Réseau

M1 - CHPS

Architecture Interne des Systèmes d'exploitations (AISE)

Jean-Baptiste Besnard
<jean-baptiste.besnard@paratools.com>

Julien Adam
<julien.adam@paratools.com>

Organisation

- Chaque session est découpée en deux parties. Un cours théorique le matin et une mise en pratique l'après-midi (TD) portant sur les connaissances vues le matin.
- Des QCMs sur les bases **importantes** au fil des semaines et portant sur un cours précédent. Le QCM aura toujours lieu durant la matinée
- Un Projet, date de rendu au **19/12/2023 Minuit**
- Un Examen final le **22/12/2023 (Matin)**

1 - Généralités sur les OS et Entrées-Sorties
2 - Compilation, Bibliothèques et Layout Mémoire
3 - Mémoire partie 2, Layout Binaire, Runtime
4 - Programmation réseau et entrées/sorties avancées
5 - Virtualisation et Conteneurs
6 - Programmation Noyau
7 - Scheduling et Temps-Réel
8 - Examen Ecrit et Présentations

Type d'Examen	Coefficient
QCMs	10 %
Projet	40 %
EXAMEN	50 %

Cours et Corrections

https://github.com/besnardjb/AISE_24

Programme

1 - Compilation, Bibliothèques et Layout Mémoire

2 - Généralités sur les OS et Entrées-Sorties

3 - Mémoire partie 2, Layout Binaire, Runtime

4 - Programmation Réseau Et Conteneurs

5 - Programmation Noyau

6 - Virtualisation et Conteneurs Avancés

7 - Scheduling et Temps-Réel

8 - Examen Ecrit et Présentations

Introduction au Réseau

Programmation Réseau

En HPC le réseau est un aspect crucial car les machines sont:

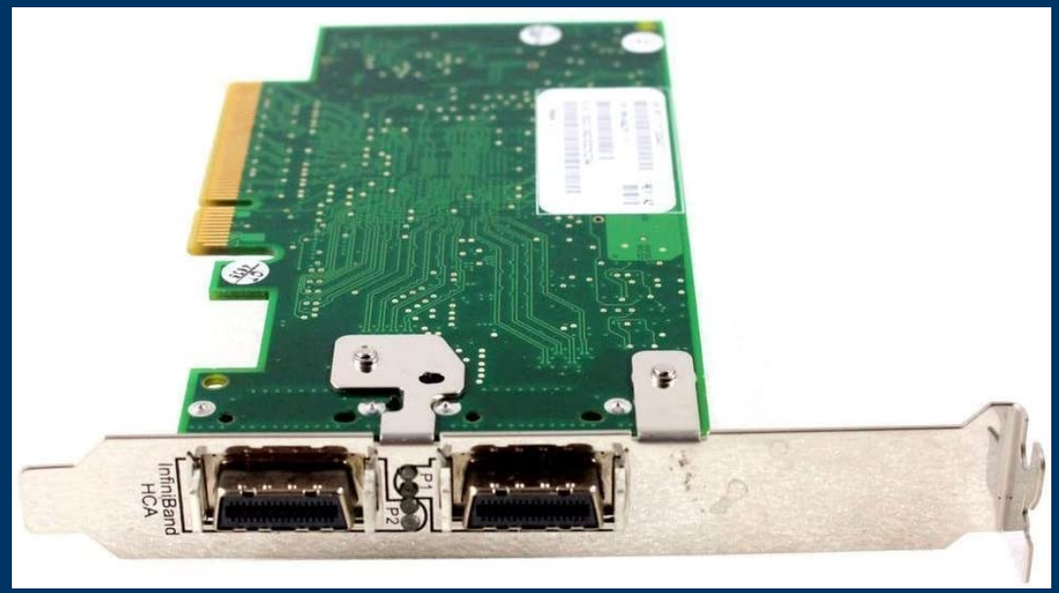
- Distribuées (milliers de noeuds)
- La mémoire par noeud est limitée

Il faut donc pouvoir interconnecter des milliers de processus UNIX pour former un calcul cohérent. MPI permet généralement cela
Mais il repose sur des réseaux et techniques plus bas niveau dont nous avons déjà vu certaines:

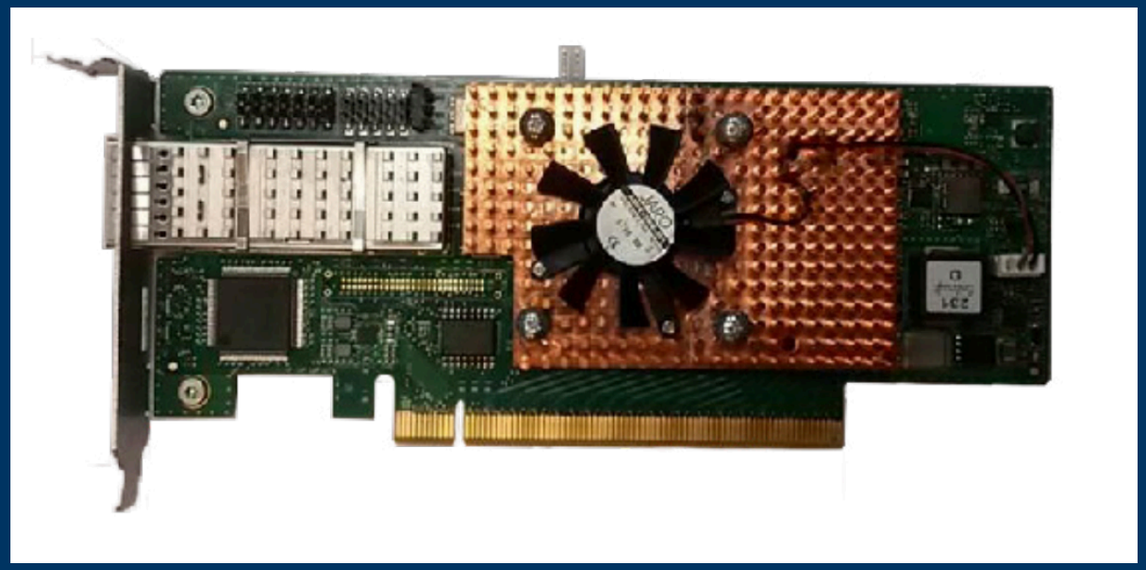
- Segment SHM
- Réseaux TCP (rare)
- Réseaux rapides (voir aperçu slide suivant)

Aujourd'hui nous allons implémenter des appels que vous faites plusieurs milliers de fois par jour à chaque fois que vous allez sur internet, consultez vos mail, parlez à votre assistant vocal …

Réseaux HPC



Infiniband (libverbs)



Bull Exascale Interconnect (Portals 4)

- Tofu interconnect
- Aries Interconnect
- Cray Slighshot
- TH Express

Base du Réseau

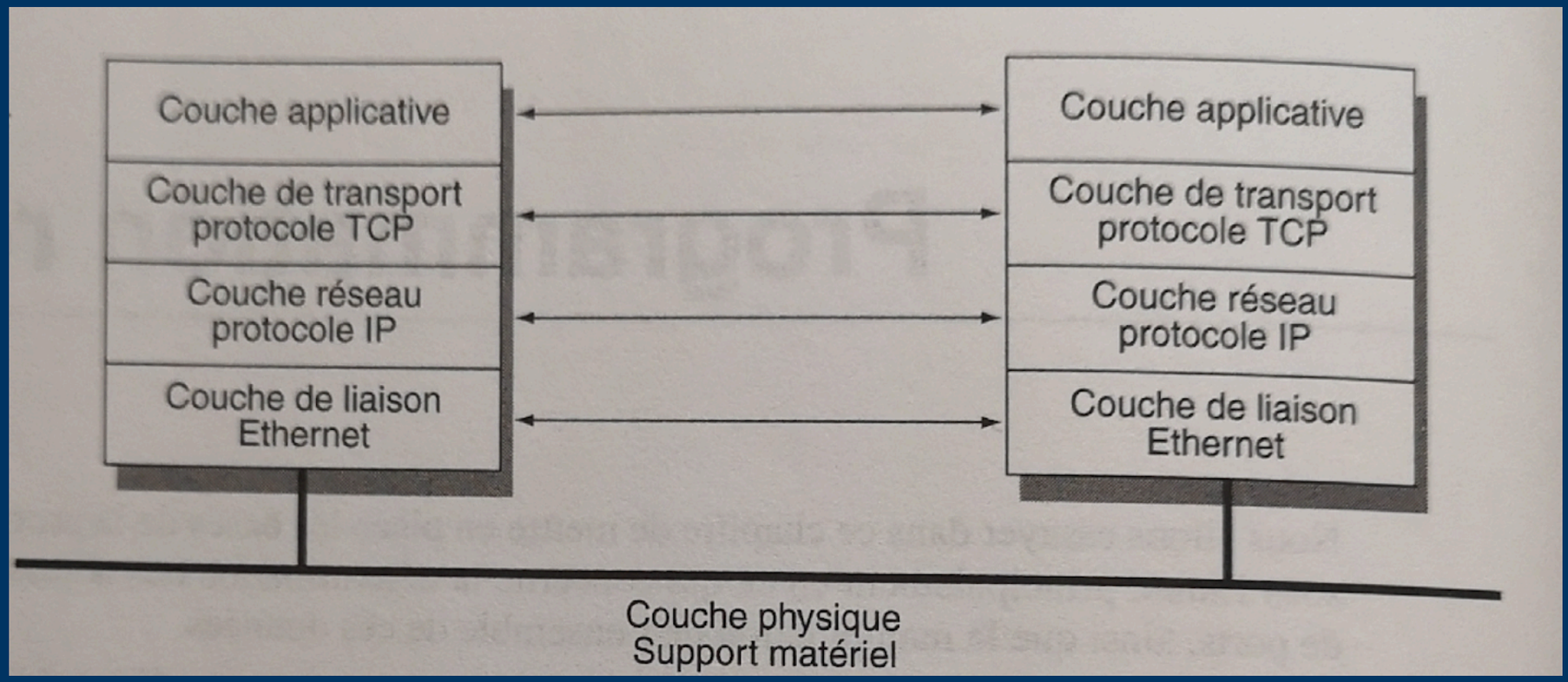
Très rapidement le but d’un réseau est de connecter des machines entre elles on peut citer comme support de réseau:

