TELECOM ParisTech

Cours Réseaux

PARTIE II

Protocoles Internet

Pascal Urien

2011

1. Internet Protocol

1.1 IP v4

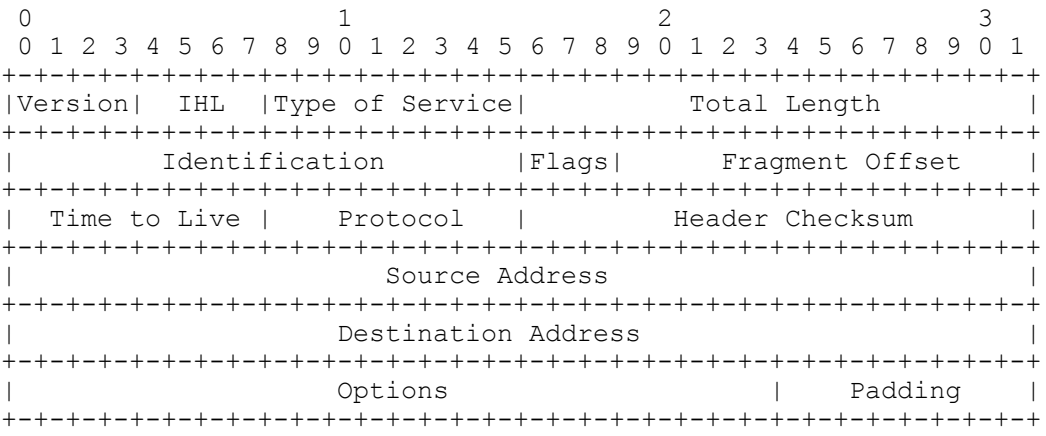
La RFC 791 constitue la spécification officielle du protocole IP. La topologie IP est constituée par des **réseaux**, chacun d'eux possédant un certain nombre d'éléments. Les différents réseaux sont reliés par des « gateways » (passerelles). ***Une passerelle comporte une couche IP associée au protocole ICMP***.

Les principales caractéristiques de IP sont *:*

- Protocole à datagrammes indépendants, les éléments du réseau peuvent détruire ces datagrammes lorsqu'ils manquent de ressources.
- Pas de correction de perte des datagrammes
- Segmentation et réassemblage des datagrammes
- Pas de garantie d'acheminement des datagrammes
- Pas de garantie du maintien de remise en ordre des datagrammes
- Destruction des datagrammes qui ont séjourné trop longtemps dans le réseau.

La qualité remarquable du réseau IP est sa capacité de reconfiguration dynamique.

1.1.1 Les PDU IPv4



- **VER**: 4 bits qui indique la version 4
- **IHL**: 4 bits longueur de l'en tête en mots de 32 bits (de 5 à 15, soit de 20 à 60 octets).
- **TOS**: 8 bits type de service b0 b1 b2 b3 b4 b5 b6 b7
  b0 b1 b2 priorité
  111 contrôle du réseau
  110 contrôle inter réseaux
  101 CRITIC/ECP
  100 flash prioritaire
  011 immédiat
  001 prioritaire
  000 routine
- **TL**: 16 bits taille, longueur totale du datagramme (en tête et données inclus), une valeur usuelle est 576
- **ID**: 16 bits, identificateur du datagramme lors de son envoi. Ce champ est conservé lors d'une fragmentation.
- FLAGS: 3 bits (b0 b1 b2)
  b0 - **RFU** - codé à 0
  b1 - **DF** don't fragment
  b2 - **MF** fragment, fragment d'un datagramme segmenté.
- **FO**: 13 bits Fragment Offset, position relative (en unité de 8 octets) des données d'un segment rapport au datagramme de référence. Seul un datagramme non segmenté (**MF=0**) peut avoir la valeur de ce champ à zéro.

- **Exemple**: soit un réseau de taille maximale 1000 (MTU), une passerelle le relie à un réseau de MTU 200.
  - Le datagramme (IHL=5,DF=0,TL=450,F0=0) est segmenté en trois datagrammes
  - IHL=5,DF=0,MF=1,TL=196,F0=0
  - IHL=5,DF=0,MF=1,TL=196,F0=22=176/8
  - IHL=5,DF=0,MF=0,TL=98,F0=44=176*2/8
- Le réassemblage utilise les champs FO,DF,MF. Une couche IP détecte une perte de fragment par l'expiration d'un chien de garde. La perte d'un fragment implique la perte de la totalité du datagramme.

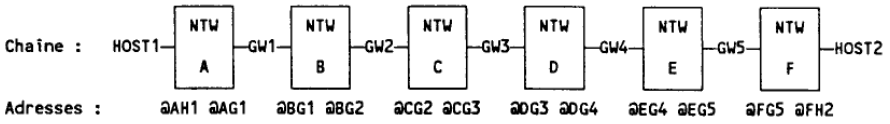
- **TTL** 8 bits une valeur de 0 à 255, cette valeur est décrémentée à chaque traversée de passerelle. tout intermédiaire ou destinataire qui détecte le **passage à zéro** de cette valeur écarte le datagramme et renvoit à l'expéditeur un datagramme ICMP contenant notamment l'en tête du datagramme détruit.
- **PROT** 8 bits, ce champ identifie le protocole encapsulé par IP
  - 0x01 ICMP, 0x02 IGMP, 0x03 GGP, 0x05 ST, 0x06 TCP, 0x08 EGP, 0x09 IGP, 0x0B NVP, 0x0C PUP, 0x11 UDP, 0x14 HMP, 0x16 XNS-IDP, 0x1B RDP, 0x1C IRTP, 0x1D ISO-TP4, 0x1E NETBLT.
- **Checksum** 16 bits. La somme de contrôle porte sur l'en tête IP uniquement et non sur les données. Les mots de l'en tête IP sont regroupés en mot de 16 bits dont on effectue la somme en modulo 65535 (2^{16}-1). Le checksum est égal au complément à un du résultat final.

  Exemple
  45 00 00 54
  88 FE 00 00
  FF 01 xx xx
  81 68 FE 06
  81 68 FE 05
  Somme totale : CF32, Cheksum 30CD
- **SOURCE** 32 bits, adresse de source du paquet IP.
- **DESTINATION** 32 bits, Adresse de destination du paquet IP.
- **OPTION**(s), la taille des options varie de 0 à 40 octets. Ce champ est constitué par une liste d'options élémentaires. Une option est identifiée soit par un ***octet unique*** (lorsque la longueur des données associées est fixe) soit par ***un octet*** (n° option), ***un octet de longueur*** (la longueur en octets soit 2 + longueur de données), et des ***données***. Le champ option doit être complété de telle sorte que sa longueur soit un multiple de 32 bits.
  - Une option se décompose en une classe (2 bit) et un numéro (5 bits), un bit CF définit la possibilité de recopier l'option dans des datagrammes fragmentés différents de l'en tête. Il existe une classe de contrôle (10 options) et une classe de debug et mesure (1 option)

b0	b1 b2	b3 b4 b5 b6 b7
CF	classe	numéro

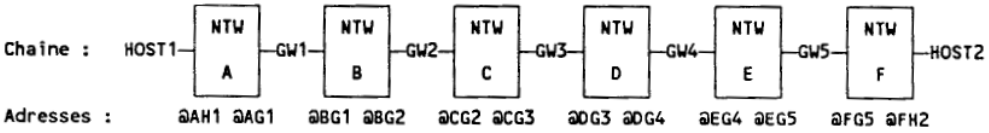
Quelques options:

- ***Classe 0, option n°2***, format de longueur variable sécurité, codage des données (9 octets)
- ***Classe 0, option n°3***, format de longueur variable, routage approximatif. Un octet est utilisé comme pointeur d'adresse (les valeurs possibles sont 4, 8, C ...).La route à suivre est représentée par une liste d'adresse IP (option non copiée en cas de fragmentation CF=0).Ce routage est approximatif car une passerelle n'est pas tenue de suivre ces indications. La taille d'un en tête IP limite la route ainsi définie à 9 passerelles.
- ***Classe 2, option n°4***, format de longueur variable, enregistrement d'instants. L'option comporte un octet «flags» un octet pointeur (de 5 à ...) et octet qui compte le nombre de passerelles qui n'ont pu copier leurs informations faute de place L'octet flag indique la nature de l'information à enregistrer (adresse 4 octets, et/ou temps 4 octets ms depuis minuit GMT, des adresses peuvent être spécifiées).
- ***Classe 0, option n°7***, format de longueur variable, enregistrement de route. Le format de cette option est identique à celui de *routage approximatif.*
- ***Classe 0, option n°9***, format de longueur variable routage impératif. Le format de cette option est identique à celui de routage *approximatif*.




```
Datagramme émis par HOST1 et reçu par GW1 :  SOURCE = @AH1
                                             DEST   = @FH2
                                             OPTION = 0x03,0x0F,0x04,@AG1,@CG3,@DG4

Datagramme émis par GW1   et reçu par GW2 :  SOURCE = @AH1
                                             DEST   = @FH2
                                             OPTION = 0x03,0x0F,0x08,@BG1,@CG3,@DG4

Datagramme émis par GW2   et reçu par GW3 :  SOURCE = @AH1
                                             DEST   = @FH2
                                             OPTION = 0x03,0x0F,0x8,@BG1,@CG3,@DG4

Datagramme émis par GW3   et reçu par GW4 :  SOURCE = @AH1
                                             DEST   = @FH2
                                             OPTION = 0x03,0x0F,0x0C,@BG1,@DG3,@DG4

Datagramme émis par GW4   et reçu par GW5 :  SOURCE = @AH1
                                             DEST   = @FH2
                                             OPTION = 0x03,0x0F,0x10,@BG1,@DG3,@EG4

Datagramme émis par GW5 et reçu par HOST2 :  SOURCE = @AH1
                                             DEST   = @FH2
                                             OPTION = 0x03,0x0F,0x10,@BG1,@DG3,@EG4
```

Option de routage Approximatif

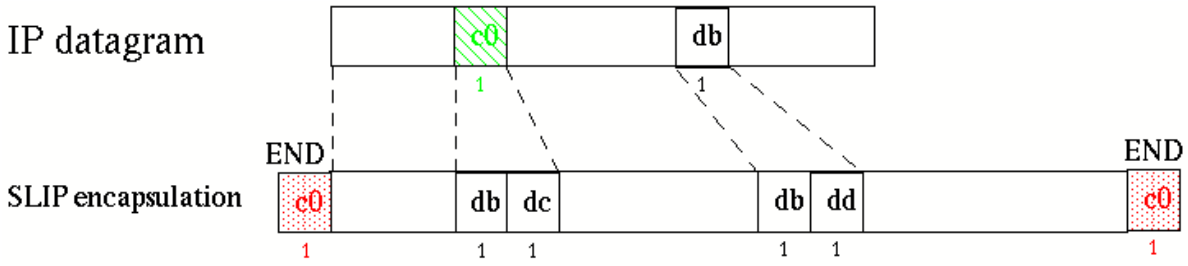



```
Datagramme émis par HOST1 et reçu par GW1 :  SOURCE = @AH1
                                             DEST   = @FH2
                                             OPTION = 0x44,0x10,0x05,0x00,0,0,0

Datagramme émis par GW1   et reçu par GW2 :  SOURCE = @AH1
                                             DEST   = @FH2
                                             OPTION = 0x44,0x10,0x09,0x00,T1,0,0

Datagramme émis par GW2   et reçu par GW3 :  SOURCE = @AH1
                                             DEST   = @FH2
                                             OPTION = 0x44,0x10,0x0D,0x00,T1,T2,0

Datagramme émis par GW3   et reçu par GW4 :  SOURCE = @AH1
                                             DEST   = @FH2
                                             OPTION = 0x44,0x10,0x11,0x00,T1,T2,T3

Datagramme émis par GW4   et reçu par GW5 :  SOURCE = @AH1
                                             DEST   = @FH2
                                             OPTION = 0x44,0x10,0x11,0x10,T1,T2,T3

Datagramme émis par GW5 et reçu par HOST2 :  SOURCE = @AH1
                                             DEST   = @FH2
                                             OPTION = 0x44,0x10,0x11,0x20,T1,T2,T3
```

Option Enregistrement d'Instant

1.1.2 Serial Line IP - RFC 1055

Ce protocole permet de transporter IP sur une liaison série dont le débit est généralement compris entre 1200 bps et 19.2 Kbps. Le protocole utilise deux caractères spéciaux END (192 en décimal) et ESC (219 en décimal). Un octet de donné dont la valeur est égale à END (192) est codé par deux octets ESC et ESC_END (219, 220). Un octet de donné dont la valeur est égale à ESC est codé par deux octets ESC et ESC_ESC (219, 221). Un paquet IP ne comporte pas de *start delimiter*, l'octet END notifie simplement la fin d'un paquet. Il faut également remarquer que le paquet ne comporte pas de CRC, les erreurs de transmissions ne sont donc pas détectées.



1.1.3 Compressed SLIP (CSLIP - RFC 1144)

La RFC 1144, *Compressing TCP/IP headers for low speed serial links*, Jacobson 1990 décrit une méthode de compression de l'en tête TCPIP (20+20 octets en règle générale). Les 40 octets typique de l'en tête sont réduit à une taille comprise en 3 et 5 octets.

1.1.4 Point to Point Protocol (PPP - RFC 1661)

flag 0x7E	address 0xFF	control 03	Protocol 2 octets	information 1500 octets max	CRC 2 octets	flag 0x7E

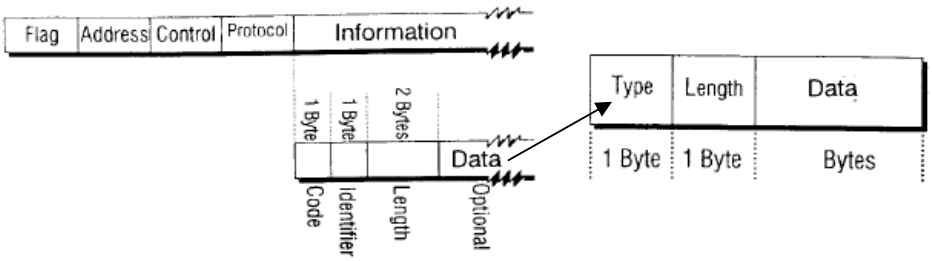
La trame PPP utilise en fait le format des trames HDLC (ISO 3309)

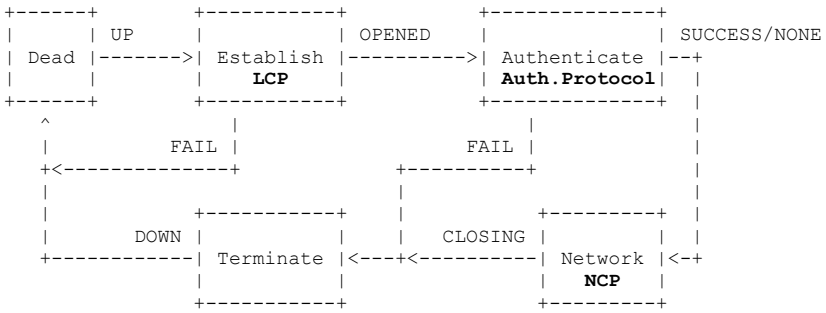
La structure de la trame est la suivante :

0x7E (Flag) 0xFF (Adresse) 0x03 (Contrôle) [PPP PDU(1500 octets max)] CRC (2 octets) 0x7E (Flag).

Un champ *Protocol* (2 octets) identifie le type de PDU PPP inclus dans la trame HDLC.

- 0x0021 : IP
- 0xC021 : Link Control Protocol (LCP)
- 0x8021 : Network Control Protocol (NCP)





Machine d'Etats du protocole PPP

1.1.4.1 LCP le protocole de contrôle de liaison

Ce protocole permet de négocier les options mises en œuvre par PPP (la taille des MTU par exemple). Il existe 11 types de paquets LCP identifiés par un code (8 bits). Les options sont encodées sous la forme Type (1 octet) Longueur (2 octets) valeur.

1.1.4.1.1 Exemple de PDUs LCP

- (1) Requête de configuration
- (2) Accusé (ACK) de configuration
- (3) Non accusé de configuration (NAK)
- (4) Requête de terminaison
- (6) Accusé de terminaison
- (7) Rejet de code
- (8) Rejet de protocole
- (9) Requête d'écho
- (10) réponse d'écho
- (11) Requête d'élimination

1.1.4.1.2 Les options de configuration de LCP

- (1) MRU Maximum Receive Unit
- (2) Protocole d'authentification
- (3) Protocole de qualité - mesure de la qualité de la ligne utilisée.
- (4) Nombre magique - détection de boucles (client et serveur sur le même système hôte).
- (5) Compression du champ protocole (de 2 octets à 1 - PFC)
- (6) Compression des champs adresse et contrôle de la trame HDLC (ACFC).

1.1.4.2 Le protocole de contrôle réseau (NCP)

Ce protocole permet de configurer des types de réseaux différents par exemple IP ou DECnet.

PPP Internet Protocol Control (IPCP) RFC 1332

Le paquet IP est encapsulé dans une trame PPP, le numéro du protocole est 0x8021. IPCP utilise seulement les 7 premiers types de PDUs de LCP.

La compression de Van Jacobson peut être utilisée (dans ce cas le champ protocole 0x002D sera utilisé dans les trames PPP).

Une adresse IP peut être attribuée par le serveur au client.

1.1.4.3 Un exemple de traces PPP

```
PPP[C021] state = starting
PPP[C023] state = starting
PPP[8021] state = starting

PPP[C021] SND CONFREQ
ID=01 LEN=24 MRU(0100) ACCM(00000000) MAGIC(0005E782) PFC ACFC
SND[C021] 0000:01 01 00 18 01 04 01 00 02 06 00 00 00 00 05 06 00 05 E7 82
SND[C021] 0014:07 02 08 02

PPP[C021] state = reqsent

PPP frame check error, 23

PPP[C021] RCV CONFREQ
ID=00 LEN=24 MRU(05DC) ACCM(00000000) MAGIC(28B03580) PFC ACFC
RCV[C021] 0000:01 00 00 18 01 04 05 DC 02 06 00 00 00 00 05 06 28 B0 35 80
RCV[C021] 0014:07 02 08 02

PPP[C021] SND CONFACK
ID=00 LEN=24 MRU(05DC) ACCM(00000000) MAGIC(28B03580) PFC ACFC
SND[C021] 0000:02 00 00 18 01 04 05 DC 02 06 00 00 00 00 05 06 28 B0 35 80
SND[C021] 0014:07 02 08 02

PPP[C021] state = acksent
```

PPP[C021] RCV CONFACK
ID=01 LEN=24 MRU(0100) ACCM(00000000) MAGIC(0005E782) PFC ACFC
RCV[C021] 0000:02 01 00 18 01 04 01 00 02 06 00 00 00 00 05 06 00 05 E7 82
RCV[C021] 0014:07 02 08 02

PPP[C021] state = opened

PPP[8021] SND CONFREQ ID=01 LEN=16 IPCP(002D0F01) IPADDR(81B63301)
SND[8021] 0000:01 01 00 10 02 06 00 2D 0F 01 03 06 81 B6 33 01

PPP[8021] state = reqsent

PPP[8021] RCV CONFREQ ID=01 LEN=10 IPCP(002D0F01)
RCV[8021] 0000:01 01 00 0A 02 06 00 2D 0F 01

PPP[8021] SND CONFACK ID=01 LEN=10 IPCP(002D0F01)
SND[8021] 0000:02 01 00 0A 02 06 00 2D 0F 01

PPP[8021] state = acksent

RCV[8021] 0000:03 01 00 0A 03 06 C2 02 91 48
PPP[8021] RCV CONFNAK ID=01 LEN=10 IPADDR(C2029148)

My IP address = 194.2.145.72

PPP[8021] SND CONFREQ ID=02 LEN=16 IPCP(002D0F01) IPADDR(C2029148)
SND[8021] 0000:01 02 00 10 02 06 00 2D 0F 01 03 06 C2 02 91 48

RCV[8021] 0000:02 02 00 10 02 06 00 2D 0F 01 03 06 C2 02 91 48
PPP[8021] RCV CONFACK ID=02 LEN=16 IPCP(002D0F01) IPADDR(C2029148)

PPP[8021] state = opened

(ICMP.request == ping(Data = "0123456789")
SND[0021] 0000:45 00 00 28 00 02 00 00 3C 01 74 E9 C2 02 91 48 81 B6 34 E9
SND[0021] 0014:08 00 31 35 01 00 C0 C0 30 31 32 33 34 35 36 37 38 39 00 00
(ICMP.response)
RCV[0021] 0000:45 00 00 28 3F 87 00 00 EB 01 86 63 81 B6 34 E9 C2 02 91 48
RCV[0021] 0014:00 00 39 35 01 00 C0 C0 30 31 32 33 34 35 36 37 38 39 00 00

1.2 Address Resolution Protocol ARP RFC 826

Ce protocole procure une correspondance entre une adresse IP de 32 bits et une adresse MAC d'un réseau physique particulier. Le proxy ARP consiste dans le fait qu'une passerelle réponde à un ARP par sa propre adresse MAC dans la mesure ou elle est capable de router le paquet vers son destinataire.

Le protocole RARP permet d'établir la correspondance entre une adresse IP et une adresse MAC , il est utilisé par exemple par des stations sans disques qui ne connaissent pas leur adresse IP lors de leur mise en tension.

Un paquet ARP est identifié dans ethernet par le mot (PID) de 16 bits 0x0806, RARP par 0x8035.

Un paquet ARP ou RARP comporte les informations suivantes :

Hardware Code (2 octet)	l'identification du type d'adresse MAC 0001= ethernet
Protocol Code (2 octets)	0800 = IP
HLEN 1 octet	longueur en octets des adresses MAC (6 octets pour l'IEEE)
PLEN 1 octet	longueur en octets des adresses protocolaires (4 pour @IP).
Datagram code 2 octets	0001 ARP_REQ, 0002=ARP_RESP, 0003=RARP_REQ, 0004=RARP_RESP.

Sender @Macsap	adresse MAC du demandeur
Sender @Prot	adresse protocolaire du demandeur
Target @Macsap	adresse mac de la cible
Target @Prot	adresse protocolaire de la cible.

Les trames ARP sont émises en diffusion dans le cas des réseaux locaux usuels (802.3, FDDI ...).

Lorsque la notion de diffusion n'existe pas on introduit la notion de ***serveur d'ARP***.

Exemple

```
FF FF FF FF FF FF 08 00 20 02 45 9E 08 06 00 01 08 00 06 04 00
01 08 00 20 02 45 9E 81 68 FE 06 00 00 00 00 00 00 81 68 FE 05

08 00 20 02 45 9E 08 00 20 07 0B 94 08 06 00 01 08 00 06 04 00
02 08 00 20 07 0B 94 81 68 FE 05 08 00 20 02 45 9E 81 68 FE 06
```

1.3 Internet Control Message Protocol ICMP RFC 792

Le protocole ICMP est un protocole administratif permettant à des équipements de gérer différents services de réseau, ou d'informer un émetteur de datagramme IP de problèmes rencontrés

1.3.1 Les PDU ICMP

Un en tête ICMP comporte 4 octets

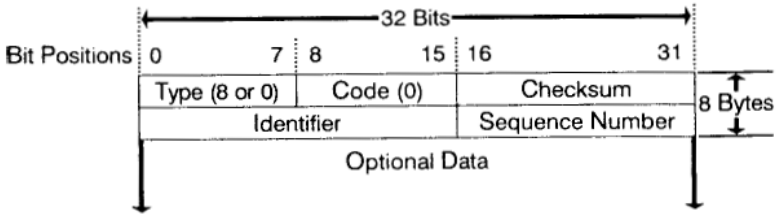
- type (1 octet) identifie le type de PDU
- code (1 octet) un code associé à certains types (n°erreur ou n°info)
- checksum (2 octets) le complément à un de la somme modulo 65535 (2^{16}-1) du contenu du paquet regroupé en mots de 16 bits.

Type	code	checksum

1.3.1.1 ICMP_ECHO_REQ (type=0x08) et ICMP_ECHO_RESP(type=0x00)

Ces trames sont associées au célèbre utilitaire ping qui teste la présence d'une adresse IP sur le réseau. Le paquet comporte un identificateur de 16 bits un numéro de séquence de 16 bits et des données (optionnelles). ***Le champ code a pour valeur zéro***.

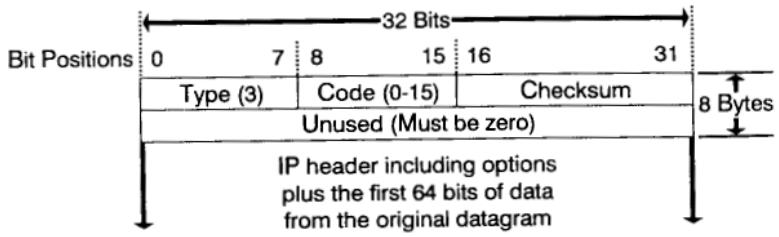
identificateur	Sequence



1.3.1.2 ICMP_UNREACHABLE_DEST (type=0x03)

Le paquet comporte un champ de 32 bits codé à zéro, l'en tête et les 8 premiers octets du datagramme rejeté.
Le champ code indique la raison de la destruction:

- 0 Réseau inaccessible
- 1 Host inaccessible
- 2 Protocole inaccessible
- 3 Port inaccessible
- 4 Fragmentation nécessaire mais interdite
- 5 Echec du Source Routing



1.3.1.3 ICMP_SOURCE_QUENCH (type=0x04)

Le format est identique à ICMP_UNREACHABLE_DEST. Il indique la destruction d'un paquet IP et notifie un engorgement d'une passerelle.

1.3.1.4 ICMP_REDIRECT(type=0x05)

Notification d'une passerelle plus opportune à utiliser. Un mot de 32 bit indique l'adresse IP de la passerelle, l'en tête IP et les 8 premiers octets du datagramme sont reproduits. Le champ code notifie les éventuelles restrictions de ce nouveau routage.

- 0 rediriger pour le réseau demandé.
- 1 ne rediriger que pour le host demandé.
- 2 ne rediriger que pour le réseau et le TOS demandé.
- 3 ne rediriger que pour le host et le TOS demandés.

1.3.1.5 ICMP_TIME_OUT (type=0x08)

Le contenu du paquet est identique à ICMP_UNREACHABLE_DEST. Le champ code indique la raison de destruction du paquet.

- 0 Champ TTL épuisé.
- 1 Attente trop longue lors d'un réassemblage de fragments.

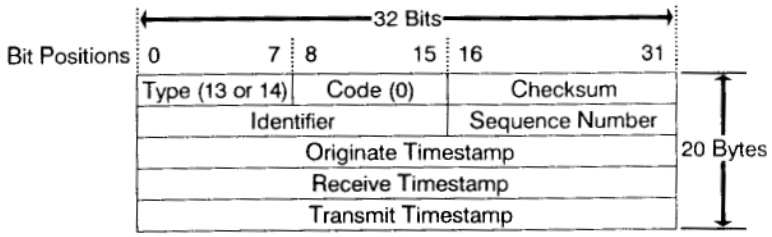
1.3.1.6 ICMP_HEADER_ERROR (type=0x0C)

Le premier octet du champ de 32 bits pointe vers la partie du datagramme anormale, l'en tête IP et les 8 premiers octets du datagramme sont reproduits.

