

# **Cours d'introduction à TCP/IP**

François Laissus

Version du 25 février 2009

## Avant propos

Les sources de ce document sont développées, gérées et conservées grâce aux services de FreeBSD<sup>1</sup>, remarquable système d'exploitation OpenSource!

Les divers fichiers qui composent le source sont édités à l'aide de l'éditeur de texte `vi` ; l'historique des modifications est confié aux bons soins de l'outil `subversion` (gestionnaire de versions). L'ensemble du processus de fabrication est piloté par une poignée de fichiers `Makefile` (commande `make`).

La mise en forme s'effectue grâce au logiciel `LATEX`. Les figures sont dessinées sous “ X Window Systems ” (X11) à l'aide du logiciel `xfig` et intégrées directement dans le document final sous forme de *PostScript* encapsulé. Les listings des exemples de code C ont été fabriqués à l'aide du logiciel `a2ps` et inclus dans le document final également en *PostScript* encapsulé.

La sortie papier a été imprimée en *PostScript* sur une imprimante de type laser, avec `dvips`. La version *pdf* est une transformation du format *PostScript* à l'aide du logiciel `dvipdfm`, enfin la version HTML est traduite directement en HTML à partir du format `LATEX` à l'aide du logiciel `latex2html`.

Tous les outils ou formats utilisés sont en accès ou usage libre, c'est à dire sans versement de droit à leurs auteurs respectifs. Qu'ils en soient remerciés pour leurs contributions inestimables au monde informatique libre et ouvert !

Je remercie également **Jean-Jacques Dhénin** et les nombreux lecteurs que je ne connais qu'au travers de leur e-mails, d'avoir bien voulu prendre le temps de relire l'intégralité de ce cours et de me faire part des innombrables erreurs et coquilles typographiques qu'il comporte, merci encore !

Ce support de cours est en consultation libre à cette url :

HTML    <http://www.laissus.fr/cours/cours.html>

Ou à télécharger au format PDF :

HTTP    <http://www.laissus.fr/cours/cours.pdf>

FTP      <ftp://ftp.laissus.fr/pub/cours/cours.pdf>

D'autres formats (`.ps`, `.dvi`, ...) sont accessibles dans ce répertoire :

HTTP    <http://www.laissus.fr/pub/cours/>

FTP      <ftp://ftp.laissus.fr/pub/cours/>

---

<sup>1</sup><http://www.freebsd.org/>

## Historique des principaux changements

À ce jour(25/02/2009), ce document existe et est accessible sur l'Internet depuis le milieu des années 90. De très nombreux internautes l'ont téléchargé et m'ont renvoyé leurs commentaires. Il était donc plus que temps de garder une trace des principales modifications et restructurations afin que ces lecteurs fidèles puissent suivre les modifications et, peut être, télécharger une nouvelle version en connaissance de cause !

**Version du 25 Février 2009** Restructuration de l'ensemble en quatre parties principales (A,B,C, D) et un index général. Ajout d'une partie " Réseaux IP avancés ". Ajout d'un chapitre sur SNMP et d'un chapitre sur le routage dynamique. Ajout d'un *changelog*, cette page. . .  
Le .pdf est maintenant réactif, les urls, les renvois de pages, le sommaire, les listes de tableaux et figures.  
Nombreuses corrections et mises à jour de tous les chapitres depuis la version du **14 octobre 2007**.



# Table des matières

Préface

xxi

## A Introduction à la pile ARPA 1

### I Réseaux locaux 3

|       |  |    |
|-------|--|----|
| 1     | Préambule . . . . .                                | 3  |
| 2     | Généralités - LANs . . . . .                       | 3  |
| 2.1   | Généralités . . . . .                              | 3  |
| 2.2   | Modèle de communication OSI . . . . .              | 4  |
| 3     | Réseaux locaux . . . . .                           | 7  |
| 3.1   | Qu'est-ce qu'un LAN? . . . . .                     | 7  |
| 3.2   | WAN - MAN . . . . .                                | 8  |
| 3.3   | Communications inter-réseaux . . . . .             | 8  |
| 4     | Couche 2 - Liaison (Data Link) . . . . .           | 9  |
| 4.1   | Caractéristiques d'Ethernet . . . . .              | 9  |
| 4.1.1 | Quelques principes fondamentaux . . . . .          | 9  |
| 4.1.2 | Format d'une " Frame Ethernet " . . . . .          | 10 |
| 4.1.3 | Adresses IEEE 802.3 ou Ethernet . . . . .          | 11 |
| 4.1.4 | Unicast, multicast et broadcast . . . . .          | 12 |
| 4.2   | Différences Ethernet - 802.2/802.3 . . . . .       | 13 |
| 5     | Interconnexion - Technologie élémentaire . . . . . | 14 |
| 5.1   | Raccordement . . . . .                             | 15 |
| 5.1.1 | 10Base5 . . . . .                                  | 15 |
| 5.1.2 | 10Base2 . . . . .                                  | 15 |
| 5.1.3 | 10BaseT . . . . .                                  | 16 |
| 5.1.4 | Fibre optique . . . . .                            | 16 |
| 5.1.5 | Conclusion . . . . .                               | 17 |
| 5.2   | Répéteur . . . . .                                 | 17 |
| 5.3   | Concentrateur . . . . .                            | 18 |
| 5.4   | Ponts . . . . .                                    | 19 |
| 5.5   | Commutateurs . . . . .                             | 20 |
| 5.6   | Passerelles — Routeurs . . . . .                   | 22 |
| 6     | Bibliographie . . . . .                            | 23 |

### II Introduction à IP 25

|                                      |  |           |
|--------------------------------------|--|-----------|
| 1                                    | TCP/IP et l'Internet - Un peu d'histoire . . . . . | 25        |
| 2                                    | Caractéristiques de TCP/IP . . . . .               | 27        |
| 3                                    | Comparaison TCP/IP — ISO . . . . .                 | 28        |
| 3.1                                  | Couche “ Application Layer ” . . . . .             | 29        |
| 3.2                                  | Couche “ Transport Layer ” . . . . .               | 29        |
| 3.3                                  | Couche “ Internet Layer ” . . . . .                | 30        |
| 3.4                                  | Couche “ Network Access ” . . . . .                | 30        |
| 4                                    | Encapsulation d'IP . . . . .                       | 30        |
| 5                                    | Bibliographie . . . . .                            | 31        |
| <b>III Anatomie d'une adresse IP</b> |  | <b>33</b> |
| 1                                    | Adressage IP . . . . .                             | 33        |
| 1.1                                  | Unicité de l'adresse . . . . .                     | 33        |
| 1.2                                  | Délivrance des adresses IPv4 . . . . .             | 34        |
| 2                                    | Anatomie d'une adresse IP . . . . .                | 35        |
| 2.1                                  | Décomposition en classes . . . . .                 | 35        |
| 2.2                                  | Adresses particulières . . . . .                   | 37        |
| 2.3                                  | Sous-réseaux . . . . .                             | 38        |
| 2.4                                  | CIDR . . . . .                                     | 40        |
| 2.5                                  | Précisions sur le broadcast . . . . .              | 41        |
| 3                                    | Adressage multicast . . . . .                      | 42        |
| 3.1                                  | Adresse de groupe multicast . . . . .              | 42        |
| 3.2                                  | Adresse multicast et adresse MAC . . . . .         | 43        |
| 4                                    | Conclusion et bibliographie . . . . .              | 44        |
| <b>IV Protocole IP</b>               |  | <b>47</b> |
| 1                                    | Datagramme IP . . . . .                            | 47        |
| 1.1                                  | Structure de l'en-tête . . . . .                   | 47        |
| 1.2                                  | Network Byte Order . . . . .                       | 48        |
| 1.3                                  | Description de l'en-tête . . . . .                 | 49        |
| 1.4                                  | Fragmentation IP - MTU . . . . .                   | 52        |
| 1.4.1                                | Fragmentation . . . . .                            | 52        |
| 1.4.2                                | Réassemblage . . . . .                             | 53        |
| 2                                    | Protocole ARP . . . . .                            | 55        |
| 2.1                                  | Fonctionnement . . . . .                           | 55        |
| 2.2                                  | Format du datagramme . . . . .                     | 57        |
| 2.3                                  | Proxy ARP . . . . .                                | 58        |
| 3                                    | Protocole RARP . . . . .                           | 58        |
| 4                                    | Protocole ICMP . . . . .                           | 59        |
| 4.1                                  | Le système de messages d'erreur . . . . .          | 59        |
| 4.2                                  | Format des messages ICMP . . . . .                 | 60        |
| 4.3                                  | Quelques types de messages ICMP . . . . .          | 61        |
| 5                                    | Protocole IGMP . . . . .                           | 63        |
| 5.1                                  | Description de l'en-tête . . . . .                 | 63        |
| 5.2                                  | Fonctionnement du protocole . . . . .              | 64        |

|           |  |           |
|-----------|--|-----------|
| 5.3       | Fonctionnement du Mbone . . . . .                      | 65        |
| 6         | Routage IP . . . . .                                   | 66        |
| 6.1       | Table de routage . . . . .                             | 67        |
| 6.2       | Routage statique . . . . .                             | 69        |
| 6.2.1     | Algorithme de routage . . . . .                        | 70        |
| 6.3       | Routage dynamique . . . . .                            | 71        |
| 6.3.1     | RIP — “ Routing Information Protocol ” . .             | 72        |
| 6.3.2     | OSPF — “ Open Shortest Path First ” . . . .            | 73        |
| 6.4       | Découverte de routeur et propagation de routes . . . . | 73        |
| 6.5       | Message ICMP “ redirect ” . . . . .                    | 74        |
| 6.6       | Interface de “ loopback ” . . . . .                    | 75        |
| 7         | Finalement, comment ça marche ? . . . . .              | 76        |
| 8         | Conclusion sur IP . . . . .                            | 78        |
| 9         | Bibliographie . . . . .                                | 79        |
| <b>V</b>  | <b>Protocole UDP</b>                                   | <b>81</b> |
| 1         | UDP – User Datagram Protocol . . . . .                 | 81        |
| 1.1       | Identification de la destination . . . . .             | 81        |
| 1.2       | Description de l’en-tête . . . . .                     | 83        |
| 1.3       | Ports réservés — ports disponibles . . . . .           | 85        |
| 1.3.1     | Attribution des ports “ancienne méthode” . .           | 86        |
| 1.3.2     | Attribution des ports “nouvelle méthode” . .           | 86        |
| 2         | Bibliographie . . . . .                                | 87        |
| <b>VI</b> | <b>Protocole TCP</b>                                   | <b>89</b> |
| 1         | TCP – Transmission Control Protocol . . . . .          | 89        |
| 1.1       | Caractéristiques de TCP . . . . .                      | 89        |
| 1.2       | Description de l’en-tête . . . . .                     | 91        |
| 2         | Début et clôture d’une connexion . . . . .             | 94        |
| 2.1       | Établissement d’une connexion . . . . .                | 94        |
| 2.2       | Clôture d’une connexion . . . . .                      | 95        |
| 2.2.1     | Clôture canonique . . . . .                            | 95        |
| 2.2.2     | Clôture abrupte . . . . .                              | 96        |
| 3         | Contrôle du transport . . . . .                        | 97        |
| 3.1       | Mécanisme de l’acquittement . . . . .                  | 97        |
| 3.2       | Fenêtres glissantes . . . . .                          | 98        |
| 4         | Compléments sur le fonctionnement de TCP . . . . .     | 100       |
| 4.1       | Algorithme de Nagle . . . . .                          | 100       |
| 4.2       | Départ lent . . . . .                                  | 101       |
| 4.3       | Évitement de congestion . . . . .                      | 101       |
| 5         | Paquets capturés, commentés . . . . .                  | 102       |
| 6         | Conclusion sur TCP . . . . .                           | 105       |
| 7         | Bibliographie . . . . .                                | 105       |

|             |  |            |
|-------------|--|------------|
| <b>B</b>    | <b>Réseaux IP avancés</b>                                      | <b>107</b> |
| <b>VII</b>  | <b>Routage dynamique d'IP</b>                                  | <b>109</b> |
| 1           | Introduction & rappels . . . . .                               | 109        |
| 1.1         | IGP, EGP, Système autonome . . . . .                           | 110        |
| 1.2         | Vecteur de distances vs État de liens . . . . .                | 111        |
| 2           | Routage avec RIP . . . . .                                     | 113        |
| 2.1         | En fonctionnement . . . . .                                    | 114        |
| 2.1.1       | Horizon partagé ou <i>Split horizon</i> . . . . .              | 116        |
| 2.1.2       | Mises à jour déclenchées ou <i>Triggered updates</i> . . . . . | 117        |
| 2.2         | Le protocole RIPv1 vs RIPv2 . . . . .                          | 118        |
| 2.3         | Algorithme Bellman-Ford . . . . .                              | 120        |
| 2.3.1       | Métrique . . . . .   | 120        |
| 2.4         | Conclusion . . . . .   | 120        |
| 2.4.1       | Points forts . . . . .   | 120        |
| 2.4.2       | Points faibles . . . . .                                       | 120        |
| 3           | Routage avec OSPF . . . . .                                    | 121        |
| 3.1         | Grandes lignes de fonctionnement . . . . .                     | 121        |
| 3.2         | RIP vs OSPF . . . . .  | 122        |
| 3.3         | Principe de propagation des états . . . . .                    | 124        |
| 3.3.1       | Valeur des états de liens . . . . .                            | 127        |
| 3.4         | Calcul du plus court chemin . . . . .                          | 127        |
| 3.5         | Hiérarchie de routeurs . . . . .                               | 127        |
| 3.6         | Fonctionnement à l'intérieur d'une zone . . . . .              | 129        |
| 3.6.1       | Voisinage et adjacence . . . . .                               | 130        |
| 3.7         | Protocole HELLO . . . . .                                      | 131        |
| 3.7.1       | Cinq types de paquets . . . . .                                | 131        |
| 3.7.2       | En-tête standard des paquets OSPF . . . . .                    | 133        |
| 3.7.3       | En-tête des paquets HELLO . . . . .                            | 133        |
| 4           | Bibliographie . . . . .  | 135        |
| <b>VIII</b> | <b>Éléments de réseaux</b>                                     | <b>137</b> |
| 1           | Hôtes ou services virtuels . . . . .                           | 137        |
| 2           | Tunnel IP . . . . .  | 139        |
| 2.1         | Tunnel IP avec l'interface <b>gif</b> . . . . .                | 140        |
| 2.2         | IPsec et VPN . . . . .   | 143        |
| 2.2.1       | IPsec dans quel but ? . . . . .                                | 143        |
| 2.2.2       | IPsec en résumé . . . . .                                      | 144        |
| 2.2.3       | Comment utiliser IPsec ? . . . . .                             | 145        |
| 2.2.4       | Implémentation d'IPsec . . . . .                               | 147        |
| 3           | Proxy . . . . .  | 148        |
| 4           | Translation d'adresses . . . . .                               | 148        |
| 4.1         | NAPT sur un routeur de type PC avec <b>natd</b> . . . . .      | 150        |
| 4.1.1       | Interactions entre <b>natd</b> et le noyau . . . . .           | 151        |
| 4.2         | Translation d'adresses vers le réseau privé . . . . .          | 152        |



|     |  |     |
|-----|--|-----|
| 4.3 | NAPT sur un routeur CISCO . . . . .              | 153 |
| 5   | Filtrage IP . . . . .                            | 154 |
| 5.1 | Filtrage IP sur un routeur CISCO . . . . .       | 154 |
| 5.2 | Le cas d' <code>ipfw</code> de FreeBSD . . . . . | 154 |
| 6   | Exemple complet . . . . .                        | 157 |
| 7   | Bibliographie . . . . .                          | 160 |

## C Protocoles applicatifs 163

### IX Serveur de noms - DNS 165

|       |  |     |
|-------|--|-----|
| 1     | Généralités sur le serveur de noms . . . . .       | 165 |
| 1.1   | Bref historique . . . . .                          | 165 |
| 1.2   | Système hiérarchisé de nommage . . . . .           | 166 |
| 1.2.1 | Domaine & zone . . . . .                           | 167 |
| 1.2.2 | Hiérarchie des domaines . . . . .                  | 168 |
| 2     | Fonctionnement du DNS . . . . .                    | 169 |
| 2.1   | Convention de nommage . . . . .                    | 169 |
| 2.1.1 | “ Completion ” . . . . .                           | 170 |
| 2.2   | Le “ Resolver ” . . . . .                          | 170 |
| 2.3   | Stratégie de fonctionnement . . . . .              | 172 |
| 2.3.1 | Interrogation locale . . . . .                     | 172 |
| 2.3.2 | Interrogation distante . . . . .                   | 173 |
| 2.3.3 | Interrogation par “ procuration ” . . . . .        | 174 |
| 2.4   | Hiérarchie de serveurs . . . . .                   | 175 |
| 2.5   | Conversion d'adresses IP en noms . . . . .         | 175 |
| 2.6   | Conclusion . . . . .                               | 177 |
| 3     | Mise à jour dynamique . . . . .                    | 177 |
| 4     | Sécurisation des échanges . . . . .                | 178 |
| 4.1   | TSIG/TKEY pour sécuriser les transferts . . . . .  | 178 |
| 4.1.1 | TSIG . . . . .                                     | 179 |
| 4.1.2 | TKEY . . . . .                                     | 179 |
| 4.2   | DNSSEC pour sécuriser les interrogations . . . . . | 179 |
| 5     | Attaque DNS par amplification . . . . .            | 180 |
| 6     | Format des “ Resource Record ” . . . . .           | 182 |
| 6.1   | RR de type <b>SOA</b> . . . . .                    | 183 |
| 6.2   | RR de type <b>NS</b> . . . . .                     | 183 |
| 6.3   | RR de type <b>A</b> . . . . .                      | 184 |
| 6.4   | RR de type <b>PTR</b> . . . . .                    | 184 |
| 6.5   | RR de type <b>MX</b> . . . . .                     | 184 |
| 6.6   | RR de type <b>CNAME</b> . . . . .                  | 185 |
| 6.7   | Autres RR... . . . .                               | 185 |
| 7     | BIND de l'ISC . . . . .                            | 186 |
| 7.1   | Architecture du daemon “ named ” . . . . .         | 186 |
| 8     | Bibliographie . . . . .                            | 187 |

|           |   |            |
|-----------|---|------------|
| <b>X</b>  | <b>Courrier électronique</b>  | <b>189</b> |
| 1         | Généralités sur le courrier électronique . . . . .                    | 189        |
| 1.1       | Métaphore du courrier postal - L'enveloppe . . . . .                  | 190        |
| 1.2       | Adresse électronique . . . . .  | 190        |
| 2         | Format d'un " E-mail " - RFC 822 . . . . .                            | 191        |
| 2.1       | Quelques champs couramment rencontrés dans les en-<br>têtes . . . . . | 192        |
| 3         | Protocole SMTP - RFC 821 . . . . .                                    | 195        |
| 3.1       | Protocole SMTP . . . . .  | 195        |
| 3.2       | Principales commandes de SMTP . . . . .                               | 197        |
| 3.2.1     | Commande HELO . . . . .   | 197        |
| 3.2.2     | Commande MAIL . . . . .   | 198        |
| 3.2.3     | Commande RCPT . . . . .   | 198        |
| 3.2.4     | Commande DATA . . . . .   | 198        |
| 3.2.5     | Commande QUIT . . . . .   | 198        |
| 3.3       | Propagation du courrier électronique . . . . .                        | 199        |
| 3.4       | Courriers indésirables - Le spam . . . . .                            | 201        |
| 3.4.1     | Caractériser le spam . . . . .  | 201        |
| 3.4.2     | Éviter le spam . . . . .  | 202        |
| 4         | Exemple de MTA - " Sendmail " et son environnement . . . . .          | 205        |
| 4.1       | Relations avec le DNS . . . . .                                       | 205        |
| 4.2       | Relations avec le système d'exploitation . . . . .                    | 206        |
| 4.3       | Le cas de POP . . . . .   | 210        |
| 4.4       | Le cas de IMAP . . . . .  | 211        |
| 5         | Configuration du Sendmail . . . . .                                   | 212        |
| 5.1       | Configuration à l'aide de M4 . . . . .                                | 212        |
| 5.2       | Configuration manuelle . . . . .                                      | 214        |
| 5.2.1     | Règles de réécriture . . . . .  | 214        |
| 5.2.2     | Exemple de sortie de debug . . . . .                                  | 217        |
| 6         | Bibliographie . . . . .   | 218        |
| <b>XI</b> | <b>Instrumentalisation de réseaux avec SNMP</b>                       | <b>221</b> |
| 1         | Nécessité d'un outil . . . . .  | 221        |
| 1.1       | Problématique de l'ISO . . . . .                                      | 221        |
| 1.2       | Système de gestion de réseau . . . . .                                | 222        |
| 1.3       | SNMP — Simple Network Management Protocol . . . . .                   | 223        |
| 1.4       | Historique du protocole SNMP . . . . .                                | 224        |
| 1.5       | Vocabulaire et architecture . . . . .                                 | 224        |
| 1.6       | Différentes versions . . . . .  | 226        |
| 1.6.1     | Trois composantes pour SNMP . . . . .                                 | 226        |
| 1.6.2     | Conclusion . . . . .  | 227        |
| 2         | SMI — Structure of Management Information . . . . .                   | 228        |
| 3         | MIB — Management Information Base . . . . .                           | 228        |
| 3.1       | OID — Objet Identifier . . . . .                                      | 230        |
| 3.2       | Types de données élémentaires . . . . .                               | 231        |

|     |  |     |
|-----|--|-----|
| 4   | La MIB-2 . . . . .                     | 232 |
| 5   | Protocole SNMP . . . . .               | 234 |
| 5.1 | Communauté . . . . .                   | 235 |
| 5.2 | PDU's . . . . .                        | 235 |
| 5.3 | SNMPv3 . . . . .                       | 237 |
| 6   | L'outil NET-SNMP . . . . .             | 238 |
| 6.1 | <b>snmptranslate</b> . . . . .         | 238 |
| 6.2 | <b>snmpget</b> . . . . .               | 242 |
| 6.3 | <b>snmpgetnext</b> . . . . .           | 242 |
| 6.4 | <b>snmpwalk</b> . . . . .              | 242 |
| 6.5 | <b>snmptable</b> . . . . .             | 243 |
| 6.6 | <b>snmpset</b> . . . . .               | 243 |
| 6.7 | Approche graphique . . . . .           | 244 |
| 7   | Glossaire des acronymes SNMP . . . . . | 247 |
| 8   | Liens & Bibliographie . . . . .        | 248 |

## D Sockets BSD et architecture de serveurs 249

### XII Généralités sur les sockets de Berkeley 251

|       |   |     |
|-------|---|-----|
| 1     | Généralités . . . . .                                 | 251 |
| 2     | Présentation des sockets . . . . .                    | 252 |
| 3     | Étude des primitives . . . . .                        | 253 |
| 3.1   | Création d'une socket . . . . .                       | 253 |
| 3.1.1 | Valeur retournée par <b>socket</b> . . . . .          | 255 |
| 3.2   | Spécification d'une adresse . . . . .                 | 256 |
| 3.2.1 | Spécification d'un numéro de port . . . . .           | 256 |
| 3.2.2 | Spécification d'une adresse IP . . . . .              | 256 |
| 3.2.3 | La primitive <b>bind</b> . . . . .                    | 256 |
| 3.2.4 | Les structures d'adresses . . . . .                   | 257 |
| 3.2.5 | Valeur retournée par <b>bind</b> . . . . .            | 258 |
| 3.3   | Connexion à une adresse distante . . . . .            | 259 |
| 3.3.1 | Mode connecté . . . . .                               | 259 |
| 3.3.2 | Mode datagramme . . . . .                             | 259 |
| 3.3.3 | Valeur retournée par <b>connect</b> : . . . . .       | 260 |
| 3.4   | Envoyer des données . . . . .                         | 260 |
| 3.4.1 | Envoi en mode connecté . . . . .                      | 260 |
| 3.4.2 | Envoi en mode datagramme . . . . .                    | 261 |
| 3.5   | Recevoir des données . . . . .                        | 262 |
| 3.5.1 | Reception en mode connecté . . . . .                  | 262 |
| 3.5.2 | Recevoir en mode datagramme . . . . .                 | 262 |
| 3.6   | Spécifier une file d'attente . . . . .                | 263 |
| 3.7   | Accepter une connexion . . . . .                      | 263 |
| 3.8   | Terminer une connexion . . . . .                      | 264 |
| 4     | Schéma général d'une session client-serveur . . . . . | 265 |

|             |  |            |
|-------------|--|------------|
| 5           | Exemples de code “ client ” . . . . .                              | 267        |
| 5.1         | Client TCP “ DTCPcli ” . . . . .                                   | 267        |
| 5.2         | Client UDP “ DUDPcli ” . . . . .                                   | 271        |
| 6           | Conclusion et Bibliographie . . . . .                              | 273        |
| <b>XIII</b> | <b>Compléments sur les sockets Berkeley</b>                        | <b>275</b> |
| 1           | Réservation des ports . . . . .                                    | 275        |
| 1.1         | Réservation de port — Ancienne méthode . . . . .                   | 276        |
| 1.2         | Réservation de port — Nouvelle méthode . . . . .                   | 276        |
| 2           | Ordre des octets sur le réseau . . . . .                           | 277        |
| 3           | Opérations sur les octets . . . . .                                | 278        |
| 4           | Conversion d’adresses . . . . .                                    | 279        |
| 4.1         | Conversion d’adresse - IPv4 seul . . . . .                         | 279        |
| 4.2         | Conversion d’adresse - Compatible IPv4 et IPv6 . . . . .           | 279        |
| 5           | Conversion hôte – adresse IPv4 . . . . .                           | 280        |
| 5.1         | Une adresse IP à partir d’un nom d’hôte . . . . .                  | 280        |
| 5.2         | Un nom d’hôte à partir d’une adresse IP . . . . .                  | 282        |
| 6           | Conversion N° de port – service . . . . .                          | 282        |
| 6.1         | Le numéro à partir du nom . . . . .                                | 282        |
| 6.2         | Le nom à partir du numéro . . . . .                                | 284        |
| 7           | Getaddrinfo, pour IPv4 et IPv6 . . . . .                           | 285        |
| 7.1         | La fonction <code>getaddrinfo</code> . . . . .                     | 285        |
| 7.1.1       | Prototype de <code>getaddrinfo</code> . . . . .                    | 285        |
| 7.1.2       | Description des arguments . . . . .                                | 286        |
| 7.1.3       | La structure <code>addrinfo</code> . . . . .                       | 286        |
| 7.1.4       | En résumé . . . . .  | 287        |
| 7.1.5       | Exemple d’usage à la place de <code>gethostbyname</code> . . . . . | 288        |
| 7.1.6       | Exemple d’usage à la place de <code>getservbyname</code> . . . . . | 290        |
| 7.1.7       | En résumé . . . . .  | 290        |
| 8           | Conversion nom de protocole – N° de protocole . . . . .            | 291        |
| 9           | Diagnostic . . . . .   | 292        |
| 10          | Exemples de mise en application . . . . .                          | 293        |
| 10.1        | Ancienne méthode (usage de <code>gethostbyname</code> ) . . . . .  | 293        |
| 10.2        | Nouvelle méthode (usage de <code>getaddrinfo</code> ) . . . . .    | 298        |
| 11          | Conclusion et bibliographie . . . . .                              | 300        |
| <b>XIV</b>  | <b>Éléments de serveurs</b>  | <b>301</b> |
| 1           | Type de serveurs . . . . .   | 301        |
| 1.1         | Serveurs itératif et concourant . . . . .                          | 301        |
| 1.2         | Le choix d’un protocole . . . . .                                  | 302        |
| 1.2.1       | Mode connecté . . . . .  | 302        |
| 1.2.2       | Mode datagramme . . . . .  | 303        |
| 1.3         | Quatre modèles de serveurs . . . . .                               | 303        |
| 2           | Technologie élémentaire . . . . .                                  | 307        |
| 2.1         | Gestion des “ tâches esclaves ” . . . . .                          | 307        |

|           |   |            |
|-----------|---|------------|
| 2.2       | fork, vfork et rfork . . . . .                                | 308        |
| 2.3       | Processus légers, les “ threads ” . . . . .                   | 309        |
| 2.4       | Programmation asynchrone . . . . .                            | 311        |
| 2.5       | La primitive <code>select</code> . . . . .                    | 312        |
| 2.6       | La primitive <code>poll</code> . . . . .                      | 314        |
| 3         | Fonctionnement des daemons . . . . .                          | 315        |
| 3.1       | Programmation d'un <i>daemon</i> . . . . .                    | 315        |
| 3.2       | Daemon <code>syslogd</code> . . . . .                         | 316        |
| 3.3       | Fichier <code>syslog.conf</code> . . . . .                    | 318        |
| 3.4       | Fonctions <code>syslog</code> . . . . .                       | 318        |
| 4         | Exemple de “ daemon ” <code>inetd</code> . . . . .            | 320        |
| 4.1       | Présentation de <code>inetd</code> . . . . .                  | 320        |
| 5         | Exemple de code serveur . . . . .                             | 322        |
| 5.1       | Guide de lecture du source <code>serv2prot.c</code> . . . . . | 322        |
| 6         | Bibliographie . . . . .                                       | 325        |
| <b>XV</b> | <b>Anatomie d'un serveur Web</b>                              | <b>327</b> |
| 1         | Le protocole HTTP . . . . .                                   | 327        |
| 1.1       | Exemple d'échange avec <code>http</code> . . . . .            | 328        |
| 1.2       | Structure d'un échange . . . . .                              | 328        |
| 2         | URIs et URLs . . . . .  | 332        |
| 2.1       | Scheme <code>http</code> . . . . .                            | 332        |
| 3         | Architecture interne du serveur Apache . . . . .              | 334        |
| 3.1       | Environnement d'utilisation . . . . .                         | 334        |
| 3.2       | Architecture interne . . . . .                                | 336        |
| 3.2.1     | Gestion des processus . . . . .                               | 337        |
| 3.2.2     | Prise en main des requêtes . . . . .                          | 342        |
| 3.2.3     | Deux types de CGI . . . . .                                   | 343        |
| 4         | Principe de fonctionnement des CGI . . . . .                  | 347        |
| 4.1       | CGI — Méthode GET, sans argument . . . . .                    | 347        |
| 4.2       | CGI — Méthode GET, avec arguments . . . . .                   | 348        |
| 4.3       | CGI — Méthode POST . . . . .                                  | 349        |
| 4.4       | Ecriture d'une CGI en Perl . . . . .                          | 350        |
| 5         | Conclusion – Bibliographie . . . . .                          | 351        |
| <b>E</b>  | <b>Index général &amp; Annexes</b>                            | <b>353</b> |
| <b>A</b>  | <b>Programme <code>serv2prot.c</code></b>                     | <b>367</b> |



# Table des figures

|        |  |    |
|--------|--|----|
| I.01   | Modèle en 7 couches de l'OSI . . . . .                   | 6  |
| I.02   | Exemple de LANs . . . . .                                | 7  |
| I.03   | trame Ethernet . . . . .                                 | 10 |
| I.04   | Différences Ethernet 802.2/802.3 . . . . .               | 13 |
| I.05   | Interconnexion - Technologie élémentaire . . . . .       | 14 |
| I.06   | Prise vampire . . . . .                                  | 15 |
| I.07   | Technologie de liaison . . . . .                         | 17 |
| I.08   | Plusieurs répéteurs mais toujours le même lan . . . . .  | 18 |
| I.09   | Concentrateur . . . . .                                  | 18 |
| I.10   | Dialogue sans pont . . . . .                             | 19 |
| I.11   | Dialogue avec pont . . . . .                             | 19 |
| I.12   | Commutateur . . . . .                                    | 21 |
| I.13   | Fonction routage . . . . .                               | 22 |
| I.14   | Traduction de protocoles . . . . .                       | 22 |
| II.01  | Comparaison ISO-ARPA . . . . .                           | 28 |
| II.02  | Architecture logicielle . . . . .                        | 29 |
| II.03  | Encapsulation d'IP . . . . .                             | 31 |
| III.01 | Décomposition en classes . . . . .                       | 35 |
| III.02 | Sous-réseaux . . . . .                                   | 38 |
| III.03 | Puissances de 2 . . . . .                                | 38 |
| III.04 | Adresses de multicast . . . . .                          | 42 |
| III.05 | Adresse physique de multicast . . . . .                  | 43 |
| III.06 | Usage combiné des adresses logique et physique . . . . . | 44 |
| IV.01  | Structure du datagramme IP . . . . .                     | 47 |
| IV.02  | " Big endian " vs " Little endian " . . . . .            | 48 |
| IV.03  | Fragmentation IP . . . . .                               | 52 |
| IV.04  | Fragment à transmettre . . . . .                         | 53 |
| IV.05  | Résumé de la fragmentation . . . . .                     | 54 |
| IV.06  | Question ARP . . . . .                                   | 55 |
| IV.07  | Réponse ARP . . . . .                                    | 56 |
| IV.08  | Datagramme ARP . . . . .                                 | 57 |
| IV.09  | Message ICMP . . . . .                                   | 60 |
| IV.10  | Format d'un message ICMP . . . . .                       | 60 |
| IV.11  | " Echo request " vs " Echo reply " . . . . .             | 61 |

|         |  |     |
|---------|--|-----|
| IV.12   | En-tête IGMP . . . . .   | 63  |
| IV.13   | Fonctionnement IGMP . . . . .  | 64  |
| IV.14   | Table de routage . . . . .   | 67  |
| IV.15   | Situation réseau lors du <code>netstat</code> . . . . .                | 69  |
| IV.16   | Exemple de nuage avec routage statique . . . . .                       | 70  |
| IV.17   | Exemple pour routage dynamique . . . . .                               | 71  |
| IV.18   | Topologie pour routage dynamique . . . . .                             | 72  |
| IV.21   | ICMP “redirect” . . . . .  | 74  |
| IV.22   | Interface de “loopback” . . . . .                                      | 75  |
| IV.23   | Illustration du routage direct et indirect . . . . .                   | 76  |
| V.01    | Numéro de port comme numéro de service . . . . .                       | 82  |
| V.02    | UDP encapsulé dans IP . . . . .  | 83  |
| V.03    | Structure de l’en-tête UDP . . . . .                                   | 84  |
| V.04    | Cas du checksum non nul . . . . .                                      | 84  |
| VI.01   | TCP encapsulé dans IP . . . . .  | 89  |
| VI.02   | Structure de l’en-tête TCP . . . . .                                   | 91  |
| VI.03   | Établissement d’une connexion . . . . .                                | 94  |
| VI.04   | Clôture d’une connexion . . . . .                                      | 95  |
| VI.05   | Émission d’un rst . . . . .  | 96  |
| VI.06   | Mécanisme de l’acquittement . . . . .                                  | 97  |
| VI.07   | Principe de la fenêtre glissante . . . . .                             | 98  |
| VI.08   | Détail de la fenêtre glissante . . . . .                               | 99  |
| VI.09   | Exemple de fenêtre glissante . . . . .                                 | 104 |
| VII.01  | Un AS, le monde extérieur, le monde intérieur ! . . . . .              | 111 |
| VII.02  | La route vers H depuis R a une métrique de 2 et passe par R1 . . . . . | 113 |
| VII.03  | Fonctionnement élémentaire . . . . .                                   | 115 |
| VII.04  | L’“horizon partagé” ne résout pas tout ! . . . . .                     | 116 |
| VII.05  | RIP est transporté par UDP/IP . . . . .                                | 118 |
| VII.06  | Format d’un message RIPv2 . . . . .                                    | 118 |
| VII.07  | Relation d’ordre entre deux LSP . . . . .                              | 125 |
| VII.08  | Propagation des LSP par inondation ou “flooding” . . . . .             | 126 |
| VII.09  | Organisation en zones – Hiérarchie de routeurs . . . . .               | 128 |
| VII.10  | Propagation d’un LSP, sans et avec un DR . . . . .                     | 129 |
| VII.11  | Organisation globale de l’en-tête du protocole OSPF . . . . .          | 132 |
| VII.12  | En-tête standard de 24 octets . . . . .                                | 133 |
| VII.13  | En-tête du paquet HELLO . . . . .                                      | 134 |
| VIII.01 | Serveur HTTP virtuel . . . . .   | 137 |
| VIII.02 | Tunnel IP - Principe . . . . .   | 139 |
| VIII.03 | Tunnel IP - cas concret . . . . .                                      | 141 |
| VIII.04 | En-têtes d’IPsec . . . . .   | 145 |
| VIII.05 | Association 1 . . . . .  | 145 |
| VIII.06 | Association 2 . . . . .  | 146 |



|   |     |
|---|-----|
| VIII.07 Association 3 . . . . .   | 146 |
| VIII.08 Association 4 . . . . .   | 146 |
| VIII.04 Proxy . . . . .   | 148 |
| VIII.10 R translate dynamiquement des couples (adresse IP,<br>numéro de port) . . . . .                         | 148 |
| VIII.11 Machine NATP en routeur . . . . .   | 150 |
| VIII.12 Interactions entre <code>natd</code> et le noyau de FreeBSD . . . . .                                   | 151 |
| VIII.13 Static Nat . . . . .  | 152 |
| VIII.14 Configuration multiservices . . . . .   | 153 |
| VIII.15 Configuration simple de filtrage . . . . .  | 155 |
| VIII.16 Translation d'adresse et filtrage IP . . . . .  | 157 |
| IX.01 Organisation hiérarchique des domaines . . . . .  | 169 |
| IX.02 Le “ resolver ” dans son environnement . . . . .  | 171 |
| IX.03 Subdivision hiérarchique des domaines . . . . .   | 172 |
| IX.03 Interrogation locale . . . . .  | 172 |
| IX.05 Interrogation distante . . . . .  | 174 |
| IX.06 Réponse à une requête non formulée . . . . .  | 180 |
| IX.07 Attaque DNS par amplification . . . . .   | 181 |
| IX.08 BIND de l'ISC . . . . .   | 186 |
| X.01 Format d'un e-mail . . . . .   | 192 |
| X.02 MUA - MSA - MTA - LDA - OS . . . . .   | 199 |
| X.03 Trajet d'un mail . . . . .   | 200 |
| X.04 MX primaire et secondaires . . . . .   | 205 |
| X.05 Relation entre Sendmail et le système d'exploitation . . . . .   | 206 |
| X.06 Le cas de POP . . . . .  | 210 |
| X.07 Concentration du mail sur un “ mailhub ” . . . . .   | 213 |
| X.08 Règles de réécriture . . . . .   | 215 |
| XI.01 Agent et Manager dans une relation de type client-serveur . . . . .                                       | 224 |
| XI.02 La racine de l'arbre des OIDs . . . . .   | 230 |
| XI.03 Des agents et un Manager . . . . .  | 234 |
| XI.04 Format des messages SNMP . . . . .  | 235 |
| XI.05 Exemple d'interrogation d'un agent avec l'outil <code>mbrowse</code> . . . . .                            | 244 |
| XI.06 Synthèse graphique des compteurs <code>ifInOctets</code> et<br><code>ifOutOctets</code> sur 24h . . . . . | 245 |
| XI.07 Exemple d'écran de surveillance avec <code>tkined</code> . . . . .  | 247 |
| XII.01 Les sockets, une famille de primitives . . . . .   | 251 |
| XII.02 Relation pile IP, numéro de port et process ID . . . . .   | 252 |
| XII.03 Structures d'adresses . . . . .  | 258 |
| XII.04 Relation client-serveur en mode connecté . . . . .   | 265 |
| XII.05 Relation client-serveur en mode non connecté . . . . .   | 266 |
| XIII.01 Ordre des octets sur le réseau . . . . .  | 277 |

|        |   |     |
|--------|---|-----|
| XIV.01 | Quatre types de serveurs . . . . .            | 303 |
| XIV.02 | Exécution avec et sans threads . . . . .      | 309 |
| XIV.03 | Syslogd . . . . .                             | 317 |
| XIV.04 | Inetd . . . . .                               | 321 |
|        |   |     |
| XV.01  | Structure d'un message HTTP . . . . .         | 329 |
| XV.02  | Environnement système . . . . .               | 334 |
| XV.03  | Algorithme de gestion des processus . . . . . | 340 |
| XV.04  | Usage de la "score board" . . . . .           | 342 |
| XV.05  | Deux type de CGIs . . . . .                   | 343 |

# Liste des tableaux

|         |  |     |
|---------|--|-----|
| I.01    | Quelques valeurs du champs type de l'en-tête IP . . . . .                        | 11  |
| I.02    | Exemples de “ Organizationally Unique Identifier ” (OUI) . .                     | 12  |
| III.01  | Adresses IP des réseaux privés . . . . .   | 34  |
| III.02  | Adresses IP avec une signification particulière . . . . .                        | 37  |
| III.03  | Partitionnement d'une classe C en quatre sous réseaux . . .                      | 38  |
| III.04  | Détail des quatre sous réseaux d'un /26 . . . . .                                | 39  |
| III.05  | Adresses IP privées, notation du CIDR . . . . .                                  | 40  |
| III.06  | Agrégations régionales des blocs IP . . . . .                                    | 41  |
| III.07  | Quelques adresses multicasts du LAN . . . . .                                    | 42  |
| IV.01   | Bits du champ TOS . . . . .  | 49  |
| IV.02   | En-tête des fragments IP vs en-tête datagramme original . .                      | 54  |
| IV.03   | Quelques drapeaux de routage de la commande <code>netstat -r</code>              | 69  |
| V.01    | Extrait succinct du fichier <code>/etc/services</code> . . . . .                 | 85  |
| VI.01   | Drapeaux du champ CODE (en-tête TCP) . . . . .                                   | 92  |
| VII.01  | Quelques valeurs d'états de liens pour OSPF . . . . .                            | 127 |
| X.01    | Quelques champs couramment rencontrés dans un tête de<br>mail . . . . .          | 193 |
| XI.01   | Extrait de la MIB II concernant l'OID <code>tcpConnTable</code> . . . .          | 229 |
| XI.02   | Extrait du début de la MIB-2 . . . . .   | 233 |
| XI.03   | Extrait de la MIB-2 concernant le début du groupe “ system ”                     | 241 |
| XII.01  | Exemples de familles de protocoles pour une socket . . . . .                     | 254 |
| XII.02  | Exemples de type de sockets . . . . .  | 254 |
| XII.03  | Exemples de protocoles associés à une socket . . . . .                           | 255 |
| XIII.01 | Exemples de codes de retours des primitives systèmes pour<br>le réseau . . . . . | 292 |
| XIV.01  | Typologie des applicatifs qui utilisent <code>syslog</code> . . . . .            | 319 |
| XIV.02  | Criticité des messages de log . . . . .  | 319 |
| XV.01   | Codes de retour du protocole HTTP . . . . .                                      | 330 |

|       |   |     |
|-------|---|-----|
| XV.02 | Configuration du modèle “ pre-forked ” d’Apache . . . . . | 335 |
|-------|---|-----|

# Préface

**Attention ! Ce document n'est qu'un support de cours, c'est à dire qu'il ne remplace pas les documents cités dans la bibliographie qui termine chacun des chapitres qui le composent.**

Évidemment imparfaites, pleines d'erreurs involontaires, et surtout incomplètes, ces pages réclament avant tout votre indulgence de lecteur bienveillant : “ Rien n'est constant, tout change ” comme le disait déjà Lao Tseu, 400 ans avant JC. Que dirait-il alors aujourd'hui, concernant les réseaux !!

Ces cours s'accompagnent de travaux pratiques dont le texte ne figure pas ici, ils sont initialement conçus pour les étudiants du Mastère de Systèmes d'Informations Ouverts<sup>2</sup> (SIO) de l'École Centrale Paris, afin de les aider à la compréhension théorique et pratique des réseaux TCP/IP.

**Ce support est en accès libre, c'est à dire mis à la disposition de tous pour un usage personnel ou collectif, sans but lucratif. Sa revente, s'il y a lieu, ne peut être envisagée que pour couvrir les frais induits par sa reproduction. Enfin, sa redistribution sous quelque forme que ce soit, ne peut se concevoir sans cette préface.**

Ne pas hésiter à me contacter en cas de doute sur l'usage :

François Laissus <[fr.laissus@laissus.fr](mailto:fr.laissus@laissus.fr)>

En aucun cas l'auteur ne pourra être tenu responsable des conséquences de l'usage de ce document, qui est fourni tel quel et sans garantie d'aucune sorte. L'usage des informations contenues est donc placé sous la responsabilité pleine et entière du lecteur.

Enfin, si vous pensez que la lecture de ce support vous a apporté quelque chose, que vous avez une remarque à me faire, ou tout simplement me complimenter (ça fait toujours plaisir quoi que l'on puisse en dire ! :) sentez-vous libres de m'envoyer un courrier électronique, je suis toujours ravi d'apprendre que ce travail a pu servir !

Enfin merci de votre intérêt pour ce document, j'espère que vous y trouverez ce que vous cherchez !

---

<sup>2</sup><http://www.mastere-sio.ecp.fr/>



# Première partie

## Introduction à la pile ARPA





# Chapitre I

## Réseaux locaux

### 1 Préambule

Ce cours n'est pas un cours général sur les réseaux mais une présentation minimale de cette technologie pour pouvoir aborder le cours de concepts et programmation TCP/IP sous UNIX.

TCP/IP est le protocole le plus répandu dans le monde grâce à l'Internet.

En 1980 il ne comptait que quelques dizaines d'hôtes, en juin 1996 ce nombre était de 12 millions de machines, réparties en près de 500 000 réseaux (Par comparaison, en février 1995, les mêmes chiffres étaient 4 850 000 machines pour plus de 71 000 réseaux locaux).

En janvier 2003, le nombre de machines<sup>1</sup> directement accessibles sur le réseau était de 180 000 000 selon l'ISC<sup>2</sup>. Depuis on ne compte plus tant la croissance est importante...Pour la France l'AFNIC propose également quelques statistiques<sup>3</sup>...Il n'existe pas de " botin " général du réseau, par contre Bill Cheswick des Bell labs l'a cartographié, et le résultat est fascinant :

<http://www.cheswick.com/ches/map/gallery/index.html>

### 2 Généralités - LANs

#### 2.1 Généralités

Un réseau informatique met en relation des ordinateurs, comme un réseau téléphonique met en relation des personnes.

Des ordinateurs sont dits " en réseaux " dès lors qu'ils partagent une technologie qui leur permet de communiquer ensemble.

Le plus souvent cette technologie se matérialise physiquement par une liaison avec un câble conducteur. Sur ce type de support, un signal électrique

---

<sup>1</sup>Source <http://www.isc.org/ds/>

<sup>2</sup> Internet Software consortium

<sup>3</sup> <http://www.nic.fr/statistiques/>

véhicule les messages informatiques. Il existe d'autres types de supports en pleine expansion comme les liaisons par ondes hertziennes, rayon laser, infra-rouge...

Sans connaissance préalable concernant les réseaux informatiques on peut imaginer quantité d'interrogations à partir de cette hypothèse de raccordement :

- Comment reconnaître un correspondant ?
- Comment dialoguer avec ?
- Comment diffuser l'information à plusieurs correspondants ?
- Comment éviter la cacophonie ?
- Il y a t-il une hiérarchie des machines ?
- Il y a t-il un chef d'orchestre ?
- ...

Toutes ces questions (et bien d'autres) trouveront une réponse dans ce cycle de cours. Ces réponses sont généralement formulées dans un " protocole ", une sorte de mode d'emploi des réseaux. Il y a des centaines de protocoles différents sur l'Internet, certains sont très populaires, d'autres absolument pas.

## 2.2 Modèle de communication OSI

Le concept de base de tout ce cours est celui de la " commutation de paquets ", une vieille idée de l'informatique<sup>4</sup> contrairement à l'approche par circuits virtuels plus utilisée en téléphonie.

Les données à transmettre d'une machine à une autre sont fragmentées à l'émission en petit blocs de quelques centaines d'octets munis de l'adresse du destinataire, envoyées sur le réseau et ré-assemblées à la réception pour reproduire les données d'origine.

Ce concept facilite le partage des possibilités physiques du réseaux (bande passante) et est parfaitement adapté pour une implémentation sur machines séquentielles travaillant en temps partagé (plusieurs communications peuvent alors avoir lieu simultanément et sur une même machine).

Partant de ce concept, un modèle d'architecture pour les protocoles de communication a été développé par l'ISO (International Standards Organisation) entre 1977 et 1984. Ce modèle sert souvent de référence pour décrire la structure et le fonctionnement des protocoles de communication, mais n'est pas une contrainte de spécification.

Ce modèle se nomme OSI comme " Open Systems Interconnection Reference Model ". Les constituants de ce modèle sont si largement employés qu'il est difficile de parler de réseaux sans y faire référence.

Le modèle OSI est constitué de sept couches. À chaque couche est associée une fonction bien précise, l'information traverse ces couches, chacune y apporte sa particularité.

Cette forme d'organisation n'est pas dûe au hasard, c'est celle sur la-

---

<sup>4</sup>Conçu par l'Américain Paul Baran et publié en 1964

quelle les informaticiens ont beaucoup travaillé dans les années soixantes pour définir les caractéristiques des systèmes d'exploitation.

Une couche ne définit pas un protocole, elle délimite un service qui peut être réalisé par plusieurs protocoles de différentes origines. Ainsi chaque couche peut contenir tous les protocoles que l'on veut, pourvu que ceux-ci fournissent le service demandé à ce niveau du modèle.

Un des intérêts majeurs du modèle en couches est de séparer la notion de communication, des problèmes liés à la technologie employée pour véhiculer les données.

Pour mémoire (*figure I.01*) :

- 7** La couche application (Application layer) est constituée des programmes d'application ou services, qui se servent du réseau. Ils ne sont pas forcément accessibles à l'utilisateur car ils peuvent être réservés à un usage d'administration.
- 6** La couche de présentation (Présentation layer) met en forme les données suivant les standards locaux ou particuliers à l'application. Comme, par exemple passer d'une représentation " big endian " ou à une représentation " little endian " ou encore plus complexe comme celle décrite pas les " XdR " (eXternal Data Representation) et qui autorise la transmission de types abstraits de données (structures complexes, arbres, listes chaînées, la liste n'est pas limitative...).
- De nos jour c'est de plus en plus le XML<sup>5</sup> qui occupe cet espace finalement assez peu normé.
- 5** La couche de session (Session layer) effectue l'aiguillage entre les divers services (7) qui communiquent simultanément à travers le même ordinateur connecté et le même réseau. Deux utilisateurs d'une même machine peuvent utiliser la même application sans risque d'inter-actions parasites.
- 4** La couche de transport (Transport layer) garantie que le destinataire obtient exactement l'information qui lui a été envoyée. Cette couche met par exemple en œuvre des règles de renvoi de l'information en cas d'erreur de réception.
- 3** La couche réseau (Network layer) isole les couches hautes du modèle qui ne s'occupent que de l'utilisation du réseau, des couches basses qui ne s'occupent que de la transmission de l'information.
- 2** La couche de donnée (Data link layer) effectue le travail de transmission des données d'une machine à une autre.
- 1** La couche Physique (Physical layer) définit les caractéristiques du matériel nécessaire pour mettre en œuvre le signal de transmission, comme des tensions, des fréquences, la description d'une prise...

---

<sup>5</sup><http://www.w3.org/XML/>

## Modèle en 7 couches de l'OSI

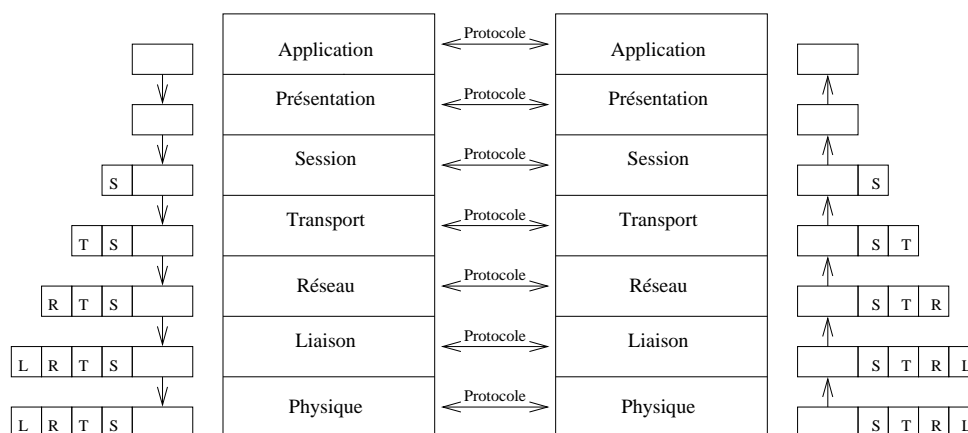


figure I.01 — Modèle en 7 couches de l'OSI

Du niveau 7 de l'application, au niveau 4 du transport, l'information circule dans ce que l'on appelle un " message ", au niveau 3 elle se nomme " packet ", puis " frame " au niveau 2 et " signal " au niveau 1.

Chaque couche ne voit et ne sait communiquer qu'avec la couche qui la précède et celle qui la suit, avec le cas particulier des couches 1 et 7.

L'intérêt de travailler en couches est que lorsque les modalités d'échanges entre chacune d'entres elles sont précisément décrites, on peut changer l'implémentation et les spécificités de la couche elle-même sans que cela affecte le reste de l'édifice.

C'est sur ce principe qu'est bâtie la suite de protocoles désignée par TCP/IP

Quand deux applications A et B discutent entre-elles via le réseau, les informations circulent de la couche 7 vers la couche 2 quand l'application A envoie de l'information sur le réseau, et de la couche 2 vers la couche 7 pour que l'application B reçoive l'information de A.

Le principe de base de cette discussion repose sur le fait que chaque couche du modèle de la machine A est en relation uniquement avec son homologue du même niveau de la machine B.

Quand l'information descend de la couche 7 vers la couche 1, chaque couche " en-capsule " les données reçues avant de les transmettre. Ainsi le volume d'informations s'est accru de quelques centaines d'octets arrivé à la couche 1.

De manière symétrique, quand l'information remonte de la couche physique vers la couche Application, chaque couche prélève les octets qui lui sont propres, ainsi l'application B ne voit-elle que les octets envoyés par l'application A, sans le détail de l'acheminement.

## 3 Réseaux locaux

Le problème intuitif et pratique qui se pose est de relier entre elles par un câble toutes les machines qui veulent communiquer : c'est impossible d'abord pour des raisons techniques, le monde est vaste, puis de politique d'emploi des ressources du réseau, tel réseau qui sert à l'enseignement ne doit pas perturber le fonctionnement de tel processus industriel.

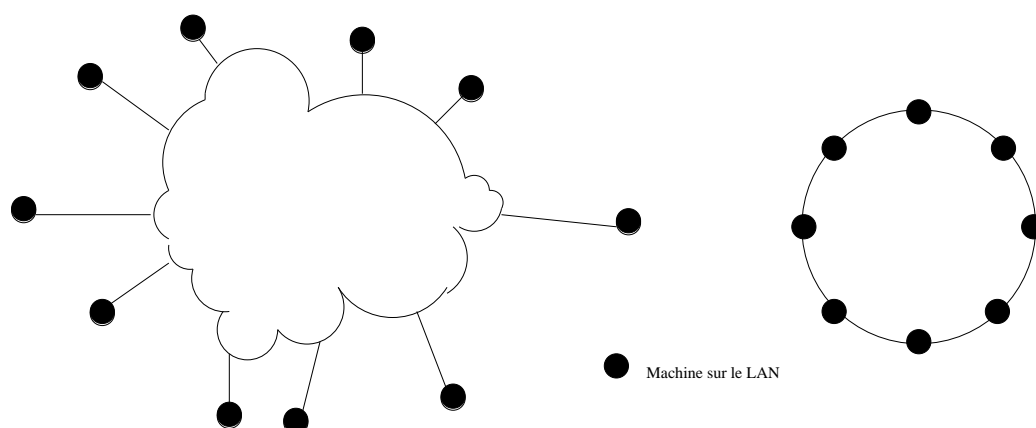
La conséquence est que les réseaux se développent d'abord en local, autour d'un centre d'intérêt commun, avant de se tourner (parfois) vers l'extérieur.

### 3.1 Qu'est-ce qu'un LAN ?

Le terme “ réseau local ” n'est pas clairement défini, cependant tout le monde s'accorde à baptiser de la sorte un réseau, dès lors qu'on lui reconnaît les caractéristiques suivantes :

- Cohabitation de plusieurs protocoles,
- Un même média (même câble par exemple) qui raccorde de multiples machines, peut être de caractéristiques différentes,
- Une bande passante élevée, partagée par tous les hôtes
- La capacité de faire du “ broadcasting ” et du “ multicasting ”,
- Une extension géographique de moins en moins limitée,
- Un nombre de machines raccordées limité,
- Des relations entre les machines placées sur un mode d'égalité, (et non par exemple sur un mode Maître/Esclave comme dans un réseau dont la topologie serait en étoile),
- Une mise en œuvre qui reste du domaine privé, c'est à dire qui ne dépend pas d'un opérateur officiel de télécommunications.

Notez que les notions de “ bande passante ” et “ nombre limité ” (etc. . .) sont volontairement qualitatives. Elles évoluent rapidement avec le temps.



*figure I.02 — Exemple de LANs*

Exemple de types de technologies utilisées dans les LANs :

- Token ring

- IEEE 802 LANs
- Ethernet et Fast-Ethernet
- FDDI (anneau en fibre optique)
- ATM
- 802.11(a,b,g,...)
- ...

## 3.2 WAN - MAN

Un WAN (Wide Area Network) désigne des ordinateurs connectés entre différentes villes (Metropolitan Area Network) ou pays. La technologie utilisée est traditionnellement moins performante que celle d'un LAN, c'est par exemple une ligne téléphonique louée fonctionnant à 64 kbps, une liaison RNIS, ou encore une liaison transatlantique à 1Mbits/secones.

Les améliorations technologiques apportées aux LANs permettent de les étendre de plus en plus géographiquement, celles apportées aux WAN augmentent considérablement les bandes passantes, ces deux tendances font que la distinction entre ces deux types de réseaux est de moins en moins claire.

## 3.3 Communications inter-réseaux

Les réseaux sont appelés à communiquer entre eux et quand cela se produit on parle de communications inter-réseaux ("internetworking").

Le rôle d'une communication inter-réseaux est de gommer les éventuelles différences de technologie d'échange pour permettre à deux réseaux, ou plus, le partage de ressources communes, l'échange d'informations.

Un moyen de faire communiquer deux réseaux distincts passe par l'utilisation de "gateway" ou passerelle.

Un tel dispositif est parfois appelé routeur (router), mais c'est un abus de langage.

- Les hommes se connectent sur les ordinateurs
- Les ordinateurs se connectent sur un réseau
- Les réseaux s'inter-connectent dans un "internet"

## 4 Couche 2 - Liaison (Data Link)

La couche 2 la plus populaire est sûrement celle que l'on nomme abusivement " Ethernet ", du nom du standard publié en 1982 par DEC, Intel Corp. et Xerox. Cette technique repose sur une méthode d'accès et de contrôle dite CSMA/CD ( " Carrier Sense, Multiple Access with Collision Detection " ).

Elle est devenue tellement populaire qu'on parle d'un câble Ethernet, d'une adresse Ethernet, d'une liaison Ethernet...

Plus tard l'IEEE ( " Institute of Electrical and Electronics Engineers " )<sup>6</sup> sous l'instance de son comité 802, publia un ensemble de standards légèrement différents, les plus connus concernant la couche 2 sont 802.2 (Contrôle logique de la liaison – LLC<sup>7</sup> ) et 802.3 (CSMA/CD)

Dans le monde TCP/IP, l'encapsulation des datagrammes IP est décrite dans la RFC 894 [Hornig 1984] pour les réseaux Ethernet et dans la RFC 1042 [Postel et Reynolds 1988] pour les réseaux 802.

En règle générale, toute machine utilisant TCP/IP sur ce type de réseaux doit :

1. être capable d'envoyer et de recevoir un paquet conforme à la RFC 894,
2. être capable de recevoir des paquets conformes aux deux standards,
3. Par contre il est seulement souhaitable que cette machine soit capable d'envoyer des paquets conformes à la RFC 1042.

Par défaut le standard est donc celui de la RFC 894, si une machine peut faire les deux, cela doit être configurable.

De nos jours la couche 802.11 (réseau sans fil - wifi) voit sa popularité croître très vite. Elle est basée sur une méthode d'accès assez proche, le CSMA/CA ( " Carrier Sense, Multiple Access with Collision Avoidance " ). En effet les collisions ne peuvent pas toujours être détectées car les hôtes ne sont pas nécessairement à portée radio directe. Les échanges, quand ils ne sont pas de type " point à point ", passent par un intermédiaire nommé en général " point d'accès " ce qui complique le protocole, et donc la trame, par rapport au CSMA/CD.

### 4.1 Caractéristiques d'Ethernet

#### 4.1.1 Quelques principes fondamentaux

1. Le support de transmission est un Segment = bus = câble coaxial. Il n'y a pas de topologie particulière (boucle, étoile, etc...).
2. Un équipement est raccordé sur un câble par un " transceiver " :  
" Transmitter + receiver = transceiver " (coupleur ou transducteur).  
On parle alors d'une station Ethernet, celle-ci a une adresse unique.

---

<sup>6</sup><http://www.ieee.org/>

<sup>7</sup> " Logical Link Control "

3. Sur le câble circulent des trames, autant de paquets de bits. Il n'y a pas de multiplexage en fréquence, pas de " full duplex " <sup>8</sup>. Une trame émise par une station est reçue par tous les coupleurs du réseau Ethernet, elle contient l'adresse de l'émetteur et celle du destinataire.
4. Un coupleur doit être à l'écoute des trames qui circulent sur le câble. Un coupleur connaît sa propre adresse, ainsi si une trame lui est destinée il la prend, sinon il n'en fait rien.
5. Une station qui veut émettre attend que toutes les autres stations se taisent. Autrement dit, si le câble est libre elle envoie sa trame, sinon elle attend.

Si deux stations émettent en même temps il y a collision. Les deux trames sont alors inexploitable, les deux (ou plus) stations détectent ce fait et réémettent ultérieurement leur paquet en attente.

6. Un réseau Ethernet est donc un réseau à caractère probabiliste car il n'y a pas de chef d'orchestre pour synchroniser les émissions. Cette absence conduit à dire que c'est un réseau égalitaire, une sorte de réunion sans animateur entre personnes polies

En conclusion, la technologie Ethernet est simple, sa mise en œuvre se fait à faible coût. Points à retenir :

- Simplicité et faible coût
- Peu de fonctions optionnelles
- Pas de priorité
- Pas de contrôle sur l'attitude des voisins
- Débit d'au moins 10Mb/s (jusqu'à 1000Mb/s théorique).
- Performances peu dépendantes de la charge, sauf en cas de collisions trop importantes.

#### 4.1.2 Format d'une " Frame Ethernet "

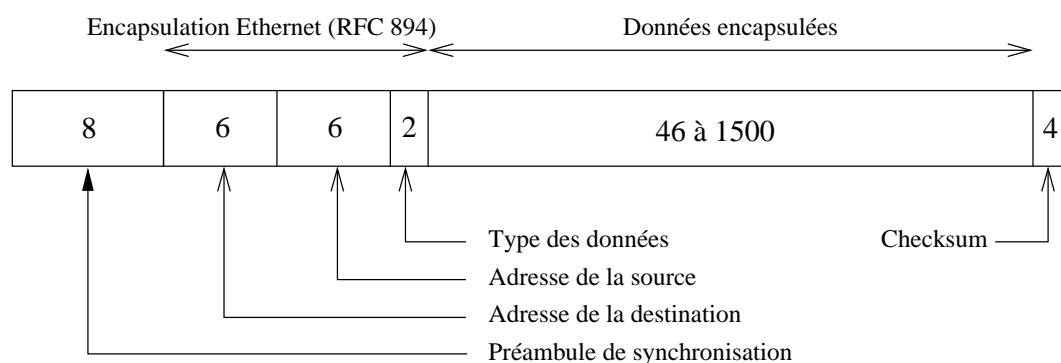


figure I.03 — trame Ethernet

<sup>8</sup> les cartes Ethernet modernes utilisent 4 fils au lieu de deux et ont ainsi des possibilités de " full duplex " que n'avaient pas leurs ancêtres des années 80



Quelques considérations en vrac :

- Dû au débit global de 10Mbits/seconde, le débit est de 10 bits par micro-seconde (en gros un facteur 1000 avec un cpu).
- Une trame a une longueur minimale (72) et une longueur maximale (1526). Si les données ne sont pas assez longues (46 octets) des caractères de remplissage sont ajoutés (“padding”).
- Les octets circulent du premier octet du préambule au dernier octet du CRC.

A l'intérieur de chaque octet le premier bit envoyé est celui de poids faible, etc..

- Le préambule et le SFD (“Start Frame Delimiter”) servent à la synchronisation.
- Adresses d'origine et de destination sont celles respectivement de la machine émettrice et de la machine destinatrice.

Remarque importante : il faut connaître l'adresse de son correspondant pour pouvoir lui envoyer un paquet ! À ce stade de l'exposé on ne sait pas encore comment faire quand on ignore cette information.

- Le champ “type” est deux octets qui désignent le type des données encapsulées :

| Type | Données |
|------|---------|
| 0800 | IP      |
| 0806 | ARP     |
| 0835 | RARP    |
| 6000 | DEC     |
| 6009 | DEC     |
| 8019 | DOMAIN  |
| ...  | ...     |

#### 4.1.3 Adresses IEEE 802.3 ou Ethernet

Pour ces deux standards, l'adresse est codée sur 6 octets soit 48 bits. Pour un hôte sur un réseau, cette adresse est ce que l'on appelle son adresse physique (“hardware adresse”) par opposition à son adresse logique qui interviendra lors de l'examen de la couche 3.

En fait cette adresse est divisée en deux parties égales, les trois premiers octets désignent le constructeur, c'est le OUI (“Organizationally Unique Identifier”) distribué par l'IEEE<sup>9</sup> les trois derniers désignent le numéro de carte, dont la valeur est laissée à l'initiative du constructeur qui possède le préfixe.

L'IEEE assure ainsi l'unicité de l'attribution des numéros de constructeurs, par tranches de 2<sup>24</sup> cartes<sup>10</sup>

Chaque constructeur assure l'unicité du numéro de chaque carte fa-

<sup>9</sup><http://standards.ieee.org/regauth/oui/index.shtml>

<sup>10</sup>La liste à jour est accessible à cette url <http://standards.ieee.org/regauth/oui/oui.txt> ou à la fin de la RFC 1700 (page 172) “Ethernet vendors address components”

briquée. Il y a au maximum  $2^{24}$  cartes par classe d'adresses.

Cette unicité est primordiale car le bon fonctionnement d'un LAN requiert que toutes les stations aient une adresse physique différente. Dans le cas contraire le réseau et les applications qui l'utilisent auront un comportement imprévisible le rendant impraticable.

Nous aurons l'occasion de rencontrer à nouveau ce soucis d'unicité de l'adresse physique lorsque nous examinerons les protocoles ARP et RARP (cf cours ARP/RARP pages 55 et 58) et avec CARP ("Common Address Redundancy Protocol") lorsque nous parlerons des hôtes virtuels, page 137.

Exemple d'adresse physique en représentation hexadécimale :

|                   |  |
|-------------------|--|
| 08:00:09:35:d5:0b | 08:00:09 est attribué à la firme Hewlett-Packard |
|                   | 35:d5:0b est l'adresse de la carte               |

D'autres constructeurs, capturés au hasard des réseaux :

|          |                               |
|----------|-------------------------------|
| 00:11:24 | Apple Computer                |
| 00:00:0C | Cisco Systems, Inc.           |
| 00:06:5B | Dell Computer Corp.           |
| 08:00:20 | Sun Microsystems              |
| AA:00:04 | Digital Equipment Corporation |
| 00:10:5A | 3Com Corporation              |
| ...      | ...                           |

#### 4.1.4 Unicast, multicast et broadcast

Dans la plupart des technologies de LAN, toutes les stations peuvent écouter toutes les trames qui leur parviennent. Beaucoup d'entre elles ne leur sont pas destinées, et s'il fallait que le système d'exploitation qui gère l'interface réseau s'interrompt à chaque fois pour les examiner, il ne serait pas très utilisable pour les applications de l'utilisateur, parceque tout le temps interrompu par ces événements réseau.

Pour éviter cette situation, le logiciel embarqué dans l'interface réseau est paramétré (par le système d'exploitation) pour filtrer les paquets non voulus car non nécessaires au bon fonctionnement en réseau. Ce paramétrage peut changer d'une station à une autre.

Il est également possible de ne pas filtrer, c'est une propriété utilisée par les analyseurs de trames, comme par exemple l'outil `tcpdump`. La carte fonctionne alors en mode dit "promiscuous", qui n'est donc pas son mode de fonctionnement standard.

Le filtrage s'appuie sur trois types d'adressages :

**unicast** L'adresse MAC est constituée de la combinaison de 48 bits qui la rend unique. Ce mode d'adressage est typique d'échanges entre deux stations uniquement. C'est l'essentiel du trafic sur un LAN. Le filtrage peut s'effectuer en ne retenant que les trames qui ont l'adresse MAC de la station locale et donc écarter les autres trames de type unicast.

**broadcast** Tous les bits de l'adresse MAC sont à 1.

Toutes les stations d'un réseau sont destinataires de tels paquets, que leur filtrage doit laisser passer, avec les inconvénients cités précédemment.

Ce mode d'adressage ne devrait être utilisé par les protocoles qu'uniquement quand il n'est pas possible de faire autrement. Par exemple pour obtenir une information que seule une station inconnue sur le LAN possède. C'est le cas des protocoles ARP et RARP (cf cours ARP/-RARP pages 55 et 58)

Utilisé abusivement, le broadcast est une gêne.

**multicast** Il existe un préfixe particulier 01:00:5E, non dédiée à un constructeur car dit de " multicast ", que nous examinerons dans le cas d'IP page 42.

Ce mode de d'adressage est réservé le plus généralement à la découverte passive (par l'écoute de messages d'avertissement) ou à la recherche (par l'émission de messages de sollicitation) de voisins de LAN ayant des propriétés particulières.

Le filtrage des sollicitations et leurs réponses peut être configuré à la carte sur chaque station, en fonction des impératifs et besoins de fonctionnement.

Ce mode de fonctionnement est assez économe des ressources du réseau, puisqu'une seule station émet une information qui est traitée par toutes celles qui sont intéressées, et elles seules.

Toutes les adresses qui ne sont ni du type broadcast ni du type multicast sont du type unicast.

## 4.2 Différences Ethernet - 802.2/802.3

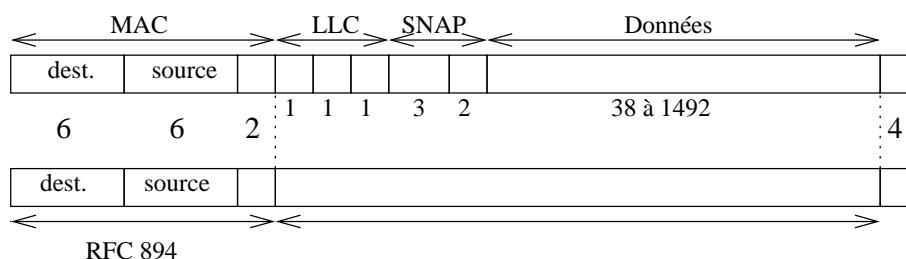


figure I.04 — Différences Ethernet 802.2/802.3

- On remarque que le champ " taille " de la trame 802.3 est à la place du champ " type " de la trame Ethernet. La différenciation s'effectue à partir de la valeur de ces deux octets. On remarque également que le comité 802 a choisi de subdiviser la couche 2 ISO en deux sous couches : MAC et LLC.

- Tous les numéros de protocole sont supérieurs à 1500<sup>11</sup> qui est la longueur maximale des données encapsulées. Donc une valeur inférieure ou égale à ce seuil indique une trame 802.3.

**MAC** = “ Medium Access Control ”

Cette couche est concernée par la gestion de l’adresse physique de la technologie de LAN employée (comme “ token-ring ” par exemple)

**LLC** = “ Logical Link Control ”

Définit ce qui est nécessaire aux multiples couches supérieures possibles pour utiliser et partager les ressources du lan en même temps.

Le comité 802.2 a également prévu plusieurs options, dont deux principalement utilisées :

LLC type 1

Les trames sont délivrées en mode datagramme c’est à dire selon le principe du “ best effort ” (on fait au mieux sans garantie de résultat).

LLC type 2

Les trames sont délivrées avec une garantie de bon acheminement. L’usage du LLC de type 2 entraîne l’ajout de champs dans l’entête pour numéroter les paquets, ajouter des acquittements, des synchronisations, etc...C’est le protocole HDLC comme “ High-level Data Link Control ”.

Un travail qui est normalement dévolu à la couche de transport et qui donc parasite beaucoup la lisibilité de l’ensemble.

## 5 Interconnexion - Technologie élémentaire

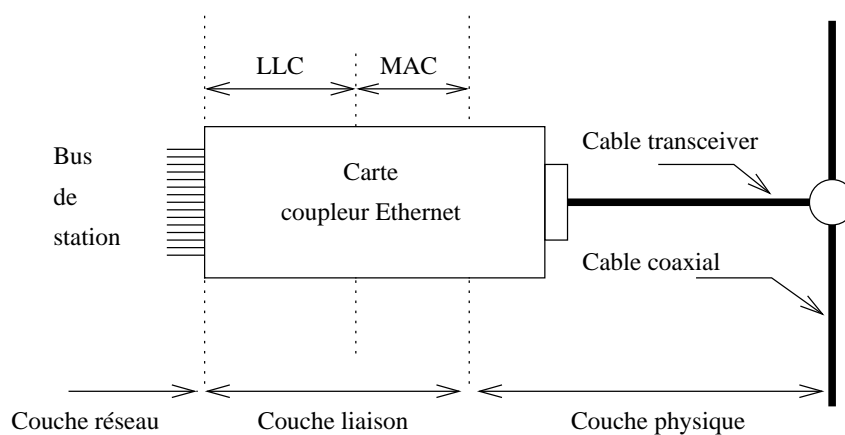


figure I.05 — Interconnexion - Technologie élémentaire

<sup>11</sup> Le plus petit numéro de protocole est celui d’IP : 0800 hexadécimal. Ce qui fait en décimal :  $8 \times 16^2 + 0 \times 16^1 + 0 \times 16^0 = 2048$

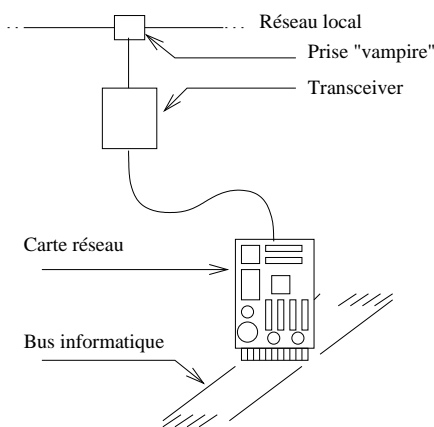
L'interconnexion ne se limite pas au niveau Ethernet.

Quelques notions de technologie de base et donc très succinctes sont nécessaires pour bien comprendre la suite de ce cours.

## 5.1 Raccordement

*Figure I.06* l'hôte est raccordé à l'aide d'une prise de type " vampire " et d'un " transceiver ".

Dans cette technologie de raccordement, le support est un gros câble jaune, dit encore " Thick Ethernet " ou Ethernet standard, ou encore 10Base5 (10 comme 10Mbits/s, Base comme " Baseband ", 5 comme 500 mètres).



*figure I.06 — Prise vampire*

### 5.1.1 10Base5

Quelques particularités du 10Base5 :

- Longueur maxi est 500 mètres, pour un maximum de 100 stations.
- C'est une " vieille " technologie très bien normalisée mais dépassée.
- Pas de perturbation quand on ajoute une station : la pose d'une nouvelle prise n'interrompt pas la continuité du réseau.
- Coût non négligeable.
- Déplacement d'une station non aisé, en plus on perd la prise vampire, elle reste sur le câble.