- Cable Ethernet (RJ45)
- Wifi (802.11a/b/g/n/ac)
- Bluetooth (802.15.1)
- Zigbee (802.15.4) (domotique)

D’un point de vue système ces différents réseaux reposent sur des interfaces relativement similaires nous allons voir aujourd’hui principalement du TCP mais sachez que cela se transpose assez facilement à d’autre types de réseaux.

Modèle en Couche (Rappel)

Version simplifiée (à 5 couche) du modèle OSI (Open System Interconnection)



- **Couche ethernet -> adresses MAC**
- **Couche IP -> addresses IP**
- **Couche TCP -> Modèle d'encapsulation des données**
- **Couche applicative -> Ce qui est transmis**

TCP = Transmission Control Protocol

TCP/IP

Couche IP

- **Chaque machine possède une adresse IP**
  ➡ sur 4 Octets (32 bits) pour IPV4
  ➡ Sur 16 octets (128 bits) IPV6

- **Elle communique sur un réseau identité par un masque:**
  ➡ 255.255.255.0 (IPV4) -> 254 machines (la valeur 255 est réservée)

```
Address:   192.168.0.1          11000000.10101000.00000000 .00000001
Netmask:   255.255.255.0 = 24   11111111.11111111.11111111 .00000000
Wildcard:  0.0.0.255            00000000.00000000.00000000 .11111111
=>
Network:   192.168.0.0/24       11000000.10101000.00000000 .00000000 (Class C)
Broadcast: 192.168.0.255        11000000.10101000.00000000 .11111111
HostMin:   192.168.0.1          11000000.10101000.00000000 .00000001
HostMax:   192.168.0.254        11000000.10101000.00000000 .11111110
Hosts/Net: 254                  (Private Internet)
```

Ce réseau est 192.168.0.0/24 (selon les bits de masquage)

Couche IP

```
Address:   192.168.0.1          11000000.10101000 .00000000.00000001
Netmask:   255.255.0.0 = 16     11111111.11111111 .00000000.00000000
Wildcard:  0.0.255.255          00000000.00000000 .11111111.11111111
=>
Network:   192.168.0.0/16       11000000.10101000 .00000000.00000000 (Class C)
Broadcast: 192.168.255.255      11000000.10101000 .11111111.11111111
HostMin:   192.168.0.1          11000000.10101000 .00000000.00000001
HostMax:   192.168.255.254      11000000.10101000 .11111111.11111110
Hosts/Net: 65534                (Private Internet)
```

65534 machines pour un réseau de masque /16

= 256 x 256 - 2

Exemple d’adresses non routable sur Internet

RFC1918 name	IP address range	Count	largest subnet mask)	host id size	mask bits	classful description[Note 1]
24-bit block	10.0.0.0 – 10.255.255.255	16777216	10.0.0.0/8 (255.0.0.0)	24 bits	8 bits	single class A network
20-bit block	172.16.0.0 – 172.31.255.255	1048576	172.16.0.0/12 (255.240.0.0)	20 bits	12 bits	16 contiguous class B networks
16-bit block	192.168.0.0 – 192.168.255.255	65536	192.168.0.0/16 (255.255.0.0)	16 bits	16 bits	256 contiguous class C networks

Afficher l'adresse IP

```
$ip address
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: enp3s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen
1000
    link/ether 98:90:96:d2:65:0e brd ff:ff:ff:ff:ff:ff
    inet 192.168.201.42/24 brd 192.168.201.255 scope global dynamic enp3s0
       valid_lft 26772sec preferred_lft 26772sec
    inet6 fe80::9a90:96ff:fed2:650e/64 scope link
       valid_lft forever preferred_lft forever
3: wlp4s0: <BROADCAST,MULTICAST> mtu 1500 qdisc mq state DOWN group default qlen 1000
    link/ether 8a:8d:54:80:7b:23 brd ff:ff:ff:ff:ff:ff
```

Afficher l’adresse IP

```
2: enp3s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 98:90:96:d2:65:0e brd ff:ff:ff:ff:ff:ff
    inet 192.168.201.42/24 brd 192.168.201.255 scope global dynamic enp3s0
       valid_lft 26772sec preferred_lft 26772sec
    inet6 fe80::9a90:96ff:fed2:650e/64 scope link
       valid_lft forever preferred_lft forever
```

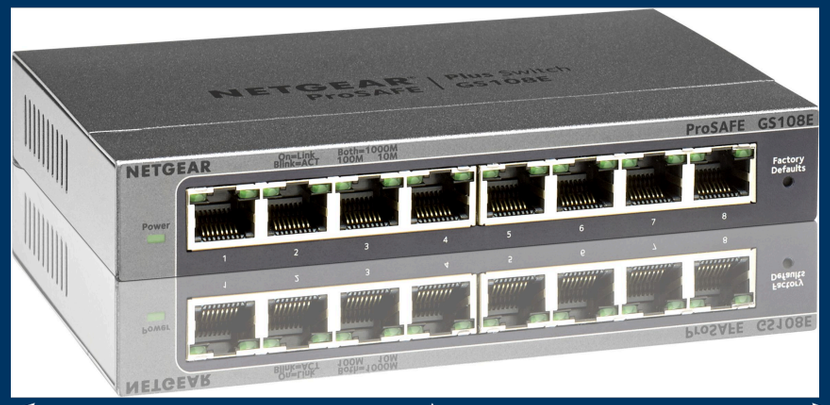
- Interface -> enp3s0
- Connectée -> oui (UP)
- Adresse IPV4 -> 192.168.201.42/24
- Adresse IPV6 -> fe80::9a90:96ff:fed2:650e/64

Afficher les Routes

```
$ip route
default via 192.168.201.1 dev enp3s0 proto static metric 100
192.168.201.0/24 dev enp3s0 proto kernel scope link src 192.168.201.42 metric 100
```

- 192.168.201.0/24 est sur enp3s0
- « default » tout le reste est sur la machine 192.168.201.1

Couche Physique



Cable RJ45

Cable RJ45

Cable RJ45

192.168.201.2

« Julius »

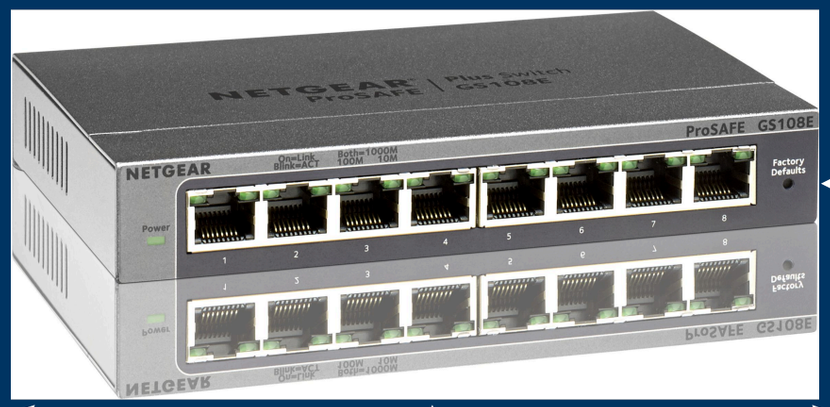
192.168.201.3

« Tommy »

192.168.201.4

« Bob »

Couche Physique



LAN IP

WAN IP

192.168.201.1

Cable RJ45

Cable RJ45

Cable RJ45

192.168.201.2

« Julius »

192.168.201.3

« Tommy »

192.168.201.4

« Bob »

