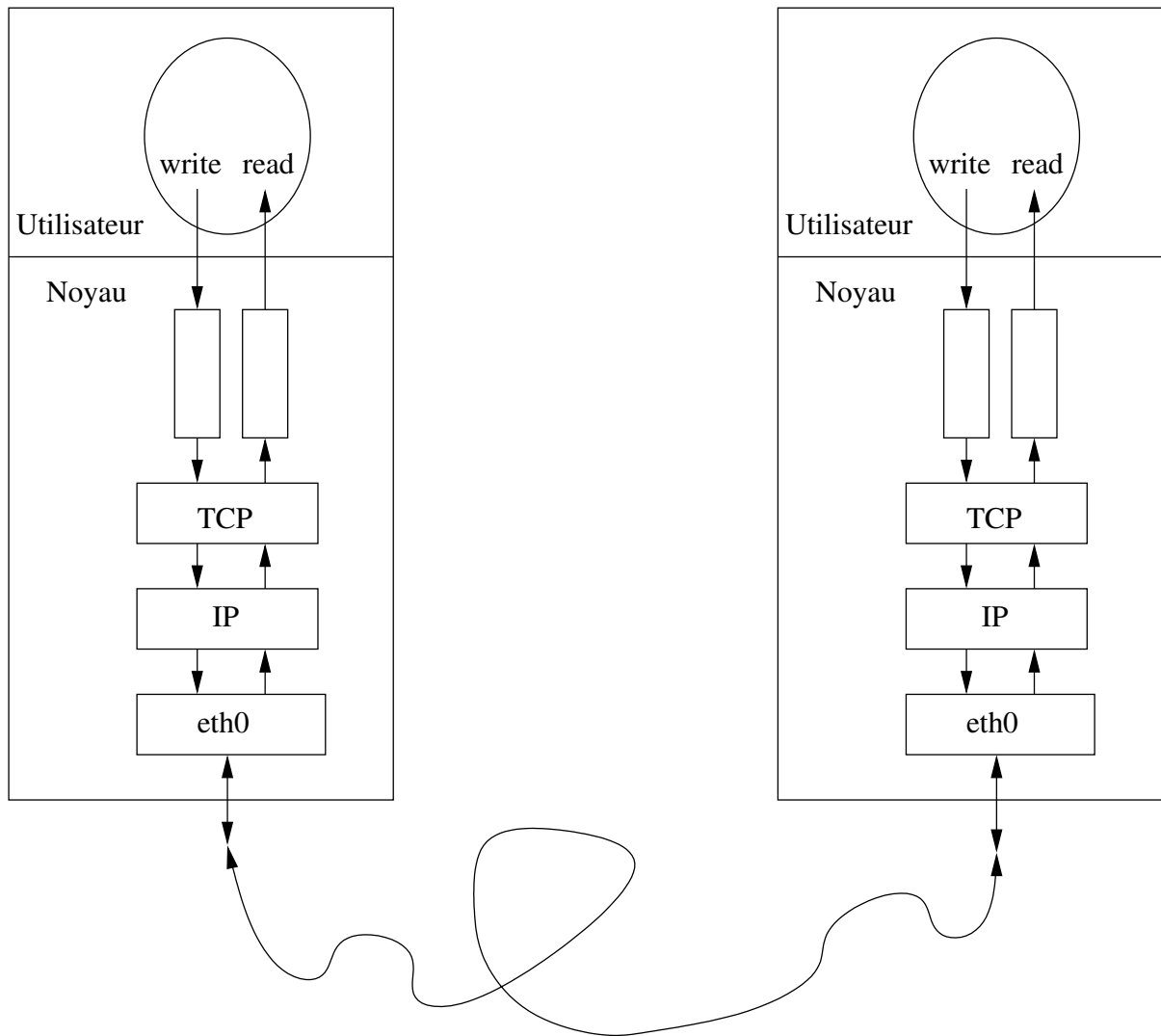
/** Partie serveur **/

/* 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 */
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);
```

# 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

```

## 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
 * 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 nfd, 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 nfd, int timeout);
```

## getaddrinfo

```
1 /* Client */

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

4 int sockfd;
5 struct addrinfo *res,*cur;
6 struct addrinfo hints;

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

10 /* Requête DNS */
11 gaierrno = getaddrinfo(host, port, &hints, &res);
12 if (gaierrno) {
13     fprintf(stderr,"getaddrinfo: %s\n", gai_strerror(gaierrno));
14     return -1;
15 }

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

18     /* Essayer d'établir la socket */
19     sockfd = socket(cur->ai_family, cur->ai_socktype, cur->ai_protocol);
20     if (sockfd < 0) {
21         if (errno != EAFNOSUPPORT) /* Éviter de râler si ipv6 n'est pas disponible */
22             perror("socket");
23         continue;
24     }

25     /* Essayer de se connecter */
26     if (connect(sockfd, cur->ai_addr, cur->ai_addrlen) < 0) {
27         perror("connect");
28         close(sockfd);
29         continue;
30     }

31     /* OK, on est connecté! */
32     break;
33 }

34 freeaddrinfo(res);

35 if (!cur) {
36     fprintf(stderr, "could not connect\n");
37     return -1;
38 }
```



```

1 /* Serveur */
2 const char *host = "";
3 const char *port = "http";

4 int sockfd;
5 struct addrinfo *res,*cur;
6 struct addrinfo hints;

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

11 /* Requête DNS */
12 gaierrno = getaddrinfo(host, port, &hints, &res);
13 if (gaierrno) {
14     fprintf(stderr,"getaddrinfo: %s\n", gai_strerror(gaierrno));
15     return -1;
16 }

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

19     /* Essayer d'établir une socket de ce type */
20     sockfd = socket(cur->ai_family, cur->ai_socktype, cur->ai_protocol);
21     if (sockfd < 0) {
22         if (errno != EAFNOSUPPORT) /* Éviter de râler si ipv6 n'est pas disponible */
23             perror("socket");
24         continue;
25     }

26     /* Essayer d'établir une socket d'écoute pour cette adresse */
27     if (bind(sockfd, ai->ai_addr, ai->ai_addrlen) < 0) {
28         perror("bind");
29         close(sockfd);
30         continue;
31     }

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

34     ... sockfd ...
35 }

36 freeaddrinfo(res);

```

Voir aussi la réciproque de `getaddrinfo` :

```

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("www.labri.fr");
byte[] message = "abcd".getBytes();
DatagramPacket p = new DatagramPacket(message, message.length, address, 1234);
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 classe `ServerSocketChannel` et `SocketChannel` de NIO. On peut aussi utiliser la méthode `setSoTimeout` des socket standard pour limiter la durée de blocage

## API Python

La documentation est concentrée sur deux pages :

<https://docs.python.org/library/socket.html>

<https://docs.python.org/library/select.html>

Les principes sont strictement identiques à l'API C, juste pythonesqués et donc bien plus agréables :)