Pour les câblages rapides on préfère le 10Base2 ou " Thin Ethernet " ou encore Ethernet fin (2 comme 200 mètres).

### 5.1.2 10Base2

Quelques particularités du 10Base2 :

- Longueur maxi de 185 mètres avec un maximum de 30 stations.
- La topologie impose de mettre les stations en série avec un minimum de 0.5 mètre entre chaque.
- Le raccord se fait avec un " transceiver " en T (BNC bien connu des électroniciens).
- Il faut un bouchon de 50 ohms à chaque extrémité du réseau (2).
- Technique très bon marché, souple, les cartes intègrent le transducteur.
- Il faut rompre la continuité du réseau pour ajouter une nouvelle station, ce qui l'empêche de fonctionner durant l'opération. C'est un in-

convénient de taille sur un réseau très utilisé.

- Cette technique est en outre assez sensible aux perturbations électromagnétiques.

Les désavantages du 10Base2 imposent généralement l'usage du 10BaseT dans toute structure dépassant quelques machines (5 à 10). Le 10BaseT règle définitivement le problème de l'ajout ou du retrait d'une machine sur le LAN (T comme " Twisted Pair " ou paires torsadées).

Cette technique impose l'usage d'une boîte noire réseau nommée " HUB " <sup>12</sup> ou moyeu. Celle-ci simule la continuité dans le cas du retrait d'une station.

### 5.1.3 10BaseT

Quelques particularités du 10BaseT :

- Une double paire torsadée de câble suffit.
- La longueur maximale entre le moyeu et la station est de 100 mètres.
- Le moyeu impose une architecture en étoile.
- Le raccordement au transducteur se fait à l'aide d'une prise du type RJ45, très fragile (ne pas marcher dessus ! :). Le raccordement du HUB (page 18) au reste du réseau se fait par 10Base2, en fibre optique, ou tout simplement par chaînage avec un autre HUB (" Daisy chain ").
- Cette technique est d'une très grande souplesse d'utilisation elle impose néanmoins l'acquisition de HUB, très peu onéreux de nos jours.
- Cette technique des paires torsadées est très sensible aux perturbations électromagnétiques.

Aujourd'hui le 100BaseT équipe la majeure partie des équipements professionnels, 100 comme 100 Mbits/s.

Enfin la fibre optique est utilisée de plus en plus souvent pour effectuer les liaisons point à point.

### 5.1.4 Fibre optique

Quelques particularités de la fibre optique :

- La plus utilisée est la fibre multimode 62.5/125.0  $\mu\text{m}$
- Usage d'un transducteur optique pour assurer la transformation entre le signal lumineux (un laser) et le signal électrique.
- La distance maximale entre deux points est 1,5 km.
- La fibre est insensible aux perturbations électromagnétiques, elle permet en outre le câblage de site important (plusieurs  $\text{km}^2$ ).
- La fibre permet d'atteindre des vitesses de transmission supérieures aux 10Mbits/100Mbits/1000Mbits maintenant courants sur des paires de fils en cuivre.
- Les nouvelles technologies issues des recherches les plus récentes promettent des fibres multifréquences (1024 canaux par fibre) avec pour

---

<sup>12</sup>Voir au paragraphe 5.3 page 18

chaque canal une bande passante de plusieurs giga-octets. Ces nouveaux médias auront une bande passante de plusieurs téra-octets par secondes...

- Son principal désavantage est un coût élevé au mètre (de l'ordre d'une dizaine d'€ pour un câble d'un mètre cinquante) et la nécessité d'avoir des transducteurs au raccordement de tous les appareils contenant de l'électronique (serveur, switch, routeur). Un tel module peut coûter de l'ordre de 500 à 1000 €...

### 5.1.5 Conclusion

Construire un réseau local consiste à juxtaposer des composants de base très bien maîtrisés, une sorte de mécano car tous les supports sont mixables.

Ne plus installer les technologies les plus anciennes 10Base5, 10Base2 ou même 10BaseT, préférer l'usage du 100BaseT ou du 1000BaseT qui sont devenus un standard courant du précablage.

En effet le câblage constitue les fondations d'un réseau, le faire proprement d'emblé évite une source continuelle d'ennuis pas la suite ! Les besoins en bande passante d'aujourd'hui ne préfigurent sans doute pas encore les besoins de demain (vidéo haute définition sur tous les postes...), il faut donc prévoir très large dès la conception initiale.

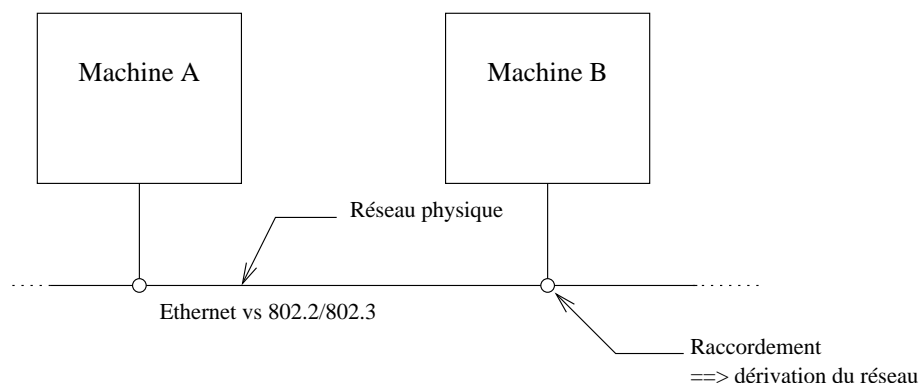


figure I.07 — Technologie de liaison

## 5.2 Répéteur

À une technologie particulière correspondent forcément des limitations dues aux lois de la physique. Par exemple en technologie Ethernet la longueur maximale d'un brin ne peut pas excéder 180 mètres. Pour pallier à cette déficience on utilise des répéteurs (" repeaters ").

Répéteurs :

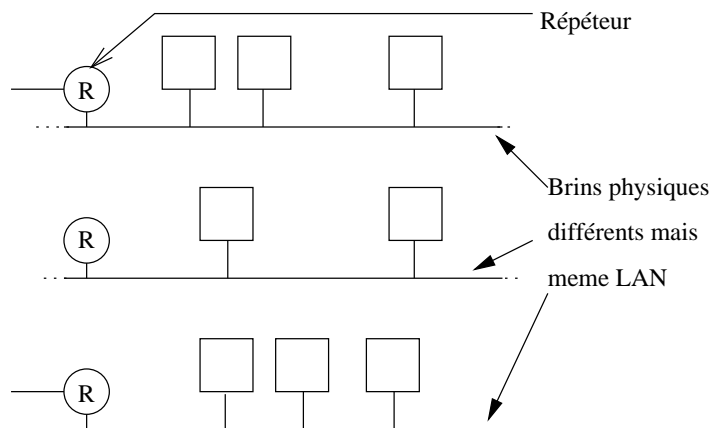


figure I.08 — Plusieurs répéteurs mais toujours le même lan

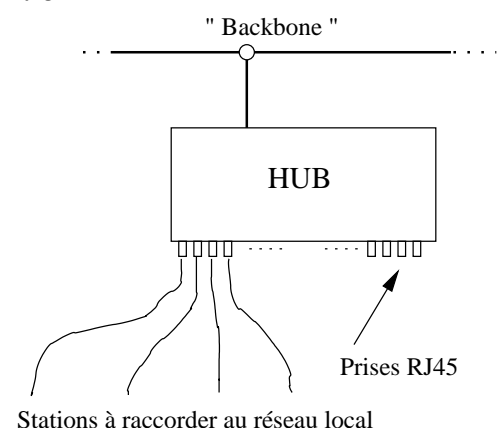
- Agit uniquement au niveau de la couche 1 ISO, c'est un “ amplificateur de ligne ” avec ses avantages et aussi l'inconvénient de transmettre le bruit sans discernement : il n'y a aucun filtrage sur le contenu.
- Relie deux brins d'une même technologie en un seul LAN car les trames sont reproduites à l'identique.
- En 10Base5, l'usage d'un répéteur fait passer la limite des 500 mètres à 1000 mètres...
- Il n'y a aucune administration particulière, sinon de brancher la boîte noire à un emplacement jugé pertinent.
- C'est un élément “ bon marché ”.

### 5.3 Concentrateur

Un concentrateur (ou “ HUB ”, moyeu) :

- Est aussi nommé étoile ou multirépéteur.
- Les HUB n'ont pas d'adresse Ethernet, sauf certains modèles évolués, gérables à distance (TELNET,SNMP,...). On parle alors de “ hubs intelligents ” parcequ'ils permettent d'associer des ports entres-eux.

figure I.09 — Concentrateur





Un hub assure la continuité du réseau sur chacune de ses prises, que l'on y branche ou pas un hôte. En cela il agit uniquement au niveau de la couche 1 ISO. Il ne limite pas le nombre de collisions et n'améliore pas l'usage de la bande passante. Son seul intérêt est de donc permettre le branchement ou le débranchement des stations sans perturber le fonctionnement global du réseau.

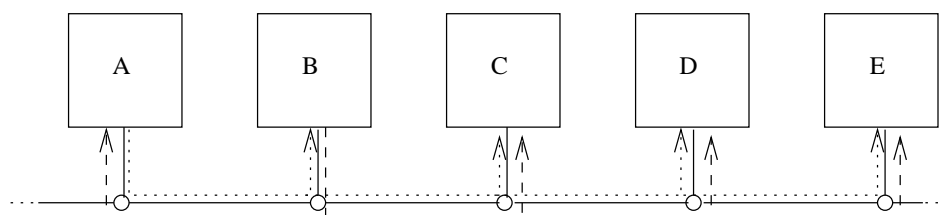
Les hubs peuvent être chaînés entres-eux ; souvent ils sont reliés au backbone local par une autre technologie que la paire torsadée (fibre optique...).

Dans le cas de " hubs intelligents " les ports sont associés les uns aux autres par groupes de fonctionnement.

## 5.4 Ponts

La technologie CSMA/CD atteint vite ses limites quand le réseau est encombré. Une amélioration possible quand on ne peut pas changer de technologie (augmentation du débit) est d'utiliser un ou plusieurs ponts (" bridges ") pour regrouper des machines qui ont entre-elles un dialogue privilégié.

Dialogue entre deux stations, sans pont :

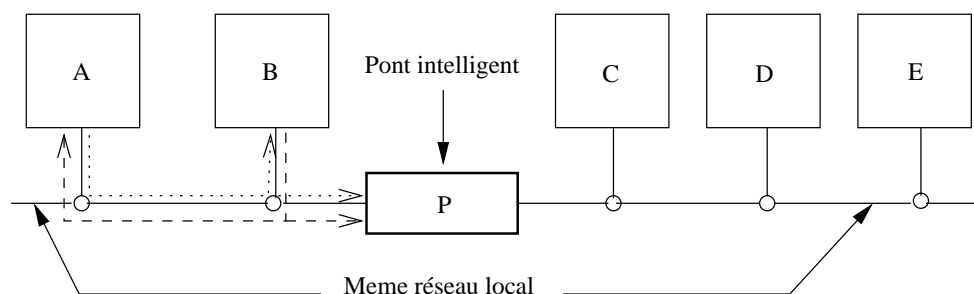


Le dialogue entre A et B perturbe l'éventuel dialogue entre D et E.

*figure I.10 — Dialogue sans pont*

De nos jours le pont en tant que tel est de moins en moins utilisé par contre le principe de son fonctionnement se retrouve, entres autres, dans les commutateurs (paragraphe suivant) et dans les points d'accès sans fil (" wireless ").

Dialogue entre deux stations, avec pont :



*figure I.11 — Dialogue avec pont*

On peut remarquer que les échanges locaux à chaque branche du pont s'effectuent au mieux des possibilités de la bande passante, le pont a donc

multiplié par deux la capacité globale du trafic réseau vis à vis de certains échanges.

Un pont :

- Agit au niveau de la couche 2 ISO, donc au niveau de la trame physique. Son action est plus que physique elle est aussi logique puisqu’il y a lecture et interprétation des octets véhiculés. Le résultat de ce travail logique (apprentissage) consiste à isoler le trafic sur certains tronçons d’un LAN. À cause de ce travail on parle généralement de “ ponts intelligents ” ou de “ ponts transparents ” car la phase d’apprentissage est automatique !
- Réduit le taux de collisions en réduisant le trafic inutile, donc améliore l’usage de la bande passante. Sur la *figure 1.11* les machines A et B peuvent dialoguer sans perturber le dialogue entre les machines D et E. Par contre dans le cas d’un dialogue entre A et E le pont ne sert à rien.
- Moins cher qu’un routeur et plus rapide (services rendus moins complets).
- Relie deux segments (ou plus) en un seul LAN, les trames transmises sont reproduites à l’identique.
- Un pont contient un cpu, il est en général administrable à distance car on peut agir sur la table de filtrages (ajout, contraintes de filtrages, etc...). Dans ce cas un pont a une adresse Ethernet.
- Les ponts interdisent que les réseaux aient des boucles, un protocole nommé STP ( “ Spanning Tree Protocol ” ) désactive automatiquement le ou les ponts qui occasionne(nt) un bouclage des trames.
- Il existe des ponts entre Ethernet et Token-ring, on parle alors de “ ponts à translations ”.
- Attention, un pont ne s’occupe que des adresses de type unicast, il ne filtre pas les types broadcast et multicast.
- On peut remarquer que dans le cas de figure ou le trafic est strictement contenu d’un côté et de l’autre du pont, alors la bande passante globale du LAN est multipliée par deux. Bien sûr cette remarque n’est plus valable dès lors qu’une trame franchit le pont.

## 5.5 Commutateurs

Aligner des stations sur un même réseau local constitue une première étape simple et de faible coût pour un réseau local d’entreprise. Le revers d’une telle architecture est que le nombre de collisions croît très vite avec le trafic, d’où une baisse très sensible de la rapidité des échanges due à ce gaspillage de la bande passante.

L’usage de ponts peut constituer une première solution mais elle n’est pas totalement satisfaisante dans tous les cas de figure, comme nous avons pu le remarquer au paragraphe précédent.

Depuis plus d’une dizaine d’années est apparue une technologie nommée “ Intelligent Switching Hub ” (ISH) – commutateur intelligent – qui utilise

le concept de commutation parallèle et qui a révolutionné l'organisation des réseaux locaux.

D'aspect extérieur ces équipements se présentent comme un hub mais ont en interne un cpu suffisamment puissant et un bus interne de données suffisamment rapide pour mettre en œuvre une logique de commutation raffinée.

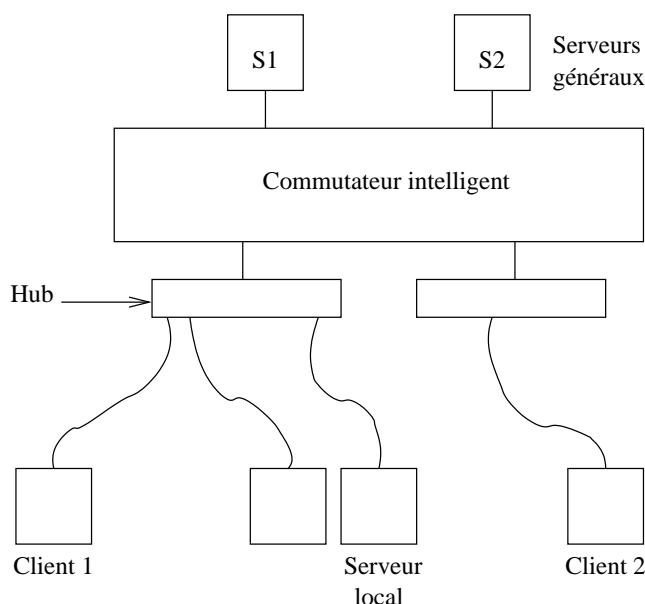
Lorsqu'une trame se présente sur l'un des ports du commutateur elle est (ou n'est pas) re-routée vers un autre port en fonction de l'adresse physique du destinataire. Il existe plusieurs différences entre un pont et un commutateur :

- Un commutateur peut mettre simultanément plusieurs ports en relation, sans que le débit de chacun en souffre. Par exemple un commutateur de 8 ports en 100BaseT peut supporter quatre connexions port source/port destination simultanées à 100 Mbit/s chacune, ce qui donne un débit global de 400 Mbit/s qui doit pouvoir être supporté par le bus interne ou “ fond de panier ”.

D'un point de vue plus théorique, un commutateur à  $N$  ports à 100 Mbit/s chacun a un débit maximum de  $N \times 100/2 = 50 \times N \text{ Mbit/s}$ .

- Si une trame est à destination d'un port déjà occupé, le commutateur la mémorise pour la délivrer sitôt le port disponible.
- Un commutateur fonctionne comme un pont pour établir sa carte des adresses mais il peut aussi travailler à partir d'une table préconfigurée.
- Un commutateur peut fonctionner par port (une seule station Ethernet par port) ou par segment (plusieurs stations Ethernet par port).

Avec un commutateur, il est aisé d'organiser un réseau en fonction de la portée des serveurs des postes clients associés. La *figure I.12* illustre ce principe :



*figure I.12 — Commutateur*

Le trafic réseau entre le “ client 1 ” et le serveur “ S2 ” ne perturbe pas

le trafic entre le “ client 2 ” et le serveur “ S1 ”. De même le trafic entre le “ client 1 ” et le “ serveur local ” n’est pas vu du “ client 2 ”.

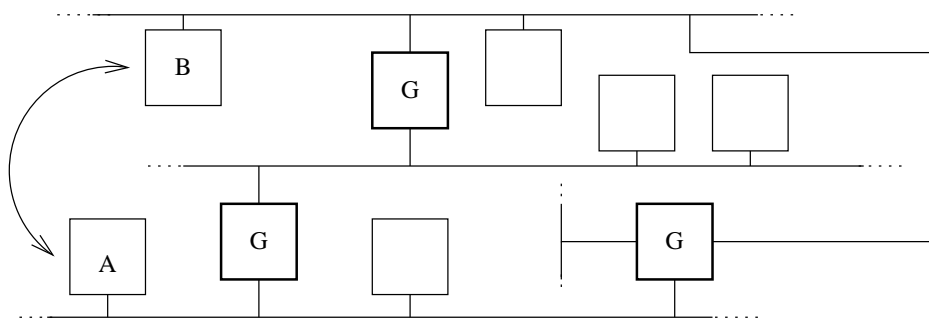
Les commutateurs étiquettent les trames avec un identificateur du VLAN auquel elles appartiennent. Cette étiquette se résume par deux octets ajoutés dans la trame, selon les recommandations du comité 802 (norme 802.1Q).

## 5.6 Passerelles — Routeurs

Pour raccorder deux LANs non forcément contigus il faut faire appel à ce que l’on désigne “ une passerelle ” ( “ gateway ”). Son rôle est de prendre une décision sur la route à suivre et de convertir le format des données pour être compatible avec le réseau à atteindre (en fonction de la route).

Souvent, et c’est le cas avec TCP/IP, la fonction de conversion n’est pas utilisée, la fonction de routage donne alors son nom à l’appareil en question (éponyme), qui devient un “ routeur ” ( “ router ”).

Le problème du routage entre A et B :



Plusieurs chemins sont possibles pour aller de A à B, d’où la nécessité d’une stratégie.

figure I.13 — Fonction routage

La fonction passerelle consiste aussi en traduction de protocoles :

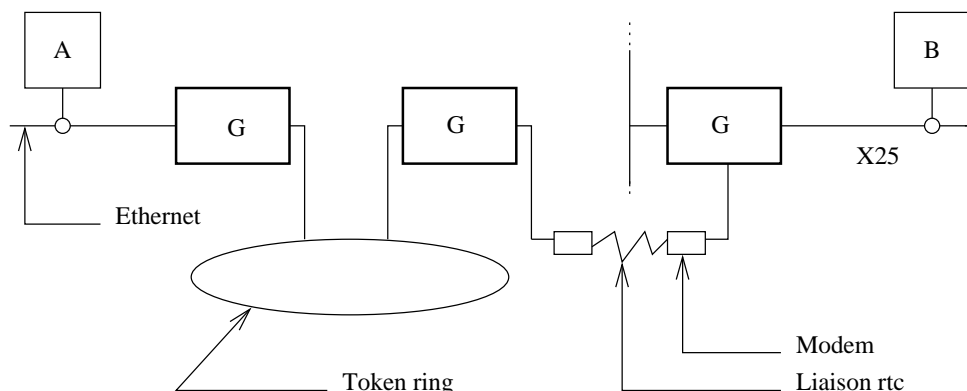


figure I.14 — Traduction de protocoles

Un routeur :

- Agit au niveau de la couche 3. Il prend des décisions de destination.

- Possède au moins deux interfaces réseau (pas forcément identiques).
- Contient un cpu et un programme très évolué, il est administrable à distance.
- Remplit également les fonctions d'un pont (B-routeur) mais les brins ainsi reliés ne forment en général plus un LAN car les adresses physiques contenues dans les trames ne servent plus à identifier le destinataire. Il faut une autre adresse qui dépend de la pile au-dessus (exemple adresse IP). Il existe cependant des possibilités de simuler un même LAN bien que les trame traversent un routeur (cf cours ARP (page 55)).

## 6 Bibliographie

Pour en savoir plus :

**RFC 0894** S C. Hornig, “ Standard for the transmission of IP datagrams over Ethernet networks ”, 04/01/1984. (Pages=3) (Format=.txt)

**RFC 1042** S J. Postel, J. Reynolds, “ Standard for the transmission of IP datagrams over IEEE 802 networks ”, 02/01/1988. (Pages=15) (Format=.txt) (Obsoletes RFC0948)

- Radia Perlman — “ Interconnections – Bridges and Routers ” — Addison–Wesley
- Radia Perlman — “ Interconnections Second Edition ” – Bridges, Routers, Switches, and Internetworking Protocols — Addison–Wesley



# Chapitre II

## Introduction à IP

### 1 TCP/IP et l'Internet - Un peu d'histoire

En 1969 aux États Unis, l'agence gouvernementale DARPA lance un projet de réseau expérimental, basé sur la commutation de paquets. Ce réseau, nommé ARPANET, fut construit dans le but d'étudier les technologies de communications, indépendamment de toute contrainte commerciale<sup>1</sup>

Un grand nombre de techniques de communication par modems datent de cette époque.

L'expérience d'ARPANET est alors si concluante que toutes les organisations qui lui sont rattachées l'utilisent quotidiennement pour leurs messages de service.

En 1975, le réseau passe officiellement du stade expérimental au stade opérationnel.

Le développement d'ARPANET ne s'arrête pas pour autant, les bases des protocoles TCP/IP sont développés à ce moment, donc après que ARPANET soit opérationnel.

En Juin 1978 Jon Postel<sup>2</sup> définit IPv4 et en 1981 IP est standardisé dans la RFC 791 [J. Postel 1981].

En 1983 les protocoles TCP/IP sont adoptés comme un standard militaire et toutes les machines sur le réseau commencent à l'utiliser. Pour faciliter cette reconversion, la DARPA demande à l'université de Berkeley d'implémenter ces protocoles dans leur version (BSD) d'unix. Ainsi commence le mariage entre ce système d'exploitation et les protocoles TCP/IP.

L'apport de l'Université de Berkeley est majeur, tant au niveau théorique (concept des sockets) qu'au niveau de l'utilisateur, avec des utilitaires très homogènes qui s'intègrent parfaitement au paradigme d'usage existant (rnp,

---

<sup>1</sup>Lancé en France en 1972, le projet " Cyclades ", sous la responsabilité de Louis Pouzin, était également basé sur la commutation de paquets et l'usage de datagrammes. Il reliait quelques grands centres universitaires en France (Lille, Paris, Grenoble,...) et en Europe. Il est resté opérationnel jusqu'en 1978, date à laquelle faute de crédit il a été abandonné au profit de X25, préféré par les opérateurs de télécoms nationaux.

<sup>2</sup>Jon Postel est décédé le 16 Octobre 1998 à l'âge de 55 ans, c'est le premier pionnier de l'Internet décédé, on peut consulter par exemple : <http://www.isi.edu/postel.html>

rsh, rlogin. . .).

Depuis cette époque, un nouveau terme est apparu pour désigner cette interconnexion de réseaux, l'Internet, avec un " i " majuscule.

Le succès de cette technologie est alors très important et suscite un intérêt croissant de la part d'acteurs très divers, et en particulier La " National Science Foundation " qui y voit un intérêt majeur pour la recherche scientifique et soutient donc ce nouveau moyen de mettre en communication tous les chercheurs.

Depuis 1990, ARPANET n'est plus, pourtant le terme Internet demeure il désigne maintenant un espace de communication qui englobe la planète tout entière. Des millions de sites partout sur la surface du globe y sont connectés.

Depuis 1994, l'Internet s'est ouvert au commerce, surtout avec l'apparition en 1991 d'un nouvel outil de consultation, le " World Wide Web " ou " Web " et ses interfaces populaires : Mosaic<sup>3</sup>, Netscape, Mozilla, Firefox, Konqueror. . . La liste n'est pas exhaustive !

Depuis 1995, pour faire face à sa popularité fortement croissante et aux demandes de transactions sécurisées, le protocole évolue et une nouvelle version, la version 6 (IPng puis tout simplement IPv6), est définie et en cours de déploiement expérimental.

Les protocoles désignés par TCP/IP ont également envahi les réseaux locaux eux-mêmes, car il est plus facile d'utiliser les mêmes protocoles en interne et en externe.

Pour les utilisateurs, l'accès à l'Internet est possible à l'aide d'une collection de programmes spécialisés si faciles à utiliser que l'on peut ignorer tout (ou presque) de leur fonctionnement interne.

Seul les programmeurs d'applications réseaux et les administrateurs de systèmes ont besoin d'en connaître les arcanes.

Les services réseaux les plus populaires sont principalement :

- Le courrier électronique qui permet l'échange de messages entre usagers.
- Les innombrables forums de discussion ( " news " ).
- Le transfert de fichiers entre machines ( " ftp " et ses dérivés comme " fetch ", " wget ", " curl " . . . ).
- Le " remote login ", ou ses équivalents cryptés ( " ssh ", qui permet à un utilisateur de se connecter sur un site distant, depuis son poste local.
- Les serveurs inter-actifs. Les " anciens " se nommaientarchie, gopher, veronica, wais... Désormais ils sont rendus obsolètes par le " web " (protocole **http**).
- Puis maintenant la radio, la vidéoconférence, la réalité virtuelle avec le VRML, le " chat ", les bourses d'échanges point à point, les " blogs " forme évoluée des pages personnelles, etc . . .

...

En conclusion de ce paragraphe sur l'historique on peut dire que l'Internet est une collection apparemment anarchique (il n'y a pas de structure hié-

<sup>3</sup> <http://archive.ncsa.uiuc.edu/SDG/Software/Mosaic/NCSAMosaicHome.html>



rarchique et centralisée) de réseaux inter-connectés et appartenant à divers propriétaires.

On distingue trois niveaux : les réseaux au sein des organisations (lans), les réseaux régionaux et les réseaux de transit.

Le site de l'Association Fnet indique quelques pointeurs intéressants sur l'historique de l'Internet<sup>4</sup> (en anglais).

## 2 Caractéristiques de TCP/IP

Le succès de TCP/IP, s'il vient d'abord d'un choix du gouvernement américain, s'appuie ensuite sur des caractéristiques intéressantes :

1. C'est un protocole ouvert, les sources (C) en sont disponibles gratuitement et ont été développés indépendamment d'une architecture particulière, d'un système d'exploitation particulier, d'une structure commerciale propriétaire. Ils sont donc théoriquement transportables sur n'importe quel type de plate-forme, ce qui est prouvé de nos jours.
2. Ce protocole est indépendant du support physique du réseau. Cela permet à TCP/IP d'être véhiculé par des supports et des technologies aussi différents qu'une ligne série, un câble coaxial Ethernet, une liaison louée, un réseau token-ring, une liaison radio (satellites, "wireless" 802.11a/b/g), une liaison FDDI 600Mbits, une liaison par rayon laser, infrarouge, xDSL, ATM, fibre optique, la liste des supports et des technologies n'est pas exhaustive...
3. Le mode d'adressage est commun à tous les utilisateurs de TCP/IP quelle que soit la plate-forme qui l'utilise. Si l'unicité de l'adresse est respectée, les communications aboutissent même si les hôtes sont aux antipodes.
4. Les protocoles de hauts niveaux sont standardisés ce qui permet des développements largement répandus et inter-opérables sur tous types de machines.

La majeure partie des informations relatives à ces protocoles sont publiées dans les RFCs (Requests For Comments). Les RFCs contiennent les dernières versions des spécifications de tous les protocoles TCP/IP, ainsi que bien d'autres informations comme des propositions d'améliorations des outils actuels, la description de nouveaux protocoles, des commentaires sur la gestion des réseaux, la liste n'est pas exhaustive.

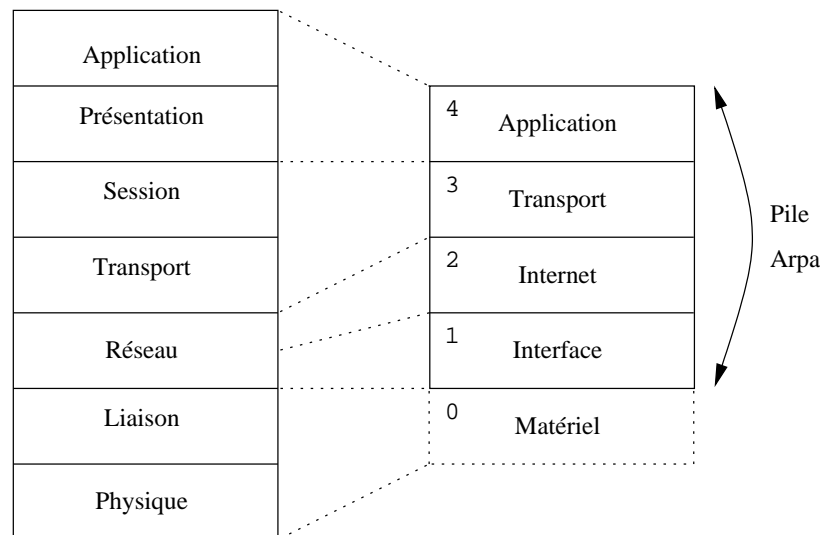
---

<sup>4</sup><http://www.fnet.fr/history/>

### 3 Comparaison TCP/IP — ISO

La suite de protocoles désignée par TCP/IP, ou encore “ pile ARPA ”, est construite sur un modèle en couches moins complet que la proposition de l’ISO. Quatre couches sont suffisantes pour définir l’architecture de ce protocole.

- 4 Couche Application (Application layer).
- 3 Couche Transport (Transport layer).
- 2 Couche Internet (Internet layer).
- 1 Couche interface réseau (Network access layer).
- 0 Matériel (n’est pas une couche comprise dans le protocole).



*figure II.01 — Comparaison ISO-ARPA*

La *figure II.01* met en comparaison les fonctionnalités des couches du modèle OSI et celles des protocoles TCP/IP.

La *figure II.02* elle, donne une vue d’ensemble de l’architecture logicielle avec quelques protocoles d’applications de la famille IP. Ils sont très nombreux, non représentés tous ici, et il s’en faut de beaucoup car il en existe des centaines. La lecture du fichier `/etc/services`, présent sur toute machine de la famille des Unix, donne un aperçu des principaux services enregistrés auprès de l’IANA. Quand nous aurons expliqué la notion “ port ” (cf page 81) cette lecture sera plus facile... Donc patience !

**IP** “ Internet Protocol ”

**SCTP** “ Stream Control Transmission Protocol ”

**TCP** “ Transmission Control Protocol ”

**UDP** “ User Datagram Protocol ”

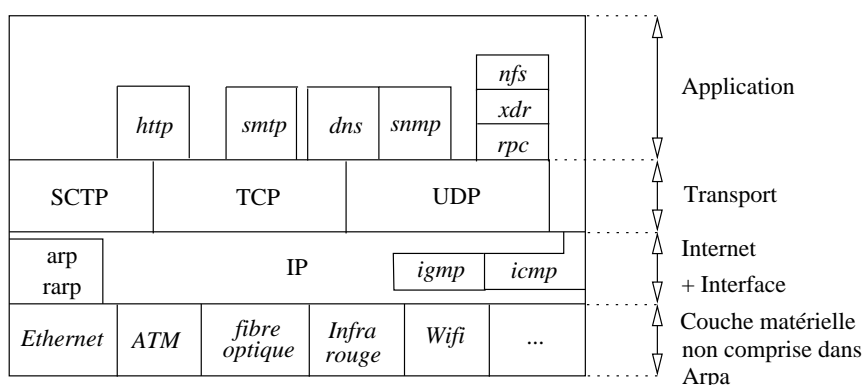


figure II.02 — Architecture logicielle

Les chapitres qui suivent donnent l'occasion d'examiner SMTP, DNS, SNMP, SCTP, TCP, UDP, ARP, RARP, IP, IGMP et ICMP !

### 3.1 Couche “ Application Layer ”

Au plus haut niveau les utilisateurs invoquent les programmes qui permettent l'accès au réseau.

Chaque programme d'application interagit avec la couche de transport pour envoyer ou recevoir des données. En fonction des caractéristiques de l'échange le programme a choisi un mode de transmission à la couche de transport.

La plus grande proportion des applications laissent à la couche de transport le soin d'effectuer le travail de “ Session ”, néanmoins il est possible pour certaines applications de court-circuiter cette fonctionnalité pour agir directement au niveau “ Réseau ”, comme on peut l'observer sur la *figure II.02* à droite.

### 3.2 Couche “ Transport Layer ”

La principale tâche de la couche de transport est de fournir la communication d'un programme d'application à un autre. Une telle communication est souvent qualifiée de “ point à point ”.

Cette couche peut avoir à réguler le flot de données et à assurer la fiabilité du transfert : les octets reçus doivent être identiques aux octets envoyés. C'est pourquoi cette couche doit gérer des “ checksums ” et savoir re-émettre des paquets mal arrivés.

Cette couche divise le flux de données en paquets (terminologie de l'ISO) et passe chacun avec une adresse de destination au niveau inférieur.

De plus, et c'est surtout valable pour les systèmes d'exploitation multi-tâches multi-utilisateurs (Unix,...), de multiples processus appartenant à des utilisateurs différents et pour des programmes d'applications différents, accèdent au réseau au même moment, ce qui implique la capacité de multiplexer et de démultiplexer les données, suivant qu'elles vont vers le réseaux

ou vers les applications ( “ Session ” ).

### 3.3 Couche “ Internet Layer ”

Cette couche reçoit des data-grammes en provenance de la couche réseau, qu'elle doit analyser pour déterminer s'ils lui sont adressés ou pas. Dans le premier cas elle doit “ décapsuler ” son en-tête du data-gramme pour transmettre les données à la couche de transport et au bon protocole de cette couche (TCP, UDP...), dans le deuxième cas elle les ignore.

Cette couche prend aussi en charge la communication de machine à machine. Elle accepte des requêtes venant de la couche de transport avec une identification de la machine vers laquelle le paquet doit être envoyé.

Elle utilise alors l'algorithme de routage (page 70) pour décider si le paquet doit être envoyé vers une passerelle ou vers une machine directement accessible.

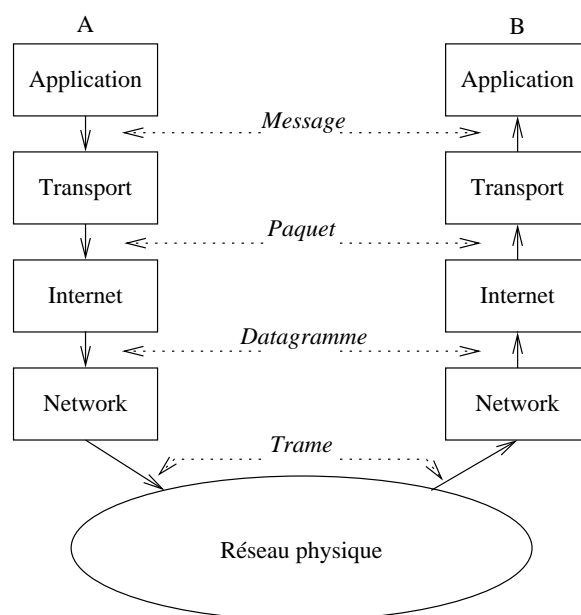
Enfin cette couche gère les datagrammes des protocoles ICMP et IGMP.

### 3.4 Couche “ Network Access ”

Le protocole dans cette couche définit le moyen pour un système de délivrer l'information à un autre système physiquement relié. Il définit comment les data-grammes IP sont transmis. La définition de ceux-ci reste indépendante de la couche réseau, ce qui leur permet de s'adapter à chaque nouvelle technologie au fur et à mesure de leur apparition.

Avant de s'intéresser au détail des data-grammes IP, nous allons examiner le problème de l'adressage IP, dans le chapitre suivant.

## 4 Encapsulation d'IP



*figure II.03 — Encapsulation d'IP*

Comme nous l'avons décrit avec le modèle des couches OSI, les couches IP fonctionnent par encapsulations progressives.

Chaque couche en-capsule la précédente avec les informations de contrôle qu'elle destine à la couche de même niveau sur la machine distante.

Cet ajout est nommé “ header ” (en-tête) parce-qu'il est placé en tête des données à transmettre.

|             |        |        |        |       |
|-------------|--------|--------|--------|-------|
| Application |        |        |        | datas |
| Transport   |        |        | Header | datas |
| Internet    |        | Header | Header | datas |
| Network     | Header | Header | Header | datas |

La taille des “ headers ” dépend des protocoles utilisés. Pour la couche IP le protocole comporte en standard 5 mots de 32 bits, même chose pour la couche TCP<sup>5</sup>.

## 5 Bibliographie

Pour en savoir plus :

**RFC 0791** S J. Postel, “ Internet Protocol ”, 09/01/1981. (Pages=45)  
(Format=.txt) (Obsoletes RFC0760)

**La Recherche** Numéro spécial Internet numéro 238 de février 2000

---

<sup>5</sup>5 mots de 32 bits =  $5 \times 4 \text{ octets} = 20 \text{ octets}$



# Chapitre III

## Anatomie d'une adresse IP

### 1 Adressage IP

Nous avons dit que l'Internet est un réseau virtuel, construit par interconnexion de réseaux physiques via des passerelles. Ce chapitre parle de l'adressage, le maillon essentiel des protocoles TCP/IP pour rendre transparents les détails physiques des réseaux et faire apparaître l'Internet comme une entité homogène.

#### 1.1 Unicité de l'adresse

Un système de communication doit pouvoir permettre à n'importe quel hôte de se mettre en relation avec n'importe quel autre. Afin qu'il n'y ait pas d'ambiguïté pour la reconnaissance des hôtes possibles, il est absolument nécessaire d'admettre un principe général d'identification.

Lorsque l'on veut établir une communication, il est intuitivement indispensable de posséder trois informations :

1. Le nom de la machine distante,
2. Son adresse,
3. La route à suivre pour y parvenir.

Le nom dit “ qui ” est l'hôte distant, l'adresse nous dit “ où ” il se trouve et la route “ comment ” on y parvient.

En règle générale les utilisateurs préfèrent des noms symboliques pour identifier les machines tandis que les processeurs de ces mêmes machines ne comprennent que les nombres exprimés au format binaire.

Les adresses IP (version 4) sont standardisées sous forme d'un nombre de 32 bits qui permet à la fois l'identification de chaque hôte et du réseau auquel il appartient. Le choix des nombres composants une adresse IP n'est pas laissé au hasard, au contraire il fait l'objet d'une attention particulière notamment pour faciliter les opérations de routage.

Nous éludons la correspondance entre ce nombre et une éventuelle représentation symbolique, c'est l'objet du serveur de noms, une application examinée page 165.

Chaque adresse IP contient donc deux informations élémentaires, une adresse de réseau et une adresse d'hôte. La combinaison des deux désigne de manière unique une machine et une seule sur l'Internet, sous réserve que cette adresse ait été attribuée par un organisme ayant pouvoir de le faire !

## 1.2 Délivrance des adresses IPv4

On distingue deux types d'adresses IP :

**Les adresses privées** que tout administrateur de réseau peut s'attribuer librement pourvu qu'il(elle) ne cherche pas à les router sur l'Internet

**les adresses publiques** délivrées par une structure mondiale qui en assure l'unicité. Ce dernier point est capital pour assurer l'efficacité du routage, comme nous le comprendrons en détaillant le fonctionnement d'IP, à partir de la page 47.

Les adresses à utiliser sur les réseaux privés sont décrites par la RFC 1918 :

|                       |                 |
|-----------------------|-----------------|
| 10.0.0.0 <sup>1</sup> | 10.255.255.255  |
| 172.16.0.0            | 172.31.255.255  |
| 192.168.0.0           | 192.168.255.255 |

Les adresses publiques (souvent une seule), sont le plus généralement fournies par le FAI<sup>2</sup>. Qu'elles soient délivrées de manière temporaire ou attribuées pour le long terme, elles doivent être uniques sur le réseau. La question est donc de savoir de qui le FAI les obtient.

C'est L'ICANN ou " Internet Corporation for Assigned Names and Numbers<sup>3</sup> " qui est chargé au niveau mondial de la gestion de l'espace d'adressage IP. Il définit les procédures d'attribution et de résolution de conflits dans l'attribution des adresses, mais délègue le détail de la gestion de ces ressources à des instances régionales puis locales, dans chaque pays, appelées RIR ou " Regional Internet Registries ".

Il y a actuellement cinq " Regional Internet Registries " opérationnels : l'APNIC<sup>4</sup> pour la région Asie-Pacifique, l'ARIN<sup>5</sup> pour l'Amérique, le RIPE NCC<sup>6</sup> pour l'Europe, l'AfriNIC<sup>7</sup> pour l'Afrique enfin LACNIC<sup>8</sup> pour l'Amérique Latine.

Pour ce qui nous concerne en Europe c'est donc le RIPE NCC (Réseaux IP européen Network Coordination Centre) qui délivre les adresses que nous utilisons.

Les adresses IP sont allouées à l'utilisateur final qui en fait la demande par un " Local Internet Registry ", ou LIR, autorisé par le RIPE NCC.

<sup>2</sup>Fournisseur d'Accès Internet

<sup>3</sup><http://www.icann.org>

<sup>4</sup><http://www.apnic.net>

<sup>5</sup><http://www.arin.net/>

<sup>6</sup><http://www.ripe.net/>

<sup>7</sup><http://www.afrinic.net/>

<sup>8</sup><http://lacnic.net/>



Un LIR est généralement un FAI ou une grande organisation (entreprise multinationale). Il est sous l'autorité de l'instance régionale de gestion de l'adressage. Ainsi pour un utilisateur (quelle que soit sa taille) changer de FAI implique aussi de changer de plan d'adressage IP, lorsque celles-ci ont été allouées statiquement par le LIR. Les adresses IP sont alors restituées puis ré-attribuées à d'autres utilisateurs.

On compte plus de 2000 de LIRs offrant leurs services en Europe selon le RIPE NCC<sup>9</sup>. Le chiffre a forcément augmenté depuis 2003, avec l'élargissement des frontières européennes.

## 2 Anatomie d'une adresse IP

Une adresse IP est un nombre de 32 bits que l'on a coutume de représenter sous forme de quatre entiers de huit bits, séparés par des points (paragraphe 1.2).

La partie réseau de l'adresse IP vient toujours en tête, la partie hôte est donc toujours en queue.

L'intérêt de cette représentation est immédiat quand on sait que la partie réseau et donc la partie hôte sont presque toujours codées sur un nombre entier d'octets. Ainsi, on a principalement les trois formes suivantes :

**Classe A** Un octet réseau, trois octets d'hôtes.

**Classe B** Deux octets réseau, deux octets d'hôtes.

**Classe C** Trois octets réseau, un octet d'hôte.

### 2.1 Décomposition en classes

| Classe | Nombre de réseaux/machines  |
|--------|---|
| A      | 1.x.y.z à 127.x.y.z<br>127 réseaux<br>16 777 216 machines ( $2^{24}$ )                |
| B      | 128.0.x.y à 191.255.x.y<br>16 384 réseaux ( $2^{14}$ )<br>65536 machines ( $2^{16}$ ) |
| C      | 192.0.0.z à 223.255.255.z<br>2 097 152 réseaux ( $2^{21}$ )<br>256 machines ( $2^8$ ) |
| D      | 224.0.0.0 à 239.255.255.255   |
| E      | 240.0.0.0 à 247.255.255.255   |

figure III.01 —  
Décomposition en classes

Pour distinguer les classes A, B, C, D et E il faut examiner les bits de poids fort de l'octet de poids fort :

**Si le premier bit est 0**, l'adresse est de classe A. On dispose de 7 bits pour identifier le réseau et de 24 bits pour identifier l'hôte. On a donc les réseaux de 1 à 127 et  $2^{24}$  hôtes possibles, c'est à dire 16 777 216 machines différentes (de 0 à 16 777 215).

Les lecteurs attentifs auront remarqué que le réseau 0 n'est pas utilisé, il a une signification particulière (" tous les réseaux "). Plus de détails au paragraphe 2.2.

<sup>9</sup> <http://www.ripe.net/ripe/docs/ar2003.html>

De même, la machine 0 n'est pas utilisée, tout comme la machine ayant le plus fort numéro dans le réseau (tous les bits de la partie hôte à 1, ici 16 777 215), ce qui réduit de deux unités le nombre des machines nommables. Il reste donc seulement 16 777 214 machines adressables dans une classe A !

**Si les deux premiers bits sont 10** , l'adresse est de classe B. Il reste 14 bits pour identifier le réseau et 16 bits pour identifier la machine. Ce qui fait  $2^{14} = 16\,384$  réseaux (128.0 à 191.255) et 65 534 (65 536 – 2) machines.

**Si les trois premiers bits sont 110** , l'adresse est de classe C. Il reste 21 bits pour identifier le réseau et 8 bits pour identifier la machine. Ce qui fait  $2^{21} = 2\,097\,152$  réseaux (de 192.0.0 à 223.255.255) et 254 (256 – 2) machines.

**Si les quatre premiers bits de l'adresse sont 1110** , il s'agit d'une classe d'adressage spéciale, la classe D. Cette classe est prévue pour faire du “ multicast ”, ou multipoint. (RFC 1112 [S. Deering, 1989]), contrairement aux trois premières classes qui sont dédiées à l'“ unicast ” ou point à point.

Ces adresses forment une catégorie à part, nous en reparlons au paragraphe 3.

**Si les quatre premiers bits de l'adresse sont 1111** , il s'agit d'une classe expérimentale, la classe E. La RFC 1700 précise “ Class E addresses are reserved for future use ” mais n'indique pas de quel futur il s'agit...

Enfin, pour conclure ce paragraphe, calculons le nombre d'hôtes adressables théoriquement à l'aide des classes A, B et C :

$$127 \times 16777212 + 16384 \times 65534 + 2097152 \times 254 = 3737091588^{10}$$

Ce total, pour être plus exact, doit être amputé des 17 890 780 hôtes des réseaux privés prévus dans la RFC 1918<sup>11</sup>, soit donc tout de même au total **3 719 200 808** hôtes adressables en utilisant IPv4 !

<sup>10</sup>Sous shell, tapez : `echo "127*16777212+16384*65534+2097152*254"|bc`

<sup>11</sup>  $16777212 + 16 \times 65534 + 256 \times 254 = 17890780$

## 2.2 Adresses particulières

Certaines adresses IP ont une signification particulière !

Par convention le numéro 0 d'hôte n'est pas attribué. Si une adresse IP contient cette zone nulle cela signifie que l'on adresse le réseau lui-même et aucun hôte en particulier, donc en règle générale l'hôte lui-même.

De même, pour toutes les pile Arpa l'adresse 127.0.0.1 indique la machine elle-même ( " localhost " – Voir chapitre IP page 47), indépendamment des autres adresses réseaux éventuellement attribuées à n'importe lequel de ses interfaces.

À l'inverse, si tous les bits de la partie hôte sont à 1, cela désigne toutes les machines du réseaux, c'est ce que l'on appelle une adresse de " broadcast ", c'est à dire une information adressée à tout le monde.

On évite au maximum l'usage d'une telle adresse IP sur les réseaux, pour des raisons d'efficacité (encombrement de la bande passante).

Quelques exemples d'adresses avec une signification particulière :

|                 |                                   |
|-----------------|-----------------------------------|
| 0.0.0.0         | Hôte inconnu, sur ce réseau       |
| 0.0.0.1         | L'hôte 1 de ce réseau             |
| 255.255.255.255 | Tous les hôtes                    |
| 138.195.52.1    | L'hôte 52.1 du réseau 138.195.0.0 |
| 138.195.0.0     | Cet hôte sur le 138.195.0.0       |
| 193.104.1.255   | Tous les hôtes du 193.104.1.0     |
| 127.0.0.1       | Cet hôte (boucle locale).         |

Remarque : les deux premières adresses, avec un numéro de réseau égal à 0, ne peuvent figurer que comme adresse source dans des cas bien particuliers comme le démarrage d'une station (cf chapitre IP page 47 et les travaux pratiques associés).

2.3 Sous-réseaux

En 1984 un troisième niveau de hiérarchie est mis en place : le “ subnet ” ou sous-réseau. pour permettre aux administrateurs de gérer plus finement de grands réseaux. La RFC 950 [J. Mogul, J. Postel, 1985] donne plus de précisions, la RFC 1878 [T. Pummill & B. Manning, 1995] est une table de tous les sous-réseaux possibles.

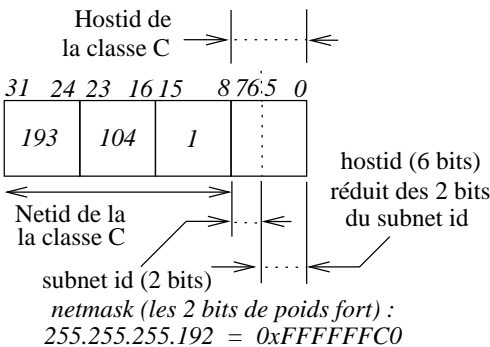


figure III.02 — Sous-réseaux

Le “ subnet ” utilise les bits de poids fort de la partie hôte de l’adresse IP, pour désigner un réseau. Le nombre de bits employés est laissé à l’initiative de l’administrateur.

Dans la figure III.02 ci-contre, les bits 6 et 7 de la partie “ host ” sont utilisés pour caractériser un sous-réseau.

Quelques révisions des propriétés des puissances de 2<sup>12</sup> sont souvent nécessaires pour bien assimiler ce paragraphe. La figure suivante en rappelle les valeurs pour les huit premiers exposants :

figure III.03 — Puissances de 2

|     |    |    |    |   |   |   |   |
|-----|----|----|----|---|---|---|---|
| 7   | 6  | 5  | 4  | 3 | 2 | 1 | 0 |
| 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |

Décomposition unique :  
255 = 128 + 64 + 32 + 16 + 8 + 4 + 2 + 1

Nous avons d’une part 2<sup>7</sup> + 2<sup>6</sup> = 192, et d’autre part 2<sup>5</sup> + 2<sup>4</sup> + 2<sup>3</sup> + 2<sup>2</sup> + 2<sup>1</sup> + 2<sup>0</sup> = 63<sup>13</sup>. Ce qui permet de caractériser 4 sous-réseaux de 62 machines (63 moins l’adresse de broascast, le “ 0 ” n’étant pas compté). Le calcul des masques et des adresses de diffusion est expliqué dans le tableau suivant :

| Numéro du réseau | “ Netmask ”     | “ Broadcast ”  | Adressage hôte |
|------------------|-----------------|----------------|----------------|
| 193.104.1.00     | 255.255.255.192 | 00 + 63 = 63   | .1 à .62       |
| 193.104.1.64     | 255.255.255.192 | 64 + 63 = 127  | .65 à .126     |
| 193.104.1.128    | 255.255.255.192 | 128 + 63 = 191 | .129 à .190    |
| 193.104.1.192    | 255.255.255.192 | 192 + 63 = 255 | .193 à .254    |

<sup>12</sup>et plus généralement de la décomposition d’un nombre en ses facteurs premiers. . .

<sup>13</sup>Donc 64 valeurs possibles de 0 à 63

Soit un total de  $62 \times 4 = 248$  hôtes possibles pour cette classe C avec un masque de sous-réseau<sup>14</sup>, au lieu des 254 hôtes sans.

La machine d'adresse 1 sur chaque sous-réseau, aura comme adresse IP :

| Sous-réseau | Adresse       | Décomposition   |
|-------------|---------------|-----------------|
| 00          | 193.104.1.1   | $00 + 1 = 1$    |
| 01          | 193.104.1.65  | $64 + 1 = 65$   |
| 10          | 193.104.1.129 | $128 + 1 = 129$ |
| 11          | 193.104.1.193 | $192 + 1 = 193$ |

Si vous pensez avoir tout compris, le remplissage du tableau suivant dans le cas de la classe C 192.168.192.0 et avec 3 bits pour définir les sous-réseaux ne devrait pas vous poser de problème...

| Numéro du subnet | Numéro du réseau | Adresse de broadcast | Première machine | Dernière machine |
|------------------|------------------|----------------------|------------------|------------------|
| 000(0)           | 192.168.192.     | 192.168.192.         |                  |                  |
| 001(1)           | 192.168.192.     | 192.168.192.         |                  |                  |
| 010(2)           | 192.168.192.     | 192.168.192.         |                  |                  |
| 011(3)           | 192.168.192.     | 192.168.192.         |                  |                  |
| 100(4)           | 192.168.192.     | 192.168.192.         |                  |                  |
| 101(5)           | 192.168.192.     | 192.168.192.         |                  |                  |
| 110(6)           | 192.168.192.     | 192.168.192.         |                  |                  |
| 111(7)           | 192.168.192.     | 192.168.192.         |                  |                  |

À toutes ces adresses il faudra appliquer le masque de sous-réseau :

0xFFFFF\_\_ soit encore 255.255.255.---

**Remarque :** On pourra vérifier que la perte d'espace d'adressage pour adresser des hôtes se calcule avec la relation  $(2^n - 1) \times 2$ , où n est le nombre de bits du masque. Ainsi avec 3 bits de masque de sous-réseau, la perte d'espace d'adressage s'élève à 14 hôtes! Les 254 possibilités (256 moins 0 et 255) de numérotation de la classe C se réduisent à 240, amputées de 31, 32, 63, 64, 95, 96, 127, 128, 159, 160, 191, 192, 223 et 224.

<sup>14</sup> " netmask "

## 2.4 CIDR

En 1992 la moitié des classes B étaient allouées, et si le rythme avait continué, au début de 1994 il n'y aurait plus eu de classe B disponible et l'Internet aurait bien pu mourir par asphyxie ! De plus la croissance du nombre de réseaux se traduisait par un usage " aux limites " des routeurs, proches de la saturation car non prévus au départ pour un tel volume de routes (voir les RFC 1518 et RFC 1519).

Deux considérations qui ont conduit l'IETF à mettre en place le " Classless InterDomain Routing " ou CIDR ou encore routage Internet sans classe, basé sur une constatation de simple bon sens :

- S'il est courant de rencontrer une organisation ayant plus de 254 hôtes, il est moins courant d'en rencontrer une de plus de quelques milliers. Les adresses allouées sont donc des classes C contigües, attribuées par région ou par continent. En générale, 8 à 16 classes C mises bout à bout suffisent pour une entreprise. Ces blocs de numéros sont souvent appelés " supernet ". Ainsi par exemple il est courant d'entendre les administrateurs de réseaux parler d'un " slash 22 " (/22) pour désigner un bloc de quatre classes C consécutives. . .
- Il est plus facile de prévoir une table de routage pour un bloc de réseaux contigües que d'avoir à le faire pour une multitude de routes individuelles. En plus cette opération allège la longueur des tables.

Plus précisément, trois caractéristiques sont requises pour pouvoir utiliser ce concept :

1. Pour être réunies dans une même route, des adresses IP multiples doivent avoir les mêmes bits de poids fort (seuls les bits de poids plus faible diffèrent) de poids faibles différent.
2. Les tables de routages et algorithmes doivent prendre en compte un masque de 32 bits, à appliquer sur les adresses.
3. Les protocoles de routage doivent ajouter un masque 32 bits pour chaque adresse IP (Cet ajout double le volume d'informations) transmise. OSPF, IS-IS, RIP-2, BGP-4 le font.

Ce masque se manifeste concrètement comme dans la réécriture du tableau du paragraphe 1.2 :

|             |                 |            |
|-------------|-----------------|------------|
| 10.0.0.0    | 10.255.255.255  | 10/8       |
| 172.16.0.0  | 172.31.255.255  | 172.16/12  |
| 192.168.0.0 | 192.168.255.255 | 192.168/16 |

Le terme " classless " vient de ce fait, le routage n'est plus basé uniquement sur la partie réseau des adresses.

Les agrégations d'adresses sont ventilées selon le tableau suivant<sup>15</sup> :

|                                       |           |                 |
|---------------------------------------|-----------|-----------------|
| Multirégionales                       | 192.0.0.0 | 193.255.255.255 |
| Europe                                | 194.0.0.0 | 195.255.255.255 |
| Autres                                | 196.0.0.0 | 197.255.255.255 |
| Amérique du Nord                      | 198.0.0.0 | 199.255.255.255 |
| Amérique centrale,<br>Amérique du Sud | 200.0.0.0 | 201.255.255.255 |
| Zone Pacifique                        | 202.0.0.0 | 203.255.255.255 |
| Autres                                | 204.0.0.0 | 205.255.255.255 |
| Autres                                | 206.0.0.0 | 207.255.255.255 |

## 2.5 Précisions sur le broadcast

Tout d'abord il faut préciser qu'une adresse de broadcast est forcément une adresse de destination, elle ne peut jamais apparaître comme une adresse source dans un usage normal des réseaux.

Quatre formes possibles de broadcast :

“ **Limited broadcast** ” (255.255.255.255) Une telle adresse ne peut servir que sur le brin local et ne devrait jamais franchir un routeur. Ce n'est malheureusement pas le cas (précisions en cours).

L'usage de cette adresse est normalement limitée à un hôte en phase d'initialisation, quand il ne connaît rien du réseau sur lequel il est connecté.

“ **Net-directed broadcast** ” Tous les bits de la partie hôte sont à 1. Un routeur propage ce type de broadcast, sur option.

“ **Subnet-directed broadcast** ” C'est le même cas que ci-dessus mais avec une adresse IP comportant des subnets.

“ **All-subnets-directed broadcast** ” C'est le cas où tous les bits des subnets et hôtes sont à 1. Ce cas possible théoriquement est rendu obsolète depuis la RFC 922 (1993).

---

<sup>15</sup>Ce tableau est très synthétique, pour une information plus détaillée et à jour consultez le site de l'IANA <http://www.iana.org/assignments/ipv4-address-space>

### 3 Adressage multicast

En règle générale l'adressage multicast est employé pour s'adresser en une seule fois à un groupe de machines.

Dans le cas d'un serveur vidéo/audio, cette approche induit une économie de moyen et de bande passante évidente quand on la compare à une démarche "unicast" : un seul datagramme est routé vers tous les clients intéressés au lieu d'un envoi massif d'autant de datagrammes qu'il y a de clients.

Les adresses de type "multicast" ont donc la faculté d'identifier un groupe de machines qui partagent un protocole commun par opposition à un groupe de machines qui partagent un réseau commun.

La plupart des adresses multicast allouées le sont pour des applications particulières comme par exemple la découverte de routeurs (que nous verrons ultérieurement lors du routage IP) ou encore la radio ou le téléphone/vidéo sur Internet ("Mbone"). Parmi les plus souvent utilisées<sup>16</sup> sur un lan :

|            |  |
|------------|--|
| 224.0.0.1  | Toutes les machines sur ce sous-réseau |
| 224.0.0.2  | Tous les routeurs sur ce sous-réseau   |
| 224.0.0.5  | Tous les routeurs OSPF (page 121)      |
| 224.0.0.9  | Tous les routeurs RIPv2 (page 113)     |
| 224.0.0.22 | Protocole IGMP (page 63)               |

#### 3.1 Adresse de groupe multicast

Si une adresse multicast démarre avec les bits 1110 par contre pour les 28 bits suivants son organisation interne diffère de celle des classes A, B et C.

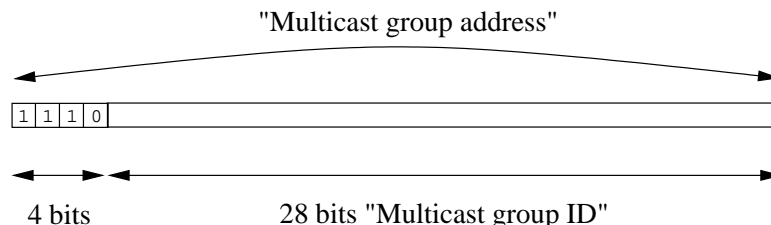


figure III.04 — Adresses de multicast

- Les 28 bits n'ont pas de structure particulière par contre on continue à utiliser la notation décimale pointée : 224.0.0.0 à 239.255.255.255.
- Un groupe d'hôtes qui partagent un protocole commun utilisant une adresse multicast commune peuvent être répartis n'importe où sur le réseau.
- L'appartenance à un groupe est dynamique, les hôtes qui le désirent rejoignent et quittent le groupe comme ils veulent.
- Il n'y a pas de restriction sur le nombre d'hôtes dans un groupe et un hôte n'a pas besoin d'appartenir à un groupe pour lui envoyer un message.

<sup>16</sup>Pour plus de précisions on pourra se reporter page 56 de la RFC 1700



### 3.2 Adresse multicast et adresse MAC

Une station Ethernet quelconque doit être configurée pour accepter le multicast, c'est à dire pour accepter les trames contenant un datagramme munis d'une adresse IP de destination qui est une adresse multicast.

Cette opération sous entend que la carte réseau sait faire le tri entre les trames. En effet les trames multicast ont une adresse MAC particulière : elles commencent forcément par les trois octets 01:00:5E<sup>17</sup>. Ceux-ci ne désignent pas un constructeur en particulier mais sont possédés par l'ICANN (ex IANA).

Restent trois octets dont le bit de poids fort est forcément à 0 pour désigner les adresses de multicast (contrainte de la RFC 1700), ce qui conduit au schéma suivant :

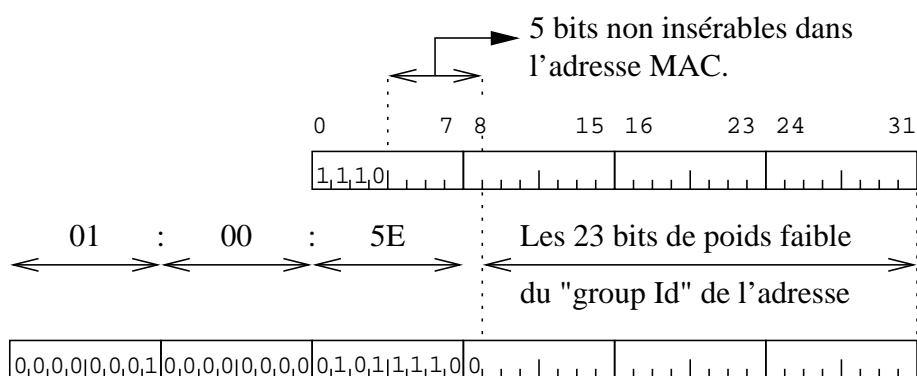


figure III.05 — Adresse physique de multicast

Du fait qu'il n'y a pas assez de place dans l'adresse MAC pour faire tenir les 28 bits du groupe multicast, cette adresse n'est pas unique. On peut même préciser que pour chaque trame comportant une adresse multicast il y a  $2^5$  adresses IP de groupes multicast possibles !

Ce qui signifie que si les 23 bits de poids faible ne suffisent pas à discriminer la trame il faudra faire appel au pilote de périphérique ou à la couche IP pour lever l'ambiguïté.

Quand une trame de type multicast est lue par la station Ethernet puis par le pilote de périphérique, si l'adresse correspond à l'une des adresses de groupe multicast préalablement configurées, le datagramme franchit la couche IP et une copie des données est délivrée aux processus qui ont "joint le groupe multicast".

La question est de savoir comment les trames de type multicast atteignent justement cette station Ethernet ? La réponse se trouve dans un protocole nommé IGMP et que nous examinerons dans le prochain chapitre concernant IP, page 63.

<sup>17</sup>Cf RFC 1700 page 171

## 4 Conclusion et bibliographie

Pour conclure ce chapitre sur l'adressage IP, il faut nous donner quelques précisions supplémentaires.

Jusqu'à présent nous avons désigné un hôte par son adresse IP. Cette démarche n'est pas exacte si on considère par exemple le cas d'une passerelle, connectée physiquement à au moins deux réseaux différents, avec une adresse IP dans chacun de ces réseaux.

On dira donc maintenant qu'une adresse IP identifie non pas un hôte mais un interface. La réciproque n'est pas vraie car un même interface peut collectionner plusieurs adresses IP. Toutes permettent d'atteindre cet interface, on parle alors d' "alias IP", d' "hôtes virtuels" et de "réseaux virtuels"... Nous aurons l'occasion de revenir sur ces notions à la fin de ce cours (page 137).

On dit d'une machine ayant au moins deux adresses IP qu'elle est du type "multi-homed".

En général une passerelle qui met en relation  $n$  réseaux possède  $n$  adresses IP différentes (une dans chaque réseau), mais ce n'est pas une obligation (nous verrons quelle peut en être l'utilité à la fin de ce cours).

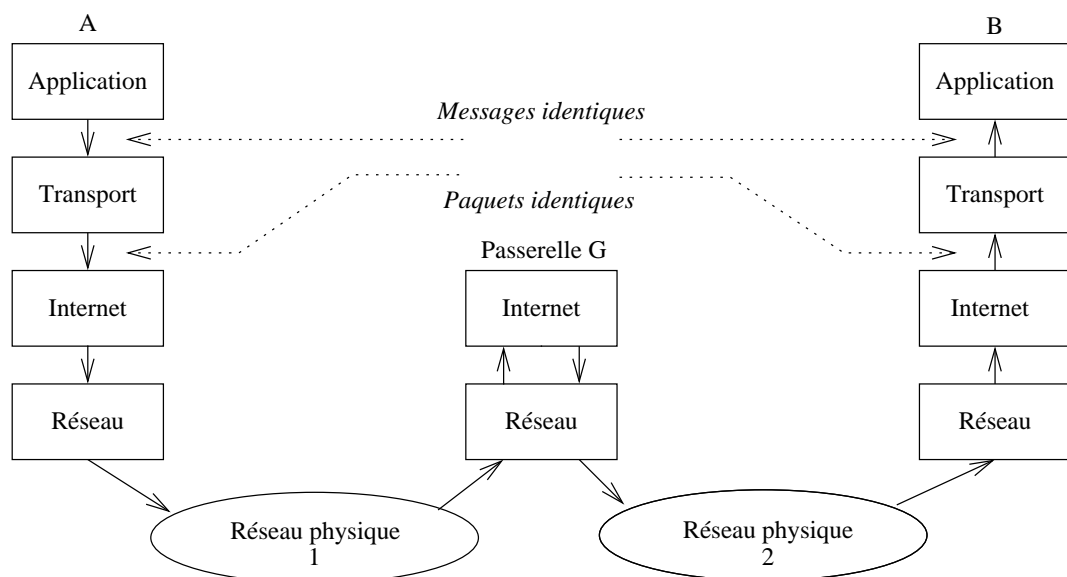


figure III.06 — Usage combiné des adresses logique et physique

La figure III.06 met en situation deux hôtes, A et B, en relation via une passerelle G. Si les "messages" et les "paquets" sont identiques, par contre les "datagrammes" et les "trames" diffèrent puisqu'il ne s'agit plus du même réseau physique. Dès que nous aurons examiné le fonctionnement de la couche IP nous reviendrons sur cette figure pour en expliquer le fonctionnement (voir page 76).

Pour en savoir plus :

- RFC 0950** S J. Mogul, J. Postel, “ Internet standard subnetting procedure ”,  
08/01/1985. (Pages=18) (Format=.txt) (STD 5)
- RFC 1112** S S. Deering, “ Host extensions for IP multicasting ”,  
08/01/1989.  
(Pages=17) (Format=.txt) (Obsoletes RFC0988) (STD 5)
- RFC 1518** “ An Architecture for IP Address Allocation with CIDR ”  
Y. Rekhter, T. Li. September 1993. (Format : TXT=72609 bytes) (Status : PROPOSED STANDARD)
- RFC 1519** PS V. Fuller, T. Li, J. Yu, K. Varadhan, “ Classless Inter-Domain  
Routing (CIDR) : an Address Assignment and Aggregation Strategy ”,  
09/24/1993. (Pages=24) (Format=.txt) (Obsoletes RFC1338)
- RFC 1466** I E. Gerich, “ Guidelines for Management of IP Address Space ”,  
05/26/1993. (Pages=10) (Format=.txt) (Obsoletes RFC1366)
- RFC 1467** “ Status of CIDR Deployment in the Internet. ”  
C. Topolcic. August 1993. (Format : TXT=20720 bytes)  
(Obsoletes RFC1367) (Status : INFORMATIONAL)
- RFC 1700** “ Assigned Numbers. ” J. Reynolds, J. Postel. October 1994.  
(Format : TXT=458860 bytes) (Obsoletes RFC1340)  
(Also STD0002) (Status : STANDARD)
- RFC 1878** “ Variable Length Subnet Table For IPv4. ”  
T. Pummill & B. Manning. December 1995.  
(Format : TXT=19414 bytes) (Obsoletes RFC1860) (Status : INFORMATIONAL)
- RFC 1918** “ Address Allocation for Private Internets. ” Y. Rekhter, B. Moskowitz,  
D. Karrenberg, G. J. de Groot & E. Lear. February 1996.  
(Format : TXT=22270 bytes) (Obsoletes RFC1627, RFC1597) (Also BCP0005)  
(Status : BEST CURRENT PRACTICE)

Quelques ouvrages qui font autorité :

- W. Richard Stevens - TCP/IP Illustrated, Volume 1 - The protocols - Addison-Wesley
- Douglas Comer - Internetworking with TCP/IP - Principles, protocols, and architecture - Prentice-Hall
- Christian Huitema - Le routage dans l'Internet - EYROLLES



# Chapitre IV

## Protocole IP

### 1 Datagramme IP

IP est l'acronyme de “ **I**nternet **P**rotocol ”, il est défini dans la RFC 791 et a été conçu en 1980 pour remplacer NCP (“ Network Control Protocol ”), le protocole de l'Arpanet.

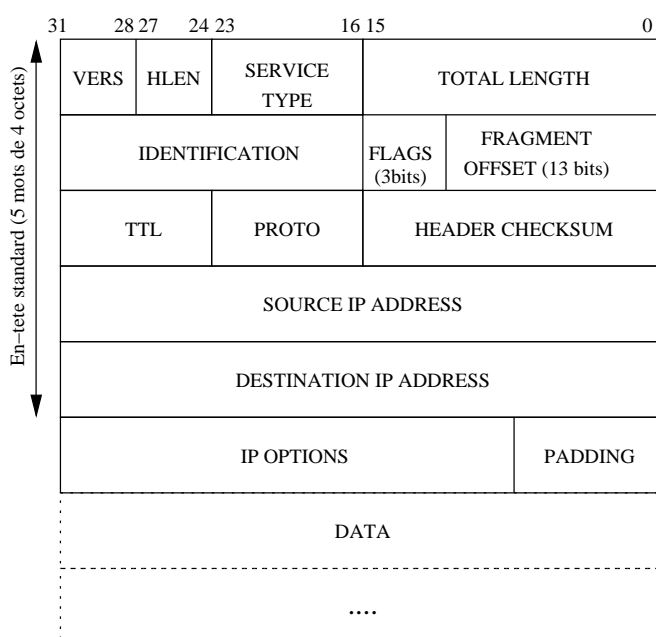
Presque trente ans après sa première implémentation, ses limitations se font de plus en plus pénalisantes pour les nouveaux usages sur les réseaux. Avant de le jeter aux orties, posons-nous la question de qui pouvait prévoir à cette époque où moins de mille ordinateurs étaient reliés ensemble, que trois décennies plus tard des dizaines de millions d'hôtes l'utiliseraient comme principal protocole de communication ?

Sa longévité est donc remarquable et il convient de l'analyser de près avant de pouvoir le critiquer de manière constructive.

#### 1.1 Structure de l'en-tête

Les octets issus de la couche de transport et encapsulés à l'aide d'un en-tête IP avant d'être propagés vers la couche réseau (Ethernet par exemple), sont collectivement nommés “ datagramme IP ”, datagramme Internet ou datagramme tout court. Ces datagrammes ont une taille maximale liée aux caractéristiques de propagation du support physique, c'est le “ Maximum Transfer Unit ” ou MTU.

*figure IV.01 — Structure du datagramme IP*



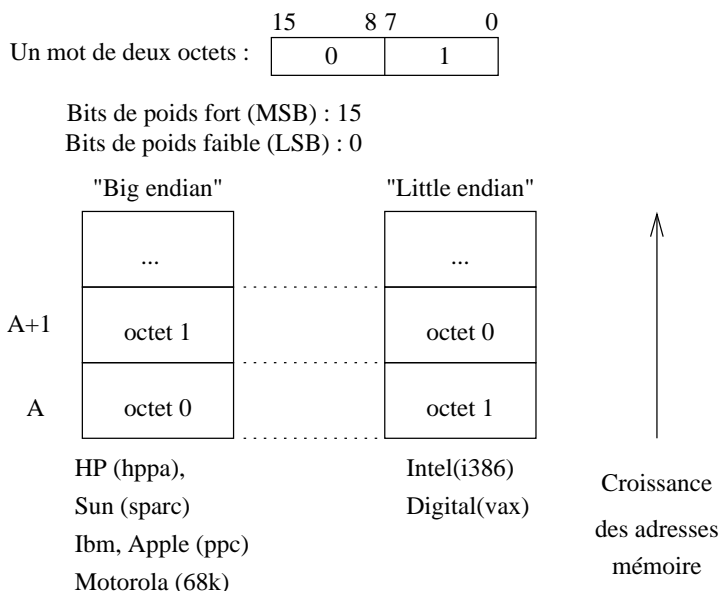
Quelques caractéristiques en vrac du protocole IP :

- IP est le support de travail des protocoles de la couche de transport, UDP, TCP et SCTP.
- IP ne donne aucune garantie quant au bon acheminement des données qu'il envoie. Il n'entretient aucun dialogue avec une autre couche IP distante, on dit aussi qu'il délivre les datagramme " au mieux ".
- Chaque datagramme est géré indépendamment des autres datagrammes même au sein du transfert des octets d'un même fichier. Cela signifie que les datagrammes peuvent être mélangés, dupliqués, perdus ou altérés !  
Ces problèmes ne sont pas détectés par IP et donc il ne peut en informer la couche de transport.
- Les octets sont lus et transmis au réseau en respectant le " Network Byte Order " ou NBO (cf paragraphe 1.2) quelle que soit l'architecture cpu de l'hôte.
- L'en-tête IP minimale fait 5 mots de 4 octets, soit 20 octets. S'il y a des options la taille maximale peut atteindre 60 octets.

## 1.2 Network Byte Order

Sur la *figure IV.01* les bits les plus significatifs de chaque mot de quatre octets sont à gauche (31...). Ils sont d'ailleurs transmis sur le réseau dans cet ordre<sup>1</sup>, c'est un standard, c'est le " Network Byte Order ".