WAN = Wide Area Network

LAN = Local Area Network

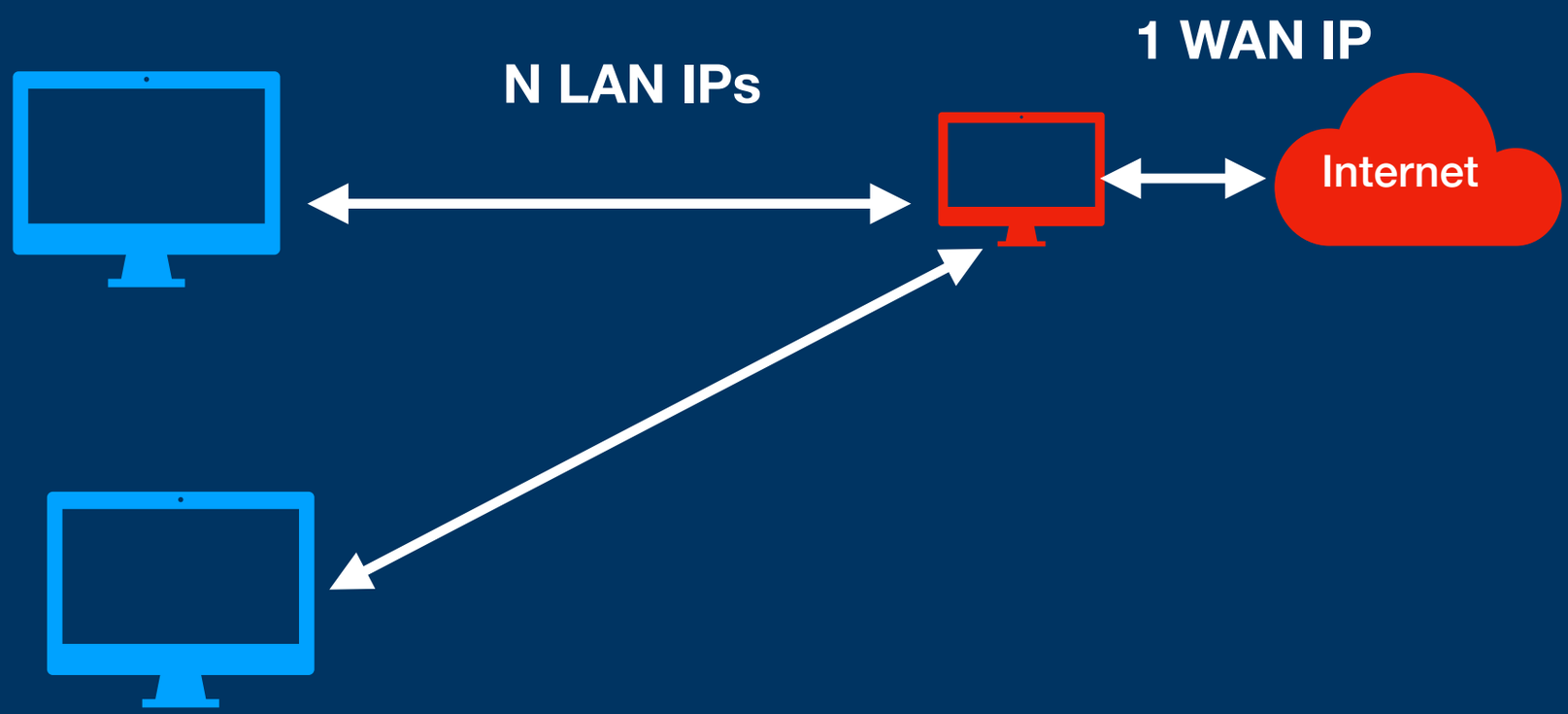
La Passerelle

- La machine qui est connectée à la fois à internet et au réseau local est appelée la « passerelle ».
- Chez les particulier c’est souvent une « Box » qui opère comme routeur entre ces deux réseau. Plus généralement c’est une machine avec au moins deux interface réseau.

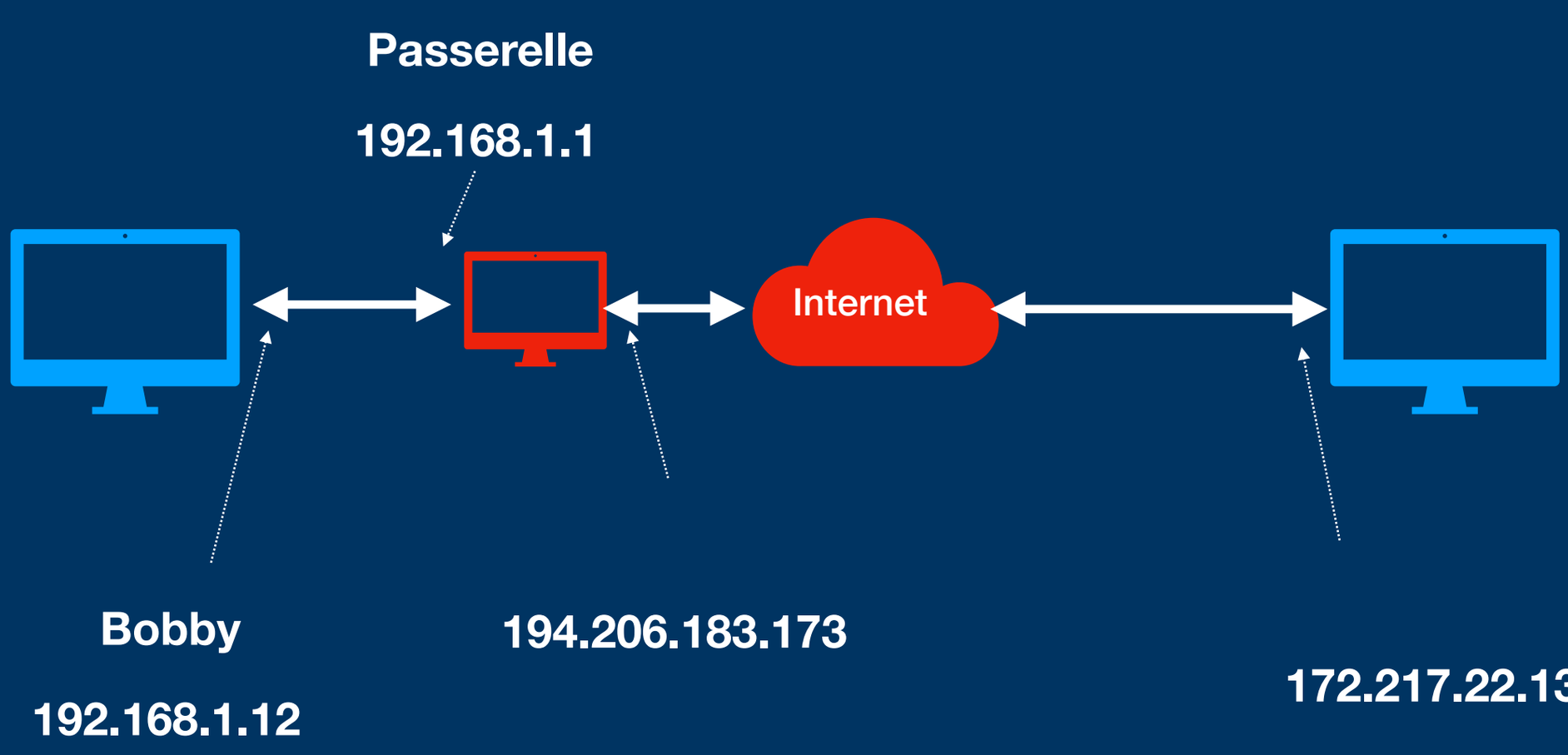
La Passerelle en Entreprise

Exemples CISCO de routeurs configurables.