1.3.1.7 ICMP_CLOCK_REQ (type=0x0D) et ICMP_CLOCK_RESP (type=0x0E)

Permet le calcul du temps de traversée dans le réseau mais également de calculer la différence des horloges entre deux machines.
ICMP_CLOCK_REQ comporte un identificateur (16 bits) et un numéro de séquence (16 bits) et une heure T1 (32 bits).
ICMP_CLOCK_RESP comporte un identificateur (16 bits) et un numéro de séquence (16 bits) l'heure T1 (32 bits), l'heure de réception T2, l'heure d'émission du message de retour T3.
Soit T4 l'heure de réception du message, si l'on admet T2-T1 = T4-T3 alors

- Temps de transit = (T4-T1-T3+T2)/2
- Delta_entre_horloges = (T2-T1+T3-T4)/2



1.3.1.8 ICMP_MASK_REQ (type=0x11) et ICMP_MASK_RESP (type=0x12)

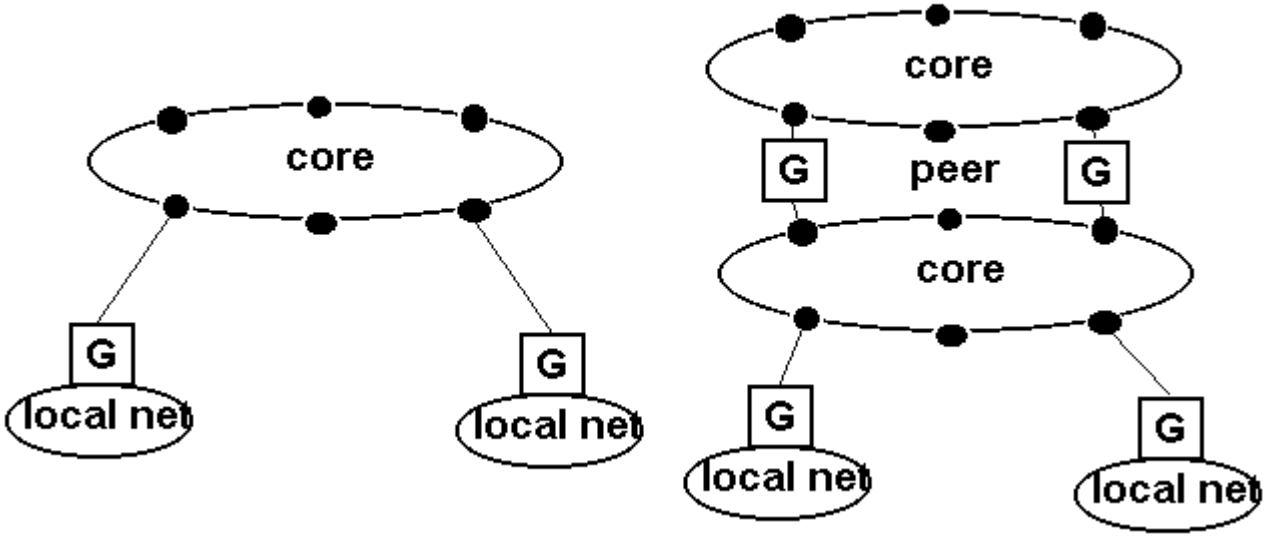
Le paquet comporte un identificateur (16 bits) et un numéro de séquence (16 bits). La réponse comporte un champ additionnel de 32 bits (le *Subnet Mask*)

1.4 Le routage IP

Les passerelles (gateways) déterminent la prochaine route à suivre à partir de leur table de routage. Les systèmes hôtes sont capables de router leurs datagrammes avec succès parce qu'ils utilisent le relais des passerelles. La route par **défaut** permet à une passerelle de router un paquet en l'absence d'informations spécifiques sur la route à suivre. Une passerelle comporte certaines informations de routage et une route par défaut.

Les passerelles internet peuvent être divisées en deux catégories les ***core gateways*** qui sont contrôlées par une autorité spécifique, et un ensemble plus important de ***noncore gateway***. L'architecture optimale consiste dans le fait que tout réseau local soit rattaché à une ***core gateway***. L'ossature du réseau repose sur un ensemble de ***core gateway*** qui ne possèdent aucune route par défaut et sont gérées par les autorités adhoc (cas du réseau ARPANET)

Cependant cette architecture se complique lorsque plusieurs réseaux de nature différentes (et donc gérés par des administrations différentes sont réunis), ces réseaux sont dits ***peers*** (***peer backbone networks).*** Par exemple les réseaux ARPANET et NFSNET.



Une passerelle mesure sa distance à une destination en unité de ***HOP***, le hop représente une traversée de passerelle (un entier nul ou positif). Une passerelle gère une table de routage qui comporte trois entrées un réseau de destination, la prochaine passerelle qui conduit vers ce réseau, et la distance de ce réseau exprimée en HOP.

Réseau Destination	@passerelle	distance en hop

Périodiquement les passerelles échangent leur table de routage et mettent à jour en conséquence leurs informations. Le couple réseau de destination distance est appelé **V*ector-Distance*** (V,D).

Il existe trois variantes de protocoles de routage :

- Les algorithmes à vecteurs de distances (***Vector Distanc*e**), par exemple GGP, RIP
- Les algorithmes à états de lien (***Link State***) par exemple EGP, OSPF
- Les algorithmes ***Path Vector****,* par exemple BGP

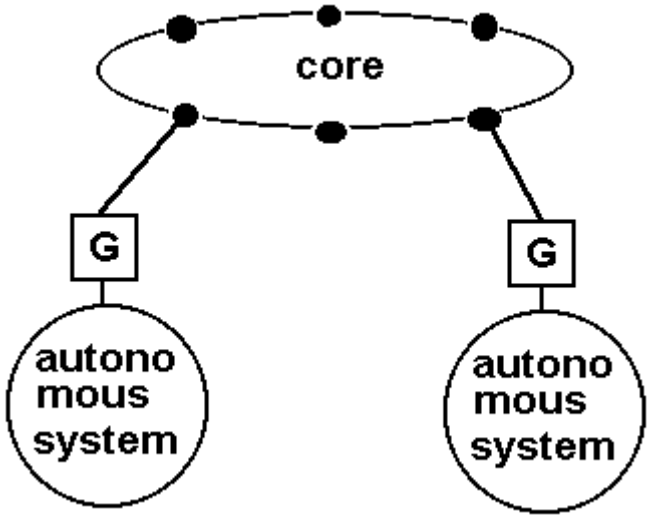
1.4.1 Gateway To Gateway Protocol (GGP - RFC 823)

Ce protocole de type (V,D) est utilisé entre les *core gateways*. La distance d'une passerelle à un réseau auquel elle est connectée est de 0 hop. Un paquet GGP comporte la liste des couples vecteur

distance, la distance est codée sur 8 bits (de 0 à 255) le réseau est codé par 24 bits (le plus grand identificateur d'un réseau de classe C comporte 24 bits).

1.4.2 Systèmes autonome, EGP RFC 904 Exterior Gateway Protocol.

Nous avons supposé précédemment que notre réseau consistait en un ou plusieurs réseaux *backbone*, ces derniers étant constitué et reliés par des *core gateways.* Dans cette architecture une passerelle *non core* utilise un *core gateway* comme route par défaut. Ce choix conduit à des configurations non optimales (ajout de HOP supplémentaires). Pour palier à ces inconvénients il est nécessaire d'introduire un mécanisme qui permette aux *non core gateways* d'apprendre des routes à partir des *core gateways*.



Une groupe de réseaux et de passerelles contrôlé par une autorité administrative unique est appelé un système autonome (*autonomous system*) Les passerelles à l'intérieur d'un système autonome sont libres d'utiliser leur propre mécanisme d'établissement de routes. Ainsi les core gateways forment un système autonome qui met en oeuvre le protocole GGP. Conceptuellement un système autonome est relié au réseau backbone par un *core gateway*. A des fins d'automatisation des procédures de routage un système autonome est caractérisé par un nombre (*autonomous system number*) délivré par une autorité centrale d'internet.

1.4.2.1 EGP

Des voisins du même système sont dits *interior neighbors*, des voisins de systèmes différents sont dits *exterior neighbors*. Le protocole utilisé par des passerelles extérieures est dit *exterior gateway protocol*.

EGP permet à des passerelles de systèmes autonomes différents d'échanger leur table de routage. Il comporte 10 types de messages. Un système autonome est identifié par un numéro de 16 bits. La version du protocole est identifée par un nombre de 8 bits.

Une passerelle émet ***acquisition_request*** pour établir la communication avec une autre passerelle, la réponse peut être ***acquisition_confirm*** ou ***acquisition_refuse***. La relation entre passerelles peut être rompue par ***Cease_Request*** et ***Cease_Confirm***.

Une trame Hello (la réponse à ***Hello*** est dénommée ***I_Heard_You***) est émise de manière périodique (***hello interval***) pour vérifier que le voisin est en vie. Un ***polling interval*** est également défini qui fixe le temps minimum entre interrogations (pour acquérir les tables de routage).

Les messages ***EGP poll request*** et ***EGP poll response*** permettent d'obtenir les routes menant à un réseau particulier défini par son adresse IP. ***EGP routing Update*** est un message émis par une passerelle qui contient toutes les routes définies à l'intérieur du système autonome lié à cette passerelle.

Une route est caractérisée par une distance (8 bits) et un réseau de destination (24 bits).

EGP ***n'utilise pas les distances***, la seule information retenue est l'existence d'une route qui mène à un sous réseau donné, l'indication retenue est « ***mon système autonome fournit un chemin vers ce réseau*** ».

EGP ne propage donc que les informations relatives à la possibilité d'atteindre un réseau donné. Il réduit par la même la topologie d'internet à une structure d'arbre, la racine étant un *core gateway* et les nœuds des systèmes autonomes. Il n'existe pas de boucles entre les systèmes autonomes qui appartiennent à cet arbre.

1.4.3 BGP, Border Gateway Protocol, RFC 4271

BGP est utilisé par les routeurs de bord des systèmes autonomies. C'est une variante d'un protocole à état de lien, dite *Path Vector*. Les routeurs BGP échangent des tables de liens relatives à des nœuds de confiance.

1.4.4 Passerelle intérieure et routage.

Les passerelles d'un même système autonome sont dites ***interior gateway***. Deux types d'approches peuvent coexister les tables de routages établies manuellement et l'utilisation de protocoles dits IGP (***interior gateway protocol***) Une passerelle peut utiliser simultanément un protocole pour ses communications aves d'autres systèmes autonomes et un protocole pour ses communications avec son propre système.

1.4.4.1 Routing Information Protocol RIP RFC 1058

RIP est un protocole de routage largement utilisé, de type Vecteur-Distance. La distance d'un réseau à une passerelle est comprise entre 1 (réseau directement connecté) et 15. 16 signifie l'infinité, c'est à dire qu'il n'existe pas de route vers ce réseau. L'adresse d'un réseau est définie par 14 octets pour répondre à certains cas particulier, dans le cas usuel seuls 4 octets de champ définiront une adresse IP associé à un réseau, 0.0.0.0 est associé à une route par défaut (vers le reste de l'internet).

RIP définit deux types de participants, les ***actifs*** (fournissent des informations - passerelles typiquement) et les ***passifs*** (qui se contentent d'écouter les informations sans en fournir machine hôtes typiquement).

Une passerelle émet en diffusion un message toutes les ***30 secondes*** (RIP fonctionne avec le port 520 de la couche UDP). Ce message contient les tables de routage courantes, c'est à dire des couples adresse IP et distance.

Un message de requête peut être émis en diffusion pour obtenir les tables de routage des différentes passerelles adjacentes. Les routes acquises par RIP sont invalidées si aucun paquet RIP de confirmation n'est reçu après ***180 secondes***.

RIP ne détecte pas les boucles, et converge lentement c'est à dire que le temps de stabilisation après l'ajout ou la suppression d'une route est important.

Une solution possible au problème de convergence est la technique dite ***split horizon time*** dans laquelle une passerelle note l'interface sur laquelle elle a reçu une information sur une route particulière, et ne propage pas d'information relative à cette route sur la même interface.

1.4.4.2 OPEN SPF Protocol (OSPF RFC 2328)

OSPF est un protocole ***d'état de liens*** (*Path Vector)* transporté par IP. Les liens d'une passerelle sont les adresses IP des voisins intérieurs de cette dernière.

1.5 IPv6 ou Ipng RFC 1550 - RFC 1752

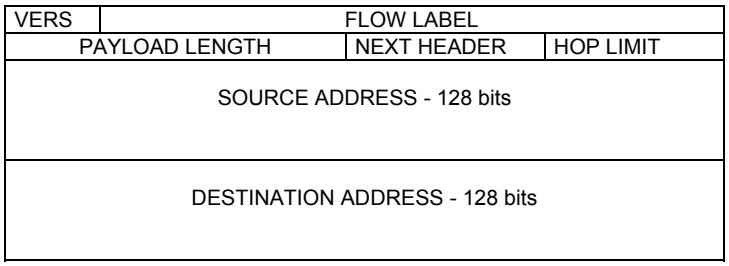
1.5.1 Introduction

La première version du protocole IP est connue sous le nom de IPv4. Les pères de ce protocole ne mesuraient pas le futur succès de leur création. Le codage des adresses IP sur 32 bits ne pourra plus

fournir la demande toujours croissante d'adresses vers 2012. La première motivation dans la genèse de IPng est donc l'augmentation du nombre d'adresses disponibles, avec comme principale contrainte la coexistence avec IPv4. D'autre part une les nouvelles ambitions des technologies TCP/IP visent le transport des flux multimédia (son ou vidéo) qui nécessitent des délais d'acheminement garantis. Enfin l'utilisation croissante d'internet pour des applications de télépaiement conduit à inclure dans IP des options de sécurité telles que l'authentification de la source.

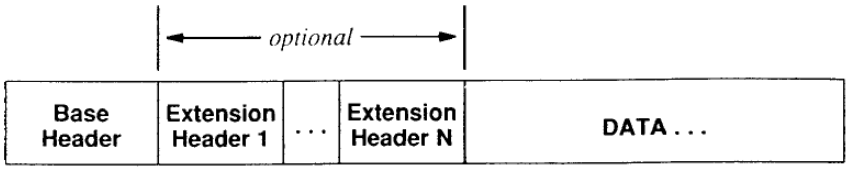
1.5.2 Caractéristiques de Ipv6

- Des adresses larges de 128 bits.
- Un en tête de taille variable, non limité à 60 octets (dans Ipv4).
- Des options améliorées.
- L'allocation de ressources, IPv6 remplace le type de service défini par Ipv4 par un mécanisme qui permet l'allocation de ressources (vidéo son ...)
- La possibilité d'étendre le protocole.



1.5.3 Le datagramme Ipv6

Le datagramme Ipv6 comporte un en tête de base (40 octets), une liste d'options, et des données.



1.5.3.1 L'en tête de base - 40 octets.

1.5.3.1.1 VERS

Un champ de 4 bits (VERS = 6)

1.5.3.1.2 FLOW LABEL

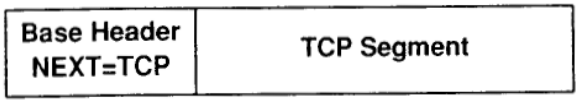
28 bits, équivalent à SERVICE TYPE de Ipv4, spécifie la qualité de service requise. FLOW LABEL se décompose en un sous champ TCLASS de 4 bits qui indique la classe de service (0..7 sensible au temps de transfert, 8...15 indicateur de priorité) et un sous champ FLOW IDENTIFIER de 24 bits.

1.5.3.1.3 PAYLOAD LENGTH

La taille du datagramme (16 bits).

1.5.3.1.4 NEXT HEADER

8 bits le protocole encapsulé, ou le prochain en tête.



1.5.3.1.5 HOP LIMIT.

8 bits, remplace le champ TIME-TO-LIVE de Ipv4

1.5.3.1.6 SOURCE ADDRESS

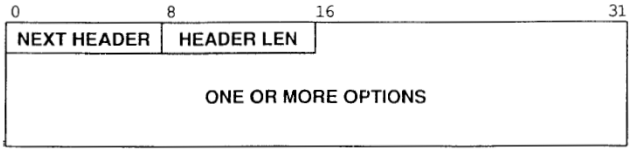
L'adresse de source 128 bits

1.5.3.1.7 DESTINATION ADDRESS

L'adresse de destination 128 bits.

1.5.3.2 Les options.

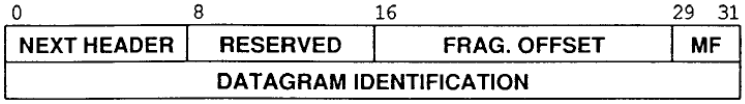
Les options sont exprimées sous forme d'une liste chaînée.



1.5.4 La segmentation et le réassemblage.

La fragmentation est assurée de bout en bout, c'est à dire que l'expéditeur du paquet doit déterminer à l'aide d'une procédure de ***Path MTU Discovery*** la valeur minimale du MTU à utiliser. Contrairement à Ipv4 les passerelles n'assurent plus le service de fragmentation. Les paquets fragments sont marqués à l'aide d'un ***Fragment Extension Header***.

Un bit MF indique si le fragment est le dernier d'un bloc fragmenté.

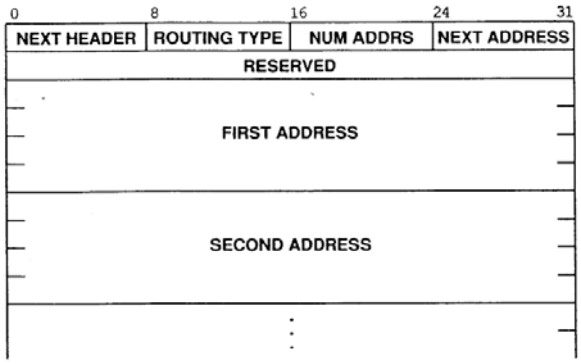


1.5.5 Problèmes posés par la fragmentation de bout en bout.

En cas de reconfiguration le MTU déterminé peut être inadéquate pour la nouvelle route. Dans ce cas les routeurs intermédiaires utilisent la technique du tunnel (**tunneling**) de IPv6 à travers IPv6. Les paquets sont segmentés et réassemblés à l'aide d'un en tête spécifique, et uniquement entre deux passerelles adjacentes.

1.5.6 Le Source Routing (routage à la source).

Un en tête spécifique indique dans ce cas la liste des adresses à suivre (128 bits).



1.5.7 La sécurité.

Un en tête ***IPv6 Encapsulating Security Header*** peut être utilisé pour échanger des données sécurisées entre deux systèmes hôtes.

1.5.8 Les options IPv6

Une option comporte un champ NEXT HEADER (8 bits) qui indique la nature du prochain en tête, un champ HEADER LEN (8 bits) indique la longueur totale de l'option. Une option est composée d'un champ type (8 bits), d'un champ longueur (8 bits) et des données associées. Les 2 bits de poids fort du champ option indiquent le comportement d'une passerelle lorsqu'elle ne comprend la signification de cette option.

1.5.9 Notation des adresses IP.

Les adresses sont représentées sous forme hexadécimale, les 128 bits sont regroupés en 8 mots de 16 bits.
Exemple
104.230.140.100.255.255.255.255.0.0.17.128.150.10.255.255 est écrit
68E6:8C64:FFFF:FFFF:0:1180:96A:FFFF
Cette notation est appelée ***colon hexadecimal notation*** (ou ***colon hex***), il est possible de compresser une suite de zéros.
FF05:0:0:0:0:0:0:B3 <=> FF05::B3
de même
0:0:0:0:0:0:128.0.2.1 <=> ::128.10.2.1

1.5.10 Les trois types principaux d'adresses IP

- Les adresses **Unicast**, associées à une machine unique.
- Les adresses **Cluster**, associées à un ensemble de machines qui partagent un même préfix d'adresse. Le paquet est alors routé vers une machine et une seule de cet ensemble.
- Les adresses **Multicast**, la destination est un ensemble de machines situées à différentes locations, chaque membre du groupe reçoit une copie du paquet. Dans IPv6 le ***broadcas***t est un cas particulier du multicast.

1.5.11 Plan d'adressage IPv6.

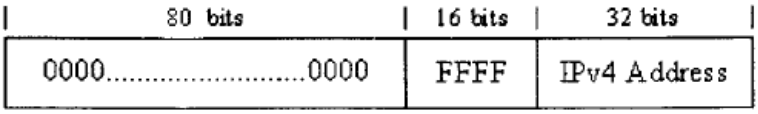
Les premiers bits de l'adresse (au maximum les 8 premiers) définissent des classes d'adresses, dont le nombre d'éléments est par ailleurs différents.

Le préfix 0000 0000 est réservé (en partie) pour l'encapsulation des adresses Ipv4. En particulier une adresse qui comporte 80 zéros puis 16 un et enfin 32 bits encapsule une adresse Ipv4.
::FFFF.x.y.z.t

Remarque: ***l'encapsulation de l'adresse Ipv4 dans une adresse Ipv6 a été choisie de telle sorte que les deux adresses aient la même somme de contrôle en modulo 65535.***

1.5.12 Communication entre IPv4 et Ipv6

Pour communiquer avec un hôte IPv4 une machine IPv6 utilise les services d'un translateur. Le translateur effectue la conversion de IPv6 vers IPv4 pour les paquets et émis et de IPv4 vers IPv6 pour les paquets reçus.



1.5.13 Un exemple d'adressage hiérarchique, NAP.

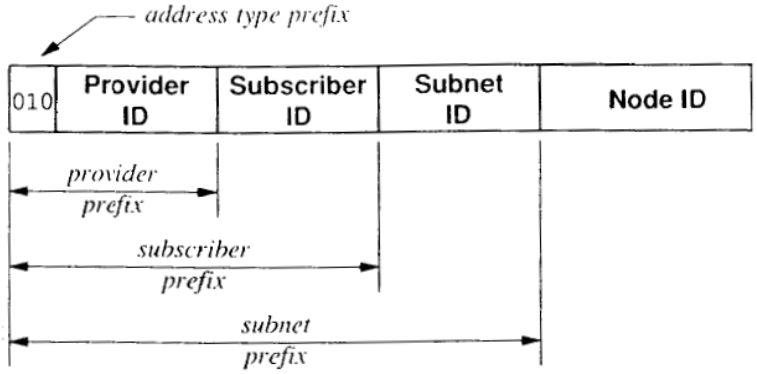
Une classe d'adresse IPv6 (***Network Access Provider***) est attribuée aux compagnies qui fournissent des accès internet à leurs clients.

Un provider est identifié par son ProviderID. Le ***provider prefix*** est obtenu en accolant au prefix d'adresse (010) le ProviderID.

Le client obtient de son provider un SubscriberID qui forme à son tour un ***subscriber prefix***.

Le client identifie ses sous réseaux par un SubNetID, qui donne naissance au ***subnet prefix***.

La partie restante de l'adresse constitue l'identificateur d'un système hôte dans un réseau identifié par son subnet prefix.



1.5.14 En exemple d'échange IPv4

```
FF FF FF FF FF FF 08 00 20 02 45 9E 08 06 00 01 08 00 06 04 00 01 08 00
20 02 45 9E 81 68 FE 06 00 00 00 00 00 00 81 68 FE 05


08 00 20 02 45 9E 08 00 20 07 08 94 08 06 00 01 08 00 06 04 00 02 08 00
20 07 08 94 81 68 FE 05 08 00 20 02 45 9E 81 68 FE 06


08 00 20 07 08 94 08 00 20 02 45 9E 08 00 45 00 00 54 88 FE 00 00 FF 01
30 CD 81 68 FE 06 81 68 FE 05 08 00 84 12 12 14 00 00 85 53 5D 27 61 5B
03 00 08 09 0A 0B 0C 0D 0E 0F 10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D
1E 1F 20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F 30 31 32 33 34 35
36 37


08 00 20 02 45 9E 08 00 20 07 08 94 08 00 45 00 00 54 B4 DD 00 00 FF 01
07 EE 81 68 FE 05 81 68 FE 06 00 00 BC 12 12 14 00 00 85 53 5D 27 61 5B
03 00 08 09 0A 0B 0C 0D 0E 0F 10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D
1E 1F 20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F 30 31 32 33 34 35
36 37


08 00 20 07 08 94 08 00 20 02 45 9E 08 00 45 00 00 54 8C 14 00 00 FF 01
30 87 81 68 FE 06 81 68 FE 05 08 00 22 87 13 14 00 00 8D 53 5D 27 E1 E6
08 00 08 09 0A 0B 0C 0D 0E 0F 10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D
1E 1F 20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F 30 31 32 33 34 35
36 37


08 00 20 02 45 9E 08 00 20 07 08 94 08 00 45 00 00 54 B4 DE 00 00 FF 01
07 ED 81 68 FE 05 81 68 FE 06 00 00 2A 87 13 14 00 00 8D 53 5D 27 E1 E6
08 00 08 09 0A 0B 0C 0D 0E 0F 10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D
1E 1F 20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F 30 31 32 33 34 35
36 37


08 00 20 07 08 94 08 00 20 02 45 9E 08 00 45 00 00 54 8C 2C 00 00 FF 01
30 9F 81 68 FE 06 81 68 FE 05 08 00 7C C4 14 14 00 00 9A 53 5D 27 81 A9
03 00 08 09 0A 0B 0C 0D 0E 0F 10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D
1E 1F 20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F 30 31 32 33 34 35
36 37


08 00 20 02 45 9E 08 00 20 07 08 94 08 00 45 00 00 54 B4 DF 00 00 FF 01
07 EC 81 68 FE 05 81 68 FE 06 00 00 B4 C4 14 14 00 00 9A 53 5D 27 81 A9
03 00 08 09 0A 0B 0C 0D 0E 0F 10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D
1E 1F 20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F 30 31 32 33 34 35
36 37
```

1.5.15 Un exemple d'échange IPv6

```
00 60 97 07 69 ea 00 00  86 05 80 da 86 dd 60 00
00 00 00 10 3a 40 3f fe  05 07 00 00 00 01 02 00
86 ff fe 05 80 da 3f fe  05 01 00 00 10 01 00 00
00 00 00 00 00 02 80 00  a5 72 76 20 00 00 01 c9
e7 36 d3 39 06 00

00 00 86 05 80 da 00 60  97 07 69 ea 86 dd 60 00
00 00 00 10 3a 3b 3f fe  05 01 00 00 10 01 00 00
00 00 00 00 00 02 3f fe  05 07 00 00 00 01 02 00
86 ff fe 05 80 da 81 00  a4 72 76 20 00 00 01 c9
e7 36 d3 39 06 00
```

2. TCP, le protocole de contrôle de transmission

2.1 Introduction

La spécification originelle de TCP est la RFC 793. Ce protocole fournit un service en mode connecté avec contrôle de flux et correction d'erreur.
Les données applicatives sont fractionnées en fragments, l'unité d'information émise est nommée ***segment*** (quelques MTU en règle générale).