Toutes les architectures de CPU ne sont pas bâties sur le même modèle :



*figure IV.02 — " Big endian " vs " Little endian "*

<sup>1</sup>Le lecteur ayant un accès aux sources d'une pile IP pourra aller consulter directement la structure de l'en-tête, par exemple le fichier `/usr/src/sys/netinet/ip.h` sur une machine FreeBSD

Les termes “ Big endian ” et “ Little endian ” indiquent quelle est la terminaison ( “ end ” ) de deux octets que l'on écrit en premier le poids fort ( “ big ” ), c'est aussi le sens de l'écriture humaine, ou le poids faible ( “ little ” ).

### 1.3 Description de l'en-tête

**VERS** 4 bits qui spécifient la version du protocole IP. L'objet de ce champ est la vérification que l'émetteur et le destinataire des datagrammes sont bien en phases avec la même version. Actuellement c'est la version 4 qui est principalement utilisé sur l'Internet, bien que quelques implémentations de la version 6 existent et soient déjà en expérimentation<sup>2</sup>.

**HLEN** 4bits qui donnent la longueur de l'en-tête en mots de 4 octets. La taille standard de cette en-tête fait 5 mots, la taille maximale fait :  $(2^3 + 2^2 + 2^1 + 2^0) \times 4 = 60$  octets<sup>3</sup>

**TOTAL LENGTH** Donne la taille du datagramme, en-tête plus données. S'il y fragmentation (voir plus loin) il s'agit également de la taille du fragment (chaque datagramme est indépendant des autres).

La taille des données est donc à calculer par soustraction de la taille de l'en-tête.

16 bits autorisent la valeur 65535... La limitation vient le plus souvent du support physique (MTU) qui impose une taille plus petite, sauf sur les liaisons de type “ hyperchannel ”.

**TYPE OF SERVICE vs DSCP/ECN** Historiquement dans la RFC 791 ce champ est nommé **TYPE OF SERVICE** et joue potentiellement deux rôles selon les bits examinés (préséance et type de service). Pratiquement, la préséance ne sert plus et la RFC 1349 définit 4 bits utiles sur les huit (3 à 6). Ceux-ci indiquent au routeur l'attitude à avoir vis à vis du datagramme.

Par exemple, des datagrammes d'un transfert de fichier (**ftp**) peuvent avoir à laisser passer un datagramme repéré comme contenant des caractères frappés au clavier (session telnet).

|      |         |                      |                 |
|------|---------|----------------------|-----------------|
| 0x00 | -       | Service normal       | Transfert banal |
| 0x10 | bit 3,D | Minimiser le délai   | Session telnet  |
| 0x08 | bit 4,T | Maximiser le débit   | Transfert ftp   |
| 0x04 | bit 5,R | Maximiser la qualité | ICMP            |
| 0x02 | bit 6,C | Minimiser le coût    | “ news ” (nntp) |

L'usage de ces bits est mutuellement exclusif.

Les nouveaux besoins de routage ont conduit l'IETF à revoir la définition de ce champ dans la RFC 3168. Celle ci partage les huit bits en deux parties, les premiers bits définissent le DSCP ou “ Differentiated Services CodePoints ” qui est une version beaucoup plus fine des quatre

<sup>2</sup>Nous examinerons les caractéristiques de la version 6 d'IP à la fin de ce cycle de cours

<sup>3</sup>On en fait encore plus simple  $(2^4 - 1) \times 4$

bits ci-dessus. Les deux derniers bits définissent l'ECN ou " Explicit Congestion Notification " qui est un mécanisme permettant de prévenir les congestions, contrairement au mécanisme plus ancien basé sur les messages ICMP de type " source quench " (voir page 61) qui tente de régler le flux en cas de congestion.

Il faut noter que les protocoles de routage qui tiennent compte de l'état des liaisons (OSPF, IS-IS...) sont susceptibles d'utiliser ce champ.

Enfin la RFC 3168 indique que ces deux écritures du champ ne sont pas compatibles entre elles...

**IDENTIFICATION, FLAGS et FRAGMENT OFFSET** Ces mots sont prévus pour contrôler la fragmentation des datagrammes. Les données sont fragmentées car les datagrammes peuvent avoir à traverser des réseaux avec des MTU plus petits que celui du premier support physique employé. Consulter la section suivante *Fragmentation IP*.

**TTL** " Time To Live " 8 bits, 255 secondes maximum de temps de vie pour un datagramme sur le net.

Prévu à l'origine pour décompter un temps, ce champ n'est qu'un compteur décrémenté d'une unité à chaque passage dans un routeur.

Couramment la valeur de départ est 32 ou même 64. Son objet est d'éviter la présence de paquets fantômes circulant indéfiniment...

Si un routeur passe le compteur à zéro avant délivrance du datagramme, un message d'erreur — ICMP (consultez le paragraphe 4) — est renvoyé à l'émetteur avec l'indication du routeur. Le paquet en lui-même est perdu.

**PROTOCOL** 8 bits pour identifier le format et le contenu des données, un peu comme le champ " type " d'une trame Ethernet. Il permet à IP d'adresser les données extraites à l'une ou l'autre des couches de transport.

Dans le cadre de ce cours, nous utiliserons essentiellement ICMP(1), IGMP(2), IP-ENCAP(4), TCP(6), UDP(17), ESP(50), AH(51), et OSPF(89).

La table de correspondance entre le symbole et le numéro du protocole est présente sur tout système d'exploitation digne de ce nom, dans le fichier */etc/protocols*.

**HEADER CHECKSUM** 16 bits pour s'assurer de l'intégrité de l'en-tête. Lors du calcul de ce " checksum " ce champ est à 0.

A la réception de chaque paquet, la couche calcule cette valeur, si elle ne correspond pas à celle trouvée dans l'en-tête le datagramme est oublié (" discard ") sans message d'erreur.

**SOURCE ADDRESS** Adresse IP de l'émetteur, à l'origine du datagramme.

**DESTINATION ADDRESS** Adresse IP du destinataire du datagramme.

**IP OPTIONS** 24 bits pour préciser des options de comportement des couches IP traversées et destinataires. Les options les plus courantes concernent :



- Des problèmes de sécurité
- Des enregistrements de routes
- Des enregistrements d'heure
- Des spécifications de route à suivre
- ...

Historiquement ces options ont été prévues dès le début mais leur implémentation n'a pas été terminée et la plupart des routeurs filtrants bloquent les datagrammes IP comportant des options.

**PADDING** Remplissage pour aligner sur 32 bits...

En conclusion partielle que peut-on dire du travail de la couche IP ?

1. Il consiste à router les datagrammes en les acheminant “ au mieux ”, on verra plus loin de quelle manière. C'est son travail principal.
2. Il peut avoir à fragmenter les données de taille supérieure au MTU du support physique à employer.

## 1.4 Fragmentation IP - MTU

La couche de liaison (Couche 2 “ Link ”) impose une taille limite, le “ Maximum Transfer Unit ”. Par exemple cette valeur est de 1500 pour une trame Ethernet, elle peut être de 256 avec SLIP (“ Serial Line IP ”) sur liaison série (RS232...).

Dans ces conditions, si la couche IP doit transmettre un bloc de données de taille supérieure au MTU à employer, il y a fragmentation !

Par exemple, un bloc de 1481 octets sur Ethernet sera décomposé en un datagramme de 1480 ( $1480 + 20 = 1500$ ) et un datagramme de 1 octet !

Il existe une exception à cette opération, due à la présence active du bit “ Don’t Fragment bit ” du champ **FLAGS** de l’en-tête IP. La présence à 1 de ce bit interdit la fragmentation dudit datagramme par la couche IP qui en aurait besoin. C’est une situation de blocage, la couche émettrice est tenue au courant par un message ICMP (cf paragraphe 4 page 59) “ *Fragmentation needed but don’t fragment bit set* ” et bien sûr le datagramme n’est pas transmis plus loin.

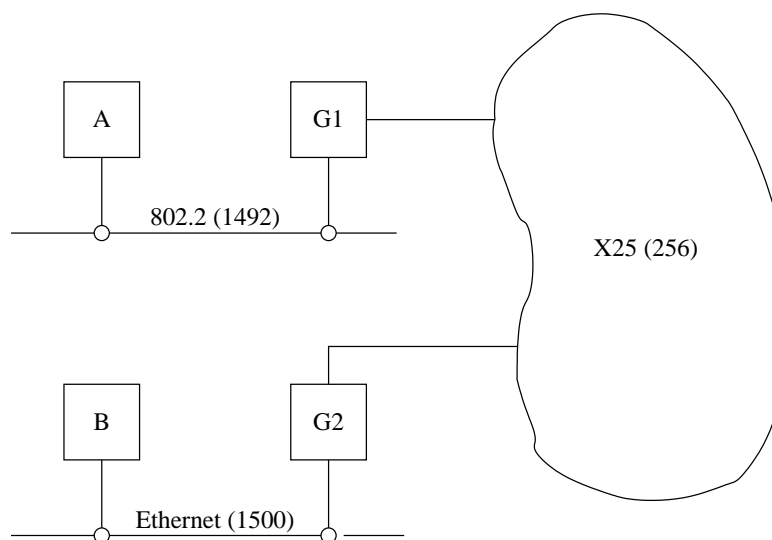
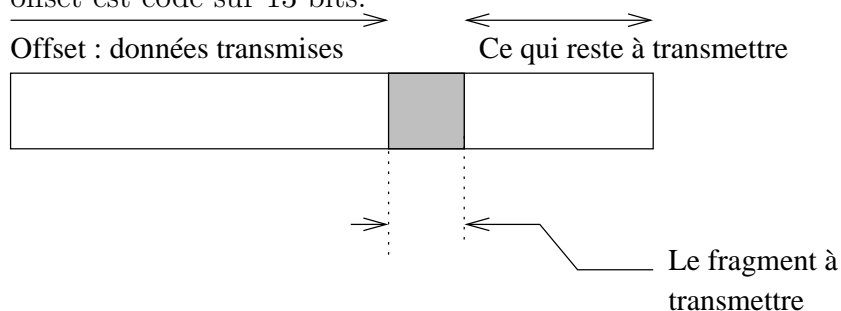


figure IV.03 — Fragmentation IP

### 1.4.1 Fragmentation

- Quand un datagramme est fragmenté, il n’est réassemblé que par la couche IP destinatrice finale. Cela implique trois remarques :
  1. La taille des datagrammes reçus par le destinataire final est directement dépendante du plus petit MTU rencontré.
  2. Les fragments deviennent des datagrammes à part entière.
  3. Rien ne s’oppose à ce qu’un fragment soit à nouveau fragmenté.
- Cette opération est absolument transparente pour les couches de transport qui utilisent IP.

- Quand un datagramme est fragmenté, chaque fragment comporte la même valeur de champ **IDENTIFICATION** que le datagramme initial. S'il y a encore des fragments, un des bits du champ **FLAGS** est positionné à 1 pour indiquer " More fragment " !  
Ce champ a une longueur de 3 bits.  
**FRAGMENT OFFSET** contient l'offset du fragment, relativement au datagramme initial.  
Cet offset est codé sur 13 bits.



*figure IV.04 — Fragment à transmettre*

Pour tous les fragments :

- Les données doivent faire un multiple de 8 octets, sauf pour le dernier fragment, évidemment.
- Le champ **TOTAL LENGTH** change.
- Chaque fragment est un datagramme indépendant, susceptible d'être à son tour fragmenté.

Pour le dernier fragment :

- **FLAGS** est remis à zéro.
- Les données ont une taille quelconque.

#### 1.4.2 Réassemblage

- Tous les datagrammes issus d'une fragmentation deviennent des datagrammes IP comme (presque) les autres.
- Ils arrivent à destination, peut être dans le désordre, dupliqués. IP doit faire le tri.
- il y a suffisamment d'information dans l'en-tête pour réassembler les fragments épars.
- **Mais** si un fragment manque, la totalité du datagramme est perdu car aucun mécanisme de contrôle n'est implémenté pour cela dans IP.

**C'est la raison principale pour laquelle il faut absolument éviter de fragmenter un datagramme IP !**

La *figure IV.05* résume l'opération de fragmentation d'un datagramme IP.

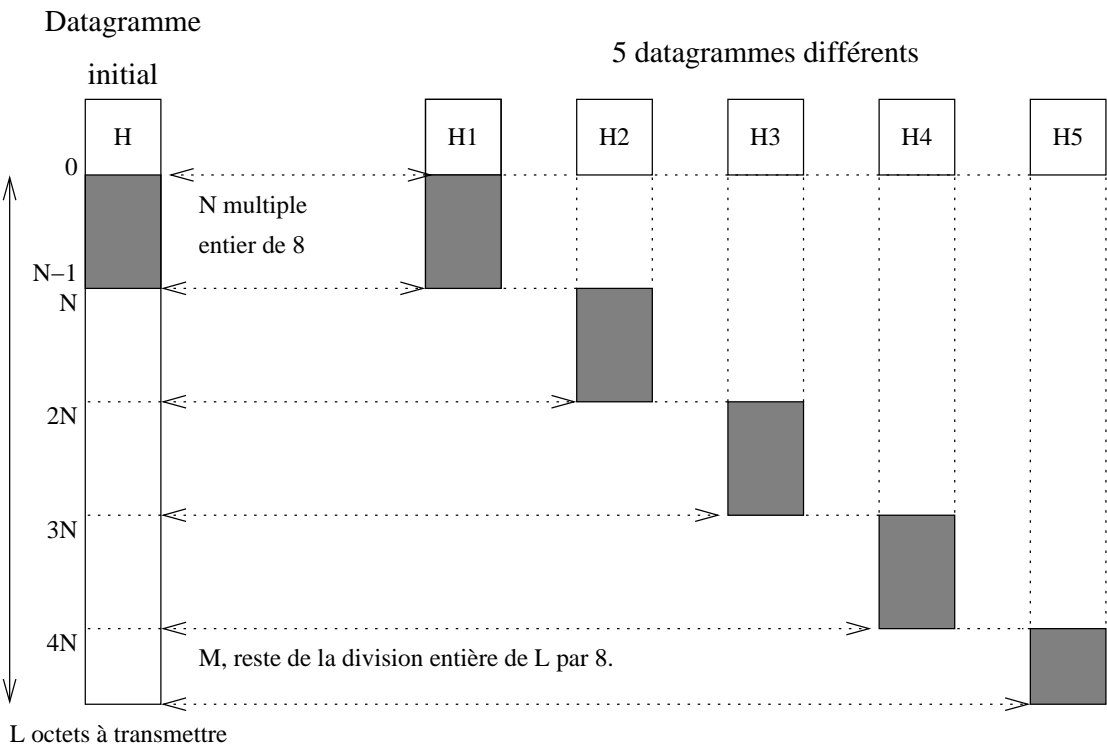


figure IV.05 — Résumé de la fragmentation

|                 | H1      | H2      | H3           | H4           | H5           |
|-----------------|---------|---------|--------------|--------------|--------------|
| IDENTIFICATION  | I       | I       | I            | I            | I            |
| FLAG            | MF      | MF      | MF           | MF           | 0            |
| OFFSET          | 0       | N       | $2 \times N$ | $3 \times N$ | $4 \times N$ |
| TOTAL LENGTH    | $H + N$ | $H + N$ | $H + N$      | $H + N$      | $H + M$      |
| HEADER CHECKSUM | $C_1$   | $C_2$   | $C_3$        | $C_4$        | $C_5$        |

Notez les variations de certains champs de l'en-tête :

1. IDENTIFICATION est le même pour tous
2. FLAG est 0 pour le dernier datagramme
3. OFFSET croît de la taille du fragment, ici N.
4. TOTAL LENGTH est généralement différent pour le dernier fragment, sauf cas particulier.
5. HEADER CHECKSUM change à chaque fois car l'OFFSET change (rappel : il ne tient pas compte des données).

## 2 Protocole ARP

ARP est l'acronyme de " Address Resolution Protocol ", il est définie dans la RFC 826.

- Le problème à résoudre est issu de la constatation qu'une adresse IP n'a de sens que pour la suite de protocole TCP/IP ; celle-ci étant indépendante de la partie matérielle il faut avoir un moyen d'établir un lien entre ces deux constituants.
- La norme Ethernet (vs IEEE) suppose l'identification unique de chaque carte construite et vendue<sup>4</sup>.
- Sur une même liaison physique (lire plus loin " même LAN "), Ethernet par exemple, deux machines peuvent communiquer  $\iff$  elles connaissent leurs adresses physiques respectives.

On suppose qu'une machine connaît sa propre adresse physique par un moyen qui n'est pas décrit ici (ne fait pas partie du protocole).

Remarque importante : Cette information n'a pas de sens dans le cadre d'une liaison de type " point à point " avec un protocole tel que ppp.

- Lors du premier échange entre 2 machines d'un même LAN, si les adresses physiques ne sont pas déjà connues (on verra pourquoi plus loin), la solution à ce problème passe par l'usage du protocole ARP.
- L'usage de ARP est complètement transparent pour l'utilisateur.

### 2.1 Fonctionnement

A demande à toutes les stations : étant donné l'adresse IP de B, que vaut son adresse physique ?

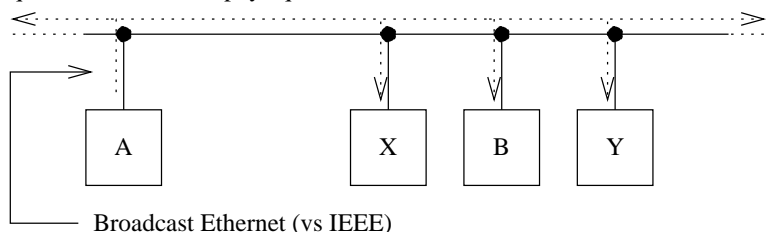


figure IV.06 — Question ARP

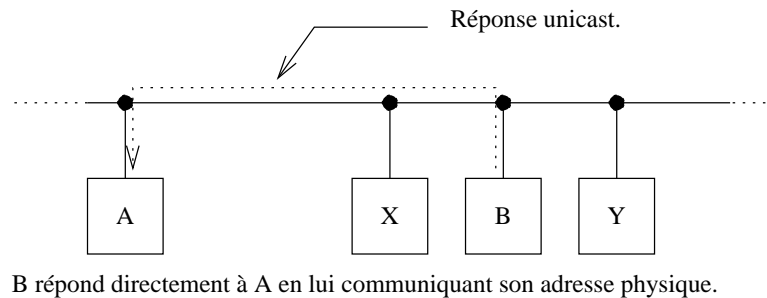
Sur la figure IV.06 la station Ethernet A ( $I_A$ ,  $P_A$ ) a besoin de connaître l'adresse physique de la station Ethernet B ( $I_B$ ,  $P_B$ ), pour ce faire elle envoie un datagramme de format spécial (cf paragraphe suivant), dédié à ARP, qui lui permet de poser la question (" Arp question ") à l'ensemble des machines actives. L'adresse de la machine qui doit répondre étant l'objet de la question, son adresse (champ destinataire) est donc remplacée par une adresse de " broadcast " (48 bits à 1).

Toutes les machines du LAN écoutent cet échange et peuvent mettre à jour leur table de conversion (adresse IP adresse Ethernet) pour la machine A.

<sup>4</sup>cf chapitre I " Réseaux locaux "

Le “broadcast”, coûteux en bande passante, est ainsi utilisé au maximum de ses possibilités. Sur la *figure IV.07* la réponse de B est du type “unicast”.

Remarque : quand une station Ethernet ne répond plus (cf ICMP) il y a suppression de l’association adresse IP - adresse MAC.



*figure IV.07 — Réponse ARP*

Si la station B ne répond pas, la station continuera à poser la question à intervalles réguliers pendant un temps infini...

Il n’est pas besoin d’utiliser ARP préalablement à chaque échange, car heureusement le résultat est mémorisé.

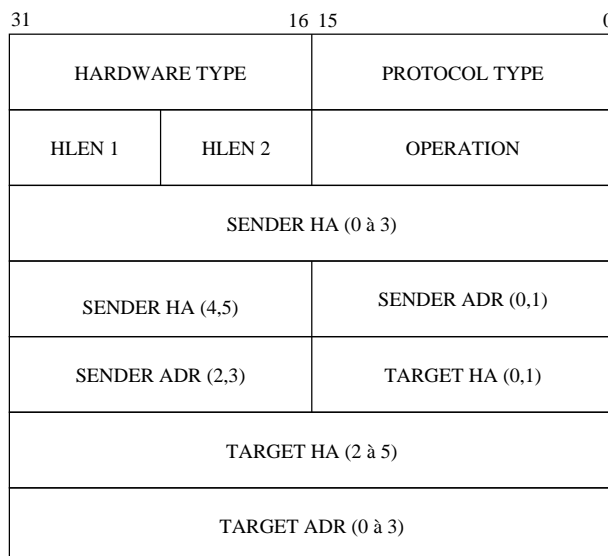
En règle générale la durée de vie d’une adresse en mémoire est de l’ordre de 20 minutes et chaque utilisation remet à jour ce compteur.

La commande `arp -a` sous Unix permet d’avoir le contenu de la table de la machine sur laquelle on se trouve, par exemple :

```
$ arp -a
soudirs.chezmoi.fr (192.168.192.10) at 8:0:9:85:76:9c
espoirs.chezmoi.fr (192.168.192.11) at 8:0:9:85:76:bd
plethore.chezmoi.fr (192.168.192.12) at 8:0:9:a:f9:aa
byzance.chezmoi.fr (192.168.192.13) at 8:0:9:a:f9:bc
ramidus.chezmoi.fr (192.168.192.14) at 0:4f:49:1:28:22 permanent
desiree.chezmoi.fr (192.168.192.33) at 8:0:9:70:44:52
pythie.chezmoi.fr (192.168.192.34) at 0:20:af:2f:8f:f1
ramidus.chezmoi.fr (192.168.192.35) at 0:4f:49:1:36:50 permanent
gateway.chezmoi.fr (192.168.192.36) at 0:60:8c:81:d5:1b
```

Enfin, et c’est un point très important, du fait de l’utilisation de “broadcast” physiques, les messages ARP ne franchissent pas les routeurs. Il existe cependant un cas particulier : le proxy ARP, que nous évoquerons succinctement à la fin de ce paragraphe.

## 2.2 Format du datagramme



*figure IV.08 — Datagramme ARP*

Le datagramme ci-dessus est encapsulé dans une trame physique du type 0x0806<sup>5</sup>.

**HARDWARE TYPE** pour spécifier le type d'adresse physique dans les champs **SENDER HA** et **TARGET HA**, c'est 1 pour Ethernet.

**PROTOCOL TYPE** pour spécifier le type d'adresse logique dans les champs **SENDER ADR** et **TARGET ADR**, c'est 0x0800 (même valeur que dans la trame Ethernet) pour des adresses IP.

**HLEN 1** pour spécifier la longueur de l'adresse physique (6 octets pour Ethernet).

**HLEN 2** pour spécifier la longueur de l'adresse logique (4 octets pour IP).

**OPERATION** ce champ précise le type de l'opération, il est nécessaire car la trame est la même pour toutes les opérations des deux protocoles qui l'utilisent.

|      | Question | Réponse |
|------|----------|---------|
| ARP  | 1        | 2       |
| RARP | 3        | 4       |

**SENDER HA** adresse physique de l'émetteur

**SENDER ADR** adresse logique de l'émetteur

**TARGET HA** adresse physique du destinataire

**TARGET ADR** adresse logique du destinataire

<sup>5</sup>voir ou revoir la *figure 11.02* du chapitre d'introduction à IP (page 25)

## 2.3 Proxy ARP

Le proxy ARP permet l'extension du lan à des hôtes qui ne lui sont pas directement physiquement reliés, mais qui s'y rattachent par exemple au travers d'une passerelle.

Un exemple très courant est celui d'un hôte qui accède à un réseau via un dialup (rtc, numéris, ...). Le NetID de son adresse IP peut alors être le même que celui du réseau rejoint, comme s'il y était physiquement raccordé. Ce subterfuge est rendu possible après configuration adéquate de la passerelle de raccordement.

## 3 Protocole RARP

RARP est l'acronyme de " Reverse Address Resolution Protocol ", il est défini dans la RFC 903 (BOOTP et DHCP en sont des alternatives avec plus de possibilités).

- Normalement une machine qui démarre obtient son adresse IP par lecture d'un fichier sur son disque dur (ou depuis sa configuration figée dans une mémoire non volatile).
- Pour certains équipements cette opération n'est pas possible voire même non souhaitée par l'administrateur du réseau :
  - Terminaux X Windows
  - Stations de travail " diskless "
  - Imprimante en réseau
  - " Boîtes noires " sans capacité autonome de démarrage
  - PC en réseau
  - ...
- Pour communiquer en TCP/IP une machine a besoin d'au moins une adresse IP, l'idée de ce protocole est de la demander au réseau.
- Le protocole RARP est adapté de ARP : l'émetteur envoie une requête RARP spécifiant son adresse physique dans un datagramme de même format que celui de ARP et avec une adresse de " broadcast " physique. Le champ OPERATION contient alors le code de " RARP question "
- Toutes les stations en activité reçoivent la requête, celles qui sont habilitées à répondre (serveurs RARP) complètent le datagramme et le renvoient directement ( " unicast ") à l'émetteur de la requête puisqu'elle connaissent son adresse physique.

Sur une machine Unix configurée en serveur RARP les correspondances entres adresses IP et adresses physiques sont enregistrées dans un fichier nommé généralement `/etc/bootptab`.



## 4 Protocole ICMP

ICMP est l'acronyme de “ Internet Control Message Protocol ”, il est historiquement défini dans la RFC 950.

Nous avons vu que le protocole IP ne vérifie pas si les paquets émis sont arrivés à leur destinataire dans de bonnes conditions.

Les paquets circulent d'une passerelle vers un autre jusqu'à en trouver une qui puisse les délivrer directement à un hôte. Si une passerelle ne peut router ou délivrer directement un paquet ou si un événement anormal arrive sur le réseau comme un trafic trop important ou une machine indisponible, il faut pouvoir en informer l'hôte qui a émis le paquet. Celui-ci pourra alors réagir en fonction du type de problème rencontré.

ICMP est un mécanisme de contrôle des erreurs au niveau IP, mais la *figure II.02* du chapitre d'introduction à IP (page 25) montre que le niveau *Application* peut également avoir un accès direct à ce protocole.

### 4.1 Le système de messages d'erreur

Dans le système que nous avons décrit, chaque passerelle et chaque hôte opère de manière autonome, route et délivre les datagrammes qui arrivent sans coordination avec l'émetteur.

Le système fonctionne parfaitement si toutes les machines sont en ordre de marche et si toutes les tables de routage sont à jour. Malheureusement c'est une situation idéale...

Il peut y avoir des ruptures de lignes de communication, des machines peuvent être à l'arrêt, en pannes, déconnectées du réseau ou incapables de router les paquets parcequ'en surcharge.

Des paquets IP peuvent alors ne pas être délivrés à leur destinataire et le protocole IP lui-même ne contient rien qui puisse permettre de détecter cet échec de transmission.

C'est pourquoi est ajouté systématiquement un mécanisme de gestion des erreurs connu sous le doux nom de ICMP. Il fait partie de la couche IP<sup>6</sup> et porte le numéro de protocole 1.

Ainsi, quand un message d'erreur arrive pour un paquet émis, c'est la couche IP elle-même qui gère le problème, la plupart des cas sans en informer les couches supérieures (certaines applications utilisent ICMP<sup>7</sup>).

Initialement prévu pour permettre aux passerelles d'informer les hôtes sur des erreurs de transmission, ICMP n'est pas restreint aux échanges passerelles-hôtes, des échanges entres hôtes sont tout à fait possibles.

Le même mécanisme est valable pour les deux types d'échanges.

---

<sup>6</sup>voir ou revoir la *figure II.02* du chapitre d'introduction à IP (page 25)

<sup>7</sup>Même figure qu'au point précédent

## 4.2 Format des messages ICMP

Chaque message ICMP traverse le réseau dans la partie DATA d'un datagramme IP :



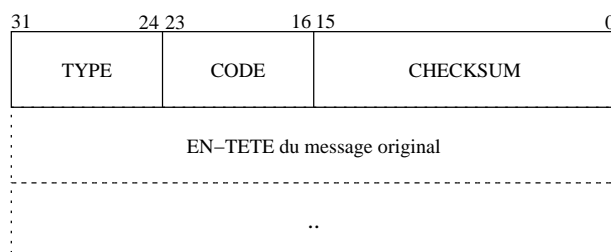
*figure IV.09 — Message ICMP*

La conséquence directe est que les messages ICMP sont routés comme les autres paquets IP au travers le réseau. Il y a toutefois une exception : il peut arriver qu'un paquet d'erreur rencontre lui-même un problème de transmission, dans ce cas on ne génère pas d'erreur sur l'erreur !

Il est important de bien voir que puisque les messages ICMP sont encapsulés dans un datagramme IP, ICMP n'est pas considéré comme un protocole de niveau plus élevé.

La raison de l'utilisation d'IP pour délivrer de telles informations, est que les messages peuvent avoir à traverser plusieurs réseaux avant d'arriver à leur destination finale. Il n'était donc pas possible de rester au niveau physique du réseau (à l'inverse de ARP ou RARP).

La *figure IV.10* décrit le format du message ICMP :



*figure IV.10 — Format d'un message ICMP*

Chaque message ICMP a un type particulier qui caractérise le problème qu'il signale. Un en-tête de 32 bits est composé comme suit :

**TYPE** contient le code d'erreur.

**CODE** complète l'information du champ précédent.

**CHECKSUM** est utilisé avec le même mécanisme de vérification que pour les datagrammes IP mais ici il ne porte que sur le message ICMP (rappel : le checksum de l'en-tête IP ne porte que sur son en-tête et non sur les données véhiculées).

En addition, les messages ICMP donnent toujours l'en-tête IP et les 64 premiers bits (les deux premiers mots de quatre octets) du datagramme qui est à l'origine du problème, pour permettre au destinataire du message d'identifier quel paquet est à l'origine du problème.

### 4.3 Quelques types de messages ICMP

Ce paragraphe examine quelques uns des principaux types de messages ICMP, ceux qui sont le plus utilisés. Il existe onze valeurs de **TYPE** différentes.

“ **Echo Request (8), Echo reply (0)** ” Une machine envoie un message ICMP “ echo request ” pour tester si son destinataire est accessible. N’importe quelle machine qui reçoit une telle requête doit formuler un message ICMP “ echo reply ” en retour<sup>8</sup>

Ce mécanisme est extrêmement utile, la plupart des implémentations le propose sous forme d’un utilitaire (**ping** sous Unix).

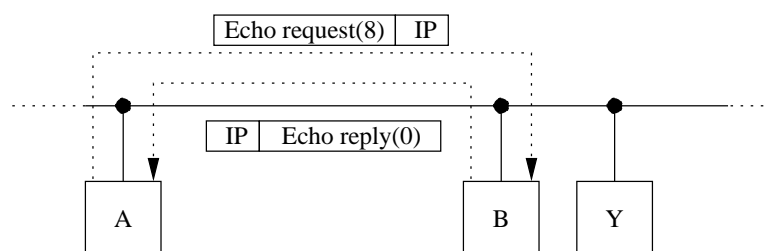


figure IV.11 — “ Echo request ” vs “ Echo reply ”

“ **Destination Unreachable (3)** ” Quand une passerelle ne peut pas délivrer un datagramme IP, elle envoie un message ICMP “ destination unreachable ” à l’émetteur.

Dans ce cas le champ **CODE** complète le message d’erreur avec :

- 0 “ Network unreachable ”
- 1 “ Host unreachable ”
- 2 “ Protocol unreachable ”
- 3 “ Port unreachable ”
- 4 “ Fragmentation needed and DF set ”
- 5 “ Source route failed ”

“ **Source Quench (4)** ” Quand un datagramme IP arrive trop vite pour une passerelle ou un hôte, il est rejeté.

Un paquet arrive “ trop vite ” quand la machine qui doit le lire est congestionnée, trop de trafic à suivre...

Dans ce cas la machine en question envoie un paquet ICMP “ source quench ” qui est interprété de la façon suivante :

L’émetteur ralentit le rythme d’envoi de ses paquets jusqu’à ce qu’il cesse de recevoir ce message d’erreur. La vitesse est donc ajustée par une sorte d’apprentissage rustique. Puis graduellement il augmente le débit, aussi longtemps que le message “ source quench ” ne revient pas

<sup>8</sup>Pour des raisons de sécurité certaines machines peuvent ne pas répondre.

Ce type de paquet **ICMP** a donc tendance à vouloir réguler le flux des datagrammes au niveau IP alors que c'est une fonctionnalité de la couche de transport (**TCP**).

C'est donc une sérieuse entorse à la règle d'indépendance des couches.

“ **Redirect (5)** ” Les tables de routage (Voir le paragraphe 6) des stations restent assez statiques durant de longues périodes. Le système d'exploitation les lit au démarrage sur le système de fichiers et l'administrateur en change de temps en temps les éléments.

Si entre deux modifications une destination change d'emplacement, la donnée initiale dans la table de routage peut s'avérer incorrecte.

Les passerelles connaissent de bien meilleures routes que les hôtes eux-mêmes, ainsi quand une passerelle détecte une erreur de routage, elle fait deux choses :

1. Elle envoie à l'émetteur du paquet un message **ICMP** “ redirect ”
2. Elle redirige le paquet vers la bonne destination.

Cette redirection ne règle pas les problèmes de routage car elle est limitée aux interactions entre passerelles et hôtes directement connectés.

La propagation des routes aux travers des réseaux multiples est un autre problème.

Le champ **CODE** du message **ICMP** peut avoir les valeurs suivantes :

- 0** “ Redirect datagram for the Net ”
- 1** “ Redirect datagram for the host ”
- 2** ...

“ **Router solicitation (10) vs Router advertisement (9)** ” Il s'agit d'obtenir ou d'annoncer des routes, nous verrons cela plus en détail dans le paragraphe 6.4.

“ **Time exceeded (11)** ” Chaque datagramme contient un champ **TTL** dit “ **TIME TO LIVE** ” appelé aussi “ hop count ”.

Afin de prévenir le cas où un paquet circulerait à l'infini d'une passerelle à une autre, chaque passerelle décrémente ce compteur et rejette le paquet quand le compteur arrive à zéro et envoie un message **ICMP** à l'émetteur pour le tenir au courant.

## 5 Protocole IGMP

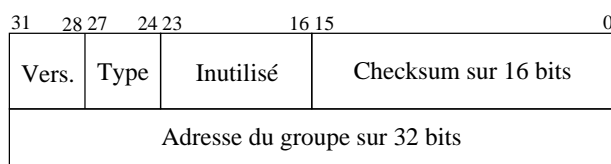
IGMP, l'acronyme de “ Internet Group Management Protocol ”, est historiquement défini dans l'Annexe I de la RFC 1112.

Sa raison d'être est que les datagrammes ayant une adresse multicast<sup>9</sup> sont à destination d'un groupe d'utilisateurs dont l'émetteur ne connaît ni le nombre ni l'emplacement. L'usage du multicast étant par construction dédié aux applications comme la radio ou la vidéo sur le réseau<sup>10</sup>, donc consommatrices de bande passante, il est primordial que les routeurs aient un moyen de savoir s'il y a des utilisateurs de tel ou tel groupe sur les LANs directement accessibles pour ne pas encombrer les bandes passantes associées avec des flux d'octets que personne n'utilise plus !

### 5.1 Description de l'en-tête

IGMP est un protocole de communication entre les routeurs susceptibles de transmettre des datagrammes multicast et des hôtes qui veulent s'enregistrer dans tel ou tel groupe. IGMP est encapsulé dans IP<sup>11</sup> avec le protocole numéro 2. Comme le montre la *figure*

*Romanchapter.12*, sa taille est fixe (contrairement à ICMP) : seulement 2 mots de 4 octets.



*figure IV.12 — En-tête IGMP*

**Version** Version 1.

**Type** Ce champ prend deux valeurs, **1** pour dire qu'il s'agit d'une question (query d'un routeur), **2** pour dire qu'il s'agit de la réponse d'un hôte.

**Inutilisé** ...

**Checksum** Le checksum est calculé comme celui d'ICMP.

**Adresse** C'est l'adresse multicast (classe D) à laquelle appartient l'hôte qui répond.

<sup>9</sup>Voir page 42

<sup>10</sup>La première expérience à grande échelle du multicast fut sans doute la conférence de l'IETF en mars 1992. Le papier `ftp://venera.isi.edu/ietf-audiocast-article.ps` relate cette expérience.

<sup>11</sup>voir ou revoir la *figure II.02* du chapitre I d'introduction à IP (page 25)

## 5.2 Fonctionnement du protocole

La RFC 1112 précise que les routeurs multicast envoient des messages de questionnement (**Type=Queries**) pour reconnaître quels sont les éventuels hôtes appartenant à quel groupe. Ces questions sont envoyées à tous les hôtes des LANs directement raccordés à l'aide de l'adresse multicast du groupe 224.0.0.1<sup>12</sup> encapsulé dans un datagramme IP ayant un champ TTL=1. Tous les hôtes susceptibles de joindre un groupe multicast écoutent ce groupe par hypothèse.

Les hôtes, dont les interfaces ont été correctement configurées, répondent à une question par autant de réponses que de groupes auxquels ils appartiennent sur l'interface réseau qui a reçu la question. Afin d'éviter une " tempête de réponses " chaque hôte met en œuvre la stratégie suivante :

1. Un hôte ne répond pas immédiatement à la question reçue. Pour chaque groupe auquel il appartient, il attend un délais compris entre 0 et 10 secondes, calculé aléatoirement à partir de l'adresse IP unicast de l'interface qui a reçu la question, avant de renvoyer sa réponse. La *figure Romanchapter.13* montre un tel échange, remarquez au passage la valeur des adresses.
2. La réponse envoyée est écoutée par tous les membres du groupe appartenant au même LAN. Tout ceux qui s'apprêtaient à envoyer une telle réponse au serveur en interrompent le processus pour éviter une redite. Le routeur ne reçoit ainsi qu'une seule réponse pour chaque groupe, et pour chaque LAN, ce qui lui suffit pour justifier le routage demandé.

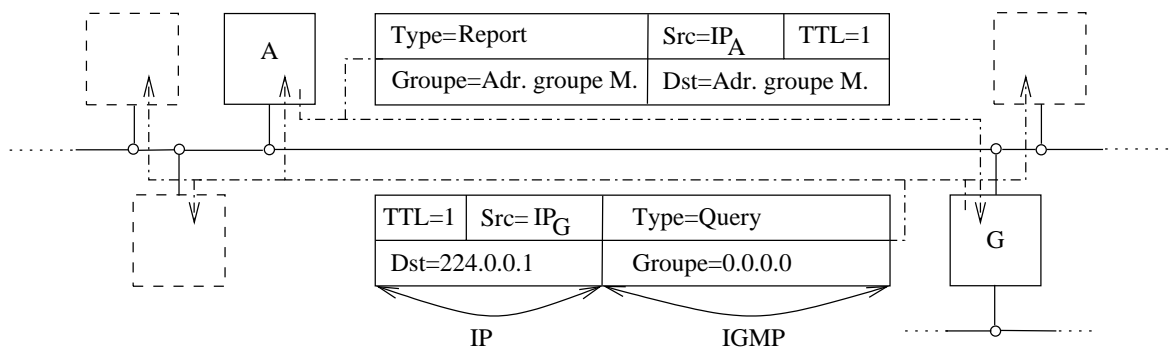


figure IV.13 — Fonctionnement IGMP

Il y a deux exceptions à la stratégie ci-dessus. La première est que si une question est reçue alors que le compte à rebours pour répondre à une réponse est en cours, il n'est pas interrompu.

La deuxième est qu'il n'y a jamais de délai appliqué pour l'envoi de datagramme portant l'adresse du groupe de base 224.0.0.1.

Pour rafraîchir leur connaissance des besoins de routage les routeurs envoient leurs questions avec une fréquence très faible de l'ordre de la minute,

<sup>12</sup> " tous les hôtes du LAN "

afin de préserver au maximum la bande passante du réseau. Si aucune réponse ne leur parvient pour tel ou tel groupe demandé précédemment, le routage s'interrompt.

Quand un hôte rejoint un groupe, il envoie immédiatement une réponse (**type=report**) pour le groupe (les) qui l'intéresse, plutôt que d'attendre une question du routeur. Au cas où cette réponse se perdrait il est recommandé d'effectuer une réémission dans un court délai.

**Remarques :**

1. Sur un LAN sans routeur pour le multicast, le seul trafic IGMP est celui des hôtes demandant à rejoindre tel ou tel groupe.
2. Il n'y a pas de **report** pour quitter un groupe.
3. La plage d'adresses multicast entre 224.0.0.0 et 224.0.0.225 est dédiée aux applications utilisant une valeur de 1 pour le champ TTL (administration et services au niveau du LAN). Les routeurs ne doivent pas transmettre de tels datagrammes.
4. Il n'y a pas de message ICMP sur les datagrammes ayant une adresse de destination du type multicast.

En conséquence les applications qui utilisent le multicast (avec une adresse supérieure à 224.0.0.225) pour découvrir des services, doivent avoir une stratégie pour augmenter la valeur du champ TTL en cas de non réponse.

### 5.3 Fonctionnement du Mbone

*Précisions en cours...*

## 6 Routage IP

Ce paragraphe décrit de manière succincte le routage des datagrammes. Sur l'Internet, ou au sein de toute entité qui utilise IP, les datagrammes ne sont pas routés par des machines Unix, mais par des routeurs dont c'est la fonction par définition. Ils sont plus efficaces et plus perfectionnés pour cette tâche par construction, et surtout autorisent l'application d'une politique de routage ("routing policy") ce que la pile IP standard d'une machine Unix ne sait pas faire. Toutefois il est courant dans les "petits réseaux", ou quand le problème à résoudre reste simple, de faire appel à une machine Unix pour ce faire<sup>13</sup>.

Dans ce paragraphe nous examinons le problème du routage de manière synthétique, nous l'aborderons plus en détail les aspects techniques du routage dynamique au chapitre VII, page 109.

Le routage des datagrammes se fait au niveau de la couche IP, et c'est son travail le plus important. Toutes les machines multiprocessus sont théoriquement capables d'effectuer cette opération.

La différence entre un "routeur" et un "hôte" est que le premier est capable de transmettre un datagramme d'un interface à un autre et pas le deuxième.

Cette opération est délicate si les machines qui doivent dialoguer sont connectées à de multiples réseaux physiques.

D'un point de vue idéal établir une route pour des datagrammes devrait tenir compte d'éléments comme la charge du réseau, la taille des datagrammes, le type de service demandé, les délais de propagation, l'état des liaisons, le trajet le plus court. . . La pratique est plus rudimentaire !

Il s'agit de transporter des datagrammes aux travers de multiples réseaux physiques, donc aux travers de multiples passerelles.

On divise le routage en deux grandes familles :

**Le routage direct** Il s'agit de délivrer un datagramme à une machine raccordée au même LAN.

L'émetteur trouve l'adresse physique du correspondant (ARP), encapsule le datagramme dans une trame et l'envoie.

**Le routage indirect** Le destinataire n'est pas sur le même LAN comme précédemment. Il est absolument nécessaire de franchir une passerelle connue d'avance ou d'employer un chemin par défaut.

En effet, toutes les machines à atteindre ne sont pas forcément sur le même réseau physique. C'est le cas le plus courant, par exemple sur l'Internet qui regroupe des centaines de milliers de réseaux différents.

Cette opération est beaucoup plus délicate que la précédente car il faut sélectionner une passerelle.

---

<sup>13</sup>On peut consulter par exemple [http://www.freebsd.org/\\$\sim\\$picobsd/](http://www.freebsd.org/$\sim$picobsd/), où le site du projet Zebra de GNU <http://www.zebra.org/>

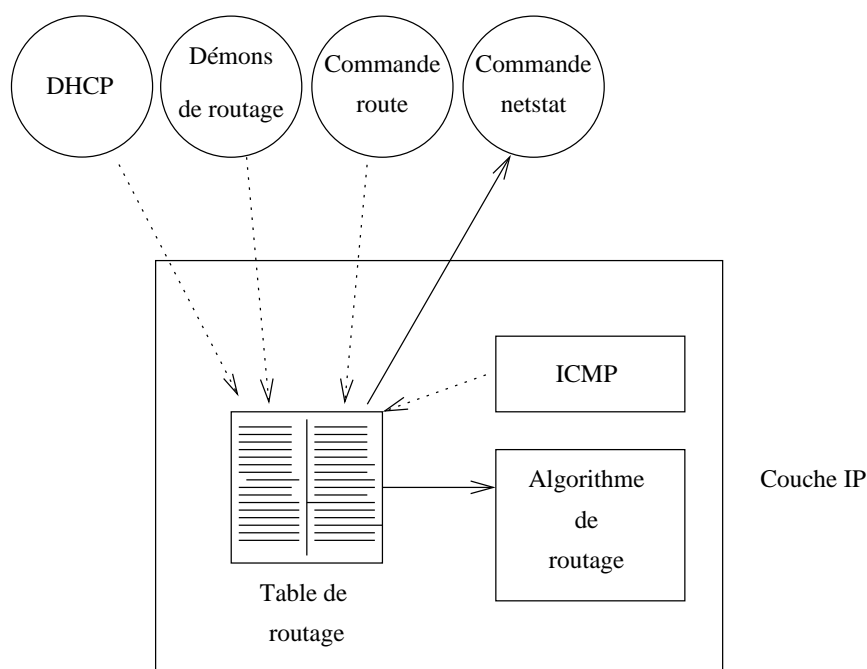


Parceque le routage est une opération fondamentalement orientée “réseau”, le routage s’appuie sur cette partie de l’adresse IP du destinataire. La couche IP détermine celle-ci en examinant les bits de poids fort qui conditionnent la classe d’adresse et donc la segmentation “network.host”. Dans certain cas (CIDR) le masque de sous réseau est aussi employé.

Muni de ce numéro de réseau, la couche IP examine les informations contenues dans sa table de routage :

## 6.1 Table de routage

Sous Unix toutes les opérations de routage se font grâce à une table, dite “table de routage”, qui se trouve dans le noyau lui-même. La *figure IV.14* résume la situation :



*figure IV.14* — Table de routage

Cette table est très fréquemment utilisée par IP : sur un serveur plusieurs centaines de fois par secondes.

### Comment est-elle créée ?

Au démarrage avec la commande **route**, invoquée dans les scripts de lancement du système, et en fonctionnement :

- Au coup par coup avec la commande **route**, à partir du shell (administrateur système uniquement).
- Dynamiquement avec les démons de routage “**routed**” ou “**gated**” (la fréquence de mise à jour est typiquement de l’ordre de 30 sec.).

- Par des messages “ ICMP redirect ”.

La commande `netstat -rn` permet de la visualiser au niveau de l'interface utilisateur ( “ Application layer ” ) :

```
$ netstat -rn
Routing tables

Internet:
Destination            Gateway                Flags
default                192.168.192.36        UGS
127.0.0.1              127.0.0.1            UH
192.168.192/27         link#1                UC
192.168.192.10         8:0:9:85:76:9c        UHLW
192.168.192.11         8:0:9:85:76:bd        UHLW
192.168.192.12         8:0:9:88:8e:31        UHLW
192.168.192.13         8:0:9:a:f9:bc         UHLW
192.168.192.14         0:4f:49:1:28:22       UHLW
192.168.192.15         link#1                UHLW
192.168.192.32/27     link#2                UC
192.168.192.33         8:0:9:70:44:52        UHLW
192.168.192.34         0:20:af:2f:8f:f1      UHLW
192.168.192.35         0:4f:49:1:36:50       UHLW
192.168.192.36         link#2                UHLW
```

On peut mémoriser cette table comme étant essentiellement composée d'une colonne origine, d'une colonne destination.

De plus, chaque route qui désigne une passerelle (ici la route par défaut) doit s'accompagner d'un nombre de sauts ( “ hop ” ), ou encore métrique, qui permet le choix d'une route plutôt qu'une autre en fonction de cette valeur. Chaque franchissement d'un routeur compte pour un saut. Dans la table ci-dessus, la métrique de la route par défaut est 1.

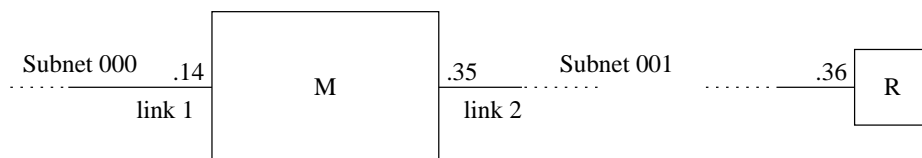
*Remarque* : la sortie de la commande `netstat -rn` ci-dessus a été simplifiée.<sup>14</sup>

Les drapeaux ( “ flags ” ) les plus courants :

- C c La route est générée par la machine, à l'usage.
- D La route a été créée dynamiquement (démons de routage).
- G La route désigne une passerelle, sinon c'est une route directe.
- H La route est vers une machine, sinon elle est vers un réseau.
- L Désigne la conversion vers une adresse physique (cf ARP).
- M La route a été modifiée par un “ redirect ”.
- S La route a été ajoutée manuellement.
- U La route est active.
- W La route est le résultat d'un clonage.

<sup>14</sup>Des colonnes Refs, Use et Netif

La *figure IV.15* précise l'architecture du réseau autour de la machine sur laquelle a été exécuté le `netstat`.



*figure IV.15* — Situation réseau lors du `netstat`

## 6.2 Routage statique

Comme nous avons pu le deviner au paragraphe précédent, les routes statiques sont celles créées au démarrage de la machine ou ajoutées manuellement par l'administrateur système, en cours de fonctionnement.

Le nombre de machines possibles à atteindre potentiellement sur l'Internet est beaucoup trop élevé pour que chaque machine puisse espérer en conserver l'adresse, qui plus est, même si cela était concevable, cette information ne serait jamais à jour donc inutilisable.

Plutôt que d'envisager la situation précédente on préfère restreindre l'étendue du "monde connu" et utiliser la "stratégie de proche en proche" précédemment citée.

Si une machine ne peut pas router un datagramme, elle connaît (ou est supposée connaître) l'adresse d'une passerelle supposée être mieux informée pour transmettre ce datagramme.

Dans l'exemple de sortie de la commande `netstat` du paragraphe 6.1, on peut reconnaître que l'administrateur système n'a configuré qu'une seule route "manuellement"<sup>15</sup>, toutes les autres lignes ont été déduites par la couche IP elle-même.

La *figure IV.16* met en situation plusieurs réseaux et les passerelles qui les relient. Voici une version très simplifiée des tables de routage statiques présentes sont les machines A, B, R1 et R2 :

**Machine A** default : 192.168.192.251

**Machine B** default : 10.1.1.1

**Routeur R1** 10 : 172.16.10.3

**Routeur R2** 192.168.192 : 172.16.10.1

<sup>15</sup>Ce n'est pas tout à fait exact, nous verrons pourquoi au paragraphe concernant l'interface de "loopback" (6.6).

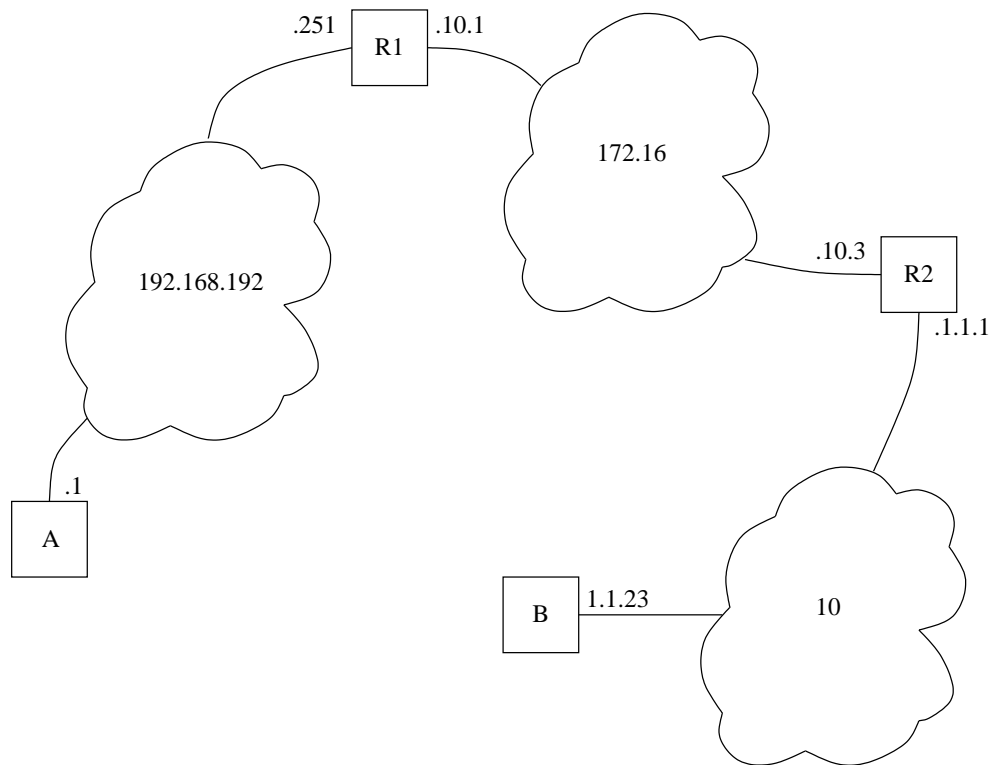


figure IV.16 — Exemple de nuage avec routage statique

### 6.2.1 Algorithme de routage

Cet algorithme simplifié résume les opérations de la couche IP pour choisir une destination, en fonction de sa table de routage. Cette opération est essentiellement basée sur le numéro de réseau,  $I_N$ , extrait de l'adresse IP,  $I_D$ . M désigne la machine sur laquelle s'effectue le routage.

**Si**  $I_N$  est un numéro de réseau auquel M est directement reliée :

- Obtenir l'adresse physique de la machine destinatrice
- Encapsuler le datagramme dans une trame physique et l'envoyer directement.

**Sinon Si**  $I_D$  apparaît comme une machine à laquelle une route spéciale est attribuée, router le datagramme en fonction.

**Sinon Si**  $I_N$  apparaît dans la table de routage, router le datagramme en fonction.

**Sinon S'il existe une route par défaut** router le datagramme vers la passerelle ainsi désignée.

**Sinon** Déclarer une erreur de routage (ICMP).

### 6.3 Routage dynamique

Si la topologie d'un réseau offre la possibilité de plusieurs routes pour atteindre une même destination, s'il est vaste et complexe, sujet à des changements fréquents de configuration. . . Le routage dynamique est alors un bon moyen d'entretenir les tables de routages et de manière automatique.

Il existe de nombreux protocoles de routage dynamique dont certains sont aussi anciens que l'Internet. Néanmoins tous ne conviennent pas à tous les types de problème, il en existe une hiérarchie.

Schématiquement on peut imaginer l'Internet comme une hiérarchie de routeurs. Les routeurs principaux ( " core gateways " ) de cette architecture utilisent entre-eux des protocoles comme GGP ( " Gateway to Gateway Protocol " ), l'ensemble de ces routeurs forment ce que l'on nomme l' " Internet Core " .

En bordure de ces routeurs principaux se situent les routeurs qui marquent la frontière avec ce que l'on nomme les " Autonomous systems ", c'est à dire des systèmes de routeurs et de réseaux qui possèdent leurs mécanismes propres de propagation des routes. Le protocole utilisé par ces routeurs limitrophes est souvent EGP ( " Exterior Gateway Protocol " ) ou BGP ( " Border Gateway Protocol " ).

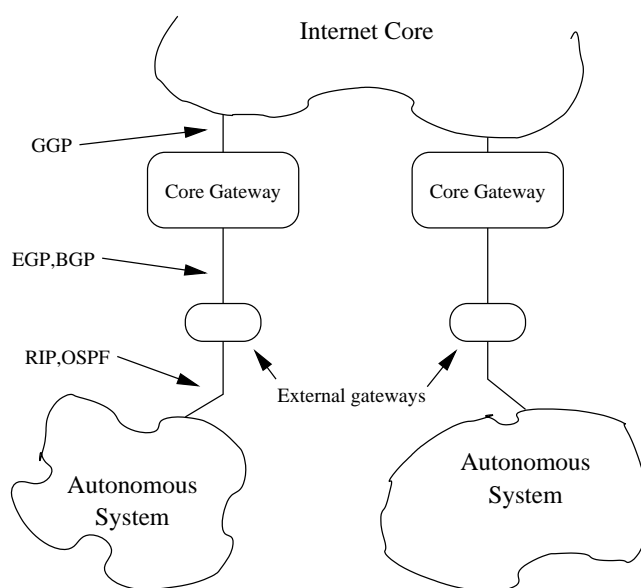


figure IV.17 — Exemple pour routage dynamique

Au sein d'un système autonome on utilise un IGP ( " Interior Gateway Protocol " ) c'est à dire un " protocole de gateways intérieurs ". Les protocoles les plus couramment employés sont RIP ( " Routing Information Protocol " ) qui est simple à comprendre et à utiliser, ou encore OSPF ( " Open Shortest Path First " ) plus récent, plus capable mais aussi beaucoup plus complexe à comprendre dans son mode de fonctionnement, ou encore IS-IS de la couche ISO de l'OSI.

### 6.3.1 RIP — “ Routing Information Protocol ”

RIP est apparu avec la version BSD d'Unix, il est documenté dans la RFC 1058 (1988 - Version 1 du protocole) et la RFC 1388 (1993 - Version 2 du protocole). Ce protocole est basé sur des travaux plus anciens menés par la firme Xerox.

RIP utilise le concept de “ vecteur de distance ”, qui s'appuie sur un algorithme de calcul du chemin le plus court dans un graphe. Le graphe est celui des routeurs, la longueur du chemin est établie en nombre de sauts (“ hop ”), ou métrique, entre la source et la destination, c'est à dire en comptant toutes les liaisons. Cette distance est exprimée comme un nombre entier variant entre 1 et 15 ; la valeur 16 est considérée comme l'infini et indique une mise à l'écart de la route.

Chaque routeur émet dans un datagramme portant une adresse IP de broadcast, à fréquence fixe (environ 30 secondes), le contenu de sa table de routage et écoute celle des autres routeurs pour compléter sa propre table. Ainsi se propagent les tables de routes d'un bout à l'autre du réseau. Pour éviter une “ tempêtes de mises à jours ”, le délais de 30 secondes est augmenté d'une valeur aléatoire comprise entre 1 et 5 secondes.

Si une route n'est pas annoncée au moins une fois en trois minutes, la distance devient “ infinie ”, et la route sera retirée un peu plus tard de la table (elle est propagée avec cette métrique).

L'adresse IP utilisée est une adresse de multipoint (“ multicast ”) comme nous verrons au paragraphe 6.4

Depuis la définition de RIPv2 les routes peuvent être accompagnées du masque de sous réseau qui les caractérise. Ainsi on peut avoir la situation suivante :

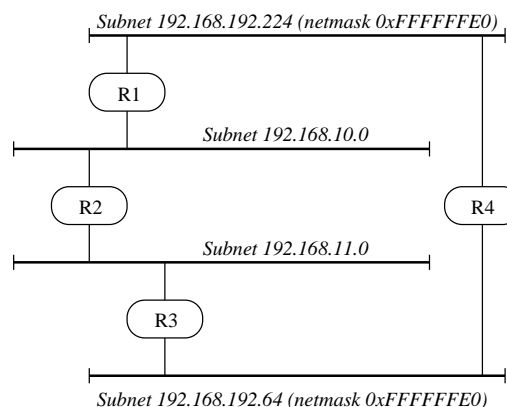


figure IV.18 — Topologie pour routage dynamique

Après propagation des routes, la table de routage du routeur R1 pourrait bien ressembler à :

| Source          | Destination | Coût |
|-----------------|-------------|------|
| 192.168.192.224 | R1          | 1    |
| 192.168.10.0    | R1          | 1    |
| 192.168.11.0    | R2          | 2    |
| 192.168.192.64  | R3          | 3    |

Avec une route par défaut qui est le routeur R2. La constitution de cette table n'est possible qu'avec RIPv2 étant donné l'existence des deux sous-réseaux de la classe C 192.168.192.

Le fonctionnement de ce protocole est détaillé page 113

### 6.3.2 OSPF — “ Open Shortest Path First ”

Contrairement à RIP, OSPF n'utilise pas de vecteur de distances mais base ses décisions de routage sur le concept d'“ états des liaisons ”. Celui-ci permet un usage beaucoup plus fin des performances réelles des réseaux traversés, parceque cette métrique est changeante au cours du temps. Si on ajoute à cela une méthode de propagation très rapide des routes par inondation, sans boucle et la possibilité de chemin multiples, OSPF, bien que beaucoup plus complexe que RIP, a toutes les qualités pour le remplacer, même sur les tous petits réseaux.

OSPF doit son nom à l'algorithme d'Edsger W. Dijkstra<sup>16</sup> de recherche du chemin le plus court d'abord lors du parcours d'un graphe. Le “ Open ” vient du fait qu'il s'agit d'un protocole ouvert de l'IETF, dans la RFC 2328...

Le fonctionnement de ce protocole est détaillé page 121

## 6.4 Découverte de routeur et propagation de routes

Au démarrage d'une station, plutôt que de configurer manuellement les routes statiques, surtout si elle sont susceptibles de changer et que le nombre de stations est grand, il peut être intéressant de faire de la “ découverte automatique de routeurs ” (RFC 1256).

À intervalles réguliers les routeurs diffusent des messages ICMP de type 9 (“ router advertisement ”) d'annonces de routes. Ces messages ont l'adresse multicast 224.0.0.1, qui est a destination de tous les hôtes du LAN.

Toutes les stations capables de comprendre le multicast (et convenablement configurées pour ce faire) écoutent ces messages et mettent à jour leur table.

Les stations qui démarrent peuvent solliciter les routeurs si l'attente est trop longue (environ 7 minutes) avec un autre message ICMP, de type 10 (“ router sollicitation ”) et avec l'adresse multicast 224.0.0.2 (à destination de tous les routeurs de ce LAN). La réponse du ou des routeurs est du type “ unicast ”, sauf si le routeur s'apprêtait à émettre une annonce.

<sup>16</sup> <http://www.cs.utexas.edu/users/EWD/>

À chaque route est associé un niveau de préférence et une durée de validité, définis par l'administrateur du réseau. Une validité nulle indique un routeur qui s'arrête et donc une route qui doit être supprimée.

Si entre deux annonces une route change, le mécanisme de “ ICMP redirect ”, examiné au paragraphe suivant, corrige l'erreur de route.

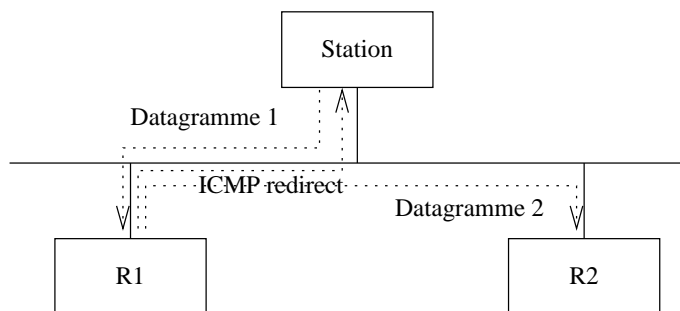
La découverte de routeur n'est pas un protocole de routage, son objectif est bien moins ambitieux : obtenir une route par défaut.

Il est intéressant de noter sur les machines FreeBSD c'est le démon de routage `routed` qui effectue ce travail à la demande <sup>17</sup>

## 6.5 Message ICMP “ redirect ”

La table de routage peut être modifiée dynamiquement par un message ICMP (IV).

La situation est celle de la *figure IV.21*.



*figure IV.21 — ICMP “ redirect ”*

- La station veut envoyer un datagramme et sa table de routage lui commande d'utiliser la route qui passe par le routeur R1.
- Le routeur R1 reçoit le datagramme, scrute sa table de routage et s'aperçoit qu'il faut désormais passer par R2. Pour se faire :
  1. Il re-route le datagramme vers R2, ce qui évite qu'il soit émis deux fois sur le LAN.
  2. Il envoie un message “ ICMP redirect ” (type 5) à la station, lui indiquant la nouvelle route vers R2.

Ce travail s'effectue pour chaque datagramme reçu de la station.

- Dès que la station reçoit le message “ ICMP redirect ” elle met à jour sa table de routage. La nouvelle route est employée pour les datagrammes qui suivent (vers la même direction).

La route modifiée est visible avec la commande `netstat -r`, elle figure avec le drapeau 'M' (modification dynamique).

Pour des raisons évidentes de sécurité, cette possibilité n'est valable que sur un même LAN.

<sup>17</sup>À condition d'activer avec `router.enable=YES` dans le fichier `/etc/rc.conf`.



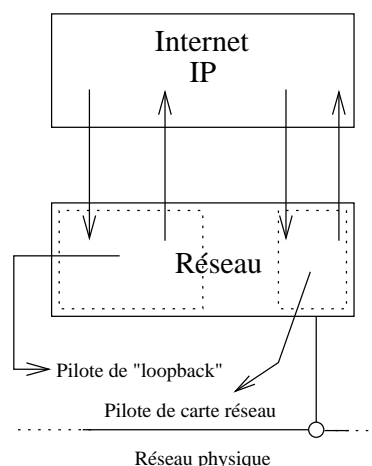
## 6.6 Interface de “ loopback ”

Toutes les implémentations d'IP supportent une interface de type “ loopback ”. L'objet de cette interface est de pouvoir utiliser les outils du réseau en local, sans passer par un interface réseau réel (associé à une carte physique).

La *figure IV.22* ci-contre, montre que la couche IP peut utiliser, selon le routage, l'interface standard du réseau, où l'interface de loopback.

Le routage est ici bien sûr basé sur l'adresse IP associée à chacune des interfaces. Cette association est effectuée sur une machine Unix à l'aide de la commande `ifconfig`, qui établit une correspondance entre un pilote de périphérique (repéré par son fichier spécial) et une adresse IP.

Dans le cas du pilote de loopback, l'adresse est standardisée à n'importe quelle adresse valide du réseau 127 (page 37).



*figure IV.22 — Interface de “ loopback ”*

La valeur courante est 127.0.0.1, d'où l'explication de la ligne ci-dessous déjà rencontrée (page 67) dans le cadre de la table de routage :

### Routing tables

#### Internet:

| Destination | Gateway   | Flags | Netif |
|-------------|-----------|-------|-------|
| ...         |           |       |       |
| 127.0.0.1   | 127.0.0.1 | UH    | lo0   |
| ...         |           |       |       |

Dans toutes les machines Unix modernes cette configuration est déjà prévue d'emblée dans les scripts de démarrage.

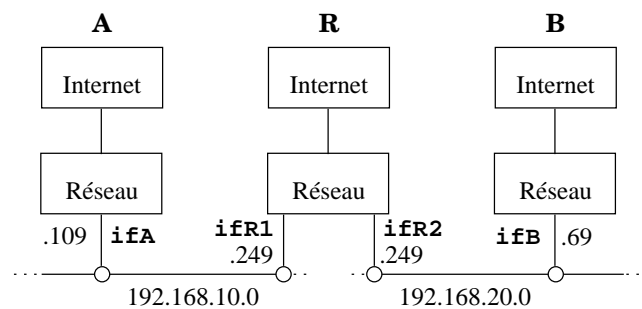
Concrètement, tout dialogue entre outils clients et serveurs sur une même machine est possible et même souhaitable sur cet interface pour améliorer les performances et parfois la sécurité<sup>18</sup>.

L'exemple d'usage le plus marquant est sans doute celui du serveur de noms (voir page 165) qui tient compte explicitement de cet interface dans sa configuration.

<sup>18</sup>Nous verrons ultérieurement (cf chapitre VIII) que le filtrage IP sur le 127/8 est aussi aisé que sur n'importe quel autre réseau

## 7 Finalement, comment ça marche ?

Dans ce paragraphe nous reprenons la *figure III.06* (page 44) et nous y apportons comme était annoncé une explication du fonctionnement qui tienne compte des protocoles principaux examinés dans ce chapitre. Pour cela nous utilisons deux réseaux privés de la RFC 1918 : 192.168.10.0 et 192.168.20.0 et nous faisons l'hypothèse que la passerelle fonctionne comme une machine Unix qui ferait du routage entre deux de ses interfaces !



*figure IV.23 — Illustration du routage direct et indirect*

Ce tableau résume l'adressage physique et logique de la situation :

| Interface | Adresse MAC       | Adresse IP     |
|-----------|-------------------|----------------|
| ifA       | 08:00:20:20:cf:af | 192.168.10.109 |
| ifB       | 00:01:e6:a1:07:64 | 192.168.20.69  |
| ifR1      | 00:06:5b:0f:5a:1f | 192.168.10.249 |
| ifR2      | 00:06:5b:0f:5a:20 | 192.168.20.249 |

Nous faisons en outre les hypothèses suivantes :

1. Les caches “ arp ” des machines **A**, **B** et **R** sont vides
2. La machine **A** a connaissance d'une route vers le réseau 192.168.20 passant par 192.168.10.249 et réciproquement la machine **B** voit le réseau 192.168.10.0 via le 192.168.20.249
3. La machine **A** a connaissance de l'adresse IP de la machine **B**

**La machine A envoie un datagramme à la machine B, que se passe-t-il sur le réseau ?**

**Étape 1** La machine **A** applique l'algorithme de routage (page 70) et s'aperçoit que la partie réseau de l'adresse de **B** n'est pas dans le même LAN (192.168.10/24 et 192.168.20/20 différent).

L'hypothèse 2 entraîne qu'une route existe pour atteindre ce réseau, passant par **R**. L'adresse IP de **R** est dans le même LAN, **A** peut donc atteindre **R** par un routage direct. La conséquence de l'hypothèse 1 implique que pour atteindre **R** directement il nous faut d'abord déterminer son adresse physique. Le protocole ARP (page 55) doit être utilisé.

**A** envoie en conséquence une trame ARP (page 57 comportant les éléments suivants :

```
SENDER HA    08:00:20:20:cf:af
SENDER ADR    192.168.10.109
TARGET HA     ff:ff:ff:ff:ff:ff
TARGET ADR    192.168.10.249
```

Avec un champ OPERATION qui contient la valeur 1, comme “ question ARP ”.

Remarquez qu’ici l’adresse IP destination est celle de **R** !

**Étape 2** **R** répond à la “ question ARP ” par une “ réponse ARP ” (OPERATION contient 2) et un champ complété :

```
SENDER HA    00:06:5b:0f:5a:1f
SENDER ADR    192.168.10.249
TARGET HA     08:00:20:20:cf:af
TARGET ADR    192.168.10.109
```

**Étape 3** **A** est en mesure d’envoyer son datagramme à **B** en passant par **R**. Il s’agit de routage indirect puisque l’adresse de **B** n’est pas sur le même LAN. Les adresses physiques et logiques se répartissent maintenant comme ceci :

```
IP SOURCE     192.168.10.109
IP TARGET     192.168.20.69
MAC SOURCE    08:00:20:20:cf:af
MAC TARGET    00:06:5b:0f:5a:1f
```

Remarquez qu’ici l’adresse IP destination est celle de **B** !

**Étape 4** **R** a reçu le datagramme depuis **A** et à destination de **B**. Celle-ci est sur un LAN dans lequel **R** se trouve également, un routage direct est donc le moyen de transférer le datagramme. Pour la même raison qu’à l’étape 1 **R** n’a pas l’adresse MAC de **B** et doit utiliser ARP pour obtenir cette adresse. Voici les éléments de cette “ question ARP ” :

```
SENDER HA    00:06:5b:0f:5a:20
SENDER ADR    192.168.20.249
TARGET HA     ff:ff:ff:ff:ff:ff
TARGET ADR    192.168.20.69
```

**Étape 5** Et la “ réponse ARP ” :

```
SENDER HA    00:01:e6:a1:07:64
SENDER ADR    192.168.20.69
TARGET HA     00:06:5b:0f:5a:20
TARGET ADR    192.168.20.249
```

**Étape 6** Enfin, dans cette dernière étape, **R** envoie le datagramme en provenance de **A**, à **B** :

|            |                   |
|------------|-------------------|
| IP SOURCE  | 192.168.10.109    |
| IP TARGET  | 192.168.20.69     |
| MAC SOURCE | 00:06:5b:0f:5a:20 |
| MAC TARGET | 00:01:e6:a1:07:64 |

Remarque, comparons avec le datagramme de l'étape 3. Si les adresses IP n'ont pas changé, les adresses MAC, diffèrent complètement !

Remarque : Si **A** envoie un deuxième datagramme, les caches ARP ont les adresses MAC utiles et donc les étape 1, 2, 4 et 5 deviennent inutiles...

## 8 Conclusion sur IP

Après notre tour d'horizon sur IPv4 nous pouvons dire en conclusion que son espace d'adressage trop limité n'est pas la seule raison qui a motivé les travaux de recherche et développement d'IPv6 :

1. Son en-tête comporte deux problèmes, la somme de contrôle (checksum) doit être calculée à chaque traitement de datagramme, chaque routeur doit analyser le contenu du champ option.
2. Sa configuration nécessite au moins trois informations que sont l'adresse, le masque de sous réseau et la route par défaut.
3. Son absence de sécurité est insupportable. Issu d'un monde fermé où la sécurité n'était pas un problème, le datagramme de base n'offre aucun service de confidentialité, d'intégrité et d'authentification.
4. Son absence de qualité de service ne répond pas aux exigences des protocoles applicatifs modernes (téléphonie, vidéo, jeux interactifs en réseau, ...). Le champ TOS n'est pas suffisant et surtout est interprété de manière inconsistante par les équipements.

## 9 Bibliographie

Pour en savoir plus :

**RFC 791** “ Internet Protocol. ” J. Postel. Sep-01-1981. (Format : TXT=97779 bytes) (Obsoletes RFC0760) (Status : STANDARD)

**RFC 826** “ Ethernet Address Resolution Protocol : Or converting network protocol addresses to 48.bit Ethernet address for transmission on Ethernet hardware. ” D.C. Plummer. Nov-01-1982. (Format : TXT=22026 bytes) (Status : STANDARD)

**RFC 903** “ Reverse Address Resolution Protocol. ” R. Finlayson, T. Mann, J.C. Mogul, M. Theimer. Jun-01-1984. (Format : TXT=9345 bytes) (Status : STANDARD)

**RFC 950** “ Internet Standard Subnetting Procedure. ” J.C. Mogul, J. Postel. Aug-01-1985. (Format : TXT=37985 bytes) (Updates RFC0792) (Status : STANDARD)

**RFC 1112** “ Host extensions for IP multicasting. ” S.E. Deering. Aug-01-1989. (Format : TXT=39904 bytes) (Obsoletes RFC0988, RFC1054) (Updated by RFC2236) (Also STD0005) (Status : STANDARD)

**RFC 1256** “ ICMP Router Discovery Messages. S. Deering. ” Sep-01-1991. (Format : TXT=43059 bytes) (Also RFC0792) (Status : PROPOSED STANDARD)

- W. Richard Stevens - TCP/IP Illustrated, Volume 1 - The protocols - Addison-Wesley
- Douglas Comer - Internetworking with TCP/IP - Principles, protocols, and architecture - Prentice-Hall
- Craig Hunt - TCP/IP Network Administration - O'Reilly & Associates, Inc.
- Christian Huitema - Le routage dans l'Internet - EYROLLES



# Chapitre V

## Protocole UDP

### 1 UDP – User Datagram Protocol

UDP est l’acronyme de “User Datagram Protocol”, il est défini dans la RFC 768 [Postel 1980]. Les données encapsulées dans un en-tête UDP sont des “paquets UDP”.

#### 1.1 Identification de la destination

*Rappel* : Au niveau de la couche *Internet* les datagrammes sont routés d’une machine à une autre en fonction des bits de l’adresse IP qui identifient le numéro de réseau. Lors de cette opération aucune distinction n’est faite entre les services ou les utilisateurs qui émettent ou reçoivent des datagrammes, ie tous les datagrammes sont mélangés.