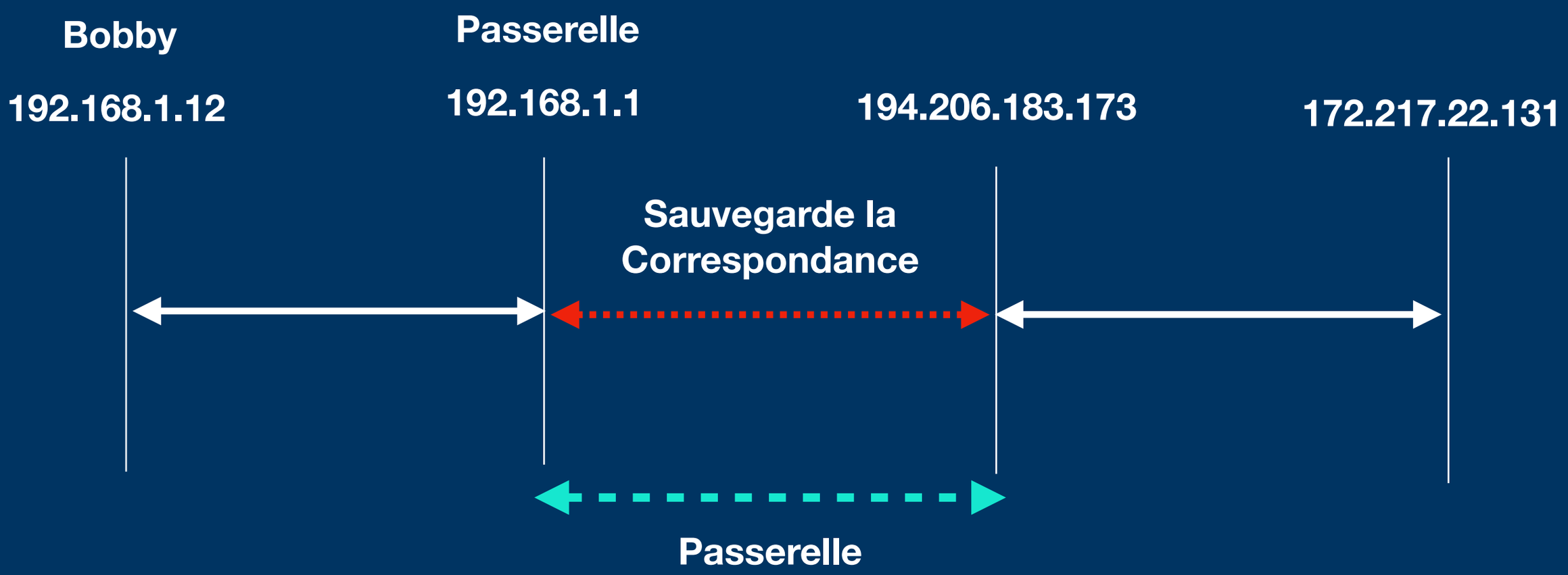
Network Address Translation



Network Address Translation



Network Address Translation



Google ne voit que votre passerelle qui se fait passer pour vous et vous renvoie les données

Transmission Control Protocol

- TCP permet d'établir des connections « fiables » entre machines en permettant:
  - ➡ Une validation de l'intégrité des données
  - ➡ Un mode connecté (A parle à B)
  - ➡ Un contrôle de flux (l'envoi bloque si la source ne lit pas assez vite par exemple)

- TCP connecte deux **IP** et deux **PORTS**
  - ➡ Un port est une « porte » vers votre machine
  - ➡ Il peut être « ouvert » ou « sortant »
  - ➡ Toute connection TCP est bidirectionnelle (deux FD)
  - ➡ Un port peut être en écoute (en attente de nouvelle connexions)
  - ➡ Plusieurs connexions peuvent utiliser le même port en écoute

192.168.201.2 : 3689

192.168.201.9 : 80

Services Communs

Protocol	Port	Name	Description
FTP	tcp/20, tcp21	File Transfer Protocol	Sends and receives files between systems
SSH	tcp/22	Secure Shell	Encrypted console access
Telnet	tcp/23	Telecommunication Network	Insecure console access
SMTP	tcp/25	Simple Mail Transfer Protocol	Transfer email between mail servers
DNS	udp/53, tcp/53	Domain Name System	Convert domain names to IP addresses
HTTP	tcp/80	Hypertext Transfer Protocol	Web server communication
POP3	tcp/110	Post Office Protocol version 3	Receive email into a email client
IMAP4	tcp/143	Internet Message Access Protocol v4	A newer email client protocol
HTTPS	tcp/443	Hypertext Transfer Protocol Secure	Web server communication with encryption
RDP	tcp/3389	Remote Desktop Protocol	Graphical display of remote devices
NetBIOS	udp/137	NetBIOS name service	Register, remove, and find Windows services by name
NetBIOS	udp/138	NetBIOS datagram service	Windows connectionless data transfer
NetBIOS	tcp/139	NetBIOS session service	Windows connection-oriented data transfer
SLP	tcp/427, udp/427	Service Location Protocol	Find Mac OS services by name
SMB	tcp/445	Server Message Block	Windows file transfers and printer sharing
AFP	tcp/548	Apple Filing Protocol	Mac OS file transfers

Résolution de Noms

Serveur DNS

Traduire un nom de domaine (Domain Name) en une adresse IP.

Par exemple:

```
$ dig google.com

; <<>> DiG 9.10.3-P4-Debian <<>> google.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 35279
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 512
;; QUESTION SECTION:
;google.com.                            IN      A

;; ANSWER SECTION:
google.com.             270     IN      A       216.58.204.110

;; Query time: 9 msec
;; SERVER: 192.168.201.127#53(192.168.201.127)
;; WHEN: Mon Feb 24 12:05:09 CET 2020
;; MSG SIZE  rcvd: 55
```

Serveur DNS

On peut faire cette traduction en IPV6 (record AAAA)

Par exemple:

$ dig **AAAA** google.com

```
; <<>> DiG 9.10.3-P4-Debian <<>> AAAA google.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 44141
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 512
;; QUESTION SECTION:
;google.com.                            IN      AAAA

;; ANSWER SECTION:
google.com.             299     IN      AAAA    2a00:1450:4007:80f::200e

;; Query time: 10 msec
;; SERVER: 192.168.201.127#53(192.168.201.127)
;; WHEN: Mon Feb 24 12:05:39 CET 2020
;; MSG SIZE  rcvd: 67
```