- Lorsque TCP émet il maintient une horloge qui attend l'acquittement de ce segment. On parle de timeout adaptatif.
- Les données sont acquittées par le récepteur
- Une somme de contrôle (en tête et données) assure le contrôle de l'intégrité des éléments protocolaires de TCP.
- IP étant un service 3 sans connexion, l'ordre d'arrivée des datagrammes IP n'est pas garanti. TCP assure la fonction de remise en ordre.
- Les datagrammes dupliqués doivent être ignorés.
- TCP assure également un contrôle de flux qui permet un dialogue correct entre deux machines dont les performances sont différentes.

2.2 Eléments protocolaires.

L'en tête TCP comporte typiquement 20 octets auxquels se rajoutent des options et généralement des données. On nomme ***segment*** les données encapsulées dans une PDU TCP.
La combinaison d'un numéro de port et d'une adresse IP est parfois appelée un ***socket*** (rfc 793). Une connexion TCP/IP est donc identifiée par une paire de sockets.
Les données de l'application sont découpées en segments, le **numéro de séquence** identifie le premier octet du segment.
Lorsqu'une connexion est établie (TCP_SEG_SYN) le numéro de séquence initial (ISN) est tel que le 1° octet émis comportera un numéro de séquence égal à ISN+1.
Le numéro d'acquittement est égal au numéro du prochain octet à recevoir (soit ISN+1 dans l'acquittement d'une demande de connexion).
La régulation de flux est assurée par à protocole à fenêtre glissante, il n'existe pas de rejets sélectifs par le récepteur (le NAK ou REJ de l'OSI).

2.2.1 Format d'un en-tête TCP

<table>
<tr><td colspan="5">Port Source</td><td colspan="3">Port Destination</td></tr>
<tr><td colspan="8">Numéro de séquence</td></tr>
<tr><td colspan="8">Numéro d'acquittement</td></tr>
<tr><td>taille en tête 4 bits</td><td>réservé 6 bits</td><td>urg</td><td>ack</td><td>psh</td><td>rst</td><td>syn</td><td>fin</td><td>taille de fenêtre (16 bits)</td></tr>
<tr><td colspan="6">Somme de contrôle TCP</td><td colspan="3">pointeur urgent</td></tr>
<tr><td colspan="9">Options</td></tr>
</table>

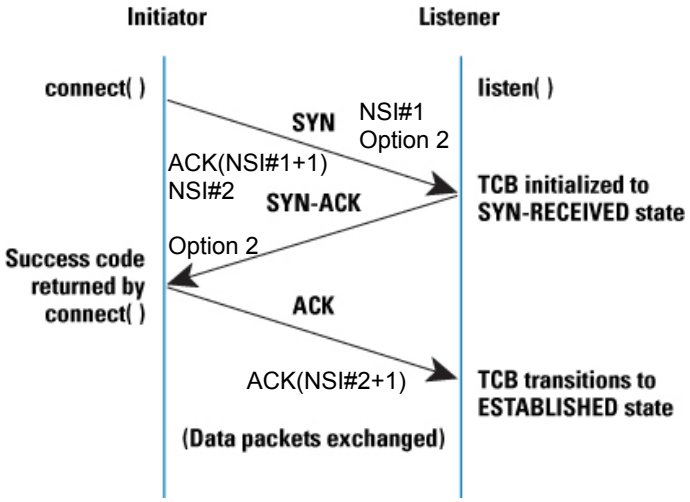
- ***Port Source SRC PORT*** , 16 bits le numéro du port qui émet le PDU
- ***Port Destination DEST PORT***, 16 bits le numéro du port destinataire du PDU
- ***Numéro de séquence***, 32 bits, SEQ Le numéro de séquence du premier octet.
  - Si TCP_SEG_SYN est présent ce champ contient le numéro du dernier octet envoyé (qui est virtuel dans l'établissement d'une connexion).
- ***Numéro d'acquittement***, 32 bits, ACK Le numéro du prochain octet attendu.
  - Ce numéro constitue donc l'acquittement des octets de numéros strictement inférieurs.
- ***Taille de l'en tête***, 4 bit, IHL la longueur de l'en tête TCP exprimée en mots de 32 bits.
- ***Bits type de segment***, 6 bits.
  - URG valide le champ URGPTR (position relative des données urgentes).
  - ACK Valide le champ ACK
  - PSH Demande la remise forcée de données (remise non différée par le récepteur).

  - RST Demande la rupture anormale de la connexion.
  - SYN Demande l'établissement de la connexion.
  - FIN Demande la fin de la connexion.
- ***Fenêtre***, 16 bits W la taille en octets à partir de l'octet ACK de la fenêtre d'émission allouée au récepteur.
- ***Checksum***, 16 bits
  - somme de contrôle de l'en tête TCP, des données et d'un pseudo en tête qui comporte des éléments de l'en tête IP (Adresse IP source, adresse IP destination, octet PROTocole encapsulé, longueur totale du segment). Le mode opératoire est identique à celui utilisé par le checksum IP.
- ***Pointeur de données urgentes***, 16 bits, URGPTR. la position des données urgentes par rapport au premier octet de données.
- **Les options**
  - Une option est constituée par un seul octet (option de longueur fixe), ou par un ensemble Type (1 octet), longueur (1 octet) valeur. La longueur totale des options doit être un multiple de 4 octets (éventuellement complété par des octets de bourrage).
  - option 0 (1 octet) fin d'option, termine l'analyse des options.
  - option 1 (1 octet) NOP sans effet.
  - option 2 (4 octets) la taille maximale du segment supporté (2 octets). Ce champ est présent dans TCP_SEG_SYN et TCP_SEG_SYN_ACK. Chacun des deux parties informe son homologue de ses exigences, mais il ne s'agit pas d'une négociation.

2.3 Etablissement et terminaison d'une connexion TCP.

2.3.1 Protocole d'établissement de la connexion.

- Une extrémité (le client) émet un segment avec le flag SYN (**TCP_SEG_SYN**), qui contient le numéro de séquence initial (**NSI#1**). Ce segment définit en fait le couple de sockets qui identifie la connexion.
- Le récepteur répond TCP_SEG_SYN_ACK avec son propre SYN qui contient son numéro de séquence (**NSI#2**) l'acquittement ACK(**NSI#1** + 1)
- Le client acquitte le SYN du serveur par un ACK(**NSI#2+1),** TCP_SEG_ACK



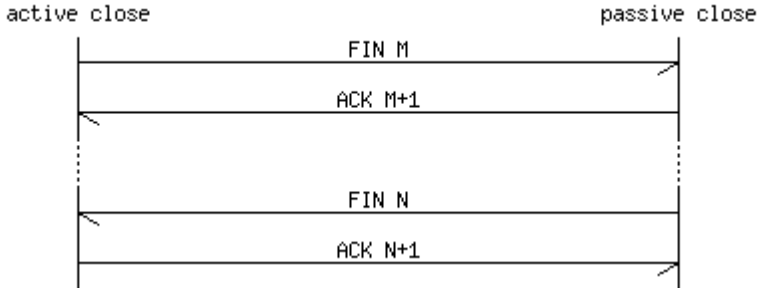
Une connexion TCP-IP comporte donc ***trois segments***, ceci est souvent appelé "triple poignée de main" (*three-way handshake*).
On dit que le côté générateur du premier SYN effectue une **ouverture active**. L'autre extrémité effectue une **ouverture passive.**
La RFC 793 spécifie que NSI devrait être vu comme un compteur de 32 bits incrémenté chaque 4 μs, soit une période voisine de 4e-6 * 4e9 = 16 000 secondes.
Cependant, dans la RFC initiale il est autorisé que chaque côté produise simultanément un SYN (TCP_SEG_SYN). Dans ce cas l'acquittement de connexion se fera par un TCP_SEG_ACK. La connexion comportera donc 4 segments, mais ce type de comportement ne correspond pas au paradigme client serveur.

2.3.2 Protocole de déconnexion.

Il faut 4 segments pour fermer une connexion TCP qui est rappelons le full duplex. Chaque direction peut être fermée indépendamment de l'autre, c'est ce que l'on nomme la ***semi-fermeture***. Chaque extrémité peut envoyer un FIN dans un segment contenant des données (TCP_SEG_FIN). La réception de ce flag indique seulement qu'une extrémité de la liaison a mis fin à ses émissions de données. L'autre extrémité de la connexion est autorisée à poursuivre ses émissions mais cette éventualité est rarement utilisée par les applications.
L'extrémité qui réalise la première fermeture effectue **la fermeture active**, l'autre extrémité effectue la **fermeture passive.** Il est possible à deux extrémités de réaliser simultanément une fermeture active.
Le schéma de fermeture comporte les étapes suivantes:

- Emission de FIN#1 (API ***close*** des sockets)
- Réception de FIN#1 - Emission de FIN#2 (End Of File)
- Réception de FIN#2, Emission de ACK(FIN#1 + 1)
- Réception de ACK(FIN#1 +1) ACK (FIN#2 + 1)

Les segments FIN comportent un numéro de séquence, exactement comme un SYN.



2.3.3 Traces d'une connexion / déconnexion

```
08 00 20 07 0B 94 08 00 20 02 45 9E 08 00 45 00 00 2C 8A 93 00 00 1E 06
13 5C 81 68 FE 06 81 68 FE 05 04 2A 00 15 55 EE 0C 00 00 00 00 00 60 02
10 00 24 D0 00 00 02 04 04 00


08 00 20 02 45 9E 08 00 20 07 0B 94 08 00 45 00 00 2C B4 38 00 00 1E 06
E9 B6 81 68 FE 05 81 68 FE 06 00 15 04 2A 2A D8 C0 00 55 EE 0C 01 60 12
10 00 39 E6 00 00 02 04 04 00


08 00 20 07 0B 94 08 00 20 02 45 9E 08 00 45 00 00 28 8A 94 00 00 1E 06
13 5F 81 68 FE 06 81 68 FE 05 04 2A 00 15 55 EE 0C 01 2A D8 C0 01 50 10
10 00 4F EF 00 00

                                        ...
08 00 20 07 0B 94 08 00 20 02 45 9E 08 00 45 00 00 28 8A D7 00 00 1E 06
13 1C 81 68 FE 06 81 68 FE 05 04 2A 00 15 55 EE 0C 62 2A D8 C1 78 50 11
10 00 4E 16 00 00


08 00 20 02 45 9E 08 00 20 07 0B 94 08 00 45 00 00 28 B4 5D 00 00 1E 06
E9 95 81 68 FE 05 81 68 FE 06 00 15 04 2A 2A D8 C1 78 55 EE 0C 63 50 10
10 00 4E 16 00 00


08 00 20 02 45 9E 08 00 20 07 0B 94 08 00 45 00 00 28 B4 5E 00 00 1E 06
E9 94 81 68 FE 05 81 68 FE 06 00 15 04 2A 2A D8 C1 78 55 EE 0C 63 50 11
10 00 4E 15 00 00


08 00 20 07 0B 94 08 00 20 02 45 9E 08 00 45 00 00 28 8A D8 00 00 1E 06
13 1B 81 68 FE 06 81 68 FE 05 04 2A 00 15 55 EE 0C 63 2A D8 C1 79 50 10
10 00 4E 15 00 00
```

2.3.4 Timeout d'établissement de connexion.

Dans les systèmes Berkeley une deuxième tentative intervient après 5.8 s et une troisième après 24 secondes. La tentative de connexion est abandonnée au bout de 75 secondes.

2.3.5 Taille maximum de segment MSS.

La taille maximum d'un segment MSS (Maximum Segment Size) est émise dans le TCP_SEG_SYN. Lorsque l'option n'est pas utilisée la valeur par défaut est 536. Les systèmes BSD utilisent une valeur multiple de 512 octets. D'autre systèmes (AIX ...) utilisent la valeur de 1460 sur des tronçons Ethernet Le MSS est normalement un multiple du MTU associé au réseau physique (taille maximale d'une trame IP).

2.3.6 La semi fermeture de TCP.

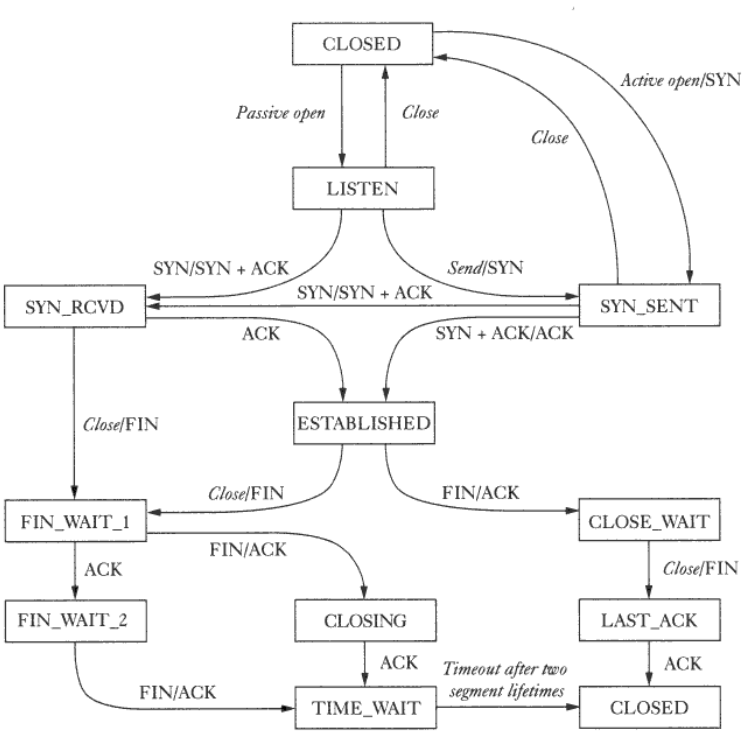
L'interprétation d'une semi fermeture est la suivante: «j'ai fini d'envoyer des données (FIN) mais j'accepte les données de l'autre extrémité jusqu'à réception de FIN. Cette fonctionnalité est réalisée dans les API sockets par ***shutdown*** avec pour deuxième argument 1.

Le schéma de semi fermeture est donc le suivant:

- Le client émet FIN#1 (shutdown)
- Le serveur répond ACK(FIN#1 +1)
- Le serveur continue à envoyer des données (write).
- Le client reçoit les données (read).
- Le serveur émet FIN#2 (close)
- Le client reçoit FIN#2 (End Of File).
- Le client émet ACK(FIN#2 +1)

2.3.7 Le diagramme d'états de TCP

Ce diagramme comporte 11 états. L'état CLOSED est l'état de non existence d'une connexion. La transition LISTEN - SYN_SENT n'est en fait pas supportée dans les implémentations dérivées de Berkeley. L'ouverture active est associée à SYN_SENT, l'ouverture passive à SYN_RCVD. La connexion est établie dans l'état ESTABLISHED. Les états CLOSE_WAIT et LAST_ACK sont associés à une fermeture passive. Les états FIN_WAIT_1 FIN_WAIT_2 CLOSING et TIME_WAIT sont associés aux fermetures passive et simultanée.



2.3.7.1 L'état TIME_WAIT- 2MSL

L'état TIME_WAIT est encore dénommé 2MSL. Il est caractérisé par un timeout égal à MSL (durée de vie maximum du segment). C'est la durée maximale qu'un segment peut exister sur le réseau avant d'être rejeté. MSL est limité à deux minutes. Dans cet état la paire de sockets définissant la connexion ne peut être réutilisée.

L'option SO_REUSEADDR permet de contourner cette interdiction.

La RFC 793 stipule qu'une machine ne peut créer aucune connexion TCP lors de sa mise en tension avant un délai égal à MSL.

2.3.7.2 L'état FIN_WAIT_2

Dans cet état FIN a été émis et acquitté. Sauf dans le cas d'une semi-fermeture nous attendons que l'autre extrémité de la connexion émette FIN. Lors de la fermeture de la deuxième extrémité le système transitera à l'état TIME_WAIT. Si le TCP_SEG_FIN n'est jamais reçu cet état est bloquant.

Pour éviter cette attente infinie TCP place la connexion dans l'état CLOSED si elle reste inactive durant 10mn et 75 secondes.

2.3.8 Segments de reset.

RESET est généré en réponse à une demande de connexion à un port qui n'est associé à aucun processus.

- SYN#1
- RST ACK(SYN#1 + 1).

Une remise ordonnée correspondant à la fin normale d'une connexion (envoi de FIN). Dans certain cas cette fin paisible est impossible (remise avortée). Un segment RST est envoyé immédiatement toutes les files d'attentes attachées à la connexion sont ignorées et abandonnées. Le récepteur interprète immédiatement RST.

Les API sockets fournissent la possibilité de générer un RST au lieu d'un FIN normal (option SO_LINGER avec un retard de t secondes). La réception d'un RST n'est pas acquittée.

La réception d'un RST se traduit sur une console par un message du type «*Connection Reset by Peer*»

2.3.9 Les connexions semi ouvertes

Une connexion semi ouverte est crée lorsque une extrémité de la liaison a fermé la connexion sans notification (coupure de secteur par exemple). La règle d'un serveur est de répondre RST à tout segment qui correspond à une connexion inexistante.

2.3.10 L'ouverture simultanée

Exemple
A port 8888 se connecte au port 7777 de B
B port 7777 se connecte au port 8888 de B
Les ports 7777 et 8888 sont à la fois utilisés par des applications clients et serveur (cette option n'est pas supportée dans Internet !).

- SYN#1, SYN#2
- ACK(SYN#1 +1) et ACK(SYN#2 + 1).

L'ouverture simultanée se traduit par la transition SYN_SENT - SYN_RCVD.

2.3.11 Fermeture simultanée

La fermeture simultanée se produit lorsque le client et le serveur émettent simultanément FIN. Les deux extrémités passent dans ce cas de ESTABLISHED à FIN_WAIT_1, puis de FIN_WAIT_1 à CLOSE.

FIN#1, FIN#2
ACK(FIN#1 + 1), ACK(FIN#2 + 1)

2.3.12 A propos des serveurs.

Sous Unix les serveurs sont réalisés par la fonction fork() ou l'usage de threads.

Par exemple la commande netstat -a -n - f - inet (SUN) affiche l'état des connexions du système hôte.

Les connexions entrantes sont placées dans une file d'attente dont la taille est ***backlog*** (compris entre 0 et 5). Un TCP_SEG_SYN est ignoré lorsque la file d'attente est pleine. Si la file d'attente comporte un emplacement libre TCP acquitte le SYN et place la connexion entrante dans la file d'attente. On ignorant SYN lorsque la file est pleine le serveur espère que le client réitère sa demande avec une file d'attente non pleine.

2.4 Mode TCP interactifs

2.4.1 Introduction

Il existe classiquement deux modes d'échanges de données, le mode transactionnel avec des paquets de faible taille, et le mode transfert de fichier, avec un débit élevé de paquets de grande taille.

2.4.2 Exemple de mode interactif.

Une application client (*telnet, rlogin, …*) génère un paquet à chaque frappe d'un caractère. Le serveur reçoit un caractère de donné et envoie l'acquittement et l'écho de retour dans le même segment. Le client acquitte à nouveau l'écho. On obtient de la sorte un flux de trois segments par frappe de caractère.

2.4.3 Acquittement retardé

Dans cette technique le processus TCP est réveillé par le noyau de manière périodique (toutes les 200ms, valeur maximale souhaitée 500 ms).
Lorsque des données sont émises l'ACK des données préalablement reçues est inséré dans le segment.
En l'absence de données à émettre, l'ensemble des données reçues toutes les 200ms est acquitté par un seul ACK.

2.4.4 L'algorithme de Nagle - RFC 896

La longueur d'un paquet qui ne comporte qu'un seul octet de données est typiquement de 41 octets (20 octets pour l'en tête IP, 20 pour l'en tête TC et un octet de donné). Ce type de datagramme est parfois appelé ***tiny datagram*** (petit segment).

L'algorithme de Nagle est le suivant : dans une connexion TCP il ne peut exister qu'un seule petit segment. Aucun petit segment ne peut exister tant que l'acquittement n'a pas été reçu. En l'attente de cet ACK les données à transmettre sont mémorisées, et seront émises lors de la réception de l'ACK attendu. En fait la notion de petit segment signifie que la quantité de données à émettre est strictement inférieure à la taille d'un segment.

Cependant il convient de remarquer que pour une faible charge le temps de retour d'un ACK sur un LAN 802.3 est de l'ordre de 16 ms, c'est à dire que l'algorithme de Nagle n'entre pas en jeu avant 60 frappes de caractères par secondes. La situation est différente à travers internet ou le temps de transit d'un paquet est plutôt de l'ordre de quelques dixièmes de seconde.

Certain protocoles exigent de pouvoir émettre sans retard des caractères spéciaux (ASCII ESC par exemple) dans ce cas l'option socket TCP_NODELAY invalide l'algorithme de Nagle.

2.5 Flux de données TCP en masse.

TCP utilise la notion de fenêtre glissante, c'est à dire qu'une extrémité de la connexion comporte une fenêtre de réception dont la taille est ***win*** octets. La taille initiale de cette fenêtre est fixée sans négociation lors de la connexion. Les données sont transportées dans des paquets TCP dont la taille

maximale a été fixée à ***mss*** lors de la connexion. Lors d'un acquittement ***ACK*** le récepteur indique le numéro du prochain octet non reçu - ack -(et donc éventuellement à émettre) et indique la largeur de la fenêtre de réception disponible win.

L'émetteur en déduit alors le numéro du prochain octet attendu (ack) ainsi que le nombre d'octets maximum (win). La fenêtre de l'émetteur sera donc comprise entre l'octet ack et l'octet ack+win-1.
Un champ win=0 dans un ACK(ack#n) signifie que le récepteur a reçu les données mais n'a pas été encore en mesure de les traiter. La réception de cet ACK a pour effet de stopper les émissions. Typiquement une trame ACK(ack=#n) avec win#0 autorise l'émission de win octets de donnés.

Par exemple supposons une connexion avec un mss de 1024 octets et une fenêtre initiale de 4096 octets. Si l'émetteur est très rapide au regard du récepteur, 4 segments sont émis de manière consécutive. Le récepteur notifie la réception des 4096 octets par ACK(ack=4097, win=0). Cette notification a pour avantage de libérer des ressources côté émetteur, l'indication win=0 lui interdit d'émettre d'autres données. Lorsque l'application a consommé les données un ACK (ack=4097 win=4096 - ***mise à jour de fenêtre***) permet la réception de prochaines données. L'émetteur génère alors quatre segments de 1024 octets, et positionne dans le dernier segment FIN et PUSH (seq#8193). Le récepteur termine la connexion par les séquences suivantes :

ack 8194, win 0
ack 8194, win 4096
FIN, ack 8194, win 4096.

2.5.1 Gestion de fenêtre glissante.

La fenêtre annoncée du récepteur est appelée ***offered windows*** (fenêtre offerte). L'émetteur calcule sa **fenêtre utilisable** comme la quantité de données qu'il peut immédiatement émettre.
La fenêtre utilisable **se ferme** lorsque des données sont émises et non acquittées.
La fenêtre utilisable **s'ouvre** lorsque l'autre extrémité lit les données et libère de l'espace dans son buffer de réception.
La fenêtre utilisable **se rétrécit** si l'autre extrémité modifie sa taille en cours de fonctionnement, mais cette possibilité n'est pas utilisée en pratique.

Quelques règles *de bon aloi* peuvent être énoncées :

- L'émetteur n'a pas à attendre une fenêtre de données soit pleine pour transmettre des informations
- Le récepteur n'a pas à attendre que la fenêtre de réception soit pleine avant d'envoyer un ACK.

Les tailles de fenêtres sont de l'ordre de quelques 1024 octets (1024 ... 16384) en fonction des systèmes utilisés.

2.5.2 Le flag PUSH.

L'utilisation de ce flag est aujourd'hui obsolète. A l'origine ce flag indiquait côté émetteur une requête d'envoi immédiat des données (c'est à dire une marque de fin de bloc à transmettre sans retard) et côté récepteur la délivrance immédiate des données reçues à l'application. Certaines implémentations passent les données à l'application lorsqu'un segment est reçu avec le flag push. Les implémentations dérivées de Berkeley ignorent le flag push, car elles ne retardent jamais la délivrance des données à l'application.
Les API sockets ne permettent pas la manipulation du flag push

2.5.3 Démarrage lent (slow start)

Le démarrage lent consiste à démarrer une connexion de telle sorte que les segments soient émis à la vitesse des ack reçus.

Une ***fenêtre de congestion cwnd*** est associée à l'émetteur. Sa valeur est exprimée en multiple de segments. La valeur initiale est 1. cwnd est incrémenté de 1 à chaque réception d'un ACK. Un émetteur transmet un compte d'octets égal à la plus petite valeur prise entre sa fenêtre annoncée et sa fenêtre de congestion. Ce qui signifie encore que le nombre maximum de segments qu'un émetteur peut générer est de (n+1) mss si n ACK ont été reçus.

En démarrage lent cwnd est incrémenté de 1 à chaque ACK, et donc le nombre de segments émis croît de manière exponentielle (dn/dt = k n(t)). Au delà d'un seuil (sthresh), cwnd évolue de manière linéaire:

cwnd = cwnd + f * taille-segment (avec f = 1/8)

2.5.4 Taille de fenêtre optimale

On nomme **RTT** (Round Trip Time) le temps de propagation aller retour de l'information. Le nombre maximum de bits émis durant RTT s'écrit donc :

C (bits) = BandePassante (bits/s) . RTT (s).

Le choix optimal de la taille de fenêtre consiste à recevoir un ack alors que la taille de la fenêtre annoncée est non nulle, soit

W = C

Par exemple si un segment mss est transmis en RTT/8 secondes (mss == 2.tpd/8 = tpd/4) la taille de fenêtre sera de 8 mss. En régime stationnaire le canal mémorise quatre segments, ce qui signifie que la fenêtre comporte 4 mss non émis et 4 émis soit 8 mss non acquittés. Un ACK est reçu chaque temps équivalent mss, avec les informations ack win=mss.

Lorsqu'une route comporte des sections de débits Ci différents, le débit utile est égal au plus petit débit Ci

2.5.5 Le mode urgent.

TCP offre la possibilité de marquer des données urgentes (flag URGENT). La fin des données urgentes est identifiée dans le segment par le pointeur de données urgentes de l'en tête TCP. Par contre (**!**) TCP n'offre aucun moyen de connaître l'emplacement du début de données urgentes dans le segment, l'application doit faire avec !. De fait une donnée urgente est généralement de taille un octet, ce qui résout le problème (puisque emplacement début égal emplacement fin).

Les API sockets appellent les données urgentes, données hors bandes (Out Of Band).

2.6 Retransmission dans TCP.

Un timer de re-transmission est utilisé losrqu'on espère un acquittement de l'autre extrémité. On appelle **RTT** le temps écoulé entre l'émission d'un paquet et la réception d'un ACK. On dénomme **RTO** (Retransmission TimeOut) la valeur du timeout (chien de garde) au bout duquel une retransmission se produit.

Le ***repli exponentiel*** du timeout consiste à doubler la valeur du chien de garde (RTO) après chaque retransmission. La valeur maximale de RTO est de l'ordre de 9 minutes.