La couche UDP ajoute un mécanisme qui permet l’identification du service (niveau *Application*). En effet, il est indispensable de faire un tri entre les divers applications (services) : plusieurs programmes de plusieurs utilisateurs peuvent utiliser simultanément la même couche de transport et il ne doit pas y avoir de confusion entre eux.

Pour le système **Unix** les programmes sont identifiés de manière unique par un numéro de processus, mais ce numéro est éphémère, non prévisible à distance, il ne peut servir à cette fonction.

L’idée est d’associer la destination à la fonction qu’elle remplit. Cette identification se fait à l’aide d’un **entier positif** que l’on baptise **port**.

- Le système d’exploitation local a à sa charge de définir le mécanisme qui permet à un processus d’accéder à un port.
- La plupart des systèmes d’exploitation fournissent le moyen d’un accès synchrone à un port. Ce logiciel doit alors assurer la possibilité de gérer la file d’attente des paquets qui arrivent, jusqu’à ce qu’un processus (*Application*) les lise. A l’inverse, l’OS, “Operating System”, bloque un processus qui tente de lire une donnée non encore disponible.

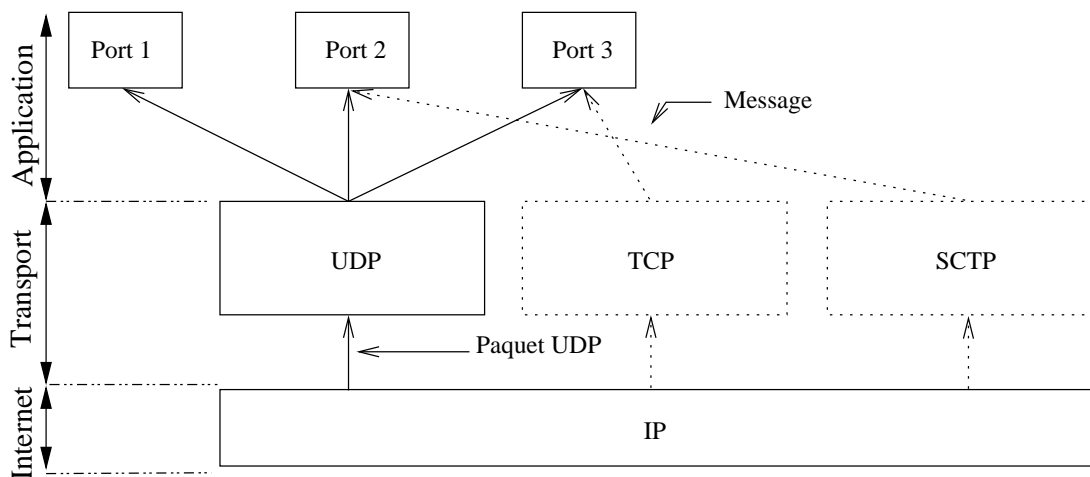
Pour communiquer avec un service distant il faut donc avoir connaissance

de son numéro de port, en plus de l'adresse IP de la machine elle-même.

On peut prévoir le numéro de port en fonction du service à atteindre, c'est l'objet du paragraphe 1.3.

La *figure V.01* explicite la notion de port. La couche IP sépare les datagrammes SCTP, TCP et UDP grâce au champ `PROTO`<sup>1</sup> de son en-tête, l'association du protocole de transport et du numéro de port identifie un service sans ambiguïté.

Conceptuellement on s'aperçoit alors que rien ne s'oppose à ce qu'un même service (Numéro de port) soit attribué conjointement aux trois protocoles (en pointillés sur la figure). Cette situation est d'ailleurs courante dans la réalité des serveurs.



*figure V.01 — Numéro de port comme numéro de service*

<sup>1</sup> Cf description page 47



## 1.2 Description de l'en-tête

Un paquet UDP est conçu pour être encapsulé dans un datagramme IP et permettre un échange de données entre deux applications, sans échange préliminaire. Ainsi, si les données à transmettre n'obligent pas IP à fragmenter (cf page 52), un paquet UDP génère un datagramme IP et c'est tout !

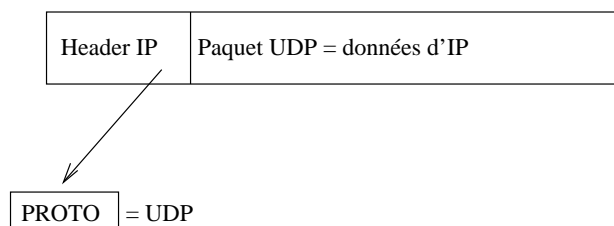


figure V.02 — UDP encapsulé dans IP

- UDP apporte un mécanisme de gestion des ports, au dessus de la couche Internet.
- UDP est simplement une interface au dessus d'IP, ainsi l'émission des messages se fait-elle sans garantie de bon acheminement. Plus généralement, tous les défauts d'IP recensés au chapitre précédent sont applicables à UDP.

Plus particulièrement, les paquets à destination d'une application UDP sont conservés dans une pile de type FIFO. Si l'application destinatrice ne les "consomme" pas assez rapidement, les plus anciens paquets risquent d'être écrasés par les plus récents... Un risque supplémentaire (par rapport aux propriétés d'IP déjà connues) de perte de données.

- Il n'y a aucun retour d'information au niveau du protocole pour apporter un quelconque moyen de contrôle sur le bon acheminement des données.

C'est au niveau applicatif qu'il convient de prendre en compte cette lacune.

- UDP est aussi désigné comme un mode de transport "non connecté", ou encore mode datagramme, par opposition à TCP ou SCTP que nous examinerons dans les prochains chapitres.

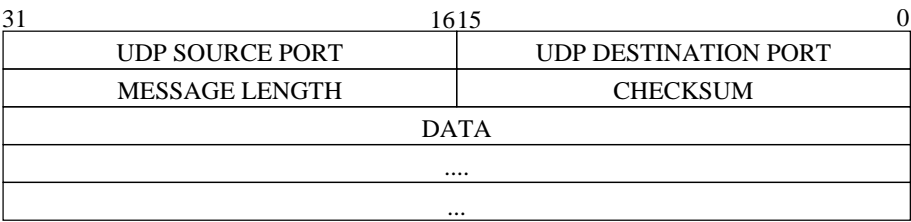
Parmi les utilisations les plus courantes d'UDP on peut signaler le serveur de noms<sup>2</sup>, base de données répartie au niveau mondial, et qui s'accommode très bien de ce mode de transport.

En local d'autres applications très utiles comme `tftp` ou `nfs` sont également susceptibles d'employer UDP.

---

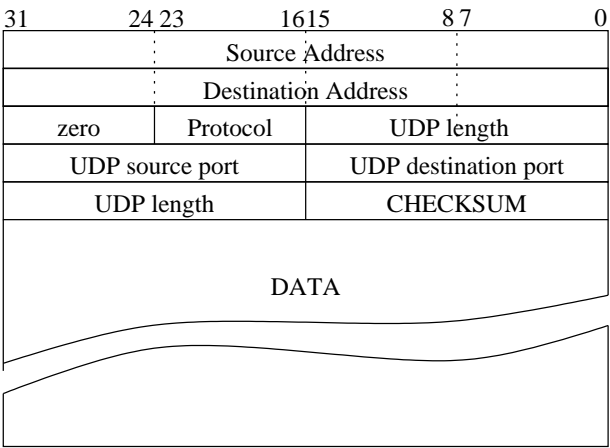
<sup>2</sup>DNS — RFC 1035— Ce service utilise UDP dans le cas d'échanges de petits paquets d'informations ( $\leq 512$  octets) sinon il utilise TCP

La *figure V.03* décrit la structure de l'en-tête.



*figure V.03 — Structure de l'en-tête UDP*

- UDP SOURCE PORT Le numéro de port de l'émetteur du paquet. Ce champ est optionnel, quand il est spécifié il indique le numéro de port que le destinataire doit employer pour sa réponse. La valeur zéro (0) indique qu'il est inutilisé, le port 0 n'est donc pas celui d'un service valide.
- UDP DESTINATION PORT Le numéro de port du destinataire du paquet.
- MESSAGE LENGTH C'est la longueur du paquet, donc comprenant l'en-tête et le message.
- La longueur minimal est 8
  - La longueur maximale est  $65\,535 - H(IP)$ . Dans le cas courant (IP sans option) cette taille maximale est donc de 65 515.
- CHECKSUM Le checksum est optionnel et toutes les implémentations ne l'utilisent pas. S'il est employé, il porte sur un pseudo en-tête constitué de la manière suivante :



*figure V.04 — Cas du checksum non nul*

Ce pseudo en-tête est prévu initialement pour apporter une protection en cas de datagrammes mal routés !

### 1.3 Ports réservés — ports disponibles

Le numéro de port est un entier 16 bits non signé, les bornes sont donc  $[0, 65535]$ , par construction. Nous avons vu précédemment que le port 0 n'est pas exploitable en tant que désignation de service valide, donc le segment réellement exploitable est  $[1, 65535]$ .

Toute machine qui utilise la pile TCP/IP se doit de connaître un certain nombre de services bien connus, repérés par une série de ports bien connus ou “well known port numbers”, pour pouvoir dialoguer avec les autres machines de l'Internet (vs Intranet). Sur une machine Unix, cette liste de services est placée dans le fichier `/etc/services` et lisible par tous les utilisateurs et toutes les applications.

En effet, comme nous l'examinerons en détail dans le cours de programmation, un service (comprendre un programme au niveau applicatif) qui démarre son activité réseau (et qui donc est considéré comme ayant un rôle de serveur) s'attribue le (les) numéro(s) de port qui lui revient (reviennent) conformément à cette table.

| Nom      | Port | Proto | Commentaire                       |
|----------|------|-------|-----------------------------------|
| echo     | 7    | tcp   |                                   |
| echo     | 7    | udp   |                                   |
| ftp-data | 20   | tcp   | #File Transfer [Default Data]     |
| ftp-data | 20   | udp   | #File Transfer [Default Data]     |
| ftp      | 21   | tcp   | #File Transfer [Control]          |
| ftp      | 21   | udp   | #File Transfer [Control]          |
| ssh      | 22   | tcp   | #Secure Shell Login               |
| ssh      | 22   | udp   | #Secure Shell Login               |
| smtp     | 25   | tcp   | mail #Simple Mail Transfer        |
| smtp     | 25   | udp   | mail #Simple Mail Transfer        |
| domain   | 53   | tcp   | #Domain Name Server               |
| domain   | 53   | udp   | #Domain Name Server               |
| http     | 80   | tcp   | www www-http #World Wide Web HTTP |
| http     | 80   | udp   | www www-http #World Wide Web HTTP |
| pop3     | 110  | tcp   | #Post Office Protocol - Version 3 |
| pop3     | 110  | udp   | #Post Office Protocol - Version 3 |
| imap     | 143  | tcp   | #Interim Mail Access Protocol     |
| imap     | 143  | udp   | #Interim Mail Access Protocol     |
| https    | 443  | tcp   | #Secure World Wide Web HTTP       |
| https    | 443  | udp   | #Secure World Wide Web HTTP       |

Le tableau de la *figure V.05* présente quelques uns des ports bien connus plus connus les plus utilisés, il y en a quantité d'autres...

Une autorité, l'IANA<sup>3</sup>, centralise et diffuse l'information relative à tous

<sup>3</sup>“Internet Assigned Numbers Authority”

les nombres utilisés sur l'Internet via une RFC. La dernière en date est la RFC 1700, elle fait plus de 200 pages !

Par voie de conséquence cette RFC concerne aussi les numéros de ports.

### 1.3.1 Attribution des ports “ancienne méthode”

Historiquement les ports de 1 à 255 sont réservés aux services bien connus, plus récemment, ce segment a été élargi à [1, 1023]. Aucune application ne peut s'attribuer durablement et au niveau de l'Internet un numéro de port dans ce segment, sans en référer à l'IANA, qui en contrôle l'usage.

À partir de 1024 et jusqu'à 65535, l'IANA se contente d'enregistrer les demandes d'usage et signale les éventuels conflits.

### 1.3.2 Attribution des ports “nouvelle méthode”

Devant l'explosion du nombre des services enregistrés l'IANA a modifié la segmentation<sup>4</sup> qui précède. Désormais les numéros de ports sont classés selon les trois catégories suivantes :

1. **Le segment** [1, 1023] est toujours réservés aux services bien connus.  
Les services bien connus sont désignés par l'IANA et sont mis en œuvre par des applications qui s'exécutent avec des droits privilégiés (**root** sur une machine **Unix**)
2. **Le segment** [1024, 49151] est celui des services enregistrés.  
Ils sont énumérés par l'IANA et peuvent être employés par des processus ayant des droits ordinaires.

Par exemple :

| Nom  | Port  | Proto | Commentaire       |
|------|-------|-------|-------------------|
| bpcd | 13782 | tcp   | VERITAS NetBackup |
| bpcd | 13782 | udp   | VERITAS NetBackup |

3. **Le segment** [49152, 65535] est celui des attributions dynamiques et des services privés ; nous en examinerons l'usage dans le cours de programmation.

---

<sup>4</sup><http://www.iana.org/assignments/port-numbers>

## 2 Bibliographie

Pour en savoir plus :

**RFC 768** “User Datagram Protocol.” J. Postel. Aug-28-1980. (Format : TXT=5896 bytes) (Status : STANDARD)

**RFC 1035** “Domain names - concepts and facilities.” P.V. Mockapetris. Nov-01-1987. (Format : TXT=129180 bytes) (Obsoletes RFC0973, RFC0882, RFC0883) (Obsoleted by RFC1065, RFC2065) (Updated by RFC1101, RFC1183, RFC1348, RFC1876, RFC1982, RFC2065, RFC2181) (Status : STANDARD)

**RFC 1700** “ASSIGNED NUMBERS.” J. Reynolds, J. Postel. October 1994. (Format : TXT=458860 bytes) (Obsoletes RFC1340) (Also STD0002) (Status : STANDARD)

**RFC 1918** “Address Allocation for Private Internets.” Y. Rekhter, B. Moskowitz, D. Karrenberg, G. J. de Groot & E. Lear. February 1996. (Format : TXT=22270 bytes) (Obsoletes RFC1627, RFC1597) (Also BCP0005) (Status : BEST CURRENT PRACTICE)

Sans oublier :

- W. Richard Stevens - TCP/IP Illustrated, Volume 1 - The protocols - Addison-Wesley — 1994
- Douglas Comer - Internetworking with TCP/IP - Principles, protocols, and architecture - Prentice-Hall

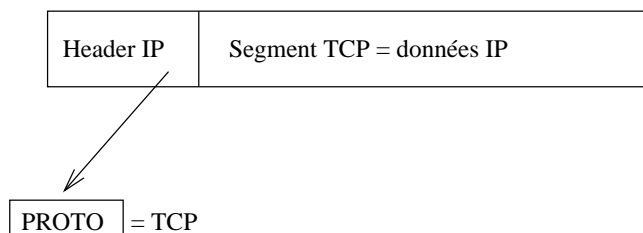


# Chapitre VI

## Protocole TCP

### 1 TCP – Transmission Control Protocol

TCP est l'acronyme de “ **T**ransmission **C**ontrol **P**rotocol ”, il est défini dans la RFC 793 [Postel 1981c]. Les données encapsulées dans un en-tête TCP sont des “ paquets TCP ”.



*figure VI.01 — TCP encapsulé dans IP*

#### 1.1 Caractéristiques de TCP

TCP est bien plus compliqué<sup>1</sup> qu’UDP examiné au chapitre précédent, il apporte en contrepartie des services beaucoup plus élaborés.

Cinq points principaux caractérisent ce protocole :

1. TCP contient un mécanisme pour assurer le **bon acheminement des données**. Cette possibilité est absolument indispensable dès lors que les applications doivent transmettre de gros volumes de données et de façon fiable.  
Il faut préciser que les paquets de données sont acquittés de bout en bout et non de point en point. D’une manière générale le réseau assure l’acheminement et les extrémités le contrôle (Dave Clark).
2. Le protocole TCP permet l’établissement d’un **circuit virtuel** entre les deux points qui échangent de l’information. On dit aussi que TCP

---

<sup>1</sup>Une simple comparaison du volume des RFC initiales est parlante : 85 pages pour TCP, 3 pour UDP !

fonctionne en mode connecté (par opposition à UDP qui est en mode non connecté ou encore mode datagramme).

- Avant le transfert les 2 applications se mettent en relation avec leurs OS<sup>2</sup> respectifs, les informant de leurs désirs d'établir ou de recevoir une communication.
- Pratiquement, l'une des deux applications doit effectuer un appel que l'autre doit accepter.
- Les protocoles des 2 OS communiquent alors en s'envoyant des messages au travers du réseau pour vérifier que le transfert est possible (autorisé) et que les deux applications sont prêtes pour leurs rôles.
- Une fois ces préliminaires établis, les modules de protocole informent les applications respectives que la connexion est établie et que le transfert peut débuter.
- Durant le transfert, le dialogue entre les protocoles continue, pour vérifier le bon acheminement des données.

Conceptuellement, pour établir une connexion — un circuit virtuel — il faut avoir réunis les éléments du quintuplet :

**Le protocole** C'est TCP mais il y pourrait y avoir d'autres transports qui assurent le même service. . .

**IP locale** Adresse de la machine qui émet.

**Port local** Le numéro de port associé au processus. Il est imposé ou est déterminé automatiquement comme nous le verrons dans le cours de programmation.

**IP distante** Adresse de la machine distante.

**Port distant** Le numéro de port associé au service à atteindre. Il est obligatoire de le connaître précisément.

L'ensemble de ces cinq éléments définit un circuit virtuel unique. Que l'un d'eux change et il s'agit d'une autre connexion !

### 3. TCP a la capacité de **mémoriser<sup>3</sup> des données** :

- Aux deux extrémités du circuit virtuel, les applications s'envoient des volumes de données absolument quelconques, allant de 0 octet à des centaines (ou plus) de Mo.
- À la réception, le protocole délivre les octets exactement comme ils ont été envoyés.
- Le protocole est libre de fragmenter le flux de données en paquets de tailles adaptées aux réseaux traversés. Il lui incombe cependant d'effectuer le réassemblage et donc de stocker temporairement les fragments avant de les présenter dans le bon ordre à l'application.

### 4. TCP est **indépendant vis à vis des données transportées**, c'est un flux d'octets non structuré sur lequel il n'agit pas.

---

<sup>2</sup> " Operating System "

<sup>3</sup> dans un bu er



5. TCP simule une connexion en “ **full duplex** ”. Pour chacune des deux applications en connexion par un circuit virtuel, l'opération qui consiste à lire des données peut s'effectuer indépendamment de celle qui consiste à en écrire.

Le protocole autorise la clôture du flot dans une direction tandis que l'autre continue à être active. Le circuit virtuel est rompu quand les deux parties ont clos le flux.

## 1.2 Description de l'en-tête

La figure suivante montre la structure d'un en-tête TCP. Sa taille normale est de 20 octets, à moins que des options soient présentes.

|                        |  |          |  |                      |                |        |         |   |  |
|------------------------|--|----------|--|----------------------|----------------|--------|---------|---|--|
| 31                     |  | 19       |  | 16 15                |                | 8 7    |         | 0 |  |
| TCP SOURCE PORT        |  |          |  | TCP DESTINATION PORT |                |        |         |   |  |
| SEQUENCE NUMBER        |  |          |  |                      |                |        |         |   |  |
| ACKNOWLEDGEMENT NUMBER |  |          |  |                      |                |        |         |   |  |
| OFF                    |  | RESERVED |  | CODE                 |                | WINDOW |         |   |  |
| CHECKSUM               |  |          |  |                      | URGENT POINTER |        |         |   |  |
| OPTIONS                |  |          |  |                      |                |        | PADDING |   |  |
| DATA                   |  |          |  |                      |                |        |         |   |  |
| ...                    |  |          |  |                      |                |        |         |   |  |

figure VI.02 — Structure de l'en-tête TCP

**TCP SOURCE PORT** Le numéro de port de l'application locale.

**TCP DESTINATION PORT** Le numéro de port de l'application distante.

**SEQUENCE NUMBER** C'est un nombre qui identifie la position des données à **transmettre** par rapport au segment original. Au démarrage de chaque connexion, ce champ contient une valeur non nulle et non facilement prévisible, c'est la séquence initiale ou ISN<sup>4</sup>

TCP numérote chaque octet transmis en incrémentant ce nombre 32 bits non signé. Il repasse à 0 après avoir atteint  $2^{32} - 1$  (4 294 967 295).

Pour le premier octet des données transmis ce nombre est incrémenté de un, et ainsi de suite...

**ACKNOWLEDGEMENT NUMBER** C'est un numéro qui identifie la position du **dernier octet reçu** dans le flux entrant.

Il doit s'accompagner du drapeau **ACK** (voir plus loin).

**OFF** pour **OFFSET**, il s'agit d'un déplacement qui permet d'atteindre les données quand il y a des options. Codé sur 4 bits, il s'agit du nombre de mots de 4 octets qui composent l'en-tête. Le déplacement maximum est donc de 60 octets ( $2^4 - 1 \times 4$  octets). Dans le cas d'un en-tête sans option, ce champ porte la valeur 5. 10 mots de 4 octets sont donc possibles pour les options.

<sup>4</sup> " Initial Sequence Number "

**RESERVED** Six bits réservés pour un usage futur !

**CODE** Six bits pour influencer sur le comportement de TCP en caractérisant l'usage du segment :

- URG** Le champ “ **URGENT POINTER** ” doit être exploité.
- ACK** Le champ “ **ACKNOWLEDGMENT NUMBER** ” doit être exploité.
- PSH** C'est une notification de l'émetteur au récepteur, pour lui indiquer que toutes les données collectées doivent être transmises à l'application sans attendre les éventuelles données qui suivent.
- RST** Re-initialisation de la connexion
- SYN** Le champ “ **SEQUENCE NUMBER** ” contient la valeur de début de connexion.
- FIN** L'émetteur du segment a fini d'émettre.

En fonctionnement normal un seul bit est activé à la fois mais ce n'est pas une obligation. La RFC 1024 [Postel 1987] décrit l'existence de paquets tcp dénommés “ Christmas tree ” ou “ paquet kamikaze ” comprenant les bits SYN+URG+PSH+FIN !

**WINDOW** Le flux TCP est contrôlé de part et d'autre pour les octets compris dans une zone bien délimitée et nommée “ fenêtre ”. La taille de celle-ci est définie par un entier non signé de 16 bits, qui en limite donc théoriquement la taille à 65 535 octets (ce n'est pas complètement exact, voir plus loin l'option **wscale**).

Chaque partie annonce ainsi la taille de son buffer de réception. Par construction, l'émetteur n'envoie pas plus de données que le récepteur ne peut en accepter.

Cette valeur varie en fonction de la nature du réseau et surtout de la bande passante devinée à l'aide de statistiques sur la valeur du RTT. Nous y reviendrons au paragraphe 4.

**CHECKSUM** Un calcul qui porte sur la totalité du segment, en-tête et données.

**URGENT POINTER** Ce champ n'est valide que si le drapeau **URG** est armé. Ce pointeur contient alors un offset à ajouter à la valeur de **SEQUENCE NUMBER** du segment en cours pour délimiter la zone des données urgentes à transmettre à l'application.

Le mécanisme de transmission à l'application dépend du système d'exploitation.

**OPTIONS** C'est un paramétrage de TCP. Sa présence est détectée dès lors que l'**OFFSET** est supérieur à 5.

Les options utilisées :

**mss** La taille maximale du segment<sup>5</sup> des données applicatives que l'émetteur accepte de recevoir. Au moment de l'établissement d'une connexion (paquet comportant le flag SYN), chaque partie annonce sa taille de MSS. Ce n'est pas une négociation. Pour de l'Ethernet la valeur est 1460 (  $= MTU - 2 \times 20$  ).

**timestamp** pour calculer la durée d'un aller et retour (RTT ou " round trip time ").

**wscale** Facteur d'échelle ( " shift " ) pour augmenter la taille de la fenêtre au delà des 16 bits du champ WINDOW ( $> 65535$  ).

Quand cette valeur n'est pas nulle, la taille de la fenêtre est de  $65535 \times 2^{shift}$ . Par exemple si " shift " vaut 1 la taille de la fenêtre est de 131072 octets soit encore 128 ko.

**nop** Les options utilisent un nombre quelconque d'octets par contre les paquet TCP sont toujours alignés sur une taille de mot de quatre octets ; à cet effet une option " No Operation " ou **nop**, codée sur 1 seul octet, est prévue pour compléter les mots.

**PADDING** Remplissage pour se caler sur un mot de 32 bits.

**DATAS** Les données transportées. Cette partie est de longueur nulle à l'établissement de la connexion, elle peut également être nulle par choix de l'application.

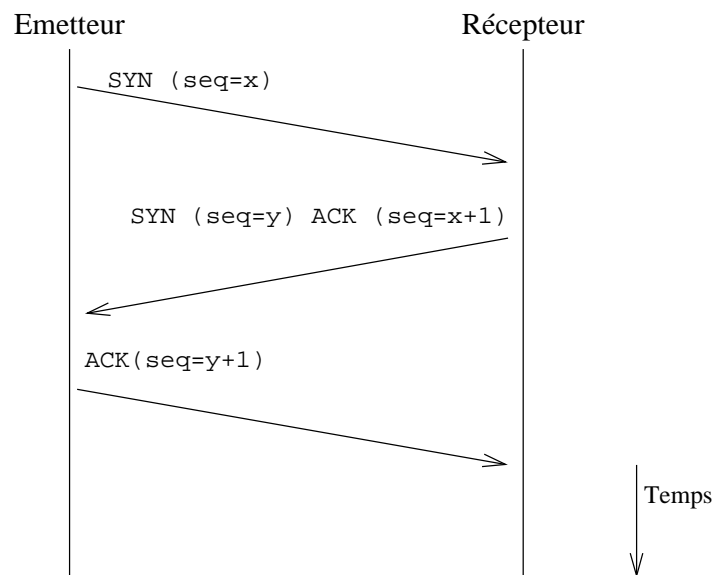
---

<sup>5</sup> MSS = " Maximum Segment Size "

## 2 Début et clôture d'une connexion

### 2.1 Établissement d'une connexion

L'établissement d'une connexion TCP s'effectue en trois temps, comme le schéma de la *figure 3* l'explique.



*figure VI.03 — Établissement d'une connexion*

On suppose que l'émetteur du premier paquet avec le bit **SYN** a connaissance du couple (adresse IP du récepteur, numéro de port du service souhaité).

L'émetteur du premier paquet est à l'origine de l'établissement du circuit virtuel, c'est une attitude généralement qualifiée de “ cliente ”. On dit aussi que le client effectue une “ ouverture active ” (*active open*).

Le récepteur du premier paquet accepte l'établissement de la connexion, ce qui suppose qu'il était prêt à le faire **avant** que la partie cliente en prenne l'initiative. C'est une attitude de “ serveur ”. On dit aussi que le serveur effectue une “ ouverture passive ” (*passive open*).

1. Le client envoie un segment comportant le drapeau **SYN**, avec sa séquence initiale ( $ISN = x$ ).
2. Le serveur répond avec sa propre séquence ( $ISN = y$ ), mais il doit également acquitter le paquet précédent, ce qu'il fait avec **ACK** ( $seq = x + 1$ ).
3. Le client doit acquitter le deuxième segment avec **ACK** ( $seq = y + 1$ ).

Une fois achevée cette phase nommée “ three-way handshake ”, les deux applications sont en mesure d'échanger les octets qui justifient l'établissement de la connexion.

## 2.2 Clôture d'une connexion

### 2.2.1 Clôture canonique

Un échange de trois segments est nécessaire pour l'établissement de la connexion ; il en faut quatre pour qu'elle s'achève de manière canonique ( “ orderly release ”).

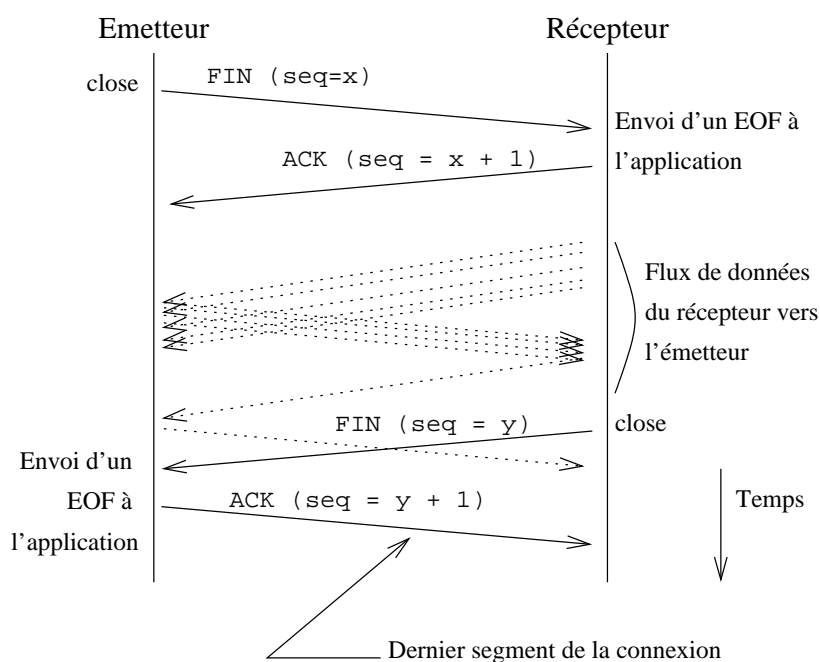


figure VI.04 — Clôture d'une connexion

La raison est qu'une connexion TCP est “ full-duplex ”, ce qui implique que les données circulent indépendamment dans un sens et dans l'autre. Les deux directions doivent donc pouvoir être interrompues indépendamment l'une de l'autre.

L'application qui envoie un paquet avec le drapeau `FIN` indique à la couche TCP de la machine distante qu'elle n'enverra plus de donnée. La machine distante doit acquitter ce segment, comme il est indiqué sur la figure VI.04, en incrémentant d'une unité le “ sequence number ”.

La connexion est véritablement terminée quand les deux applications ont effectué ce travail. Il y a donc échange de 4 paquets pour terminer la connexion.

Au total, sans compter les échanges propres au transfert des données, les deux couches TCP doivent gérer 7 paquets, il faut en tenir compte lors de la conception des applications !

Sur la figure on constate que le serveur continue d'envoyer des données bien que le client ait terminé ses envois. Le serveur a détecté cette attitude par la réception d'un caractère de EOF (en C sous Unix).

Cette possibilité a son utilité, notamment dans le cas des traitements distants qui doivent s'accomplir une fois toutes les données transmises, comme par exemple pour un tri.

### 2.2.2 Clôture abrupte

Au lieu d'un échange de quatre paquets comme précédemment, un mécanisme de reset est prévu pour terminer une connexion au plus vite (*abortive release*).

Ce type d'arrêt est typiquement géré par la couche TCP elle-même quand l'application est brutalement interrompue sans avoir effectué un appel à la primitive `close(2)`, comme par exemple lors d'un appel à la primitive `abort(2)`, ou après avoir rencontré une exception non prise en compte ("core dump"...).

L'extrémité qui arrête brutalement la connexion émet un paquet assorti du bit **RST**, après avoir (ou non) envoyé les derniers octets en attente<sup>6</sup>. Ce paquet clôt l'échange. Il ne reçoit aucun acquittement.

L'extrémité qui reçoit le paquet de reset (bit **RST**), transmet les éventuelles dernières données à l'application et provoque une sortie d'erreur du type "Connection reset par peer" pour la primitive de lecture réseau. Comme c'est le dernier échange, si des données restaient à transmettre à l'application qui a envoyé le **RST** elles peuvent être détruites.

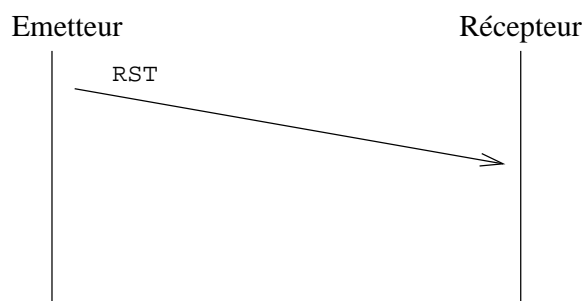


figure VI.05 — Émission d'un rst

<sup>6</sup>Voir dans le cours de programmation, l'option `SO_LINGER`

## 3 Contrôle du transport

Le bon acheminement des données applicatives est assuré par un mécanisme d'acquittement des paquets, comme nous avons déjà pu l'examiner partiellement au paragraphe précédent.

### 3.1 Mécanisme de l'acquittement

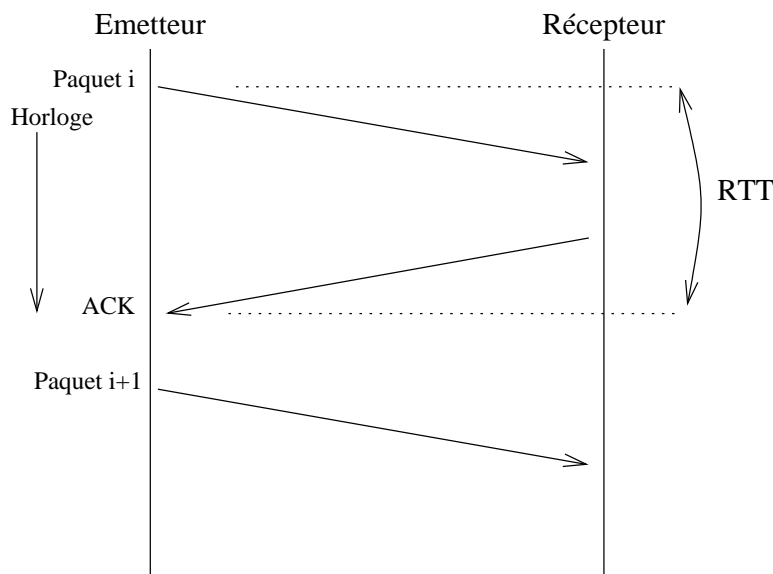


figure VI.06 — Mécanisme de l'acquittement

- Au départ du *Paquet i* une horloge se déclenche. Si cette horloge dépasse une valeur limite avant réception de l'ACK le *Paquet i* est retransmis. Cette valeur limite est basée sur la constante  $MSL$ <sup>7</sup> qui est un choix d'implémentation, généralement de 30 secondes à 2 minutes. Le temps maximum d'attente est donc de  $2 \times MSL$ .
- Le temps qui s'écoule entre l'émission d'un paquet et la réception de son acquittement est le  $RTT$ <sup>8</sup>, il doit donc être inférieur à  $2 \times MSL$ . Il est courant sur l'Internet actuel d'avoir un  $RTT$  de l'ordre de la seconde. Il faut noter que le  $RTT$  est la somme des temps de transit entre chaque routeur et du temps passé dans les diverses files d'attente sur les routeurs.
- L'émetteur conserve la trace du *Paquet i* pour éventuellement le renvoyer.

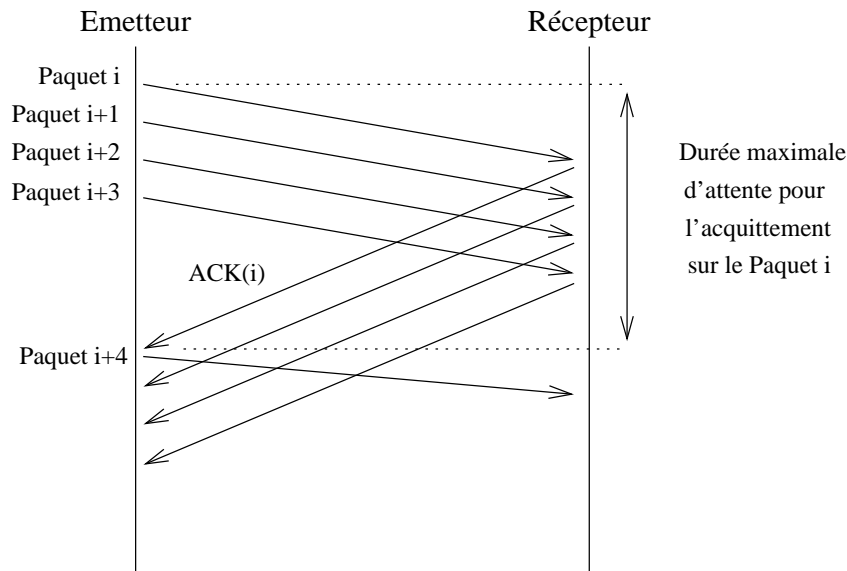
Si on considère des délais de transmission de l'ordre de 500 ms (voire plus), un tel mécanisme est totalement inadapté au transfert de flux de données. On peut aussi remarquer qu'il sous-emploie la bande passante du réseau.

<sup>7</sup> " Maximum Segment Lifetime "

<sup>8</sup> " Round Trip Time ", calculé à l'aide de l'option " timestamp " - voir page 93

### 3.2 Fenêtres glissantes

Cette attente de l'acquittement est pénalisante, sauf si on utilise un mécanisme de “fenêtres glissantes<sup>9</sup>”, comme le suggère la *figure VI.07* :



*figure VI.07 — Principe de la fenêtre glissante*

- Avec ce principe, la bande passante du réseau est beaucoup mieux employée.
- Si l'un des paquets doit être réémis, la couche TCP du destinataire aura toute l'information pour le replacer dans le bon ordre.
- À chaque paquet est associée une horloge comme sur la *figure VI.06*.
- Le nombre de paquets à envoyer avant d'attendre le premier acquittement est fonction de deux paramètres :
  1. La largeur de la fenêtre précisée dans le champ `WINDOW` de l'en-tête. Des valeurs courantes sont de l'ordre de 4096, 8192 ou 16384. Elle change dynamiquement pour deux raisons :
    - (a) L'application change la taille du “buffer de la socket”<sup>10</sup> qui correspond à la taille de cette fenêtre.
    - (b) Chaque acquittement `ACK` envoyé est assorti d'une nouvelle valeur de taille de la fenêtre, permettant ainsi à l'émetteur d'ajuster à tout instant le nombre de segment qu'il peut envoyer simultanément. Cette valeur peut être nulle, comme par exemple lorsque l'application cesse de lire les données reçues. C'est ce mécanisme qui assure le contrôle de flux de TCP.

<sup>9</sup> “sliding windows”

<sup>10</sup>voir le cours de programmation



2. La taille maximale des données, ou MSS<sup>11</sup> vaut 512 octets par défaut. C'est la plus grande taille du segment de données que TCP enverra au cours de la session.

Le datagramme IP a donc une taille égale au MSS augmentée de 40 octets ( $20 + 20$ ), en l'absence d'option de TCP.

Cette option apparaît uniquement dans un paquet assorti du drapeau SYN, donc à l'établissement de la connexion.

Comme de bien entendu cette valeur est fortement dépendante du support physique et plus particulièrement du MTU<sup>12</sup>.

Sur de l'Ethernet la valeur maximale est  $1500 - 2 \times 20 = 1460$ , avec des trames l'encapsulation 802.3 de l'IEEE un calcul similaire conduit à une longueur de 1452 octets.

Chaque couche TCP envoie sa valeur de MSS en même temps que le paquet de synchronisation, comme une option de l'en-tête. Cette valeur est calculée pour éviter absolument la fragmentation de IP au départ des datagrammes.

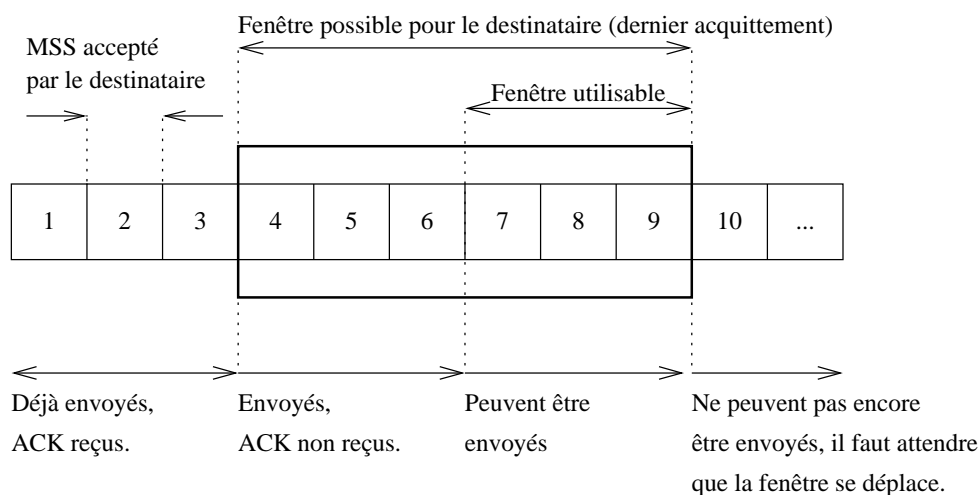


figure VI.08 — Détail de la fenêtre glissante

Le débit obtenu dépend de la taille de la fenêtre et bien sûr de la bande passante disponible. On conçoit aisément qu'entre la situation de la figure VI.06 et celle de la figure VI.07 l'usage de la bande passante s'améliore. Par contre l'agrandissement de la taille de la fenêtre ne se conçoit que jusqu'à une limite optimale au delà de laquelle des paquets sont perdus parce qu'envoyés trop rapidement pour être reçus par le destinataire. Or, pour fonctionner de manière optimale, TCP se doit de limiter au maximum la perte de paquets et donc leur réémission.

<sup>11</sup> " Maximum Segment Size " - Cf page 93

<sup>12</sup> " Maximum Transfer Unit " - Cf page 52

Cette taille limite optimale de la largeur de la fenêtre est, comme on peut le deviner, fonction de la bande passante théorique du réseau et surtout de son taux d'occupation instantané. Cette dernière donnée est fluctuante, aussi TCP doit-il asservir continuellement les tailles de fenêtre pour en tenir compte.

## 4 Compléments sur le fonctionnement de TCP

L'usage du protocole TCP diffère considérablement en fonction des applications mises en œuvre et des réseaux à parcourir.

D'après [W. Richard Stevens], 10% des données échangées sur le réseau concernent des applications interactives et 90% des applications qui échangent des flux de données.

Si le protocole TCP reste le même pour tous, les algorithmes qui le pilotent s'ajustent en fonction de l'usage.

Pour le trafic en volume ( “ bulk data ” ), TCP tente d'utiliser des paquets les plus larges possibles pour maximiser le débit, alors que le trafic interactif utilise des paquets quasiment vides émis le plus souvent à la fréquence de frappe des utilisateurs ou au rythme des mouvements d'une souris.

Un exemple typique est celui de l'application `telnet` pour laquelle les caractères sont envoyés un à un dans un paquet différent, chaque caractère étant à l'origine de quatre paquets : émission d'un caractère, acquittement, retour de l'écho du caractère, acquittement.

Si ce comportement n'est absolument pas pénalisant sur un réseau rapide (LAN) par contre dès que la bande passante commence à être saturée il est préférable de regrouper un maximum d'octets (deux ou trois en pratique) dans un seul paquet pour en diminuer le nombre. C'est ce que fait l'algorithme de Nagle.

### 4.1 Algorithme de Nagle

Pour réduire le trafic de ces “ tinygrams ” (RFC 896), l'algorithme de Nagle (1984) dit qu'une connexion TCP ne peut pas attendre plus d'un acquittement. Deux cas se présentent donc :

1. Le réseau est lent. Dans ce cas TCP accumule dans un même buffer les octets en partance. Dès réception de l'acquittement il y a émission du contenu du buffer en un seul paquet.
2. Le réseau est rapide. Les acquittements arrivent rapidement les agrégats d'octets peuvent tendre vers un seul caractère par paquet.

La qualité lent/rapide du réseau est calculée à partir du “ timestamp ” envoyé dans les options de TCP et qui est établi dès le premier échange (puis réévaluée statistiquement par la suite).

L'élégance de cet algorithme est qu'il est très simple et qu'il s'auto-régule suivant les délais de propagation.

Certaines applications désactivent cet algorithme<sup>13</sup> comme le serveur *Apache* ou le système de multi-fenêtrage *X11*.

## 4.2 Départ lent

Un paquet est réémis parcequ'il arrive corrompu ou parcequ'il n'arrive jamais. Une réémission entraîne un blocage de l'avancement de la “fenêtre glissante”, pénalisant pour le débit (cf conclusion du chapitre page 105).

TCP considère qu'un paquet perdu est la conséquence d'un routeur (ou plus) congestionné, c'est à dire pour lequel les files d'attente ne sont pas assez larges pour absorber tous les paquets entrants<sup>14</sup>

Dans ce contexte, on comprend bien qu'il vaut mieux ne pas envoyer la totalité du contenu de la fenêtre dès le début de la connexion. Au contraire, TCP utilise un algorithme nommé “slow start” qui asservit l'émission des paquets au rythme de la réception de leurs acquittements, plutôt que de les émettre d'un coup aussi rapidement que l'autorise le système ou le débit théorique du réseau.

Ainsi, au début de chaque connexion ou après une période de calme (“idle”) l'émetteur envoie un premier paquet de taille maximale (le “mss” du destinataire), et attend son acquittement. Quand celui-ci est reçu, il envoie deux paquets, puis quatre, et ainsi de suite jusqu'à atteindre l'ouverture maximale de la fenêtre.

Durant cette progression, si des paquets sont perdus, il y a congestion supposée sur l'un des routeurs franchis et l'ouverture de la fenêtre est réduite pour retrouver un débit qui minimise le taux de retransmission.

L'ouverture de la fenêtre est nommée fenêtre de congestion ou encore “congestion window”.

## 4.3 Évitement de congestion

Le contrôle du flux évoqué précédemment, pour éviter la congestion des routeurs, est implémenté à l'aide d'une variable (**cwnd**) nommée “congestion window” que l'on traduit par fenêtre de congestion.

Concrètement, le nombre maximum de segments de données ( $\times MSS$  en octets) que l'émetteur peut envoyer avant d'en recevoir le premier acquittement est le minimum entre cette variable (**cwnd**) et la taille de la fenêtre annoncée par le récepteur à chaque acquittement de paquet.

Le contenu de cette variable est piloté par les algorithmes de départ lent — “slow start”, voir 4.2 — et d'évitement de congestion (“congestion avoidance”) examiné ici.

La limite de croissance de la variable **cwnd** est la taille de la fenêtre annoncée par le récepteur. Une fois la capacité de débit maximale atteinte, si un paquet est perdu l'algorithme d'évitement de congestion en diminue

<sup>13</sup>à l'aide de l'option TCP\_NODELAY

<sup>14</sup>Ce cas arrive fréquemment quand un routeur sépare un réseau rapide d'un réseau lent

linéairement la valeur (contrairement au “slow start” qui l’augmente exponentiellement).

## 5 Paquets capturés, commentés

Le premier exemple montre un échange de paquets de synchronisation (SYN) et de fin (FIN) entre la machine `clnt.chezmoi` et la machine `srv.chezmoi`. L’établissement de la connexion se fait à l’aide de la commande `telnet` sur le port `discard` (9) du serveur `srv.chezmoi`. La machine qui est à l’origine de l’établissement de la connexion est dite cliente, et celle qui est supposée prête à répondre, serveur. Pour information, le service `discard` peut être considéré comme l’équivalent du fichier `/dev/null` sur le réseau : les octets qu’on lui envoie sont oubliés (“discard”).

L’utilisateur tape :

*Simultanément la capture des paquets est lancée, par exemple dans une autre fenêtre xterm.*

```
$ telnet srv discard
Trying...
Connected to srv.chezmoi.
Escape character is '^]'.

telnet> quit
Connection closed.
```

Et l’outil d’analyse réseau<sup>15</sup> permet la capture pour l’observation des échanges suivants. Le numéro qui figure en tête de chaque ligne a été ajouté manuellement, le nom de domaine “chezmoi” a été retiré, le tout pour faciliter la lecture :

```
0  13:52:30.274009 clnt.1159 > srv.discard: S 55104001:55104001(0)
                                     win 8192 <mss 1460>
1  13:52:30.275114 srv.discard > clnt.1159: S 2072448001:2072448001(0)
                                     ack 55104002 win 4096 <mss 1024>
2  13:52:30.275903 clnt.1159 > srv.discard: . ack 1 win 8192
3  13:52:33.456899 clnt.1159 > srv.discard: F 1:1(0) ack 1 win 8192
4  13:52:33.457559 srv.discard > clnt.1159: . ack 2 win 4096
5  13:52:33.458887 srv.discard > clnt.1159: F 1:1(0) ack 2 win 4096
6  13:52:33.459598 clnt.1159 > srv.discard: . ack 2 win 8192
```

Plusieurs remarques s’imposent :

1. Pour améliorer la lisibilité les numéros de séquences “vrais” ne sont indiqués qu’au premier échange. Les suivants sont relatifs. Ainsi le `ack 1` de la ligne 2 doit être lu 2072448002 (2072448001 + 1).  
À chaque échange la valeur entre parenthèses indique le nombre d’octets échangés.

<sup>15</sup>`tcpdump` que nous aurons l’occasion d’utiliser en TP

2. Les tailles de fenêtre (**win**) et de segment maximum (**mss**) ne sont pas identiques. Le **telnet** du client fonctionne sur HP-UX alors que le serveur **telnetd** fonctionne sur une machine BSD.
3. La symbole **>** qui marque le sens du transfert.
4. Le port source **1159** et le port destination **discard**.
5. Les flags **F** et **S**. L'absence de flag, repéré par un point.

Le deuxième exemple montre une situation de transfert de fichier avec l'outil **ftp**<sup>16</sup>.

Il faut remarquer que l'établissement de la connexion TCP est ici à l'initiative du serveur, ce qui peut laisser le lecteur perplexe... L'explication est simple. En fait le protocole **ftp** fonctionne avec deux connexions TCP, la première, non montrée ici, est établie du client vers le serveur, supposé à l'écoute sur le port 21. Elle sert au client pour assurer le contrôle du transfert. Lorsqu'un transfert de fichier est demandé via cette première connexion, le serveur établit une connexion temporaire vers le client. C'est cette connexion que nous examinons ici. Elle est cloturée dès que le dernier octet demandé est transféré.

Extrait du fichier **/etc/services**, concernant **ftp** :

|                 |        |                               |
|-----------------|--------|-------------------------------|
| <b>ftp-data</b> | 20/tcp | #File Transfer [Default Data] |
| <b>ftp-data</b> | 20/udp | #File Transfer [Default Data] |
| <b>ftp</b>      | 21/tcp | #File Transfer [Control]      |
| <b>ftp</b>      | 21/udp | #File Transfer [Control]      |

Dans cette exemple nous pouvons suivre le fonctionnement du mécanisme des fenêtres glissantes. Les lignes ont été numérotées manuellement et la date associée à chaque paquet supprimée.

```

0  srv.20 > clnt.1158: S 1469312001:1469312001(0)
                                win 4096 <mss 1024> [tos 0x8]
1  clnt.1158 > srv.20: S 53888001:53888001(0) ack 1469312002
                                win 8192 <mss 1460>
2  srv.20 > clnt.1158: . ack 1 win 4096 [tos 0x8]
3  srv.20 > clnt.1158: P 1:1025(1024) ack 1 win 4096 [tos 0x8]
4  clnt.1158 > srv.20: . ack 1025 win 8192
5  srv.20 > clnt.1158: . 1025:2049(1024) ack 1 win 4096 [tos 0x8]
6  srv.20 > clnt.1158: . 2049:3073(1024) ack 1 win 4096 [tos 0x8]
7  clnt.1158 > srv.20: . ack 3073 win 8192
8  srv.20 > clnt.1158: . 3073:4097(1024) ack 1 win 4096 [tos 0x8]
9  srv.20 > clnt.1158: P 4097:5121(1024) ack 1 win 4096 [tos 0x8]
10 srv.20 > clnt.1158: P 5121:6145(1024) ack 1 win 4096 [tos 0x8]
11 clnt.1158 > srv.20: . ack 5121 win 8192
12 srv.20 > clnt.1158: P 6145:7169(1024) ack 1 win 4096 [tos 0x8]
13 srv.20 > clnt.1158: P 7169:8193(1024) ack 1 win 4096 [tos 0x8]
14 clnt.1158 > srv.20: . ack 7169 win 8192

```

<sup>16</sup>du nom du protocole applicatif utilisé : " File Transfer Protocol "

```

15  srv.20 > clnt.1158: P 8193:9217(1024) ack 1 win 4096 [tos 0x8]
16  srv.20 > clnt.1158: P 9217:10241(1024) ack 1 win 4096 [tos 0x8]
17  clnt.1158 > srv.20: . ack 9217 win 8192
18  srv.20 > clnt.1158: P 10241:11265(1024) ack 1 win 4096 [tos 0x8]
19  srv.20 > clnt.1158: P 11265:12289(1024) ack 1 win 4096 [tos 0x8]
20  clnt.1158 > srv.20: . ack 11265 win 8192
... ..
21  srv.20 > clnt.1158: P 1178625:1179649(1024) ack 1 win 4096 [tos 0x8]
22  clnt.1158 > srv.20: . ack 1178625 win 8192
23  srv.20 > clnt.1158: P 1212417:1213441(1024) ack 1 win 4096 [tos 0x8]
24  srv.20 > clnt.1158: P 1213441:1214465(1024) ack 1 win 4096 [tos 0x8]
25  srv.20 > clnt.1158: P 1214465:1215489(1024) ack 1 win 4096 [tos 0x8]
26  clnt.1158 > srv.20: . ack 1213441 win 8192
27  clnt.1158 > srv.20: . ack 1215489 win 8192
28  srv.20 > clnt.1158: P 1215489:1215738(249) ack 1 win 4096 [tos 0x8]
29  srv.20 > clnt.1158: F 1215738:1215738(0) ack 1 win 4096 [tos 0x8]
30  clnt.1158 > srv.20: . ack 1215739 win 8192
31  clnt.1158 > srv.20: F 1:1(0) ack 1215739 win 8192
32  srv.20 > clnt.1158: . ack 2 win 4096 [tos 0x8]

```

Remarques :

1. Le P symbolise le drapeau PSH. La couche TCP qui reçoit un tel paquet est informée qu'elle doit transmettre à l'application toutes les données reçues, y compris celles transmises dans ce paquet.  
Le positionnement de ce drapeau est à l'initiative de la couche TCP émettrice et non à l'application.
2. Le type de service (" Type Of service " `tos 0x8`) est demandé par l'application pour maximiser le débit (consulter le cours IP page 49).

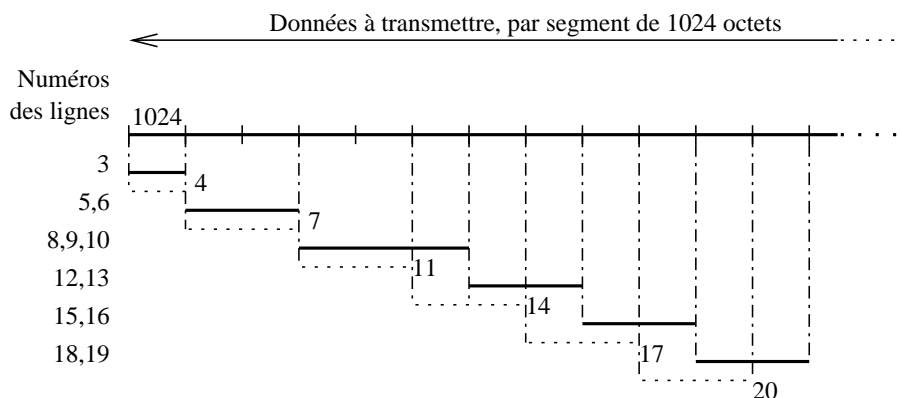


figure VI.09 — Exemple de fenêtre glissante

## 6 Conclusion sur TCP

Le protocole TCP a été conçu à une époque où l'usage de la commande ligne était universel, et les applications graphiques utilisant le réseau très rares. ...!

Une trentaine d'années plus tard, on peut faire le constat pratiquement inverse : les applications textes interactives (beaucoup de petits messages applicatifs) disparaissent au profit d'applications moins interactives et qui sont plus orientées flux de données (vidéo, audio, téléphonie...) avec des échanges plus volumineux et des besoins en transport qui ont évolué.

Le principe de la fenêtre glissante, si performant qu'il soit pour assurer le bon acheminement des données, est bloquant pour certaines applications comme le web. En effet, si le paquet de données de tête n'est pas acquitté, les suivants, même reçus, sont en attente avant d'être délivrés à l'application.

Si la réponse comporte par exemple de nombreuses zones graphiques et textuelles différentes la fluidité de la consultation est considérablement amoindrie, et tenter de la compenser en établissant un grand nombre de connexions simultanées pour récupérer individuellement les éléments de la page, consomme beaucoup de ressources système et réseaux (celles de l'établissement des connexions) qui ne compense que partiellement ce soucis.

L'indépendance de TCP vis à vis de la structure des données est également un inconvénient dans certaines applications comme la téléphonie pour laquelle la notion de messages successifs est bien plus intéressante.

Depuis le début des années 2000 l'IETF met au point le protocole SCTP qui fournit des services similaires à ceux de TCP, en abandonne certains et apporte les nouvelles fonctionnalités adaptées aux nouveaux besoins.

## 7 Bibliographie

**RFC 793** “ Transmission Control Protocol. ” J. Postel. Sep-01-1981. (Format : TXT=177957 bytes) (Status : STANDARD)

**RFC 1025** “ TCP and IP bake off. ” J. Postel. Sep-01-1987. (Format : TXT=11648 bytes) (Status : UNKNOWN)

**RFC 1700** “ ASSIGNED NUMBERS. ” J. Reynolds, J. Postel. October 1994. (Format : TXT=458860 bytes) (Obsoletes RFC1340) (Also STD0002) (Status : STANDARD)

Sans oublier :

- W. Richard Stevens - TCP/IP Illustrated, Volume 1 - The protocols - Addison-Wesley
- Douglas Comer - Internetworking with TCP/IP - Principles, protocols, and architecture - Prentice-Hall
- McKusick – Bostic – Karels – Quateman — “ The Design and implementation of the 4.4 BSD Operating System ” (chapitre 13) — Addison-Wesley — 1996





Deuxième partie

Réseaux IP avancés



# Chapitre VII

## Routage dynamique d'IP

### 1 Introduction & rappels

La notion de routage est inhérente au fonctionnement du datagramme IP (examiné page 47).

Le routage des datagrammes IP n'est rien d'autre que l'opération qui consiste à trouver une route pour les conduire vers la destination, c'est à dire l'adresse du champ destination de l'en-tête (page 49).

Un premier examen du routage nous a conduit à distinguer le routage direct, sur un même lan et associé à l'usage des services du protocole ARP (Voir page 55) puis le routage indirect, appelé ainsi parcequ'il fait appel aux services d'une ou plusieurs passerelles avant d'atteindre la destination. Dans les deux cas la décision de routage porte sur la partie réseau de l'adresse IP du destinataire, ou encore le *netid*.

Le routage indirect se subdivise en deux catégories : le **routage statique**, qui implique l'usage d'une passerelle par défaut et enfin le **routage dynamique**, sujet qui concerne ce chapitre.

L'idée d'une route statique est séduisante par sa facilité de mise en œuvre pour l'organisation des “ petits réseaux ”. Elle se résume le plus souvent à ajouter une simple ligne dans la configuration de l'appareil à raccorder au réseau, et cette information vitale peut même être récupérée automatiquement à l'aide de protocoles comme BOOTP ou DHCP !

Sur un routeur cisco<sup>1</sup>, la ligne de configuration :

```
ip route 0.0.0.0 0.0.0.0 138.195.52.129
```

Indique que tous les datagrammes non routables directement doivent être envoyés à l'adresse 138.195.52.129. Une seule route par défaut peut être définie pour une pile IP comme il a été expliqué lors de l'analyse de l'algorithme de routage page 70.

Cette disposition très simple est bien commode car elle évite de se poser des questions compliquées sur le choix de la route, en déléguant à d'autres ce travail délicat. En effet, il faudra bien à un certain moment du trajet suivi par

---

<sup>1</sup><http://www.cisco.com>

les datagrammes, qu'un dispositif particulier, étymologiquement un routeur, prenne une décision face à des possibilités multiples.

Ce routeur plus intelligent sera probablement celui qui permet à vos datagrammes de rejoindre l'internet si vous appartenez à une entité qui a son autonomie (voir plus loin), ou le routeur du prestataires FAI pour un particulier, ou une plus petite entité, abonné à un service xDSL quelconque.

La présence de plusieurs routes possibles pour rejoindre une destination implique de facto l'usage d'un protocole de routage dynamique. Une route statique privilégie une seule route et ignore les autres. L'existence de plusieurs routes est une nécessité pour assurer la redondance du service, voire même l'équilibrage du trafic sur plusieurs liens.

## 1.1 IGP, EGP, Système autonome

Au commencement, l'Arpanet était un seul réseau géré de manière homogène, du moins par un ensemble de personnes dépendantes de la même entité administrative, ce qui permettait d'en orienter le développement de la même manière partout. Le protocole de routage dynamique était un ancêtre du protocole RIP étudié dans ce chapitre. Ce protocole, comme on va le voir, implique que les routeurs s'échangent continuellement des informations sur les meilleures routes à employer. Sans entrave, chaque routeur finit par avoir une route pour atteindre tout le monde, partout !

L'extension de ce réseau à des entités très différentes entre elles, a conduit les architectes réseaux de l'époque à créer la notion de système autonomes (*autonomous systems* ou AS dans le texte), afin de permettre à chacun de développer son réseau interne sans risque d'en diffuser le contenu à l'extérieur (soucis de confidentialité et de sécurité).

Un système autonome se caractérise par un numéro, ou numéro d'AS, sur 16 bits dont l'attribution dépend de l'IANA et de ses délégations, par exemple `autonomous-system 2192` que l'on pourrait retrouver dans la configuration d'un *external gateway* de la *figure VII.01*.

Cette nouvelle architecture entraine des changements dans l'usage des protocoles de routage. Certains sont plus adaptés que d'autres à router des blocs d'adresses IP conformément à des politiques de routage (*routing policy*) internationales, ce sont les EGP comme *external gateway protocol*. Leur ancêtre se nomme d'ailleurs EGP, il est remplacé aujourd'hui par BGP (*Border Gateway Protocol*).

À l'intérieur du système autonome, les protocoles de routages sont des IGP comme *Interior Gateway Protocol* et ne sont plus du tout adaptés à la gestion de l'internet moderne. Par contre ils répondent plus ou moins bien au besoin des réseaux internes si compliqués et vastes soient-ils. Ces IGP échangent bien entendu des routes avec les EGP, le routage ne serait pas possible sans cela.

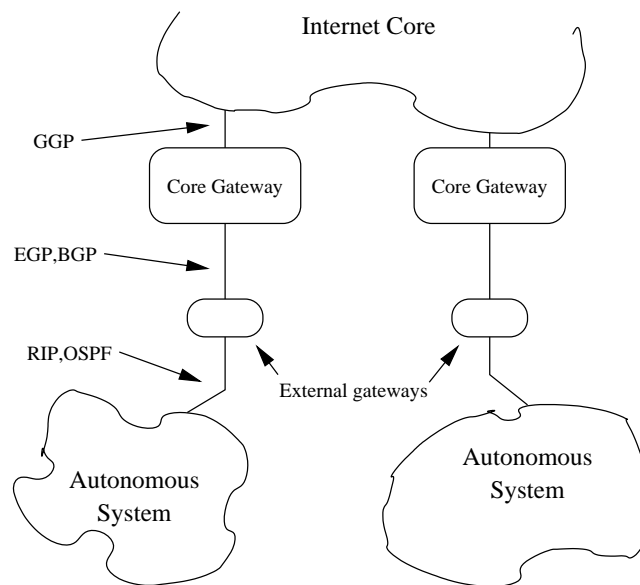


figure VII.01 — Un AS, le monde extérieur, le monde intérieur !

Ce chapitre de cours examine deux IGP très classiques, RIP et OSPF ! Si ces deux protocoles se rencontrent très fréquemment sur les réseaux, ils diffèrent beaucoup dans leurs propriétés comme nous allons le voir. . .

## 1.2 Vecteur de distances vs État de liens

RIP **R**outing **I**nformation **P**rotocol et OSPF **O**pen **S**hortest **P**ath **F**irst sont construits sur des approches différentes.

Les **algorithmes de routage à vecteur de distances** (basés sur l'algorithme de Bellman-Ford) conduisent les routeurs à transmettre à leurs voisins réseaux immédiats une copie de leur table de routage. Ces tables se modifient au fur et à mesure de leur propagation, car chaque route est associée à une métrique qui croît par défaut d'une unité au passage de chaque routeur (le routeur voisin accessible sur le même LAN est associé à une métrique de 1, etc. . .). Le choix de la meilleure route est établi par chaque routeur en considérant la valeur minimale de cette métrique pour toutes les routes qui aboutissent à la même destination. Seule la meilleure route est propagée, les autres sont oubliées.

Pour ces considérations on dit que le calcul de la route est distribué et par conséquent chaque routeur n'a pas la connaissance de la topologie globale du réseau : il n'en connaît qu'une version interprétée par ses voisins.

Les **algorithmes à états de liens** bâtissent les tables de routages différemment. Chaque routeur est responsable de la reconnaissance de tous ses voisins, plus ou moins lointains, à qui il envoie une liste complète des noms et des coûts (en terme de bande passante, par défaut) contenu dans une base de données à sa charge et qui représente l'intégralité de tous les routeurs du nuage avec lesquels il doit travailler.

Chaque routeur a donc une connaissance exhaustive de la topologie du “ nuage ” dans lequel il se situe et c’est à partir de cette représentation qu’il calcule ses routes à l’aide d’un algorithme connu de recherche du plus court chemin dans un graphe : celui de Dijkstra<sup>2</sup>

---

<sup>2</sup> <http://www.cs.utexas.edu/users/EWD/>

## 2 Routage avec RIP

RIP est l'acronyme de *Routing Information Protocol*. C'est le protocole historique de routage d'Arpanet<sup>3</sup>, défini dans la RFC 1058 (historique) de 1988. Il est amusant de constater que l'écriture de cette RFC vient de l'analyse fonctionnelle du démon `routed` présent sur les machines BSD de l'époque<sup>4</sup>.

Le principe de fonctionnement de RIP est basé sur le **calcul distribué** du **chemin le plus court** dans un graphe, selon l'algorithme Bellman-Ford<sup>5</sup> décrit à la fin des années 1950.

Le terme **chemin le plus court** désigne implicitement l'usage d'une métrique pour comparer les longueurs. Ici, la métrique est basiquement le nombre de sauts (*hops*) entre deux routeurs. Pour tout routeur, les réseaux directement rattachés sont accessibles avec un nombre de saut égal à 1 (par défaut). La métrique pour s'atteindre soit-même étant toujours 0 par hypothèse.

Les routes qui sont propagées d'un routeur à un autre voient leur métrique augmenter de 1 (ou plus) à chaque franchissement d'un routeur. En pratique on ne dépasse guère une profondeur de quelques unités, sinon le protocole devient inefficace comme on le verra au paragraphe suivant. Plus précisément, une route assortie de la métrique 16 est considérée comme *infinie*, donc désigne une destination (devenue) inaccessible.

Cette limitation du protocole laisse quand même aux architectes d'infrastructures réseaux la possibilité de concevoir des réseaux séparés les uns des autres par un maximum de 15 routeurs. . . Au delà de cette limite il faut forcément envisager l'usage d'un autre protocole !

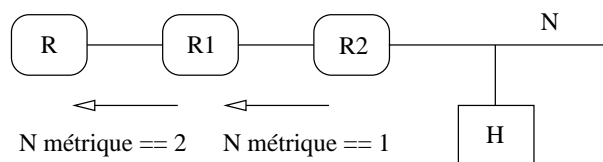


figure VII.02 — La route vers H depuis R a une métrique de 2 et passe par R1

Sur la figure VII.02 Le routeur R peut atteindre l'hôte H avec une route dont la métrique est 2 et qui passe par le routeur R1.

Il faut bien noter qu'avec RIP, chaque routeur n'est en relation qu'avec ses voisins directs, c'est à dire ceux avec lesquels il partage un LAN<sup>6</sup>. Typiquement un routeur qui fait du RIP a au moins deux interfaces (elles peuvent

<sup>3</sup>" old ARPANET routing "

<sup>4</sup>Il l'est encore de nos jours!

<sup>5</sup><http://brassens.upmf-grenoble.fr/IMSS/mamass/graphecomp/bellmannFord.htm> pour une explication très visuelle et soignée du fonctionnement de l'algorithme

<sup>6</sup>Cf page 7

être virtuelles), donc voit deux LANs. Ici R1 a un rôle central et incontournable car ni R ni R2 ne s'échangent directement des routes.

Pour cette raison, les routes sont globalement issues d'un **calcul distribué**. Pour chaque routeur l'établissement de sa table de routage s'effectue à partir des informations fournies par les routeurs de son voisinage, c'est à dire ceux qu'il peut atteindre par un routage direct (cf page 66).

La connaissance des routes acquises par chaque routeur ne s'effectue qu'au travers du résultat des calculs de routes effectués par ses voisins, calculs qu'il confrontera à sa propre table de routage et à son propre calcul de route (le choix d'une route plus courte à l'aide de la métrique annoncée), puis diffusera à son tour. Par ce principe, dans la *figure VII.02*, R1 a connaissance du réseau N indirectement grâce aux annonces de routes diffusées par R2.

Le terme **vecteurs de distances** est employé parceque la propagation des routes s'effectue sous la forme de vecteurs : “ **Pour atteindre telle destination, il faut passer par ce routeur et la métrique associée vaut cette valeur** ”. Donc une direction et une métrique, d'où l'analogie avec un vecteur.

Le moyen de propagation des tables de routes est un broadcast IP (adresse 255.255.255.255 *Limited broadcast* pour RIPv1) ou des annonces multicast (adresse 224.0.0.9 si on utilise RIPv2<sup>7</sup>).

## 2.1 En fonctionnement

1. Au démarrage, chaque routeur a connaissance des réseaux auxquels il est directement rattaché, ainsi que du coût associé à chacune de ses liaisons (1 par défaut).

Le coût de la liaison locale, c'est à dire celle du routeur vers lui-même, est “ 0 ” alors que celle pour atteindre n'importe quel autre point est “ infini ” (valeur 16 par défaut).

Le routeur envoie un paquet de questionnement (*request packet*) à ses voisins pour constituer sa table de routage initiale.

La RFC 2453 précise que celle-ci contient 5 informations pour chaque entrée :

- (a) L'adresse IPv4 de la destination,
- (b) La métrique pour atteindre cette destination,
- (c) L'adresse IPv4 de première passerelle (*next router*) à utiliser,
- (d) Un drapeau qui indique si la route a changé récemment (*route change flag*)
- (e) Deux chronomètres associés à la route, l'un pour signifier que la route n'est plus utilisable (*timeout*), l'autre pour compter le temps

<sup>7</sup>Rappelons (cf page 42) que les adresses du groupe 224.0.0.0/24 ont un TTL de 1 et donc ne sont pas routées en dehors du LAN



durant lequel une route non utilisable doit être maintenue dans la table avant d'être supprimée et l'espace mémoire utilisé recyclé (*garbage-collection*).

2. En fonctionnement chaque routeur transmet son vecteur de distance à ses voisins directs (LAN) soit par un broadcast, soit par un multicast. Le port de destination est toujours 520 ;

Cet événement a lieu périodiquement (30 secondes) où dès que quelque chose change dans la table de routage (*Triggered updates* page 117), ou encore à réception d'un paquet de demande de route, par exemple par un hôte d'un réseau directement raccordé (voir page 73) ;

3. Chaque routeur calcule son propre vecteur de distance, le coût minimum est le critère de sélection. Ce calcul intervient dès que :

Le routeur reçoit un vecteur de distance qui diffère avec ce qu'il a déjà en mémoire,

Le constat de la perte de contact (*link* ou absence de réception des annonces) avec un voisin.

4. Quand une route n'a pas été rafraîchie depuis 180 secondes (6 paquets de broadcast non reçus) sa métrique prend la valeur infinie (16) puis elle est détruite (deuxième chronomètre défini précédemment).

Le fonctionnement de RIP a un côté " magique " et pourtant l'algorithme converge vers un état stable, c'est démontré !

Examinons-en le fonctionnement élémentaire sur un réseau théorique de trois routeurs alignés lors d'un démarrage " à froid " :

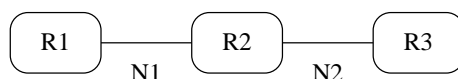


figure VII.03 — Fonctionnement élémentaire

**À l'instant 0** Chaque routeur découvre les réseaux qui lui sont directement rattachés et se connaît lui-même, c'est à dire qu'il connaît sa ou ses adresses IP. On peut le formaliser par un triplet (Destination, gateway, métrique), pour R1 ça donne (R1,local,0)<sup>8</sup> ;

- R1 annonce (R1,local,0) sur N1
- R2 annonce (R2,local,0) sur N1 et N2
- R3 annonce (R3,local,0) sur N2

**En final** Chacun annonce ses routes de manière asynchrone, met à jour sa table de routage et annonce celle-ci, tout ça de manière un peu asynchrone. On examine les tables de routages une fois ces échanges stabilisés.

---

<sup>8</sup>Notons au passage qu'une route peut être formulée vers un hôte ou un réseau, indifféremment

- R2 reçoit les annonces de route en provenance de R1 et R3. Il ajoute le coût de la liaison et obtient en final une table qui ressemble à : (R2,local,0) (R1,R1,1) (R2,R2,1),
  - De la même manière R1 enrichit sa table avec deux routes : (R2,R2,1) et (R3,R2,2).
- Pour R3 de manière symétrique : (R2,R2,1), (R1,R2,2).

Que se passe t-il maintenant si R3 devient inaccessible (coupure réseau, hôte arrêté...)?

Basiquement R2 devrait supprimer la route vers R3. Il n'en fait rien pour l'instant puisque R1 annonce une route (R3,R2,2). R2 décide donc qu'il existe une route (R3,R1,3) et annonce sa nouvelle table.

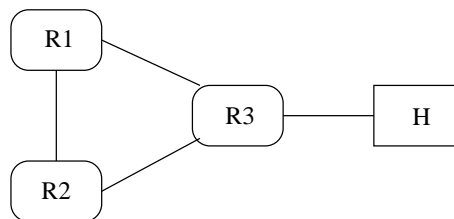
R1 ayant reçu une route modifiée de R2 modifie sa propre route qui devient (R3,R2,4) et ainsi de suite dans une boucle infernale qui tend à compter jusqu'à l'infini. Fort heureusement le calcul s'arrêtera à 16 par défaut, R1 et R2 conclueront alors que R3 n'est pas (plus) accessible et finiront par retirer la route de leur table!

On peut aisément se rendre compte de la stupidité de cette démarche ainsi (et c'est surtout ce qu'on lui reproche) de la perte de temps engendrée. D'où les améliorations apportées :

### 2.1.1 Horizon partagé ou *Split horizon*

Le concept est simple, il suffit de constater (toujours dans la *figure VII.03*) qu'il est stupide si R1 route via R2 des paquets pour R3, d'essayer pour R2 de router ces paquets vers R1.

Donc R1 ne doit pas annoncer à R2 des routes passant par R2. Les routes annoncées ne sont donc plus identiques sur chaque réseau mais tiennent compte des destinations qui sont atteintes via chacune de ces liaisons pour éviter ce type d'annonce.



*figure VII.04 : L' " horizon partagé " ne résout pas tout !*

Il existe une variante plus efficace encore, qui consiste, pour R1, à annoncer à R2 la route (R3,R2,16), donc une route infinie. R2 ne pourra donc pas utiliser cette route pour atteindre R3 via R1. Il n'y a pas de boucle de comptage à l'infini et R2 conclura tout de suite à l'inaccessibilité de R3. Ces deux astuces réunies sont repérées dans la RFC sous le terme *split horizon with poisoned reverse*.