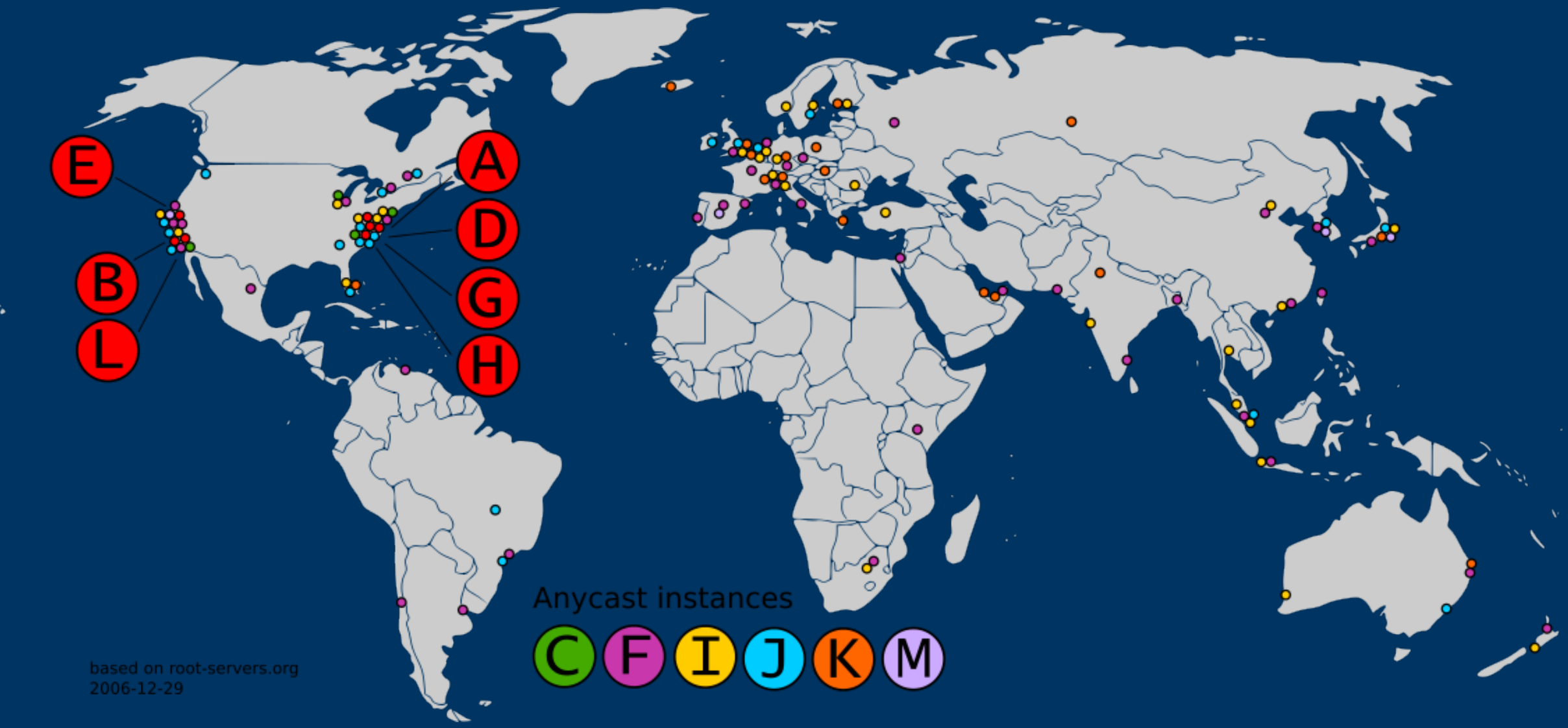
Serveur DNS

13 Serveur racine (source wikipedia)

Lettre	adresse IPv4[9]	adresse IPv6[9]	Ancien nom	Société	Localisation	Logiciel
A[10]	198.41.0.4	2001:503:ba3e::2:30	ns.internic.net	VeriSign	trafic distribué par anycast	BIND
B[11]	199.9.14.201[Notes 1]	2001:500:200::b	ns1.isi.edu	Université de Californie du Sud	Marina Del Rey, Californie, États-Unis	BIND
C[13]	192.33.4.12[14]	2001:500:2::c	c.psi.net	Cogent Communications	trafic distribué par anycast	BIND
D[15]	199.7.91.13[16],[Notes 2]	2001:500:2d::d	terp.umd.edu	Université du Maryland	College Park, Maryland, États-Unis	BIND
E[17]	192.203.230.10[17]	2001:500:a8::e	ns.nasa.gov	NASA	Mountain View, Californie, États-Unis	BIND
F[18]	192.5.5.241[19]	2001:500:2f::f	ns.isc.org	Internet Systems Consortium	trafic distribué par anycast	BIND
G[20]	192.112.36.4[Notes 3]	2001:500:12::d0d	ns.nic.ddn.mil	Defense Information Systems Agency	trafic distribué par anycast	BIND
H[21]	198.97.190.53[22],[Notes 4]	2001:500:1::53	aos.arl.army.mil	United States Army Research Laboratory (en)	Aberdeen, Maryland, États-Unis	NSD
I[23]	192.36.148.17[24]	2001:7fe::53	nic.nordu.net	Autonomica (Netnod (en))	trafic distribué par anycast	BIND
J[25]	192.58.128.30[25],[Notes 5]	2001:503:c27::2:30		VeriSign	trafic distribué par anycast	BIND
K[27]	193.0.14.129[27]	2001:7fd::1		RIPE NCC	trafic distribué par anycast	BIND, NSD[27]
L[29]	199.7.83.42[Notes 6]	2001:500:3::42		ICANN	trafic distribué par anycast	NSD[29]
M[31]	202.12.27.33[32]	2001:dc3::35		WIDE Project (en)	trafic distribué par anycast	BIND

Serveur DNS

13 Serveur racine (source wikipedia)



gethostbyname

struct hostent *gethostbyname(const char *name);

The hostent structure is defined in <netdb.h> as follows:

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

#define h_addr h_addr_list[0] /* for backward compatibility */
```

The members of the hostent structure are:

h_name The official name of the host.

h_aliases
An array of alternative names for the host, terminated by a null pointer.

h_addrtype
The type of address; always AF_INET or AF_INET6 at present.

h_length
The length of the address in bytes.

h_addr_list
An array of pointers to network addresses for the host (in network byte order), terminated by a null pointer.

h_addr The first address in h_addr_list for backward compatibility.

gethostbyname

```c
#include <netdb.h>
#include <stdio.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>


int main(int argc, char **argv)
{
        if(argc < 2)
                return 1;

        struct hostent *ret = gethostbyname(argv[1]);

        if(!ret)
        {
                perror("gethostbyname");
                return 1;
        }


        printf("Host resolves to %s\n", ret->h_name);

        unsigned int i=0;
        while ( ret->h_addr_list[i] != NULL) {
                printf( "%s\n", inet_ntoa( *( struct in_addr*)( ret->h_addr_list[i])));
                i++;
        }


        return 0;
}
```

gethostbyname

```
$ ./a.out cnn.com
Host resolves to cnn.com
151.101.65.67
151.101.1.67
151.101.193.67
151.101.129.67
```

```
$ ./a.out facebook.com
Host resolves to facebook.com
157.240.21.35
```

```
$ ./a.out elysee.fr
Host resolves to elysee.fr
45.60.151.214
45.60.155.214
```

```
$ ./a.out www.uvsq.fr
Host resolves to preprod.uvsq.fr
193.51.33.8
```

UNIQUEMENT IPV4

getaddrinfo

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>

int getaddrinfo(const char *node,     // Domaine par exemple google.com
                const char *service,  // Port ou service (80 ou www par exemple)
                  const struct addrinfo *hints, // Hints souvent un memset 0 de la struct
                  struct addrinfo **res); // Valeur de retour
```

```
struct addrinfo {
    int              ai_flags;     // AI_PASSIVE, AI_CANONNAME, etc.
    int              ai_family;    // AF_INET, AF_INET6, AF_UNSPEC
    int              ai_socktype;  // SOCK_STREAM, SOCK_DGRAM
    int              ai_protocol;  // use 0 for "any"
    size_t           ai_addrlen;   // size of ai_addr in bytes
    struct sockaddr *ai_addr;      // struct sockaddr_in or _in6
    char            *ai_canonname; // full canonical hostname

    struct addrinfo *ai_next;      // linked list, next node
};
```

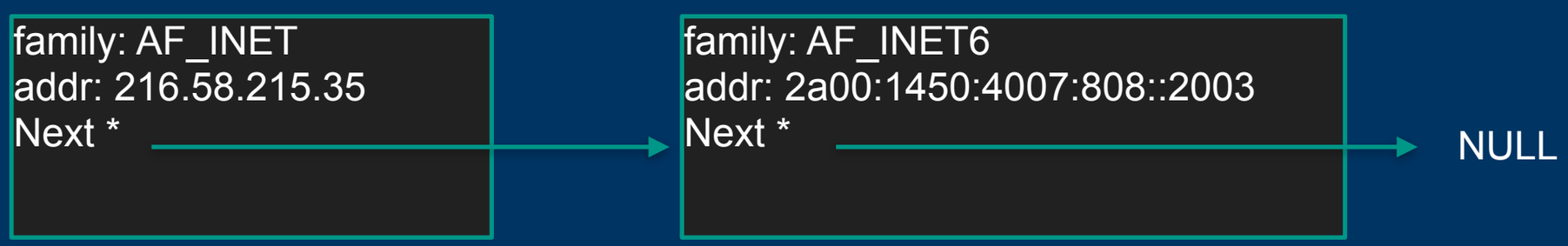
Convertir adresse DNS google.fr en:

IPV4 : 172.217.22.131

IPV6 : 2a00:1450:4007:815::2003

getaddrinfo

```
struct addrinfo {
    int              ai_flags;     // AI_PASSIVE, AI_CANONNAME, etc.
    int              ai_family;    // AF_INET, AF_INET6, AF_UNSPEC
    int              ai_socktype;  // SOCK_STREAM, SOCK_DGRAM
    int              ai_protocol;  // use 0 for "any"
    size_t           ai_addrlen;   // size of ai_addr in bytes
    struct sockaddr *ai_addr;      // struct sockaddr_in or _in6
    char            *ai_canonname; // full canonical hostname

    struct addrinfo *ai_next;      // linked list, next node
};
```



Convertir adresse DNS google.fr en:

IPV4 : 216.58.215.35

IPV6 : 2a00:1450:4007:808::2003

Résolution de Nom DNS

```c
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>
#include <arpa/inet.h>
#include <stdio.h>
#include <string.h>

int main( int argc, char ** argv ) {
    struct addrinfo *res = NULL;
    struct addrinfo hints;
    memset(&hints, 0, sizeof(hints));
    hints.ai_family = AF_UNSPEC;
    hints.ai_socktype = SOCK_STREAM;
    int ret = getaddrinfo(argv[1], argv[2],
            &hints,
            &res);
    if( ret < 0) {
        herror("getaddrinfo");
        return 1;
    }
    struct addrinfo *tmp;
    for (tmp = res; tmp != NULL; tmp = tmp->ai_next) {
        char ip[INET6_ADDRSTRLEN];
        ip[0] = '\0';
        if(tmp->ai_family == AF_INET) {
            struct sockaddr_in* saddr = (struct sockaddr_in*)tmp->ai_addr;
            inet_ntop(AF_INET, &saddr->sin_addr, ip, sizeof(ip));
            printf("IPV4 : %s\n", ip);
        }
        else if(tmp->ai_family == AF_INET6) {
            struct sockaddr_in6* saddr6 = (struct sockaddr_in6*)tmp->ai_addr;
            inet_ntop(AF_INET6, &saddr6->sin6_addr, ip, sizeof(ip));
            printf("IPV6 : %s\n", ip);
        }
    }
    return 0;
}
```

```c
hints.ai_family = AF_UNSPEC;
```

```
IPV4 : 172.217.22.131
IPV6 : 2a00:1450:4007:815::2003
```

```c
hints.ai_family = AF_INET;
```

```
IPV4 : 172.217.22.131
```

```c
hints.ai_family = AF_INET6;
```

```
IPV6 : 2a00:1450:4007:815::2003
```

getaddrinfo

```
$ ./getaddr cnn.com
IPV4 : 151.101.193.67
IPV4 : 151.101.129.67
IPV4 : 151.101.1.67
IPV4 : 151.101.65.67
IPV6 : 2a04:4e42:600::323
IPV6 : 2a04:4e42:400::323
IPV6 : 2a04:4e42::323
IPV6 : 2a04:4e42:200::323
```

```
$ ./getaddr google.com
IPV4 : 216.58.209.238
IPV6 : 2a00:1450:4007:812::200e
```

```
$ ./getaddr free.fr
IPV4 : 212.27.48.10
IPV6 : 2a01:e0c:1::1
```

```
$ ./getaddr facebook.com
IPV4 : 157.240.21.35
IPV6 : 2a03:2880:f130:83:face:b00c:0:25de
```