2.6.1 Mesure du temps d'aller et retour RTT

La spécification initiale de TCP définit un coefficient de lissage de RTT, noté **R**, à l'aide d'un filtre passe bas:

$$R_{k+1} = a\, R_k + (1-a)\, M_k \;,\; M_{k+1} = R_{k+1}$$

M_k est l'ancienne estimation de RTT
R_k est la valeur du RTT mesurée pour cet aller et retour.
R_{k+1} est la nouvelle estimation de RTT.

Avec a =0.9, 90% de la nouvelle valeur de M provient de son ancienne estimation et 10 % de la dernière valeur mesurée de RTT.

Dans la RFC 793 la valeur recommandée de RTO est :

RTO = R β, valeur recommandée β=2

Jacobson (1988) a proposé calcul de RTO basé à la fois sur la moyenne et la variance de RTT

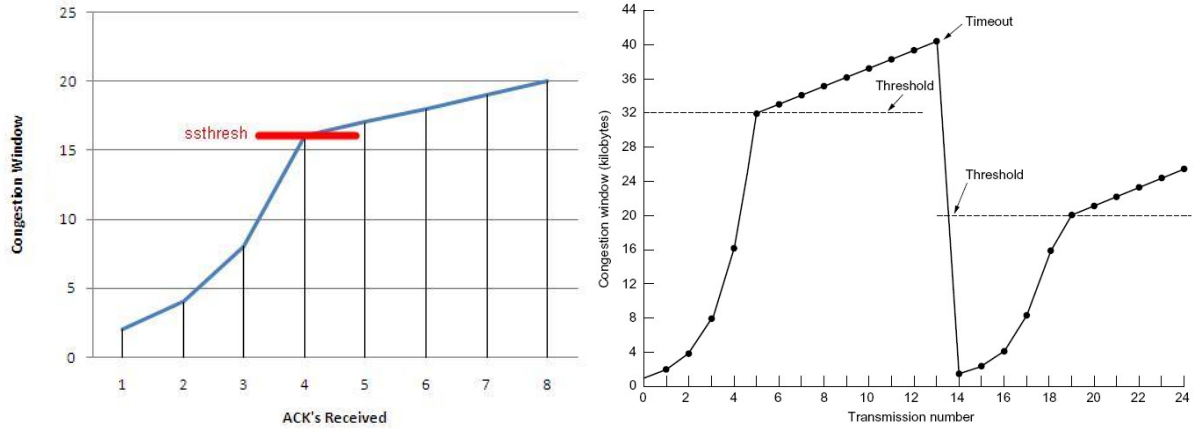
2.6.2 Algorithme de Karn.

La question est comment calculer RTO lors d'une retransmission (ambiguïté de la retransmission). La réponse proposée est que les ACK reçus sont ignorés pour le calcul de RTO. En cas de retransmission le repli exponentiel est appliqué (doublement de RTO), le calcul de RTO normal est repris à la réception d'un ACK qui ne correspond pas à un segment retransmis.

2.6.3 Détection de paquets perdus.

Un récepteur détecte un *trou* dans la suite des paquets reçus. Il notifie cet évènement à l'émetteur par l'émission d'un ACK du paquet immédiatement inférieur, puisque cet ACK a déjà été transmis préalablement on parle d'***ACK dupliqué***.

2.6.4 Algorithme d'évitement de congestion (Congestion Avoidance)



L'évitement de congestion permet de diminuer le débit de transmission de l'information. Cet algorithme utilise la variable ***cwnd*** préalablement définie (fenêtre de congestion utilisée pour le démarrage lent) et un paramètre ***ssthresh***.

1. Lors de l'initialisation de la connexion ***cwnd=1*** (1 segment) et ***ssthresh = 65535***.

2. TCP m'émet jamais plus de données que minimum de cwnd et de la fenêtre annoncée (win). cwnd est contrôlé par l'émetteur (cwnd est incrémenté de un à chaque réception d'un ACK) win par le récepteur.

3. Lorsque la congestion se produit (ack dupliqué ou timeout) la moitié de la fenêtre courante (minimum entre cwnd et la fenêtre annoncée, mais au moins 2 segments) est sauvegardée dans ssthresh. Si la congestion est due à un timeout cwnd est mis à un, c'est à dire qu'un démarrage lent se produit.

4. si cwnd est inférieur à ssthresh, on se trouve dans le mode de démarrage lent. Dans le cas contraire on se trouve en mode d'évitement de congestion.

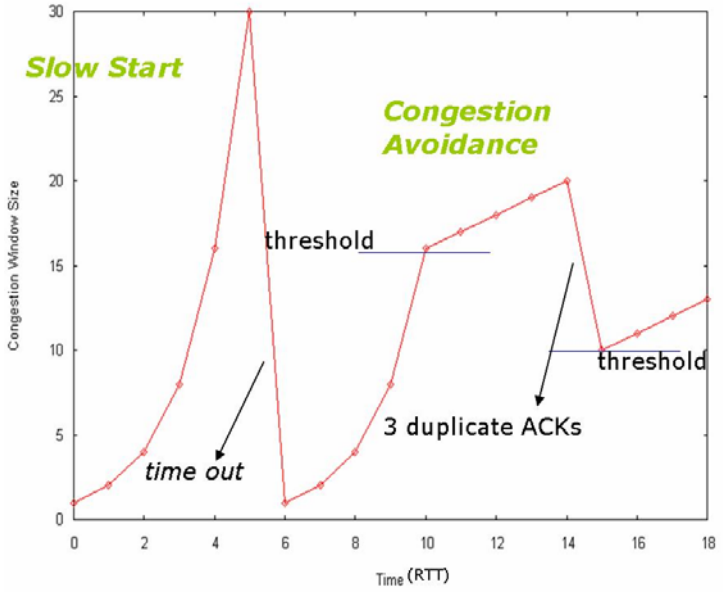
En démarrage lent cwnd est incrémenté de 1 à chaque ACK, et donc le nombre de segments émis croît de manière exponentielle (dn/dt = k n(t)). Lorsque cwnd devient égal à ssthresh la gestion de cwnd est modifiée de la manière suivante :

cwnd = cwnd + f * taille-segment (avec f = 1/8)

2.6.5 Algorithme de re-transmission et de recouvrement rapides

On applique les règles suivantes :

- 1. Lorsque le troisième ACK dupliqué est reçu mettre la valeur de ssthresh à cwnd/2 (la moitié de la fenêtre de congestion courante).
- 2. Retransmettre le segment manquant. Positionner cwnd à la valeur de ssthresh plus 3 fois la taille du segment.
- 3. A chaque réception d'un ACK dupliqué, incrémenter cwnd de la taille de segment, et transmettre un paquet (si cela est autorisé par la nouvelle valeur de cwnd).
- 4. Lors de la réception du premier ACK non dupliqué mettre la valeur de cwnd à la valeur de ssthresh définie au point n°1.



2.6.6 Repaquettisation.

Lorsque TCP détecte un timeout, il n'est pas dans l'obligation d'émettre un segment identique, il peut émettre un paquet de taille supérieure, à condition de ne pas dépasser la valeur MSS. C'est le phénomène de **"repaquettisation"**.

2.6.7 Timer persistant de TCP.

Un récepteur stoppe les émissions de l'émetteur à l'aide d'un ACK win 0. Le récepteur notifie son redémarrage par ACK win#0. Si une perte de cette information se produit, la connexion se trouve alors dans un état bloqué, l'émetteur étant en attente de la notification ACK win du récepteur.

L'émetteur utilise un *timer persistant* pour interroger périodiquement le récepteur endormi (puisque win=0) Les segments en provenance de l'émetteur sont appelés sondes de fenêtres (*windows probes*). Une sonde de fenêtre est un paquet TCP qui contient ***un*** octet de donné (#seq). Le récepteur de ce segment répond par un ACK(#seq) win=0 si sa fenêtre de réception est pleine (une bonne réception impliquerait ACK(#seq+1).

Le premier timeout est calculé à 1.5s pour une connexion LAN typique. Le repli exponentiel (1.5*1, 2, 4, 8...) est appliqué jusqu'à atteindre une valeur de 60 secondes qui reste stable. Une connexion TCP ne renonce ***jamais*** à l'envoi de sondes de fenêtre.

Le syndrome de la fenêtre stupide (*silly window syndrome* SWS), consiste à échanger de petits volumes de données au lieu d'utiliser des segments de pleine taille.

Les deux extrémités d'une connexion évitent le SWS au moyen des règles suivantes:

- Le récepteur ne doit pas annoncer de petits segments, de préférence il choisit un segment de pleine taille(MSS) ou la moitié de son buffer de réception.
- L'émetteur peut attendre un segment MSS de données, ou une moitié de la fenêtre annoncée par l'autre extrémité.

2.6.8 Le timer Keepalive.

Une connexion TCP peut rester inactive c'est à dire que plus aucun paquet TCP n'est échangé entre les deux extrémités de la liaison. Une sonde *keepalive* est constituée d'un segment dont le numéro de séquence est inférieur de un à la valeur attendue par le récepteur (#seq-1), et de zéro donnée. Sa réception provoque l'émission d'un ACK(#seq).

L'utilisation du *timer keepalive* est optionnelle.

2.7 Traces d'une requête HTTP

```
netscape 129.182.52.66
http://129.182.52.206/cgi-dos-add?

length= 60 IPseq#14224
129.182.52.66:1045 => 129.182.52.206:80

seq#= 2222325760 ack#= 1 win= 4096 SYN
02 60 8C 40 C1 20 00 00 C0 22 CB AE 08 00 45 00
00 2C 37 90 00 00 3C 06 DA BF 81 B6 34 42 81 B6
34 CE 04 15 00 50 84 76 00 00 00 00 00 01 60 02
10 00 92 CD 00 00 02 04 05 B4 00 00
length= 60 IPseq#29918
129.182.52.206:80 => 129.182.52.66:1045
seq#= 630019433 ack#= 2222325761 win= 8712 SYN ACK
00 00 C0 22 CB AE 02 60 8C 40 C1 20 08 00 45 00
00 2C 74 DE 40 00 20 06 79 71 81 B6 34 CE 81 B6
34 42 00 50 04 15 25 8D 55 69 84 76 00 01 60 12
22 08 05 C7 00 00 02 04 05 AC 20 20
length= 60 IPseq#14225
129.182.52.66:1045 => 129.182.52.206:80
seq#= 2222325761 ack#= 630019434 win= 4096 ACK
02 60 8C 40 C1 20 00 00 C0 22 CB AE 08 00 45 00
00 28 37 91 00 00 3C 06 DA C2 81 B6 34 42 81 B6
34 CE 04 15 00 50 84 76 00 01 25 8D 55 6A 50 10
10 00 2F 84 00 00 02 04 05 AC 20 20
length= 282 IPseq#14226
129.182.52.66:1045 => 129.182.52.206:80
seq#= 2222325761 ack#= 630019434 win= 4096 PSH ACK
02 60 8C 40 C1 20 00 00 C0 22 CB AE 08 00 45 00
01 0C 37 92 00 00 3C 06 D9 DD 81 B6 34 42 81 B6
34 CE 04 15 00 50 84 76 00 01 25 8D 55 6A 50 18
10 00 8A A4 00 00 47 45 54 20 2F 63 67 69 2D 64
6F 73 2D 61 64 64 3F 20 48 54 54 50 2F 31 2E 30
0D 0A 52 65 66 65 72 65 72 3A 20 68 74 74 70 3A
2F 2F 31 32 39 2E 31 38 32 2E 35 32 2E 32 30 36
2F 63 67 69 2D 64 6F 73 2D 61 64 64 3F 0D 0A 43
length= 60 IPseq#30174
129.182.52.206:80 => 129.182.52.66:1045
seq#= 630019434 ack#= 2222325989 win= 8484 ACK
00 00 C0 22 CB AE 02 60 8C 40 C1 20 08 00 45 00
00 28 75 DE 40 00 20 06 78 75 81 B6 34 CE 81 B6
34 42 00 50 04 15 25 8D 55 6A 84 76 00 E5 50 10
21 24 1D 7C 00 00 20 20 20 20 20 20
length= 254 IPseq#30430
129.182.52.206:80 => 129.182.52.66:1045
seq#= 630019434 ack#= 2222325989 win= 8484 PSH ACK
```

```
00 00 C0 22 CB AE 02 60 8C 40 C1 20 08 00 45 00
00 F0 76 DE 40 00 20 06 76 AD 81 B6 34 CE 81 B6
34 42 00 50 04 15 25 8D 55 6A 84 76 00 E5 50 18
21 24 86 41 00 00 48 54 54 50 2F 31 2E 30 20 32
30 30 20 4F 4B 0D 0A 53 65 72 76 65 72 3A 20 57
73 70 6C 75 67 2F 33 2E 30 0D 0A 43 6F 6E 74 65
6E 74 2D 54 79 70 65 3A 20 74 65 78 74 2F 68 74
6D 6C 0D 0A 0D 0A 3C 68 74 6D 6C 3E 0D 0A 3C 68
length= 60 IPseq#14227
129.182.52.66:1045 => 129.182.52.206:80
seq#= 2222325989 ack#= 630019634 win= 3896 ACK
02 60 8C 40 C1 20 00 00 C0 22 CB AE 08 00 45 00
00 28 37 93 00 00 3C 06 DA C0 81 B6 34 42 81 B6
34 CE 04 15 00 50 84 76 00 E5 25 8D 56 32 50 10
0F 38 2E A0 00 00 48 54 54 50 2F 31
length= 89 IPseq#30686
129.182.52.206:80 => 129.182.52.66:1045
seq#= 630019634 ack#= 2222325989 win= 8484 PSH ACK
00 00 C0 22 CB AE 02 60 8C 40 C1 20 08 00 45 00
00 4B 77 DE 40 00 20 06 76 52 81 B6 34 CE 81 B6
34 42 00 50 04 15 25 8D 56 32 84 76 00 E5 50 18
21 24 74 34 00 00 75 62 6D 69 74 3E 0D 0A 3C 48
52 3E 0D 0A 57 73 70 6C 75 67 3C 2F 62 6F 64 79
3E 3C 2F 68 74 6D 6C 3E 1A
length= 60 IPseq#30942
129.182.52.206:80 => 129.182.52.66:1045
seq#= 630019669 ack#= 2222325989 win= 8484 FIN ACK
00 00 C0 22 CB AE 02 60 8C 40 C1 20 08 00 45 00
00 28 78 DE 40 00 20 06 75 75 81 B6 34 CE 81 B6
34 42 00 50 04 15 25 8D 56 55 84 76 00 E5 50 11
21 24 1C 90 00 00 20 20 20 20 20 20
length= 60 IPseq#14228
129.182.52.66:1045 => 129.182.52.206:80
seq#= 2222325989 ack#= 630019670 win= 3861 ACK
02 60 8C 40 C1 20 00 00 C0 22 CB AE 08 00 45 00
00 28 37 94 00 00 3C 06 DA BF 81 B6 34 42 81 B6
34 CE 04 15 00 50 84 76 00 E5 25 8D 56 56 50 10
0F 15 2E 9F 00 00 20 20 20 20 20 20
length= 60 IPseq#14229
129.182.52.66:1045 => 129.182.52.206:80
seq#= 2222325989 ack#= 0 win= 0 RST
02 60 8C 40 C1 20 00 00 C0 22 CB AE 08 00 45 00
00 28 37 95 00 00 3C 06 DA BE 81 B6 34 42 81 B6
34 CE 04 15 00 50 84 76 00 E5 00 00 00 00 50 04
00 00 B9 A3 00 00 20 20 20 20 20 20

Fin de la requête http.
```

3. L'interface Socket.

3.1 Introduction.

Les primitives d'entré sortie Unix (IO) suivent un paradigme parfois nommé open-read-write-close. L'appel de la fonction open retourne un entier appelé descripteur de fichier (*file descriptor*). Le process utilise les fonctions read et write pour lire ou écrire des données. Enfin l'utilisateur notifie à l'Operating System l'abandon de ces opérations d'IO par la primitive close.

Depuis l'origine les concepteurs d'Unix ont attribués des noms de fichiers réservés qui permettent de manipuler des objets de type terminaux par exemple «/dev/tty».
Le paradigme des sockets consiste à ouvrir un socket (primitive socket()) dans le cas d'un datagramme le socket est lié (*binding*) à une adresse de destination à chaque utilisation, dans le cas d'un mode connecté (TCP) le lien est fixe. Une fois le socket crée l'utilisateur utilise les commandes traditionnelles read et write pour émettre/recevoir de l'information. La fonction close libère toutes les ressources préalablement allouées au socket.

Il existe une bibliothèque de sockets sous windows connue sous le nom de Windows Socket (associée à la DLL winsock.dll) et formellement standardisée en 1993 (winsock version 1.1). Une version 2 est wsock32.dll.

3.2 La bibliothèque des sockets.

3.2.1 La création d'un socket, socket()

int socket(int AdressFamily, int Type, int Protocol)

Cette fonction crée un socket.

AdressFamily (af) indique la famille de socket utilisée. Dans le cas qui nous interesse, à savoir Internet Protocol la valeur de ce paramètre sera toujours **AF_INET** (AdressFamily == INternET), mais d'autres familles existent telles que Xerox Corporation PUP internet (AF_PUP), Apple Computer Incorporated Appletalk network (AF-APPLETALK) ou system de fichier UNIX (AF_UNIX).
Type indique le type de protocole utilisé « au dessus » de IP.

- ***SOCK_STREAM***, un protocole connecté, TCP en fait
- ***SOCK_DGRAM***, un protocole non connecté, en mode datagramme, UDP en fait.
- ***SOCK_RAW***, un protocole encapsulé par IP tel qu'ICMP par exemple.

Protocol identifie le protocole particulier mis en oeuvre par le socket, 0 indique un protocole par défaut, 0 sera utilisé les types autres que SOCK_RAW.

La valeur retournée est **-1** dans le cas d'un échec de la création du socket.

Exemples
int s;
s = socket(AF_INET,SOCK_STREAM,0), création d'un socket TCP
s = socket(AF_INET,SOCK_DGRAM,0), création d'un socket UDP
s = socket(AF_INET,SOCK_RAW,IPPROTO_ICMP), création d'un socket ICMP
(1<=>IPPROTO_ICMP).
if (s == -1) perror(«Socket Error »);

3.2.2 La fermeture d'un socket, close()

close(int s).

Cette fonction ferme un socket s. Sous Unix la fermeture d'un process pour quelque raison que ce soit ***ferme*** tous les sockets ouverts.

3.2.3 Adresse locale.

Un socket est créé sans association à aucune adresse locale ou distante. Une adresse de socket est définie de la manière suivante:

```
struct in_addr
        {
         u_long s_addr ;
        };
in_addr est l'adresse IP définie sur 32 bits
struct sockaddr_in
        {
        short sin_family
        u_short sin_port
        struct in_addr sin_addr
        char sin_zero[8]
        }
```

L'adresse d'un socket est donc large de 16 octets. De fait dans le cas d'internet (sin_familly = AF_INET) l'adresse d'un socket comporte un sin_port (n° du port de protocole) et une adresse IP (sin_addr 4 octets), c'est à dire que seulement les 8 premiers octets de la structure sont utilisés.

La définition d'une adresse consiste donc typiquement en :
struct sock_addr_in servername ;
servername.sin_family = AF_INET
serveurname.sin_addr = inet_addr(«129.5.24.1»)
servername.sin_port = htons(1024)

inet_addr est une fonction qui convertit une chaîne en une adresse IP.

htons est une fonction qui convertit un mot de 16 bits «host» en un mot de 16 bits network (croisement des octets de poids faible et fort si nécessaire).

On pourrait utiliser également:
serveurname.sin_addr = INADDR_ANY, pour indiquer que l'on utilise toutes les adresses IP de la machine hôte

3.2.4 Allouer une adresse à un socket, bind()

int bind (int s, struct sockaddr *my_addr,int adrlen)

```
struct sockaddr {
                u_short sa_family ;
                char sa_data[14]  ;
            }
```

sockaddr est une structure de 8 octets, qui contient en fait l'image des 8 premiers octets de la structure sockaddr_in.
s est un descripteur de socket.
adrlen est la longueur de la structure sockaddr (sizeof(sockaddr) = 8).
La fonction bind lie un socket à une adresse locale.
La valeur retournée est zéro en cas de succès et -1 en cas d'erreur. Sous UNIX l'erreur est notée par la variable errno.

EADDRINUSE
EADDRNOTAVAIL
AFNOSUPPORT
ENOTSOCK
...

Exemple:

```
struct sock_addr_in servername ;
servername.sin_family = AF_INET
serveurname.sin_addr = INADDR_ANY
servername.sin_port    = htons(1024)

err = bind(Socket,(struct sockaddr *)&servername,sizeof(struct sockaddr))
```

3.2.5 Identification d'un socket qui reçoit des connexions, listen()

int listen(int Socket, int Backlog)

La primitive listen identifie un socket qui peut accepter des connexions (TCP). Le nombre de connexion en attente est limité à Backlog.
SOMAXCONN fixe la valeur maximale de Backlog.
La valeur retournée est zéro en cas de succès ou -1. Sous UNIX la variable errno mémorise le type d'erreur.

EBADF (paramètre Socket non valide)
ECONNREFUSED (service refusé, valeur de BackLog erronée)
ENOTSOCK (Socket est un descripteur de fichier)
EOPNOTSUPP (le Socket référencé ne supporte pas ce type de service).

3.2.6 Socket Bloquant ou Non Bloquant.

Un socket est dit bloquant lorsque certaines opérations sont bloquantes, c'est à dire que le socket reste bloqué jusqu'à la réalisation du service demandé. Par défaut les sockets sont bloquants. Un socket non bloquant signale à son utilisateur que le service demandé n'est pas encore réalisé, l'utilisateur doit alors procéder à des interrogations périodiques (technique dite de *polling*) pour connaître l'état d'avancement de sa requête.

Exemple, réception de donnée
Bloquant
nbr = recv(...), l'utilisateur est bloqué en attente de la réception de données.
Non Bloquant
err = recv()
if ((err == -1) && (errno == EWOULDBLOCK)) pas de données en attente.

3.2.7 Socket Asynchrone et Synchrone.

Par défaut un socket est synchrone. Sous UNIX un socket Asynchrone signale la possibilité d'une opération IO (entrée/sortie) par l'émission d'un signal SIGIO. Cette notion est étendue par les winsockets (windows) asynchrone qui émettent différents messages indiquant la disponibilité d'IO ou des connexions ou déconnexions.

3.2.8 Définir les options d'un socket, setsockopt()

int setsockopt(int Socket, int Level, int OptionName, char *OptionValue, int OptionLength).

Level, définit le protocole utilisé pour le contrôle de l'option (SOL_SOCKET en règle générale).
OptionName: le nom de l'option
Option Value, un pointeur sur la valeur d'option.
OptionLength, la longueur de l'option en octets.
La valeur retournée est zéro en cas de succès, -1 indique une erreur dont le type est dans errno.

En règle générale la mise en/hors service d'une option s'écrit
int err,onoff, onoff = 0 (FAUX) ou 1 VRAI
err = setsockopt(Socket,SOL_SOCKET,(char*)&onoff,sizeoff(int))

Principales options

SO_BROADCAST, autorise les messages broadcast.
SO_DONTROUTE, ignore la procédure de routage usuelle.
SO_KEEPALIVE, la procédure keepalive de TCP
SO_LINGER
    struct linger l ;
    l.l_onoff = 0 ou 1
    l.l_linger = un temps exprimé en secondes.
Cette option permet de vider le buffer d'émission en un temps limité, lorsque la primitive close est invoqué. L'appel à close est dans ce cas bloquant.
SO_OOBINLINE, autorise la réception de données hors bande, c'est à dire la réception de données urgentes de TCP.
SO_RCVBUF, fixe la taille du buffer de réception (en octets).
SO_RCVLOWAT, fixe la marque d'eau inférieure du récepteur (en octets)
SO_RCVTIMEO, fixe le timeout de la réception (usage ?)
SO_SNDBUF, fixe la taille du buffer de réception (en octets)
SO_SNDLOWAT, fixe la marque d'eau inférieure de l'émetteur (en octets)
SO_SNDTIMEO, fixe le timeout en émission (usage ?)

NB:
L'émission est bloquante entre la marque haute et SO_SNDLOWAT.
La réception est bloquante entre la marque haute et SO_RCVLOWAT.

3.2.9 L'utilisation d'ioctl()

Certaines options des sockets sont activées/désactivées au moyen de la fonction ioctl, c'est le cas en particulier de la propriété bloquante ou de la sélection du mode asynchrone (sous UNIX seulement). ioctl s'applique de manière plus générale à tout descripteur de fichier (File Descriptor).

int ioctl(int Socket, int op, char *arg)

Socket, descripteur de socket
op, numéro de l'option
arg, un pointeur sur les paramètres de l'option.
La valeur retournée est zéro en cas de succès ou -1 en cas d'erreur. Sous Unix le type d'erreur est mémorisé par errno.
    ENOTSOCK
    EINVAL

EOPNOTSUPP
EFAULT

int flag;
mode Asynchrone ;flag=1ioctl(Socket,**FIOASYNC**,&flag).
mode Synchrone ; flag=0; ioctl(Socket,**FIOASYNC**,&flag).

mode Non Bloquant;flag=1ioctl(Socket,**FIONBIO**,&flag).
mode Bloquant; flag=0; ioctl(Socket,**FIONBIO**,&flag).
Une autre méthode consiste à utiliser
NON bloquant : fcntl(Socket,F_SETFL,O_NONBLOCK)

Connaître le nombre d'octets reçus.
int nbytes;
ioctl(Socket,**FIONREAD**,&nbytes)

3.2.10 Obtenir l'état des options d'un socket, getsockopt()

int getsockopt(int Socket, int Level, int OptionName, char *OptionValue, int *OptionLength)

Socket, un descripteur de socket
Level, généralement égal à SOL_SOCKET
OptionName, le numéro de l'option
OptionValue, un pointeur sur la valeur de l'option
OptionLength, un pointeur sur la longueur de l'option.

La valeur retournée est zéro en cas de succès et -1 en cas d'échec. Sous Unix errno mémorise le type de l'erreur.

3.2.11 Le traitement des connexions entrantes, accept()

int accept(int Socket, struct sockaddr *Address, int *AddressLength)

Cette fonction est bloquante (si le socket est bloquant)

La valeur retournée est positive ou nulle en cas de succès (c'est un descripteur de socket) et est égale à -1 en cas d'échec. Dans ce cas errno contient la cause du problème, attention le cas d'un socket non bloquant est particulier.
if ((err==-1) && (errno==EWOULDBLOCK)) => en attente de connexion.

Lorsqu'une connexion TCP est établie la structure sockaddr contient l'adresse de l'appelant (adresse IP et port TCP). Le contenu de AddressLength sera égal à 8 (sizeof(struct sockaddr)). Accept() crée un socket et le lie à l'adresse distante.

3.2.12 Création d'un serveur TCP

Les étapes à suivre sont donc:
- socket() => création d'un socket
- bind()=> le socket est lié à l'adresse de la machine hôte
- listen()=>le socket est déclaré de type serveur TCP
- client=accept()=> attente de connexion client
- recv(),send,read(),write()
- close(client)=>le client est déconnecté.