La *figure VII.03* représente donc un cas théorique facile. En pratique, un des intérêts du routage dynamique étant d'avoir plusieurs routes possibles pour atteindre une destination, on aura plutôt la situation de la *figure VII.04* ce qui amène au cas de figure suivant :

Supposons que l'hôte H devienne inaccessible, la technique ci-dessus empêchera R3 de tenter de router les datagrammes vers R1 et R2, mais, du fait du caractère asynchrone des mises à jours, R1 ayant reçu de R3 le fait que H est inaccessible peut conclure que R2 est le meilleur chemin avec un coût de 3 et cette fausse information peut se propager à R3 via R2 et le comptage à l'infini est reparti...

Pour y remédier le protocole comporte un dispositif de mises à jours déclenchées :

### 2.1.2 Mises à jour déclenchées ou *Triggered updates*

L'éventualité d'une situation de comptage à l'infini évoquée dans le contexte de la *figure VII.04* peut être endiguée par ce dispositif.

Seules des mises à jours très rapides peuvent conduire R1 et R2 à converger vers la conclusion que la distance vers H est devenue infinie.

La règle initiale est que quand un routeur change la métrique d'une route, il doit envoyer un message de mise à jour aussi vite que possible à tous ses voisins immédiats. Ce message ne contient que ce qui a changé et non l'intégralité de la table.

Mises à jour rapides ne signifient pas pour autant " tempêtes de paquets sur le réseau ", d'une part parce que le principe de l'horizon partagé est conservé, et que d'autre part, une temporisation aléatoire (de 1 à 5 secondes) limite la fréquence d'émission de chaque mise à jour. Durant ce laps de temps la réception d'une mise à jour peut être également prise en compte et donc entraîner un changement des routes établies.

La différence entre ce dispositif et les annonces régulières tient à sa fréquence d'émission et au contenu restreint aux seules routes dont la métrique a changé.

La RFC 2453 conclue toutefois " However, counting to infinity is still possible ". Qui n'est vraiment pas très satisfaisant...

2.2 Le protocole RIPv1 vs RIPv2

RIP est encapsulé dans paquet UDP avec 520 comme port de destination :

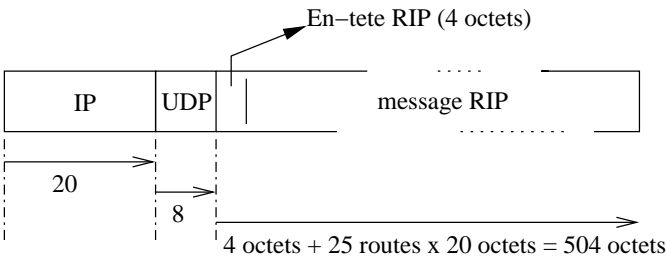


figure VII.05 — RIP est transporté par UDP/IP

Étant donné le mode de propagation des annonces de routes, le choix du protocole UDP est tout à fait approprié. Cependant, la RFC 2453 spécifie que le nombre maximum de routes est limité à 25, comme il faut 20 octets (figure VII.06) pour décrire une route, la partie utile du datagramme fait au plus  $4 + 25 \times 20 = 504$  octets et le datagramme complet 532 octets au maximum, le risque de fragmentation est nul sur des lans et via les liaisons point à point modernes (PPP par exemple).

Par contre, s'il faut propager plus de 25 routes, il faut envisager l'émission d'autant de datagramme que nécessaire !

À l'intérieur du message RIP de la figure VII.03 les octets s'organisent de la manière suivante :

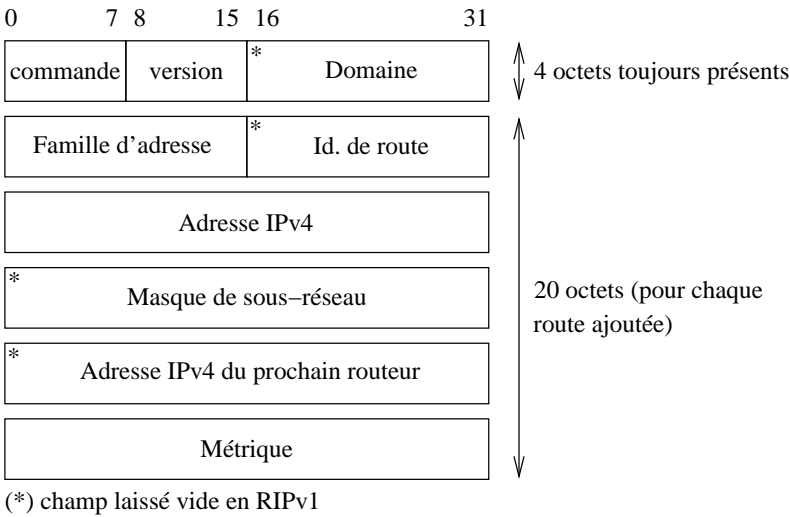


figure VII.06 — Format d'un message RIPv2

Le format d'un message RIP laisse plein d'espace vide (surtout quand il s'agit de RIPv1 — les champs marqués d'une \* sur la figure). L'alignement des champs sur des mots de 32 bits en est à l'origine.

**Commande 1** pour signifier une demande, *request*, et **2** pour une réponse, *reply*. D'autres commandes existent mais elles sont obsolètes ou non documentées dans la RFC...

**Version 1** pour RIPv1 et **2** pour RIPv2.

**Domaine** (RIPv2) Pour pouvoir faire tourner simultanément sur une même machine plusieurs instances du daemon **routed**, ce champs contient un identifiant (PID...) qui permet de discriminer la provenance des routes.

**Famille d'adresse** (*Address family identifier* - AFI) Famille d'adresse, comme pour une **socket** (cf page 253) donc **AF\_INET** pour IPv4.

Ce champ peut également contenir la valeur hexadécimale **0xffff** pour indiquer qu'il s'agit un bloc d'authentification. Dans ce cas l'**identifiant de route** contient la valeur 2 et les 16 octets qui suivent un mot de passe en clair, aligné à gauche et complété par des zéros!

Rien n'est défini dans la RFC 2453 pour être plus confidentiel...

**Identifiant de route** (*Route tag* - RIPv2)

C'est un " traceur " pour identifier une route qui provient d'un autre IGP voire d'un autre EGP et qui est propagée par RIP.

**Adresse IPv4** Il s'agit de la destination à atteindre par le routeur qui émet cette annonce.

**Masque de sous-réseau** (RIPv2) Le masque de sous-réseau à appliquer au champ qui précède. C'est un des apports principaux de RIPv2 par rapport à RIPv1.

**Adresse IPv4 du prochain routeur** (*Next hop* - RIPv2)

En fonctionnement normal l'adresse 0.0.0.0 signifie que la route passe par celui qui l'annonce. Ici il s'agit d'une autre adresse IPv4, différente de celle de l'annonceur. Celui-ci n'utilise pas RIP (sinon il ferait l'annonce lui-même), mais sans doute un autre protocole de routage.

Ce cas de figure arrive à la frontière entre deux réseaux, quand par exemple un routeur interne annonce une meilleure route via un routeur du même lan.

**Métrique** Il s'agit de l'annonce de la métrique, de 0 (hôte local) à 16 (infini non accessible) en pratique.

En résumé, les apports de RIPv2 sont les suivants :

1. Transmission d'un masque de sous-réseau avec chaque route. Ce point est majeur parcequ'il permet d'utiliser RIP avec des réseaux comportant des sous-réseaux ;
2. Authentification (très insuffisante puisque le mot de passe circule en clair). Le constructeur Cisco a ajouté des extension permettant l'usage de MD5, c'est mieux ;
3. Indication d'un prochain routeur qui n'est pas celui qui annonce la route ;

4. Indication de routes de provenances externes, ou *Route tag*;
5. Usage de l'adresse multicast 244.0.0.9 pour propager des routes (plutôt qu'un *limited broadcast* IP, plus perturbateur parceque lu par tous les hôtes.

## 2.3 Algorithme Bellman-Ford

Pour le fonctionnement de l'algorithme nous invitons le lecteur à consulter l'excellente simulation mise à disposition par l'Université Pierre Mendès France de Grenoble, à cette url (cliquer sur le bouton " Appliquette ") :

<http://brassens.upmf-grenoble.fr/IMSS/mamass/graphecomp/bellmannFord.htm>

### 2.3.1 Métrique

Dans les réseaux simples, le plus courant est d'utiliser le nombre de sauts, *hop*, c'est à dire le nombre de routeurs à franchir pour arriver à destination. Les réseaux plus complexe, on privilégiera une métrique basée sur le délais, par exemple.

## 2.4 Conclusion

L'apparition des protocoles à états de liens n'a pas empêché son développement, la RFC 2453 de 1998, décrit RIPv2 encore en usage dans bon nombre de (petits) réseaux.

### 2.4.1 Points forts

- Simplicité de mise en œuvre ;
- Simplicité du protocole permettant une compréhension aisée des échange ;
- Robustesse des implémentations.

### 2.4.2 Points faibles

- Limitation à une profondeur de 15 ;
- Problème de la vitesse de convergence (lente) de l'algorithme, et du comptage éventuel jusqu'à la route infinie ;
- La métrique n'est pas adaptée à des réseaux dont les nœuds sont séparés par des liaisons utilisant des bandes passantes disparates ;
- L'authentification de l'émetteur des données est très pauvre en fonctionnalité et pas du tout " secure " ;
- La topologie des réseaux RIP reste à un seul niveau (pas de hiérarchie par exemple entre l'arête centrale d'un réseau (*backbone*) et des réseaux terminaux.

## 3 Routage avec OSPF

L'origine du protocole OSPF, et de la technologie de routage par “ états des liaisons ”, datent du tout début des années 1980, pour faire face aux insuffisances du protocole à vecteurs de distances, constatées sur les réseaux Arpanet et Cyclades. Son développement est dû aux efforts du groupe OSPF de l'IETF.

### 3.1 Grandes lignes de fonctionnement

Les explications qui suivent font l'hypothèse d'un réseau IP qui supporte la propagation de trames avec une adresse de destination de type multicast<sup>9</sup>, autrement dit ne traite pas le cas des réseaux sans diffusion ce type ou encore NBMA (*Non Broadcast Multi-Access networks*).

Basiquement un protocole à états de liens a un fonctionnement simple :

1. Chaque routeur est responsable de la reconnaissance de ses voisins (et donc de leur nom) directs, c'est à dire accessibles sur un des LANs directement raccordés ;
2. Chaque routeur établit un paquet nommé *link state packet* (LSP) qui contient la liste des noms et des coûts (paragraphe 3.3.1) dans la métrique choisie pour atteindre chacun de ses voisins ;
3. Le LSP est propagé à tous les routeurs et chacun conserve le plus récent LSP reçu des autres routeurs dans une base de données (*link-state database*). Chaque routeur du nuage travaille ainsi à partir des mêmes données, une sorte de carte globale des états ;
4. Chaque routeur a la responsabilité par ses propres moyens (puissance CPU) du calcul du chemin à coût minimum (*shortest path*) à partir de lui-même et pour atteindre tous les nœuds du réseau ;
5. Les changements de topologie du nuage (comme la perte de connectivité sur un interface) de routeurs sont rapidement détectés, annoncés au voisinage, et pris en compte pour recalculer les routes.

En résumé un tel protocole a deux grandes activités, la première est de propager ses états et d'écouter ceux de ces voisins au sein de l'AS, c'est ce qu'on appelle le *flooding*, en français procédé par inondation, la deuxième est de calculer des routes à partir de tous les états de liens reçus. Ce calcul est effectué à l'aide de l'algorithme de Dijkstra de recherche du plus court chemin dans un graphe.

Pour les réseaux complexes, OSPF permet le groupement de routeurs en zones distinctes, *areas*, qui établissent des nuages autonomes qui routent les datagrammes entres eux, mais ne laissent pas filtrer leur trafic interne de LSP. Se dégage ainsi une hiérarchie de routeurs, ceux qui sont au milieu de

---

<sup>9</sup>Page 42

la zone, ceux qui en sont à la frontière et assurent les échanges avec les autres zones, enfin ceux qui assurent les échanges avec les EGP (comme BGP) pour le trafic externe à l'AS.

Tous les échanges sont authentifiés. Par ce biais, seulement les routeurs prévus dans la configuration participent au routage. La technique d'authentification peut différer d'une zone à une autre (cf paragraphe 3.7.2)

Enfin, les échanges de données sont structurées autour d'un protocole nommé HELLO. Ce protocole applicatif est véhiculé par deux adresses multicast (page 42) qui lui sont attribuées : principalement 224.0.0.5 et 224.0.0.6 dans certains cas, voir le paragraphe 3.6). Le protocole est encapsulé directement dans les datagrammes IP, et le champ PROTO contient la valeur 89 (fichier `/etc/protocols`).

### 3.2 RIP vs OSPF

Le choix de l'un ou de l'autre est assujéti à l'examen de ce qui les différencie, que l'on résume dans les X points suivants :

1. RIP est limité à 15 sauts (*hop*), qui limite *de facto* la structure du nuage de routeurs ;
2. Il faut utiliser RIPv2 car RIPv1 ne supporte la notion de masque de sous-réseau (*Variable Length Subnet Mask* ou VLSM) ;
3. Plus le nuage est important et les liaisons moins performantes (comme sur un WAN) plus la diffusion périodique des tables de routages consomme des ressources (bande passante) ;
4. RIP converge beaucoup plus lentement qu'OSPF. La conséquence d'un changement de la topologie peut mettre plusieurs minutes à être complètement intégrée, même avec l'usage des adresses multicast, des mises à jours déclenchées et du concept d'horizon partagé ;
5. Le principe fondateur, nombre de sauts, ne tient pas compte des délais de propagation, le plus court chemin en terme de nombre de sauts ne désigne pas nécessairement le chemin qui offre le meilleur débit, sauf si toutes les liens qui le composent sont de la même technologie (Ethernet 100BT par exemple) ;
6. L'absence de la possibilité d'une structuration des routeurs RIP en zones ne permet pas une structuration intelligente des grands réseaux, surtout quand ils sont organisés avec des classes d'adresses que l'on puisse agréger entre elles en *supernet* (page 40) ;
7. RIP n'a aucun mécanisme fiable d'authentification des annonces, ainsi n'importe quel hôte du réseau peut empoisonner l'ensemble avec des routes farfelues (ou malveillantes, ou les deux...) ;
8. Le calcul des routes est distribué pour RIP, chaque routeur n'ayant qu'une vue partielle du nuage, alors que pour OSPF chaque routeur de



la zone a une vue complète de l'état de tous les liens et établit lui-même le calcul des routes en se plaçant à la racine du graphe de destination.

En synthèse OSPF est plus performants sur les points suivants :

1. Pas de limitation en nombre de sauts, cette donnée n'entre pas en ligne de compte puisque ce sont des états de liens qui sont propagés ;
2. Les états de liens sont envoyés avec une adresse de destination multi-cast, et seules des mises à jour des états qui changent sont envoyées. La bande passante est préservée au maximum ;
3. OSPF converge très vite, du fait de son mécanisme de propagation rapide (*flooding*) des états ;
4. Le calcul du plus court chemin peut conduire à des routes de même valeur et OSPF est capable de gérer alors efficacement l'équilibre de la charge (*load balancing*) entre tous ces cheminements possibles ;
5. L'organisation des grands réseaux en zones est complètement possibles, ce qui d'une part réduit le trafic des états de liens et d'autre part permet des regroupement plus logiques basés sur les classes d'adresses IP ;
6. Les informations échangées entre routeurs peuvent être authentifiées selon plusieurs méthodes, voir paragraphe 3.7.2 ;
7. Les routes peuvent être étiquetées, ainsi les routes en provenance des EGP seront tracées et traitées spécifiquement.

### 3.3 Principe de propagation des états

L'établissement des tables de routages dépend de la complétude d'une table appelée *link-state database*, base de données d'états de liens, présente à l'identique sur chaque routeur de la zone. Cette table est alimentée par les états de liens, *Link State Packet* (LSP), que s'envoient les routeurs entre eux. Or cette distribution dépend du routage...

Contrairement à ce que l'on pourrait en déduire, il n'y a pas de problème de précedence entre ces deux opérations, car la stratégie de distribution repose sur l'usage d'une adresse de multicast, valable uniquement dans un LAN (page 42) donc qui ne dépend pas de l'état de la table de routage!

Chaque changement d'état sur un lien doit être signalé au plus vite à tous les voisins excepté celui qui a signalé le changement. C'est un procédé par inondation, ou *flooding* dans la littérature.

Intuitivement ce modèle de propagation semble rapide mais génère potentiellement un nombre exponentiel de copies de chaque paquet...

Pour éviter une tempête prévisible de LSP, l'idée initiale des concepteurs consiste à ajouter à chaque LSP un numéro de séquence :

- Chaque routeur conserve un trace du dernier numéro de séquence utilisé. Quand il génère un nouveau LSP il incrémente cette valeur ;
- Quand un routeur reçoit un LSP depuis un voisin, il compare son numéro de séquence avec celui éventuellement déjà présent dans sa base de données.
  - Si le numéro est plus ancien, il oublie le paquet,
  - Si le numéro est plus récent il remplace éventuellement celui déjà présent en mémoire.

Ce dispositif tend à modérer la tempête de mises à jour mais induit d'autres interrogations :

1. Que faire quand on arrive à la valeur maximale du compteur (même avec des registre 64bits ça arrive un jour) ? Plus généralement comment déterminer la relation d'ordre entre deux LSP de valeur a et b ?
2. Que se passe t-il quand un routeur redémarre ?  
Il annonce des LSP avec un numéro de séquence plus petit que ceux déjà en circulation et qui donc seront ignorés, même si plus pertinents. Cette constatation est voisine de la situation de deux parties d'un même réseau, séparées à la suite d'une rupture de connectivité et qui se retrouvent mais après que l'un des compteurs soit repassé par 0 ?

Qui amènent les réponses suivantes :

1. Le numéro de séquence est un compteur, avec une valeur minimale et maximale finie. Quand la valeur maximale est atteinte, il repasse par zéro, exactement comme un **counter** de SMI (page 228, chapitre concernant SNMP).

Ensuite, pour établir une relation d'ordre entre les LSP, la règle suivante est adoptée :

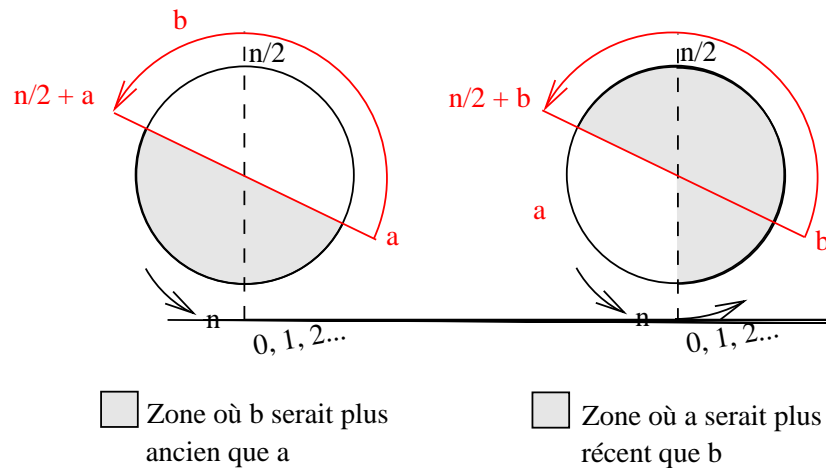


figure VII.07 — Relation d'ordre entre deux LSP

On peut déclarer que le LSP b est plus récent que a si les conditions :

$$\begin{cases} |a - b| < \frac{n}{2} \\ a < b \end{cases} \quad \text{ou encore} \quad \begin{cases} |a - b| > \frac{n}{2} \\ a > b \end{cases}$$

sont réunies. C'est ce que schématise graphiquement la *figure VII.07* ci-dessus ;

2. Pour répondre à la deuxième interrogation, on introduit une nouvelle donnée : l'âge du LSP.

C'est une valeur numérique (codée sur 16 bits selon la RFC 2328) positionnée non nulle par le routeur qui émet le LSP.

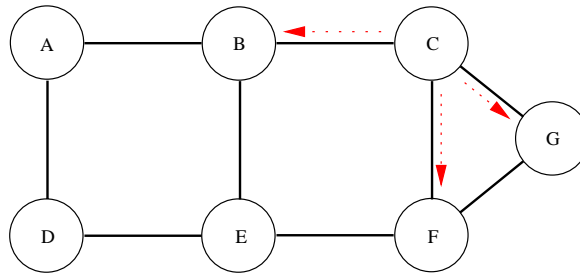
- Chaque routeur qui reçoit un LSP doit décrémenter l'âge d'au moins 1 unité et continuera ainsi dans le temps jusqu'à la valeur 0,
- À l'âge 0 le LSP ne doit plus être transmis mais peut participer encore au calcul des routes,
- N'importe quel LSP (en terme de numéro de séquence) qui arrive avec un âge non nul peut remplacer un LSP d'âge nul.

En résumé :

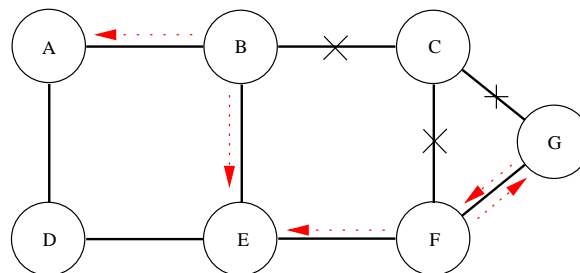
- Quand un routeur R génère un LSP, son numéro de séquence doit être plus grand de 1 (modulo n, cette dernière valeur étant la valeur maximale du compteur lui même, par exemple  $2^{32} - 1$ ) que la précédente séquence générée. L'âge doit être positionné à une valeur maximale.

- Quand un routeur autre que R reçoit le LSP, il l'accepte en remplacement de tout LSP avec un plus petit numéro de séquence (donc plus ancien).
- Si l'âge du LSP stocké était 0, le nouvel LSP le remplace de manière inconditionnelle : on ne peut propager un LSP d'âge nul.
- Le véritable algorithme est plus complexe, voir la RFC 2328 page 143, *The flooding procedure*.

Etape 1 :



Etape 2 :



Etape 3 :

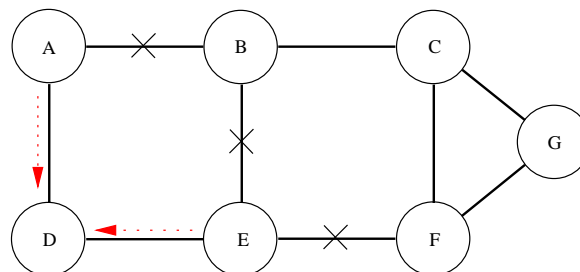


figure VII.08 — Propagation des LSP par inondation ou "flooding"

La figure VII.08 montre un exemple simple de propagation d'un changement d'état initié par le nœud C. En trois étapes tous les routeurs sont mis au courant. À l'étape 2 on peut remarquer que les routeurs B, F et G s'interdisent de renvoyer le LSP à C, son émetteur. On remarque également que F et G s'envoient le même paquet, mais celui issu de F a un âge plus ancien, il sera donc oublié immédiatement, comme celui issu de G, pour la même raison.

Le Nœud E reçoit le même LSP depuis B et F, le premier arrivé sera pris en compte, le deuxième oublié. Même remarque pour D à la fin de la troisième étape.

La durée totale de ces trois étapes est pratiquement celle nécessaire pour propager les datagrammes (donc fortement dépendante de la bande passante), de quelques milli-secondes à quelques centaines de milli-secondes, donc !

### 3.3.1 Valeur des états de liens

Le coût des liens, nommé également la métrique, agit directement sur le choix d'une route plutôt qu'une autre comme on le voit dans le paragraphe qui suit.

Le constructeur Cisco préconise<sup>10</sup> une formule qui est reprise partout :

$$cost = \frac{100000000}{\text{bande passante en bps}}$$

*bps* signifie *bits per second*. Ce qui signifie qu'une liaison Ethernet à 10Mbps (dix millions de bits par seconde, un million d'octets par secondes) a un coût de  $10^8/10^7 = 10$ .

Le petit tableau ci-dessous indique quelques valeurs pour des débit connus :

| Média                | Coût |
|----------------------|------|
| Liaison série 56kbps | 1785 |
| T1 (série 1544 kbps) | 64   |
| E1 (série 2048 kbps) | 48   |
| Token ring 4Mbps     | 25   |
| Ethernet 10Mbps      | 10   |
| Token ring 16Mbps    | 6    |
| Ethernet 100Mbps     | 1    |
| ...                  | ...  |

Bien entendu on peut toujours imposer manuellement sa propre valeur de coût d'une liaison, pour influencer le routage !

## 3.4 Calcul du plus court chemin

Pour le fonctionnement de l'algorithme de Dijkstra nous invitons le lecteur à consulter l'excellente simulation mise à disposition par l'Université Pierre Mendès France de Grenoble, à cette url (cliquer sur le bouton " Appliquette ") :

<http://brassens.upmf-grenoble.fr/IMSS/mamass/graphecomp/dijkstra.htm>

## 3.5 Hiérarchie de routeurs

Les réseaux à administrer peuvent être vastes et complexes, dans ces conditions il est souvent pertinent de les regrouper en sous-ensembles. La

<sup>10</sup><http://www.cisco.com/warp/customer/104/1.html>

conception d'OSPF permet de le faire, il s'agit d'un concept nommé zone ou (*area*) et qui se traduit par une hiérarchisation du routage.

Outre la structuration plus claire du réseau global en sous réseaux, l'avantage de cette approche est également de diminuer le nombre de routes sur lequel porte le calcul de plus court chemin, et aussi de diminuer le trafic des mises à jour, non négligeable sur un réseau vaste et complexe.

La RFC précise qu'une zone doit faire le lien avec toutes les autres, il s'agit forcément de la zone 0, qui joue donc le rôle de l'arête centrale (*OSPF Backbone*).

De cette structuration découle le fait que tous les routeurs n'ont pas le même rôle, certains sont au milieu d'une zone et d'autres à la frontière entre deux zones, voire même à la frontière entre le nuage OSPF et d'autres mécanismes de routage, vers d'autres AS :

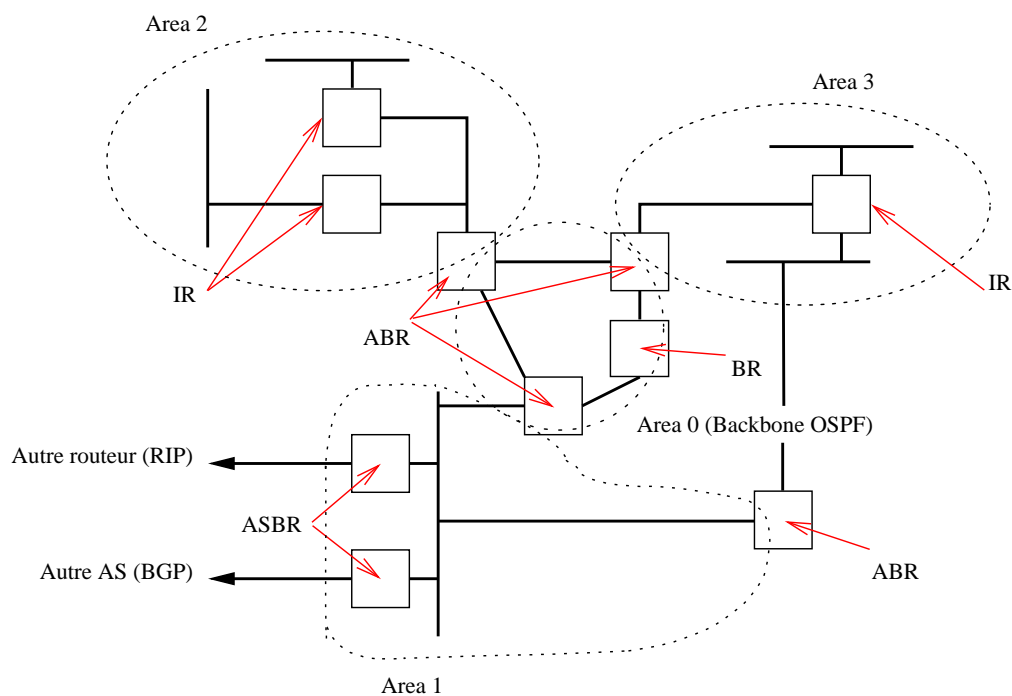


figure VII.09 — Organisation en zones – Hiérarchie de routeurs

La RFC précise quatre types de routeurs dans ce cas de figure :

**Internal routers (IR)** C'est le cas le plus simple d'un routeur au milieu d'un nuage à l'intérieur d'une zone. Il n'a qu'une seule base d'états de liens qu'il met à jour avec les autres routeurs de son voisinage ;

**Area border routers (ABR)** Ces routeurs se trouvent attachés à au moins deux zones. Ils possèdent autant de bases de données d'états de liens qu'ils ont d'interfaces connectés à des zones différentes. Ces bases diffèrent car elles concernent des nuages différents. Elles doivent être propagées vers la zone 0 sous forme d'une route résumée (*summarized*)

qui utilise au mieux les possibilités du CIDR. Bien entendu cela suppose que les réseaux puissent être agrégés entre eux ;

**Backbone routers** (BR) Il s'agit de routeurs qui sont raccordés au moins à la zone 0. La RFC n'est pas claire sur leur signification exacte...

**Autonomous system boundary routeurs** (ASBR) C'est le (les) routeur(s) qui marque(nt) la frontière d'influence de l'IGP. Il peut être en relation avec n'importe quel autre protocole de routage, par exemple RIP et BGP sur la *figure VII.09* avec lesquelles il établit des passerelles et échange des routes. Les usagers de l'IGP ont besoin d'échanges avec l'extérieur (autres réseaux, autres AS).

### 3.6 Fonctionnement à l'intérieur d'une zone

Le fonctionnement du mécanisme d'inondation à l'intérieur d'une zone, tel que nous l'avons succinctement décrit au paragraphe 3.3, induit que lors de la diffusion d'un LSP chaque routeur propage le changement d'état reçu à son voisinage réseau. Ce comportement induit un trafic en  $N^2$ , si N est le nombre de routeurs sur le LAN en question.

OSPF essaie de réduire ce nombre à seulement N en faisant jouer un rôle particulier à l'un des routeurs, le routeur désigné, ou *Designated Router* (DR). Celui-ci reçoit les mises à jours car il écoute sur une autre adresse multicast, 224.0.0.6 (tous les routeurs OSPF désignés) et si besoin est propage à nouveau cette information vers les autres routeurs du LAN, avec ce coup-ci l'adresse 224.0.0.5 (tous les routeurs OSPF).

Se pose immédiatement la question de la panne éventuelle du routeur DR, celle-ci bloquerait la mise à jour des bases d'états de liens. À cet effet un routeur désigné de sauvegarde est également élu, c'est le *Backup Designated Router*, mis à jour en même temps que le DR mais qui reste muet sur le réseau tant que le protocole HELLO n'a pas détecté un dysfonctionnement du DR.

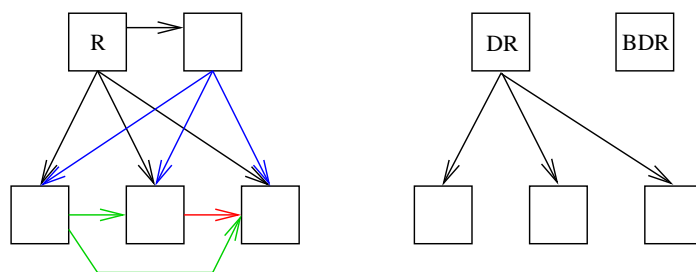


figure VII.10 — Propagation d'un LSP, sans et avec un DR

Sur la *figure VII.10*, schéma de gauche, le routeur R propage un nouvel LSP à tous ses voisins, puis chacun propage ce qu'il a reçu vers les voisins

pour lesquels il n'a rien reçu. Le nombre de paquets est alors en théorie celui du nombre de paires possibles<sup>11</sup>, soit encore :  $N \times \frac{N-1}{2}$ , 10 pour cet exemple.

Pour le schéma de droite, le routeur DR a reçu un LSP et le diffuse aux trois routeurs concernés. Notons que le BDR ne fait rien, il a également reçu la mise à jour mais s'abstient de toute action tant que le DR est opérationnel.

Avant de pouvoir établir une hiérarchie entre eux et d'échanger efficacement des états de liens, les routeurs doivent déterminer qui sont leurs voisins, autrement dit la topologie du réseau qui les entoure.

### 3.6.1 Voisinage et adjacence

La RFC 2328 définit une progression selon 8 états (7 pour les réseaux avec propagation par multicast) pour chaque routeur OSPF, avant de pouvoir échanger efficacement avec ses voisins. Il est utile d'en avoir connaissance pour diagnostiquer une situation.

L'établissement (ou non) de ces états repose sur la structuration du protocole HELLO, qui se décline en cinq types de paquets différents, nous les examinons au paragraphe 3.7

**Down** C'est l'état initial, préalable à tout établissement d'une conversation avec le voisinage. Il indique qu'aucune activité de voisinage n'a été détectée depuis un moment ;

**Init** Les routeurs envoient des paquets de type *Hello* à fréquence régulière (environ 10 secondes). La réception d'un tel paquet suffit pour passer à cet état.

Dans la liste des voisins transmise dans le paquet le routeur n'apparaît pas, la communication reste uni-directionnelle ;

**Two-way** Un routeur entre dans cet état s'il se voit dans le paquet *Hello* propagé par un voisin. La communication est alors bi-directionnelle. Cet état est la relation de voisinage la plus basique.

Pour pouvoir échanger des états de liens et construire des routes, chaque routeur doit former une contiguïté (*adjacency*) avec chacun de ses voisins. C'est une relation avancée entre routeurs OSPF. Elle s'établit en commençant par l'état suivant ;

**ExStart** C'est le premier pas pour constituer une contiguïté de routeurs entre deux voisins. Le but de cette étape est de décider qui sera le maître et l'esclave dans la relation. Des paquets de type *DataBase Description paquet* (cf page 131) sont échangées, et le routeur ayant la plus forte valeur de RID (*Router ID*) gagne. Cette dernière valeur est fonction de l'adresse IP la plus élevée pour tous les interfaces du routeur, et d'un coefficient configuré manuellement (non démocratique) ;

<sup>11</sup>C'est le nombre de paires du triangle de Pascal [http://fr.wikipedia.org/wiki/Triangle\\_de\\_Pascal](http://fr.wikipedia.org/wiki/Triangle_de_Pascal)



**Exchange** Les routeurs s'échangent l'intégralité de leur base d'états de liens à l'aide de paquets DBD ;

**Loading** À ce stade les routeurs terminent de compléter leur table de liens. Les états qui ont besoin d'être rafraîchis font l'objet de requêtes à l'aide de paquets de type *Link-state request* (LSR) auxquels sont répondus des paquets de type *Link-state update* (LSU) (Voir paragraphe 3.7) qui contiennent les LSP, appelés LSA en pratique, cœur du fonctionnement du protocole.

Les LSU sont acquittés par des *Link-state acknowledgment* (LSAck) ;

**Full** Une fois atteint cet état, l'adjacence d'un routeur avec un voisin est complète. Chaque routeur conserve une liste de ses voisins dans une base de données *adjacency database*.

### 3.7 Protocole HELLO

Le protocole HELLO est en charge de l'établissement et du maintien des relations de voisinage entre routeurs. Il s'assure également que les communications entre chaque voisin sont bi-directionnelles. Comme nous l'avons précisé en introduction ce paquet est encapsulé dans un datagramme IP, donc en lieu et place d'un protocole de transport (qu'il ne remplace pas).

Des paquets de type HELLO sont envoyés à fréquence périodique sur tous les interfaces des routeurs. Les communications sont repérées comme étant bi-directionnelles si un routeur se reconnaît dans la liste (des voisins connus) émise dans le paquet HELLO d'un voisin. Le protocole sert également à l'élection du Routeur Désigné (DR).

Sur les réseaux permettant le multicast, chaque routeur s'annonce lui-même en envoyant périodiquement des paquets HELLO. Ce dispositif permet aux routeurs voisins de se connaître dynamiquement, de vérifier continuellement l'accessibilité des voisins déjà connus.

#### 3.7.1 Cinq types de paquets

Le protocole HELLO se compose principalement d'un en-tête de 6 mots de 4 octets (24 octets) et d'un complément qui dépend du type de paquet. Ce type est défini dès le premier mot de l'en-tête.

**Hello (type 1)** Ce paquet établit et maintient les relations de voisinage (*adjacency information*) ;

**DataBase Description paquet (type 2)** (DBD) Sert à décrire le contenu des bases de données d'états de liens des routeurs OSPF lors de l'établissement d'une contiguïté de routeurs. De multiples paquets de ce type peuvent être envoyés pour décrire l'intégralité de la base de données ;

**Link-state request (type 3)** (LSR) Une fois échangée la descriptions de la base d'états, un routeur peut s'apercevoir qu'une partie des liens sont

périmés (date de fraîcheur). Ce type 3 est alors utilisé pour requérir du voisin une mise à jour. De multiples paquets peuvent être envoyés ;

**Link-state update (type 4) (LSU)** Ces paquets sont utilisés par le procédé d'inondation présenté au paragraphe 3.3. Chacun d'eux transporte une collection de LSP (on les nomme également LSA) à destination du voisinage immédiat. Pour rendre la procédure d'inondation efficace ces paquets doivent être explicitement acquittés par des paquets de type 5 ;

**Link-state acknowledgment (type 5) (LSAck)** Chaque LSA envoyé est acquitté par l'émission d'un paquet de type 5. Plusieurs acquittements peuvent être combinés dans un seul paquet.

L'adresse IP de destination peut prendre une valeur multicast ou unicast.

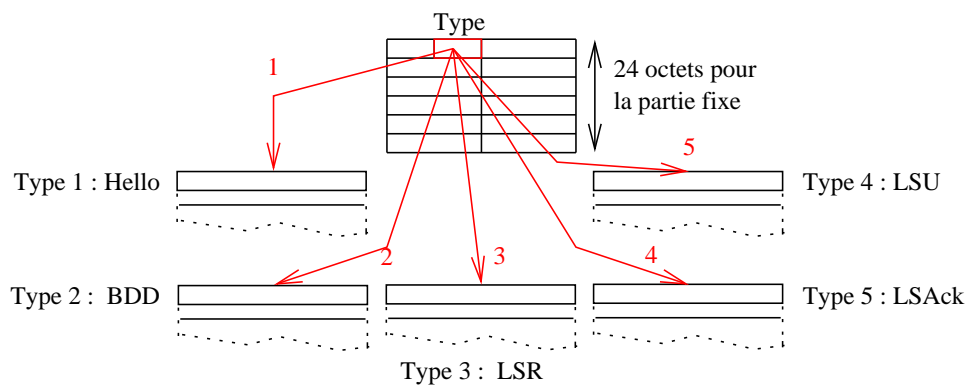


figure VII.11 — Organisation globale de l'en-tête du protocole OSPF

### 3.7.2 En-tête standard des paquets OSPF

Tous les paquets OSPF démarrent par un en-tête standard de 24 octets :

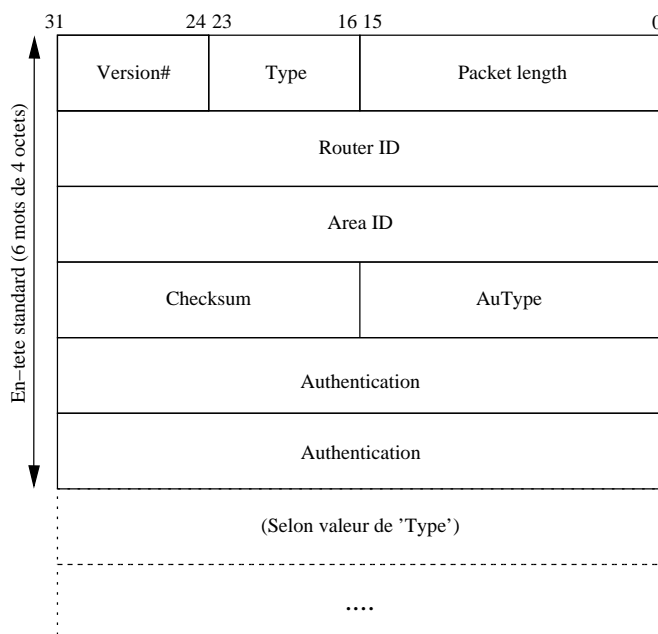


figure VII.12 — En-tête standard de 24 octets

**Version** La valeur 2 est requise, c'est la version du protocole.

**Type** Une valeur comprise entre 1 et 5 qui détermine la partie variable de l'en-tête.

**Packet length** Longueur du paquet en octets, y compris l'en-tête.

**Routeur ID** C'est l'identifiant du routeur (RID).

**Area ID** C'est le numéro de la zone. La représentation décimale pointée est utilisée, par exemple pour la zone *backbone* ce champ vaut 0.0.0.0.

**Checksum** Il porte sur la totalité du paquet moins cette zone et les 8 octets du champ *Authentication*.

**AuType** Tous les échanges sont authentifiés. Ce champ en décrit la méthode. Trois valeurs sont prévues par la RFC :

| AuType | Description                         |
|--------|-------------------------------------|
| 0      | Pas d'authentification              |
| 1      | Mot de passe en clair sur le réseau |
| 2      | Crypto à partir d'un secret partagé |

**Authentication** 64 bits qui sont utilisés selon la valeur du champ précédent.

### 3.7.3 En-tête des paquets HELLO

Un paquet de type 1 (*Hello*) est envoyé périodiquement sur tous les interfaces des routeurs qui participent au nuage OSPF. L'objectif est de maintenir

les relations de voisinages et d'adjacences comme vu précédemment. C'est une sorte de *Keep-alive* pour les besoins du protocole.

Les octets de la *figure VII.13* sont à placer en continuité de ceux de la *figure VII.12*, pour former un en-tête de  $24 + 24 = 48$  octets minimum. La taille s'accroît ensuite de quatre octets par RID supplémentaire de voisin.

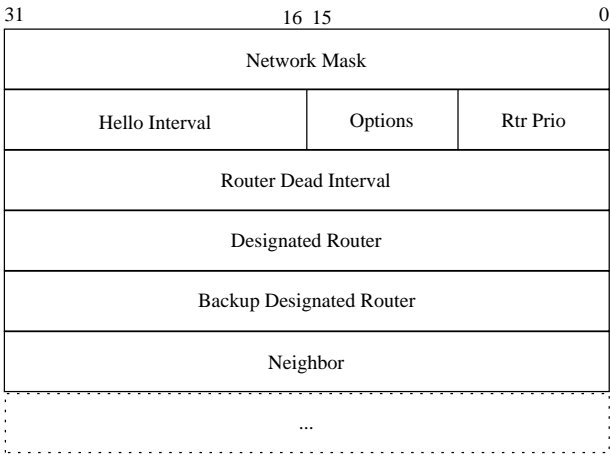


figure VII.13 — En-tête du paquet HELLO

- Network Mask** Le masque de sous réseau associé à l'interface.
- Options** Les options du routeur. Cinq bits sont utilisés seulement pour décrire des possibilités annexes au fonctionnement global.
- Hello Interval** Le nombre de secondes entre deux paquets de ce type.
- Rtr Prio** La priorité de ce routeur. C'est une valeur positionnée manuellement dans la configuration et qui a un impact direct sur le résultat de l'élection des DR et BDR. Une valeur 0 n'a aucun impact, alors que 255 assure quasiment le routeur d'être DR.
- Router Dead Interval** Le nombre de secondes avant de déclarer inatteignable un routeur devenu silencieux.
- Designated Router** C'est l'adresse IP du DR pour ce LAN. Ce champ est à 0.0.0.0 s'il n'y en a pas.
- Backup Designated Router** Idem pour le BDR.
- Neighbor** Il s'agit de la liste des RID (*Router ID*) des voisins connus et de qui on a reçu récemment (c'est à dire avec un délai inférieur à la valeur du champ Router Dead Interval) un paquet de type 1.

La description des 4 autres types de paquets se trouve à l'annexe A.3 de la RFC.

## 4 Bibliographie

Pour en savoir plus :

**RFC 1058** “ Routing Information Protocol. ” C.L. Hedrick. June 1988.  
(Format : TXT=93285 bytes) (Updated by RFC1388, RFC1723) (Status : HISTORIC)

**RFC 1247** “ OSPF Version 2. ” J. Moy. July 1991. (Format : TXT=433332, PS=989724, PDF=490300 bytes) (Obsoletes RFC1131) (Obsoleted by RFC1583) (Updated by RFC1349) (Also RFC1246, RFC1245) (Status : DRAFT STANDARD)

**RFC 2328** “ OSPF Version 2. ” J. Moy. April 1998. (Format : TXT=447367 bytes) (Obsoletes RFC2178) (Also STD0054) (Status : STANDARD)

**RFC 2453** “ RIP Version 2. ” G. Malkin. November 1998. (Format : TXT=98462 bytes) (Obsoletes RFC1723) (Also STD0056) (Status : STANDARD)

Sites web :

**CISCO OSPF Design Guide** <http://www.cisco.com/warp/customer/104/1.html>

**Algorithme de Bellman-Ford** <http://brassens.upmf-grenoble.fr/IMSS/mamass/graphecomp/bellmannFord.htm>

**Algorithme de Dijkstra** <http://brassens.upmf-grenoble.fr/IMSS/mamass/graphecomp/dijkstra.htm>

Ouvrages de références :

- o W. Richard Stevens - TCP/IP Illustrated, Volume 1 - The protocols - Addison-Wesley
- o Christian Huitema - Le routage dans l'Internet - EYROLLES
- o Radia Perlman — “ Interconnections Second Edition ” – Briges, Routers, Switches, and Internetworking Protocols — Addison-Wesley



# Chapitre VIII

## Éléments de réseaux

### 1 Hôtes ou services virtuels

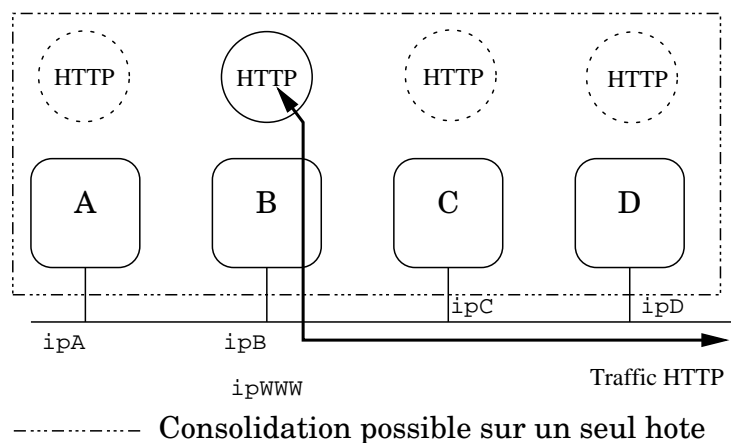


figure VIII.01 — *Serveur(s) HTTP virtuel(s)*

La machine B (d'adresse IP fixe ipB) héberge par exemple le service HTTP d'adresse ip ipWWW. Cette opération est rendue possible si le système d'exploitation de B autorise la notion d'alias IP.

Si les machines A, C et D exécutent également un serveur HTTP, elles peuvent potentiellement prendre le relais de la machine B, dès lors que l'adresse ipWWW aura été retirée de la machine B pour être reconfigurée sur l'une d'elles.

Cette opération peut se faire manuellement ou via un outil d'administration. Elle permet de faire transiter très rapidement des services d'une machine à une autre, sans rupture ou presque de la continuité. Vu des clients il s'agit toujours de la même machine, mais elle est virtuelle puisqu'elle n'existe pas physiquement.

Les systèmes d'exploitations modernes facilitent la construction de ma-

chines virtuelles. FreeBSD a un mécanisme très adapté nommé “ jail<sup>1</sup> ”, autrement dit une prison. C’est une version très améliorée de la primitive unix `chroot`. Les “ jails ” permettent de virtualiser à la demande les services puisqu’ils peuvent être démarrés ou stoppés à la demande.

Solaris 10, possède un mécanisme qui fonctionne de la même manière...Nommé “ zone<sup>2</sup> ”.

Aussi bien les zones de Solaris que les jails de FreeBSD peuvent utiliser des alias IP pour assurer leur autonomie sur le réseau, mais ces deux mécanismes manquent à ce jour d’une virtualisation complète de la stack IP qui leur permettrait d’avoir une route par défaut dans chaque instance virtuelle du système, ce qui les rendrait beaucoup plus indépendants de l’hôte hébergeur et autoriserait des configurations beaucoup plus souples.

La consolidation des hôtes A, B,C et D (et potentiellement en nombre bien plus grand encore) est possible de nos jours sur une seule machine. L’énorme montée en puissance des processeurs multi-cores et de l’évolution des architectures SMP<sup>3</sup> d’une part, et d’autre part la maturité des technologies de virtualisation des systèmes d’exploitation<sup>4</sup>

Cette opération permet d’éviter l’éparpillement des “ petits serveurs ” au profit de machines sur lesquelles on peut concentrer un effort de maintenance matérielle plus grand tout en réalisant même une économie d’échelle pour le matériel. Au niveau de la maintenance des systèmes d’exploitation l’effort d’administration reste le mêmes, puisque proportionnel au nombre d’instances en exploitation..

---

<sup>1</sup><http://docs.freebsd.org/44doc/papers/jail/jail.html>

<sup>2</sup>[http://www.sun.com/software/whitepapers/solaris10/grid\\_containers.pdf](http://www.sun.com/software/whitepapers/solaris10/grid_containers.pdf)

<sup>3</sup><http://www.sun.com/smi/Press/sunflash/2005-01/sunflash.20050118.1.xml>

<sup>4</sup>Les produits commerciaux sont bien connus, l’OpenSource n’est pas en reste avec le projet XEN <http://www.cl.cam.ac.uk/research/srg/netos/xen/>



## 2 Tunnel IP

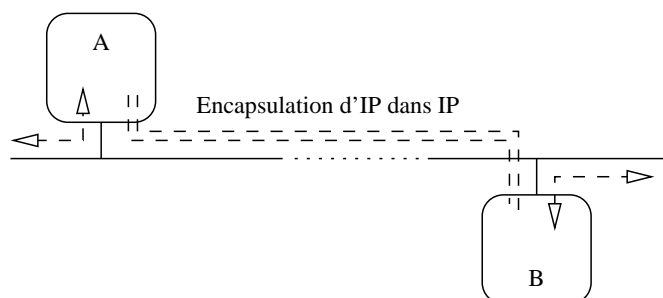


figure VIII.02 — Tunnel IP - Principe

Le tunnel permet d'encapsuler un protocole dans un autre de même niveau ou supérieur. Précédemment, page 30, nous avons déjà analysé l'encapsulation des couches de la pile Arpa selon une progression naturelle de fonctionnalités. Ici, si le principe d'encapsulation est conservé, la logique initiale de construction, elle, est bousculée. Par exemple on peut envisager d'encapsuler IP dans de multiple protocole autres qu'Ethernet, comme PPP<sup>5</sup>, IP, dans une couche de transport comme TCP, voire même dans une couche applicative comme HTTP. Ce dernier exemple peut paraître “ contre nature ” et pourtant cela fonctionne...

Construire un tunnel a un coût : d'une part celui de la perte d'espace de données dans le datagramme (il faut loger un ou plusieurs en-têtes supplémentaires et le MTU reste constant lui !) et d'autre part celui du traitement supplémentaire (décapsulation, analyse) engendré par l'ajout de ces nouveaux en-têtes.

En résumé, construire un tunnel se traduit par une **perte d'espace** pour les données dans les datagrammes et par une **consommation accrue de cycles cpus** pour traiter les en-têtes supplémentaires. Heureusement le gain en fonctionnalités pour le réseau est substantiel, comme les exemples qui vont suivre tâchent de l'illustrer !

Les tunnels qui transitent par une couche de transport sont gérés par une application (par exemple `sshd` ou `httptunnel`). Aussi le trafic de datagrammes remonte au niveau applicatif pour redescendre au niveau IP, ce qui a l'avantage de pouvoir être mis en œuvre par un utilisateur n'ayant pas nécessairement les droits de l'administrateur<sup>6</sup>, mais par contre, outre une consommation supplémentaire de cycles cpu et des changements de contexte inhérents à l'architecture logicielle<sup>7</sup>, a l'inconvénient d'être dédié à un seul

<sup>5</sup> “ Point to Point Protocol ”, lui-même éventuellement encapsulé dans de l'Ethernet (PPPoE RFC 2516) ou de l'ATM (PPPoA pour l'ADSL, RFC 2364)

<sup>6</sup> Pour encapsuler IP dans IP par exemple, il faut pouvoir écrire directement dans IP ce qui nécessite une socket en mode raw et donc un uid 0 à l'exécution

<sup>7</sup> Rappelons que les processus applicatifs standards s'exécutent en mode utilisateur, et que les transferts entre la couche de transport (dans le noyau) et la couche applicative s'effectuent via un jeu de primitives du système, voir la description des sockets de Berkeley page 251

port (par exemple celui d'une application non cryptée comme `pop` au travers une liaison `ssh`. Il faut noter que depuis la version 4.3 d'OpenSSH les tunnels sont possibles, non limités à un seul port)<sup>8</sup>.

Encapsuler IP dans IP a l'avantage de rester généraliste du point de vue des applications. Sur la *figure VIII.02* le tunnel IP encapsule donc de l'IP dans lui même. Pour les routeurs qui acheminent ce trafic il s'agit de datagrammes IP avec le type 4 (cf le fichier `/etc/protocols` au lieu des types 1 (icmp) 6 (tcp) ou 17 (udp) plus habituels.

## 2.1 Tunnel IP avec l'interface gif

La *figure*

*Romanchapter.03* illustre l'encapsulation d'IP dans IP grâce à l'usage du " `generic tunnel interface` " <sup>9</sup>. Il s'agit d'un pseudo-device (pas d'interface réel associé au device), qui permet d'encapsuler de l'IP (version 4 ou 6) dans de l'IP (version 4 ou 6)<sup>10</sup>.

Le but de cet exemple de tunnel est de montrer un routage de datagrammes issus d'un réseau privé, le `192.168.2.0/24` (RFC 1918), depuis la machine B ( $IP_B$ ), vers la machine A ( $IP_A$ ) et qui traverse un réseau public routé quelconque, non nommé sur la figure, de telle sorte que A soit intégrée au LAN `192.168.2.0/24`.

- Par hypothèse la machine A sait comment router vers le `192.168.2.0/24`. Un de ses interfaces réseaux peut être surchargé avec une adresse dans cette classe C.
- Le réseau `192.168.249.0/30` sert de réseau d'interconnexion entre les deux machines. Concrètement, il s'agit d'attribuer une adresse IP à chacun des pseudo-devices, qui ne soit pas déjà dans l'un des réseaux attachés à chacune des machines.

Conceptuellement, il serait parfaitement possible d'utiliser, par exemple, des adresses dans le `192.168.2.0/24`, mais l'auteur préfère l'usage d'un réseau d'interconnexion qui permet de bien séparer fonctionnellement les adresses IP qui constituent le tunnel en lui-même de celles qui sont amenées à l'emprunter.

De plus, si on souhaite (et c'est une quasi obligation quand on utilise des tunnels) ajouter un filtrage IP sur la machine B, il est beaucoup plus aisé pour la conception des règles de filtrage de considérer l'origine des datagrammes ayant une adresse source dans le `192.168.2.0/24` uniquement derrière le filtre.

<sup>8</sup>La mise en œuvre d'un tunnel au travers http pour contourner le filtrage en sortie d'un site n'est absolument pas recommandée par l'auteur de ces pages et laissée à la complète responsabilité du lecteur

<sup>9</sup>" `man gif` " sous FreeBSD, NetBSD, OpenBSD ou Mac OS X

<sup>10</sup>Nous avons déjà rencontré un tel interface virtuel avec l'interface de " `loopback` " `lo0`

Examinons maintenant quelle pourrait être la configuration spécifique à ce tunnel, Sur la machine A :

```
ifconfig gif0 create
ifconfig gif0 inet tunnel IP(A) IP(B)
ifconfig gif0 inet 192.168.249.1 192.168.249.2 netmask 0xffffffffc
route add -net 192.168.2.0 192.168.249.2
```

Notez l'ajout de la route spécifique vers le réseau non directement raccordé. Puis, exécution des opérations symétriques sur la machine B :

```
ifconfig gif0 create
ifconfig gif0 inet tunnel IP(B) IP(A)
ifconfig gif0 inet 192.168.249.2 192.168.249.1 netmask 0xffffffffc
```

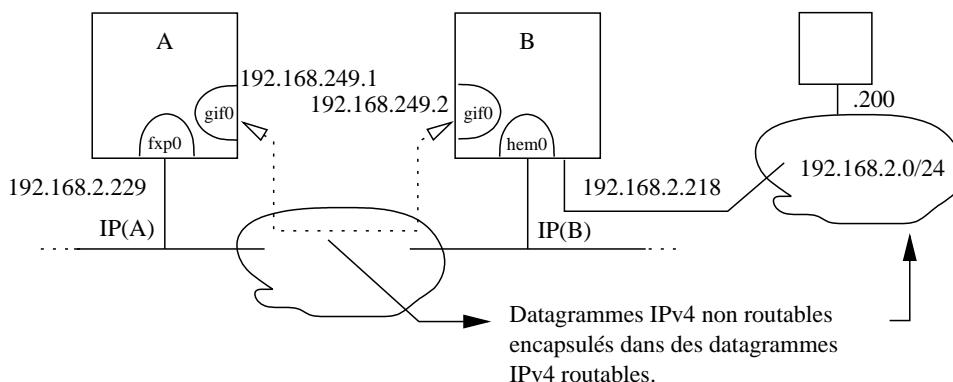


figure VIII.03 — Tunnel IP - cas concret

Notez que la première ligne de configuration précise la source et la destination réelle des datagrammes alors que la deuxième indique l'adresse locale et distante des extrémités du tunnel. C'est une écriture particulière, adaptée au pilote de l'interface gif0 pour la configuration des tunnels.

Sur la machine B, on peut voir le résultat de la configuration comme ça :

```
$ ifconfig gif0
gif0: flags=8011<UP,POINTOPOINT,MULTICAST> mtu 1280
    tunnel inet IP(B) --> IP(A)
    inet 192.168.249.2 -> 192.168.249.1 netmask 0xffffffffc
$ netstat -f inet -rn
...
192.168.249.1      192.168.249.2      UH          0      9779      -   gif0
```

Et sur la machine A (remarquez la plus petite valeur de MTU) :

```
$ ifconfig gif0
gif0: flags=8011<UP,POINTOPOINT,MULTICAST> mtu 1280
    tunnel inet IP(A) --> IP(B)
    inet 192.168.249.1 -> 192.168.249.2 netmask 0xffffffffc
$ netstat -f inet -rn
...
192.168.2.0/24    192.168.249.2      UGS         0       83      gif0
192.168.249.2     192.168.249.1      UH          1     8941      gif0
```

Enfin, si on examine<sup>11</sup> sur les interfaces `hme0` puis `gif0` de B le passage des datagrammes d'un `ping`, envoyés depuis A vers 192.168.2.200, l'observation pratique rejoint la théorie : on retrouve bien sur l'interface du tunnel (`gif0`) l'en-tête 2, décapsulé de son en-tête 1. Le datagramme est alors disponible au niveau de la pile IP de B pour être routé (routage direct<sup>12</sup> ici) vers 192.168.2.200.

Le tableau qui suit résume le contenu des en-têtes observées :

|                                   |           |            |               |
|-----------------------------------|-----------|------------|---------------|
| Sur l'interface <code>hme0</code> | En-tête 1 |            | En-tête 2     |
|                                   | Src       | $IP_A$     | 192.168.249.1 |
|                                   | Dst       | $IP_B$     | 192.168.2.200 |
|                                   | Code      | ipencap(4) | icmp          |

|                                   |         |               |
|-----------------------------------|---------|---------------|
| Sur l'interface <code>gif0</code> | En-tête |               |
|                                   | Src     | 192.168.249.1 |
|                                   | Dst     | 192.168.2.200 |
|                                   | Code    | icmp          |

### Remarques :

Attention, les routeurs filtrants doivent être spécialement configurés pour laisser passer ce type de trafic<sup>13</sup>. Les “ core gateway ” le laissent passer.

Il est intéressant de noter que le déploiement d'IPv6 est d'abord basé sur l'encapsulation de trames de la version 6 dans des trames de la version 4, en attendant que tous les routeurs soient capables de router IPv6 nativement.

Enfin pour conclure, est-il nécessaire de préciser que même encapsulés dans IP, nos datagrammes n'en sont pas moins lisibles par des yeux indiscrets ? Pour s'en prémunir il nous faut examiner une technologie complémentaire... Dans le paragraphe suivant !

<sup>11</sup>Avec `tcpdump -i hme0` puis `tcpdump -i gif0` par exemple

<sup>12</sup>Cf page 66

<sup>13</sup> Pour un routeur de type cisco qui protégerait la machine B, il faudrait ajouter des règles du genre “`permit ipinip host IPB IPA`” et “`permit ipinip host IPA IPB`”

## 2.2 IPsec et VPN

IPsec est un protocole de sécurité inclus dans la couche IP elle-même. Il est défini dans la RFC 2401. Ce paragraphe aborde brièvement la question du point de vue du protocole et de sa place dans la pile Arpa, tout en laissant volontairement dans un certain flou les aspects liés à la cryptographie, clairement absents des objectifs de ce cours<sup>14</sup>.

### 2.2.1 IPsec dans quel but ?

IPsec est un point de passage obligé pour tout administrateur de réseau qui souhaite mettre en place une politique de sécurité. D'ailleurs, pour faire le lien avec le paragraphe qui précède, précisons qu'IPsec encapsulé dans IP (formellement, un tunnel) porte un nom, le VPN (" Virtual Private Network ")! Nous avons examiné comment un tunnel accroît l'étendue d'un réseau au travers d'autres réseaux. Ajouter Ipsec introduit, entre autres, une dimension de sécurité très utilisée pour relier des machines - ou des réseaux - physiquement localisés n'importe où il y a un accès IP, en réseaux virtuels sécurisés! C'est pour cette raison qu'IPsec est un artefact incontournable de la panoplie sécuritaire sur les réseaux.

Nous aurions pu conclure le chapitre sur IP page 47 par cette constatation que le protocole IP est lisible par tout le monde y compris par les indiscrets et que quasiment n'importe quel " bricoleur " peut forger de faux datagrammes (" fake datagrams ") pour empoisonner un réseau, voire détourner les services d'une machine. Ainsi, tout ce qui renforce la sécurité IP est une bonne chose, surtout à l'heure des réseaux " wifi " dont les limites de portée ne sont pas maîtrisables.

IPsec renforce la sécurité d'IP sur plusieurs points :

**Confidentialité** Les données d'IP (protocole de transport et données applicatives) sont cryptées, donc normalement non inspectables avec tout outil d'analyse de réseau accessible sur le réseau lui-même.

**Authentification** La source du datagramme ne peut être qu'un seul émetteur, et non un intermédiaire non prévu.

**Intégrité** La totalité des données est protégée par une somme de contrôle (checksum), travail normalement dévolu à la couche de transport mais qui au niveau d'IP permet d'écarter tout datagramme qui aurait été modifié pendant son transport.

**Dispositif " anti-rejeux "** pour éviter les attaques du type " man-in-the-middle " consistants à capturer un ou plusieurs datagrammes (cryptés) dans le but de les envoyer à nouveau pour bénéficier du même effet produit que l'envoi initial.

---

<sup>14</sup>Les RFCs données page 160 sont le bon point de départ pour se documenter sur les aspects cryptographiques d'IPsec

### 2.2.2 IPsec en résumé

Ipsec (RFC 2401) est un assemblage de quatre protocoles :

**ESP** (“ Encapsulating Security Payload ”) est défini par la RFC 2406. Il assure la **confidentialité** par l’usage d’algorithmes de cryptage comme “ DES ” (RFC 2405) , “ 3DES ” (RFC 2451), “ CAST-128 ” (RFC 2144) ou encore “ blowfish ” (RFC 2451), la liste n’est pas exhaustive. . . Il faut juste noter qu’il s’agit d’algorithmes basés sur l’existence un secret partagé (manuellement dans un fichier ou crée dynamiquement avec IKE, voir plus bas) entre les parties qui échangent des messages, et non sur l’échange d’une clef publique. Cette remarque a un impact sur la manière avec laquelle on doit les configurer !

**AH** (“ Authentication Header ”) est défini par la RFC 2402. Il assure l’**authentification**, c’est à dire qu’il cherche à certifier que les deux couches IP qui dialoguent sont bien celles qu’elles prétendent être, puis l’**intégrité** des données par le calcul d’un checksum. Il faut noter que ce dernier travail empiète largement sur les attributions de la couche de transport mais se justifie compte-tenu des exigences inhérentes au fonctionnement d’IPsec.

**IPcomp** (“ IP payload compression ”) sert à compresser les données avant de les crypter. Son action est rendue nécessaire pour tenter de compenser la perte de la place occupée par les en-têtes ajoutés. Bien entendu IPcomp peut être utilisé seul.

**IKE** (“ Internet Key Exchange ”) est défini par la RFC 2409. Ce protocole n’est pas formellement indispensable au bon fonctionnement d’IPsec mais lui apporte un mécanisme d’échange de clefs, au démarrage des échanges et au cours du temps. Ainsi la clef de chiffrement n’est plus définie de manière statique dans un fichier mais change continuellement au cours du temps, ce qui est meilleur du point de vue de la sécurité.

Du point de vue de l’administration système et réseau, la mise en place de ce protocole passe par l’usage d’un *daemon*<sup>15</sup>, par exemple **racoon**, et par une ouverture de port UDP (**isakmp**/500) supplémentaire dans le filtrage du réseau. Une négociation décrite dans la RFC 2409 se déroule entre les hôtes qui doivent dialoguer, ce qui peut entrainer une certaine latence au début des échanges.

Les 32 bits de l’adresse IP de destination<sup>16</sup> permettent théoriquement d’exprimer un adressage de type unicast ou multicast ou broadcast. Si ces cas de figures sont tous théoriquement possibles, les implémentations d’IPsec ne supportent que l’unicast. La conséquence est importante sur le déploiement d’IPsec, il est effectué “ point à point ” plutôt que généralisé pour tout un réseau.

---

<sup>15</sup>Voir page 315 pour le fonctionnement des *daemons*

<sup>16</sup>Révision possible page 35

Ce travail est inclus dans ce qui est nommé “ politique de sécurité ” dans la RFC 2401.

Pour AH comme pour ESP, l’ajout de données vient se placer entre l’en-tête IP et les données. Les deux protocoles peuvent être utilisés ensemble ou séparément, c’est un choix qui relève de la politique de sécurité. Pour en tenir compte, la négociation qui a lieu lors de l’établissement d’IPsec repose sur ce que la RFC appelle des SA ( “ Security Association ”).

Une SA est formellement un triplet unique constitué d’un index unique , le SPI

( “ Security Parameter Index ”) sorte de numéro d’identification IP supplémentaire<sup>17</sup> inclus dans l’en-tête AH ou ESP, de l’adresse IP du destinataire et du protocole ESP ou d’AH. Si les deux protocoles doivent être utilisés, il faut négocier deux SAs.

### 2.2.3 Comment utiliser IPsec ?

Aux deux protocoles AH et ESP, s’ajoutent deux manières d’utiliser IPsec, soit directement d’une machine à une autre, on parle alors de “ mode transport ” soit encore en l’encapsulant dans un tunnel comme vu au paragraphe 2 page 139 et on parle de “ mode tunnel ”, plus connu sous le vocable “ VPN ”.

La RFC 2401 indique que toute implémentation se réclamant d’IPsec doit supporter les 4 associations qui suivent.

La sécurité entre deux hôtes qui supporte IPsec, au travers l’Internet, en mode transport ou en mode tunnel. Les datagrammes peuvent avoir les structures suivantes :

#### Mode transport

[IP1] [AH] [Transport] [Data]  
[IP1] [ESP] [Transport] [Data]  
[IP1] [AH] [ESP] [Transport] [Data]

#### Mode tunnel

[IP2] [AH] [IP1] [Transport] [Data]  
[IP2] [ESP] [IP1] [Transport] [Data]

figure  
Romanchapter.04 —  
En-têtes d’IPsec

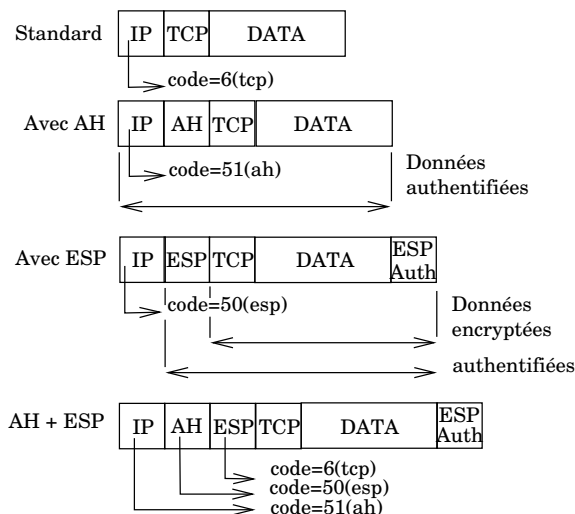
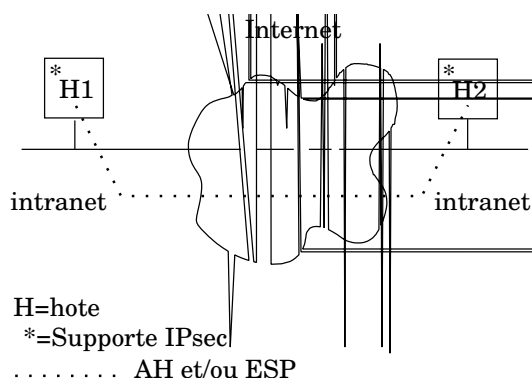


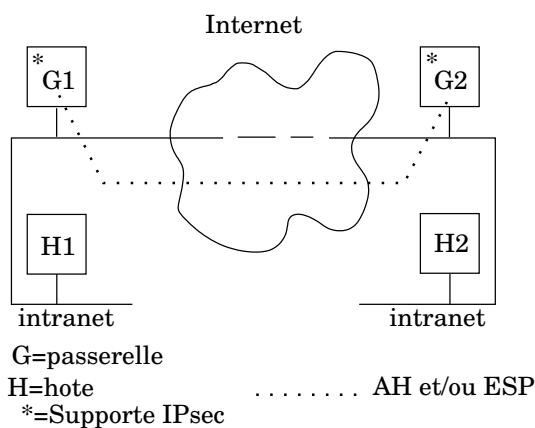
figure VIII.05 — Association 1



<sup>17</sup>Voir page 50

Remarque : En mode tunnel pour ce premier cas il n'y a pas d'obligation du support d'AH et ESP simultanément. Quand ils sont appliqués tous les deux, il faut d'abord appliquer ESP, puis AH aux datagrammes.

figure VIII.06 — Association 2



Le mode tunnel est le seul requis ici. Nous avons donc une structure de datagramme qui a ces formes possibles :

#### Mode tunnel

[IP2] [AH] [IP1] [Transport] [Data]  
[IP2] [ESP] [IP1] [Transport] [Data]

C'est la combinaison des deux premiers cas, on ajoute la sécurité entre les hôtes terminaux à celle déjà apportée par les routeurs.

La propagation du trafic de type ISAKMP (protocole IKE) au travers les routeurs est un plus.

figure VIII.07 — Association 3

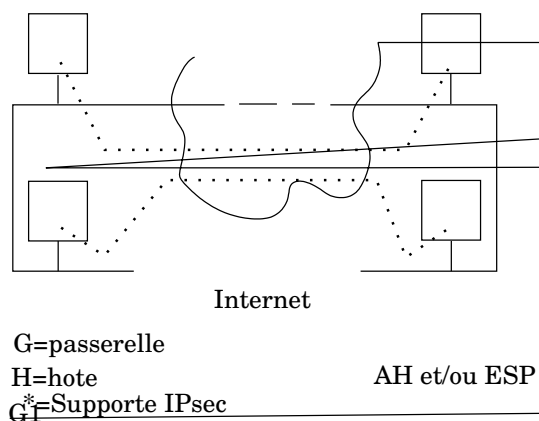
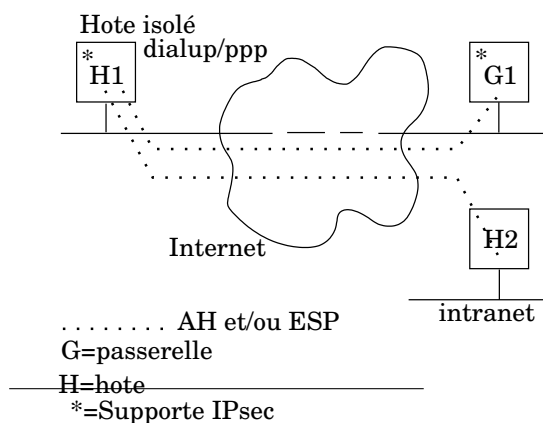


figure VIII.08 — Association 4



Ce dernier cas est celui d'un poste isolé qui se raccorde par exemple à l'intranet de son entreprise via un modem ou un accès IP non sûr, et qui utilise un protocole non crypté comme AppleTalk, ou PoP, par exemple.

Le mode tunnel est seul requis entre l'hôte H1 et la passerelle G1. Ensuite, entre H1 et H2 on revient au premier cas.



### 2.2.4 Implémentation d'IPsec

L'implémentation d'IPsec sur les machines FreeBSD et NetBSD est issue du projet KAME<sup>18</sup> et est ainsi fortement lié au développement de la pile IPv6.

Les protocoles AH, ESP et IPcomp sont inclus dans le noyau directement. La gestion des clefs s'effectue via une commande externe, **setkey** qui pilote une table de gestion des clefs située elle aussi dans le noyau, grâce à des socket de type **PF\_KEY**.

Les clefs sont soit placées de manière semi-définitive dans le fichier de configuration d'ipsec lui-même (par exemple `/etc/ipsec.conf` soit confiée aux bons soins d'un programme externe qui se charge de les créer et de les propager à l'aide du protocole IKE. Quelques **daemons** savent faire cela, notamment **racoon** du projet KAME.

Si nous reconsidérons la *figure VIII.03* les machines A et B jouent le rôle des passerelles G1 et G2 de la *figure VIII.06* (association 2). Les fichiers de configuration IPsec (AH et ESP) pourraient être :

#### Sur la machine A

```
spdadd IP(A) IP(B) any -P out ipsec \
    esp/tunnel/192.168.249.1-192.168.249.2/require \
    ah/tunnel/192.168.249.1-192.168.249.2/require;
spdadd IP(B) IP(A) any -P in ipsec \
    esp/tunnel/192.168.249.2-192.168.249.1/require \
    ah/tunnel/192.168.249.2-192.168.249.1/require;
```

**spdadd** est une instruction de la commande **setkey**. Il faut définir sa politique de sécurité, c'est à dire ce que l'on souhaite en entrée (**in**), en sortie (**out**) puis un choix de protocole (**esp**, **ah**, **ipcomp**), un mode (**tunnel** ici) avec l'entrée et la sortie du tunnel, enfin un niveau d'usage (**require** ici indique que tous échanges doivent utiliser IPsec).

#### Sur la machine B

```
spdadd IP(B) IP(A) any -P out ipsec \
    esp/tunnel/192.168.249.2-192.168.249.1/require \
    ah/tunnel/192.168.249.2-192.168.249.1/require;
spdadd IP(A) IP(B) any -P in ipsec \
    esp/tunnel/192.168.249.1-192.168.249.2/require \
    ah/tunnel/192.168.249.1-192.168.249.2/require;
```

La clef de cryptage ne figure pas dans ce fichier car l'exemple utilise IKE pour cela, à l'aide de l'outil **racoon**.