Créer un Socket

Créer un Socket TCP

Un socket est un descripteur de fichier correspondant à un flux réseau

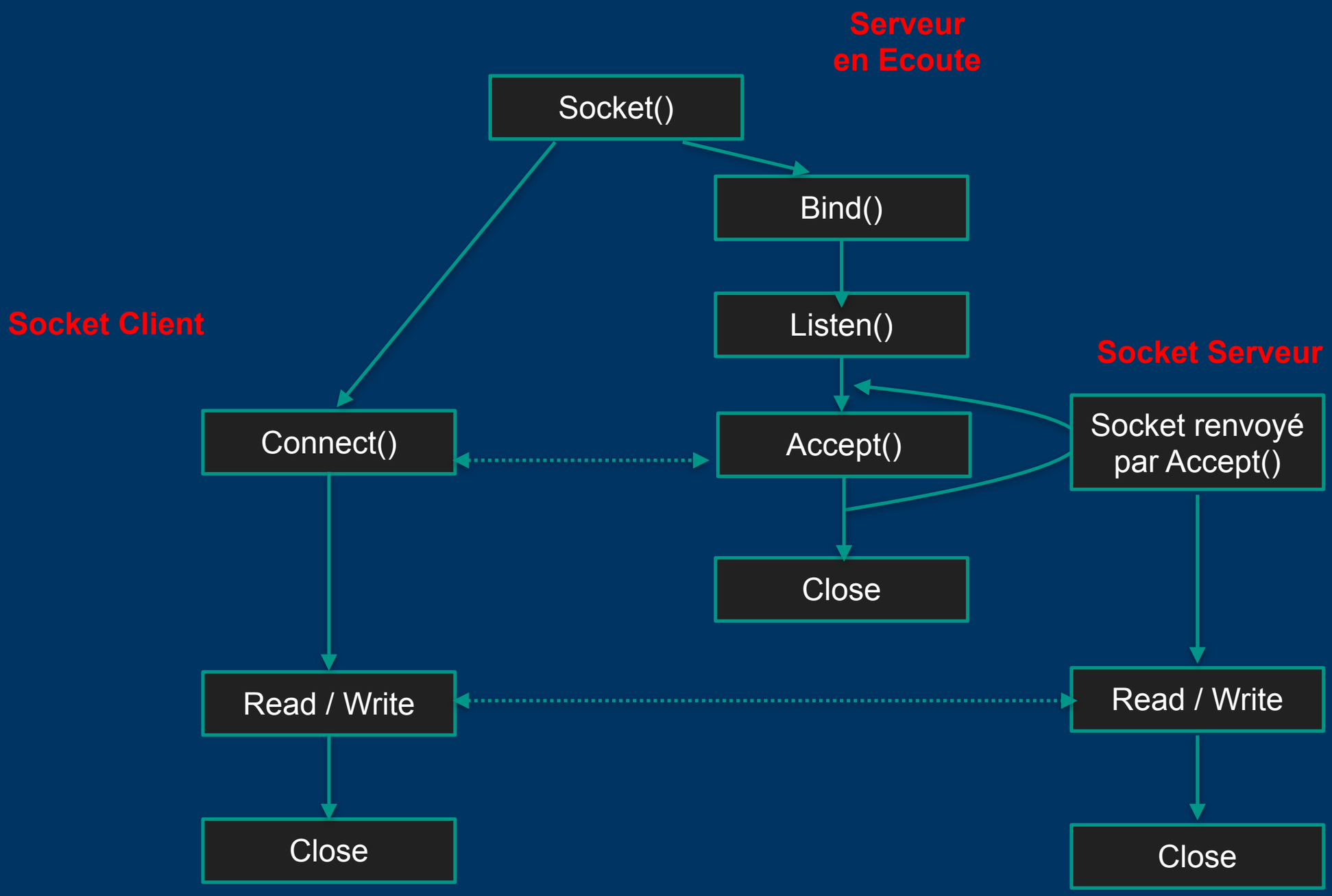
```
#include <sys/socket.h>

int socket(int domain, int type, int protocol);
```

- Domain -> type de socket
  - ➡ AF_UNIX -> socket UNIX reposant sur un fichier (socket local)
  - ➡ AF_INET -> socket IPV4
  - ➡ AF_INET6 -> socket IPV6

- Type -> Mode de communications
  - ➡ SOCK_STREAM -> flux de donnée TCP
  - ➡ SOCK_DGRAM -> datagramme (UDP)
  - ➡ SOCK_RAW -> accès bas niveau à l'interface (uniquement root)

- Protocol -> Protocole à utiliser
  - ➡ En général on laisse 0

Etats d’un Socket



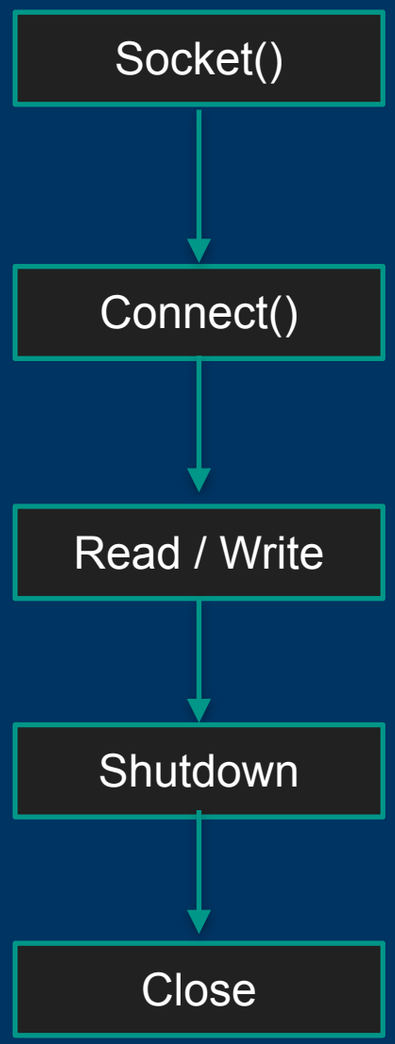
Connecter un Socket

Connect

```
NAME
       connect - initiate a connection on a socket

SYNOPSIS
       #include <sys/types.h>          /* See NOTES */
       #include <sys/socket.h>

       int connect(int sockfd, const struct sockaddr *addr, socklen_t addrlen);
```

Astuce : utiliser les valeurs de retour de gethostbyname !

Socket Client



Connect via Gethostbyname

```c
/* Argument check */
if ( argc != 3 )
{
        return 1;
}

/* Name resolution */
struct hostent *server_info = gethostbyname( argv[1] );

if ( !server_info )
{
        herror( "gethostbyname" );
        return 1;
}

/* Create Socket */
int sock = socket( AF_INET, SOCK_STREAM, 0 );

if ( sock < 0 )
{
        perror( "socket" );
        return 1;
}

/* Configure client socket */
struct sockaddr_in server_conf;
/* Insert address family */
server_conf.sin_family = server_info->h_addrtype;
/* Insert PORT */
server_conf.sin_port = htons( atoi( argv[2] ) );
/* Copy destination addr from server_info */
memcpy( &server_conf.sin_addr, server_info->h_addr_list[0], server_info->h_length );

/* Connect the socket to the server */
if ( connect( sock, ( struct sockaddr * )&server_conf, sizeof( struct sockaddr_in ) ) < 0 )
{
        perror( "Connect" );
        return 1;
}
```

Connect via getaddrinfo

```c
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>
#include <arpa/inet.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>

int main( int argc, char ** argv )
{
     struct addrinfo *res = NULL;
     struct addrinfo hints;
    memset(&hints, 0, sizeof(hints));
    hints.ai_family = AF_UNSPEC;
    hints.ai_socktype = SOCK_STREAM;

    int ret = getaddrinfo(argv[1], argv[2],
            &hints,
            &res);

    if( ret < 0)
    {
         herror("getaddrinfo");
         return 1;
    }
```

Suite du code ici …

```c
    struct addrinfo *tmp;
    int sock = -1;
    int connected = 0;

    for (tmp = res; tmp != NULL; tmp = tmp->ai_next) {
         sock = socket(tmp->ai_family,
                              tmp->ai_socktype,
                               tmp->ai_protocol);
         if( sock < 0) {
              perror("sock");
              continue;
         }
         int ret = connect( sock, tmp->ai_addr,
                              tmp->ai_addrlen);
              if( ret < 0 ) {
                   close(sock);
                   perror("connect");
                   continue;
              }
              connected = 1;
              break;
         }

         if(!connected) {
              fprintf(stderr, "Failed to connect to %s:%s\n »,
                              argv[1], argv[2]);
              return 1;
         }
         /* Use the socket */
         close(sock);
         return 0;
}
```

Créer un Serveur (Socket en écoute)

Bind & Listen

Attacher le socket à un port donné:

```
#include <sys/types.h>          /* See NOTES */
#include <sys/socket.h>

int bind(int sockfd, const struct sockaddr *addr,
         socklen_t addrlen);
```