Exemple

```
#include <stdio.h>
#include <stdlib.h>
```

```
#include <types.h>
#include <netinet\in.h>
#include <sys\socket.h>
...
   unsigned short port;
   char buf[12];
   struct sockaddr_in client;
   struct sockaddr_in server;
   int s,ns,namelen;

if ((s = socket(AF_INET, SOCK_STREAM, 0)) < 0)
   {
      perror("Socket()");
      exit(2);
   }

   /*
    * Bind the socket to the server address.
    */
   server.sin_family = AF_INET;
   server.sin_port   = htons(port);
   server.sin_addr.s_addr = INADDR_ANY;

   if (bind(s, (struct sockaddr *)&server, sizeof(server)) < 0)
   {
     perror("Bind()");
     exit(3);
   }

   /*
    * Listen for connections. Specify the backlog as 1.
    */
   if (listen(s, 1) != 0)
   {
      perror("Listen()");
      exit(4);
   }

   /*
    * Accept a connection.
    */
   namelen = sizeof(client);
   if ((ns = accept(s, (struct sockaddr *)&client, &namelen)) == -1)
   {
      perror("Accept()");
      exit(5);
   }

   /*
    * Receive the message on the newly connected socket.
    */
   if (recv(ns, buf, sizeof(buf), 0) == -1)
   {
      perror("Recv()");
      exit(6);
   }

   /*
    * Send the message back to the client.
    */
```

```
    if (send(ns, buf, sizeof(buf), 0) < 0)
    {
        perror("Send()");
        exit(7);
    }

    close(ns);
    close(s);

    printf("Server ended successfully\n");
    exit(0);
}
```

3.2.13 Fermeture ou semi fermeture d'une connexion TCP, shutdown()

int shutdown(int Socket, int How)

Socket, un descripteur de socket
How, 0=arrêt de réception, 1=arrêt d'émission, 2=arrêt d'émission et de réception
retourne la valeur 0 en cas de succès et -1 en cas d'échec, errno contient alors le type de l'erreur.

Le socket n'est pas fermé par shutdown, il faut utiliser close() ultérieurement.

Exemple: shutdown(s,2);

3.2.14 Ouverture d'un serveur UDP.

Dans le cas du protocole UDP il n'existe pas de notion de connexion. Les primitives listen() et accept() sont donc sans objet.

La séquence des opérations à effectuer est donc la suivante:
socket() => création d'un socket.
bind() => lier l'adresse de la machine hôte au socket.
recvfrom(), sendto() => envoi réception de données.
close() =>fermeture du socket

Exemple :

```
int sockint,s, namelen, client_address_size;
  struct sockaddr_in client, server;
  char buf[32];

f ((s = socket(AF_INET, SOCK_DGRAM, 0)) < 0)
  {
     perror("socket()");
     exit(1);
  }

  server.sin_family         = AF_INET;       /* Server is in Internet Domain */
  server.sin_port           = 0;             /* Use any available port       */
  server.sin_addr.s_addr = INADDR_ANY;/* Server's Internet Address     */

  if (bind(s, (struct sockaddr *)&server, sizeof(server)) < 0)
  {
     perror("bind()");
     exit(2);
  }

  /* Find out what port was really assigned and print it */
  namelen = sizeof(server);
```

```
  if (getsockname(s, (struct sockaddr *) &server, &namelen) < 0)
  {
     perror("getsockname()");
     exit(3);
  }

  printf("Port assigned is %u\n", ntohs(server.sin_port));

client_address_size = sizeof(client);

  if(recvfrom(s, buf, sizeof(buf), 0, (struct sockaddr *)&client,\
                &client_address_size)<0)
  {
     perror("recvfrom()");
     exit(4);
  }
printf("Received message %s from domain %s port %d internet address %s\n",
     buf,
     (client.sin_family == AF_INET?"AF_INET":"UNKNOWN"),
     ntohs(client.sin_port),
     inet_ntoa(client.sin_addr));

close(s);
```

3.2.15 La connexion de clients connect().

int connect(int Socket, struct sockaddr *Name, int NameLength)

Retourne la valeur zéro en cas de succès, -1 en cas d'erreur. Le type de l'erreur est alors mémorisé dans errno.
Dans le cas d'une connexion TCP cette fonction est bloquante.
err = connect(s,(struct sockaddr *)&server,sizeof(server))
if ((err == -1) && (errno==EINPROGRESS)) => connexion en cours

EBDAF
ENOTSOCK
EADDRNOTAVAIL
EAFNOSUPPORT
EISCONN
ETIMEOUT
ECONNREFUSED
ENETUNREACH
EFAULT
EINPROGRESS
EINVAL

Cette primitive réalise une connexion entre deux sockets. Si le socket associé est de type SOCK_DGRAM connect réalise une connexion TCP. Si le socket est de type SOCK_STREAM connect associe une adresse distante au socket.

3.2.16 Exemple de client TCP

La connexion d'un client TCP comporte les étapes suivantes:
socket() => création d'un socket
connect() => connexion TCP au serveur
recv,send(),read(),write()
close() => fermeture du socket

Exemple:

```
   unsigned short port;
```

```
   char buf[12];
   struct hostent *hostnm;
   struct sockaddr_in server;
   int s;

   hostnm = gethostbyname("machine.domaine.fr");
   if (hostnm == (struct hostent *) 0)
   {
      fprintf(stderr, windows probes "Gethostbyname failed\n");
      exit(2);
   }

port = (unsigned short) atoi("1024");

strcpy(buf, "the message");

   server.sin_family      = AF_INET;
   server.sin_port        = htons(port);
   server.sin_addr.s_addr = *((unsigned long *)hostnm->h_addr);

if ((s = socket(PF_INET, SOCK_STREAM, 0)) < 0)
   {
      perror("Socket()");
      exit(3);
   }

if (connect(s, (struct sockaddr *)&server, sizeof(server)) < 0)
   {
      perror("Connect()");
      exit(4);
   }

   if (send(s, buf, sizeof(buf), 0) < 0)
   {
      perror("Send()");
      exit(5);
   }

if (recv(s, buf, sizeof(buf), 0) < 0)
   {
      perror("Recv()");
      exit(6);
   }

close(s);

   printf("Client Ended Successfully\n");
   exit(0);
```

3.2.17 Exemple de client UDP.

Les opérations suivantes sont effectuées:

socket() => création d'un socket datagramme
sendto(), recvfrom()
close()

Exemple:

```
int s;
  unsigned short port;
```

```
  struct sockaddr_in server;
  char buf[32];

port = htons(atoi(« 1024 »));

if ((s = socket(AF_INET, SOCK_DGRAM, 0)) < 0)
  {
     perror("socket()");
     exit(1);
  }

  server.sin_family      = AF_INET;                    /* Internet Domain    */
  server.sin_port        = port;                       /* Server Port        */
  server.sin_addr.s_addr = inet_addr(« 129.182.70.55 »); /* Server's Address   */

  strcpy(buf, "Hello");

if (sendto(s, buf, (strlen(buf)+1), 0, (struct sockaddr *)&server, sizeof(server)) < 0)
  {
     perror("sendto()");
     exit(2);
  }

close(s);
```

3.2.18 Autres protocoles, les sockets raw, exemple ICMP (ping)

La marche à suivre est la suivante :
socket() => création d'un socket RAW
recvfrom(), sendto()
close()

Exemple:

```
if ((s = socket(AF_INET, SOCK_RAW,IPPROTO_ICMP)) < 0)
  {
     perror("socket()");
     exit(1);
  }

  icmp.sin_family      = AF_INET;
  icmp.sin_port        = 0          ;
  icmp.sin_addr.s_addr = inet_addr(« 129.182.70.55 »);

  strcpy(buf, "Hello");

if (sendto(s, buf, (strlen(buf)+1), 0, (struct sockaddr *)&icmp, sizeof(icmp)) < 0)
  {
     perror("sendto()");
     exit(2);
  }

close(s);
```

Remarque: le programme TraceRoute
Ce programme utilise l'option IP_TTL

```
in err,TimeToLive=5 ;
err = setsockopt(s,IPPROTO_IP,IP_TTL,(char*)&TimeToLive,sizeof(int));
```

3.2.19 Le mode connecté.

3.2.19.1 Réception de données, Recv()

int recv(int Socket, char *Buffer, int len, int flags)

Cette opération est bloquante. Retourne -1 en cas d'erreur, 0 en cas de déconnexion, sinon le nombre d'octets lus
err = recv
if (err == 0) => déconnexion
if ((err==-1) && (errno==EWOULDBLOCK)) =>rien à lire

flags : un OU logique des flags suivants
MSG_PEEK, les données sont lues mais non extraites du buffer de réception.
MSG_OOB, lecture de données hors bandes.
En l'absence d'option, flag est codé à zéro.

3.2.19.2 Envoi de données, Send()

int send(int Socket, char * Buffer, int len, int flags)

Cette opération est bloquante. Son comportement est identique à recv(). Les options sont différentes.
MSG_DONTROUTE
MSG_OOB.

3.2.20 Le mode non connecté.

3.2.20.1 Réception de donnés, recvfrom()

int recvfrom(int Socket, char *Buffer, int Length, int Flags, struct sockaddr *From,int *FromLength).

Cette fonction est bloquante. Son comportement est identique à recv() si ce n'est que l'adresse source est spécifiée par From (struct sockaddr).

3.2.20.2 Emission de données, sendto()

int sendto(int Socket, Char *Buffer, int Length, int Flags, struct sockaddr *To, int ToLength)

Cette opération est bloquante. Son comportement est identique à send() si ce n'est que l'adresse destination est spécifiée par To (struct sockaddr).

3.2.21 Gestion bloquantes de plusieurs sockets, select()

int select(int Nfds, fd_set *ReadFds, fd_set *WriteFds, fd_set *ExceptFds, struct timeval *Timeout).

```
struct timeval { int tv_sec ;   /* secondes        */
                 int tv_usec; /* microsecondes */
               }
```

Si Timeout est mis à NULL le timeout est ignoré. Select est bloquant et attend qu'un événement se produisent sur des Sockets et plus généralement des descripteurs de fichiers.

Valeur retournée:
- 0 si Timeout
- 0 si un socket connecté (TCP) est déconnecté, dans ce cas la déconnexion sera notifiée par une valeur de retour nulle de recv().
- 0 en case de succès
- -1 en cas d'erreur, le type de l'erreur est contenu dans errno.

Nfds, la plus grande des files descripteurs (int associé au socket) **plus** 1.

Trois files fd_set associent les descripteurs de fichiers à des événements en lecture, écriture et erreurs. Une liste Set mise à NULL n'est pas utilisée.

FD_ZERO(fd_set *set), remis à zéro la liste des descripteurs.
FD_SET(int fd, fd_set *set), ajoute fd à la liste set
FD_CLR(int fd, fd_set *set), retire un élément de la liste fd_set
FD_ISSET(int fd, fd_set *set), teste si fd se trouve dans la liste set.

Exemple:

```
int sclient1 ;
int sclient2 ;
fd_set read_fd, write_fd;

int nmfds;

while (FOREVER)
 {
   FD_ZERO(&read_fd);
   FD_SET(sclient1 &read_fd);
   FD_SET(sclient2,&read_fd);

   select (10, &read_fd, NULL, NULL, NULL);
   if (FD_ISSET (sclient1 &read_fd))
   {    if ((len = read (sclient1, buf, 2048)) <= 0) termine();
   }

   if (FD_ISSET (sclient2, &read_fd))
   {
      if ((len = read (sclient2, buf, 2048)) <= 0) termine();
   }
 }
```

3.2.22 Divers utiles

3.2.22.1 Conversions Host<=>Network

unsigned long htonl(unsigned long HostLong)
unsigned short htons(unsigned short HostShort)
unsigned long ntohl(unsigned long NetLong)
unsigned short ntohs(unsigned short NetShort)

3.2.22.2 Adresses et Noms

```
struct hostent {
                        char *h_name ; /* nom officiel */
                        char **h_aliases; / * NULL ou un pointeur sur un alias*/
                        int h_addrtype;    /* type d'adresse (AF_INET) */
                        int h_length; /* longueur de l'adresse, en octets */
                        char **h_addr_list; /* NULL ou un pointeur sur une liste
                        d'adresse, dans l'ordre octet du réseau */
                }
#define h_addr h_addr_list[0] /* la 1° adresse de la liste */
```

Equivalence nom de hôte <=> adresse IP
struct hostent * gethostbyname(char *name)
returne NULL en cas d'erreur
h->h_name, h->h_addr_list[0]

inet_ntoa(*((struct in_addr*)h->h_addr))

Obtenir son nom de machine.
int gethostname(char *Name, int NameLength)
Retourne 0 en cas de succès -1 en cas d'échec
Exemple strcpy(MonNom, » »);gethostname(MonNom,50);

Convertir une chaîne en adresse IP 32 bits
unsigned long inet_addr(char * CharString)
Convertit une chaîne analogue à une adresse IP en en mot de 32 bits.
Retourne 0xFFFFFFFF en cas d'échec.

Convertir une chaîne en adresse IP 32 bits
unsigned long inet_network(char *CharString)
retourne 0xFFFFFFFF en cas d'erreur ou concrtit une chaine analogue à une adresse IP en un mot de 32bits dans l'ordre réseau.

Convertir une Adresse IP 32 réseau en une chaîne.
char * inet_ntoa(struct in_addr InternetAdrr)
Convertit une adresse IP (32 bits en ordre réseau) en une chaîne.

Obtenir son nom de domaine
getdomainname(char *name,int length).

3.2.23 L'accès au système de serveur de noms.

L'idée générale de cette famille de fonction est de produire une requête, de l'envoyer au serveur et d'attendre la réponse.

Un programme appel res_init() avant toute autre opération.

res_mkquery() construit la requête

res_send() envoit la requête au serveur de noms et attend la réponse.

Les noms de domaines peuvent être compressés et décompressés à l'aide des fonctions dn_expand() et dn_comp().

3.3 Exemples sous UNIX

3.3.1 proxy.c

```
/*******************************
    Proxy.c (c) P.Urien
    Un proxy pour le debug
    de protocoles diverses
*******************************/

#include <stdio.h>

#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <sys/ioctl.h>
#include <sys/select.h>
#include <fcntl.h>
#include <signal.h>

#define DEBUG   1
#define FOREVER 1

char buf[2048];
```

```
struct sockaddr_in sn ;
struct sockaddr_in adr;
struct sgttyb old, new;
int lenadr  ;
int sclient ;
int sproxy  ;
fd_set read_fd, write_fd;

termine ()
{ shutdown (sclient,2);
  shutdown (sproxy,2) ;
  exit (0);
}

proxy ()
{ int len,numfds;

 sproxy = socket (AF_INET,SOCK_STREAM,0);

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

 sn.sin_port        = 80                          ;
 sn.sin_addr.s_addr = inet_addr ("192.90.70.11");
 sn.sin_family      = AF_INET                     ;

 if (connect (sproxy, &sn, lenadr) == -1)
 {  perror ("server");
    exit (1);
 }

#if DEBUG
 printf ("connect to proxy (%d)\n", sproxy);
#endif

 signal (SIGPIPE,termine); /* ferme en lecture */

 if (sclient >= sproxy) numfds = sclient+1 ;
 else                   numfds = sproxy+1  ;

 while (FOREVER)
 {
    FD_ZERO (&read_fd);
    FD_SET  (sclient, &read_fd);
    FD_SET  (sproxy , &read_fd);

    select ( 10, &read_fd, NULL, NULL, NULL);
    if (FD_ISSET (sclient, &read_fd))
    {    if ((len = read (sclient, buf, 2048)) <= 0) termine();
         write (sproxy, buf,len);
    }

    if (FD_ISSET (sproxy, &read_fd))
    {
        if ((len = read (sproxy, buf, 2048)) <= 0) termine();
        write (sclient, buf, len);
    }
  }
 }

main ()
{
```

```
 int serv   ;
 int on = -1;

 #if DEBUG
 printf ("Proxy is running\n");
 #endif

signal (SIGCLD, SIG_IGN) ;    /* SIGCL == Child End, SIG_IGN == Ignore */

lenadr = sizeof (struct sockaddr_in);

 serv =socket (AF_INET, SOCK_STREAM,0);
 if (!serv)
 {
    perror ("socket");
    exit (1);
 }
 sn.sin_port         = 6666      ;
 sn.sin_addr. s_addr = INADDR_ANY;
 sn.sin_family       = AF_INET   ;

 bind (serv,&sn, sizeof (sn));

 listen (serv,5);

 /*
 close(0) ;
 close(1) ;
 close(2) ;
 */

  /*
  This function is implemented to support job control.

  The setpgrp subroutine in the libc.a library supports a subset of
  the function of the setpgid subroutine.  It has no parameters.
  It sets the  process  group  ID  of the calling process to be the
  same as its process ID and returns the new value.
  */

  setpgrp();

 while (FOREVER)
 {
    lenadr  = sizeof (struct sockaddr_in)  ;
    sclient = accept (serv, &adr, &lenadr) ;

#if DEBUG
    printf ("accept client (%d)\n", sclient);
#endif
    switch (fork ())
    {
      case 0:
       close (serv)   ;
       proxy (sclient);
       break          ;

      default:
       close (sclient);
       break;
    }
}

shutdown(serv) ;
}
```

```
/* End Of Proxy.c Program */
```

3.3.2 Server.c

```
/*****************************
  Serveur.c
  Un serveur qui fork
  non bloquant
  avec signaux (c) P.Urien
*****************************/

#include <stdio.h>
#include <stdlib.h>

#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <sys/ioctl.h>
#include <sys/select.h>
#include <sys/errno.h>
#include <fcntl.h>
#include <signal.h>

#define DEBUG   1
#define FOREVER 1

char buf[2048],buf2[4096];

struct sockaddr_in sn ;
struct sockaddr_in adr;
struct sgttyb old, new;
int lenadr  ;
int sclient ;
int ttydes=NULL      ;
unsigned long rxct=0 ;
extern int errno     ;

fd_set read_fd, write_fd;

int csignal=0;

pret()
{ int rc,flag=0;
  printf("IO is ready\n");
  csignal=1 ;
}

termine ()
{ shutdown (sclient,2) ;
  printf("%lu bytes received\n",rxct) ;
  exit (0);
}

/*------------------------------
              Exemple
EWOULDBLOCK..
IO is ready
Received byte =[512]
Received byte =[512]
Received byte =[512]
Received byte =[512]
Received byte =[512]
Received byte =[512]
Received byte =[512]
```

```
Received byte =[45]
Received byte =[-1]
EWOULDBLOCK..
Timeout !
Received byte =[0]
IO is ready
9773 bytes received
-----------------------------------*/

client()
{int len,len2,i,j,err        ;
 int rxsize=512,rc           ;
 int owner,noblock=1,flag=1  ;
 long tct                    ;

 /* owner = getpgrp(); */
    owner = getpid();

rc= setsockopt(sclient,SOL_SOCKET,SO_RCVBUF,&rxsize,sizeof(rxsize));
if (rc == 0) printf("RCVBUF OK\n");


rc = ioctl(sclient,FIOASYNC,&flag);
if (rc < 0)
 {  perror ("ioctl error");
    termine() ;
 }

rc = ioctl(sclient,FIOSETOWN,&owner);
if (rc < 0)
 {  perror ("ioctl error");
    termine() ;
 }



/* fcntl(sclient,F_SETFL,O_NONBLOCK) ; */
rc = ioctl(sclient,FIONBIO,&noblock);
if (rc < 0)
 {  perror ("ioctl error");
    termine() ;
 }

signal (SIGPIPE,termine); /* ferme en lecture */

 while (FOREVER)
 {

    len2=1024 ;

    if (len2 >0)
    { len  = recv(sclient,buf,len2,0)    ;
      printf("Received byte =[%d]\n",len);
      if (len == 0) termine();
      if (len == -1)
      { err = errno                      ; /* BLOCK or error */
        if (err ==  EWOULDBLOCK )
        { printf("EWOULDBLOCK..\n");
          csignal=0                ;
          signal (SIGIO,pret)      ;
          len  = ioctl(sclient,FIONREAD,&len2,0) ;
          if (len == -1) termine()               ;

          tct=0;
          while((!csignal) && (len2==0) )
```

```
          {tct++;  /* SIGIO n'est pas émis sur deconnexion du distant !*/
           if (tct > 100000L) { printf("Timeout !\n") ; csignal=1 ;}
          }
          len=0   ;
        }
        else  termine()     ;
      }
      rxct += (unsigned long) len ;

    }


    len2=0;
    j=0;
    while(len2 > 0)
    { len  = send(sclient,&buf2[j],len2,0) ;
      if (len == -1)
      { err = errno                 ;
        if    (err == EWOULDBLOCK ) {signal (SIGIO,pret);pause() ;} /* no
data sent*/
        else  termine()             ;
      }
      if (len>0) { j+=len;len2 -= len ;}
    }


  }
}

main ()
{
 int serv   ;
 int on = -1;
 int rxsize=512,rc ;

 #if DEBUG
 printf ("Serveur is running\n");
 #endif

signal (SIGCLD, SIG_IGN) ;    /* SIGCL == Child End, SIG_IGN == Ignore */

lenadr = sizeof (struct sockaddr_in);

 serv =socket (AF_INET, SOCK_STREAM,0);
 if (!serv)
 {
    perror ("socket");
    exit (1);
 }
 sn.sin_port         = 6666      ;
 sn.sin_addr. s_addr = INADDR_ANY;
 sn.sin_family       = AF_INET   ;

 rc= setsockopt(serv,SOL_SOCKET,SO_RCVBUF,&rxsize,sizeof(rxsize));
 if (rc == 0) printf("RCVBUF OK\n");

 bind (serv,&sn, sizeof (sn));

 listen (serv,5);

 /*
 close(0) ;
 close(1) ;
 close(2) ;
 */
```

```
  /*
  This function is implemented to support job control.

  The setpgrp subroutine in the libc.a library supports a subset of
  the function of the setpgid subroutine.  It has no parameters.
  It sets the  process  group  ID  of the calling process to be the
  same as its process ID and returns the new value.
  */

  setpgrp();

 while (FOREVER)
 {
    lenadr  = sizeof (struct sockaddr_in)  ;
    sclient = accept (serv, &adr, &lenadr) ;

#if DEBUG
    printf ("accept client (%d)\n", sclient);
#endif
    switch (fork ())
    {
      case 0:
       close  (serv)   ;
       client ()       ;
       break           ;

      default:
       close (sclient);
       break;
    }
}

shutdown(serv) ;

}

/****************************/
/* end of serveur.c program */
/****************************/
```

3.3.3 Server1.c

```
/*****************************
   serveur1.c
   un serveur qui fork
   et qui utilise select
*****************************/
#include <stdio.h>
#include <stdlib.h>


#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <sys/ioctl.h>
#include <sys/select.h>
#include <sys/errno.h>
#include <fcntl.h>
#include <signal.h>

#define DEBUG   1
#define FOREVER 1

char buf[2048],buf2[4096];
```

```
struct sockaddr_in sn ;
struct sockaddr_in adr;
struct sgttyb old, new;
int lenadr  ;
int sclient ;
int ttydes=NULL      ;
unsigned long rxct=0 ;
extern int errno     ;

fd_set read_fd, write_fd;

termine ()
{ shutdown (sclient,2) ;
  printf("%lu bytes received\n",rxct) ;
  exit (0);
}

client()
{int len,len2,i,j,err ;

 signal (SIGPIPE,termine); /* ferme en lecture */

 while (FOREVER)
 {
    FD_ZERO (&read_fd)         ;
    FD_SET  (sclient, &read_fd);
    FD_SET  (0,       &read_fd);

    select (sclient+1,&read_fd, NULL, NULL, NULL);

    if (FD_ISSET (sclient, &read_fd))
    {    if ((len = read (sclient, buf, 2048)) <= 0) termine();
         buf[len]=0;
         write(1,buf,len) ;
    }

    if (FD_ISSET (0, &read_fd))
    {
        if ((len = read (0,buf,2048)) <= 0) termine();
        buf[len] = 0;
        rxct += (unsigned long) len ;
        for(i=0;i<len;i++)
        {
           if (buf[i] != '\n') write (sclient, &buf[i], 1) ;
           if (buf[i] == '\n') write (sclient, "\r\n", 2) ;
        }

    }

  }


}

main ()
{
 int serv   ;
 int on = -1;

 #if DEBUG
 printf ("Serveur is running\n");
 #endif

signal (SIGCLD, SIG_IGN) ;    /* SIGCL == Child End, SIG_IGN == Ignore */
```

```
lenadr = sizeof (struct sockaddr_in);

 serv =socket (AF_INET, SOCK_STREAM,0);
 if (!serv)
 {
    perror ("socket");
    exit (1);
 }
 sn.sin_port         = 6666      ;
 sn.sin_addr. s_addr = INADDR_ANY;
 sn.sin_family       = AF_INET   ;

 bind (serv,&sn, sizeof (sn));

 listen (serv,5);

 /*
 close(0) ;
 close(1) ;
 close(2) ;
 */

  /*
  This function is implemented to support job control.

  The setpgrp subroutine in the libc.a library supports a subset of
  the function of the setpgid subroutine.  It has no parameters.
  It sets the  process  group  ID  of the calling process to be the
  same as its process ID and returns the new value.
  */

  setpgrp();

 while (FOREVER)
 {
    lenadr  = sizeof (struct sockaddr_in)  ;
    sclient = accept (serv, &adr, &lenadr) ;

#if DEBUG
    printf ("accept client (%d)\n", sclient);
#endif
    switch (fork ())
    {
      case 0:
       close  (serv)   ;
       client ()       ;
       break           ;

      default:
       close (sclient);
       break;
    }
}

shutdown(serv) ;

}

/****************************/
/* End Of Program serveur1.c */
/****************************/
```

3.3.4 Client1.c

```
/*********************
   client1.c
   un client à base
   de select()
*********************/
#include <stdio.h>
#include <stdlib.h>

#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <sys/ioctl.h>
#include <sys/select.h>
#include <fcntl.h>
#include <signal.h>

#define DEBUG   1
#define FOREVER 1

char buf[2048];
struct sockaddr_in sn ;
struct sockaddr_in adr;
struct sgttyb old, new;
int lenadr  ;
int sclient ;
int sproxy  ;
fd_set read_fd, write_fd;

termine ()
{ shutdown (sclient,2);
  exit (0);
}

main ()
{ int lenadr,i,len  ;
  short MyPort = 25 ;

 printf ("Client is running\n");

 lenadr = sizeof (struct sockaddr_in);

 sclient = socket (AF_INET,SOCK_STREAM,0);
 if (!sclient)
 {
    perror ("socket");
    exit (1);
 }

 sn.sin_port        = htons(MyPort)               ;
 sn.sin_addr.s_addr = inet_addr ("129.182.51.227");
 sn.sin_family      = AF_INET                     ;

 if (connect (sclient, &sn, lenadr) == -1)
 {  perror ("Connect failed");
    exit (1);
 }

 signal (SIGPIPE,termine); /* ferme en lecture */

 while (FOREVER)
 {
    FD_ZERO (&read_fd)         ;
    FD_SET  (sclient, &read_fd);
```

```
    FD_SET  (0,       &read_fd);

    select (sclient+1,&read_fd, NULL, NULL, NULL);

    if (FD_ISSET (sclient, &read_fd))
    {    if ((len = read (sclient, buf, 2048)) <= 0) termine();
         buf[len]=0;
         write(1,buf,len) ;
    }

    if (FD_ISSET (0, &read_fd))
    {
        if ((len = read (0,buf,2048)) <= 0) termine();
        buf[len] = 0;
        for(i=0;i<len;i++)
        {
           if (buf[i] != '\n') write (sclient, &buf[i], 1) ;
           if (buf[i] == '\n') write (sclient, "\r\n", 2) ;
        }

    }

  }
termine();
}
/****************************/
/* End of Program client1.c */
/****************************/
```