Enfin, un excellent document de configuration se trouve sur le site du projet NetBSD :

<http://www.netbsd.org/Documentation/network/ipsec/>

---

<sup>18</sup><http://www.kame.net/>

### 3 Proxy

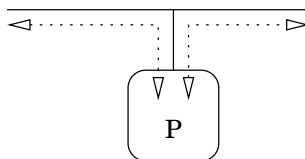


figure VIII.09 — Proxy

Le propos d'un proxy est de concentrer le trafic réseau via une seule machine pour toute une variété de protocoles (telnet, http, smtp, ...). Il existe des proxy spécialisés sur tel ou tel protocole, qui effectuent donc des tâches potentiellement très complexes (par exemple `squid` pour `http`) ou très généraux et donc moins performants sur chaque protocole (cf `nat` au paragraphe suivant).

Tout le trafic réseau qui passe par un proxy s'y arrête pour en repartir. Les conditions de ce "rebond" peuvent être paramétrées par des règles d'accès, ce qui en fait un élément utile en sécurité des réseaux (voir la RFC 1919).

*Section en chantier, précisions en cours...*

### 4 Translation d'adresses

La pénurie d'adresses IP est à l'origine du besoin de translation des adresses. Son principe se trouve décrit dans la RFC 1631.

Un tel dispositif se situe généralement à la frontière entre un réseau de type privé et un autre de type public. Le cas le plus général est lorsque le réseau public est l'internet lui-même, et le réseau privé celui d'une entité quelconque abonnée aux services d'accès réseau d'un FAI, mais ce n'est pas une obligation conceptuelle.

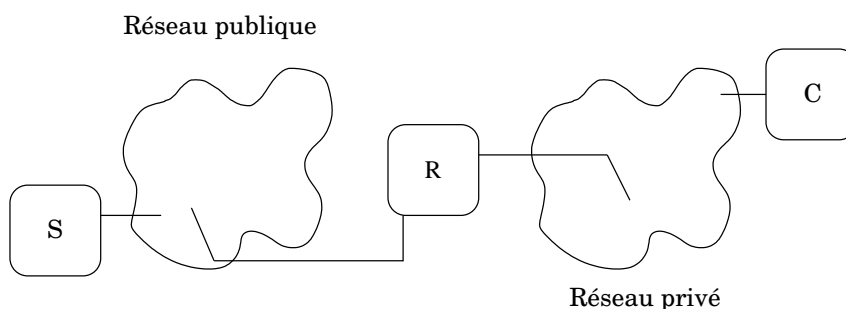


figure VIII.10 — *R* translate dynamiquement des couples (adresse IP, numéro de port)

Sur la figure 10 le réseau privé comporte plus d'hôtes que d'adresses IP fournies dans le réseau public. Pour pouvoir se développer en s'affranchissant de cette contrainte, l'usage de la translation d'adresses et de ports — NAT et PAT, ou encore NAPT comme "Network Address Port Translation" — est incontournable parce que le réseau privé est bâti avec des adresses non

routables (cf page 34) de la RFC 1918, potentiellement illimitées à l'échelle d'une entité privée, même grande. . .

R dispose de quelques adresses (un pool d'une adresse au minimum) routables sur le réseau public, qu'il peut assigner aux hôtes du réseau privé (C) qui initient une connexion vers le réseau public (S). Cette assignation peut être dynamique ou statique.

Un datagramme qui part de C vers S a une adresse source non exploitable sur le réseau public. R maintient une table, si C n'est pas déjà associé à une adresse routable du pool alloué à R, celui-ci lui en attribue une et modifie à la volée l'adresse source du datagramme, de telle sorte que le retour de S puisse être routé convenablement jusqu'à R. Puis R modifie l'adresse de destination du datagramme pour lui donner l'adresse de C, sur le réseau privé.

Si on fait l'hypothèse que la plupart des hôtes du réseau privé n'établissent pas simultanément des connexions vers le réseau public, le pool d'adresses publiques peut rester beaucoup plus petit que le nombre d'hôtes du réseau privé. Mais cette hypothèse est fragile considérant les besoins toujours plus grands d'accéder à l'information répartie.

Ce premier mécanisme se complète alors d'un second qui est le NAPT. En plus de traduire l'adresse IP à la volée, R attribue également un numéro de port différent. Ce dispositif autorise l'usage simultané d'une même adresse IP publique par des milliers d'hôtes du réseau privé.

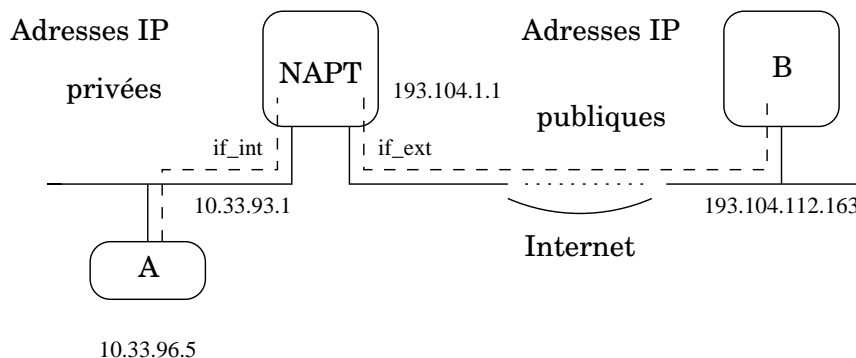
Le fonctionnement de la translation d'adresse et de port engendre une propriété intéressante pour R : il ne laisse passer aucun paquet du réseau public vers le réseau privé qui ne soit pas la réponse à une sollicitation venue du réseau privé, c'est donc en standard un fonctionnement à sens unique. Cette propriété peut être remise en question, voir le paragraphe 4.2.

Enfin, du fait du changement d'adresse source à l'aller puis d'adresse de destination au retour du datagramme, le NAPT rend impossible l'usage d'IPSEC (page 143) entre une machine quelconque du réseau public et l'interface de R dans ce réseau et sur laquelle s'effectue le travail de translation (il a modification de l'en-tête, ce contre quoi justement IPSEC est sensé nous protéger. . .). Le seul moyen dans ce cas de figure est de passer par l'usage d'un tunnel, comme vu paragraphe 2 (page 139).

Tous les routeurs modernes ont les fonctions de translation d'adresses et de ports incluses dans leurs fonctionnalités standards.

## 4.1 NAPT sur un routeur de type PC avec natd

Natd est l'outil logiciel libre bien connu des administrateurs réseaux. Il fonctionne selon le modèle de la *figure 11*<sup>19</sup>.



*figure VIII.11 — Machine NAPT en routeur*

Dans la *figure 11*, la machine “NAPT” est par hypothèse configurée en routeur. Elle représente la route par défaut pour la machine A.

Natd convertit les adresses IP à la volée. Un datagramme voit ses adresses sources (et éventuellement de destination, voir plus loin) changer dynamiquement. Examinons en détail les composantes d'une connexion établie depuis A vers B, donc lors d'un trafic “sortant” vis à vis de R.

### Pour la machine A

- La machine A s'adresse directement à la machine 193.104.112.163 en utilisant son routeur par défaut.
- L'utilisateur de la machine A “voit” la connexion soit la forme :  
**{tcp, IP Hôte A, port A, IP Hôte B, port B}**

### Pour la machine B

- La machine B voit une connexion arriver en provenance de “NAPT”.
- La machine B n'a pas connaissance de la machine A, elle dialogue avec la machine NAPT selon :  
**{tcp, IP Hôte B, port B, IP Hôte NAT, port A'}**

### Pour la machine NAPT

- La machine NAPT a connaissance des 2 réseaux, elle translate dynamiquement les adresses et les ports dans les deux sens.
- La machine NAPT fait le travail d'un proxy transparent pour la couche 3 ISO puisque chaque connexion s'y arrête puis en repart sans configuration particulière de la part de l'administrateur de A ou de B.
- La translation (ou “IP masquerading”) s'effectue dynamiquement selon l'adresse demandée.

<sup>19</sup>Les implémentations commerciales que l'on trouve dans les routeurs, si elles ne se configurent pas de la même manière, ont des propriétés très voisines en fonctionnement

- La translation d'adresse s'effectue pour les datagrammes qui traversent l'interface `if_ext`. Le dialogue entre cette machine et les autres machines du réseau "privé", via l'interface `if_int` ne fait pas l'objet d'une translation d'adresse.
- La situation de la machine A est plutôt celle d'un poste client car non vu de l'extérieur de son réseau. Être serveur est toutefois possible comme il l'est expliqué avec l'usage de `natd` au paragraphe ??.

#### 4.1.1 Interactions entre `natd` et le noyau

L'usage de `natd` sur un PC est un travail consommateur de ressources cpu parceque les datagrammes font l'objet de deux recopies et de deux changements de contexte supplémentaires : ils sont traités par un processus qui s'exécute en mode utilisateur. Sur la *figure 12* le processus `natd` ouvre une socket en mode raw pour communiquer directement avec la couche IP :

```
divertInOut = socket (PF_INET, SOCK_RAW, IPPROTO_DIVERT);
```

Le noyau IP, muni du mécanisme adéquat <sup>20</sup> redirige tout le trafic entrant et sortant d'un interface réseau, vers un numéro de port convenu à la configuration, par exemple le port 6668, à ajouter dans `/etc/services` :

```
natd 6668/divert # Network Address Translation socket
```

`Natd` lit tous les datagrammes et ne retient pour traitements que ceux qui sont à destination du port dédié.

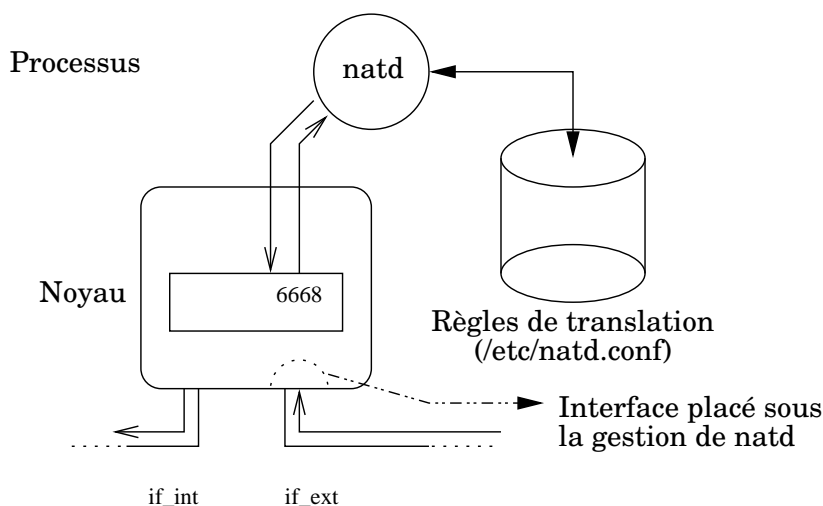


figure 12 — Interactions entre `natd` et le noyau de FreeBSD

Compte tenu du fichier de configuration, les adresses IP des datagrammes ainsi que les numéros de ports sont réécrits et reinjectés dans le noyau IP qui les traite comme d'autres datagrammes (routage, filtrage...).

<sup>20</sup>Par exemple pour FreeBSD il faut ajoute l'option `IPDIVERT` dans la configuration du noyau

## 4.2 Translation d'adresses vers le réseau privé

Les figures qui précèdent ne concernent que les connexions sortantes du réseau privé, mais on peut envisager l'inverse. Bien entendu vu du réseau public on ne voit que les adresses du pool attribué au routeur R. Le mécanisme de translation de port permet éventuellement de ventiler les connexions entrantes vers une ou plusieurs machines privées. Le critère discriminant est le numéro de port demandé.

On distingue deux attitudes, soit tout le flux entrant est redirigé sur une seule machine, soit il est effectué en fonction du port, donc du service demandé.

La littérature appelle ce cas le “static nat”. À l'insu des utilisateurs de la machine “NAPT” du réseau public, tout le trafic IP (c'est ainsi qu'il faut comprendre l'adresse IP 0.0.0.0) est renvoyé sur la machine S, et celle-ci n'est pas “visible” du réseau public.

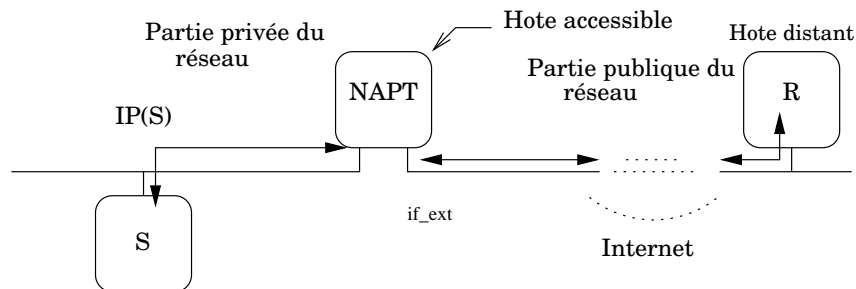


figure VIII.13 — “Static Nat ”

La configuration du natd pourrait être :

```
natd -interface if_ext -redirect_address IP(S) 0.0.0.0
```

La *figure 14* nous montre un exemple de trafic éclaté en fonction du service demandé, ce qui permet une gestion beaucoup plus fine des ressources du réseau.

Une demande de connexion de l'hôte distant R sur la machine NAT et au port 80 va être réacheminée vers la machine interne HTTP et sur le port que l'on souhaite, par exemple 8080.

Même remarque pour les deux autres services présentés.

La machine HTTP voit la connexion en provenance de la machine R sous sa forme exacte :

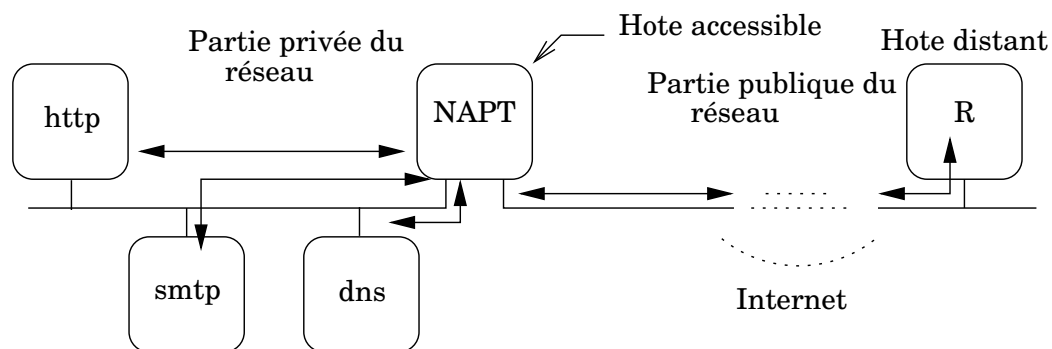
**{tcp, IP Hôte R, Port R, IP Hôte HTTP, 8080}**

La machine R ne voit que la partie “ publique ” de la connexion :

**{tcp, IP Hôte R, Port R, IP Hôte NAT, 80}**

La configuration NAPT pourrait ressembler à :

```
#
# Configuration multiservices
#
redirect_port tcp http:8080 80
redirect_port tcp smtp:25 25
redirect_port tcp dns:domain domain
redirect_port udp dns:domain domain
```



*figure 14 — Configuration multiservices*

### 4.3 NAPT sur un routeur CISCO

*Voir en travaux pratiques...*

## 5 Filtrage IP

Le propos du filtrage IP est d'appliquer des règles de filtrage à un flux de datagrammes IP afin de prendre une décision qui est le plus souvent binaire : laisser passer ou ne pas laisser passer avec en option de conserver une trace de passage (des logs).

Par son usage on cherche à protéger un site ou une machine d'un flux de datagrammes pour lesquels on suspecte que tous ne sont pas envoyés par des utilisateurs bienveillants. Le filtre s'efforce d'éliminer le trafic indésirable à partir de considérations **à priori**, mais il ne représente pas la panacée en matière de sécurité sur les réseaux, autrement dit il ne faut pas penser qu'un filtre IP suffit à régler tous les problèmes de sécurité d'un site ou d'un hôte.

En effet, à partir du moment où le filtre laisse passer certains datagrammes, même **à priori** innocents, une porte est ouverte au détournement de l'usage initial du service offert. Dans ces conditions il faut se rendre à l'évidence : il n'y a pas de sécurité absolue sur le réseau<sup>21</sup> !

Dans la littérature, un routeur filtrant est nommé " FireWall ", qu'il faut traduire en " pare-feux ".

Le filtrage IP est une caractéristique essentielle de tout routeur digne de ce nom !

Il est aussi possible de faire du filtrage IP avec les Unix libres, c'est cette approche qui est choisie dans les paragraphes qui suivent parcequ'accessible à toutes les bourses. . .

Si les détails de mise en œuvre diffèrent d'une implémentation à une autre, le fond du problème reste le même. L'implémentation choisie ici est `ipfw`, le filtre natif de FreeBSD<sup>22</sup>. Il existe d'autres filtre, notamment `ipf`, encore une fois le principe reste toujours le même.

### 5.1 Filtrage IP sur un routeur CISCO

*Voir en travaux pratiques. . .*

### 5.2 Le cas d'`ipfw` de FreeBSD

Le filtre IP en question est implémenté dans le noyau, il est activé dès lors que l'option `IPFIREWALL` est ajoutée dans le noyau. On peut également y adjoindre l'option `IPFIREWALL_VERBOSE` pour le rendre bavard<sup>23</sup>, ce qu'apprécient par dessus tout les administrateurs réseaux, soucieux d'avoir une connaissance précise de la nature du trafic sur leurs réseaux. . .

Le filtre est un module du noyau, chargé au démarrage, et qui se paramètre à l'aide de la commande `ipfw`. Celle-ci est utilisée dans les scripts de démarrage pour dicter au noyau les règles de filtrage, lues dans un fichier nommé

<sup>21</sup> Les seuls réseaux sûrs sont isolés du monde extérieur dans une cage de Faraday. . .

<sup>22</sup> <http://www.freebsd.org>

<sup>23</sup> Via `syslogd`



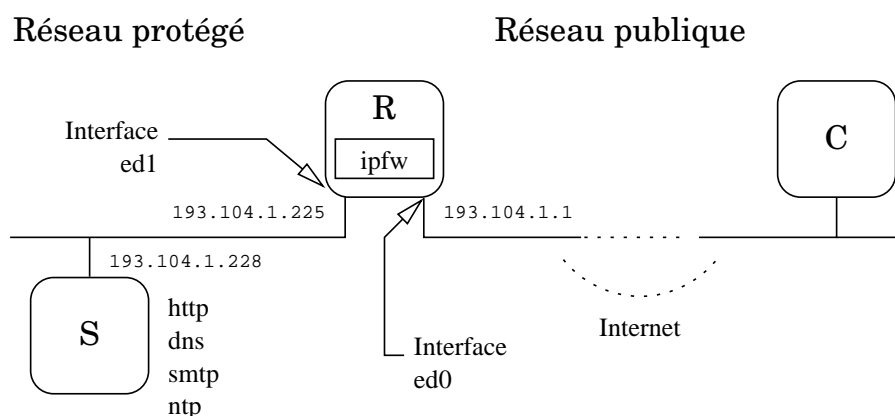
par défaut `/etc/rc.firewall`. les scripts de démarrage pour dicter au noyau les règles de filtrage,

Établir des règles de filtrage IP sous-entend avoir une connaissance exhaustive de tous les éléments qui s’y rattachent :

- Nom des interfaces réseaux impliquées
- Protocoles réseaux employés (tcp, udp, icmp,...)
- Protocoles applicatifs autorisés (smtp, domain, http...)
- Adresses IP, numéro de ports, masque de sous-réseaux
- Sens du trafic par rapport aux interfaces ci-dessus

Il est assez aisé de mettre en place un filtrage simple, par contre cette opération peut devenir complexe dès lors qu’on utilise des protocoles applicatifs compliqués, mettant en jeu une stratégie d’utilisation des ports et des adresses non triviale.

Considérons un site simple, comme celui de la *figure VIII.15*. La machine C accède depuis l’extérieur à la machine S, protégée par le filtrage IP activé sur la machine R, qui agit donc en tant que routeur filtrant.



*figure VIII.15 — Configuration simple de filtrage*

Adaptons-y les règles du modèle de base, extraites du fichier `/etc/rc.firewall` de la configuration standard d’une machine FreeBSD récente (c’est un script shell). L’examen de ces règles nous permet de découvrir la nature du trafic autorisé ou non.

```

1  # --- Interface externe
2  oif="ed0"
3  onet="193.104.1.0"
4  omask="255.255.255.224"
5  oip="193.104.1.1"
6
7  # --- Interface interne
8  iif="ed1"
9  inet="193.104.1.224"
10 imask="255.255.255.224"
11 iip="193.104.1.225"
12
13 # --- Ne pas laisser passer le ``spoofing``
14 ipfw add deny all from ${inet}:${imask} to any in via ${oif}
15 ipfw add deny all from ${onet}:${omask} to any in via ${iif}
16
17 # --- Ne pas router les adresses de la RFC1918
18 ipfw add deny all from 192.168.0.0:255.255.0.0 to any via ${oif}
19 ipfw add deny all from any to 192.168.0.0:255.255.0.0 via ${oif}
20 ipfw add deny all from 172.16.0.0:255.240.0.0 to any via ${oif}
21 ipfw add deny all from any to 172.16.0.0:255.240.0.0 via ${oif}
22 ipfw add deny all from 10.0.0.0:255.0.0.0 to any via ${oif}
23 ipfw add deny all from any to 10.0.0.0:255.0.0.0 via ${oif}
24
25 # --- Laisser passer les connexions TCP existantes
26 ipfw add pass tcp from any to any established
27
28 # --- Permettre l'arrivée du courrier SMTP
29 ipfw add pass tcp from any to 193.104.1.228 25 setup
30
31 # --- Permettre l'accès au serveur HTTP
32 ipfw add pass tcp from any to 193.104.1.228 80 setup
33
34 # --- Rejetter et faire des logs de toutes les autres demandes de connexion
35 ipfw add deny log tcp from any to any in via ${oif} setup
36
37 # --- Permettre l'établissement des autres connexions (via $iif).
38 ipfw add pass tcp from ${inet}:${imask} to any setup in via ${iif}
39
40 # --- Permettre le trafic UDP/DOMAIN vers/depuis les serveurs DNS externes
41 ipfw add pass udp from any 53 to any 53
42
43 # --- Permettre le trafic NTP vers/depuis les serveurs de dates
44 ipfw add pass udp from any 123 to any 123
45
46 # --- Permettre le passage de tous les paquets ICMP
47 ipfw allow icmp from any to any
48
49 # --- Tout ce qui n'est pas explicitement autorisé est
50 #   implicitement interdit (cf comportement par défaut d'ipfw).
51 ipfw deny ip from any to any

```

Quelques considérations :

- Les règles sont parcourues de la première à la dernière, si aucune convient, l'action par défaut consiste à bloquer le trafic (peut être changée).
- Dès qu'une règle convient, elle est appliquée et le filtrage s'arrête.
- Le filtrage IP consomme des ressources cpu, donc pour améliorer les performances il vaut mieux placer en tête de liste les règles employées le plus couramment.

Il faut remarquer que la machine 193.104.1.228 est visible depuis l'extérieure et utilise une adresse officiellement routée.

Une tentative d'accès sur un service non autorisé se traduit par un mes-

sage d'erreur (`syslogd`). Par exemple supposons que l'utilisateur de la station " C " exécute la commande suivante :

```
telnet 193.104.1.228
```

Il va obtenir le message :

```
telnet : Unable to connect to remote host : Connection timed out
```

Tandis que l'administrateur du réseau 193.104.1.0 va constater la tentative d'intrusion par le message :

```
ipfw : 3310 Deny TCP Adr.IP Hôte C :2735 193.104.1.228 :23 in via
ed0
```

Par contre, si l'intrusion consiste à détourner l'usage du service SMTP, l'administrateur du réseau 193.104.1.0 ne verra rien par ce biais puisque l'accès SMTP est autorisé sur la machine 193.104.1.228<sup>24</sup>

## 6 Exemple complet

Dans cette partie nous examinons le cas de la configuration de la *figure VIII.16*, synthèse des figures Romanchapter.13, Romanchapter.14 et Romanchapter.15. C'est une configuration très employée du fait de la distribution parcimonieuse d'adresses IP par les fournisseurs d'accès.

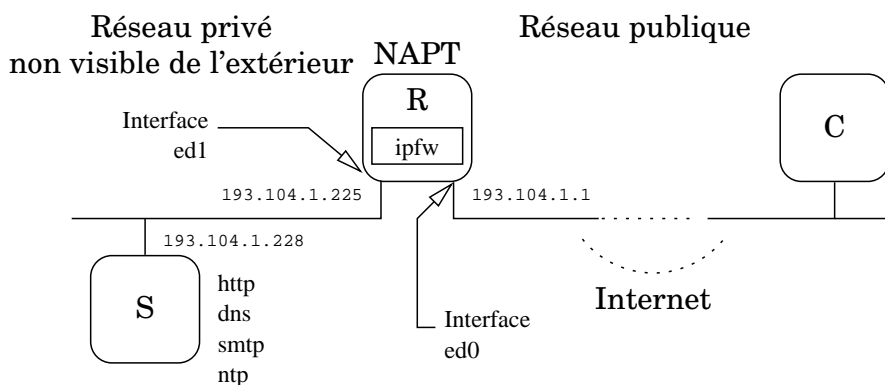


figure VIII.16 — Translation d'adresse et filtrage IP

Le propos est de mettre en place un routeur filtrant effectuant en plus la translation d'adresses IP. La juxtaposition des deux services induit peu de changements dans la configuration des règles de filtrage.

<sup>24</sup>Toute ressemblance avec la configuration réelle de ce réseau ne peut être que fortuite

Commençons par les règles de filtrage :

```

1  # --- Interface externe
2  oif="ed0"
3  onet="193.104.1.0"
4  omask="255.255.255.224"
5  oip="193.104.1.1"
6
7  # --- Interface interne
8  iif="ed1"
9  inet="193.104.1.224"
10 imask="255.255.255.224"
11 iip="193.104.1.225"
12 #
13 # --- Usage de 'natd' pour transformer tout ce qui passe sur l'interface
14 #      "ed0" donc le subnet public.
15 ipfw add divert 8668 all from any to any via ${oif}
16
17 # --- Ne pas laisser passer le ``spoofing``
18 ipfw add deny all from ${inet}:${imask} to any in via ${oif}
19 ipfw add deny all from ${onet}:${omask} to any in via ${iif}
20
21 # --- Ne pas router les adresses de la RFC1918
22 ipfw add deny all from 192.168.0.0:255.255.0.0 to any via ${oif}
23 ipfw add deny all from any to 192.168.0.0:255.255.0.0 via ${oif}
24 ipfw add deny all from 172.16.0.0:255.240.0.0 to any via ${oif}
25 ipfw add deny all from any to 172.16.0.0:255.240.0.0 via ${oif}
26 ipfw add deny all from 10.0.0.0:255.0.0.0 to any via ${oif}
27 ipfw add deny all from any to 10.0.0.0:255.0.0.0 via ${oif}
28
29 # --- Laisser passer les connexions TCP existantes
30 ipfw add pass tcp from any to any established
31
32 # --- Permettre l'arrivée du courrier SMTP
33 ipfw add pass tcp from any to 193.104.1.228 25 setup
34
35 # --- Permettre le trafic TCP/DOMAIN
36 ipfw add pass tcp from any to 193.104.1.228 53 setup
37
38 # --- Permettre l'accès au serveur HTTP
39 ipfw add pass tcp from any to 193.104.1.228 80 setup
40
41 # --- Rejeter et faire des logs de tout autre demande de connexion
42 ipfw add deny log tcp from any to any in via ${oif} setup
43
44 # --- Permettre l'établissement des autres connexions (via $iif).
45 ipfw add pass tcp from ${inet}:${imask} to any setup in via ${iif}
46
47 # --- Permettre le trafic UDP/DOMAIN vers/depuis les 'forwarders'
48 ipfw add pass udp from any 53 to 193.104.1.228 53
49 ipfw add pass udp from 193.104.1.228 53 to any 53
50
51 # --- Permettre le trafic DTP/NTP
52 ipfw add pass udp from any 123 to 193.104.1.228 123
53 ipfw add pass udp from 193.14.1.228 123 to any 123
54
55 # --- Permettre le passage des paquets ICMP (ping, traceroute...)
56 ipfw add pass icmp from any to any via ${oif} icmp type 0,3,8,11
57 ipfw add pass udp from any 32768-65535 to any 32768-65535 out xmit ${oif}
58 ipfw add log icmp from any to any in recv ${oif}
59 ipfw add pass icmp from any to any via ${iif}
60
61 # --- Tout ce qui n'est pas explicitement autorisé est
62 #      implicitement interdit (cf comportement par défaut d'ipfw).
63 ipfw deny ip from any to any
64

```

Dans le principe l'hôte 193.104.1.228 n'est plus visible de l'extérieur, les services sont en apparence hébergés par la machine R qui se charge de re-router les datagrammes en modifiant dynamiquement l'adresse de destination.

Dans l'ordre des opérations, la translation d'adresses est effectuée avant le filtrage IP. Ce sont donc des adresses IP modifiées qui sont introduites dans les règles de filtrage !

Terminons avec la configuration de `natd`. Voici le contenu du fichier `/etc/natd.conf` pour cette situation :

```
redirect_port tcp 193.104.1.228:80 80
redirect_port tcp 193.104.1.228:25 25
redirect_port tcp 193.104.1.228:53 53
redirect_port udp 193.104.1.228:53 53
redirect_port udp 193.104.1.228:123 123
```

Où l'on s'aperçoit que la configuration n'a pratiquement pas changé fondamentalement ormis par l'introduction de la règle :

**`ipfw add divert 6668 all from any to any via ${oif}`**

Qui indique au filtre que tout ce qui provient de l'interface " oif " est à lire sur le port 6668, donc a déjà subi la translation d'adresse avant d'être soumis au filtrage IP. Ainsi une demande de connexion sur le port 25 de la machine 193.104.1.1 sera transformée en une demande de connexion sur le port 25 de la machine 193.104.1.228, qui est autorisé.

Pour l'utilisateur de la station " C " la machine 193.104.1.228 n'est plus visible, seule la machine d'adresse 193.104.1.1 semble cumuler tous les services !

## 7 Bibliographie

- RFC 1631** “ The IP Network Address Translator (NAT). ” K. Egevang & P. Francis. May 1994. (Format : TXT=22714 bytes) (Status : INFORMATIONAL)
- RFC 1918** “ Address Allocation for Private Internets. ” Y. Rekhter, B. Moskowitz, D. Karrenberg, G. J. de Groot & E. Lear. February 1996. (Format : TXT=22270 bytes) (Obsoletes RFC1627, RFC1597) (Also BCP0005) (Status : BEST CURRENT PRACTICE)
- RFC 1825** “ Security Architecture for the Internet Protocol. ” R. Atkinson. August 1995. (Format : TXT=56772 bytes) (Obsoleted by RFC2401) (Status : PROPOSED STANDARD)
- RFC 2364** “ PPP Over AAL5. ” G. Gross, M. Kaycee, A. Li, A. Malis, J. Stephens. July 1998. (Format : TXT=23539 bytes) (Status : PROPOSED STANDARD)
- RFC 2401** “ Security Architecture for the Internet Protocol. ” S. Kent, R. Atkinson. November 1998. (Format : TXT=168162 bytes) (Obsoletes RFC1825) (Updated by RFC3168) (Status : PROPOSED STANDARD)
- RFC 2402** “ IP Authentication Header. ” S. Kent, R. Atkinson. November 1998. (Format : TXT=52831 bytes) (Obsoletes RFC1826) (Status : PROPOSED STANDARD)
- RFC 2406** “ IP Encapsulating Security Payload (ESP). ” S. Kent, R. Atkinson. November 1998. (Format : TXT=54202 bytes) (Obsoletes RFC1827) (Status : PROPOSED STANDARD)
- RFC 2409** “ The Internet Key Exchange (IKE). ” D. Harkins, D. Carrel. November 1998. (Format : TXT=94949 bytes) (Status : PROPOSED STANDARD)
- RFC 2516** “ A Method for Transmitting PPP Over Ethernet (PPPoE). ” L. Mamakos, K. Lidl, J. Evarts, D. Carrel, D. Simone, R. Wheeler. February 1999. (Format : TXT=32537 bytes) (Status : INFORMATIONAL)
- RFC 3168** “ The Addition of Explicit Congestion Notification (ECN) to IP. ” K. Ramakrishnan, S. Floyd, D. Black. September 2001. (Format : TXT=170966 bytes) (Obsoletes RFC2481) (Updates RFC2474, RFC2401, RFC0793) (Status : PROPOSED STANDARD)
- RFC 1919** “ Classical versus Transparent IP Proxies ”. M. Chatel. March 1996. (Format : TXT=87374 bytes) (Status : INFORMATIONAL)

Sans oublier :

- W. Richard Stevens - TCP/IP Illustrated, Volume 1 - The protocols - Addison-Wesley
- “ Firewalls and Internet Security ” - William R. Cheswick, Steven M. Bellovin - Addison-Wesley 1994.
- “ Building Internet Firewalls ” - D. Brent Chapman and Elizabeth D. Zwicky - O'Reilly - 1995. Steven M. Bellovin - Addison-Wesley 1994.





**Troisième partie**  
**Protocoles applicatifs**



# Chapitre IX

## Serveur de noms - DNS

### 1 Généralités sur le serveur de noms

S'il est obligatoire d'attribuer au moins une adresse IP à une pile ARPA pour pouvoir l'interconnecter en réseau avec d'autres piles de même nature, en revanche, lui attribuer un nom symbolique n'est absolument pas nécessaire au bon fonctionnement de ses trois couches basses.

Ce nommage symbolique est simplement beaucoup plus naturel pour nos cerveaux humains que la manipulation des adresses IP, même sous forme décimale pointée (adresses IP page 33). Il n'intervient donc qu'au niveau applicatif, ainsi la majeure partie des applications réseaux font usage de noms symboliques avec, de manière sous-jacente, une référence implicite à leur(s) correspondant(s) numérique(s).

Ce chapitre explore les grandes lignes du fonctionnement de ce que l'on nomme le “ serveur de noms ”, lien entre cette symbolique et l'adressage IP qui lui est associé .

Pour terminer cette introduction, il n'est pas innocent de préciser que le serveur de noms est en général le premier service mis en route sur un réseau, tout simplement parceque beaucoup de services le requièrent pour accepter de fonctionner (le courrier électronique en est un exemple majeur). C'est la raison pour laquelle l'usage d'adresses IP sous la forme décimale pointée reste de mise lors de la configuration des éléments de commutation et de routage<sup>1</sup>.

#### 1.1 Bref historique

Au début de l'histoire de l'Internet, la correspondance entre le nom (les noms s'il y a des synonymes ou “ alias ”) et l'adresse (il peut y en avoir plusieurs associées à un seul nom) d'une machine est placée dans le fichier `/etc/hosts`, présent sur toutes les machines unix dotées d'une pile Arpa.

Ci-après le fichier en question, prélevé (et tronqué partiellement) sur une machine FreeBSD<sup>2</sup> à jour. On y remarque qu'il ne contient plus que l'adresse

---

<sup>1</sup>Éviter un blocage dû à l'interrogation des serveurs de noms

<sup>2</sup>[www.freebsd.org](http://www.freebsd.org)

de “ loopback ” en ipv6 et ipv4.

```
# $FreeBSD$
#
# Host Database
#
# This file should contain the addresses and aliases for local hosts that
# share this file.  Replace 'my.domain' below with the domainname of your
# machine.
#
# In the presence of the domain name service or NIS, this file may
# not be consulted at all; see /etc/nsswitch.conf for the resolution order.
#
#
::1                localhost localhost.my.domain
127.0.0.1          localhost localhost.my.domain
```

Au début des années 1980 c'est le NIC<sup>3</sup> qui gère la mise à jour continue de cette table (**HOSTS.TXT**), avec les inconvénients suivants :

- Absence de structure claire dans le nommage d'où de nombreux conflits entre les noms des stations. Par exemple entre les dieux de la mythologie grecque, les planètes du système solaire, les héros historiques ou de bandes dessinées...
- Centralisation des mises à jour, ce qui entraîne :
  1. Une lourdeur du processus de mise à jour : il faut passer par un intermédiaire pour attribuer un nom à ses machines.
  2. Un trafic réseau (**ftp**) en forte croissance ( $N^2$  si  $N$  est le nombre de machines dans cette table) et qui devient rapidement ingérable au vu des bandes passantes de l'époque (quelques kilo bits par seconde), et surtout jamais à jour compte tenu des changements continuels.

D'après Douglas E. Comer, au milieu des années 1980 (1986) la liste officielle des hôtes de l'Internet contient 3100 noms et 6500 alias !

**La forte croissance du nombre des machines, a rendu obsolète cette approche.**

## 1.2 Système hiérarchisé de nommage

L'espace de noms, préalablement non structuré, est désormais réorganisé de manière hiérarchique, sous forme d'un arbre (et non d'un graphe).

Cette organisation entraîne une hiérarchisation des noms de machines et des autorités qui ont le pouvoir de les nommer, de les maintenir.

Chaque nœud de l'arbre porte un nom, la racine n'en a pas. Les machines, feuilles de l'arbre, sont nommées à l'aide du chemin parcouru de la feuille (machine) à la racine (non nommée).

<sup>3</sup> Network Information Center " (<http://www.internic.net/>)

Le séparateur entre chaque embranchement, ou nœud, est le point décimal. Voici un exemple de nom de machine :

`www.sio.ecp.fr`

Derrière ce nom il faut imaginer un point (.) qui est omis la plupart du temps car il est implicite<sup>4</sup>. La lecture s'effectue naturellement de gauche à droite, par contre la hiérarchie de noms s'observe de droite à gauche.

### 1.2.1 Domaine & zone

Le réseau peut être considéré comme une hiérarchie de domaines. L'espace des noms y est organisé en tenant compte des limites administratives ou organisationnelles. Chaque nœud, appelé un domaine, est baptisé par une chaîne de caractères et le nom de ce domaine est la concaténation de toutes les étiquettes de nœuds lues depuis la racine, donc de droite à gauche. Par exemple :

|                   |   |
|-------------------|---|
| <b>fr</b>         | Domaine <b>fr</b>                                       |
| <b>ecp.fr</b>     | Domaine <b>ecp.fr</b> sous domaine du <b>fr</b>         |
| <b>sio.ecp.fr</b> | Domaine <b>sio.ecp.fr</b> sous domaine de <b>ecp.fr</b> |

Par construction, tout nœud est un domaine, même s'il est terminal, c'est à dire n'a pas de sous domaine. Un sous domaine est un domaine à part entière et, exceptée la racine, tout domaine est un sous domaine d'un autre.

Bien que le serveur de noms, “ **Domain Name Server** ” fasse référence explicitement au concept de domaine, pour bien comprendre la configuration d'un tel service il faut également comprendre la notion de “ zone ”.

Une zone est un point de délégation dans l'arbre DNS, autrement dit une zone concerne un sous arbre du DNS dont l'administration lui est propre. Ce sous arbre peut comprendre plusieurs niveaux, c'est à dire plusieurs sous domaines. Une zone peut être confondue avec le domaine dans les cas les plus simples.

Dans les exemples ci-dessus, on peut parler de zone **sio.ecp.fr** puisque celle-ci est gérée de manière autonome par rapport à la zone **ecp.fr**.

Le serveur de noms est concerné par les “ zones ”. Ses fichiers de configuration<sup>5</sup> précisent la portée de la zone et non du domaine.

Chaque zone doit avoir un serveur principal ( “ master ” ) qui détient ses informations d'un fichier configuré manuellement ou semi manuellement (DNS dynamique). Plusieurs serveurs secondaires ( “ slave ” ) reçoivent une copie de la zone via le réseau et pour assurer la continuité du service (par la redondance des serveurs).

Le fait d'administrer une zone est le résultat d'une délégation de pouvoir de l'administrateur de la zone parente et se concrétise par la responsabilité de configurer et d'entretenir le champ **SOA** ( “ start of authority ” , page 183) de la-dite zone.

---

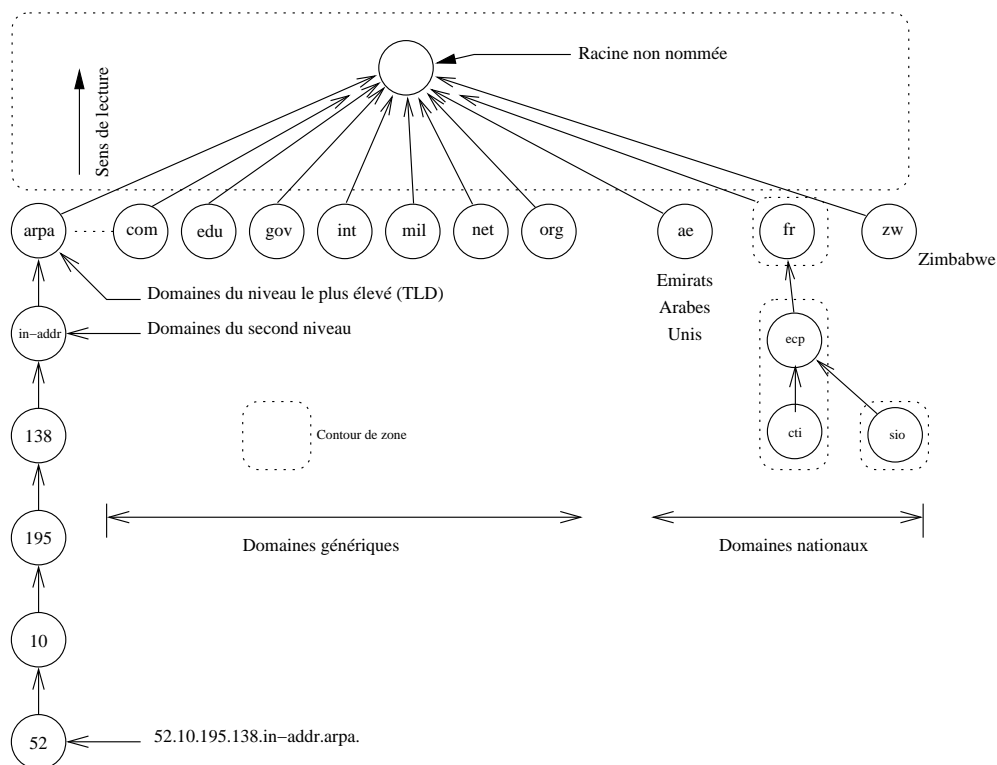
<sup>4</sup>Sauf justement dans les fichiers de configuration du serveur de noms, voir plus loin

<sup>5</sup>named.boot vs named.conf

### 1.2.2 Hiérarchie des domaines

Cette organisation du nommage pallie aux inconvénients de la première méthode :

- Le NIC gère le plus haut niveau de la hiérarchie, appelé aussi celui des “ top levels domains ” (TLD).
- Les instances régionales du NIC gèrent les domaines qui leur sont dévolus. Par exemple le “ NIC France ” <sup>6</sup> gère le contenu de la zone **.fr**. Le nommage sur deux lettres des pays est issu de la norme ISO 3166<sup>7</sup>.
- Chaque administrateur de domaine (universités, entreprises, associations, entités administratives,. .) est en charge de son domaine et des sous domaines qu’il crée. Sa responsabilité est nominative vis à vis du NIC. On dit aussi qu’il a l’autorité sur son domaine ( “ authoritative for the domain ” )<sup>8</sup>



<sup>6</sup><http://www.nic.fr/>

<sup>7</sup>On peut les voir en détail à cette adresse <http://www.nw.com/zone/iso-country-code>

<sup>8</sup>Une base de données sur les administrateurs de DNS est entretenue par les NICs, c’est la base “ whois ”, interrogeable par l’utilitaire du même nom. Consulter le site <http://www.ripe.net/> pour plus d’informations, également “ man whois ” sur une machine unix

*figure IX.01 — Organisation hiérarchique des domaines*

- Les éventuels conflits de nommage sont à la charge des administrateurs de domaine. Du fait de la hiérarchisation, des machines de même nom peuvent se trouver dans des domaines différents sans que cela pose le moindre problème.

Le nom “ **www** ” est de loin le plus employé <sup>9</sup>, pourtant il n’y a aucune confusion possible entre ces machines, comme par exemple entre les machines **www.ecp.fr** et **www.sio.ecp.fr**.

- Chaque domaine entretient une base de données sur le nommage de ses machines. Celle-ci est mise à disposition de tous les utilisateurs du réseau.

Chaque site raccordé de manière permanente procède de cette manière, ainsi il n’y a pas une base de données pour l’Internet mais un ensemble structuré de bases de données reliées entre elles et formant une gigantesque **base de données distribuée**.

## 2 Fonctionnement du DNS

### 2.1 Convention de nommage

La RFC 1034 précise que les noms de machines sont développés un peu comme les noms d’un système de fichiers hiérarchisés (Unix,...) et utilisent les caractères ascii 7 bits assortis des contraintes suivantes :

- Le “ . ” est le séparateur
- Chaque nœud ne peut faire que 63 caractères au maximum ; “ le bon usage ” les limite à 12 caractères et commençant par une lettre.
- Les majuscules et minuscules sont indifférenciées.
- Les chiffres [0-9] et le tiret peuvent être utilisés, le souligné ( \_ ) est un abus d’usage.
- Le point “ . ” et le blanc “ ” sont proscrits.
- Les chaînes de caractères comme “ NIC ” ou d’autres acronymes bien connus sont à éviter absolument, même encadrées par d’autres caractères.
- Les noms complets ne doivent pas faire plus de 255 caractères de long.

Il y a des noms “ relatifs ” et des noms “ absolus ”, comme des chemins dans un système de fichiers. L’usage du “ . ” en fin de nom, qui indique un nommage absolu<sup>10</sup>, est réservé à certains outils comme **ping** ou **traceroute** et aux fichiers de configuration du serveur de noms. En règle générale il n’est pas utile de l’employer.

---

<sup>9</sup>901 961 instances en janvier 2003 contre 1 203 856 instances en janvier 2002, selon le site du “ Network Wizards Internet Domain Suvey ” ([www.nw.com](http://www.nw.com)), “ Top 100 Host Names ”

<sup>10</sup>FQDN, comme “ Fully Qualified Domain Name ”

### 2.1.1 “ Completion ”

Sur un même réseau logique on a coutume de ne pas utiliser le nom complet des machines auxquelles on s’adresse couramment et pourtant ça fonctionne !

La raison est que le “ resolver ”, partie du système qui est en charge de résoudre les problèmes de conversion entre un nom de machine et son adresse IP, utilise un mécanisme de complétion (“ domain completion ”) pour compléter le nom de machine simplifié, jusqu’à trouver un nom plus complet que le serveur de noms saura reconnaître dans sa base de données.

Le “ resolver ” connaît par hypothèse le ou les noms de domaine (lus dans le fichier de configuration `/etc/resolv.conf`) qui concernent la machine locale. Une station de travail n’en a généralement qu’un seul alors qu’un serveur peut en comporter plusieurs, par exemple si on souhaite consolider toute une palette de services pour plusieurs domaines sur une même machine.

Exemple d’un tel fichier :

```
domain      sio.ecp.fr
search      sio.ecp.fr., ecp.fr.
nameserver  138.195.52.68
nameserver  138.195.52.69
nameserver  138.195.52.132
```

Plus généralement ce nom de domaine se présente sous forme  $d_1.d_2\dots d_n$ . Ainsi, en présence d’un nom symbolique  $x$ , le “ resolver ” teste pour chaque  $i$ ,  $i \in \{1, 2, \dots, n\}$  l’existence de  $x.d_i.d_{i+1}\dots d_n$  et s’arrête si celle-ci est reconnue. Dans le cas contraire la machine en question n’est pas atteignable.

Exemple, avec le domaine ci-dessus :

- a) machine = `www` (requête)  
`www.sio.ecp.fr`  $\Rightarrow$  Succès!
- b) machine = `www.sio` (requête)  
`www.sio.sio.ecp.fr`  $\Rightarrow$  Échec!  
`www.sio.ecp.fr`  $\Rightarrow$  Succès!

## 2.2 Le “ Resolver ”

Le “ resolver ” désigne un ensemble de fonctions<sup>11</sup> placées dans la bibliothèque standard (`gethostbyname` vu en cours de programmation invoque les fonctions du “ resolver ”) qui font l’interface entre les applications et les serveurs de noms. Par construction les fonctions du “ resolver ” sont compilées avec l’application qui les utilise (physiquement dans la `libc`, donc accessibles par défaut).

<sup>11</sup>`res_query`, `res_search`, `res_mkquery`, `res_send`, `res_init`, `dn_comp`, `dn_expand` - Faire “ man resolver ” sur une machine unix



Le “ resolver ” applique la stratégie locale de recherche, définie par l’administrateur de la machine, pour résoudre les requêtes de résolution de noms. Pour cela il s’appuie sur son fichier de configuration `/etc/resolv.conf` et sur la stratégie locale (voir paragraphe suivant) d’emploi des possibilités (serveur de noms, fichier `/etc/nsswitch.conf`, NIS,...).

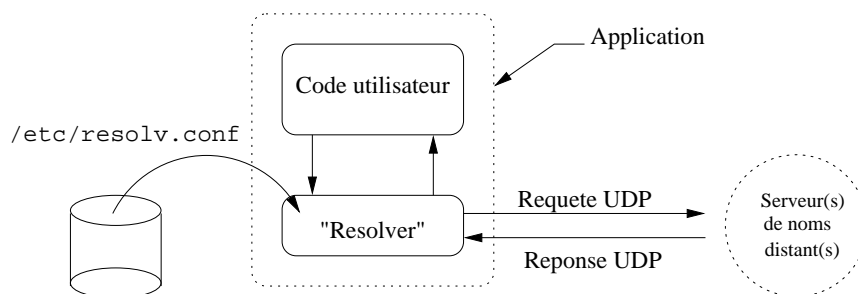


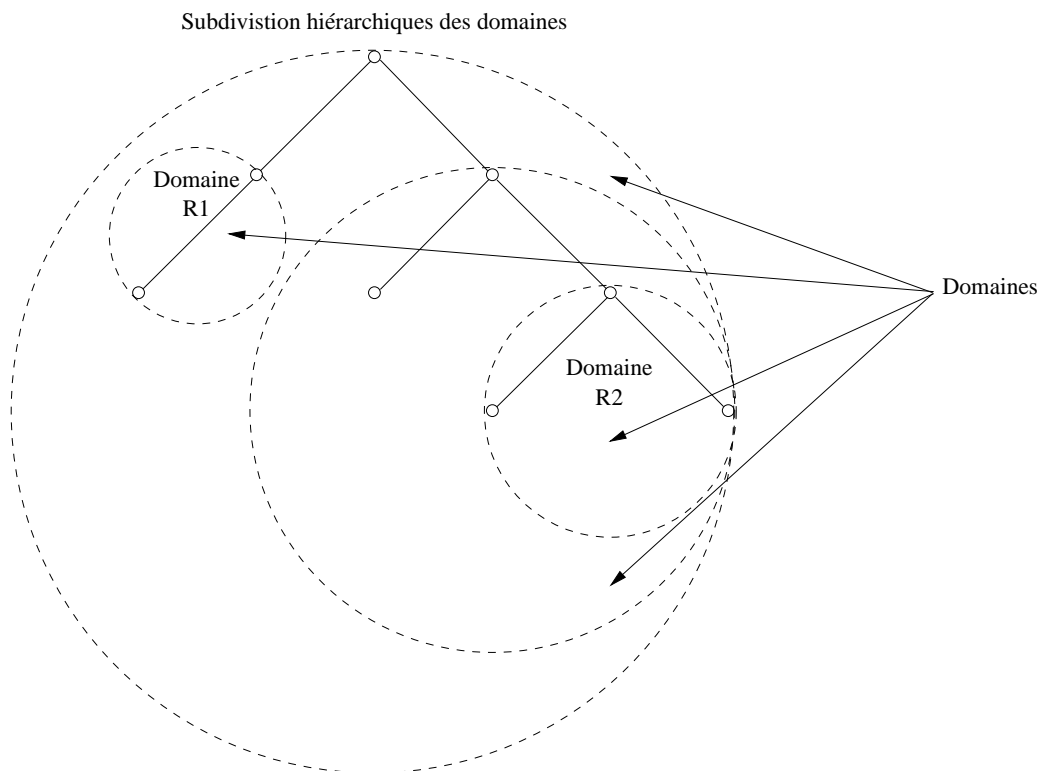
figure IX.02 — Le “ resolver ” dans son environnement

Le fichier `/etc/resolv.conf` précise au moins le domaine local assorti de directives optionnelles.

## 2.3 Stratégie de fonctionnement

La *figure IX.03* illustre le fait que chaque serveur de noms a la maîtrise de ses données mais doit interroger ses voisins dès qu'une requête concerne une zone sur laquelle il n'a pas l'autorité de nommage.

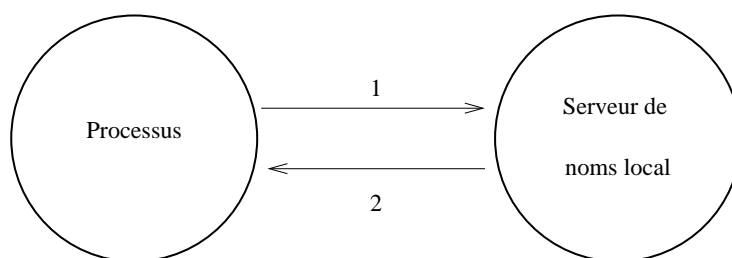
Ainsi, un hôte du domaine " R2 " qui veut résoudre une adresse du domaine " R1 " doit nécessairement passer par un serveur intermédiaire pour obtenir l'information. Cette démarche s'appuie sur plusieurs stratégies possibles, que nous examinons dans les paragraphes suivants.



*figure IX.03 — Subdivision hiérarchique des domaines*

### 2.3.1 Interrogation locale

La figure ci-dessous illustre la recherche d'un nom dans le domaine local.



*figure IX.04 — Interrogation locale*

Un processus ( “ browser ” `http` par exemple) recherche l’adresse d’un nom de serveur. Sur les machines Unix cela se traduit par l’appel à la fonction `gethostbyname`. Cette fonction est systématiquement présente dans la bibliothèque standard (`libc`) et est donc accessible potentiellement à tout exécutable lors d’une compilation.

La fonction `gethostbyname` fait systématiquement appel au “ resolver ” déjà cité. C’est donc toujours en passant par ce mécanisme que les processus accèdent à l’espace de noms. Le “ resolver ” utilise une stratégie générale à la machine (donc qui a été choisie par son administrateur) pour résoudre de telles requêtes :

1. Interrogation du serveur de noms (DNS) si présent
2. Utilisation des services type “ YP ” (NIS) si configurés
3. Utilisation du fichier `/etc/hosts`

Cette stratégie est paramétrable en fonction du constructeur. Le `nsswitch` sous HP-UX<sup>12</sup> ou Solaris<sup>13</sup> permet de passer de l’un à l’autre en cas d’indisponibilité, le fichier `/etc/nsswitch.conf` sous FreeBSD effectue un travail assez proche.

Enfin, quelle que soit l’architecture logicielle le “ resolver ” est configuré à l’aide du fichier `/etc/resolv.conf`.

Sur la *figure IX.04* :

1. Le processus demande l’adresse IP d’un serveur. Le “ resolver ” envoie la demande au serveur local.
2. Le serveur local reçoit la demande, parcequ’il a l’autorité sur le domaine demandé (le sien), il répond directement au “ resolver ”.

### 2.3.2 Interrogation distante

1. Un processus demande l’adresse IP d’une machine. Le “ resolver ” envoie sa requête au serveur local.
2. Le serveur local reçoit la requête et dans ce deuxième cas il ne peut pas répondre directement car la machine n’est pas dans sa zone d’autorité. Pour lever l’indétermination il interroge alors un serveur racine pour avoir l’adresse d’un serveur qui a l’autorité sur la zone demandée par le processus.
3. Le serveur racine renvoie l’adresse d’un serveur qui a officiellement l’autorité sur la zone
4. Le serveur local interroge ce nouveau serveur distant.
5. Le serveur distant renvoie l’information demandée au serveur local.
6. Le serveur local retourne la réponse au “ resolver ”

---

<sup>12</sup>Unix de “ Hewlett-Packard ”

<sup>13</sup>Unix de “ Sun Microsystems ”

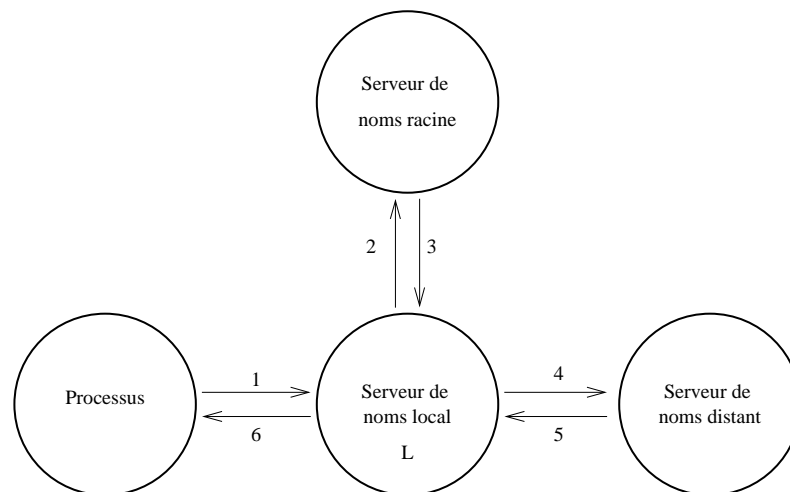


figure IX.05 — Interrogation distante

Remarques importantes :

- Un mécanisme de cache accélère le processus ci-dessus : Si un processus redemande la même machine distante on se retrouve dans le cas d’une interrogation “ locale ”, du moins pendant la durée de validité des données (cf page 186).
- Si un processus demande une machine du même domaine que la précédente (mais pas du même nom ! :), les étapes 2 et 3 deviennent inutiles et le serveur local interroge alors directement le serveur distant. La durée de vie de l’adresse du serveur distant est elle aussi assortie d’une date limite d’utilisation.
- Dans le cas général les serveurs racines ne voient pas plus de 1 ou deux niveaux en dessous. Ainsi, si un processus demande A.B.C.D.net :
  1. Le serveur racine donne l’adresse d’un serveur pour D
  2. Le serveur pour D donnera peut être l’adresse d’un serveur pour C et ainsi jouera le rôle de serveur racine de l’étape précédente.
- On dit également que le serveur L de la figure IX.05 fonctionne en mode récursif.

### 2.3.3 Interrogation par “ procuration ”

Le processus de recherche décrit au paragraphe précédent ne convient pas dans tous les cas, notamment vis à vis des deux critères suivants :

1. Sécurité d’un domaine
2. Conservation de la bande passante

#### 1. La figure

Romanchapter.05 montre le serveur local qui interroge directement les serveurs distants, cette démarche pose des problèmes de sécurité dans le cas d’un domaine au sein duquel seuls un ou deux serveurs sont autorisés.

Par exemple le serveur de noms du domaine **sio.ecp.fr** n'interroge pas directement le serveur racine, il passe par le serveur officiel qui est **piston.ecp.fr** (138.195.33.3).

**2.** Le trafic destiné au serveur de noms peut consommer une partie non négligeable de la bande passante, c'est pourquoi il peut être stratégique de concentrer les demandes vers un seul serveur régional et donc de bénéficier au maximum de l'effet de cache décrit précédemment.

## 2.4 Hiérarchie de serveurs

Si tous les serveurs de noms traitent de données d'un format identique, leur position dans l'arborescence leur confère un statut qui se nomme :

**serveur racine** (“ root name server ”) Un serveur ayant autorité sur la racine de l'espace de nommage. Actuellement il y a 13 serveurs de ce type, nommés [A-M].ROOT-SERVERS.NET<sup>14</sup>

**serveur primaire** (“ master ”) Un serveur de noms qui a l'autorité pour un ou plusieurs domaines (est détenteur d'autant de **SOA** – Voir page 183). Il lit ses données dans un fichier stocké sur disque dur, à son démarrage. L'administrateur du (des) domaine(s) met à jour les informations des domaines concernés depuis cette machine.

**serveur secondaire** (“ slave ”) Dans le cas d'une panne ou d'un engorgement du serveur primaire, les serveurs secondaires reçoivent en prévision une copie de la base de données.

- Stratégiquement il est préférable de les placer en dehors du domaine, sur le réseau d'un autre FAI. Il peut y avoir autant de serveurs secondaires que souhaité, de l'ordre de trois ou quatre est souvent rencontré.
- Au démarrage ils reçoivent les informations du serveur primaire, ou ils les lisent sur leur disque dur s'ils ont eu le temps de les y stocker au précédent arrêt du serveur, et si elles sont encore valides.

Par exemple, le serveur PISTON.ECP.FR a comme serveurs secondaires NS2.NIC.FR, SOLEIL.UVSQ.FR, MANOUL.CTI.ECP.FR.

## 2.5 Conversion d'adresses IP en noms

On dit aussi questions inverses (“ inverse queries ” vs “ reverse queries ”).

Cette possibilité est indiquée comme optionnelle dans la RFC 1034 mais est néanmoins bien commode voire même fréquemment requise pour le client réseau de services comme le courrier électronique, l'établissement de sessions à distance avec **ssh** ou même les serveurs de fichiers anonymes (**ftp**). Si une machine est enregistrée dans le TLD **in-addr.arpa**, c'est un indicateur favorable quant à la qualité d'administration du réseau qui l'héberge, mais ne prouve rien quant aux bonnes intentions de son (ses) utilisateur(s).

<sup>14</sup> fichier **named.root**, par exemple dans le répertoire **/etc/namedb**

Il faut ajouter que le bon usage sur les réseaux est de prévoir une entrée dans la zone reverse pour toutes les machines utiles et utilisées d'un réseau accessible de l'Internet. Le contraire provoque bien souvent la grogne (à juste titre) des administrateurs.

Il faut reconsidérer la *figure IX.01*. À gauche de la figure on distingue un domaine un peu particulier “ in-addr.arpa ”. Toutes les adresses sont exprimées dans le “ top level domain ” :

### **in-addr.arpa**

Du fait de la lecture inverse de l'arbre, les adresses IP sont exprimées en “ miroir ” de la réalité. Par exemple pour la classe B de l'ecp :

### **195.138.in-addr.arpa (Classe B 138.195)**

Le fonctionnement par délégation est calqué sur celui utilisé pour les noms symboliques (c'est la justification de son insertion dans la *figure ChapterRoman.01*). Ainsi, on peut obtenir la liste des serveurs ayant autorité sur la zone **195.138.in-addr.arpa** en questionnant d'abord les serveurs du TLD **in-addr.arpa** puis ceux pour la zone **138.in-addr.arpa**, et ainsi de suite...

Chaque administrateur de zone peut aussi être en charge de l'administration des “ zones reverses ”, portion du domaine “ arpa ”, des classes d'adresses dont il dispose pour nommer ses machines, s'il en reçoit la délégation. Il faut bien noter que cette délégation est une opération indépendante de celle qui a lieu pour les autres domaines.

Notons également que la notion de sous réseau (cf page 38) n'est pas applicable au domaine “ in-addr.arpa ”, ce qui signifie que les adresses selon les contours naturels des octets.

Ainsi, pour les clients de fournisseurs d'accès n'ayant comme adresses IP officielles que celles délimitées par un masque de sous réseau large seulement que de quelques unités (< 254), la gestion de la zone reverse reste du domaine du prestataire (FAI) et non du client.

## 2.6 Conclusion

### Qu'est-ce qu'un DNS ?

Un serveur de noms repose sur trois constituants :

1. Un espace de noms et une base de données qui associe de manière structurée des noms à des adresses IP.
2. Des serveurs de noms, qui sont compétents pour répondre sur une ou plusieurs zones.
3. Des “ resolver ” capables d'interroger les serveurs avec une stratégie définie par l'administrateur du système.

### TCP ou UDP ?

Le port 53 “ bien connu ” pour le serveur de noms est prévu pour fonctionner avec les deux protocoles.

- Normalement la majeure partie du trafic se fait avec UDP, mais si la taille d'une réponse dépasse les 512 octets, un drapeau de l'en-tête du protocole l'indique au client qui reformule sans question en utilisant TCP.
- Quand un serveur secondaire démarre son activité, il effectue une connexion TCP vers le serveur primaire pour obtenir sa copie de la base de données. En général, toutes les trois heures (c'est une valeur courante) il effectue cette démarche.

## 3 Mise à jour dynamique

La mise à jour dynamique de serveur de noms (RFC 2136) est une fonctionnalité assez récente sur le réseau, elle permet comme son nom l'indique de mettre à jour la base de donnée répartie.

Aussi bien au niveau du réseau local qu'à l'échelle de l'Internet il s'agit le plus souvent de faire correspondre un nom de machine fixe avec une adresse ip changeante. C'est typiquement le cas d'un tout petit site qui a enregistré son domaine chez un vendeur quelconque et qui au gré des changements d'adresse ip (attribuée dynamiquement par exemple avec DHCP<sup>15</sup>) par son fournisseur d'accès, met à jour le serveur de noms pour être toujours accessible.

Avec comme mot clef “ dyndns ”, les moteurs de recherche indiquent l'existence de sites commerciaux ou à caractère associatif, qui proposent cette fonctionnalité.

---

<sup>15</sup>Cf <http://www.isc.org/products/DHCP/>

## 4 Sécurisation des échanges

Le serveur de noms est la clef de voûte des réseaux, et c'est en même temps un de ses talons d'Achille parceque les programmes que nous employons quotidiennement utilisent sans discernement l'information acquise du réseau. En effet, qu'est-ce qui vous assure que le site web de votre banque sur lequel vous venez de taper votre mot de passe est bien le vrai site officiel de cet établissement ? L'apparence de la page de garde ?

Typiquement il y a deux situations de vulnérabilité :

1. Dialogue serveur à serveur, notamment lors de transferts de zones
2. Interrogation d'un serveur par un résolveur

Pour faire confiance en ce que vous dit le serveur de noms interrogé il faut d'une part que vous soyez certains d'interroger la bonne machine et d'autre part que celle-ci soit détentrice d'une information incontestable.

C'est une chaîne de confiance, comme toujours en sécurité, qui remonte par construction du fonctionnement du serveur de noms interrogé par votre application (comme nous l'avons examiné dans les paragraphes qui précèdent) jusqu'aux serveurs racines.

La version ISC (consultez le paragraphe 7) du programme BIND utilise deux stratégies différentes, selon les cas ci-dessus. Dans le premier cas il s'agit d'un mécanisme nommé TSIG/TKEY, dans le deuxième DNSSEC.

TSIG/TKEY utilisent une clef symétrique, donc partagée par les deux serveurs (cette clef leur est connue par des mécanismes différents). DNSSEC utilise un mécanisme basé sur le principe d'un échange de clefs publiques.

Outre les dysfonctionnements dûs à une information erronée on observe également des attaques de type “ déni de service<sup>16</sup> ” utilisant le fonctionnant intrinsèque du protocole (voir plus loin<sup>5</sup>).

### 4.1 TSIG/TKEY pour sécuriser les transferts

L'usage d'une clef symétrique indique qu'il s'agit d'un secret partagé entre 2 serveurs. La même clef sert au chiffrement et au déchiffrement des données. Le bon usage veut que l'on dédie une clef à un certain type de transaction (par exemple le transfert d'une zone) entre deux serveurs donnés. Cette manière de procéder se traduit donc rapidement par un grand nombre de clefs à gérer ce qui interdit un déploiement généralisé sur l'Internet.

Pour éviter de trop longs temps de chiffrement, ce ne sont pas les données à transférer qui sont chiffrées (de plus elles ne sont pas confidentielles), mais leur empreinte ( “ fingerprints ”) avec un algorithme de type MD5 ou SHA1<sup>17</sup>.

<sup>16</sup>[http://fr.wikipedia.org/wiki/D\u00f9nhbox\u00f9voidb\u00f9xbgroup\u00f9let\u00f9nhbox\u00f9voidb\u00f9x\u00f9setbox\u00f9@tempboxa\u00f9hbox\u00f9{e\u00f9global\u00f9mathchardef\u00f9accent@spacefactor\u00f9spacefactor\u00f9}\u00f9accent19e\u00f9egroup\u00f9spacefactor\u00f9accent@spacefactorni\\_de\\_service](http://fr.wikipedia.org/wiki/D\u00f9nhbox\u00f9voidb\u00f9xbgroup\u00f9let\u00f9nhbox\u00f9voidb\u00f9x\u00f9setbox\u00f9@tempboxa\u00f9hbox\u00f9{e\u00f9global\u00f9mathchardef\u00f9accent@spacefactor\u00f9spacefactor\u00f9}\u00f9accent19e\u00f9egroup\u00f9spacefactor\u00f9accent@spacefactorni_de_service)

<sup>17</sup>Ne pas hésiter à faire un `man md5` ou `man sha1` sur une machine Unix pour en savoir plus !



Cette empreinte, seule, est cryptée.

Le serveur qui reçoit un tel paquet signé, calcule l'empreinte du paquet avec le même algorithme, déchiffre celle jointe avec la clef secrète partagée et compare les deux empreintes. Le résultat de cette comparaison dit si les données sont valides ou non.

L'intérêt de ces transferts signés est que les serveurs secondaires sont certains de mettre à jour leur zones avec des données qui proviennent bien du détenteur du SOA et qui sont absolument semblables à ce qui a été émis.

#### 4.1.1 TSIG

TSIG comme " Transaction SIGNature " est la méthode décrite dans la RFC 2845 et basée sur l'usage d'une clef symétrique. La génération de cette clef peut être manuelle ou automatisée avec le programme " `dnssec-keygen` ".

La propagation de cette clef est manuelle (`scp`...Éviter absolument l'usage de tout protocole diffusant un mot de passe en clair sur le réseau), donc mise en place au coup par coup.

TSIG sert également à la mise à jour dynamique (" dynamic update "), la connaissance de la clef par le client sert à la fois à l'authentifier et à signer les données <sup>18</sup>.

#### 4.1.2 TKEY

TKEY, décrit dans la RFC 2930, rend les mêmes services que TSIG tout en évitant le transport du secret (TSIG). Cette caractéristique est basée sur le calcul la clef symétrique automatiquement avec l'algorithme de Diffie-Hellman plutôt que par un échange " manuel " <sup>19</sup>.

Par contre, cet algorithme à base du tandem clef publique – clef privée suppose l'ajout d'un champ **KEY** dans les fichiers de configuration du serveur. Comme d'ailleurs le mécanisme suivant...

## 4.2 DNSSEC pour sécuriser les interrogations

DNSSEC décrit dans la RFC 2535 permet :

1. La distribution d'une clef publique (champ **KEY**)
2. La certification de l'origine des données
3. L'authentification des transactions (transferts, requêtes)

Mis en place, le DNSSEC permet de construire une chaîne de confiance, depuis le " top level " jusqu'au serveur interrogé par votre station de travail.

---

<sup>18</sup>cf le programme `nsupdate` et la RFC 2136

<sup>19</sup>On peut trouver une explication de cet algorithme sur ce site : <http://www.rsasecurity.com/rsalabs/faq/3-6-1.html>

## 5 Attaque DNS par amplification

Le fonctionnement repose sur UDP, protocole pour lequel l'en-tête (page 84) est facilement falsifiable, notamment sur l'adresse de retour. Il est ainsi très facile d'envoyer une requête à un serveur, avec une adresse de retour qui est celle d'une machine victime plutôt que la sienne :

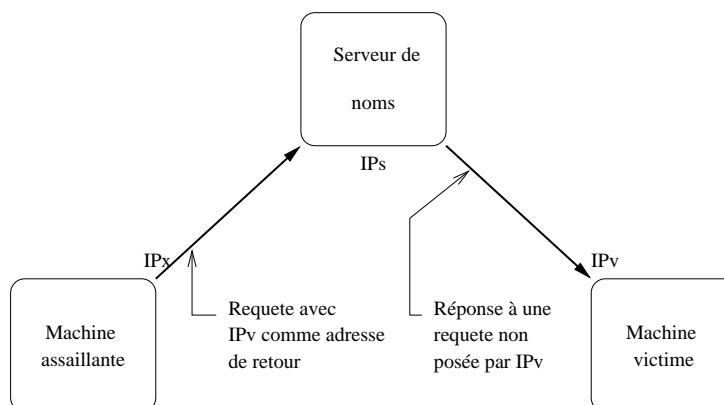


figure IX.06 — Réponse à une requête non formulée

Sur la figure IX.06 la machine d'adresse IPv reçoit un message du serveur de noms d'adresse IPs, non sollicité. Il est bien évident qu'un seul message de ce type reste sans effet, cependant :

1. Le volume en octets de la réponse peut être considérablement plus important que celui de la requête, notamment si le serveur de noms est configuré par l'assaillant. Par exemple d'un facteur 5 ou 10.
2. L'assaillant peut envoyer un très grand nombre de requêtes à des serveurs ouverts en mode récursif pour toute requête ne portant pas sur les domaines sur lesquels ils ont autorité.

La machine victime est alors submergée par un flot de réponses qui satureront complètement ses accès réseaux, c'est une attaque DNS par amplification<sup>20</sup> et qui provoque un déni de service sur le site qui la subit.

Le schéma d'ensemble d'une telle attaque est résumé sur la figure IX.07.

La machine assaillante (elles peuvent être nombreuses, des centaines de milliers) bombardent les serveurs (S1, S2, ... Sn) de fausses requêtes.

Ces serveurs sont utilisés parcequ'ils combinent deux propriétés intéressantes :

1. Ils sont ouverts aux requêtes extérieures même et surtout celles qui ne portent pas sur leurs données. Cette propriété est héritée de l'époque où l'Internet était encore un réseau d'universitaires et d'informaticiens. Cette propriété devrait tendre à disparaître, mais c'est loin d'être encore

<sup>20</sup><http://www.isotf.org/news/DNS-Amplification-Attacks.pdf>

le cas<sup>21</sup> puisque la configuration standard des outils l'autorise et que les compétences techniques ne sont pas assez nombreuses.

2. Ils utilisent un cache DNS. L'effet de ce cache est que même si la machine " source " est isolée du réseau, les enregistrements lus, pourvu qu'ils soient assortis d'un temps de vie suffisant (TTL, page 183) peuvent continuer d'être exploités.

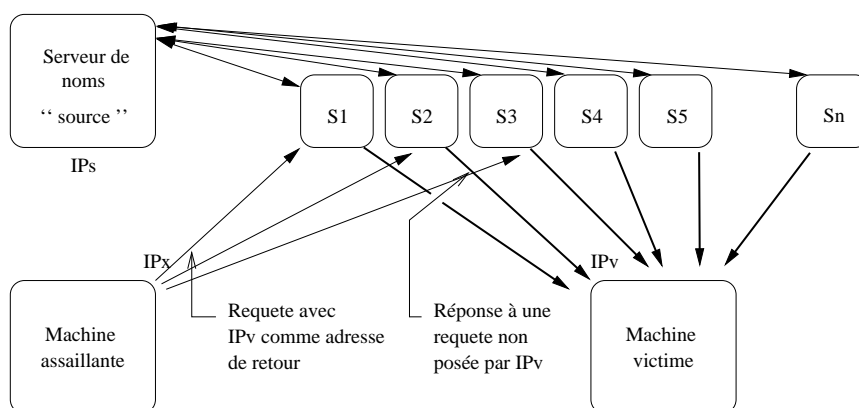


figure IX.07 — Attaque DNS par amplification

Quelques remarques :

1. Le serveur de noms " source " n'est pas nécessairement complice, c'est tout simplement un serveur avec de gros enregistrements.
2. Les serveurs S1 à Sn sont utilisés à leur insu mais une configuration soigneuse peut éviter cet abus d'usage.
3. Une fois attaqué le serveur victime ne peut pas faire grand chose. Ses services ne sont plus accessibles car le réseau est saturé.
4. La parade avec un serveur de type Bind de l'ISC (page 186) consiste à explicitement limiter l'ouverture extérieure du serveur aux seules données sur lesquelles il a autorité <sup>22</sup>.