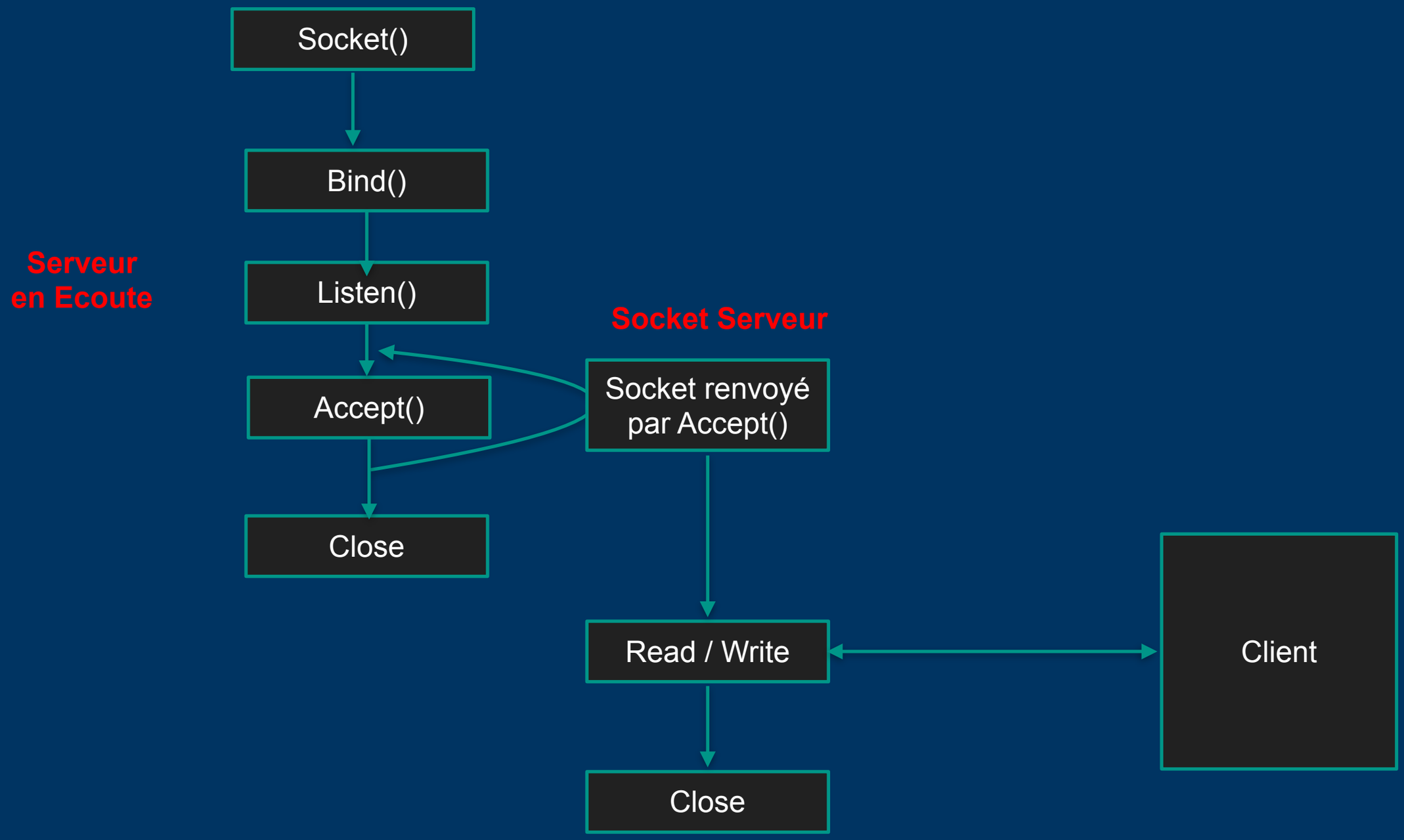
Se mettre en attente de connexions avec Listen:

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

 int listen(int sockfd, int backlog);
```

Accepter une connexion entrante:

```
#include <sys/types.h>          /* See NOTES */
#include <sys/socket.h>

int accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen);
```

Etats d’un Socket



Avec Getaddrinfo

```
struct addrinfo *res = NULL;
    struct addrinfo hints;
    memset(&hints, 0, sizeof(hints));

    hints.ai_family = AF_UNSPEC;
    hints.ai_socktype = SOCK_STREAM;
    hints.ai_flags = AI_PASSIVE;

    int ret = getaddrinfo(NULL, PORT, &hints, &res);
```

PORT : est un string avec soit, un numéro ou bien un descripteur de service par exemple « www ».

Comme pour connect, cet appel prépare pour différentes configuration les paramètre à bind.

Serveur

(Suite)

```c
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>
#include <arpa/inet.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>

int main( int argc, char ** argv ) {
      struct addrinfo *res = NULL;
      struct addrinfo hints;
      memset(&hints, 0, sizeof(hints));
      hints.ai_family = AF_UNSPEC;
      hints.ai_socktype = SOCK_STREAM;
      hints.ai_flags = AI_PASSIVE;

      if(argc != 2)
            return 1;

      int ret = getaddrinfo(NULL, argv[1], &hints, &res);

      if( ret < 0) {
            herror("getaddrinfo");
            return 1;
      }

      struct addrinfo *tmp;
      int listen_sock = -1;
      int binded = 0;

      for (tmp = res; tmp != NULL; tmp = tmp->ai_next) {
            listen_sock = socket(tmp->ai_family,
                                 tmp->ai_socktype,
                                 tmp->ai_protocol);
            if( listen_sock < 0) {
                  perror("sock");
                  continue;
            }

            ret = bind( listen_sock, tmp->ai_addr, tmp->ai_addrlen);

            if( ret < 0 ) {
                  close(listen_sock);
                  perror("bind");
                  continue;
            }
            binded = 1;
      }

      if(!binded)
      {
            fprintf(stderr, "Failed to bind on 0.0.0.0:%s\n", argv[1]);
            return 1;
      }

      /* Start listening */
      ret = listen(listen_sock, 2);

      if( ret < 0)
      {
            perror("listen");
            return 1;
      }


      /* Now accept one connection */
      struct sockaddr client_info;
      socklen_t addr_len;
      fprintf(stderr,"Before accept\n");
      int client_socket = accept(listen_sock, &client_info, &addr_len);
      fprintf(stderr,"After accept\n");

      if( client_socket < 0)
      {
            perror("accept");
            return 1;
      }

      fprintf(stderr,"Closing client socket\n");
      close(client_socket);


      close(listen_sock);

      return 0;
}
```

Serveur

Paramétrage serveur

Démarrage du serveur

Création Socket

Attache adresse au Socket

Accueil des clients

```c
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>
#include <arpa/inet.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>

int main( int argc, char ** argv ) {
	struct addrinfo *res = NULL;
	struct addrinfo hints;
	memset(&hints, 0, sizeof(hints));
	hints.ai_family = AF_UNSPEC;
	hints.ai_socktype = SOCK_STREAM;
	hints.ai_flags = AI_PASSIVE;

	if(argc != 2)
		return 1;

	int ret = getaddrinfo(NULL, argv[1], &hints, &res);

	if( ret < 0) {
		herror("getaddrinfo");
		return 1;
	}

	struct addrinfo *tmp;
	int listen_sock = -1;
	int binded = 0;

	for (tmp = res; tmp != NULL; tmp = tmp->ai_next) {
		listen_sock = socket(tmp->ai_family,
		                     tmp->ai_socktype,
		                     tmp->ai_protocol);

		if( listen_sock < 0) {
			perror("sock");
			continue;
		}

		ret = bind( listen_sock, tmp->ai_addr, tmp->ai_addrlen);

		if( ret < 0 ) {
			close(listen_sock);
			perror("bind");
			continue;
		}
```

```c
		if(!binded)
		{
			fprintf(stderr, "Failed to bind on 0.0.0.0:%s\n", argv[1]);
			return 1;
		}

		/* Start listening */
		ret = listen(listen_sock, 2);

		if( ret < 0)
		{
			perror("listen");
			return 1;
		}


		/* Now accept one connection */
		struct sockaddr client_info;
		socklen_t addr_len;
		fprintf(stderr,"Before accept\n");
		int client_socket = accept(listen_sock, &client_info, &addr_len);
		fprintf(stderr,"After accept\n");

		if( client_socket < 0)
		{
			perror("accept");
			return 1;
		}

		fprintf(stderr,"Closing client socket\n");
		close(client_socket);


		close(listen_sock);

		return 0;
}
```

Parallélisme de Serveur

Serveur Multi-Clients

Serveur en Ecoute

Comment gérer ces sockets ?

Serveur qui « Fork »

```c
while ( 1 )
{
        /* On accepte un client et on récupére un nouveau FD */
        int client_socket = accept( listen_sock, &client_info, &addr_len );
        fprintf( stderr, "After accept\n" );

        if ( client_socket < 0 )
        {
                perror( "accept" );
                return 1;
        }

        pid_t c = fork();

        if ( !c )
        {
                int j;

                for ( j = 0 ; j < 128 ; j++ )
                {
                        char message[128];
                        snprintf( message, 128, "Salut %d\n", j );
                        /* On salut a celui qui s'est connecté */
                        write( client_socket, message, strlen( message ) );
                        sleep( 1 );
                }

                fprintf( stderr, "Closing client socket\n" );
                /* On se déconnecte du client */
                close( client_socket );
        }
        else
        {
                close( client_socket );
        }
}
```

Serveur multi-thread