3.3.5 Client2.c

```
/*********************
   Client2, un client
   non bloquant
*********************/
#include <stdio.h>
#include <stdlib.h>

#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <sys/ioctl.h>
#include <sys/select.h>
#include <sys/errno.h>
#include <fcntl.h>
#include <signal.h>

extern int errno ;

#define DEBUG   1
#define FOREVER 1

char buf[2048],buf2[4096];
struct sockaddr_in sn ;
struct sockaddr_in adr;
struct sgttyb old, new;
int lenadr  ;
int sclient ;
int sproxy  ;
fd_set read_fd, write_fd;

int ttydes ;

int mode = 1 ; /* 0 normal 1 raw */
```

```
termine ()
{ shutdown (sclient,2);
  close(ttydes)       ;
  if (mode) system("stty -raw ");
  exit (0);
}

main ()
{ int lenadr,i,j,len,rc,len2  ;
  short MyPort = 6666  ;
  int dontblock = 1,err;
  int more = 1;



 ttydes = open("/dev/tty",O_RDWR|O_NDELAY|O_NONBLOCK) ;
 if (ttydes == NULL)
 {  perror("Impossible d'ouvrir /dev/tty\n");
    exit(1);
 }

 printf ("Client is running\n");
 if (mode)  system("stty raw ");

 lenadr = sizeof (struct sockaddr_in);

 sclient = socket (AF_INET,SOCK_STREAM,0);

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


 sn.sin_port        = htons(MyPort)               ;
 sn.sin_addr.s_addr = inet_addr ("129.182.52.233");
 sn.sin_family      = AF_INET                     ;



 /* le connect est bloquant, la gestion d'un connect
    non bloquant est plus complexe */

 if (connect (sclient, &sn, lenadr) == -1)
 { perror("Connect failed\n") ;
   exit (1);
 }

 fcntl(sclient,F_SETFL,O_NONBLOCK) ;

 /* rc = ioctl(sclient,FIONBIO,(char*)&dontblock,sizeof(dontblock));
 if (rc < 0)
 {  perror ("ioctl error");
    exit (1);
 }
 */

 signal (SIGPIPE,termine); /* ferme en lecture */

 while (FOREVER)
 {
    len  = ioctl(sclient,FIONREAD,&len2,0) ;
    if (len == -1)
    { err = errno               ;
```

```
      termine()                  ;
    }

    if (len2 > 1000) len2=1000 ;

    if (len2 >0)
    { len  = recv(sclient,buf,len2,0)    ;
      if (len == 0) termine();
      if (len == -1)
      { err = errno                      ; /* BLOCK or error */
        if (err ==  EWOULDBLOCK )len=0   ; /* no data        */
        else  termine()                  ;
      }
    }


    if (len>0) write(ttydes,buf,len);


    if ((len = read (ttydes,buf,2048)) < 0) termine();
    buf[len] = 0;

    j=len2=0;

    for(i=0;i<len;i++)
        {  if (mode && (buf[i] == 0x3) )/* ctrl C */
           termine();


           if ( (buf[i] != '\n') && (buf[i] != '\r') )
           { buf2[j] = buf[i];j++ ; len2++;}

           if (buf[i] == '\r') { write(ttydes,"\n",1);
                                 buf2[j] = 0xD;j++;
                                 buf2[j]= 0xA ;j++;
                                 len2 +=2 ;}
           if (buf[i] == '\n') { buf2[j] = 0xD;j++;
                                 buf2[j]= 0xA ;j++;
                                 len2 +=2 ;}
        }

    j=0;
    while(len2 > 0)
    {
    len  = send(sclient,&buf2[j],len2,0) ;
    if (len == -1)
    { err = errno                 ;
      if (err == EWOULDBLOCK )    ; /* no data sent*/
      else  termine()             ;
    }
    if (len>0) { j+=len;len2 -= len ;}
    }


  }
termine();
}

/***************************/
/* End Of Program client2.c */
/***************************/
```

3.3.6 Client3.c

```
/********************
```

```
    Client3.c
  un client bloquant
*********************/
#include <stdio.h>
#include <stdlib.h>

#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <sys/ioctl.h>
#include <sys/select.h>
#include <sys/errno.h>
#include <fcntl.h>
#include <signal.h>

extern int errno ;

#define DEBUG   1
#define FOREVER 1

char buf[40000]       ;
struct sockaddr_in sn ;
struct sockaddr_in adr;
struct sgttyb old, new;
int lenadr  ;
int sclient ;
int sproxy  ;
fd_set read_fd, write_fd;

int fh = NULL;



termine ()
{ shutdown (sclient,2)      ;
  if (fh != NULL) close(fh) ;
  exit (0)                  ;
}



main ()
{ int lenadr,i,j,len,rc,len2  ;
  short MyPort = 6666  ;
  int dontblock = 1,err;
  int more = 1;
  int txsize=512     ; /* txsize de data dans les paquets TCP */
  unsigned long txct ;

 fh = open("exemple.txt",O_RDONLY) ;
 if (fh == NULL)
 {  perror("Impossible d'ouvrir exemple.txt");
    exit(1);
 }

 printf ("Client is running\n");

 lenadr = sizeof (struct sockaddr_in);
 sclient = socket (AF_INET,SOCK_STREAM,0);
 if (!sclient)
 {  perror ("socket error");
    exit (1);
 }

 sn.sin_port        = htons(MyPort)               ;
```

```
 sn.sin_addr.s_addr = inet_addr ("129.182.52.233") ;
 sn.sin_family      = AF_INET                      ;

 signal (SIGPIPE,termine); /* ferme en lecture */


 rc= setsockopt(sclient,SOL_SOCKET,SO_SNDBUF,&txsize,sizeof(txsize));
 if (rc == 0) printf("SNDBUF OK\n");

 /* le connect est bloquant */

 if (connect (sclient, &sn, lenadr) == -1)
 {   perror("Connect failed\n") ;
     close(fh);exit (1);
 }

 while (FOREVER)
 {
    len = 1024 ;

    len = read(fh,buf,len)         ;
    if (len <= 0) termine()        ;
    printf("File Bytes = %d\n",len);

    len2=len;
    j=0;

    while(len2>0)
    {
    len  = send(sclient,&buf[j],len2,0) ;
    printf("Bytes Sent =%d\n",len);

    if (len == 0)  termine();
    if (len == -1) termine();

    if (len>0)
    { txct += (unsigned long) len;
      j+=len;
      len2 -= len ;
    }

   }



  }


termine();

}

/****************************/
/* End Of Program Client3.c */
/****************************/
```

3.4 Exemples sous windows

```
#include <windows.h>
#include <stdarg.h>
#include <stdlib.h>
#include <stdio.h>
#include <winsock.h>
#include <sys/timeb.h>
#include <time.h>
// ========================
// fonctions diverses ....
//========================

int DoSetAddress(SOCKADDR_IN *sin,unsigned short port,char *host)
{sin->sin_family      = AF_INET          ;
 sin->sin_port        = htons(port)      ;
 sin->sin_addr.s_addr = inet_addr (host) ;

if (sin->sin_addr.s_addr == INADDR_NONE)
{//struct hostent FAR *phe;
PHOSTENT phe;
phe = gethostbyname (host);
if (phe == NULL) return(FALSE);
else  memcpy((char FAR *)&(sin->sin_addr),phe->h_addr,4);
if (sin->sin_addr.s_addr == INADDR_NONE) return(FALSE);
}
return(TRUE);
}

int DoConnect(SOCKET s,int temps,unsigned short port,char *host)
{int ierr;
 u_long dontblock=1L       ;
 fd_set a_fd_set           ;
 int errnum                ;
 struct timeval timeout    ;
 SOCKADDR_IN sin           ;

if( !DoSetAddress(&sin,port,host) ) return(FALSE);
ierr = ioctlsocket(s,FIONBIO,&dontblock);
if (ierr < 0) return(FALSE) ;
ierr = connect ( s, (LPSOCKADDR) &sin, sizeof (sin) );
if (ierr == SOCKET_ERROR)
{ errnum =   WSAGetLastError() ;
 if (errnum == WSAEWOULDBLOCK)
 {dontblock=0;
  ierr = ioctlsocket(s,FIONBIO,&dontblock);
  if (ierr < 0) return(FALSE);

  timeout.tv_sec  = temps ;
  timeout.tv_usec = 0L    ;
  FD_ZERO(&a_fd_set       ;
  FD_SET(s,&a_fd_set)     ;

  ierr = select (0,NULL,&a_fd_set,&a_fd_set,&timeout);
  if (ierr > 0) return(TRUE)  ;
  else
  { if (ierr == 0)                  ;
    else if (ierr == SOCKET_ERROR) ;
  }
 }
}
return(FALSE);
}
```

```
int DoBind(SOCKET s,unsigned long ip,unsigned short port)
{SOCKADDR_IN sin;
 sin.sin_family      = AF_INET       ;
 sin.sin_addr.s_addr = ip            ;
 sin.sin_port        = htons(port)   ;

 if (bind(s, (LPSOCKADDR)&sin, sizeof (sin)) == 0) return (TRUE) ;
 return(FALSE) ;
}

int DoRead(SOCKET s, char *Buf,int len, int temps)
{struct timeval timeout  ;
 int ierr,iread,more=TRUE,Pt=0  ;
 fd_set a_fd_set ;

timeout.tv_sec  = temps ;
timeout.tv_usec = 0L    ;

while(more)
{
FD_ZERO(&a_fd_set)      ;
FD_SET(s,&a_fd_set)     ;

if (temps != 0) ierr = select (s+1,&a_fd_set,NULL,&a_fd_set,&timeout);
else            ierr = select (s+1,&a_fd_set,NULL,&a_fd_set,NULL)    ;

if (ierr  > 0)
{ iread = recv (s,&Buf[Pt],len,0) ;
  if (iread == 0)                 return(Pt);
  else if (iread == SOCKET_ERROR) return(Pt);
  else                            Pt += iread ;
}
else return(Pt);
return(Pt);
}

int DoWrite( SOCKET s, char * buf, int len, int flags )
{
 int n_send_total, n_last;
 n_send_total = send (s, buf, min(len,512), flags);
 if (n_send_total == SOCKET_ERROR) return (FALSE) ;

 while (n_send_total < len)
 {n_last = send(s, &(buf[n_send_total]),min(len-n_send_total,512),flags);
  if (n_last == SOCKET_ERROR) return (FALSE) ;
  n_send_total +=n_last;
 }
 return (TRUE) ;
}
```

```
int SmallClient(char *buf,int len,char * host,unsigned short port)
{ SOCKET s          ;
  s = socket(AF_INET,SOCK_STREAM,0) ;
  if (!s) return(FALSE) ;

  if ( !DoConnect(s,60,port,host) )
  { closesocket(s);
    return(FALSE) ;
  }
  if(!DoWrite(s,buf,strlen(buf),0))
  {shutdown(2,s); closesocket(s);return(FALSE);}


  if(!DoRead(s,buf,len,600))
  {shutdown(2,s); closesocket(s);return(FALSE);}

  shutdown(2,s); closesocket(s);
  return(TRUE);
}

//=====================================================================
// Simple Client/Serveur Code
//=====================================================================
int SetConnectAddress(SOCKADDR_IN *sin,unsigned short port,char *host)
{ PHOSTENT phe;

 sin->sin_family      = AF_INET;
 sin->sin_port        = htons(port);
 sin->sin_addr.s_addr = inet_addr (host);


 if (sin->sin_addr.s_addr == INADDR_NONE)
 {//struct hostent FAR *phe;
  phe = gethostbyname (host);
  if (phe == NULL) return(FALSE);
  else  memcpy(&(sin->sin_addr),phe->h_addr,4);
  if (sin->sin_addr.s_addr == INADDR_NONE)
  return(FALSE);
 }

return(TRUE);
}

void client(void)
{
 SOCKET client   ;          // Descripteur de socket
 SOCKADDR_IN sin ;          // Adresse socket client
 int err         ;
 WSADATA wsaData ;

 char hostname[]={"www.inria.fr"};

 err = WSAStartup( MAKEWORD( 1, 1 ),&wsaData );
 if ( err != 0 ) return;

 client = socket (PF_INET,SOCK_STREAM,0); // mode connecté
 SetConnectAddress(&sin,80,hostname)    ;

 err = connect (client,(LPSOCKADDR)&sin,sizeof(sin) );

 shutdown(client,2)  ;
 closesocket(client) ;
 WSACleanup();
return;}
```

```
void serveur(void)
{SOCKET serveur,client;     // Descripteur de socket
 SOCKADDR_IN sin ;          // Adresse socket serveur
 SOCKADDR_IN csin;          // Adresse socket client
 int err,len     ;
 WSADATA wsaData ;

 err = WSAStartup( MAKEWORD( 1, 1 ),&wsaData );
 if ( err != 0 ) return ;

 // Création de la socket
 serveur = socket (PF_INET,SOCK_STREAM,0); // mode connecté
 sin.sin_family = AF_INET ;  // Affectation famille de protocoles Internet
 sin.sin_port = htons(80) ; // Affectation du numéro de port TCP
 sin.sin_addr.s_addr =  0 ; // Affectation de l'adresse IP

 // Affecte une adresse au socket
 err = bind (serveur,(LPSOCKADDR) &sin, sizeof (sin));

 // Création de la file d'attente
 listen(serveur,5);

 // Attente de connexion d'un client quelconque (bloquant)
 len =  sizeof (csin) ;
 client = accept (serveur,(LPSOCKADDR) &csin, &len);

 shutdown(client,2)  ;
 closesocket(client) ;

 // Fermeture socket
 closesocket(serveur);

 WSACleanup();
return;}
```

4. HTTP, Hyper Text Transfer Protocol

4.1 Introduction

HTTP est un protocole de niveau application qui comporte la simplicité et la vitesse nécessaires aux applications distribuées ou coopérantes des systèmes d'informations multimédia. Ce protocole est utilisé par le World Wide Web (www) depuis les années 90. Les RFC 1945, puis 2616 décrivent les versions HTTP/1.0 et HTTP/1.1. HTTP s'appuie également sur des RFC décrivant les *Uniform Ressource Identifier* (URI, RFC 1630), Uniform Ressource Locator (URL RFC 1738). Les messages sont encodés de manière analogue à *Multipurpose Internet Mail Extension* (MIME - RFC 1521).

4.2 Les URI.

Un *Uniform Ressource Identifier* est une chaîne qui identifie l'adresse d'un objet (on parle dans ce cas d'une URL) ou la représentation du nom d'un objet enregistré dans un espace particulier on parle dans ce cas d'une URN, *Universal Ressource Name*.

4.2.1 Structure d'une Uniform Ressource Locator (URL).

Une URL comporte quatre parties
aaaa://bbb.bbb.bbb/ccc/ccc/ccc?ddd

4.2.1.1 La méthode d'accès

aaaa : la méthode d'accès

- http: Hyper text Transfer Protocol
- https: HTTP secure, http sécurisé par le protocole SSL/TLS.
- file: accès d'un fichier sur le disque de la machine hôte
- ftp: File Transfer Protocol, transfert de fichier
- mailto: courrier électronique, eMail
- news: consultation des news
- gopher: l'ancêtre du browser WEB
- wais: Wide Area Information Server
- telnet: le protocole TELNET

4.2.1.2 Le noeud internet

//bbb.bbb.bbb
Une double barre oblique (//) définit le nom du hôte distant ou l'adresse IP de cette machine. Lorsque cet élément n'existe pas la requête est effectuée sur la machine locale.
Pour certaines méthodes d'accès (ftp, telnet) le nom de la machine hôte peut être précédé par un identificateur de login et un nom de passe.

//user[:password]@nom_du_système_hôte.

Un numéro de port peut être indiqué, il est délimité par le caractère :
//bbb.bbb.bbb:port

4.2.1.3 Le chemin (path) et le nom de l'objet

/ccc/ccc/ccc
La 1° barre oblique indique le répertoire racine. Dans les requêtes locales son absence indique la référence au répertoire courant.

4.2.1.4 Les arguments

?dddd
Les arguments sont définis en fonction du type de méthode d'accès utilisées.
#nom identifie une partie de document html
?chaîne_de_caractère est utilisé dans les requêtes CGI.

4.2.2 Exemples d'URL

http://www.inria.fr/a_file.html
http:a_file.html
http:/a_file.html
mailto:dupont@compuserve.com
news:
ftp://anonymous@ftp.inria.fr/
ftp://moi:mon_mot_de_passe@ftp.adresse.fr/

4.3 HTTP 0.9, la version informelle de http.

La première version de http est connue sous le nom de http 0.9. Une requête s'effectue de la manière suivante :
- le client réalise une connexion TCP sur le serveur http, généralement sur le port 80, mais d'autres valeurs de ports sont possibles.
- Une fois la connexion réalisée, le client émet un message de la forme:

GET /path/file CR LF

Le serveur émet en retour l'image du fichier requis. Lorsque le serveur a terminé de transmettre le contenu du fichier il effectue la déconnexion TCP. Le client sait que le fichier entier à été transmis lorsqu'il constate la déconnexion du serveur.

4.4 HTTP 1.0, 1.1

Une requête HTTP 1.x comporte un message de requête et un message de réponse (qui inclut l'objet requis).

4.4.1 En tête généraux.

Certains en-têtes peuvent figurer à la fois dans les requêtes et les réponses

Date: HTTP-date
Exemple Date: Tue, 15 Nov 1994 08:12:31 GMT

Pragma:
Spécifie des directives propres à 'l'implémentation.

4.4.2 Message de requêtes.

Le message est constitué d'une suite de ligne. Les lignes sont terminées par la séquence de caractère CR LF.

4.4.2.1 La ligne de requête

La 1° ligne de la requête est une requête 1.x (*full request*) ou 0.9 (*simple request*)
En 1.0

METHOD SP Request-URI SP HTTP-Version CR LF

Les méthodes couramment employées sont les suivantes
- GET, demande d'on objet au serveur.
- HEAD, identique à GET, mais le serveur ne retourne pas l'objet pointé par URI (seulement des informations le concernant)
- POST, permet de poster un message.

HTTP-Version= HTTP/1.1

Request-URI désigne un objet sur le système hôte, la partie droite d'un URL en fait (la méthode d'accès - http, et le nom du système hôte sont ignorés).

4.4.2.2 En-tête généraux

4.4.2.3 L'en tête de requête

Authorization:
Identification/authentification d'un utilisateur
par exemple le schéma d'identification ***basic*** d'un utilisateur est son login et son mot de passe séparés par le caractère : (login:mode de passe) encodé par une chaîne base64 (voir MIME).
Autorization: Basic QWxhZGRpbjpvcGVuIHNlc2FtZQ==
(Aladdin:open sesame)

From:
Désigne un eMail internet

If-Modified-Since: HTTP-Date
Exemple: If-Modified-Since: Sat,29 Oct 1994 19:43:31 GMT
L'objet est retourné seulement s'il a été modifié, dans le cas contraire on récupère le message 304 («not modified»)

Referer:
Indique l'adresse (URI) du client
Exemple Referer: http://machine.fr/fichier.html

User-Agent:
Le browser qui effectue la requête
exemple: User-Agent: Mozilla/3.0Gold (Win16; I)

L'en tête de requête se termine par une ligne vide (CR LF)

Content-Length:
La longueur du corps de message lorsqu'il est présent.

4.4.2.4 Le corps du message.

Une requête qui comporte un corps de message (body) doit comporter un entête Content-Length.

4.4.3 Exemples de requêtes

GET /home.htm HTTP/1.0[CR][LF]
Connection: Keep-Alive[CR][LF]
User-Agent: Mozilla/3.0Gold (Win16; I)[CR][LF]
Pragma: no-cache[CR][LF]
Host: me[CR][LF]
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */*[CR][LF]
[CR][LF]

Un formulaire en méthode POST.

```
<FORM  METHOD=POST ACTION="mailbox">
Name <INPUT NAME="name"  SIZE="40"><br>
eMail <INPUT NAME="email"  SIZE="40"><br>
<P>
Please, use this area, to let me a message <br>
<TEXTAREA NAME="text" TYPE=TEXT ROWS=3 COLS=40, size=40,3>
</textarea><br><br>
<INPUT VALUE="Send"  TYPE=submit><INPUT VALUE="Cancel"  TYPE=reset>
</FORM>
```

POST /mailbox HTTP/1.0[CR][LF]

Referer: http://me/home1.htm[CR][LF]
Connection: Keep-Alive[CR][LF]
User-Agent: Mozilla/3.0Gold (Win16; I)[CR][LF]
Host: me[CR][LF]
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */*[CR][LF]
Content-type: application/x-www-form-urlencoded[CR][LF]
Content-length: 54[CR][LF]
[CR][LF]
name=UrienP&email=UrienP@frcl.bull.fr&text=Hello+world[CR][LF]

Un formulaire en méthode GET.

```
<FORM ACTION="mailbox">
Name <INPUT NAME="name"  SIZE="40"><br>
eMail <INPUT NAME="email"  SIZE="40"><br>
<P>
 Please, use this area, to let me a message <br>
<TEXTAREA NAME="text" TYPE=TEXT ROWS=3 COLS=40, size=40,3>
</textarea><br><br>
<INPUT VALUE="Send"  TYPE=submit><INPUT VALUE="Cancel"  TYPE=reset>
</FORM>
```

GET /mailbox?name=UrienP&email=UrienP@compuserve.com&text=Hello+World HTTP/1.0[CR][LF]
Referer: http://me/home.htm[CR][LF]
Connection: Keep-Alive[CR][LF]
User-Agent: Mozilla/3.0Gold (Win16; I)[CR][LF]
Host: me[CR][LF]
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */*[CR][LF]
[CR][LF]

4.4.4 Messages de réponses

Une réponse peut être simple ou entière (simple, full response). La réponse simple correspond au protocole http 0.9

4.4.4.1 La ligne de status

HTTP-Version SP Status-Code SP Reason-Phrase CR LF

HTTP-Version == HTTP/1.1
Status Code = un nombre entier représenté par 3 digits.

1xx pour information, RFU
2xx Success
- 200 OK
- 201 Created, ressource créée par une méthode POST.
- 202 Accepted, requête en cours de traitement.
- 204, No Content, pas d'information en réponse à la requête.

3xx Redirection
- 300 Multiple Choices, la réponse comporte une liste de choix possible
- 301 Moved Permanently, la réponse comporte un lien sur la nouvelle location.
- 302 Moved Temporarily, la réponse comporte la nouvelle location.
- 304 Not modified, en réponse à une requête GET conditionnelle

4xx Erreur du client
- 400 Bad Request, syntaxe erronée
- 401 Unauthorized
- 403 Forbidden
- 404 Not Found

5xx Erreur du Serveur
- 500 Internal server error
- 501 Not Implemented
- 502 Bad Gateway
- 503 Service Unavailable

Reason Phrase, une suite de caractères ASCII

4.4.4.2 En tête généraux

4.4.4.3 En tête de réponse

Location: absoluteURI
Location exacte associée à la requête
Exemple, Location: http://www.w3.org/hypertext/WWW/NewLocation.html

Server:
Le nom du serveur
Exemple, Server: CERN/3.0 libwwww/2.17

WWW-Authenticate:
Authentification de l'émetteur de la réponse.

4.4.4.4 En tête d'entité

Content-Encoding:
Le type d'encode utilsé, par exemple x-gzip ou x-compress

Content-Length:
La longueur de l'entité body transférée.

Content-Type:
Le contenu du corps de message, sous la forme type/sous-type
Exemple, Content-Type: text/html

Expires: HTTP-Date
Date d'expiration d'une ressource
Exemple, Expires: Thu, 01 Dec 1994 16:00:00 GMT

Last-Modified: HTTP-Date

4.4.4.5 En tête d'extensions

En têtes qui ne sont pas définis par cette version de http.

4.4.4.6 Corps de message

Une suite d'octets.

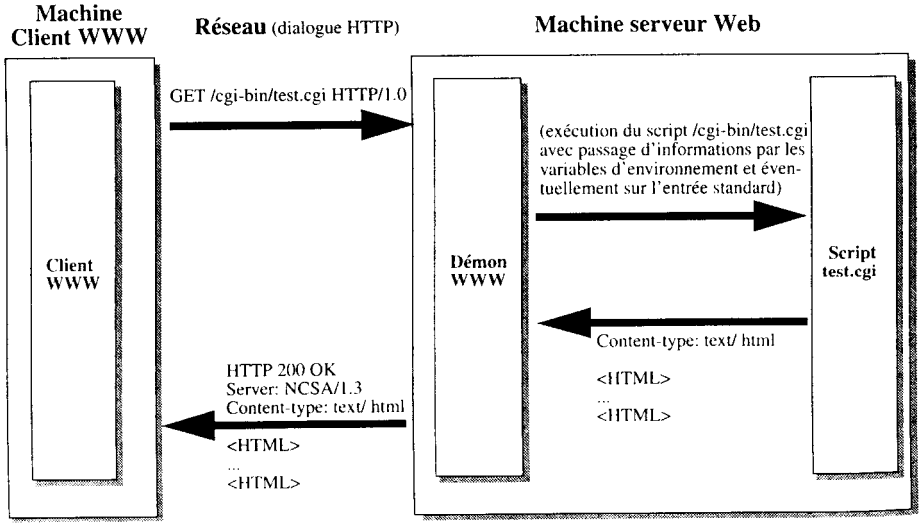
4.4.4.7 Exemple.

HTTP/1.0 200 Document follows CR LF
MIME-Version: 1.0 CR LF
Server: CERN/3.0 CR LF
Date: Saturday, 01-Mar-97 17:20:02 GMT CR LF
Content-Type: text/html CR LF
Content-Length: 2979 CR LF
Last-Modified: Tuesday, 25-Feb-97 10:29:08 GMT CR LF
CR LF
<HTML>
...
</HTML>

4.5 Le Common Gateway Interface (CGI)

Le CGI est un standard de fait, il permet la programmation des serveurs WEB c'est à dire qu'une requête est interprétée comme le nom d'un script ou d'une application. Le serveur exécute le script après à l'aide de variables d'environnement.

Le répertoire des scripts cgi est généralement nommé /cgi-bin. Sous UNIX le serveur exécute le script et transmet au client la sortie standard associée à ce script. Les scripts CGI sont écrits dans des langages tels que C, Fortran, Perl, Shell Unix ...
Le script génère soit un en tête spécifique CGI ou bien directement les en têtes http nécessaires ainsi que le corps de message si nécessaire.