L'accès aux données dans le cache doit également être protégé car d'autres techniques existent pour peupler les caches, par exemple envoyer un mail qui nécessite l'interrogation du serveur source.

<sup>21</sup>Un test sur son serveur depuis une machine hors de son réseau local est possible à cette url <http://dns.measurement-factory.com/cgi-bin/openresolvercheck.pl>

<sup>22</sup>Directives `allow-recursion` et `allow-cache` du fichier de configuration

## 6 Format des “ Resource Record ”

Comme pour toute base de données, le serveur de nom a un format pour ses champs, ou “ Resource Record ”, RR dans la suite de ce texte, défini à l’origine dans la RFC 1035.

En pratique toutes les distributions (commerciales ou libres) du serveur de noms conservent ce format de base de données, la mise en œuvre du serveur seule change (fichier de configuration du `daemon`).

Un serveur de noms a autorité (responsabilité du SOA) sur une ou plusieurs zones, celles-ci sont repérées dans ses fichiers de configuration (`named.conf` ou `named.boot` selon les versions). S’il est serveur primaire d’une ou plusieurs zones, le contenu de ces zones est inscrit dans des fichiers ASCII ; leur syntaxe est succinctement décrite dans le paragraphe suivant.

S’il est serveur secondaire, le fichier de configuration indique au serveur de quelle(s) zone(s) il est secondaire (il peut être secondaire d’un grand nombre de zones) et donc où (adresse IP) il devra aller chercher cette information. Cette action se traduit par ce que l’on nomme un “ transfert de zone ”. Ce transfert est effectué automatiquement à la fréquence prévue par l’administrateur du champ SOA et donc connue dès le premier transfert. En cas de changement sur le serveur principal, celui-ci avertit ( “ Notify ”) ses serveurs secondaires de la nécessité de recharger la zone pour être à jour.

Le propos de ce qui suit n’est pas de se substituer à une documentation nombreuse et bien faite sur le sujet, mais d’apporter quelques éléments fondamentaux pour en aider la lecture.

Le constituant de base d’un serveur de noms est une paire de fichiers ASCII contenant les enregistrements, les “ Resource Record ”.

Ceux-ci sont en général écrits sur une seule ligne de texte (sauf pour le champ SOA qui s’étale sur plusieurs lignes. Le marqueur de fin de ligne (CR+LF) est aussi celui de la fin de l’enregistrement. Le contenu général d’un tel enregistrement a la forme suivante (les accolades indique des données optionnelles) :

```
{name}      {ttl}      addr-class      Record Type      Record Specific data
```

Cinq enregistrements, ou “ Resource Record ”, ou en RR, sont absolument fondamentaux pour faire fonctionner un serveur de noms : SOA, NS, A, MX et PTR.

## 6.1 RR de type SOA

```

$(ORIGIN) sio.ecp.fr.
name      {ttl}      addr-class  SOA      Origin      Person in charge
@          IN         SOA         sio.ecp.fr.  hostmaster.sio.ecp.fr. (
                                2007100801    ; Serial
                                10800      ; Refresh (3h)
                                3600       ; Retry (1H)
                                3600000    ; Expire (5w6d16h)
                                86400     ) ; Minimum ttl (1D)

```

SOA est l'acronyme de “ **Start Of Authority** ” et désigne le début obligé et unique d'une zone. Il doit figurer dans chaque fichier `db.domain` et `db.adresse`. Le nom de cette zone est ici repéré par le caractère `@` qui signifie la zone courante, repérée par la ligne au dessus “ `$(ORIGIN) sio.ecp.fr.` ”.

La ligne aurait également pu s'écrire :

```
sio.ecp.fr. IN SOA sio.ecp.fr. hostmaster.sio.ecp.fr. (...)
```

Un problème concernant cette zone devra être signalé par e-mail à `hostmaster@sio.ecp.fr` (notez le “.” qui s'est transformé en “@”).

Les paramètres de ce **SOA** sont décrits sur plusieurs lignes, regroupées entres parenthèses. Le caractère “;” marque le début d'un commentaire, qui s'arrête à la fin de ligne.

Les points en fin de noms sont nécessaires.

Le numéro de série doit changer à chaque mise à jour de la zone (sur le serveur principal). Le **Refresh** indique la fréquence, en secondes, à laquelle les serveurs secondaires doivent consulter le primaire pour éventuellement lancer un transfert de zone (si le numéro de série est plus grand). Le **Retry** indique combien de secondes un serveur secondaire doit attendre un transfert avant de le déclarer impossible. Le **Expire** indique le nombre de secondes maximum pendant lesquelles un serveur secondaire peut se servir des données du primaire en cas d'échec du transfert. **Minimum ttl** est le nombre de secondes par défaut pour le champ **TTL** si celui-ci est omis dans les RR.

## 6.2 RR de type NS

Il faut ajouter une ligne de ce type (“ **Name Server** ”) pour chaque serveur de noms pour le domaine. Notez bien que rien dans la syntaxe ne permet de distinguer le serveur principal de ses secondaires.

Dans le fichier `db.domain` :

```

{name}      {ttl}      addr-class  NS      Name servers name
                                IN      NS      ns-master.sio.ecp.fr.
                                IN      NS      ns-slave1.sio.ecp.fr.
                                IN      NS      ns-slave2.sio.ecp.fr.

```

Dans le fichier qui renseigne la zone “ reverse ”, par exemple `db.adresse`, on trouvera :

```
52.195.138.in-addr.arpa. IN  NS  ns-master.sio.ecp.fr.
52.195.138.in-addr.arpa. IN  NS  ns-slave1.sio.ecp.fr.
52.195.138.in-addr.arpa. IN  NS  ns-slave2.sio.ecp.fr.
```

### 6.3 RR de type A

Le RR de type **A**, ou encore “ **Address record** ” attribue une ou plusieurs adresses à un nom, c’est donc celui qui est potentiellement le plus fréquemment utilisé. Il doit y avoir un RR de type **A** pour chaque adresse d’une machine.

```
{name}      {ttl}  addr-class  A  address
gw-sio                      IN  A  138.195.52.2
                        IN  A  138.195.52.33
                        IN  A  138.195.52.65
```

### 6.4 RR de type PTR

Le RR de type **PTR**, ou encore “ **PoinTeR record** ” permet de spécifier les adresses pour la résolution inverse, donc dans le domaine spécial **IN-ADDR.ARPA**. Notez le “.” en fin de nom qui interdit la complétion (il s’agit bien du nom FQDN).

```
name      {ttl}  addr-class  PTR  real name
2                      IN  PTR  gw-sio.sio.ecp.fr.
33                     IN  PTR  gw-sio.sio.ecp.fr.
65                     IN  PTR  gw-sio.sio.ecp.fr.
```

### 6.5 RR de type MX

Le RR de type **MX**, ou encore “ **Mail eXchanger** ” concerne les relations entre le serveur de noms et le courrier électronique. Nous examinerons son fonctionnement ultérieurement dans le chapitre sur le courrier électronique (cf page 205).

```
sio.ecp.fr.      IN  MX  10  smtp.ecp.fr.
sio.ecp.fr.      IN  MX  20  mailhost.laissus.fr.
```

## 6.6 RR de type CNAME

Le RR de type CNAME, ou encore “ **canonical name** ” permet de distinguer le nom officiel d’une machine de ses surnoms.

|                              |    |       |                                     |
|------------------------------|----|-------|-------------------------------------|
| <code>www.sio.ecp.fr.</code> | IN | CNAME | <code>msio-bipro.cti.ecp.fr.</code> |
|------------------------------|----|-------|-------------------------------------|

Dans l’exemple ci-dessus, la machine `www.sio.ecp.fr` est un surnom de la machine `msio-bipro.cti.ecp.fr`. Le fait que ces deux appellations soient dans la même zone (`ecp.fr.`) n’aide en rien au bon fonctionnement du dispositif. La machine `msio-bipro` pourrait être hébergée n’importe où ailleurs sur un autre réseau dans une autre zone...!

Cette possibilité est très employée pour constituer des machines virtuelles, comme nous le verrons au chapitre VIII.

## 6.7 Autres RR...

Il existe d’autres RR, entres autres HINFO , TXT, WKS et KEY, non traités dans cette présentation parcequ’ils n’apportent rien à la compréhension du fonctionnement du serveur de noms. Le lecteur est fortement incité à se reporter au “ Name Server Operations Guide ” pour plus d’informations.

## 7 BIND de l'ISC

L'Internet Software Consortium<sup>23</sup> est une organisation non commerciale qui développe et favorise l'emploi de l'outil " Open Source " comme BIND (acronyme de " Berkeley Internet Name Domain ").

Cette version libre du serveur de nom est la plus employée sur les machines Unix du réseau, ce qui justifie que l'on s'y intéresse. Elle fournit une version du *daemon* " *named* " et un " *resolver* " intégré dans la *libc*. On peut aisément installer ce logiciel sur à peu près toutes les implémentations d'unix connues (cf le fichier *INSTALL* du répertoire *src*).

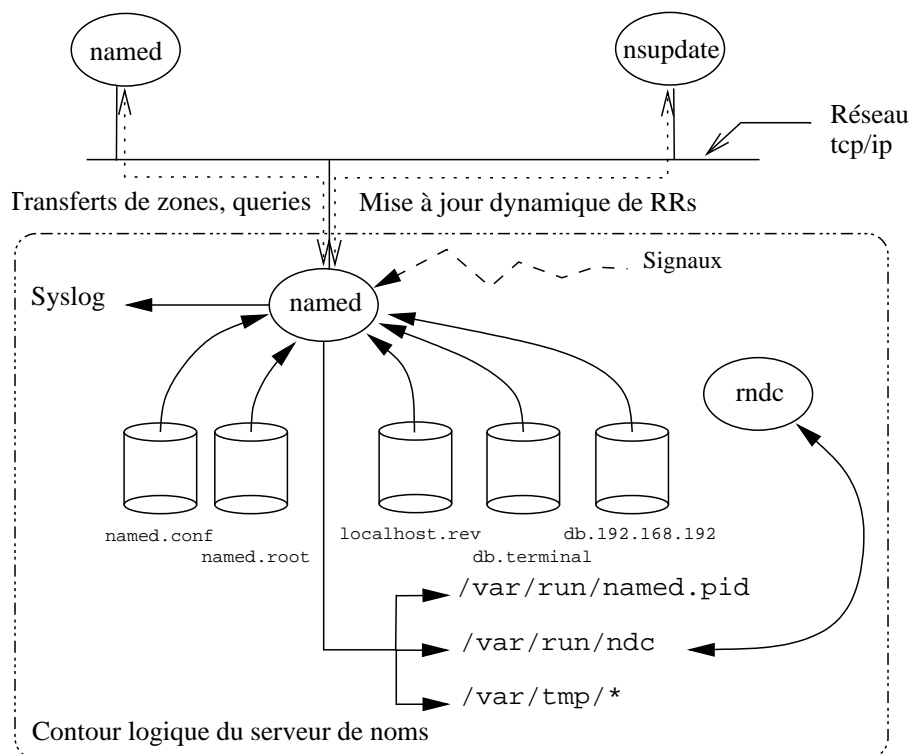


figure IX.08 — BIND de l'ISC

### 7.1 Architecture du daemon " named "

La figure IX.06 montre le schéma général de l'organisation logicielle du daemon " named ".

Au démarrage celui-ci lit sa configuration dans un fichier qui peut se nommer *named.boot* ou *named.conf* selon que l'on est en version 4.9.11, 8.3.6 ou 9.2.9 et les suivantes du logiciel<sup>24</sup>.

**named.conf** C'est le fichier de configuration lu au démarrage. Sa structure dépend de la version du logicielle, heureusement dans les deux cas la sémantique reste proche !

<sup>23</sup><http://www.isc.org/>

<sup>24</sup>taper " *named -v* " pour les discerner entre-elles



**named.root** Ce fichier contient la liste des serveurs de la racine, leur nom et adresse IP.

**localhost.rev** Ce fichier contient la base de donnée du “ localhost ”. Personne ne possède en particulier le réseau 127, donc chacun doit le gérer pour lui-même. L’absence de ce fichier n’empêche pas le serveur de fonctionner, mais ne lui permet pas de résoudre 127.0.0.xx (où xx est le numéro de la machine courante, souvent 1).

**db.terminal** Exemple de fichier de base de données pour le domaine factice **terminal.fr** qui est utile durant les travaux pratiques. Ce fichier permet la conversion des noms en adresses IP.

**db.192.168.192** Ce fichier contient la base de données de la zone “ reverse ” pour le domaine **terminal.fr**, c’est à dire le fichier qui permet au logiciel de convertir les adresses IP en noms.

**rndc** ou “ **name server control utility** ” est comme son nom l’indique un outil d’administration du programme **named** lui-même. C’est une alternative à l’usage direct des signaux. Le canal de communication entre les deux programmes est une socket unix **AF\_UNIX** vs **AF\_LOCAL** (cf cours de programmation page 251).

Un certain nombre de signaux modifient le comportement du serveur, ils seront examinés en travaux pratiques, tout comme les fichiers lus ou écrits dans les répertoires **/var/run** et **/var/tmp/**.

Enfin la flèche vers **syslog** signifie que **named** utilise ce service pour laisser une trace de son activité (cf cours sur l’architecture des serveurs).

Enfin, le BOG, c’est à dire le “ Bind Operations Guide ”, détaille le contenu des champs de la base de données des versions 4.x et 8.x. Pour la version 9.x est distribuée avec “ BIND 9 Administrator Reference Manual ” une documentation également très bien faite.

## 8 Bibliographie

Quand on “ sait déjà ”, la page de man de “ **named** ” suffit à vérifier un point obscur ! Sinon il existe une documentation très fournie sur le sujet, avec notamment :

- Kein J. Dunlap & Michael J. Karels — “ Name Server Operations Guide ” — Ce document est accessible sur le serveur de l’Internet Software Consortium <sup>25</sup>
- By the Nominum BIND Development Team — “ BIND 9 Administrator Reference Manual ” — Version 9.1.3 <sup>26</sup>
- Douglas E. Comer — “ Internetworking with TCP/IP – Volume I ” (chapter 18) — Prentice All — 1988

<sup>25</sup>On trouve le BOG dans la distribution de “ bind ” à cette adresse : <http://www.isc.org/products/BIND/>

<sup>26</sup>on trouve également la dernière version de ce document sur le site de l’ISC

- Paul Albitz & Cricket Liu — “ DNS and BIND ” — O'Reilly & Associates, Inc. — 1992
- “ Installing and Administering ARPA Services ” — Hewlett Packard — 1991
- W. Richard Stevens — “ TCP/IP Illustrated Volume I ” (chapter 14) — Prentice Hall — 1994

Et pour en savoir encore plus...

**RFC 1034** “ Domain names - concepts and facilities ”. P.V. Mockapetris. Nov-01-1987. (Format : TXT=129180 bytes) (Obsoletes RFC0973, RFC0882, RFC0883) (Obsoleted by RFC1065, RFC2065) (Updated by RFC1101, RFC1183, RFC1348, RFC1876, RFC1982, RFC2065, RFC2181) (Status : STANDARD)

**RFC 1035** “ Domain names - implementation and specification ”. P.V. Mockapetris. Nov-01-1987. (Format : TXT=125626 bytes) (Obsoletes RFC0973, RFC0882, RFC0883) (Obsoleted by RFC2065) (Updated by RFC1101, RFC1183, RFC1348, RFC1876, RFC1982, RFC1995, RFC1996, RFC2065, RFC2181, RFC2136, RFC2137) (Status : STANDARD)

**RFC 1101** “ DNS encoding of network names and other types ”. P.V. Mockapetris. Apr-01-1989. (Format : TXT=28677 bytes) (Updates RFC1034, RFC1035) (Status : UNKNOWN)

**RFC 1123** “ Requirements for Internet hosts - application and support ”. R.T. Braden. Oct-01-1989. (Format : TXT=245503 bytes) (Updates RFC0822) (Updated by RFC2181) (Status : STANDARD)

**RFC 1713** “ Tools for DNS debugging ”. A. Romao. November 1994. (Format : TXT=33500 bytes) (Also FYI0027) (Status : INFORMATIONAL)

**RFC 2136** “ Dynamic Updates in the Domain Name System (DNS UPDATE) ”. P. Vixie, Ed., S. Thomson, Y. Rekhter, J. Bound. April 1997. (Format : TXT=56354 bytes) (Updates RFC1035) (Updated by RFC3007) (Status : PROPOSED STANDARD)

**RFC 2535** “ Domain Name System Security Extensions ”. D. Eastlake 3rd. March 1999. (Format : TXT=110958 bytes) (Obsoletes RFC2065) (Updates RFC2181, RFC1035, RFC1034) (Updated by RFC2931, RFC3007, RFC3008, RFC3090, RFC3226, RFC3445) (Status : PROPOSED STANDARD)

**RFC 2845** “ Secret Key Transaction Authentication for DNS (TSIG) ”. P. Vixie, O. Gudmundsson, B. Wellington. May 2000. (Format : TXT=32272 bytes) (Updates RFC1035) (Status : PROPOSED STANDARD)

**RFC 2930** “ Secret Key Establishment for DNS (TKEY RR) ”. D. Eastlake 3rd. September 2000. (Format : TXT=34894 bytes) (Status : PROPOSED STANDARD)

# Chapitre X

## Courrier électronique

### 1 Généralités sur le courrier électronique

Le courrier électronique, ou “ mail ” est l’un des deux services les plus populaires sur le réseau, avec le web.

C’est aussi l’un des plus vieux services du réseau, bien avant que le réseau existe sous la forme que l’on pratique aujourd’hui<sup>1</sup>. La préface de la [RFC 822], document fondamental parmi les documents fondamentaux pour ce chapitre, laisse supposer l’existence de nombreux formats d’échanges électroniques sur l’Arpanet, et ce avant 1977.

Sa popularité repose sur sa grande souplesse et rapidité d’emploi. Il permet aussi bien les échanges professionnels que les échanges privés ; son mode d’adressage donne la possibilité d’envoyer un courrier à une personne comme à une liste de personnes ou encore à un automate capable de rediffuser vers un groupe ( “ mailing-list ” ).

De nombreux outils développés, à l’origine essentiellement sur le système Unix, autour de ce concept ouvrent un vaste champs de possibilités aux utilisateurs de tous les systèmes d’exploitation, comme la ventilation des courriers par thème, le renvoi automatique, le répondeur (pendant les absences), l’accès à sa boîte aux lettres depuis des endroits différents, la réception de fax, . . . La liste ne peut pas être exhaustive !

C’est souvent pour avoir l’usage du courrier électronique que les entités (s’il en reste) non encore reliées à l’Internet franchissent le pas. L’usage des autres services arrivent plus tard, si besoin est.

---

<sup>1</sup>Un historique intéressant <http://www.fnet.fr/history/>

## 1.1 Métaphore du courrier postal - L'enveloppe

Un courrier postal (ou de surface, “ s-mail ”) a fondamentalement besoin de l'adresse du destinataire et de l'adresse de l'émetteur (pour la réponse). L'usage du timbre et de l'enveloppe répondent à d'autres critères.

Une fois dans la boîte aux lettres, l'enveloppe est routée de la poste locale vers la poste la plus proche du destinataire, pour être finalement délivrée par un facteur.

**Pour un courrier électronique les besoins sont quasiment identiques !**

Le concept d'enveloppe est conservé, il s'agit de l'adresse de l'émetteur du courrier et de celle(s) du (des) destinataire(s), propagées de manière bien séparée du corps du message afin que le protocole SMTP qui joue le rôle du service postale (Voir page 195) puisse router et finalement délivrer le courrier à son (ses) destinataire(s).

Il existe de très nombreux outils pour lire/écrire un mail, des outils pour jouer le rôle du bureau de poste et/ou du facteur. Sous Unix le facteur est le système lui-même, le bureau de poste un programme nommé “ sendmail<sup>2</sup> ”. Il existe d'autres alternatives non abordées dans ce document, comme le programme “ qmail<sup>3</sup> ” ou encore le programme “ postfix<sup>4</sup> ”.

## 1.2 Adresse électronique

Tous les courriers électroniques ont un destinataire précisé par son adresse électronique, ou “ E-mail<sup>5</sup> ”. Celle-ci précise le nom du destinataire et le site où il reçoit son courrier électronique.

Le nom du destinataire est une chaîne de caractères. Traditionnellement et pour des raisons techniques, sur le système Unix, le login de l'utilisateur peut être également le nom de sa boîte aux lettres. Cette possibilité est de moins en moins vraie à mesure que d'autres systèmes avec d'autres logiques de fonctionnement existent également sur le réseau (notamment la lecture du mail via un interface `html` ou encore lorsque le mail est délivré directement dans une base de données et non délivré dans un fichier).

Par exemple, il est assez fréquent de voir employer le nom complet (prénom et nom de famille) pour désigner l'interlocuteur distant. La conversion ultime entre cette convention et la boîte aux lettres de l'utilisateur est l'affaire du “ bureau de poste le plus proche ”, c'est à dire le programme “ sendmail ” pour ce document (voir plus loin au paragraphe 4).

Le caractère “ @ ” (lire “ at ”) sépare l'identificateur du destinataire de la destination.

---

<sup>2</sup>Version 8.13.5 en septembre 2005 — <http://www.sendmail.org/>

<sup>3</sup><http://www.qmail.org/>

<sup>4</sup><http://postfix.eu.org/start.html>

<sup>5</sup>Terme francisé en “ mèl ”, ou “ couriel ” pour les documents administratifs... ; -)

La destination est peut être vide (il s'agit alors d'un destinataire sur la machine courante, ou d'un synonyme (" alias ") que le sendmail local sait traiter), être un nom de machine du domaine local, le nom d'un autre domaine ou d'une machine sur un autre domaine.

Les adresses suivantes ont un format valide :

**user1** Destinataire local.

**user2@nom\_de\_machine** Destinataire sur une machine du domaine courant (rappelez-vous, il existe un mécanisme de complétion dans le " resolver " page 170!).

**user3@nom\_de\_machine.domaine** Destinataire sur une machine particulière d'un domaine particulier (non forcément local).

**user4@domaine** Destinataire sur un domaine particulier (même remarque que ci-dessus).

On devine aisément que le fonctionnement du courrier électronique sur une machine distante est fortement liée au bon fonctionnement du serveur de noms (chapitre IX).

Qui plus est, lorsque seul un nom de domaine est précisé à droite du caractère " @ ", une information manque apparemment quant à la machine susceptible de recevoir le mail.

Le lecteur en quête de plus de précisions trouvera une description exhaustive de la syntaxe d'une adresse au paragraphe 6 de la [RFC 822].

## 2 Format d'un " E-mail " - RFC 822

Les octets qui composent un courrier électronique obéissent à une structure bien définie par la [RFC 822] de David H. Crocker : un en-tête et un corps de message, séparés par une ligne blanche (deux CRLF<sup>6</sup> qui se suivent).

Le contenu de l'en-tête dans son intégralité n'est pas toujours spontanément montré par les outils qui nous permettent de lire et d'envoyer du courrier électronique. Une option est toujours accessible pour ce faire, comme " h " sous **mutt** <sup>7</sup>.

Une partie de l'en-tête est générée automatiquement par le programme qui se charge du transfert (le paragraphe suivant nous dira qu'il s'agit d'un MTA), une autre est ajoutée par le programme qui permet de composer le mail, le MUA, une autre enfin est tapée par l'utilisateur lui-même.

L'en-tête est constitué de lignes construites sur le modèle :

**identificateur : [ valeur ] CRLF**

---

<sup>6</sup> Il s'agit respectivement des caractères 13 et 10 de la table ASCII - cf " **man ascii** "

<sup>7</sup> <http://www.mutt.org/> — le MUA (Mail User Agent) préféré de l'auteur :-)

L'**identificateur** ne peut pas contenir le caractère “ : ” parcequ’il sert de séparateur avec la partie droite. Il est constitué de caractères ASCII codés sur 7 bits et imprimables (c’est à dire comprise dans le segment [33,126]), excepté l’espace.

**Valeur** est optionnelle. L’usage des majuscules ou des minuscules est indifférencié.

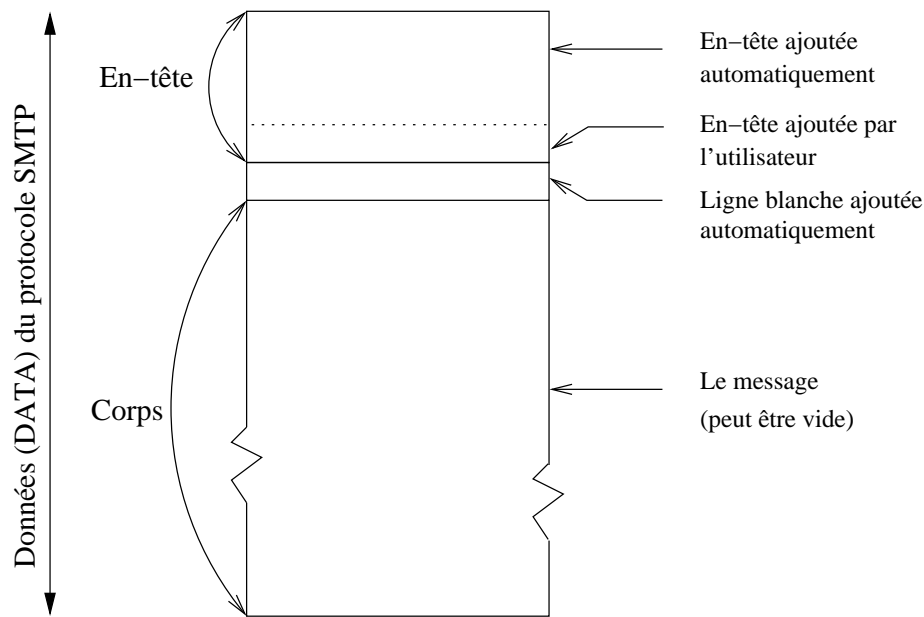


figure X.01 — Format d'un e-mail

L’ordre d’apparition de ces champs est quelconque. Seule l’organisation de la *figure X.01* doit être globalement respectée. Le lecteur soucieux d’une description exhaustive de ce en quoi peut être constitué un en-tête pourra se reporter au paragraphe 4.1 de la RFC (SYNTAX).

Certains champs de l’en-tête proviennent de la configuration du MTA, d’autres sont créés en interne par le MTA lui-même, d’autres enfin sont gérées par le MUA, donc accessible à l’utilisateur final.

2.1 Quelques champs couramment rencontrés dans les en-têtes

| Type d'information          | Noms des champs                         |
|-----------------------------|---|
| Destinataire(s) du courrier | To, Cc, Bcc                             |
| Origine du courrier         | From, Reply-To                          |
| Identification du courrier  | Message-ID                              |
| Cheminement du courrier     | Received, Priority                      |
| Nature du contenu           | Content-Transfer-Encoding, Content-Type |
| Divers                      | Date, Subject                           |
| Champs étendus              | X-★                                     |

**To (The primary recipients)** Il s'agit du (des) destinataire(s) principaux du message ("recipient"). Ce champ peut éventuellement être vide, le MTA prend alors une décision paramétrable pour le compléter. La valeur **undisclosed-recipient** ; est courante dans ce cas.

**Cc (Carbon copy)** Ce champ est dans le fonctionnement un doublon de **To**, mais l'usage en nuance le sens : c'est une copie pour information qui est transmise au(x) destinataire(s) listé(s).

**Bcc (Blind carbon copy)** Une copie du message est transmise au(x) destinataire(s) listé(s), sans que les destinataires principaux des champs **To** et **Cc** en soit informés.

**From (The sender)** il s'agit de l'émetteur du message. Le plus souvent il s'agit d'une seule personne, quand ce champ en liste plusieurs (le séparateur est la virgule " , ") le champ **Sender** doit préciser l'adresse de celui qui a effectivement envoyé le message.

**Reply-To (Alternative reply address)** Ce champ précise une adresse alternative à celle du champ **From** pour l'envoi de la réponse. Cette disposition est utilisée par les robots de gestion des mailing-list, pour distinguer l'auteur du message et le destinataire de la réponse.

**Message-ID (Unique identifier for message)** Ce champ est censé identifier de manière unique le message. Il est fabriqué dès sa soumission au premier MTA (MSA). Il est constitué traditionnellement de la manière suivante :

`nombre_de_secondes.identificateur_de_queue@domaine`

`nombre_de_secondes` Correspond à la date courante en secondes calculées depuis le 01/01/1970<sup>8</sup>

`identificateur_de_queue` Identifie la queue locale sur laquelle ce mail est déposé en entrée.

`domaine` C'est le domaine d'émission du message.

Exemple : **Message-ID:** <20051104121857.GC44788@laissus.fr>

**Received (Trace routing of mail)** C'est une trace du routage suivi par le message, depuis sa soumission jusqu'à sa délivrance finale. Chaque MTA ajoute un champ de ce type. Le cheminement est à suivre en commençant en fin de l'en-tête.

Chaque champ est constitué au minimum de **from**, le nom canonique de la machine de qui le MTA a reçu le message, de **by** le nom canonique du MTA qui a reçu le message et ajouté ce champ et enfin de la date de la transaction.

---

<sup>8</sup> " Epoch " pour les unixiens !

Exemple :

```
Received: from mailhost.sio.ecp.fr (mailhost.sio.ecp.fr [138.195.52.34])
  by leopard.ecp.fr (Postfix) with ESMTP id A6C1D37C91
  for <fr.laissus@laissus.fr>; Thu, 3 Nov 2005 13:56:58 +0100 (CET)
```

**Priority (Determine timeouts in the queue)** En fonction d'une valeur qui est **urgent**, **normal** ou **non-urgent** les messages qui ne peuvent être délivrés immédiatement sont placés dans une file d'attente dont la date d'expiration est d'autant plus courte que le message est **urgent**. L'émetteur du message reçoit d'abord un avertissement puis une erreur si le message n'est pas délivré quand arrive la date d'expiration.

**Content-Transfer-Encoding (Auxiliary MIME encoding)** Indique comment est encodé le corps du message pour supporter les caractères hors jeu ascii 7 bits (SMTP ne transporte que des caractères 7 bits). Des valeurs courantes sont `base64`, `quoted-printable`, `8bit`,....

**Content-Type (The nature of the body of the message)** Ce champ indique comment est constitué le corps du message. Par défaut il est supposé être constitué que de caractères 8 bits dont le bit de poids fort est sans signification (7 bits effectifs).

Pour écrire les caractères accentués du français, par exemple il faut avoir un champ de cette forme `Content-Type: text/plain; charset=ISO-8859-1`

Dans ce cas, le corps du message ne contient que du texte. Le cas contraire est celui d'un message qui contient des pièces jointes, une balise introduite en supplément dans l'en-tête va servir à séparer les différentes parties du message comme dans :

`Content-Type: multipart/mixed; boundary="opJtzjQTFsWocga"`

La chaîne " `opJtzjQTFsWocga` " sert alors de marqueur pour repérer chaque partie du mail (corps du message et pièces jointes).

**Date (The origin date)** C'est la date à laquelle le message a été envoyé initialement. Ce champ est obligatoire.

**Subject (Topic of the message)** C'est une courte chaîne de caractères qui résume le message. Les MUA montre ce champ pour permettre une meilleure sélection des messages avant de les lire.

**MIME-Version (This message conforms to MIME standards)** Niveau de MIME pour l'encodage du corps de message (voir page 197).

**X-★** C'est un en-tête spécifiquement ajouté par le MUA ou par un processus de la chaîne de traitement du courrier. Un exemple parmi tellement d'autre :

`X-Greylist: Sender IP whitelisted, not delayed by milter-greylist-2.0.2`

Ajouté par un mécanisme extérieur au MTA, qui agit contre le spam, et nommé le Greylisting<sup>9</sup>

<sup>9</sup><http://projects.puremagic.com/greylisting/whitepaper.html>



## 3 Protocole SMTP - RFC 821

Le protocole SMTP, ou “ Simple Mail Transfer Protocol ” a comme objet le transport du courrier électronique de manière fiable et efficace. Il est défini dans la [RFC 821] de Jonathan B. Postel.

Indépendant par sa conception d’un quelconque sous-système de transport, il est principalement aujourd’hui encapsulé dans des paquets TCP à destination du port 25 (cf le fichier /etc/services). Dans un passé pas si lointain l’accès réseau de beaucoup de sites se résumait au courrier électronique encapsulé dans des trames du protocole UUCP<sup>10</sup>, donc sur liaison série via modem !

### 3.1 Protocole SMTP

SMTP est un protocole ASCII (7 bits, “ human readable ”). La partie cliente de la transaction se connecte sur le port 25 du serveur et envoie des commandes auxquelles le serveur répond par des codes numériques qui indiquent le statut de la prise en compte de la commande.

C’est pourquoi il est aisé de se connecter sur un MTA avec un simple `telnet`<sup>11</sup> :

```
$ telnet localhost 25
Connected to localhost.
Escape character is '^]'.
220 host.mondomain.fr ESMTP Sendmail 8.12.6; Mon, 20 Jan 2003 15:34:45 +0100 (CET)
NOOP
250 2.0.0 OK
QUIT
221 2.0.0 host.mondomain.fr closing connection
Connection closed by foreign host.
```

Dans cet exemple le MTA est le programme `Sendmail`<sup>12</sup>, qui répond à la connexion par un code 220 pour dire que le service est opérationnel (“ service ready ”), suivi du nom de la machine, de la bannière du programme, de la version de sa configuration, et de sa date courante.

Puis l’utilisateur a tapé la commande `NOOP` qui n’a d’autre effet que de forcer le serveur à répondre et renvoyer un code (250) pour dire que tout va bien.

Enfin L’utilisateur a tapé `QUIT` pour finir proprement la transaction. La réponse du serveur est un code 221 pour signifier la fin canonique de la connexion.

---

<sup>10</sup>[http://fr.wikipedia.org/wiki/Unix\\_to\\_Unix\\_Copy\\_Protocol](http://fr.wikipedia.org/wiki/Unix_to_Unix_Copy_Protocol)

<sup>11</sup>Attention toutefois de ne pas abuser de cette pratique car de nombreuses attaques de sites ont démarré par le passé à l’aide d’un détournement de `sendmail`. Les administrateurs réseaux sont donc attentifs au trafic sur le port 25 ; il est préférable de réserver ce genre de tests uniquement sur son propre site.

<sup>12</sup><http://www.sendmail.org/>

Dans un deuxième essai nous utilisons l'option `-v` du programme `mail`, pour visualiser les échanges entre le MUA (machine `athome.mondomain.fr`) et le MTA local (machine `mailhub.mondomain.fr`).

Essayons :

```
$ sendmail -v user@mondomain.fr
Subject: test
Ca passe ?

.          <<<--- A taper pour marquer la fin du mail dans ce mode.
EOT
user@mondomain.fr... Connecting to mailhub.mondomain.fr. via relay...
220 mailhub.mondomain.fr HP Sendmail (1.37.109.4/user-2.1) ready at Mon,
26 Jan 98 14:08:57 +0100
>>> HELO athome.mondomain.fr
250 mailhub.mondomain.fr Hello athome.mondomain.fr, pleased to meet you
>>> MAIL From:<user@mondomain.fr>
250 <user@mondomain.fr>... Sender ok
>>> RCPT To:<user@mondomain.fr>
250 <user@mondomain.fr>... Recipient ok
>>> DATA
354 Enter mail, end with "." on a line by itself
>>> .
250 Ok
user@mondomain.fr... Sent (Ok)
Closing connection to mailhub.mondomain.fr.
>>> QUIT
221 mailhub.mondomain.fr closing connection
```

Et le courrier reçu, lu aussi avec `mail` :

```
Message 208/208 User Lambda                               Jan 26, 98 02:09:06 pm +0100

From user@mondomain.fr Mon Jan 26 14:09:08 1998
Received: from mailhub.mondomain.fr by athome.mondomain.fr with SMTP (8.8.7/8.8.7/f
la.2.1) id OAA27655; Mon, 26 Jan 1998 14:09:08 +0100 (CET)
Received: from athome.mondomain.fr by mailhub.mondomain.fr with SMTP (1.37.109.4/fl
a-2.1) id AA06996; Mon, 26 Jan 98 14:08:57 +0100
From: User Lambda <user@mondomain.fr>
Received: by athome.mondomain.fr
Date: Mon, 26 Jan 1998 14:09:06 +0100 (CET)
Message-Id: <199801261309.OAA27653@athome.mondomain.fr>
To: user@mondomain.fr
Subject: test
```

Ca passe ?

Manifestement, ça passe ! :)

Il est également intéressant d'observer à ce niveau que les caractères du courrier ont été considérablement enrichis par un en-tête volumineux (relativement).

En effet, chaque nœud traversé (MTA), ajoute un champ "Received" permettant après coup de suivre le trajet du courrier. Les autres champs

comme `Date :`, `Subject :` ou `Message-Id :` sont ajoutés dans l'en-tête par le MUA de l'écrivain du message.

Cette partie de l'en-tête ajoutée par le MUA d'origine est souvent destinée à piloter le comportement du MUA du destinataire du message plus que pour être lue. Cette attitude s'est généralisée au point de devenir assez compliquée et être formalisée dans un ensemble de règles baptisées MIME comme " Multipurpose Internet Mail Extensions " ([RFC 2184]).

La fonction la plus répandue et la plus simple de MIME est d'autoriser l'usage des caractères accentués (codage sur 8 bits ou plus) à l'intérieur du corps du message (l'en-tête SMTP reste codée sur 7 bits). L'utilisateur voit alors apparaître des lignes supplémentaires comme celles-ci :

```
X-Mime-Autoconverted: from 8bit to quoted-printable by bidule.domain id SAA23150
X-MIME-Autoconverted: from quoted-printable to 8bit by mamachine.ici id QAA29283
```

Le "quoted-printable" est une forme possible du codage des caractères accentués, définie dans la [RFC 822]. Le plus souvent on trouve des lignes comme celles-ci :

```
Mime-version: 1.0
Content-Type: text/plain; charset=ISO-8859-1
Content-Transfert-Encoding: quoted-printable
Content-ID: Content-ID: <Pine.FBSD.3.14.1592654.19971998X.domain>
Content-Description: arlg.c
```

D'autres formes de MIME peuvent conduire à l'exécution d'un programme extérieur au MUA, ce qui constitue une dangereuse faille potentielle dans la sécurité des réseaux, donc à éviter.

## 3.2 Principales commandes de SMTP

Expérimentalement nous avons découvert quelques uns des mots réservés du protocole : `HELO`, `MAIL`, `RCPT`, `DATA`, `QUIT`. Une implémentation minimale de SMTP en comprend deux autres en plus : `RSET` et `NOOP`. C'est donc un protocole assez simple, du moins dans sa version de base.

Les codes de retour sont organisés sur trois chiffres, le premier chiffre donne le sens général de la réponse, très succinctement ce qui débute par 1,2 ou 3 a une signification positive, 4 ou 5 signifie une erreur. Une information plus détaillée se trouve à l'annexe E de la RFC.

Les cinq commandes découvertes précédemment s'utilisent toujours dans cet ordre. Examinons succinctement leur usage :

### 3.2.1 Commande HELO

**Synopsis :** `HELO` <espace> <domaine> <CRLF>

Cette commande est utilisée pour identifier l'émetteur du mail. L'argument qui suit, **domain** est le nom de la machine ou du domaine d'où provient la connexion.

En réponse le serveur envoie une bannière dans laquelle il s'identifie et donne la date courante. Cette information est optionnelle, ce qui compte c'est le code de retour pour confirmer l'aptitude au travail du serveur !

### 3.2.2 Commande MAIL

**Synopsis : MAIL <espace> FROM : <chemin inverse> <CRLF>**

Cette commande débute un transfert de mail. En argument sont transmis (**chemin inverse**) l'adresse e-mail de l'émetteur et la liste des machines qui ont relayé le mail courant. La première machine est la plus récente. Cette information est utilisée pour renvoyer, s'il y a lieu, une notification de problème à l'émetteur du mail.

Par exemple :

```
MAIL FROM:<@mailhub.ici:@mailhost.labas:Lambda@mondomain.fr>
```

### 3.2.3 Commande RCPT

**Synopsis : RCPT <espace> TO : <destinataire> <CRLF>**

Cette commande est la deuxième étape dans la procédure d'envoi d'un mail. Il y a une commande de ce type par destinataire du courrier (" recipient ").

Par exemple :

```
RCPT TO:<Lambda@mondomain.fr>
```

Il est intéressant de noter que les arguments de cette commande et ceux de la précédente (MAIL) forment **l'enveloppe du mail** (expéditeur et destinataire) comme nous en avons signalé l'existence conceptuelle page 190.

### 3.2.4 Commande DATA

**Synopsis : DATA <CRLF>**

Après réception de la commande, le serveur lit les lignes de texte en provenance du client jusqu'à rencontrer la séquence **<CRLF>.<CRLF>** qui marque la fin du message. Il faut remarquer que celui-ci comprend l'intégralité de la *figure X.01*.

### 3.2.5 Commande QUIT

**Synopsis : QUIT <CRLF>**

Marque la fin de la session et entraîne la clôture de la connexion.

### 3.3 Propagation du courrier électronique

SMTP est défini comme un protocole de transfert, donc un moyen pour router et délivrer le message à son (ses) destinataire final (finaux).

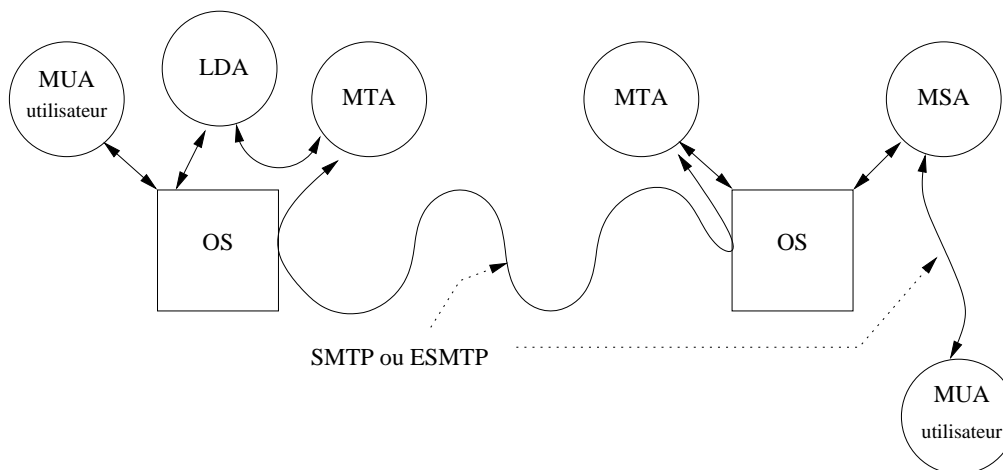


figure X.02 — MUA - MSA - MTA - LDA - OS

**MUA** “ Mail User Agent ” ou encore “ mailer ”, c’est le programme qui permet de lire et écrire le corps du courrier et de paramétrer quelques éléments de l’en-tête, principalement l’adresse du destinataire, et le sujet du message.

Il existe un très grand nombre de MUAs sous Unix, il est courant de rencontrer `mail`, `mailx`, `elm`, `pine`, `mutt`, `mh`, `eudora`, `kmail`, `thundermail`, `sylpheed`... Il y en a pour tous les goûts !

**MSA** “ Mail Submission Agent ”, c’est une “ nouveauté ” définie par la [RFC 2476] et qui joue le rôle d’interface entre le MUA et le MTA.

L’objet du MSA est de séparer les fonctions de transfert du courrier et d’acceptation de nouveaux courriers émis depuis les MUA. Cette séparation des tâches améliore deux aspects :

**La sécurité** Les nouveaux mails sont soumis à un `daemon` qui ne n’exécutent pas avec les droits du `root`<sup>13</sup>.

**La conformité aux standards** Les messages proviennent de MUAs qui ne respectent pas forcément tous les prérequis de formulation des en-têtes.

Le rôle du MSA est de vérifier et de compléter ces en-têtes avant de soumettre les mails au MTA pour le routage.

**MTA** “ Mail Transfer Agent ”, c’est le programme que prend en charge le transfert du courrier. Sous Unix c’est un `daemon`. Par exemple : `MMDF`, `Smail`, `Zmailer`, `sendmail`, `postfix`, `qmail`...

<sup>13</sup>SUID bit

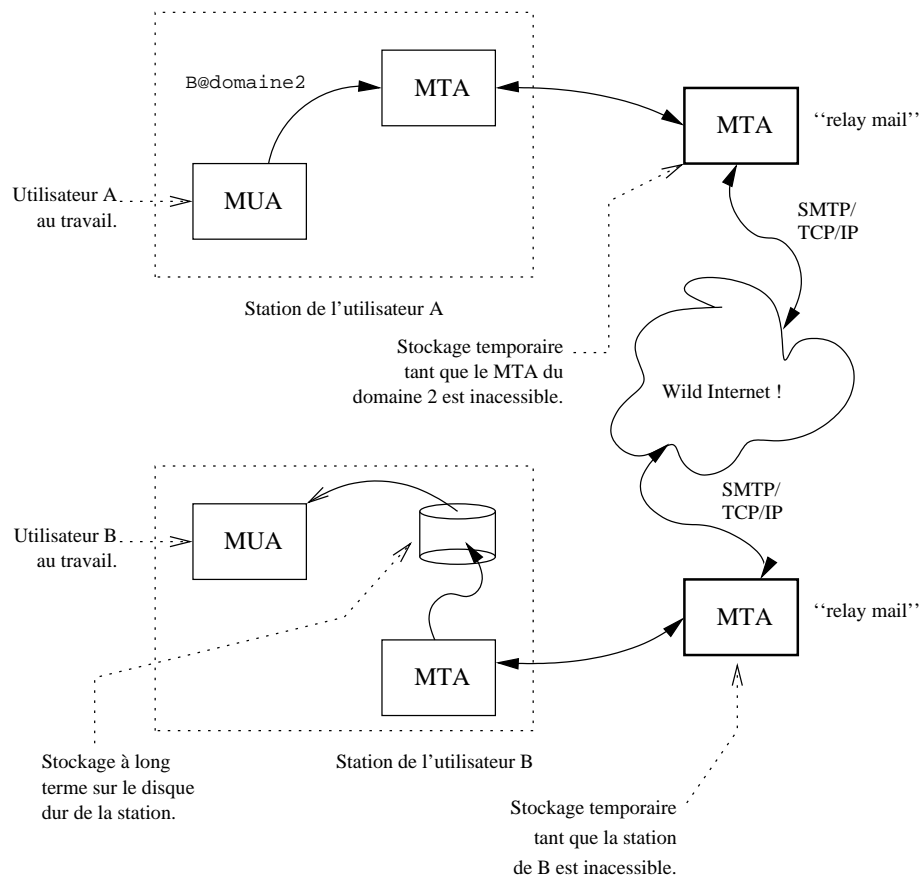
Les MTAs écoutent le réseau sur le port 25 et dialoguent entre-eux avec le protocole SMTP (ESMTP<sup>14</sup>).

**LDA** “ Local Delivery Agent ”, c’est l’entité qui délivre effectivement le mail, soit dans une boîte au lettres soit dans une base de données, par exemple une base Cyrus<sup>15</sup>.

**OS** “ Operating System ”, le système d’exploitation sur lequel fonctionnent ces programmes.

La *figure X.2* illustre la possibilité la plus simple d’échange entre deux MTA : la connexion directe. Cela signifie que le MTA de la station émettrice contacte le MTA de la station réceptrice et lui délivre directement le message.

La vie “ réelle ” est plus compliquée car elle tient compte de l’organisation hiérarchique des réseaux et surtout de la sécurité qui est un aspect devenu important sur l’Internet. Cela se traduit par un emploi généralisé de machines relais ou “ mailhub ”.



*figure X.03 — Trajet d'un mail*

Une telle machine concentre tous les courriers électroniques d’un site, vers l’extérieur et inversement. Elle a des avantages, notamment :

<sup>14</sup>Extended SMTP

<sup>15</sup><http://asg.web.cmu.edu/cyrus/>

- Avoir une politique de sécurité concentrée sur un petit nombre de machines exposées<sup>16</sup>, plutôt que sur toutes les stations du réseau : le routeur filtrant n'autorise les accès extérieurs que sur le port smtp(25) de ces quelques machines dédiées.
- Avoir une politique centralisée pour le filtrage des contenus indésirables (virus) et des émetteurs suspects (spams).
- Limiter le nombre de configurations compliquées de `sendmail` à un petit nombre de machines. Les stations des utilisateurs peuvent se contenter d'une configuration standard plus facile à distribuer et à adapter automatiquement.
- Permettre de masquer plus facilement les machines internes du réseau vis à vis de l'extérieur. En clair, les courriers auront l'air de provenir de cette machine plutôt que de la station d'un utilisateur sur le réseau interne.

L'adresse de l'émetteur aura la forme :

`user@domaine`

au lieu de :

`user@machine.domaine.`

- Permettre le stockage intermédiaire du courrier en attente d'une délivrance : les stations des utilisateurs ne sont pas toujours en fonctionnement.

Cette architecture est théorique, en pratique il peut y avoir une hiérarchie de “ relay mail ” plus compliquée. Par exemple une grappe de machines distinctes suivant que le courrier entre ou sort du site, une arborescence de machines relais quand l'entreprise est elle-même répartie sur plusieurs sites géographiques et ne possède qu'une liaison vers l'extérieur,...

### 3.4 Courriers indésirables - Le spam

*Le spam est l'aspect très désagréable du courrier électronique.*

Par “ spam ” on désigne ces innombrables courriers, le plus souvent à caractère commercial, qui envahissent nos boîtes aux lettres électroniques. Certaines estimations tablent sur au moins 30% de spam dans le trafic mail mondial et cette estimation est régulièrement revue à la hausse.

Deux questions se posent, comment le caractériser et surtout comment l'éviter ?

#### 3.4.1 Caractériser le spam

1. Un contenu commercial, publicitaire, financier, ou qui tente de retenir l'attention du lecteur à partir d'une histoire dont l'issue est toujours pécuniaire et au détriment du destinataire.

---

<sup>16</sup>Ce qui n'exclue pas bien entendu d'avoir une politique de sécurité pour le mail sur le réseau interne

2. Une importante liste de destinataires. Le champ `Cc` : peut contenir par exemple des centaines de destinataires.
3. Un en-tête de message truqué. Par exemple le champ `Message-ID` : qui est censé identifier le message de manière unique est absent ou incohérent (page 193).
4. Un grand nombre d'exemplaires du même message envoyé dans un court laps de temps. Cette caractéristique ne concerne pas le contenu du mail mais la manière avec laquelle il est envoyé. C'est le MTA qui reçoit les demandes de connexions qui peut détecter cette caractéristique.
5. Utilisation de l'adresse d'un destinataire sans son consentement explicite pour ce type d'envoi.
6. Usage d'un site "open mail relay" pour l'émission. L'émetteur du mail peut alors usurper le nom du domaine d'émission.  
Pour mémoire, un site "open mail relay" autorise un `RCPT` qui ne désigne pas un destinataire pour lequel la délivrance des mails est autorisée sur ce site. Le site sert alors en quelque sorte de tremplin pour le mail, avec un effet de dissimulation du site émetteur véritable. Les versions modernes des MTA interdisent cette possibilité par défaut.
7. Champs `To` et `From` de l'en-tête invalides  
Comme par exemple l'utilisation comme adresse e-mail d'origine du mail celle d'un utilisateur n'ayant strictement rien à voir avec le message (c'est l'enveloppe qui compte pour la délivrance du mail).
8. Le contenu du mail contient un virus, soit dans le corps du message soit dans une pièce jointe. Par abus ce genre de mail est parfois traité comme du spam.

### 3.4.2 Éviter le spam

C'est une question ouverte...

S'il est évident que l'œil humain reconnaît un tel courrier de manière quasi infaillible, il n'en est pas de même pour la machine.

Il n'existe pas de méthode de lutte unique et infaillible. Un bon résultat (plus de 99% du trafic de spam bloqué) peut cependant être atteint en passant par l'usage d'une palette d'outils et de dispositifs extérieurs. Toutefois l'ensemble de ce dispositif a un coût non négligeable en terme de cycles cpu consommés pour un mail délivré, d'une part, et d'autre part en terme de coût de maintenance car l'architecture logicielle qui en découle est complexe et demande du temps de spécialiste de bon niveau pour sa maintenance.

Tout d'abord, une bonne partie de l'origine du spam vient du fait que le protocole SMTP lui-même est beaucoup trop permissif.

Conçu à une époque où le réseau était constitué sur la base de la confiance et de la collaboration entre sites honorables, il n'est plus adapté à ce qu'est devenu le réseau. La faille la plus navrante est que l'enveloppe transmise lors



de l'échange protocolaire n'est pas nécessairement identique à ce qui figure dans l'en-tête du mail. L'exemple qui suit illustre cette faille :

```
telnet mailhost.mondomain.fr 25
Connected to mailhost.mondomain.fr.
Escape character is '^]'.
220 mailhost.mondomain.fr ESMTP
HELO UnSiteQuelconque.com
250 mailhost.mondomain.fr Hello UnSiteQuelconque.com, pleased to meet you
MAIL From:<>
250 2.1.0 <>... Sender ok
RCPT To:<lambda@mondomain.fr>
250 2.1.5 <lambda@mondomain.fr>... Recipient ok
DATA
354 Enter mail, end with "." on a line by itself
From:<NePasRepondre@XXX.com>
To:<TuPeuxToujoursEssayer@YYY.com>

Unsolicited Bulk Email (UBE) or
Unsolicited Commercial Email (UCE).
.
```

Et le mail sera délivré dans la boîte aux lettres de l'utilisateur " lambda " avec des champs From : et To : complètement inexploitable !

Pour éviter la délivrance de ce mail, la configuration du MTA de mondomain.fr pourrait mettre en place **une protection agissant en trois temps : lors de l'établissement de la connexion, lors de la réception de l'enveloppe puis à la réception du message lui-même.**

Le protocole SMTP ne comprend pas d'accusé de réception. Si l'honnête utilisateur de ce protocole envoie un courrier routé silencieusement, par erreur, dans une boîte aux lettres de courriers indésirables (en général non lus et souvent supprimés automatiquement), c'est regrettable et d'autant plus préjudiciable que le contenu du mail est important. Il est donc bien plus efficace de refuser le message tant que la connexion avec le MTA qui l'émet est maintenue, car quel que soit le contenu de l'enveloppe (les **Reply-to** et **From** sont peut être faux ou inexistant), le MTA qui route ce mail est à l'autre bout de la connexion et est supposé, par construction, être le mieux placé pour prévenir l'auteur du mail que la transaction actuelle est refusée.

Autrement dit, il est bien préférable d'effectuer le tri en temps réel plutôt qu'en temps différé lors de la délivrance dans la boîte de l'utilisateur ou après rapatriement (voir page 210) des mails par son MUA.

Si l'origine du mail est honnête, l'émetteur sera tout de suite prévenu (mail d'erreur) et pourra agir en conséquence, dans le cas contraire le spammeur réceptionnera autant de messages d'erreurs que de spams envoyés (un rêve...) ce qui ne manquera pas de le gêner considérablement. Nous ne le plaindrons pas.

**À l'établissement de la connexion** le MTA peut vérifier que le taux de demandes de connexions n'excède pas un certain ratio (par exemple pas plus de 30 connexions TCP par minutes).

Les éléments de la connexion (voir page 90) fournissent l'adresse IP et le port d'origine.

L'adresse IP doit pouvoir être résolue dans le tld `in-addr.arpa`, le cas contraire est rédhibitoire.

L'adresse IP peut être tout simplement réfutée localement, tout comme le domaine (au sens du DNS).

L'adresse IP peut être également réfutée après interrogation des DNSBL ( " Domain Name Services BlackList ") qui sont des bases de données de sites reconnus comme étant à l'origine de spams ou connus comme " open mail relay ".

**À la réception de l'enveloppe** le MTA peut demander une authentification de l'émetteur (login et mot de passe) du mail.

Le MTA peut également consulter une base locale de champs `From` et `To` refusés.

Il existe un mécanisme assez ingénieux et récent, nommé le " GreyListing<sup>17</sup> " qui stoppe bon nombre de spams en spéculant sur le fait que les spammeurs sont des gens pressés et que leur mails sont envoyés en très grand nombre (des centaines de milliers d'unités) et le plus vite possible. En conséquence, si l'établissement du protocole SMTP ne fonctionne pas du premier coup, la plupart d'entre eux se découragent et ne réessaient pas (contrairement à ce que le protocole SMTP prévoit). Ce dispositif consiste donc à faire patienter tout le monde (sauf peut être une liste d'adresses de correspondants ou de domaines réputés fiables) et seuls ceux qui respectent le protocole à la lettre finissent par pouvoir transmettre leur mail, d'autant plus que s'ils ont patienté une fois ils sont placés dans une liste d'accès sans délai pendant un temps programmable (par exemple 24h). En pratique la connexion est coupée après émission d'un message d'erreur qui invite à recommencer ultérieurement.

**À la réception du corps du mail** des filtres peuvent être appliqués sur l'en-tête pour en vérifier la consistance, sur le corps du message pour réagir sur la présence de tels ou tels mots clefs (de tels filtres ont en général un fonctionnement statistique basé sur l'apprentissage d'une base de spams et d'une base de non-spams et sont enrichis continuellement avec les messages indésirables) enfin les pièces jointes sont extraites du courrier et examinées à l'aide d'un dispositif de reconnaissance des virus (une base de virus doit être mise à jour très régulièrement).

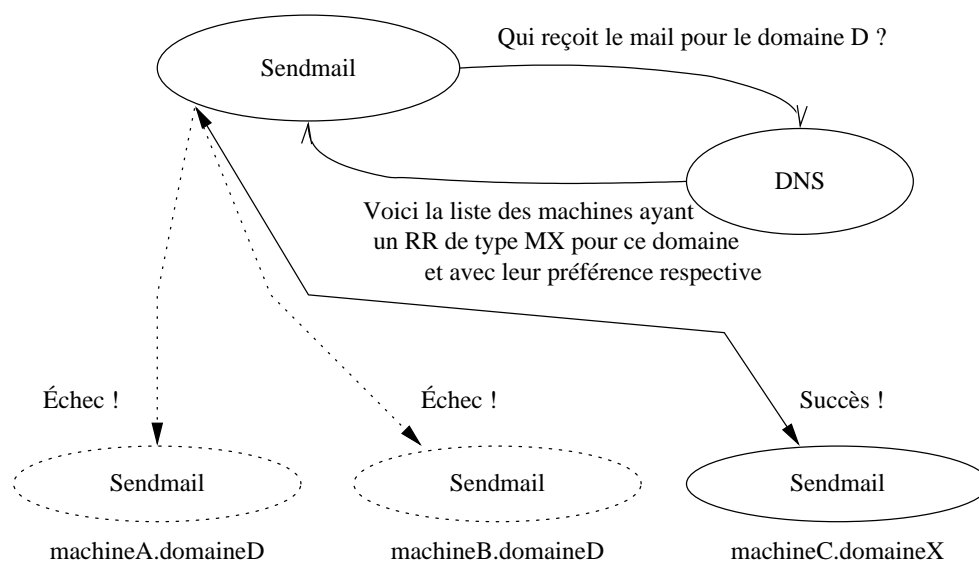
---

<sup>17</sup><http://projects.puremagic.com/greylisting/whitepaper.html>

## 4 Exemple de MTA - “ Sendmail ” et son environnement

### 4.1 Relations avec le DNS

Comme nous l’avons évoqué au paragraphe 1.2 page 190, la relation entre le MTA et le DNS est étroite. Sendmail a besoin du serveur de noms pour les opérations suivantes :



MTA pouvant récupérer le courrier pour le domaine D

figure X.04 — MX primaire et secondaires

1. Transformer le nom de la machine distante en adresse IP
2. Canoniser le nom de la machine locale.
3. Canoniser le nom de la machine qui se connecte
4. Déterminer quelles sont les machines susceptibles de recevoir du courrier pour le domaine à atteindre.

Le quatrième point est le plus crucial. Si le DNS du domaine à atteindre (une adresse est toujours mise sous la forme “ nom@domain ”) ne désigne pas de machine capable de recevoir le courrier, le mail ne passera jamais pour ce domaine.

Le champ RR (“ Resource Record ”) correspondant est du type MX (“ Mail Exchanger ”). Il spécifie une liste d’hôtes pondérés par des préférences, à qui on peut envoyer du courrier. La pondération indique l’ordre à suivre pour les tentatives de connexions : il faut commencer par la valeur la plus basse. Si cette liste est explorée de bout en bout sans succès il y a échec de la transmission du courrier.

S'il y a échec de la réémission, le mail est conservé un certain temps, puis est finalement rejeté s'il y a persistance de l'échec. Le résultat est matérialisé dans un fichier nommé `dead.letter`

*Figure X.4*, Le contenu du champ RR renvoyé par le DNS pourrait avoir la constitution suivante :

```

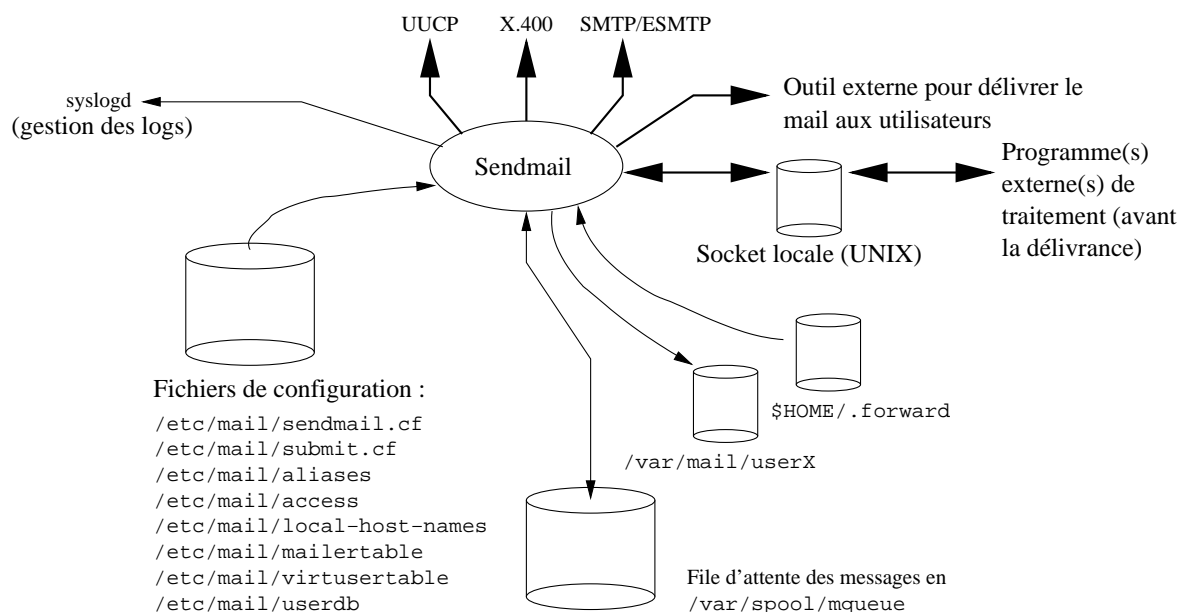
;
;[name]      [ttl][class] MX preference      mail exchanger
;
domaineD      IN    MX      10      machineA.domaineD
               IN    MX      20      machineB.domaineD
               IN    MX      30      machineC.domaineX

```

Il est important de remarquer qu'une machine baptisée " MX " par le DNS n'est pas forcément localisée dans le domaine pour lequel elle reçoit le courrier, c'est même souvent le cas pour les machines " relay ". C'est le cas de la troisième ligne, `machineC.domaineX`

## 4.2 Relations avec le système d'exploitation

Sendmail a de multiples relations avec le système d'exploitation. La *figure X.5* en fait un résumé :



*figure X.05 — Relation entre Sendmail et le système d'exploitation*

L'activité opérationnelle de `sendmail` est consignée à l'aide de `syslog`<sup>18</sup>, ce qui explique la présence de la ligne suivante trouvée dans le fichier `/etc/syslog.conf` : `mail.info /var/log/maillog`

<sup>18</sup>Chapitre III du cours de programmation — Éléments de serveurs

UUCP, X.400, SMTP Sont autant de moyens de propager le courrier. Ces supports peuvent cohabiter au sein d'une même configuration ; autant de " Mailer " sélectionnés en fonction de l'adresse du destinataire (cf `sendmail.cf`).

`/etc/mail/sendmail.cf` Est le fichier de configuration de `sendmail` qui fonctionne en tant que MTA. Sa présence est indispensable.

La configuration standard livrée est en générale à adapter aux impératifs de fonctionnement du réseau local. Voir à ce propos le paragraphe 5 page 212.

`/etc/mail/submit.cf` Est le fichier de configuration de `sendmail` en tant que MSA. Sa présence est optionnelle si le fichier précédent indique explicitement le contraire.

`/etc/mail/aliases` est une base de synonymes. Quand `sendmail` reçoit un courrier, il tente de reconnaître le nom du destinataire dans cette base et si c'est le cas de lui appliquer la transformation prescrite.

Un certain nombre d'alias sont requis par la [RFC 1123], d'autres sont conseillés par la [RFC 2142]. Un court extrait du-dit fichier :

```
# Basic system aliases -- these MUST be present
MAILER-DAEMON: postmaster
postmaster: root
# Well-known aliases -- these should be filled in!
root:          user
info:          root
marketing:     root
sales:         root
support:       root
www:           webmaster
webmaster:     root
...
```

Cela signifie par exemple que chaque fois qu'un courrier est envoyé au " postmaster " de ce site, `sendmail` transforme " postmaster " en " root ", puis " root " en " user ". Si cette dernière chaîne ne fait pas l'objet d'une autre transformation par cette table, il s'agit d'un utilisateur de la machine courante.

L'entretien de la table des alias est de la responsabilité de l'administrateur de la machine. La table des alias d'un domaine est un fichier stratégique qu'il convient de mettre à jour soigneusement (droits d'accès, utilisateurs inexistantes, boucles...).

À chaque changement dans cette table l'administrateur doit fabriquer une table d'accès rapides (" hash table ") à l'aide de la commande " `sendmail -bi` " souvent liée à " `newaliases` "

`/etc/mail/access` C'est un fichier qui regroupe des autorisations spécifiques d'accès ou de rejet des mails entrants. Par exemple :

|                       |        |
|-----------------------|--------|
| MonNouveauDomain.tld  | RELAY  |
| Connect:microsoft.com | REJECT |

Acceptera de relayer tout mail pour le domaine MonDomain.tld mais rejettera tout mail en provenance du domaine suivant.

**/etc/mail/local-host-names** Ce fichier collecte tous les noms sous lesquels la machine qui exécute le MTA est connue ce qui évite que celui rejette le mail en refusant de le relayer.

**/etc/mail/mailertable** Un MTA peut accepter d'effectuer le routage du courrier pour un grand nombre de domaines. Ce fichier permet d'effectuer un routage en fonction du domaine, par exemple :

|                     |                        |
|---------------------|------------------------|
| un.autre.domain.tld | smtp:autre.machine.tld |
| mon.domain.a.moi    | local:                 |

**/etc/mail/virtusertable** Ce fichier permet d'effectuer des réécritures d'adresses d'un domaine vers un autre domaine avec plus de possibilités d'expression que le fichier des aliases.

Par exemple :

|                                 |                              |
|---------------------------------|------------------------------|
| @MonAncienDomaine.tld           | %1-old@MonNouveauDomaine.tld |
| webmaster@MonNouveauDomaine.tld | wbm-new@AutreDomaine.tld     |

Dans la première ligne le %1 est remplacé dynamiquement par tout ce qui précède le @ de @MonAncienDomaine.tld ce qui permettra par exemple d'effectuer un tri au moment de la délivrance des messages, entre ceux envoyés pour le nouveau domaine et l'ancien.

**/etc/mail/userdb** Cette table, "User Database" permet au sendmail d'effectuer une traduction de chaîne sur les noms des utilisateurs pour les courriers sortants. Cette disposition permet de traduire un nom de login en "Prenom.Nom", donc d'avoir une adresse de retour de la forme "Prenom.Nom@domaine" ce qui fait toujours plus chic!

**/var/spool/mqueue** Dans ce répertoire sont stockés les mails en attente d'une délivrance. Il peuvent y rester plusieurs jours (c'est un paramètre de la configuration du sendmail), on peut visualiser cette file d'attente avec la commande **mailq**.

Si un grand nombre de courriers sont en attente, et ça peut être le cas pour les machines relais, la section du disque dur qui supporte cette partition (ici **/var**) doit faire l'objet d'un dimensionnement en conséquence, sous peine d'obliger **sendmail** à refuser les mails faute de place disque.

**/var/mail/userX** Chaque utilisateur de la machine locale (il peut ne pas y en avoir sur un serveur) a une boîte aux lettres ("mail box") repérée comme un fichier ayant comme nom son login. Par exemple **/var/spool/mail/root**.

Ce fichier est mis à jour automatiquement par le MTA local en cas d'arrivée de courrier.

De même que pour le répertoire de file d'attente, le répertoire des boîtes aux lettres des utilisateurs doit faire l'objet d'une attention particulière de la part de l'administrateur ; la prolifération des “ documents attachés ” aux courriers électroniques est un fléau contre lequel il est difficile de se prémunir sauf à agrandir perpétuellement la taille de la partition `/var... ! : (`

`${HOME}/.forward` Avant d'être finalement délivré dans la boîte aux lettres de l'utilisateur, `sendmail` lit le contenu de ce fichier, `${HOME}` étant le répertoire racine des fichiers de l'utilisateur en question.

Le fichier `.forward` est la base personnelle d'alias pour chaque utilisateur, ça permet de renvoyer son courrier vers d'autres sites, voire aussi d'effectuer des transformations avant de stocker les mails (`procmail`).

Si le `.forward` contient la chaîne suivante : `moi@ailleurs.tld` Tous les courriers envoyés à cette utilisateurs sont renvoyés à l'adresse indiquée.

Ou encore : `"|exec /usr/local/bin/procmail"`

Qui permet d'invoquer l'usage du programme `procmail`, celui-ci est un très puissant filtre qui permet de faire un tri des courriers électroniques avec des expressions régulières (indispensable pour gérer de multiples abonnements à des “ mailing-lists ”). Par exemple, avec la configuration `virtusertable` ci-dessus, pour forcer la délivrance des mails adressés à `@MonAncienDomaine.tld` dans un fichier spécial plutôt que la boîte par défaut, on pourrait écrire un fichier `.procmailrc` de configuration contenant les lignes suivantes :

```
:0 H:
* ^To[ :]+.*-old@MonNouveauDomaine.tld
${HOME}/Mail/Rouges
```

**Socket locale** Sendmail peut communiquer avec des programmes extérieurs par le biais d'une socket locale (UNIX). Le dialogue est facilité par une bibliothèque nommée `MILTER`<sup>19</sup> liée à `sendmail` lors de sa compilation.

L'idée est qu'il est plus intéressant de refuser éventuellement un mail dès que les premiers éléments du protocole SMTP (ou ultérieurement en examinant le corps du message) sont connus, plutôt que d'attendre que la connexion soit close et que le mail soit délivré. De ce fait, l'émetteur, quel qu'il soit (honnête utilisateur ou spammeur) sera immédiatement prévenu que son mail est refusé et pourra agir en conséquence.

De nombreux outils sont capable de fonctionner avec `sendmail` et son interface `MILTER`, liste non exhaustive : `MIMEDefang`<sup>20</sup>, `Clamav`<sup>21</sup>, `milter-greylist`<sup>22</sup>,...

<sup>19</sup><http://www.milter.org>

<sup>20</sup><http://www.mimedefang.org/>

<sup>21</sup><http://www.clamav.net/>

<sup>22</sup><http://hcpnet.free.fr/milter-greylist/>

**Outil externe de délivrance** Le message prêt à être délivré est confié par `sendmail` aux bons soins d'un programme extérieur. Si la délivrance s'effectue dans une boîte aux lettres unix (un fichier au format mailbox qui porte comme nom le login de l'utilisateur et est situé généralement dans le répertoire `/var/mail/`), ce programme se nomme `local.mail` en standard. Il peut être remplacé par d'autres, notamment par le programme `procmail` déjà cité, si on souhaite effectuer un filtrage supplémentaire à ce niveau du traitement, par exemple pour mettre dans une boîte aux lettres spéciale les mails considérés comme étant du spam en laissant à l'utilisateur le soin de les détruire par lui-même.

Enfin on peut remarquer qu'aucun signal n'est prévu pour indiquer à `sendmail` qu'il faut relire son fichier de configuration, c'est voulu par le concepteur. Lors de la mise au point de ce fichier, il faut arrêter puis le relancer manuellement<sup>23</sup>.

### 4.3 Le cas de POP

POP est l'acronyme de “ Post Office Protocol ”, il permet l'accès à un serveur de courrier depuis des clients PC sous Windows, voire même des stations unix distantes, par exemple via `ppp`, qui ne sont pas configurées pour faire un trafic SMTP entrant. POP dans sa version 3 est défini par la [RFC 1939].

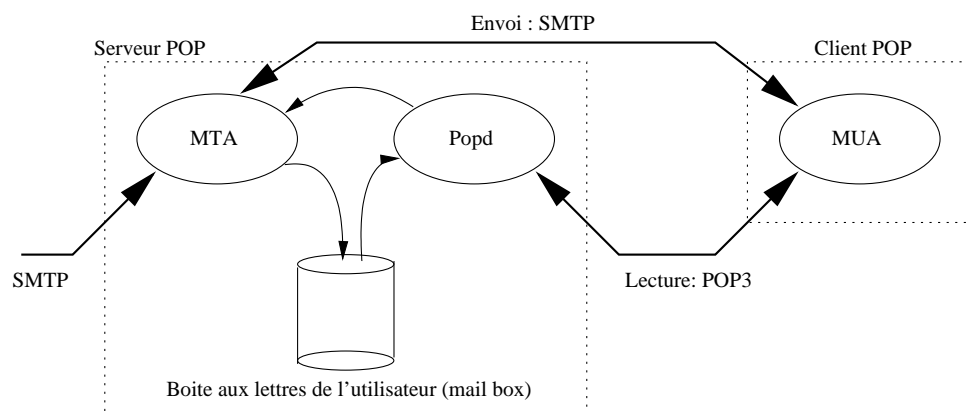


figure X.06 — Le cas de POP

Les clients POP sont légions sur les PCs et sur les stations de travail sous Unix. Pour celles-ci citons : `kmail`<sup>24</sup>, `mh`, `pine`, `elm`, `mutt`, `gnus` pour `emacs`, ou encore `sylpheed` et `thunderbird`. La liste n'est pas exhaustive, loin s'en faut.

<sup>23</sup>Par exemple sur une machine NetBSD/FreeBSD en tapant `/etc/rc.d/sendmail restart`

<sup>24</sup>De l'environnement de bureau KDE - <http://www.kde.org/>



POP est un protocole très simple qui fonctionne parfaitement mais qui n'est pas dénué de défauts :

1. L'authentification (login/password) est bien souvent échangée en “ clair ” sur le réseau
2. Sur l'architecture Unix, l'utilisateur doit avoir un compte sur la machine serveur (Une base de données des utilisateurs est toutefois possible)
3. Les messages doivent être récupérés sur le poste client pour être manipulés (en POP3 un double peut rester sur le serveur)
4. La boîte aux lettres ne peut être consultée que par un seul client à la fois

Ces points deviennent vite rédhibitoires quand le poste client doit accéder au serveur au travers d'un réseau non sûr, et surtout lorsque le détenteur de la boîte aux lettres veut consulter ses mails depuis des postes différents. Ce cas de figure est de plus la réalité de toute personne qui se déplace et souhaite un contact mail permanent avec ses correspondants.