```c
while ( 1 )
{
        /* On accepte un client et on récupére un nouveau FD */
        int client_socket = accept( listen_sock, &client_info, &addr_len );
        fprintf( stderr, "After accept\n" );

        if ( client_socket < 0 )
        {
                perror( "accept" );
                return 1;
        }

        struct client_infos *infos = ( struct client_infos * )malloc( sizeof( struct client_infos ) );

        infos->client_socket = client_socket;

        pthread_t th;

        pthread_create( &th, NULL, client_loop, ( void * ) infos );
}
```

Appel Select

```c
/* According to POSIX.1-2001, POSIX.1-2008 */
#include <sys/select.h>

/* According to earlier standards */
#include <sys/time.h>
#include <sys/types.h>
#include <unistd.h>

int select(int nfds, /* Nombre de FDs */
           fd_set *readfds, /* Sortie: FD prêts en lecture */
           fd_set *writefds, /* Sortie: FD prêts en écriture */
           fd_set *exceptfds, /* Sortie FD eb erreur */
           struct timeval *timeout); /* Timeout optionnel */
```

Ensemble de FDs:

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

Appel Select

Le but est de déléguer le blocage au noyau (au lieu de le faire en espace utilisateur). De plus on peut alors avoir une seule boucle répondant aux différents évènements aussi bien en lecture qu'en écriture.

Select manipule des ensemble de descripteurs de fichier:

Appel	Description
FD_ZERO	Initialise un set à 0
FD_SET	Ajoute au set
FD_CLR	Retire du set
FD_ISSET	Test de présence

L'assignation de sets est possible.

```c
void fdsetprint(fd_set * set)
{
        int i;


        printf("=================\n");
        for( i = 0 ; i < FD_SETSIZE; i++)
        {
                if( FD_ISSET(i, set))
                {
                        printf("%d is in set\n", i);
                }

        }
        printf("=================\n");

}
```

```c
int main(int argc, char ** argv )
{
        fd_set set;

        FD_ZERO(&set);

        FD_SET(19, &set);
        FD_SET(20, &set);
        fdsetprint(&set);

        FD_CLR(19, &set);
        fdsetprint(&set);

        fd_set second = set;
        fdsetprint(&second);

        return 0;
}
```

Serveur multiplexé

```c
fd_set active_fd_set, read_fd_set;
/* Initialize the set of active sockets. */
FD_ZERO ( &active_fd_set );
FD_SET ( listen_sock, &active_fd_set );

while ( 1 )
{
        /* Block until input arrives on one or more active sockets. */
        read_fd_set = active_fd_set;

        if ( select ( FD_SETSIZE, &read_fd_set, NULL, NULL, NULL ) < 0 )
        {
                perror ( "select" );
                exit ( EXIT_FAILURE );
        }

        int i;

        /* Service all the sockets with input pending. */
        for ( i = 0; i < FD_SETSIZE; ++i )
                if ( FD_ISSET ( i, &read_fd_set ) )
                {
                        if ( i == listen_sock )
                        {
                                /* Event was on the listen Socket */
                                struct sockaddr_in client_info;
                                /* Connection request on original socket. */
                                int new;
                                unsigned int addr_size = sizeof ( struct sockaddr_in );
                                new = accept ( listen_sock, ( struct sockaddr * ) &client_info,  &addr_size );

                                if ( new < 0 )
                                {
                                        perror ( "accept" );
                                        exit ( EXIT_FAILURE );
                                }

                                fprintf ( stderr,
                                          "Server: connect from host %s, port %hd.\n",
                                          inet_ntoa ( client_info.sin_addr ),
                                          ntohs ( client_info.sin_port ) );
                                FD_SET ( new, &active_fd_set );
                        }
                        else
                        {

                                /* Data arriving on an already-connected socket. */
                                if ( read_from_client ( i ) < 0 )
                                {
                                        fprintf( stderr, "Client left\n" );
                                        close ( i );
                                        FD_CLR ( i, &active_fd_set );
                                }
                        }
                }
}
```

Socket UNIX

Socket UNIX

- Classiquement un socket est identifié par un couple

HOST:PORT

- Cependant, il existe des sockets associés à un fichier ils sont donc locaux à un système et soumis aux droits standards sur les fichiers.

/tmp/my.sock (par exemple)

```c
/* UNIX socket descriptor */
struct sockaddr_un addr;
/* Clear it */
memset( &addr, 0, sizeof( addr ) );
/* Set family to UNIX */
addr.sun_family = AF_UNIX
                  /* Set socket PATH */;
strncpy( addr.sun_path, argv[1], sizeof( addr.sun_path ) - 1 );
/* Create socket FD */
int listen_socket = socket( AF_UNIX, SOCK_STREAM, 0 );

if ( listen_socket < 0 )
{
        perror( "socket" );
        return 1;
}

/* BIND the socket to UNIX socket */
int ret = bind( listen_socket, ( struct sockaddr * )&addr, sizeof( addr ) );

if ( ret < 0 )
{
        perror( "bind" );
        fprintf( stderr, "Failed to bind on 0.0.0.0:%s\n", argv[1] );
        return 1;
}

/* On commence a ecouter */
ret = listen( listen_socket, 2 );

if ( ret < 0 )
{
        perror( "listen" );
        return 1;
}
```

Socket UNIX

Socker UNIX

$ ./server /tmp/test.sock

ATTENTION: toutes les version ne netcat ne supportent pas les sockets UNIX il vous faut une version qui comprend le flag -
ici on propose d’utiliser Socat.

$ socat - UNIX-CONNECT:/tmp/test.sock
SALUT!

Protocole de Base

HTTP

Requête GET

Operation

Page demandée

Standard HTTP

```
GET / HTTP/1.1
Host: deneb.france.paratools.com:8080
Connection: keep-alive
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9,fr-FR;q=0.8,fr;q=0.7
Cookie: doxygen_width=300
```

(OPTIONNEL)

Réponse Serveur

Operation

Page demandée

Standard HTTP

GET / HTTP/1.1

Ligne VIDE

Réponse OK → **HTTP/1.0 200 OK**

Type de contenu → **Content-Type: text/html; charset=ISO-8859-1**

Longueur du contenu → Content-Length: 15

Ligne VIDE

Contenu → <h1>Hello</h1>

```c
void *client_loop( void *param )
{
	struct client_infos *info = ( struct client_infos * )param;

	char buffer[1024];

	FILE* socket = fdopen(info->client_socket, "rw+");

	if(!socket)
	{
		perror("fdopen");
		exit(1);
	}


	while(fgets(buffer, 1024, socket))
	{
		if(strlen(buffer) < 6)
		{
			continue;
		}

		char *http = strstr(buffer, " HTTP");

		if(http)
		{
			*http = '\0';
		}

		if( buffer[0] == 'G' && buffer[1] == 'E' && buffer[2] == 'T')
		{
			char * file_path = &buffer[4];

			printf("GET == '%s'\n", file_path);

			if(*file_path == '/' && strlen(file_path) == 1)
			{
				file_path="index.html";
			}
			else
			{
				/* Skip / */
				file_path++;
			}

			ssize_t content_size = get_file_size(file_path);

			if(0 < content_size )
			{
				write_http_header(content_size, 200, "text/html", info->client_socket);
				sendfile(file_path, content_size, info->client_socket);
				break;
			}
			else
			{
				write_http_header(0, 404, "text/html", info->client_socket);
			}
		}
	}


	fprintf( stderr, "Closing client socket\n" );
	/* On se déconnecte du client */
	fclose( socket );
}
```

Serveur HTTP Ultra-Basique

Faire Son Protocole

Approche Textuelle

Commande [PARAM] [SEPARATEUR]

Exemple:
GET TOTO\n

Format de données structurées

Exemple JSON:
{‘command’:’get’, ‘param’:’lol’}

Approche Binaire

```c
typedef enum
{
    MSG_METRIC_MSG_DESC=0,    /**< Register a new metric IN:
MSG_metric_descriptor_t */
    MSG_METRIC_MSG_VAL=1,      /**< Send a metric value IN: MSG_metric_event_t*/
}MSG_metric_msg_type_t;

static const char * const MSG_metric_msg_type_name[] =
{
    "MSG_METRIC_MSG_DESC",
    "MSG_METRIC_MSG_VAL",
};

typedef struct {
    char name[METRIC_STRING_SIZE];
    char doc[METRIC_STRING_SIZE];
    int type;
}MSG_metric_descriptor_t;

typedef struct {
    char name[METRIC_STRING_SIZE];
    double value;
    double update_ts;
}MSG_metric_event_t;

typedef struct {
    MSG_metric_msg_type_t type;
    union {
        MSG_metric_descriptor_t desc;
        MSG_metric_event_t event;
    }payload;
    char canary;
}MSG_metric_msg_t;
```