4.6 Un aperçu de Hypertext Markup Language - HTML RFC 1866

HTML est une application de SGML *Standard Generalized Markup Languages*, iso 8879:1986.
Un document html est codé en ASCII, il comporte du texte, des images, et des hyperliens. Un hyperlien permet d'inclure une URI dans le texte. Les images sont identifiées par un URI, une URL en règle générale.
Un hyperlien est délimité par deux ancres (une ancre de tête, et une ancre de queue, une ancre est typiquement marquée par un élément <A>)
Un document html est identifié par les tag <html> et </html>. Il comporte deux éléments principaux l'en tête (<HEAD></HEAD>) et le corps (<BODY></BODY>). Le titre (<TITLE> </TITLE>) est un élément de l'en tête

```
<HTML>
        <HEAD>
                <TITLE>
                </TITLE>
        </HEAD>
        <BODY>
        </BODY>
</HTML>
```

Les caractères utilisés par le langage (tels que &, >, <) doivent être identifiés de manière différente à l'intérieur du texte, par exemple:

& => &amp ou &#38
< => &lt ou &#60
> => &gt ou &#62

Les tags délimitent les éléments tels que en tête<H1>...<H6>, paragraphes<P>, listes <UL>, éléments de listes <LI> ...
Un tag de début commence par < et se termine par >, un tag de fin commence par </ et se termine par >.

Certains éléments ne comportent que des tags de début (
 line break, <HR> trait horizontal, <IMG> ...)
Un tag comporte un nom d'élément et des attributs, une chaine associée à un attribué est encadré par une apostrophe simple ou double.
<Element_Name Attribut_Name=«a_Name»>
<IMG SRC=«img.jpg» alt=«le titre»>
Un commentaire utilise le tag <!>, exemple <! un commentaire>

Un exemple de texte HTML
<!Un exemple>
<html>
<head>
<title>Le titre</title>
</head>
<body>
<H1> Premier en tete</H1>
<P>un paragraphe</P>
<UL> une liste non numérotée
<LI> le premier élément
<LI> le deuxième élément
</UL>
<IMG SRC=« une_image.gif alt=« une image »>
</body></html>

4.6.1 Element html

Un élément html comporte un entête (head) et un corps (body)

4.6.2 Head

Une tête est un ensemble d'information non ordonnée.
<TITLE>titre</TITLE>
<BASE> URL de base associée au document
<ISINDEX> un ensemble d'hyperliens
<LINK> un hyperlien
<META> un ensemble d'informations associées au document

4.6.3 Body

Le corps du document

4.6.3.1 En têtes H1...H6, </H1>...</H6>

4.6.3.2 Paragraphes <P></P>

4.6.3.3 Texte préformatté <PRE></PRE>

4.6.3.4 Liste non numérotée <UL></UL> <LI>

4.6.3.5 Caractère gras <B></B>

4.6.3.6 Italique <I></I>

4.6.4 Ancre <A> </A>

L'élément <A> identifie un hyperlien. Un hyperlien comporte deux partie un URI (par exemple http://...) et un identificateur de fragment séparé par le caractère #. (http:ce_document#un fragment).
Les attributs possibles sont les suivants:
HREF= un URI

NAME= identificateur du fragment

4.6.5 Retour à la ligne

4.6.6 Trait horizontal <HR>

4.6.7 Image <IMG>

Les attributs possibles sont les suivants:
ALIGN=, alignement TOP MIDDLE BOTTOM
RIGHT CENTER LEFT
ALT=, « texte » texte utilisé à la place de l'image sourcec
ISMAP, indique une image « map »
SRC=, URI de la ressource image associée
Remarque:
<img ismap src="img1.gif"> génère l'URL
http://nom_du_hôte/cgi-bin/imagemap/?0.0 (ou par exemple ?34.78)
La réponse met en oeuvre un script cgi.

4.6.8 Caractères

Le jeu de caractères utilisé est par défaut iso-latin1 (ISO 8859-1).

4.6.9 Les formulaires

4.6.9.1 <FORM> </FORM>

ACTION= ,(URI) par exemple
un URL (http://...) si METHOD=GET
un nom de répertoire si METHOD=POST

METHOD=, GET ou POST

ENCTYPE=, le type d'encodage, la valeur par défaut est « application/x-www-form-url-encoded »

4.6.9.2 <INPUT>

NAME=, un nom
MAXLENTGH=, le nombre maximum de caractères de cette entrée
SIZE=, le nombre de caractères max à afficher.
VALUE=, la valeur initiale

TYPE=CHECKBOX, pour un choix boolean
NAME=
VALUE=
CHECKED si l'état initial est on

TYPE=RADIO
choix boolean associé à un bouton
NAME=
VALUE=
CHECKED

TYPE=IMAGE, implique un traitement identique à TYPE=SUBMIT
NAME=
ALIGN=
SRC=

TYPE=HIDDEN
VALUE=, représente un champ caché

TYPE=SUBMIT, affiche un bouton pour la soumission du formulaire
NAME=
VALUE=

TYPE=RESET
VALUE=

4.6.9.3 SELECT, </SELECT>

Permet un choix dans une liste de valeur. Les valeurs sont définies par <OPTION>
MULTIPLE, une option peut être inclue dans une valeur.
NAME=
SIZE=, taille des objets visibles

4.6.9.4 <OPTION>

<OPTION> est situé à l'intérieur d'un élément SELECT
SELECTED, option sélectionnée par défaut
VALUE=, la valeur par défaut d'une otion est son contenu.

4.6.9.5 <TEXTAREA>, </TEXTAREA>

Un champ texte qui comporte plusieurs lignes.

COLS=, nombre de colonnes visibles

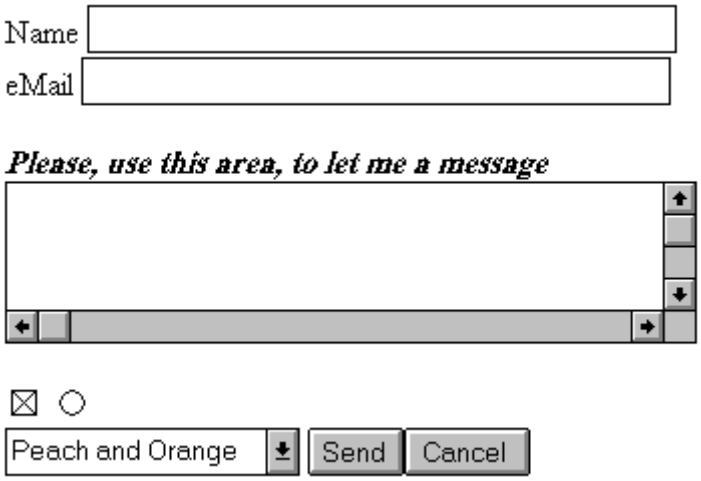
NAME=

ROWS=, le nombre de colonnes visibles

4.6.9.6 Soumission des questionnaires

Les espaces sont remplacés par '+'
Les caractères réservés sont remplacés par %HH, un nombre hexadécimal de deux digits.
Les lignes sont séparées par CR LF, soit %0D%0A
Les entrées du formulaire sont séparées par '&', le nom d'une entrée et sa valeur sont séparés par '='

4.6.9.7 Exemple.




```
<FORM METHOD=GET ACTION="mailbox">
Name  <INPUT NAME="name"   SIZE="40"><br>
eMail <INPUT NAME="email"  SIZE="40"><br>
<br>
<B><I>Please, use this area, to let me a message</B></I><br>
```

```
<TEXTAREA NAME="text" TYPE=TEXT ROWS=3 COLS=40, size=40,3>
</textarea><br><br>
<INPUT TYPE=CHECKBOX NAME="oui/non" CHECKED>
<INPUT TYPE=RADIO    NAME="yes/no"><br>
<SELECT name="flavor">
<OPTION>Vanilla
<OPTION>Strawberry
<OPTION value="RumRaisin">Rum and Raisin
<OPTION selected>Peach and Orange
</SELECT>
<INPUT VALUE="Send"  TYPE=submit><INPUT VALUE="Cancel"  TYPE=reset>
</FORM>
```

```
GET
/mailbox?name=Pascal+Urien&email=UrienP@compuserve.com&text=hello+world+%21%0D%0APas
cal+Urien&oui%2Fnon=on&flavor=Peach+and+Orange HTTP/1.0[CR][LF]
Referer: http://me/home2.htm[CR][LF]
Connection: Keep-Alive[CR][LF]
User-Agent: Mozilla/3.0Gold (Win16; I)[CR][LF]
Host: me[CR][LF]
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */*[CR][LF]
[CR][LF]
```

```
POST /mailbox HTTP/1.0[CR][LF]
Referer: http://me/home2.htm[CR][LF]
Connection: Keep-Alive[CR][LF]
User-Agent: Mozilla/3.0Gold (Win16; I)[CR][LF]
Host: me[CR][LF]
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */*[CR][LF]
Content-type: application/x-www-form-urlencoded[CR][LF]
Content-length: 55[CR][LF]
[CR][LF]
name=&email=&text=&oui%2Fnon=on&flavor=Peach+and+Orange[CR][LF]
```

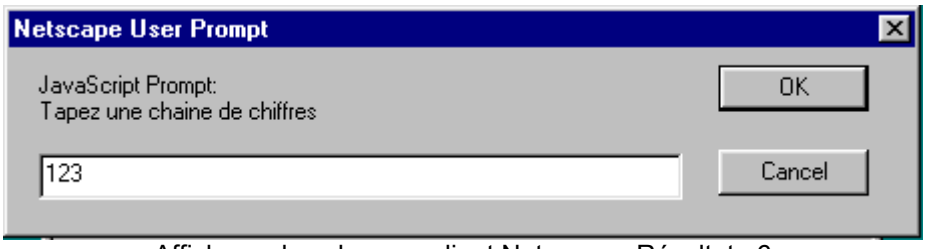
4.7 JavaScript

JavaScript est un langage interprété, intégré aux browsers, conçu par les sociétés Netscape et Sun Microsystem.

Exemple de Script :

```
<HTML>
<HEAD>
<TITLE>Exemple de Javascript</TITLE>
<script language="JavaScript">
var n=0 ;
function somme(chaine)
{ s=0
  for(i=0; i<chaine.length; i++)
    s += eval(chaine.charAt(i)) ;
  if (s > 9)  somme(s.toString());
  return s ;
}
document.write("Resulat : "  + somme(prompt("Tapez une chaine de chiffres", "")) + "<BR>")
</script>
</HEAD>
<BODY>

</BODY>
</HTML>
```

Exécution du script



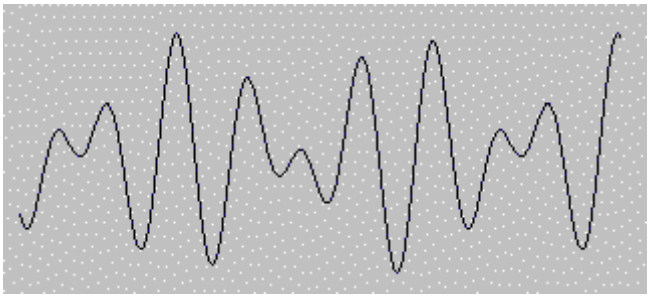
Affichage dans la zone client Netscape : Résultat : 6

4.8 JAVA, ou l'exécution d'Applets.

Le langage java est un langage objet interprété. Il génère des fichiers au format code byte, qui est un mélange de code machine et d'instructions objets. Le code byte est interprété en temps réel par une machine virtuelle java à l'aide d'un JIT (Just In Time Compiler). Les Applets sont une classe dérivée du langage java. Le code byte d'un applet peut être exécuté par tout browser web qui implémente une machine virtuelle Java 5JVM)
Dans le cas du CGI le client émet une requête vers le serveur qui traite la demande et retourne les résultats obtenus. Avec un Applet le client émet une requête vers le serveur qui lui retourne un applet initialisé avec les valeurs de la demande. L'applet s'exécute côté client et non serveur, les ressources sont réparties dans ce cas entre client et serveur.

java exemple 1

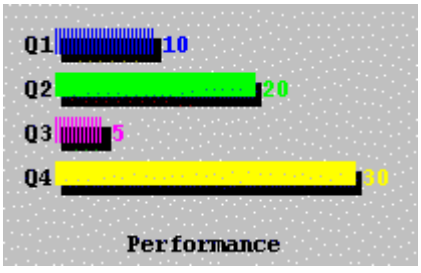
```
import java.awt.Graphics;
public class graphapplet extends java.applet.Applet {
double f(double x) {
        return (Math.cos(x/5) + Math.sin(x/7) + 2) * size().height / 4; }
public void paint(Graphics g) {
        for (int x = 0 ; x < size().width ; x++) {
                g.drawLine(x, (int)f(x), x + 1, (int)f(x + 1));}
        }}
```



<applet code=**graphapplet**.class width=300 height=120></applet>

java exemple 2

```
<html>
<head>  <title>  Bar Chart  </title>  </head>
<body>
<applet code="chart.class" width=251 height=125>
<param name=c2_color value="green">
....
</applet>
</body>  </html>
```



5. XML – Extensible Markup Langage

La version 1.0 d'XML a été publiée en 1998, la version 1.1 publiée en 2004 apporte des améliorations dans le support des différentes versions d'Unicode.

Un document XML est constitué par des entités transportées par le protocole HTTP. C'est un arbre d'éléments, chacun d'entre eux étant délimité par une balise (*tag*). Une balise comporte de manière optionnelle des attributs. Un élément a un contenu qui peut être un autre élément ou un objet XML (par exemple des caractères ou une entité XML)

Un DTD (*Data Type Declaration*) décrit la structure (grammaire) d'un arbre XML

Un document XML n'est pas affiché directement par un browser WEB; une page HTML est construite à partir d'éléments XML à l'aide de processeur de fichiers *Extensible Style Sheet Language* (XSL), ou *Cascading Style Sheets* (CSS).

5.1 Exemple

mycarte.xml

```
<?xml version='1.0'  standalone='no'?>
<?xml-stylesheet type="text/xsl" href="carte.xsl"?>
<!DOCTYPE document [
<!ELEMENT document ANY>
<!ENTITY % c SYSTEM 'carte.dtd' >
<!ENTITY % d SYSTEM 'def.dtd'   >
%c;
%d;
]>
<document>

<carte>
 <person>
  <name>&name;</name>
  <phone>&phone;</phone>
  <photo>&photo;</photo>
  <address>&address;</address>
  <city>&city;</city>
  <zipcode>&zipcode;</zipcode>
  <state>&state;</state>
  </person>
</carte>

</document>
```

carte.dtd

```
<?xml version='1.0' ?>
<!ELEMENT  carte   ANY >
<!ELEMENT  person  ANY >
<!ELEMENT  name    ANY >
<!ELEMENT  address ANY >
<!ELEMENT  photo   ANY >
<!ELEMENT  zipcode ANY >
<!ELEMENT  city    ANY >
<!ELEMENT  state   ANY >
<!ELEMENT  phone   ANY >
```

def.dtd

```
<?xml version='1.0' ?>
<!ENTITY name      SYSTEM "name.xml">
<!ENTITY phone     SYSTEM "phone.xml">
<!ENTITY address   SYSTEM "address.xml">
<!ENTITY city      SYSTEM "city.xml">
<!ENTITY zipcode   SYSTEM "zipcode.xml">
<!ENTITY state     SYSTEM "state.xml">
<!ENTITY photo     "photo.jpg" >
```

carte.xsl

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl"
xmlns:HTML="http://www.w3.org/Profiles/XHTML-transitional">

<xsl:template><xsl:apply-templates/></xsl:template>
<xsl:template match="text()"><xsl:value-of/></xsl:template>

  <xsl:template match="/">
  <HTML>
  <HEAD><TITLE></TITLE></HEAD>
   <BODY>
```

```
  <TABLE BORDER="1" WIDTH="200">
  <TR><TD>
  <H1>
  <xsl:value-of select="document/carte/person/name"/>
  <IMG>
    <xsl:attribute name="src"><xsl:value-of select="document/carte/person/photo"/></xsl:attribute>
    <xsl:attribute name="align">right</xsl:attribute>
    <xsl:attribute name="title"><xsl:value-of select="document/carte/person/name"/></xsl:attribute>
   </IMG>
   </H1>
   <xsl:for-each select="document/carte">
      <P> <xsl:value-of select="person/address"/></P>
      <P> <xsl:value-of select="person/zipcode"/>,<xsl:value-of select="person/city"/></P>
      <P> <xsl:value-of select="person/state"/></P>
      <P>Tel: <xsl:value-of select="person/phone"/></P>
   </xsl:for-each>
 </TD></TR></TABLE>
 </BODY>
 </HTML>

</xsl:template>

  <xsl:template match="@*">
  <xsl:copy><xsl:value-of/></xsl:copy>
  </xsl:template>

</xsl:stylesheet>
```

6. AJAX - Asynchronous Javascript and XML

Le terme Ajax a été introduit par *Jesse James Garett* (informaticien Américain), en 2005 dans un article sur le site Web Adaptive Path.

La technologie AJAX permet de modifier dynamiquement le contenu d'une page HTML, à l'aide de mécanismes javascript, DOM, et XMLHttpRequest.

L'interface de programmation Document Object Model (DOM) a été normalisée par le W3C en 1998. Elle permet de modifier dynamiquement, typiquement à partir d'un *javascript*, le contenu d'une page HTML.

XMLHttpRequest est initialement un composant ActiveX créé en 1998 par Microsoft, et mis en œuvre sur la plupart des navigateurs du marché depuis 2005.

Le principe de la technologie AJAX consiste à ouvrir une requête http à l'aide d'un script javascript, à obtenir une réponse en syntaxe XML, puis à parser son contenu pour modifier le contenu de la page HTML initiale.

6.1 Exemple

file.xml

```
<doc>
<item>
<title>MonTitre</title>
<content>MonContenu </content>
</item>
</doc>
```

ajax.html

```
<html>
<head>
<TITLE></TITLE>

<script type="text/javascript">

var req;

function doit()
{req = new XMLHttpRequest();
req.onreadystatechange = processReqChange;
req.open("POST", document.mycmd.action, true);
req.send(document.mycmd.v.value);
return false ; }

function processReqChange()
{
  if (req.readyState == 4)
  {    if (req.status == 200) display() ;
       else alert("Error");
  }
}

function display()
{ var i=0;
  var content="", t="", c="" ;
  var itemArray =  req.responseXML.getElementsByTagName("item") ;

  for (i = 0; i < itemArray.length; i++)
  {
   t= itemArray[i].getElementsByTagName("title")[0].firstChild.nodeValue;
   c= itemArray[i].getElementsByTagName("content")[0].firstChild.nodeValue;
   content = content + "title: " + t + " content: " + c +  "<br/>" ;
  }

  var div = document.getElementById("details");
  div.innerHTML = "";
  div.innerHTML = content ;
}

</script>
</head><body>


<H1>Hello</H1>

<form action="file.xml"  method="post" name="mycmd" onsubmit="return doit()">
<input TYPE="submit" id="tx"  value="go" >
<input TYPE="hidden" id= "v"  value="HelloWorld">
</form>
<hr>
<div id="details"></div>
</body></html>
```

7. FTP File Transfer Protocol RFC 959

7.1 Introduction

FTP est un protocole de transfert de fichier, conçu pour fonctionner entre machines différentes. Il supporte un nombre limité de type de fichiers (ASCII, binaires, ...) et de structures de fichiers (flux octets ou orientés enregistrements).

7.2 Le protocole.

7.2.1 La représentation des données.

FTP introduit les notions de *type de fichiers*, *format des données*, ***structure de fichiers***, et ***mode de transmission.***

- Type de fichiers
  - ASCII
  - EBCDIC
  - binaire (suite d'octets)
  - local., le fichier est organisé en mots de n bits. Par exemple transfert de nombre flottants larges de 32 bits vers un système à nombre flottant de 32 et 64 bits.
- Contrôle de format

Ce choix n'est disponible que pour les fichiers ASCII ou EBCDIC

  - non imprimable (format par défaut)
  - format telnet
  - format fortran (ASA)
- Structure
  - Structure de fichier (par défaut)
  - Structure d'enregistrements
  - Structure de page (du système d'exploitation TOPS-20)
- Mode de transmission
  - ***Mode flux*** (par défaut), le fichier est transmis comme un flux de données sans en tête. Pour une structure d'enregistrement la fin du fichier est marquée par 2 octets.
  - ***Mode Bloc***, l'information est transférée par blocs, munis d'en tête.
  - ***Mode compressé***, les appariions successives d'un même octet sont compressées.

Les implémentations courantes de FTP comprennent les choix suivants:

- Type, ASCII ou binaire.
- Format non imprimable uniquement
- Structure de fichier.
- Transmission en mode flux.

7.3 Les commandes de FTP

Une commande est une suite de caractères ASCII terminée par CR LF. Des commandes TELNET (IAC) peuvent interrompre le transfert d'un fichier.

7.3.1 Les commandes de contrôle d'accès

USER nom de l'utilisateur
PASS mot de passe
ACCT compte utilisateur
CWD changement de répertoire
CDUP retour au répertoire père
SMNT Monter une structure
REIN réinitialisation
QUIT quitter le serveur

7.3.2 Les commandes de paramètres de transfert

PORT définit le port de données éphémères PORT 129,182,51,227,PORT_MSB,PORT_LSB
PASV demande au serveur de se mettre à l'écoute (serveur) sur un port de données. La réponse sera identique au contenu port (adresse IP et Port de données)
TYPE, type de représentation d'un fichier (AN->ASCCI Non print, I->Image))
STRU structure de fichier F->fichier D->Record P->Page)
MODE mode de transfert (S->Flot-Défaut B->Bloc C->compressé)

7.3.3 Les commandes de services

RETR, transfert de fichier serveur->client
STOR, transfert de fichier client->serveur
STOU, (proche de STOR) transfert d'un fichier sous un nom unique
APPE, (proche de STOR) ajoute le fichier à un fichier existant si nécessaire
ALLO nombre_octets, avant une commande STOR ou APPEND
REST emplacement, demande au serveur de retransmettre le fichier à partir d'un certain rang
REFR chemin d'un fichier que l'on veut renommer
RNTO nouveau chemin du fichier que l'on veut renommer (REFR RNTO)
ABOR demande au serveur d'interrompre une opération de transfert de fichier
DELE effacement d'un fichier
RMD effacement d'un répertoire
MKD création d'un répertoire
PWD affichage du répertoire courant
LIST permet d'obtenir la liste des fichiers d'un répertoire ou du répertoire courant.
NLIST permet d'obtenir une liste de nom de fichiers du répertoire spécifié ou du répertoire courant
SITE, un serveur peut fournir des services spécifiques. La nature de ces services est obtenue en réponse à la commande HELP SITE
STAT retourne l'état du serveur
HELP demande d'information sur les options supportées par le serveur
NOOP, permet de tester l'état de la connexion.

7.4 Les réponses FTP

Une réponse comporte 3 digits ASCII avec un message optionnel

- Signification du premier digit

1 Réponse préliminaire positive, une autre réponse est espérée.
2 Réponse complète positive, une autre commande peut être traitée.
3 Réponse intermédiaire positive, une autre commande doit être émise.
4 Réponse négative transitoire, répéter la commande ultérieurement
5 Réponse complète négative permanente.

- Signification du deuxième digit

0 Erreur de syntaxe
1 Information
2 Connexions - Connexions de contrôle ou de données
3 Authentification et accréditation. Pour le login et le numéro de compte.
4 Non spécifié jusqu'alors
5 Etat du système de fichier

- Le troisième digit affine la réponse.

7.5 Arrêt d'un transfert de fichier

Le système utilisateur émet IAC IP (**I**nterruption **P**rocessus TELNET), IAC DM (Telnet Synch) sur la connexion de contrôle puis insère la commande ABOR.

7.6 FTP anonyme

Le login est anonymous et le password une adresse courrier (ou pas de password).

7.7 Exemple de session FTP

Le client se logge par la commande USER xxxx CR LF
Il obtient usuellement en retour le message

331 PASSWD required for xxxx
Le client émet PASS yyyy CR LF, il obtient la réponse
230 xxx logged in CR LF

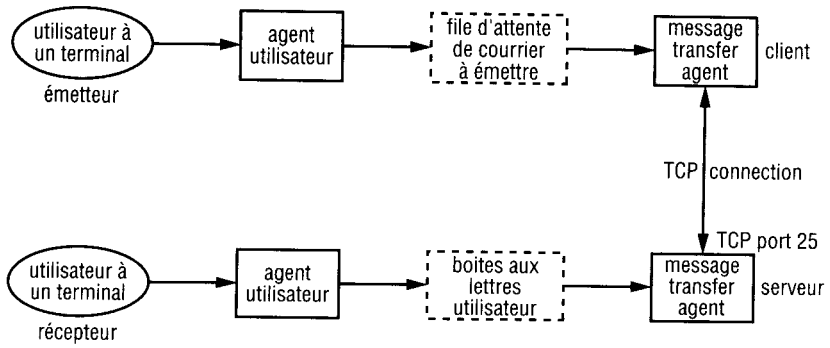
220 halliday FTP server (Version 4.9 Thu Sep 2 20:35:07 CDT 1993) ready.[CR][LF]
USER dupont[CR][LF]
331 Password required for dupont.[CR][LF]
PASS dupond[CR][LF]
230 User dupont logged in.[CR][LF]
PWD[CR][LF]
257 "/tmp_mnt/users/NF/dupont" is current directory.[CR][LF]
SYST[CR][LF]
215 UNIX Type: L8 Version: BSD-44[CR][LF]
HELP[CR][LF]
214- The following commands are recognized (* =>'s unimplemented).[CR][LF]
  USER PORT STOR MSAM* RNTO NLST MKD CDUP [CR][LF]
  PASS PASV APPE MRSQ* ABOR SITE XMKD XCUP [CR][LF]
  ACCT* TYPE MLFL* MRCP* DELE SYST RMD STOU [CR][LF]
  SMNT* STRU MAIL* ALLO CWD STAT XRMD SIZE [CR][LF]
  REIN MODE MSND* REST XCWD HELP PWD MDTM [CR][LF]
  QUIT RETR MSOM* RNFR LIST NOOP XPWD [CR][LF]
214 Direct comments to ftp-bugs@halliday.[CR][LF]
PORT 129,182,51,227,4,6[CR][LF]
200 PORT command successful.[CR][LF]
LIST[CR][LF]
150 Opening data connection for /bin/ls.[CR][LF]
226 Transfer complete.[CR][LF]
MKD test[CR][LF]
257 MKD command successful.[CR][LF]
PWD[CR][LF]
257 "/tmp_mnt/users/NF/dupont" is current directory.[CR][LF]
PORT 129,182,51,227,4,7[CR][LF]
200 PORT command successful.[CR][LF]
LIST[CR][LF]
150 Opening data connection for /bin/ls.[CR][LF]
226 Transfer complete.[CR][LF]
CWD test[CR][LF]
250 CWD command successful.[CR][LF]
PWD[CR][LF]
257 "/tmp_mnt/users/NF/dupont/test" is current directory.[CR][LF]
PORT 129,182,51,227,4,8[CR][LF]
200 PORT command successful.[CR][LF]
LIST[CR][LF]
150 Opening data connection for /bin/ls.[CR][LF]
226 Transfer complete.[CR][LF]
TYPE I[CR][LF]
200 Type set to I.[CR][LF]
PORT 129,182,51,227,4,9[CR][LF]
200 PORT command successful.[CR][LF]
STOR autoexec.000[CR][LF]
150 Opening data connection for autoexec.000.[CR][LF]
226 Transfer complete.[CR][LF]
PWD[CR][LF]
257 "/tmp_mnt/users/NF/dupont/test" is current directory.[CR][LF]
TYPE A[CR][LF]
200 Type set to A; form set to N.[CR][LF]
PORT 129,182,51,227,4,10[CR][LF]
200 PORT command successful.[CR][LF]
LIST[CR][LF]

150 Opening data connection for /bin/ls.[CR][LF]
226 Transfer complete.[CR][LF]
PORT 129,182,51,227,4,11[CR][LF]
200 PORT command successful.[CR][LF]
STOR autoexec.001[CR][LF]
150 Opening data connection for autoexec.001.[CR][LF]
226 Transfer complete.[CR][LF]
PWD[CR][LF]
257 "/tmp_mnt/users/NF/dupont/test" is current directory.[CR][LF]
PORT 129,182,51,227,4,12[CR][LF]
200 PORT command successful.[CR][LF]
LIST[CR][LF]
150 Opening data connection for /bin/ls.[CR][LF]
226 Transfer complete.[CR][LF]
DELE autoexec.001[CR][LF]
250 DELE command successful.[CR][LF]
PWD[CR][LF]
257 "/tmp_mnt/users/NF/dupont/test" is current directory.[CR][LF]
PORT 129,182,51,227,4,13[CR][LF]
200 PORT command successful.[CR][LF]
LIST[CR][LF]
150 Opening data connection for /bin/ls.[CR][LF]
226 Transfer complete.[CR][LF]
TYPE I[CR][LF]
200 Type set to I.[CR][LF]
PORT 129,182,51,227,4,14[CR][LF]
200 PORT command successful.[CR][LF]
RETR autoexec.000[CR][LF]
150 Opening data connection for autoexec.000 (1717 bytes).[CR][LF]
226 Transfer complete.[CR][LF]

8. SMTP Simple Mail Transfer Protocol RFC 821-822

8.1 Introduction

Le service mail est un des services les plus utilisés sous UNIX ou internet. L'utilisateur (humain) utilise un **agent utilisateur** c'est à dire une implémentation spécifique (mail, eudora ...). L'échange de courrier sous TCP est géré par un *agent de transfert de message* **MTA**. Sous Unix l'agent de transfert de courrier le plus courant est **sendmail**. Dans le sens émission de courrier il existe une file d'attente entre l'agent utilisateur et le MTA. Dans le sens réception le MTA transfère les messages incidents dans les boîtes aux lettres utilisateurs qui sont en fait matérialisées par un fichier dans /users/spool/mail/nom_de_l'utilisateur.
La **RFC 821** décrit la communication entre deux MTAs connectés à travers TCP (port 25). **La RFC 822** régit le format du message de courrier électronique échangé entre deux MTAs



8.2 Le protocole SMTP

Le protocole utilise l'ASCII NVT. Le client émet des commandes vers le serveur. Le serveur répond sous forme de code numérique et de commentaires optionnels.

8.2.1 Les commandes

HELO, spécifie le nom de la machine cliente (qui envoie le message)
MAIL, initie une transaction de courrier, comporte une liste optionnelle de systèmes hôtes et l'adresse (eMail) de l'émetteur du message.
MAIL, initialise une transaction vers une ou plusieurs boîtes aux lettres. L'argument de la commande est une liste d'hôtes et la boîte aux lettres de l'émetteur**.** Cette liste peut être utilisée pour retourner un courrier dont l'adresse est erronée.
RCPT (RECIPIENT), identifie le destinataire du courrier, plusieurs destinataires sont possibles par plusieurs occurrences de cette commande. Lorsqu'une liste d'hôtes est présente, le courrier doit être relayé (source routing).
DATA, contenu du courrier. Une ligne se termine par CR LF. La fin des données est définie par la séquence **CR LF . CR LF.**
SEND, initialise une transaction avec un ou plusieurs terminaux. L'argument de la commande est une liste d'hôte et la boîte aux lettres de l'émetteur.
SAML (SEND AND MAIL), initialise une transaction avec des terminaux et des boîtes aux lettres. L'argument de la commande est une liste d'hôte et la boîte aux lettres de l'émetteur.
SOML (SEND OR MAIL), initialise une transaction avec des terminaux et des boîtes aux lettres. L'argument de la commande est une liste d'hôte et la boîte aux lettres de l'émetteur.
RST (RESET) abandon de la transaction en cours
VRFY (VERIFY), l'argument de la commande identifie un utilisateur. Demande de confirmation d'un utilisateur.
EXPN (EXPAND), l'argument est une mailing list. La réponse fournit l'adresse complète des membres de la liste.
HELP, demande d'information d'aide
NOPP, force une réponse OK
QUIT, termine une connexion. Le récepteur répond par OK
TURN, demande au récepteur de prendre le rôle de l'émetteur.

Remarque: seules cinq commandes sont utilisées pour envoyer du courrier: HELO, MAIL, RCPT,DATA et QUIT. Sendmail ne supporte pas la commande TURN. Les commandes SEND SAML SOML sont rarement utilisées.

8.2.2 Les réponses

Une réponse est constituée de 3 chiffres et d'un commentaire.

211 System status, system help reply
214 Help message
220 Service Ready
221 Service Closing transmission channel
250 requested mail action OK
251 user not local; will forward to
354 Start mail input; end with CR LF . CR LF
4xx problèmes divers (action NOK)
5xx Erreurs diverses

8.2.3 Enveloppes, En-têtes, Corps.

Le courrier électronique est composé de trois parties.

L'**enveloppe** est utilisée par le MTA pour acheminer le courrier. Par exemple

MAIL From:
RCPT To:

sont les deux composantes de l'enveloppe. La RFC 821 spécifie le contenu et l'interprétation de l'enveloppe.
Les **en têtes** sont utilisés par les agents utilisateurs. La RFC 822 spécifie le format et l'interprétation de ces éléments. Les en têtes qui commencent X- sont définis par l'utilisateur.

X-Sender:
X-Mailer:
Mime-Version:
Content-Type:
Date:
To:
From:
Subject:

Le **corps** est le contenu du message envoyé de l'expéditeur au destinataire. La RFC 822 spécifie le corps en tant que lignes de texte ASCII NVT. Les en têtes et le corps sont séparés par une ligne vide (CR LF). Une ligne comporte au plus 1000 octets.

8.2.4 Agents MTA locaux et Agents relais.

Dans un environnement UNIX on peut résumer le scénario d'envoi d'un courrier électronique de la manière suivante:

- Un utilisateur se sert d'un terminal (X par exemple) pour lancer un agent utilisateur (tel que mailx ou eudora par exemple). Une fois le courrier généré, une action SEND permet de se connecter sur le port 25 d'une machine hôte du **domaine** de l'entreprise ou du provider.
- Un nom de domaine comporte une racine non nommée, puis des domaines de premier rang (.com,.edu,.fr ...), et enfin des domaines de rang supérieurs. Un serveur de nom de domaine (DNS) permet d'obtenir différentes informations relatives à un domaine, en particulier le nom de la machine hôte capable de recevoir le courrier adressé à ce domaine. En règle générale le mailer d'un domaine est nommé mail@nom_du_domaine ou mailhost@nom_du_domaine.
- A l'intérieur d'un domaine on différencie les local MTA et un relay MTA. Les machines du domaine sont configurées de telle sorte que le courrier destiné à un domaine extérieur soit relayé à travers des *local MTA* vers le *relay MTA* du domaine.
- A travers internet le courrier chemine à travers différents relay MTA. En règle générale le courrier est acheminé par les deux relay MTA des domaines concernés.
- A l'intérieur d'un domaine le courrier est dirigé depuis le relay MTA vers les local MTA, jusqu'à atteindre la machine hôte qui contient le répertoire /users/spool/mail/nom_du_destinataire.

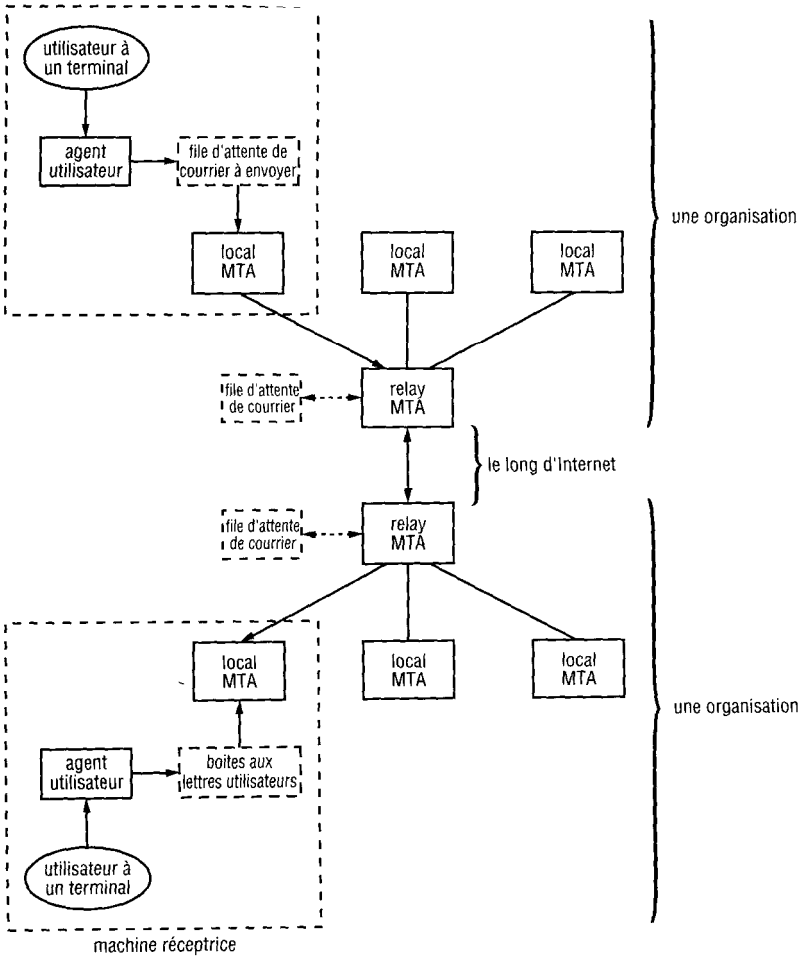


Figure 28.3 *Courrier électronique Internet, avec un système relais aux deux extrémités*

8.2.5 ASCII NVT

L'ascii NVT est un code de 7bits, transmis sous forme d'octets avec le bit de poids fort égal à zéro.

8.2.6 Version étendu de SMTP

Elle est décrite par la **RFC 1993**. Le client qui souhaite utiliser cette version émet la commande EHLO à la place de HELO.

8.2.7 Caractères non ASCII dans les en têtes.

La **RFC 1522** définit la manière d'envoyer des caractères non ASCII dans l'en tête.
Le champ d'un en tête est codé de la manière suivante
=?charset?encoding?encoded-text=
charset

us-ascii iso-8859-X (X=1), le jeu de caractères utilisés

encoding?, le type d'encodage utilsé pour les caractères non ascii

Q, quoted printable, un caractère avec le bit8 à 1 est codé par = suivi de deux digits héxadécimaux

B, codage en base 64, **A,B,...Z,a,b,...,z,0,1,2,...,9,+,/ <=> 0,1,2,....,63**

8.2.8 Multipurpose Internet Mail extension MIME.

MIME décrit par la **RFC 1521** définit la nature du corps du texte. Un corps de message peut être ainsi découpé en plusieurs parties (multipart) dont on indique le type de contenu.

- **Mime-Version:**

La version de MIME (1.0)

- **Content-Type:**

Le contenu du message (TEXT/PLAIN; charset=US-ASCII)
La RFC 1521 recommande quoted printable pour le texte, base64 pour les images et l'audio.
Le type multipart indique un message contenant plusieurs parties
multipart/mixed; Boundary=« NextPart » (parties autonomes, la chaine --NextPart sépare les différentes composantes)
multipart/parallel, multipart/digest, multipart/alternative

- **Content-Transfer-Encoding:**

7 bit (ASCII NVT)
quoted-printable
base64
8bit
binary

- **Content-ID:**

Un identificateur du message

- **Content-Description:**

Ajout de compléments d'information jugé nécessaire.

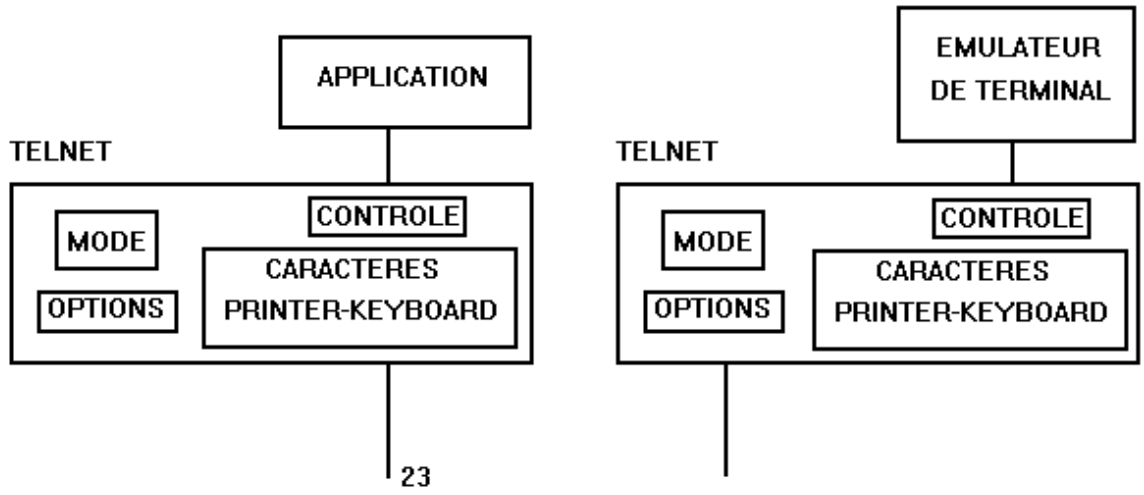
8.2.9 Un exemple

220 gabin Sendmail AIX 3.2/UCB 5.64/4.03 ready at Fri, 7 Mar 1997 16:09:53 +0100[CR][LF]
HELO me[CR][LF]
250 gabin Hello me ([129.182.51.227])[CR][LF]
RSET[CR][LF]
250 Resetting the state.[CR][LF]
MAIL FROM:<P.Urien@frcl.bull.fr>[CR][LF]
250 <P.Urien@frcl.bull.fr>... Sender is valid.[CR][LF]
RCPT TO:<P.Urien@frcl.bull.fr>[CR][LF]
250 <P.Urien@frcl.bull.fr>... Recipient is valid.[CR][LF]
DATA[CR][LF]
354 Enter mail. End with the . character on a line by itself.[CR][LF]
X-Sender: urien@129.182.51.188[CR][LF]
X-Mailer: Windows Eudora Light Version 1.5.2[CR][LF]
Mime-Version: 1.0[CR][LF]
Content-Type: text/plain; charset="us-ascii"[CR][LF]
Date: Thu, 27 Feb 1997 12:09:44 -0200[CR][LF]
To: P.Urien@frcl.b</1><1>ull.fr[CR][LF]
From: P.Urien@frcl.bull.fr[CR][LF]
Subject: Bonjour[CR][LF]
[CR][LF]
Hello World !!![CR][LF]
[CR][LF]
.[CR][LF]
250 Ok[CR][LF]
QUIT[CR][LF]
221 gabin: closing the connection.[CR][LF]

9. TELNET - RFC 854 - port 23

Le protocole TELNET (TELcommunication NETwork) est bâti à partir de trois idées principales:

- Le concept de Network Virtual Terminal (NVT) qui offre une représentation virtuelle de terminaux à travers le réseau internet.
- Le principe d'options négociées, mis en oeuvre par l'échange de PDU de négociation spécifiques.
- Une vue symétrique des terminaux et des *process*. TELNET est utilisable entre un terminal et une application mais également entre deux terminaux ou deux applications.



9.1 ASCII NVT

ASCII NVT est un code de 7 bits transmis sur un octet dont le bit de poids fort est codé à zéro. Une fin de ligne est notifiée par CR LF. Un retour chariot est codé par CR NULL (NUL==0)

9.2 Caractères de contrôle

9.2.1 Interrupt Process

IP (244) - Demande de suspension, d'interruption, de fin du *process* utilisateur

9.2.2 Abort Output

AO (245) Demande l'abandon de la remise des données en cours

9.2.3 Are You Here

AYH (246) Demande au récepteur de signaler sa présence par l'envoi de données (non définies par le standard)

9.2.4 Erase Character

EC (247) Demande de destruction du dernier caractère reçu.

9.2.5 Erase Line

EL (248) Demande l'effacement de la ligne courante.

9.2.6 Data Mark Mécanisme de synchronisation

Data Mark (242) . Ce caractère est émis en mode *Out Of Band*, c'est à dire dans le segment de données urgentes de TCP. Typiquement la réception de DM provoque le non traitement de tous les caractères exceptés les caractères de contrôle. Une nouvelle réception de DM termine le mécanisme de synchronisation.
Par exemple l'émission des caractères de contrôle IP ou AO est suivie d'une séquence de synchronisation.

9.3 Caractères claviers ou imprimantes

NULL	NUL	0
Line Feed	LF	10
Carriage Return	CR	13
BELL	BEL	7, beep sonore

Back Space	BS	8, Retour Arrière
Horizontal Tab	HT	9, Tabulation horizontale
Vertical Tab	VT	11, Tabulation Verticale
Form Feed	FF	12, saut de page

Le clavier associé au NVT doît être capable de générer le caractères suivants:

Data Mark	DM	
Break	BRK	129
Interrupt Process	IP	
Abort Output	AO	
Are You Here	AYT	
Erase Character	EC	
Erase Line	EL	

9.4 Négociation d'option.

Les commandes Telnet sont précédées du caractère IAC - 255 (Interpret as Command). L'octet suivant est un octet de commande
WILL (251) WON'T (252) DO (253) DON'T (254)

Emetteur	Recepteur	
WILL	DO	L'émetteur veut valider une option? - OUI
WILL	DONT	L'émetteur veut valider une optionvalider une option ? - NON
DO	WILL	L'émetteur veut que le récepteur valide l'option ->OUI
DO	WONT	L'émetteur veut que le récepteur valide une option->NON
WONT	DONT	L'émetteur veut invalider l'option, le récepteur doit dire OK
DONT	WONT	L'émetteur veut que le récepteur invalide l'option le récepteur doit dire OK

GA (249) GO AHEAD
SB (250) début de sous option.
SE (240) fin de sous option.

Une négociation d'option comporte donc 3 octets IAC WILL/DO/WONT/DONT et un octet qui identifie l'option.

Option (décimal)	Nom	RFC
1	echo	857
3	supression go ahead	858
5	status	859
6	timing mark	860
24	terminal type	1091
31	windows size	1073
33	remote flux control	1372
34	linemode	1184
36	variables d'environnement	1408

Certaines options requièrent plus de précision que validé ou invalidé. La **RFC 1091** définie une procédure de négociation de sous option.
Par exemple pour la négociation du type de terminal le client émet
IAC WILL 24 (FF FB 18) (je suis un terminal unix)
Le serveur accepte l'option
IAC DO 24 (FF FD 18)

Le serveur demande au client son type de terminal
IAC SB 24 1 IAC SE (FF FA 18 01 FF F0), 1=> requête
Le client indique son type de terminal
IAC SB 24 0 'V' 'T' '1' '0' '0' IAC SE (FF FA 18 00 56 54 31 30 30 FF F0), 0=> réponse

9.5 Principaux mode d'opérations

9.5.1 Semi Duplex

C'est le mode par défaut. Le caractère GO AHEAD tient lieu dans ce cas de jeton. Ce mode est aujourd'hui peu utilisé. Les options ECHO et SUPRESS GO AHEAD permettent de positionner le mode ***un caractère à la fois***.

9.5.2 Un caractère à la fois.

Le client tape des caractères au clavier. Le serveur retourne un echo de la plupart des caractères reçus.
Ce mode peut être demandé par le client par DO SUPPRESS GO AHEAD (FF FD 03) ou par le serveur par WILL SUPPRESS GO AHEAD (FF FB 03). Le serveur demande de restituer l'echo par WILL ECHO (FF FB 01)

9.5.3 Une ligne à la fois

Ce mode est défini par le fait qu'une des deux options ECHO ou GO AHEAD n'est pas positionnée. C'est en fait le mode ligne activé sans faire appel à la RFC 1184

9.5.4 Mode Ligne

Valide l'option 34 définie par la RFC 1184

9.6 Echappement du client.

Un client telnet permet de redéfinir des options en cours de la session, le caractère d'échappement est ***Control]*** (représenté par ***^]***). On obtient alors le prompt ***telnet>.*** On dispose alors de commandes telles que status, mode ...).

9.7 Une trace TELNET

```
129.104.254.6 -> 129.104.254.5
45 00 00 2C 8A 16 00 00 1E 06 13 D9 81 68 FE 06 81 68 FE 05 04 29 00 17   E..,.......Y.h~..h~..)..
53 CB 2C 00 00 00 00 00 60 02 10 00 06 F2 00 00 02 04 04 00               SK,.....`....r......

129.104.254.5 -> 129.104.254.6
45 00 00 2C B3 F5 00 00 1E 06 E9 F9 81 68 FE 05 81 68 FE 06 00 17 04 29   E..,3u....iy.h~..h~....)
28 B4 E6 00 53 CB 2C 01 60 12 10 00 F8 2B 00 00 02 04 04 00               (4f.SK,.`...x+......

129.104.254.6 -> 129.104.254.5
45 00 00 28 8A 17 00 00 1E 06 13 DC 81 68 FE 06 81 68 FE 05 04 29 00 17   E..(.......\.h~..h~..)..
53 CB 2C 01 28 B4 E6 01 50 10 10 00 0E 35 00 00                           SK,.(4f.P....5..

129.104.254.6 -> 129.104.254.5
45 00 00 2E 8A 18 00 00 1E 06 13 D5 81 68 FE 06 81 68 FE 05 04 29 00 17   E..........U.h~..h~..)..
53 CB 2C 01 28 B4 E6 01 50 18 10 00 0F 11 00 00 FF FD 03 FF FB 18         SK,.(4f.P........)..(.

129.104.254.5 -> 129.104.254.6
45 00 00 28 B3 F6 00 00 1E 06 E9 FC 81 68 FE 05 81 68 FE 06 00 17 04 29   E..(3v....i|.h~..h~....)
28 B4 E6 01 53 CB 2C 07 50 10 0F FA 0E 35 00 00                           (4f.SK,.P..z.5..

129.104.254.5 -> 129.104.254.6
45 00 00 2B B3 F9 00 00 1E 06 E9 F6 81 68 FE 05 81 68 FE 06 00 17 04 29   E..+3y....iv.h~..h~....)
28 B4 E6 01 53 CB 2C 07 50 18 0F FA F6 2B 00 00 FF FD 18                  (4f.SK,.P..zv+...).

129.104.254.6 -> 129.104.254.5
45 00 00 28 8A 1A 00 00 1E 06 13 D9 81 68 FE 06 81 68 FE 05 04 29 00 17   E..(.......Y.h~..h~..)..
53 CB 2C 07 28 B4 E6 04 50 10 10 00 0E 2C 00 00                           SK,.(4f.P....,..

129.104.254.5 -> 129.104.254.6
45 00 00 31 B3 FA 00 00 1E 06 E9 EF 81 68 FE 05 81 68 FE 06 00 17 04 29   E..13z....io.h~..h~....)
28 B4 E6 04 53 CB 2C 07 50 18 10 00 1E 07 00 00 FF FB 03 FF FA 18 01 FF   (4f.SK,.P........{..z...
F0                                                                        p

129.104.254.6 -> 129.104.254.5
45 00 00 33 8A 1B 00 00 1E 06 13 CD 81 68 FE 06 81 68 FE 05 04 29 00 17   E..3.......M.h~..h~..)..
53 CB 2C 07 28 B4 E6 0D 50 18 10 00 4D 90 00 00 FF FA 18 00 56 54 31 30   SK,.(4f.P...M....z..VT10
30 FF F0                                                                  0.p

129.104.254.5 -> 129.104.254.6
45 00 00 4C B3 FB 00 00 1E 06 E9 D3 81 68 FE 05 81 68 FE 06 00 17 04 29   E..L3{....iS.h~..h~....)
28 B4 E6 0D 53 CB 2C 12 50 18 10 00 A5 10 00 00 FF FB 01 FF FD 01 0D 0A   (4f.SK,.P...%....{..}...
0D 0A 53 75 6E 4F 53 20 55 4E 49 58 20 28 6C 69 66 6F 75 29 0D 0A 0D 00   ..SunOS UNIX (lifou)....
0D 0A 0D 00                                                               ....

129.104.254.6 -> 129.104.254.5
45 00 00 2E 8A 1C 00 00 1E 06 13 D1 81 68 FE 06 81 68 FE 05 04 29 00 17   E..........Q.h~..h~..)..
53 CB 2C 12 28 B4 E6 31 50 18 10 00 0F E7 00 00 FF FD 01 FF FC 01         SK,.(4f1P....g...}..|.

129.104.254.5 -> 129.104.254.6
45 00 00 2B B3 FC 00 00 1E 06 E9 F3 81 68 FE 05 81 68 FE 06 00 17 04 29   E..+3|....is.h~..h~....)
28 B4 E6 31 53 CB 2C 18 50 18 10 00 0C E4 00 00 FF FE 01                  (4f1SK,.P....d...~.

129.104.254.6 -> 129.104.254.5
45 00 00 28 8A 1E 00 00 1E 06 13 D5 81 68 FE 06 81 68 FE 05 04 29 00 17   E..(.......U.h~..h~..)..
53 CB 2C 18 28 B4 E6 34 50 10 10 00 0D EB 00 00                           SK,.(4f4P....k..

129.104.254.5 -> 129.104.254.6
45 00 00 2F B3 FD 00 00 1E 06 E9 EE 81 68 FE 05 81 68 FE 06 00 17 04 29   E../3}....in.h~..h~....)
28 B4 E6 34 53 CB 2C 18 50 18 10 00 AB C8 00 00 6C 6F 67 69 6E 3A 20      (4f4SK,.P...+H..login:

129.104.254.6 -> 129.104.254.5
45 00 00 28 8A 20 00 00 1E 06 13 D3 81 68 FE 06 81 68 FE 05 04 29 00 17   E..(. .....S.h~..h~..)..
53 CB 2C 18 28 B4 E6 3B 50 10 10 00 0D E4 00 00                           SK,.(4f;P....d..

129.104.254.6 -> 129.104.254.5
45 00 00 29 8A 21 00 00 1E 06 13 D1 81 68 FE 06 81 68 FE 05 04 29 00 17   E..).!.....Q.h~..h~..)..
53 CB 2C 18 28 B4 E6 3B 50 18 10 00 A6 DA 00 00 67                        SK,.(4f;P...&Z..g
```