C'est pourquoi d'autres solutions se sont développées et sont de plus en plus utilisées : les messageries accessibles via un browser web (webmail), donc qui utilisent comme support le protocole HTTP (voir page 327), ou un remplaçant du protocole POP, le protocole IMAP !

## 4.4 Le cas de IMAP

Bien que largement déployé que depuis quelques années, le protocole IMAP — “ Internet Message Access Protocol ” — a été développé à l'université de Stanford en 1986. C'est actuellement la version 4rev1 qui est utilisée, définie dans la [RFC 3501].

L'architecture présentée *figure X.06* reste valable, pratiquement il suffit d'y remplacer le mot “ POP ” par “ IMAP<sup>25</sup> ” !

Les fonctionnalités sont plus riches et surtout pallient aux inconvénients listés pour POP. En clair, les points négatifs listés pour POP sont tous réglés. Imap est conçu pour pouvoir accéder à ses boîtes aux lettres depuis de multiples machines, n'importe où sur le réseau, alors que POP est plus adapté à une machine unique.

Voici ses objectifs principaux :

1. Être compatible avec les standards, MIME par exemple
2. Permettre l'accès et la gestion des mails depuis plus d'une machine
3. Fournir un mode de fonctionnement en-ligne et hors-ligne
4. Supporter les accès concurrents aux mêmes boîtes aux lettres
5. Être indépendant du stockage des mails (fichiers ou base de données, par exemple)

---

<sup>25</sup>Consultez <http://www.imap.org/> pour plus d'informations

Un excellent comparatif des deux protocoles est accessible ici :

<http://www.imap.org/papers/imap.vs.pop.brief.html>

## 5 Configuration du Sendmail

Le programme `sendmail` s'appuie sur un ensemble de règles de réécritures (*figure X.05* page 206) par défaut regroupées dans les fichiers `/etc/mail/sendmail.cf` et `/etc/mail/submit.cf`.

La plupart des cas de la vie courante se traitent sans avoir besoin de modifier manuellement ces deux fichiers. L'usage d'un jeu de macros `m4`<sup>26</sup> très complet et puissant suffit pour générer les configurations ci-dessus, après l'écriture d'un fichier de requêtes de quelques lignes. `M4` génère ensuite les fichiers attendus à partir de ces requêtes, et d'un modèle ("template") installé avec le programme `sendmail`.

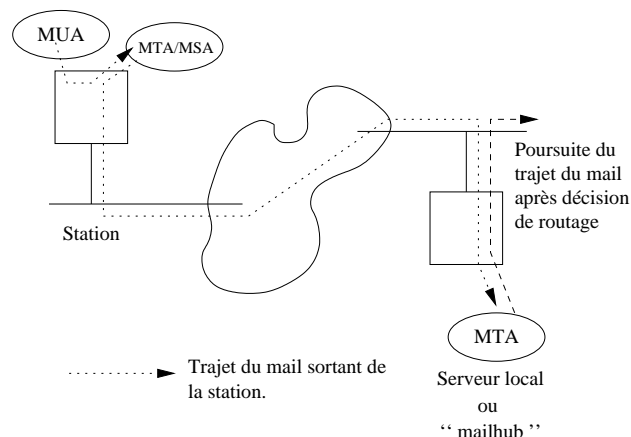
Il faut noter qu'un certain nombre d'administrateurs système, formés à la "vieille école", aiment bien conserver la maîtrise totale de ce qu'ils produisent et préfèrent donc écrire eux-mêmes manuellement leurs règles, ces macros ne leur sont donc pas destinées !

### 5.1 Configuration à l'aide de M4

Le point d'entrée pour utiliser cet outil est une documentation livrée avec la distribution du programme `sendmail` et nommée :

`<préfixe d'installation>/cf/README`

Considérons la situation réseau de la *figure X.07*. Le courrier au départ de la station, soumis par exemple au MSA local, doit être routé systématiquement vers une machine nommée `mailhub.mondomain.fr`, et qui concentre tout le trafic local sortant, d'où d'ailleurs son nom de "mailhub". Celle-ci est censée savoir router le mail qu'elle reçoit, mais ce n'est pas notre préoccupation ici.



<sup>26</sup>Macro processeur bien connu dans le monde Unix

*figure X.07 — Concentration du mail sur un " mailhub "*

Le fichier de configuration pour la station pourrait être :

```

1  divert(0)
2  VERSIONID('mondomain-fla-04-05-2005')dnl
3  OSTYPE(freebsd5)dnl
4  define('confSMTP_LOGIN_MSG','$j --- STATION LAMBDA --- $b')dnl
5  dnl
6  FEATURE(always_add_domain)dnl
7  MASQUERADE_AS(mondomaine.fr)dnl
8  FEATURE(allmasquerade)dnl
9  dnl
10 define('MAIL_HUB','smtp:mailhub.mondomain.fr.')dnl
11 define('SMART_HOST','smtp:mailhub.mondomain.fr.')dnl
12 dnl
13 MAILER(smtp)dnl

```

Et ce fichier (`config.mc`) est utilisé de cette manière :

```
m4 m4/cf.m4 config.mc > sendmail.cf
```

Pour générer en final le fichier attendu, dont le nombre de lignes excèdent 1400! Il faut noter que le MSA se configurent de manière similaire, à partir de son propre fichier de configuration.

**Ligne 1** C'est la définition d'un canal de sortie pour `m4` (cf `man m4`).

**Ligne 2** C'est une étiquette (" tag ") insérée dans le fichier généré et qui servira à l'identifier, par exemple avec la commande `ident` si l'étiquette est un identifiant de `rcs` ou `cvs`.

Remarque : `dnl` est un mot clef de `m4`, au même titre que `divert`, et qui signifie qu'on peut ignorer (**d**iscard) tous les caractères jusqu'au prochain retour à la ligne (**n**l).

**Ligne 3** C'est un identifiant du système d'exploitation pour que `m4` puisse faire les choix adaptés (choix des chemins standards par exemple).

**Ligne 4** Définition de la bannière de HELO du protocole SMTP. `$j` est une variable qui contient le nom canonique (FQDN - Voir page 169)) de la machine hôte. La bannière de HELO smtp de la station ressemblera à ça :

```
220 stationYXZ.mondomain.fr ESMTP --- STATION LAMBDA --- (date du jour)
```

**Ligne 6** Ajout systématique du nom de domaine, même et surtout pour les courriers locaux (donc dans le domaine local par défaut).

**Ligne 7 et 8** Ces deux lignes entraînent la réécriture des adresses From de telle sorte quelles se présentent toujours sous la forme `<untel@mondomain.fr>` au lieu de leur forme par défaut `<untel@station.mondomain.fr>` qui est nuisible au retour du courrier car la machine `station.mondomain.fr` n'est très vraisemblablement pas atteignable directement depuis le réseau global sur son port 25.

**Ligne 10** Tout le mail local doit être envoyé à la machine `mailhub.mondomain.fr`.

**Ligne 11** Tout le mail autre que local doit être envoyé à la machine `mailhub.mondomain.fr`.

**Ligne 13** Normalement inutile dans le cas présent (pas de “ delivery ” sur cette machine).

Pour conclure, `sendmail` comme tous les outils, évolue plusieurs fois par an. Si à chaque version il est nécessaire de reconstruire manuellement son fichier `sendmail.cf` il est probable que votre emploi du temps va être sérieusement amputé d’un temps précieux... Mieux vaut avoir juste à taper “ `make` ” dans le bon répertoire pour reconstruire un fichier de configuration tout beau tout propre !

## 5.2 Configuration manuelle

Cette section est une extraction libre et incomplète du paragraphe 5 du document intitulé “ **Sendmail Installation and Operation Guide** ”, disponible dans toute distribution de la V8. On peut également trouver ce document dans la section “ System Manager’s Manual ” (SMM) des systèmes BSD<sup>27</sup>.

Le fichier de configuration (`sendmail.cf` est organisé comme une série de lignes, le premier caractère de chacune d’elles en précise le type. Une ligne vide ou qui débute par un `#` est ignorée (commentaire), une ligne qui débute par un espace ou une tabulation est la continuation de la précédente.

### 5.2.1 Règles de réécriture

Les règles de réécritures sont repérables comme ces lignes qui démarrent par un **S** (début d’un paquet de règles) ou un par un **R** (simple règle).

Les paquets de règles (organisation d’ensemble *figure X.7* ) ont comme but essentiel d’analyser puis de prendre les bonnes décisions en fonction des adresses trouvées dans l’en-tête. C’est une démarche purement formelle.

Ces règles utilisent une syntaxe dense qui rebute généralement les administrateurs. Il faut imaginer qu’elles sont intégralement analysées chaque fois que le programme `sendmail` est invoqué, c’est à dire en gros pour chaque e-mail entrant ou sortant. Elles doivent donc être faciles à analyser, même par un cpu de modeste performance (ou très chargé, ce qui revient finalement au même).

L’ensemble fonctionne comme un système de production, c’est à dire qui consomme des règles lues séquentiellement et les applique à un jeu de données initiales, ici une ou plusieurs adresses électroniques.

La partie gauche (ou **lhs**<sup>28</sup>) sert de déclencheur (“ pattern matching ”)

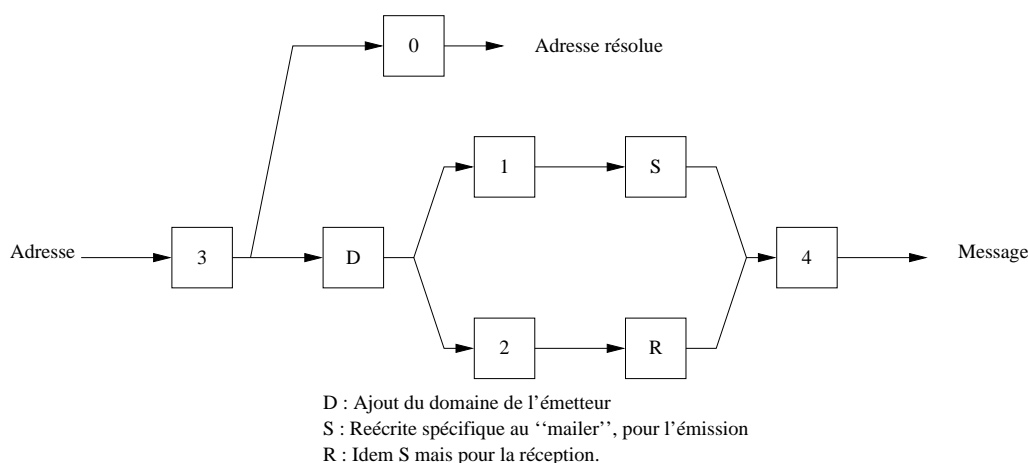
<sup>27</sup>pour FreeBSD, NetBSD ou OpenBSD ça se trouve dans le répertoire `/usr/share/doc/smm/08.sendmailop/`

<sup>28</sup>“ left hand side ”

pour une règle.

La partie droite (ou **rhs**<sup>29</sup>) est déclenchée si le motif de la partie gauche est reconnu dans le flux de données. Le résultat produit, s'il existe, est reinjecté dans le système de production et ce jusqu'à épuisement des possibilités.

La *figure X.7* donne un aperçu du fonctionnement de l'automate, les chiffres (0,1,2,3,4) sont autant de "ruleset", comprendre des paquets de règles qui sont regroupées ensembles parcequ'elles traitent d'un objectif commun.



*figure X.08 — Règles de réécriture*

Les règles sont numérotées, le premier chiffre dit à quel paquet elles appartiennent.

Le paquet de règles 0 sert essentiellement à déterminer le "mailer" c'est à dire le moyen d'envoyer le courrier (SMTP, ESMTP, UUCP...).

Les paquets de règles 1 et 2 sont appliqués respectivement à l'en-tête des messages qui sortent ("send") ou qui entrent ("receive").

Le paquet de règles 3 est appliqué systématiquement à toutes les adresses.

Le paquet de règles 4 est appliqué à toutes les adresses dans le message, typiquement pour passer d'une forme interne à une forme externe.

Le paquet de règles 5, non représenté sur l'automate, sert à traiter les adresses locales après que les aliases aient été appliqués.

Les paquets de règles sont repérés par la lettre **S**, qui en balise le nom, comme dans :

```
#####
### Ruleset 0 -- Parse Address ###
#####
```

Sparse=0

Quant aux règles, elles démarrent toutes avec la lettre **R**, comme dans :

<sup>29</sup> " right hand side "

|      |                       |                         |
|------|-----------------------|-------------------------|
| R\$* | \$: \$>Parse0 \$1     | initial parsing         |
| R<@> | \$#local \$: <@>      | special case error msgs |
| R\$* | \$: \$>ParseLocal \$1 | handle local hacks      |
| R\$* | \$: \$>Parse1 \$1     | final parsing           |

Deux remarques s'imposent :

1. Chaque ligne forme une règle, sur le modèle :

*Rlhs rhs commentaire*

2. Le séparateur de champ entre les trois parties de ces règles est la tabulation (une au minimum)

L'adresse **postmaster@mondomain.fr**, par exemple, quand elle se présente devant le paquet de règles 0 qui démarre ci-dessus, a été mise sous une forme canonique par d'autres règles appliquées préalablement (notamment dans la "ruleset 3") :

postmaster < @ mondomain . fr . >

Le "pattern matching" va tenter de rapprocher cette suite de mots ou tokens de la partie gauche des règles (**lhs**) pour déterminer celles qui peuvent être déclenchées.

Dans la partie gauche **\$\*** s'applique à toute suite de tokens, même vide. Donc la première ligne convient. La deuxième ne le pourrait pas car la chaîne "**postmaster**" précède le caractère "<" et le "@" est suivi de "**mon-domain . fr .**".

La troisième et la quatrième règle sont également déclenchables mais présentement l'ordre d'apparition est également l'ordre de déclenchement, cette possibilité sera donc examinée éventuellement plus tard.

La partie droite de la première règle commence par **\$ : \$>Parse0** ce qui signifie l'appel d'un autre paquet de règles que l'on pourra trouver plus loin dans le fichier **sendmail.cf** :

```
#
# Parse0 -- do initial syntax checking and eliminate local addresses.
#      This should either return with the (possibly modified) input
#      or return with a #error mailer.  It should not return with a
#      #mailer other than the #error mailer.
#
```

S\_Parse0

Le **\$1** signifie que l'on ne transmet que le premier des tokens reconnus dans la partie gauche ("**postmaster**" dans l'exemple)...

### 5.2.2 Exemple de sortie de debug

Il est utile d'avoir ce schéma en tête quand on débogue les règles : avec l'option `-bt` de `sendmail` on peut suivre la progression de la transformation, règle par règle.

```
ADDRESS TEST MODE (ruleset 3 NOT automatically invoked)
Enter <ruleset> <address>
> 3,0 postmaster@mondomain.fr
rewrite: ruleset 3 input: postmaster @ mondomain . fr
rewrite: ruleset 96 input: postmaster < @ mondomain . fr >
rewrite: ruleset 96 returns: postmaster < @ mondomain . fr . >
rewrite: ruleset 3 returns: postmaster < @ mondomain . fr . >
rewrite: ruleset 0 input: postmaster < @ mondomain . fr . >
rewrite: ruleset 199 input: postmaster < @ mondomain . fr . >
rewrite: ruleset 199 returns: postmaster < @ mondomain . fr . >
rewrite: ruleset 98 input: postmaster < @ mondomain . fr . >
rewrite: ruleset 98 returns: postmaster < @ mondomain . fr . >
rewrite: ruleset 198 input: postmaster < @ mondomain . fr . >
rewrite: ruleset 90 input: < mondomain . fr > postmaster < @ mondomain . fr . >
rewrite: ruleset 90 input: mondomain . < fr > postmaster < @ mondomain . fr . >
rewrite: ruleset 90 returns: postmaster < @ mondomain . fr . >
rewrite: ruleset 90 returns: postmaster < @ mondomain . fr . >
rewrite: ruleset 95 input: < mailhub . mondomain . fr > postmaster < @ mondomain . fr . >
rewrite: ruleset 95 returns: $# relay $# mailhub . mondomain . fr $: postmaster < @ mondomain . fr . >
rewrite: ruleset 198 returns: $# relay $# mailhub . mondomain . fr $: postmaster < @ mondomain . fr . >
rewrite: ruleset 0 returns: $# relay $# mailhub . mondomain . fr $: postmaster < @ mondomain . fr . >
>
```

Plus en TP...

## 6 Bibliographie

Pour en savoir davantage, on pourra consulter “ les bons auteurs ” suivants :

- Eric Allman — “ Sendmail Installation and Operation Guide ” — document au format PostScript jointe à toutes les distributions de la **V8.xx**.
- Bryan Costales with Eric Allman & Neil Rickert — “ Sendmail ” — O'Reilly & Associates Inc. — 1994
- “ Installing and Administering ARPA Services ” — Hewlett-Packard — 1991
- Douglas E. Comer — “ Internetworking with TCP/IP – Volume I ” (chapter 19) — Prentice All — 1988
- W. Richard Stevens — “ TCP/IP Illustrated Volume I ” (chapter 28) — Prentice All — 1994

Et pour en savoir encore plus...

**RFC 821** “ Simple Mail Transfer Protocol. ” J. Postel. Aug-01-1982. (Format : TXT=124482 bytes) (Obsoletes RFC0788) (Also STD0010) (Status : STANDARD)

**RFC 822** “ Standard for the format of ARPA Internet text messages. ” D. Crocker. Aug-13-1982. (Format : TXT=109200 bytes) (Obsoletes RFC0733) (Updated by RFC1123, RFC1138, RFC1148, RFC1327) (Also STD0011) (Status : STANDARD)

**RFC 974** “ Mail routing and the domain system. ” C. Partridge. Jan-01-1986. (Format : TXT=18581 bytes) (Status : STANDARD)

**RFC 1123** “ Requirements for Internet hosts - application and support. R.T. ” Braden. Oct-01-1989. (Format : TXT=245503 bytes) (Updates RFC0822) (Updated by RFC2181) (Status : STANDARD)

**RFC 1652** “ SMTP Service Extension for 8bit-MIMEtransport. ” Klensin, N. Freed, M. Rose, E. Stefferud & D. Crocker. July 1994. (Format : TXT=11842 bytes) (Obsoletes RFC1426) (Status : DRAFT STANDARD)

**RFC 1939** “ Post Office Protocol - Version 3. ” J. Myers & M. Rose. May 1996. (Format : TXT=47018 bytes) (Obsoletes RFC1725) (Updated by RFC1957) (Also STD0053) (Status : STANDARD)

**RFC 2060** “ Internet Message Access Protocol - Version 4rev1. ” M. Crispin. December 1996. (Format : TXT=166513 bytes) (Obsoletes RFC1730) (Status : PROPOSED STANDARD)

**RFC 2184** “ MIME Parameter Value and Encoded Word Extensions : Character Sets, Languages, and Continuations. ” N. Freed, K. Moore. August 1997. (Format : TXT=17635 bytes) (Updates RFC2045, RFC2047, RFC2183) (Status : PROPOSED STANDARD)

**RFC 2476** “ Message Submission. ” R. Gellens, J. Klensin. December 1998. (Format : TXT=30050 bytes) (Status : PROPOSED STANDARD)



- RFC 2821** “ Simple Mail Transfer Protocol. ” J. Klensin, Ed. April 2001. (Format : TXT=192504 bytes) (Obsoletes RFC0821, RFC0974, RFC1869) (Status : PROPOSED STANDARD)
- RFC 2822** “ Internet Message Format. ” P. Resnick, Ed. April 2001. (Format : TXT=110695 bytes) (Obsoletes RFC0822) (Status : PROPOSED STANDARD)



# Chapitre XI

## Instrumentalisation de réseaux avec SNMP

### 1 Nécessité d'un outil

La majeure partie des activités informatiques dépendent du bon fonctionnement des réseaux et des services associés. Leurs nombres et leur complexité ne cessent de s'accroître, mais bien souvent le personnel responsable de l'évolution et du bon fonctionnement de l'ensemble ne voit pas ses effectifs humains évoluer dans le même sens, du moins aussi rapidement que le parc de machines à administrer !

Or, le bon fonctionnement d'un grand réseau ne peut dépendre pour seule composante que de l'effort intellectuel d'individus ou de groupe d'individus, fussent-ils compétents et dévoués. Il faut des outils !

#### 1.1 Problématique de l'ISO

L'ISO, s'est penché sur la question et a segmenté le problème de la gestion technique de réseau en cinq points :

**La gestion des pannes** (Fault management) Il s'agit de détecter les pannes, de les localiser et d'y remédier en minimisant l'impact de la perte de fonctionnalité sur le reste du système d'information.

La panne n'est pas une erreur, mais un grand nombre d'erreurs peuvent conduire à déclarer une panne. Par exemple la croissance anormale du nombre de collisions sur un réseau, l'engorgement d'un disque. . .

**La comptabilisation de l'usage des ressources** (Accounting management) Il s'agit d'archiver et de mettre en ordre tous les compteurs générés par les applicatifs et les couches réseaux afin de pouvoir tirer un enseignement de l'usage des ressources.

L'aspect confidentiel de ces données doit être pris en compte.

**La gestion des configurations** (Configuration and name management) Il s'agit de la mise en œuvre et de la configuration de tous les équipements qui inter-agissent sur le réseau.

**L'audit des performances** (Performance management) Il s'agit d'avoir une approche quantitative sur le fonctionnement du réseau afin de pouvoir répondre à des questions aussi basiques que :

1. Quel est le niveau actuel d'utilisation ?
2. Il y a-t-il un (des) trafic(s) excessif(s) ?
3. Le débit nominal est-il réduit à une valeur inacceptable ?
4. Où sont les goulots d'étranglement ?
5. Quelle est l'évolution du temps de réponse ?

**La gestion de la sécurité** (Security management) Il s'agit de maintenir cohérent et effectif l'ensemble des protections sur les autorisations d'accès et données sensibles collectées.

Les logs (**syslog**) sont un point important de la gestion de la sécurité.

En conclusion, les buts d'une gestion technique efficace d'un réseau sont multiples : il s'agit d'offrir aux usagers un service de qualité, de permettre les évolutions du système d'information en y incluant de nouvelles fonctionnalités, d'optimiser l'usage des ressources et de minimiser les coûts d'exploitation ou d'investissement.

## 1.2 Système de gestion de réseau

Un système de gestion de réseau (Network Management System) est une collection d'outils pour la surveillance et le contrôle afin de permettre à un opérateur d'effectuer la plupart des opérations de gestion depuis une interface la plus simple et ergonomique possible !

C'est un ensemble de logiciels (Network Management Entity) associés éventuellement à des matériels spécifiques, qui sont déployés sur tous les composants du système d'information.

Un NMS est donc conçu pour donner une image unifiée du réseau, quelle que soit son étendue et son hétérogénéité. Le logiciel utilisé pour visualiser l'image du réseau est un NMA (Network Management Application).

Un NME :

- Collecte des données sur l'activité réseau
- Conserve ces données dans une base
- Répond aux requêtes du NMA, notamment sur les points suivants :
  1. Transmission des données collectées
  2. Changement d'un paramètre de configuration
  3. Fourniture de statut de composants logiciels ou matériels
  4. Génération de tests
- Envoi des messages d'alerte (**trap**) en cas de détection d'événements exceptionnels.

Au moins un nœud du réseau est désigné comme étant le manager, et supportant le NMA. Cette architecture n'est pas nécessairement centralisée, la supervision du réseau peut s'effectuer par secteurs.

### 1.3 SNMP — Simple Network Management Protocol

SNMP est un terme un peu générique qui désigne à la fois un protocole réseau applicatif bien précis, une collection de spécifications pour le management de réseau et la définition de structures de données ainsi que leurs concepts associés.

SNMP est né en 1988 de la nécessité de disposer d'un outil de supervision du réseau dès lors que celui-ci comporte un grand nombre d'hôtes qui interagissent, stations, serveurs, éléments de routage ou de commutation ou encore boîtes noires. Leur nombre grandissant sur les LANs (des machines en clusters par exemples) implique d'avoir un outil qui permette "d'expliquer" le réseau. Ce besoin est moins évident quand tout va bien, mais il suffit parfois d'un simple petit grain de sable... Dans ces moments là, disposer d'un outil qui délivre une information de synthèse est indispensable !

Les logs, au sens de **syslog** (paragraphe 3.2 page 316) , même concentrés, filtrés, et triés ne délivrent une information parfois trop verbeuse et en tout cas structurée différemment selon les applications ou les noyaux. Les événements réseaux y sont le plus souvent absents, sauf dans le cas très particulier de démons qui surveillent le réseau, **arpwatch**<sup>1</sup> en est un exemple. Le tri par niveau de criticité ne retire rien ou presque au fait que c'est une information brute qu'il faut filtrer pour en extraire l'information pertinente.

L'architecture d'un réseau géré avec SNMP comporte essentiellement deux entités : le **manager** et l'**agent**, ou encore le client et le serveur. Le client (manager) interroge le serveur pour récolter de l'information ou configurer une valeur, le serveur (agent) est capable de prévenir le client en cas d'événements exceptionnels (traps).

Enfin, n'importe quelle machine munie d'une stack IP est susceptible de supporter SNMP, depuis le calepin électronique, en passant par la borne wifi et jusqu'au mainframe.

En quelques mots, SNMP permet :

- De cartographier le réseau
- De fournir un panel très complet de résultats de mesures pour chaque hôte
- De mesurer en temps réel la consommation de ressources d'une application
- De signaler les dysfonctionnements
- De changer certains paramètres réseaux de fonctionnement

Avantages :

- Protocole est simple et facile d'utilisation
- Permet une gestion centralisée d'un parc
- Dispose d'un modèle extensible
- Est indépendant de l'architecture matérielle

---

<sup>1</sup>Site officiel <http://www-nrg.ee.lbl.gov/>

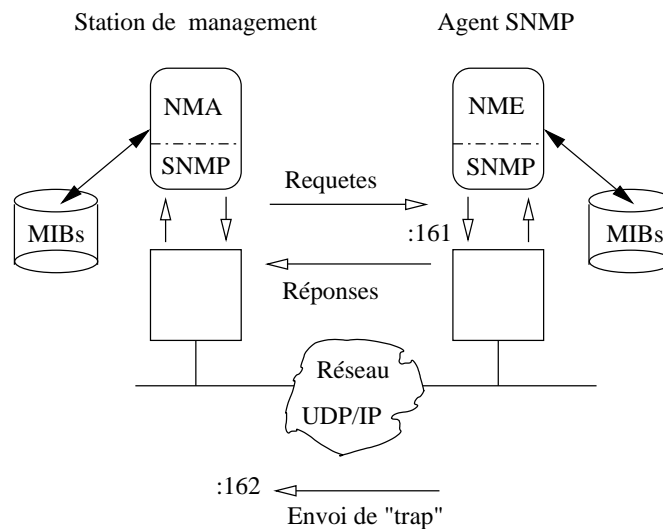


figure XI.01 — Agent et Manager dans une relation de type client-serveur

## 1.4 Historique du protocole SNMP

Avant 1987/1988 il n'y a rien d'autre qu'ICMP pour recevoir de l'info entres routeurs et hôtes. Le couple (`echo request`, `echo reply`) est le plus utilisé pour maintenir un état des machines accessibles.

Le point de départ est SGMP (" Simple Gateway Monitoring Protocol " RFC 1028 de novembre 1987) mais trop orienté sur la gestion des routeurs. Du coté de l'OSI d'autres tentatives avec CMIS et CMIP (" Common Management Information Service & Protocol ")

SNMP est sorti en 1988, comme une version améliorée de SGMP. Les RFCs fondatrices sont les 1155, 1156 et 1157 conservées actuellement au rang d'historiques bien que tous les équipements soient théoriquement encore compatibles SNMPv1 (même s'ils répondent à un niveau de version SNMP plus récent, c'est à dire 2 ou 3).

## 1.5 Vocabulaire et architecture

Un système d'exploitation peut être vu comme une vaste collection de compteurs et d'horloges auxquels SNMP nous permet d'accéder à distance pour les lire et les modifier (certaines, sous réserve d'y avoir accès).

Afin que les agents et les managers soient inter-opérables les variables sont collectionnées selon une représentation arborescente très structurée et standardisée, ce sont les MIBs (" Management Information Base "). On les retrouve partout où SNMP est supporté. Ainsi, une même information se nomme de la même manière quelle que soit l'implémentation de SNMP et indépendamment de sa valeur qui est fonction du contexte. C'est donc très commode pour automatiser les traitements (scripts de collecte et de surveillance...) dans un réseau qui est hétérogène la plupart du temps!

Les feuilles de cet arbre sont les variables et on y accède en connaissant le chemin à priori depuis la racine, un peu à la manière d'un système de fichiers, sauf qu'ici les chemins sont codifiés à l'avance.

Actuellement c'est la MIB-2 qui est la plus répandue (RFC 1213), elle répond parfaitement aux besoins élémentaires. Si un appareil ou un système a des besoins spécifiques il est toujours possible d'ajouter des branches au tronc commun, un embranchement est prévu pour cela, on parle alors d'une extension " vendor ".

Tous les vendeurs d'hôtes réseaux prévoient des MIBs pour leurs équipements (" mibs vendors " donc), dès lors qu'ils sont accessibles via ip (routeurs, commutateurs, ponts, hôtes, imprimantes, boîtes noires diverses) même si celle-ci n'est pas montrée à l'utilisateur final. Telle borne wifi d'un célèbre constructeur informatique " à la pomme ", utilise SNMP pour se configurer mais l'utilisateur ne le voit pas, c'est masqué par une interface utilisateur conviviale.

La *figure XI.01* présente la relation entre les deux entités logicielles qui dans le cas de SNMP se nomment :

**Agent SNMP, ou NME (le serveur)** C'est un logiciel qui s'exécute sur l'appareil que l'on souhaite administrer à distance. Il répond aux requêtes du gestionnaire, et génère des alarmes (traps) si besoin est. La configuration d'un agent est en général assez simple (par rapport à celle d'un logiciel Manager).

**Manager NMA (le client)** C'est le logiciel qui s'exécute sur la station d'administration. Sa configuration est forcément plus délicate que celle de l'agent parcequ'il nécessite une adaptation au réseau local qui est toujours un cas particulier. Il existe de nombreux logiciels HP OpenView, SUN Net Manager, IBM Netview, Spectrum, ISM OpenMaster, SNMPc....

L'Open Source n'est pas en reste et sans être aussi complet, l'outil " tkined " est déjà très satisfaisant pour l'essentiel des besoins (voir la recopie d'écran page 247).

**Sonde RMON (alternative de serveur)** La *figure 01* est en fait incomplète dans le cadre d'une architecture de supervision globale de réseau : si chaque agent sur chaque hôte peut répondre individuellement sur les événements réseau le concernant, il manque un maillon plus global qui fasse la supervision du réseau en lui-même, le véritable " networking management ". Cet élément existe, c'est ce qu'on appelle une sonde RMON, ou encore " Remote Monitoring ".

C'est une entité logicielle, comme un agent SNMP. Elle s'appuie sur une extension de la MIB de base. On la trouve principalement sur les éléments de commutation ou de routage, là où se concentre le trafic réseau, mais on peut la trouver sur un hôte également, par exemple sur un serveur critique.

L'Open Source nous en fournit un très bel exemple avec le logiciel

ntop<sup>2</sup>.

La représentation des données dans les variables n'est pas laissée au hasard des besoins des développeurs mais est structurée selon une spécification appelé SMI " Structure of Management Information ", définie par la RFC 1155, qui dit par exemple qu'un entier positif va de 0 à  $2^{32} - 1$ . Pour être indépendant du formalisme local de la plateforme (problématique de la couche 6 OSI).

## 1.6 Différentes versions

Enfin cet échange de données prend place dans un protocole réseau qui est défini par les RFC 1155 à 1157, nommé " Simple Network Management Protocol " (version 1).

Beaucoup de travail dans les rfc depuis...

Actuellement il y a 3 versions de SNMP, v1, v2c et v3. La v1 est supportée pour des raison " historiques ", la v2 est la plus couramment supportée par les appareillages. Elle pose quelques soucis que tente de régler la v3 (notamment la sécurisation de l'authentification).

La première version souffrait d'un certain nombre de lacunes au niveau du protocole et de la sécurité que la deuxième version (SNMPv2c " Community-based SNMPv2 ") définie par les RFC 1901 à 1908, tente de combler. Rien n'est malheureusement fait coté sécurité dans cette deuxième version mais des améliorations sont apportées aux mibs standards et au protocole.

Plus récemment un nouveau cadre de travail a été développé, qui s'affranchit complètement de la notion de " communauté ", obstacle à l'usage de SNMP en écriture, et qui introduit des améliorations significatives de la sécurité (RFC 3411 à 3418).

L'inertie des habitudes retarde son déploiement généralisé, ainsi que la nécessité de continuer à gérer des appareils ne le supportant pas (encore).

### 1.6.1 Trois composantes pour SNMP

D'après la RFC 1213 (MIB II) le cadre de travail de SNMP repose sur trois composantes :

**SMI** définit les types d'objets utilisés dans les mibs. C'est une sorte de méta modèle de données. Par exemple pour définir une adresse physique (MAC)

```
PhysAddress ::=
    OCTET STRING
-- This data type is used to model media addresses. For many
-- types of media, this will be in a binary representation.
-- For example, an ethernet address would be represented as
-- a string of 6 octets.
```

---

<sup>2</sup><http://www.ntop.org/>



**La MIB** décrit une collection structurée des ressources à gérer. Une ressource à gérer est représentée par un objet.

**Le protocole SNMP** qui régit le contenu des dialogues clients/serveurs c'est à dire l'interrogation des données structurées par la MIB.

### 1.6.2 Conclusion

le protocole SNMP est simple dans sa conception ce qui permet son déploiement sur de très nombreux appareils hétérogènes mis en réseau.

En pratique la situation est moins simple du fait de la coexistence de 3 versions des MIB non toutes supportées par tous les hôtes du réseau.

La configuration de la station d'administration demande du temps, une connaissance très approfondie de la topologie du réseau à administrer, et beaucoup de compétences techniques. C'est un travail à haute valeur ajoutée !

## 2 SMI — Structure of Management Information

La RFC 1155 fondatrice pose le cadre de travail à l'intérieur duquel on peut bâtir les MIBs. En effet, la SMI précise les types de données et les ressources qui peuvent être spécifiées dans une MIB.

Les données ont été prévues simples, le tableau (par exemple pour représenter un ensemble de connexions tcp `tcpConnTable`) et la liste (les éléments d'un quintuplet tcp `tcpConnEntry`) sont les formes les plus complexes prévues.

Ces structures de données sont remplies avec les 5 types suivants :

**networkaddress** Il s'agit d'une zone pouvant contenir une adresse réseau, avec comme format possible **IpAddress** (ipv4 32 bits).

**counter** C'est un compteur qui prend sa valeur maxi à  $2^{32} - 1$  (on reconnaît un entier 32 bits non signé) et qui ne peut pas être décrémenté. Quand il atteint sa valeur maxi il repasse à 0.

**gauge** C'est un compteur, qui a la même valeur maximale que le précédent mais qui au contraire peut être décrémenté. Par contre il ne repasse pas automatiquement à 0 en cas de valeur maximale atteinte.

**timeticks** Le nombre de secondes écoulées depuis **epoch**, c'est à dire le 1er janvier 1970.

**opaque** C'est un flux d'octets banalisés qui permet d'encoder tout ce qui ne relève pas des types précédents, une sorte de fourre-tout en quelque sorte...

Toutes les ressources auxquelles on souhaite accéder décrites dans un document qui est une MIB.

## 3 MIB — Management Information Base

Les MIBs sont des fichiers au format ascii qui décrivent dans le détail chacune des ressources à quantifier. Ces ressources sont des éléments simples (scalaire ou tableaux à deux dimensions). Chaque unité de description se nomme un " objet " (sans aucun rapport avec la programmation du même nom). Une MIB est une collection structurée de tous ces objets. Un même objet est accessible de la même manière partout sur le réseau.

Le propos d'une MIB peut être celui de respecter un standard ouvert, décrit alors par une RFC et distribué librement, ou d'être spécifique pour un type particulier d'appareil et de constructeur (" mibs vendor "). Sa diffusion est alors à l'initiative de son auteur.

Le contenu d'une MIB est toujours décrit à l'aide d'un langage formel nommé ASN.<sup>13</sup> utilisé généralement pour définir des structures de données

---

<sup>13</sup>développé et standardisé par le CCITT (X.208) et l'ISO (ISO 8824)

applicatives complexes (couche 6 du modèle de l'OSI) et qui est indépendant de tout langage de programmation.

Un extrait de la MIB II (RFC 1213) concernant le début de la description de l'objet `tcpConnTable`, à savoir la table des connexions `tcp`, celle là même que l'on peut observer avec la commande `netstat -p tcp`.

```

1  -- the TCP Connection table
2
3  -- The TCP connection table contains information about this
4  -- entity's existing TCP connections.
5
6  tcpConnTable OBJECT-TYPE
7      SYNTAX  SEQUENCE OF TcpConnEntry
8      ACCESS  not-accessible
9      STATUS  mandatory
10     DESCRIPTION
11         "A table containing TCP connection-specific
12         information."
13     ::= { tcp 13 }
14
15  tcpConnEntry OBJECT-TYPE
16      SYNTAX  TcpConnEntry
17      ACCESS  not-accessible
18      STATUS  mandatory
19      DESCRIPTION
20         "Information about a particular current TCP
21         connection.  An object of this type is transient,
22         in that it ceases to exist when (or soon after)
23         the connection makes the transition to the CLOSED
24         state."
25      INDEX   { tcpConnLocalAddress,
26               tcpConnLocalPort,
27               tcpConnRemAddress,
28               tcpConnRemPort }
29     ::= { tcpConnTable 1 }

```

Ce petit exemple montre que l'identification d'un objet repose sur cinq champs :

1. Le nom de l'objet, `tcpConnTable` qui balise le début de la définition
2. (SYNTAX) La syntaxe d'usage. Ici une liste d'objets `tcpConnEntry`. On y reconnaîtra sans peine les éléments du quintuplet vus en cours TCP.
3. (ACCESS) L'accès (lecture, écriture, lecture-écriture, pas accessible). Ici On ne peut ni lire ni écrire dans cet objet.
4. (STATUS) L'état de l'objet, valeur à prendre dans obligatoire (MANDATORY), obsolète ou optionnel.
5. (DESCRIPTION) Un texte qui décrit ce que représente l'objet.

Les lignes qui débutent par un " – " sont des commentaires.

Enfin on peut remarquer que ce bloc de texte se termine par `::= { tcp 13 }` qui signifie que cet objet est le treizième fils de l'objet père `tcp`.

Tous les objets de toutes les MIBs (propriétaires ou non) sont organisés dans un seul arbre, donc avec une seule racine commune. Pour identifier un objet dans cet ensemble, on parle de son OID.

### 3.1 OID — Objet Identifier

Le nommage des objets utilise une représentation arborescente dont la racine est figée mais qui est extensible à volonté. Le nommage d'un objet passe par la définition (en ASN.1) d'un "Objet Identifier" ou OID, qui peut s'apparenter au "path" d'un fichier.

Ce chemin peut s'exprimer de manière symbolique, par exemple `.iso.org.dod.internet.mgmt.mib-2.system.sysDescr.0` ou encore dans une représentation numérique absolument équivalente `.1.3.6.1.2.1.1.1`. En pratique on pourra le plus fréquemment faire référence seulement à `sysDescr.0`<sup>4</sup>. La racine de l'arbre est à gauche, contrairement, par exemple, au système de nommage du DNS qui place la racine à droite.

En effet un OID est une séquence d'entiers, parcourus dans l'arbre de la racine jusqu'à la feuille terminale. Chaque noeud traversé est étiqueté par un nombre et un bref texte descriptif. Bien entendu l'unicité de l'étiquetage à un niveau donné de l'arbre est primordial pour son bon fonctionnement.

La racine n'est pas nommée, d'où le point (.) à gauche des deux écritures qui précèdent.

À ce jour deux entités se partagent les trois noeuds du premier niveau de l'arbre : l'ISO<sup>5</sup> et le CCITT<sup>6</sup>, le troisième noeud s'explique par une entité mixte des deux et nommée "joint-iso-ccitt".

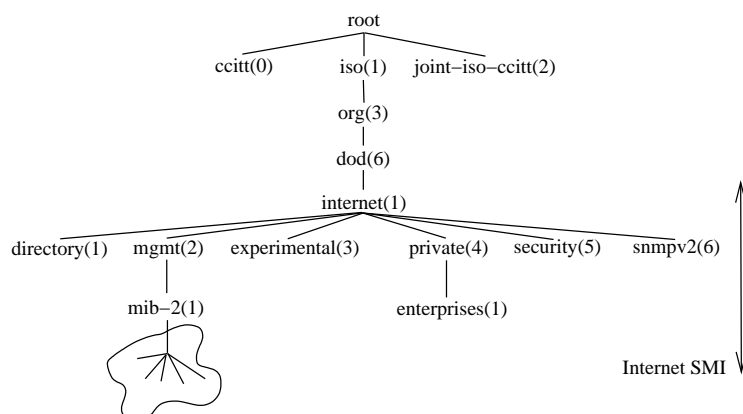


figure XI.02 — La racine de l'arbre des OIDs

Sous le noeud de l'iso un sous arbre est prévu pour d'autres organisations, l'une d'elle est le département de la défense US (dod). La RFC 1155 pose le fait qu'un sous arbre du dod est alloué à l'IAB (Internet Activity Board), sans doute une trace des origines militaires de la pile Arpa.

Et voila pourquoi les OIDs standards sont placés sous le noeud nommé

<sup>4</sup>Notez la présence du " 0 " en fin de chaîne qui ne fait pas partie du nommage mais est un artifice à l'interrogation pour indiquer qu'il s'agit d'une feuille de l'arbre et non par exemple la base d'un tableau

<sup>5</sup>International Organization for Standardisation

<sup>6</sup>International Telegraph and Telephone Consultative Committee

**mib-2(1)** et que le préfixe le plus commun est `.1.3.6.1.2.1`, indices des noeuds pères traversés à partir de la racine !

**Directory(1)** Réservé pour l'OSI

**mgmt(2)** L'administration de ce sous arbre est délégué à l'IANA<sup>7</sup> et est donc régit par des RFCs !

**Experimental(3)** Utilisé pour identifier des objets utilisés pour des déploiements expérimentaux sur l'Internet. Délégué à l'IANA.

**Private(4)** Comme son nom l'indique ce sous arbre est celui des délégations privées. Le sous arbre **enterprise(1)** permet aux entreprises d'y placer leurs MIBs, après s'être enregistrées auprès de l'IANA.

...

## 3.2 Types de données élémentaires

Le type des objets utilisés dans les MIBs est limité à un sous ensemble des types disponibles dans ASN.1, mais suffisant pour exprimer les compteurs, les tables et les identificateurs que l'on trouve dans la mémoire d'un système d'exploitation.

**INTEGER** De nombreux compteurs du système d'exploitation utilisent un tel type, comme par exemple ceux des statistiques extraites du noyau par la commande `netstat -s -p ip`.

**OCTETS STRINGS** Pour définir une chaîne de caractères comme une suite de 0 ou plus octets de 8 bits.

**NULL** Pour dire qu'il n'y a pas de valeur.

**OBJECT IDENTIFIER** Pour définir les objets (OID).

**SEQUENCE** Se rapproche de la notion de structure du langage C, autrement dit pour grouper plusieurs types dans un seul.

**SEQUENCE-OF** Introduit la notion de vecteur.

Bien que ces types de données puissent ressembler à ceux de tel ou tel langage de programmation, leur représentation interne diffère très certainement puisqu'elle respecte les " Basic Encoding Rules<sup>8</sup> ", ou BER, afin d'être absolument portables sur tout type de plateforme. BER est une méthode d'encodage des valeurs pour tous les types définis par ASN.1, sous forme d'une chaîne d'octets et basée sur l'usage d'un triplet de valeurs (type, longueur, valeur) ou TLV.

Ainsi par exemple les chaînes de caractères ne sont pas terminées par un caractère null (Ascii 0) comme dans le langage C mais sont encodées directement avec leur longueur.

---

<sup>7</sup>Internet Assigned Numbers Authority

<sup>8</sup>développé et standardisé par le CCITT (X.209) et l'ISO (ISO 8825)

## 4 La MIB-2

la mib standard la plus courante est la MIB-2 définie dans la RFC 1213, c'est un sur-ensemble de la mib d'origine (MIB-I) définie dans la RFC 1156. Cette mib regroupe les compteurs les plus courants associés à une pile Arpa et d'autres comme ceux associés à la technologie Token-Ring, FDDI, Microsoft Lan Manager, DECnet, pour information.

La racine du sous-arbre concernée est clairement `mgmt`, c'est à dire `.1.3.6.1.2` et le noeud concerné est `mib-2(1)` qui est l'OID défini ligne 15.

Puis viennent les 10 sous arbres décrits plus avant dans cette mib :

**system(1)** Le groupe **system** fournit des informations d'ordre général sur le system lui-même, comme l'e-mail d'un contact, la valeur de l' "uptime" ou encore la location physique de l'appareil.

**interfaces(2)** Le groupe **interfaces** regroupe toutes les informations sur les interfaces physiques ou virtuels présents, leur type, le fabricant, leur caractéristiques et enfin les statistiques d'usage.

**at(3)** Ce groupe est une seule table de correspondances entre les adresses physiques et logiques. Pour une pile Arpa il s'agit de la table des adresses physique(MAC), telle qu'elle peut être extraite par la commande `arp -an`.

**ip(4)** Le groupe **ip** contient toutes informations relatives à ce protocole (adresse, netmask), notamment la table de routage et tous les compteurs auxquels on peut accéder à l'aide de `netstat -s -p ip`.

**icmp(5)** Le groupe **icmp** contient toutes les informations relatives à ce protocole. Le compteur du nombre d' "echo request" est par exemple accessible, tout comme il peut l'être avec un `netstat -s -p icmp`. Tous les messages sont associés à deux compteurs.

**tcp(6)** Le groupe **tcp** contient toutes les informations relatives à ce protocole, par exemple celles que l'on peut obtenir à l'aide d'un `netstat -s -p tcp` plus d'autres comme la liste des connexions en cours avec leur état.

**udp(7)** Le groupe **udp** regroupe toutes les informations relative à ce protocole (`netstat -s p -udp`). Gère également la liste des applications utilisant ce protocole.

**egp(8)** Le groupe **egp** regroupe les informations relatives au protocole de routage " Exterior Gateway Protocol ".

**transmission(10)** Le groupe **transmission** regroupe des interfaces déjà définies dans le **Interfaces(2)**. mais selon d'autres critères, comme par exemple les protocoles supportés.

**snmp(11)** Ce groupe donne des informations sur l'implémentation et l'exécution de SNMP lui-même, c'est à dire le nombre de message entrants, sortants, la répartition du type de requêtes reçues, émises.

Voici un extrait du début de cette mib :

```
1  RFC1213-MIB DEFINITIONS ::= BEGIN
2
3  IMPORTS
4      mgmt, NetworkAddress, IpAddress, Counter, Gauge,
5      TimeTicks
6      FROM RFC1155-SMI
7      OBJECT-TYPE
8      FROM RFC-1212;
9
10 -- This MIB module uses the extended OBJECT-TYPE macro as
11 -- defined in [14];
12
13 -- MIB-II (same prefix as MIB-I)
14
15 mib-2      OBJECT IDENTIFIER ::= { mgmt 1 }
16
17 -- textual conventions
18
19 DisplayString ::=
20     OCTET STRING
21 -- This data type is used to model textual information taken
22 -- from the NVT ASCII character set.  By convention, objects
23 -- with this syntax are declared as having
24
25 --
26 --     SIZE (0..255)
27
28 PhysAddress ::=
29     OCTET STRING
30 -- This data type is used to model media addresses.  For many
31 -- types of media, this will be in a binary representation.
32 -- For example, an ethernet address would be represented as
33 -- a string of 6 octets.
34
35 -- groups in MIB-II
36
37 system      OBJECT IDENTIFIER ::= { mib-2 1 }
38
39 interfaces  OBJECT IDENTIFIER ::= { mib-2 2 }
40
41 at          OBJECT IDENTIFIER ::= { mib-2 3 }
42
43 ip          OBJECT IDENTIFIER ::= { mib-2 4 }
44
45 icmp        OBJECT IDENTIFIER ::= { mib-2 5 }
46
47 tcp         OBJECT IDENTIFIER ::= { mib-2 6 }
48
49 udp         OBJECT IDENTIFIER ::= { mib-2 7 }
50
51 egp         OBJECT IDENTIFIER ::= { mib-2 8 }
52
53 -- historical (some say hysterical)
54 -- cmot      OBJECT IDENTIFIER ::= { mib-2 9 }
55
56 transmission OBJECT IDENTIFIER ::= { mib-2 10 }
57
58 snmp        OBJECT IDENTIFIER ::= { mib-2 11 }
59
```

## 5 Protocole SNMP

le protocole SNMP est du type client/serveur classique. Un vocabulaire spécifique : le serveur est nommé “ Agent SNMP ” alors que le client est un “ Manager ” ou encore “ Network Management Software ” (NMS).

Le serveur (agent) écoute les requêtes du client (manager) sur le port 161 (UDP) et peut lui envoyer des messages d'exception (trap) sur son port 162.

Le choix du transport UDP est justifié par le trafic de petits datagrammes (la RFC 1157 “ An implementation of this protocol need not accept messages whose length exceeds 484 octets ”. Les besoins ont évolué mais ce choix initial perdure. Aux parties applicatives la tâche de faire le travail d'une couche de transport si besoin est (gestion d'un “ time-out ” et de la réémission des datagrammes manquants).

Le client (manager), interroge à son rythme les agents sur leur port 161 et écoute sur le port 162 (UDP) les éventuels messages d'exceptions envoyés par ces mêmes agents. Il faut noter que le protocole permet non seulement la lecture de variables mais aussi leur modification, ce qui pose des problèmes d'authentification et de confidentialité, non résolus avec SNMPv1 et SNMPv2. En effet, ce qui fait office de mécanisme d'authentification est une chaîne de caractères qui circule “ en clair ” sur le réseau, c'est la fameuse “ communauté ” dont la valeur par défaut sur les équipement est traditionnellement “ public ”.

La plupart du temps on se borne donc à l'aspect “ read only ” du protocole et seulement pour des échanges sur des réseaux qui devraient être protégés, par exemple cantonnés sur un vlan d'administration sur lequel ne circule aucun trafic applicatif inutile et surtout auquel aucun utilisateur standard n'accède.

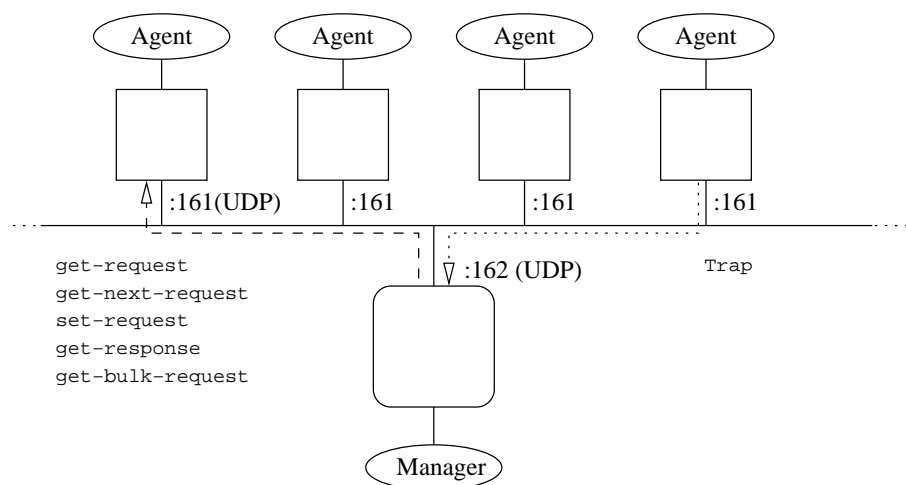


figure XI.03 — Des agents et un Manager



## 5.1 Communauté

La communauté SNMP est une relation entre un agent et les stations d'administration qui l'interrogent. Cette unique chaîne de caractères définit à la fois l'authentification et le contrôle d'accès.

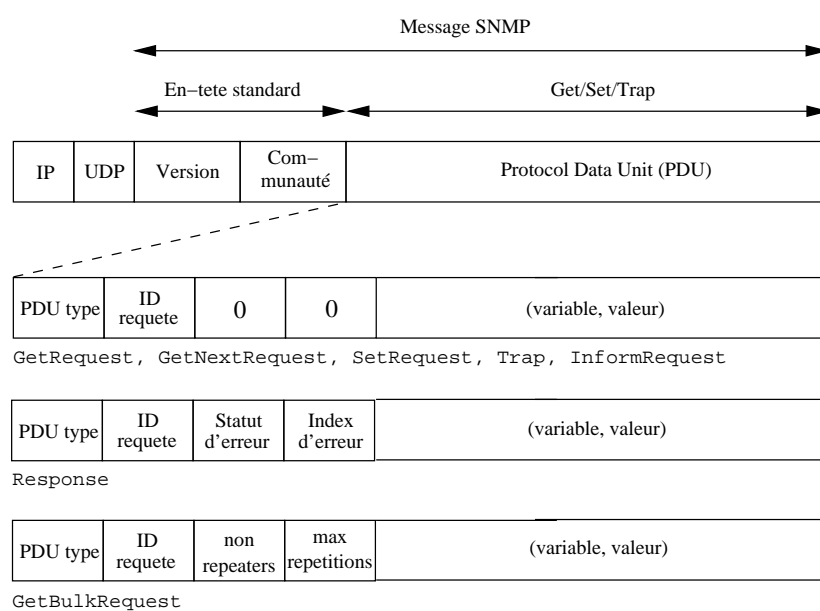
Il peut y avoir autant de communautés que d'agents, c'est au manager d'en conserver la liste.

Chaque message d'une station d'administration vers un agent comporte le nom de la communauté et donc permet à l'agent d'authentifier la source de la requête. Ce mode d'authentification n'est bien sûr plus adapté aux contraintes de sécurité qu'impose l'exploitation moderne des réseaux.

Le minimum pour exploiter malgré tout SNMPv2 est d'avoir au moins trois communautés différentes : une pour la lecture (GET), une pour l'écriture (SET) et une troisième pour les traps.

## 5.2 PDUs

Que ce soit pour des requêtes, des réponses aux requêtes, ou l'envoi d'un trap, SNMPv2 s'appuie sur un message dont le format est décrit succinctement dans la *figure 04* :



*figure XI.04 — Format des messages SNMP*

La partie standard de l'en-tête, comporte deux champs :

**Version** Il s'agit de la version du protocole, 1, 2 ou 3.

**Communauté** Une chaîne d'octets qui identifie la communauté, "public" par défaut...

Ensuite le message est composé d'une partie dont la longueur et le contenu sont assez variables, selon les opérations. C'est ce qu'on appelle le PDU (" Protocol Data Unit "). Il y en a sept possibles. En effet, le protocole de base (SNMPv1) prévoit cinq types de requêtes :

**GetRequest** C'est une question du manager à l'agent. Une liste de couples (variable,valeur) est fournie. Les valeurs sont positionnées à `unSpecified`.

**GetNextRequest** Cette requête est assez voisine de la précédente à ceci près que l'OID exact de la variable est déterminé en prenant le plus proche dans l'ordre lexicographique (d'où le sens de " next ").

**SetRequest** C'est une demande du manager à l'agent pour positionner une certaine valeur à chacune des variables listées.

**GetResponse** C'est la réponse à toutes les requêtes Get/Set qui précèdent.

**Trap** Envoyé depuis l'agent vers le manager, associé à une liste de couples (variable,valeur). Il n'y a pas de réponse à un trap.

Auquels SNMPv2 en ajoute deux autres :

**GetBulkRequest** Pour récupérer des données de grande taille, c'est à dire des morceaux complets de l'arbre. Les deux champs `non repeaters` et `max repetitions` servent alors à paramétrer les limites de ce transfert, dans la limite de la taille d'un message.

**InformRequest** Sert à la communication entre managers. Une station d'administration envoie des données vers une autre station qui centralise les informations contenues dans la MIB " manager to manager ". Le message a le même format qu'un Get. Ce type de message est une sorte de mécanisme de traps entre managers (configuration d'alarmes, ensemble d'événements choisis).

Ainsi le champ PDU type peut-il prendre l'une de ces sept valeurs et conduire à autant de PDUs différents, en taille et en signification.

Chaque champ de l'en-tête SNMP à une taille variable, selon l'implémentation des OIDs de la MIB.

**PDU type** Valeur à prendre dans la liste `get-request`, `get-next-request`, `get-bulk-request`, `response`, `set-request`, `inform-request`, `snmpv2-trap`.

**RequestID** C'est un numéro de requête, la réponse doit porter le même numéro que celui de la requête.

**Error-status** Une valeur non nulle indique une erreur pendant le traitement de la requête.

**Error-index** Quand `error-status` n'est pas nul, ce champ identifie le numéro d'ordre du couple (variable,valeur) qui pose problème. Le premier a 1 comme index.

(variable,valeur) Il s'agit de couple, la variable est un OID et la valeur est celle associée à l'OID.

## 5.3 SNMPv3

## 6 L'outil NET-SNMP

D'abord nommé UCD-SNMP au début des années 1990 à l'université de Carnegie-Mellon, le projet s'est transformé en NET-SNMP au début des années 2000 et est maintenant la base de nombreux outils open-source ou non. Sa fiabilité est telle qu'un OS industriel tel que Solaris 10<sup>9</sup> n'hésite pas à le placer dans son " core OS " :

```
#solaris10$ /usr/sfw/sbin/snmpd -version
```

```
NET-SNMP version: 5.0.9
```

```
Web: http://www.net-snmp.org/
```

```
Email: net-snmp-coders@lists.sourceforge.net
```

L'outil `net-snmp` est d'abord un outil d'administrateur système donc est essentiellement composé de commandes à taper dans un shell. Il existe d'autres approches plus graphiques, nous donnerons quelques pistes dans le paragraphe suivant.

Commandes pour interroger un agent :

`snmpget`, `snmpgetnext`, `snmpwalk` `snmptable`, `snmpdelta`

Commande pour positionner une valeur :

`snmpset`

Un daemon pour recevoir les notifications :

`snmptrapd`

Un agent :

`snmpd`

Nous allons mettre en œuvre quelques exemples d'usage de ces outils avec certains des OIDs de la MIB-2. On suppose que la machine `localhost` est accessible avec `public` comme valeur pour la communauté d'accès en mode lecture seule.

### 6.1 `snmptranslate`

Dans sa forme la plus simple cette commande prend un OID et affiche la valeur textuelle correspondante. Prenons le cas de l'uptime c'est à dire au sens de SNMP le nombre de centièmes de seconde écoulées depuis que la partie réseau a été initialisée. Son nom textuel est `SNMPv2-MIB::sysUpTime`.

```
$ snmptranslate -On SNMPv2-MIB::sysUpTime
.1.3.6.1.2.1.1.3
```

```
$ snmptranslate -Of SNMPv2-MIB::sysUpTime.0
.iso.org.dod.internet.mgmt.mib-2.system.sysUpTime.sysUpTimeInstance
```

On peut également utiliser une expression régulière :

---

<sup>9</sup><http://www.sun.com>

```
$ snmptranslate -TB sys.*Time
SNMPv2-MIB::sysORUpTime
SNMPv2-MIB::sysUpTime
DISMAN-EVENT-MIB::sysUpTimeInstance
HOST-RESOURCES-MIB::hrSystemUptime
```

Mais aussi l'utiliser pour obtenir plus d'information sur l'OID :

```
$ snmptranslate -On -Td SNMPv2-MIB::sysUpTime
.1.3.6.1.2.1.1.3
sysUpTime OBJECT-TYPE
    -- FROM SNMPv2-MIB, RFC1213-MIB
    SYNTAX TimeTicks
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION "The time (in hundredths of a second) since the
                  network management portion of the system was last
                  re-initialized."
 ::= { iso(1) org(3) dod(6) internet(1) mgmt(2) mib-2(1) system(1) 3 }
```

```
$ snmptranslate -IR -Tp SNMPv2-MIB::system
+--system(1)
|
+-- -R-- String      sysDescr(1)
|      Textual Convention: DisplayString
|      Size: 0..255
+-- -R-- ObjID       sysObjectID(2)
+-- -R-- TimeTicks   sysUpTime(3)
|  |
|  +--sysUpTimeInstance(0)
|
+-- -RW- String      sysContact(4)
|      Textual Convention: DisplayString
|      Size: 0..255
+-- -RW- String      sysName(5)
|      Textual Convention: DisplayString
|      Size: 0..255
+-- -RW- String      sysLocation(6)
|      Textual Convention: DisplayString
|      Size: 0..255
+-- -R-- INTEGER     sysServices(7)
|      Range: 0..127
+-- -R-- TimeTicks   sysORLastChange(8)
|      Textual Convention: TimeStamp
|
+--sysORTable(9)
|
+--sysOREntry(1)
|  Index: sysORIndex
|
+-- ---- INTEGER     sysORIndex(1)
|      Range: 1..2147483647
+-- -R-- ObjID       sysORID(2)
+-- -R-- String      sysORDescr(3)
|      Textual Convention: DisplayString
```

```
|          Size: 0..255
+-- -R-- TimeTicks sysORUpTime(4)
          Textual Convention: TimeStamp
```

Le lecteur curieux d'en savoir plus pourra essayer la commande `snmptranslate -IR -Tp` ! L'extrait suivant de la MIB montre le début de la description du groupe `system`.

```
1  -- the System group
2
3  -- Implementation of the System group is mandatory for all
4  -- systems.  If an agent is not configured to have a value
5  -- for any of these variables, a string of length 0 is
6  -- returned.
7
8  sysDescr OBJECT-TYPE
9      SYNTAX  DisplayString (SIZE (0..255))
10     ACCESS  read-only
11     STATUS  mandatory
12     DESCRIPTION
13         "A textual description of the entity.  This value
14         should include the full name and version
15         identification of the system's hardware type,
16         software operating-system, and networking
17         software.  It is mandatory that this only contain
18         printable ASCII characters."
19     ::= { system 1 }
20
21  sysObjectID OBJECT-TYPE
22      SYNTAX  OBJECT IDENTIFIER
23      ACCESS  read-only
24      STATUS  mandatory
25      DESCRIPTION
26          "The vendor's authoritative identification of the
27          network management subsystem contained in the
28          entity.  This value is allocated within the SMI
29          enterprises subtree (1.3.6.1.4.1) and provides an
30          easy and unambiguous means for determining 'what
31          kind of box' is being managed.  For example, if
32          vendor 'Flintstones, Inc.' was assigned the
33          subtree 1.3.6.1.4.1.4242, it could assign the
34          identifier 1.3.6.1.4.1.4242.1.1 to its 'Fred
35          Router'."
36      ::= { system 2 }
37
38  sysUpTime OBJECT-TYPE
39      SYNTAX  TimeTicks
40      ACCESS  read-only
41      STATUS  mandatory
42      DESCRIPTION
43          "The time (in hundredths of a second) since the
44          network management portion of the system was last
45          re-initialized."
46      ::= { system 3 }
47
48  sysContact OBJECT-TYPE
49      SYNTAX  DisplayString (SIZE (0..255))
50      ACCESS  read-write
51      STATUS  mandatory
52      DESCRIPTION
53          "The textual identification of the contact person
54          for this managed node, together with information
55          on how to contact this person."
56      ::= { system 4 }
57
58
```

## 6.2 snmpget

Cette commande correspond à l'opération la plus élémentaire du protocole SNMP, aller chercher l'information relative à un OID précis sur un agent.

```
$ snmpget -v 2c -c public localhost SysUpTimeInstance
DISMAN-EVENT-MIB::sysUpTimeInstance = Timeticks: (33310293) 3 days, 20:31:42.93
```

```
$ snmpget -v 2c -c public localhost SNMPv2-MIB::sysLocation
SNMPv2-MIB::sysLocation = No Such Instance currently exists at this OID
```

Une erreur courante avec cette commande est d'oublier l'index (" instance subidentifier ") de la donnée demandée. Le cas le plus courant est pour les données de type scalaire, il n'y a qu'une seule valeur alors il ne semble pas nécessaire de préciser un index. Cet index est toujours un simple zéro (0) comme l'exemple ci-dessous le montre.

```
$ snmpget -v 2c -c public localhost SNMPv2-MIB::sysLocation.0
SNMPv2-MIB::sysLocation.0 = STRING: S110
```

Ce point de détail technique s'avère vite lassant, c'est pourquoi on utilise plus fréquemment la commande suivante...

## 6.3 snmpgetnext

```
$ snmpgetnext -v 2c -c public localhost SNMPv2-MIB::sysUpTime.0
SNMPv2-MIB::sysContact.0 = STRING: Francois Laissus
```

La commande renvoie la valeur associée à l'OID (ou aux OIDs) suivant, ainsi on a l'impression que l'exemple ci-dessus est difficilement utilisable en pratique! En fait il n'en est rien et l'exemple suivant devrait dissiper cette fausse impression :

```
$ snmpgetnext -v 2c -c public localhost SNMPv2-MIB::sysUpTime
DISMAN-EVENT-MIB::sysUpTimeInstance = Timeticks: (34853433) 4 days, 0:48:54.33
```

Un usage très employé de cette commande est de l'utiliser avec une OID incomplète, par exemple sans l'index (" instance subidentifier ") et l'agent en détermine la prochaine instance complète associée à sa valeur. C'est une sorte de mécanisme rudimentaire de complétion.

## 6.4 snmpwalk

Cette commande effectue des requête de type get-next pour explorer toute l'arborescence des sous-arbres liés à un OID. Par exemple :

```
$ snmpwalk -v 2c -c public localhost 1.3.6.1.2.1.5
IP-MIB::icmpInMsgs.0 = Counter32: 205413
IP-MIB::icmpInErrors.0 = Counter32: 0
IP-MIB::icmpInDestUnreachs.0 = Counter32: 205039
```



```

IP-MIB::icmpInTimeExcds.0 = Counter32: 43
IP-MIB::icmpInParmProbs.0 = Counter32: 0
IP-MIB::icmpInSrcQuenchs.0 = Counter32: 0
IP-MIB::icmpInRedirects.0 = Counter32: 0
IP-MIB::icmpOutDestUnreachs.0 = Counter32: 203947
...

```

Permet d'accéder d'un seul coup à toutes les compteurs relatifs à SNMP (la sortie a été tronquée). Le lecteur pourra essayer l'OID 1.3.6 en argument...

## 6.5 snmptable

Comme son nom le suggère, cette commande est plutôt utilisée pour manipuler des tables. Ici il s'agit de la liste des interfaces et de leurs compteurs associés.

```

$ snmptable -v 2c -c public -Os -Cw 80 localhost ifTable
SNMP table: ifTable

```

| ifIndex | ifDescr | ifType           | ifMtu | ifSpeed    | ifPhysAddress  | ifAdminStatus |
|---------|---------|------------------|-------|------------|----------------|---------------|
| 1       | em0     | ethernetCsmacd   | 1500  | 1000000000 | 0:6:5b:f:5a:31 | up            |
| 2       | em1     | ethernetCsmacd   | 1500  | 1000000000 | 0:6:5b:f:5a:32 | down          |
| 3       | lo0     | softwareLoopback | 16384 | 0          |                | up            |

SNMP table ifTable, part 2

| ifOperStatus | ifLastChange    | ifInOctets | ifInUcastPkts | ifInNUcastPkts | ifInDiscards |
|--------------|-----------------|------------|---------------|----------------|--------------|
| up           | 0:0:00:00:00.00 | 2318958978 | 2767895021    | 2              | 0            |
| down         | 0:0:00:00:00.00 | 0          | 0             | 0              | 0            |
| up           | 0:0:00:00:00.00 | 90441064   | 717640        | 0              | 0            |

SNMP table ifTable, part 3

| ifInErrors | ifInUnknownProtos | ifOutOctets | ifOutUcastPkts | ifOutNUcastPkts |
|------------|-------------------|-------------|----------------|-----------------|
| 0          | 0                 | 2137908126  | 2767886776     | 0               |
| 0          | 0                 | 0           | 0              | 0               |
| 0          | 0                 | 90441687    | 717644         | 0               |

SNMP table ifTable, part 4

| ifOutDiscards | ifOutErrors | ifOutQLen | ifSpecific |
|---------------|-------------|-----------|------------|
| 0             | 0           | ?         | ?          |
| 0             | 0           | ?         | ?          |
| 0             | 0           | ?         | ?          |

## 6.6 snmpset

Pour changer la valeur d'une donnée, donc déconseillé en SNMPv2.

## 6.7 Approche graphique

Ce premier exemple graphique montre un outil open-source nommé **mbrowse**<sup>10</sup> simple et bien commode pour interroger n'importe quel agent à partir d'un interface graphique assez intuitif.

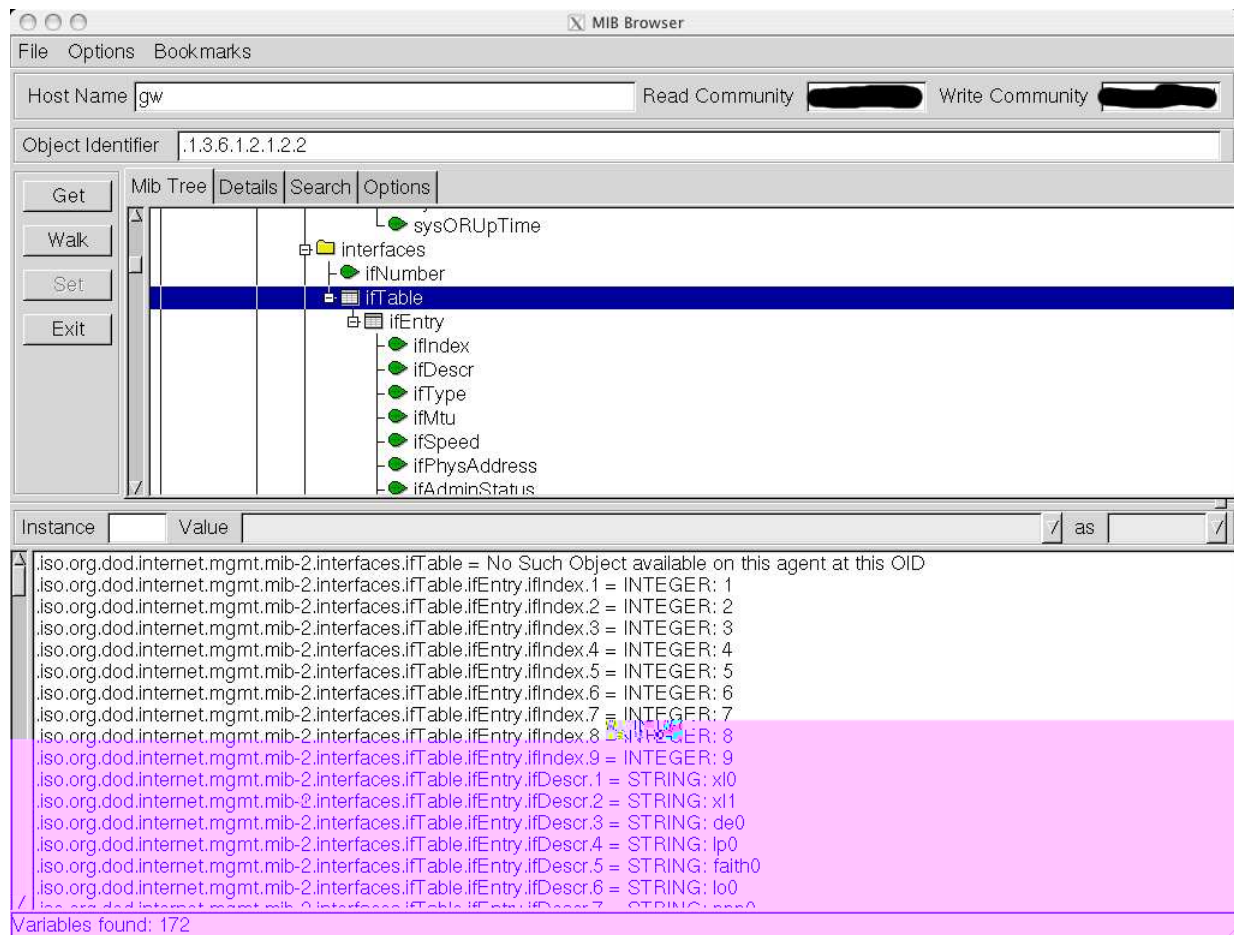


figure XI.05 — Exemple d'interrogation d'un agent avec l'outil **mbrowse**

Dans un deuxième exemple, il s'agit d'extraire le contenu de la table **ifTable**, comme nous avons pu le faire précédemment avec **snmptable**, ce coup-ci avec la commande **snmpwalk**, puis de traiter graphiquement le résultat.

```
$ snmpwalk -c public -v2c gw ifTable
IF-MIB::ifIndex.1 = INTEGER: 1
IF-MIB::ifIndex.2 = INTEGER: 2
IF-MIB::ifIndex.3 = INTEGER: 3
IF-MIB::ifDescr.1 = STRING: fxp0
IF-MIB::ifDescr.2 = STRING: plip0
IF-MIB::ifDescr.3 = STRING: lo0
```

<sup>10</sup><http://www.kill-9.org/mbrowse/>

```

IF-MIB::ifType.1 = INTEGER: ethernetCsmacd(6)
IF-MIB::ifType.2 = INTEGER: para(34)
IF-MIB::ifType.3 = INTEGER: softwareLoopback(24)
IF-MIB::ifMtu.1 = INTEGER: 1500
IF-MIB::ifMtu.2 = INTEGER: 1500
IF-MIB::ifMtu.3 = INTEGER: 16384
IF-MIB::ifSpeed.1 = Gauge32: 100000000
IF-MIB::ifSpeed.2 = Gauge32: 0
IF-MIB::ifSpeed.3 = Gauge32: 0
IF-MIB::ifPhysAddress.1 = STRING: 0:2:b3:3d:22:5
IF-MIB::ifPhysAddress.2 = STRING:
IF-MIB::ifPhysAddress.3 = STRING:
...
IF-MIB::ifInOctets.1 = Counter32: 29017357
...
IF-MIB::ifOutOctets.1 = Counter32: 3117625
...

```

Où l'on s'aperçoit que cette machine a trois interfaces, nommées respectivement `fxp0`, `plip0` et `lo0`. On voit également que le mtu de l'interface de loopback (`lo0`) est de 16384 octets et que l'interface `fxp0` est le seul qui travaille réellement au vu des compteurs qui lui sont associés, 29017357 octets en entrée et 3117625 en sortie, depuis que la machine est en route (voir `sysUpTime`).

Bien entendu cette approche manuelle est trop lourde pour être utilisée telle que pour surveiller un réseau ! Il est indispensable d'utiliser des outils capables de faire la synthèse de tous ces compteurs, par exemple pour les présenter sous forme graphique. L'outil `mrtg`<sup>11</sup> est capable de produire très simplement un tel graphique avec jusqu'à une année d'historique pour n'importe quel interface :

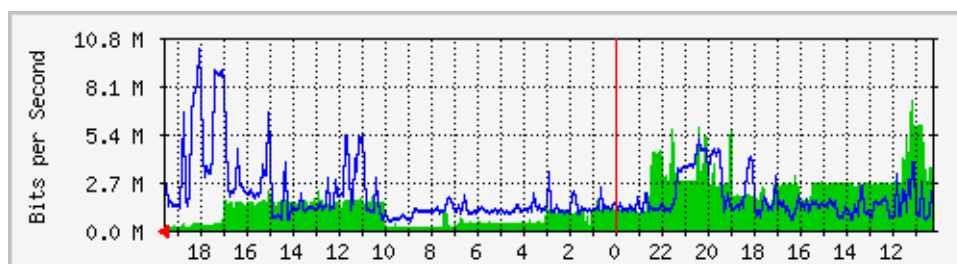
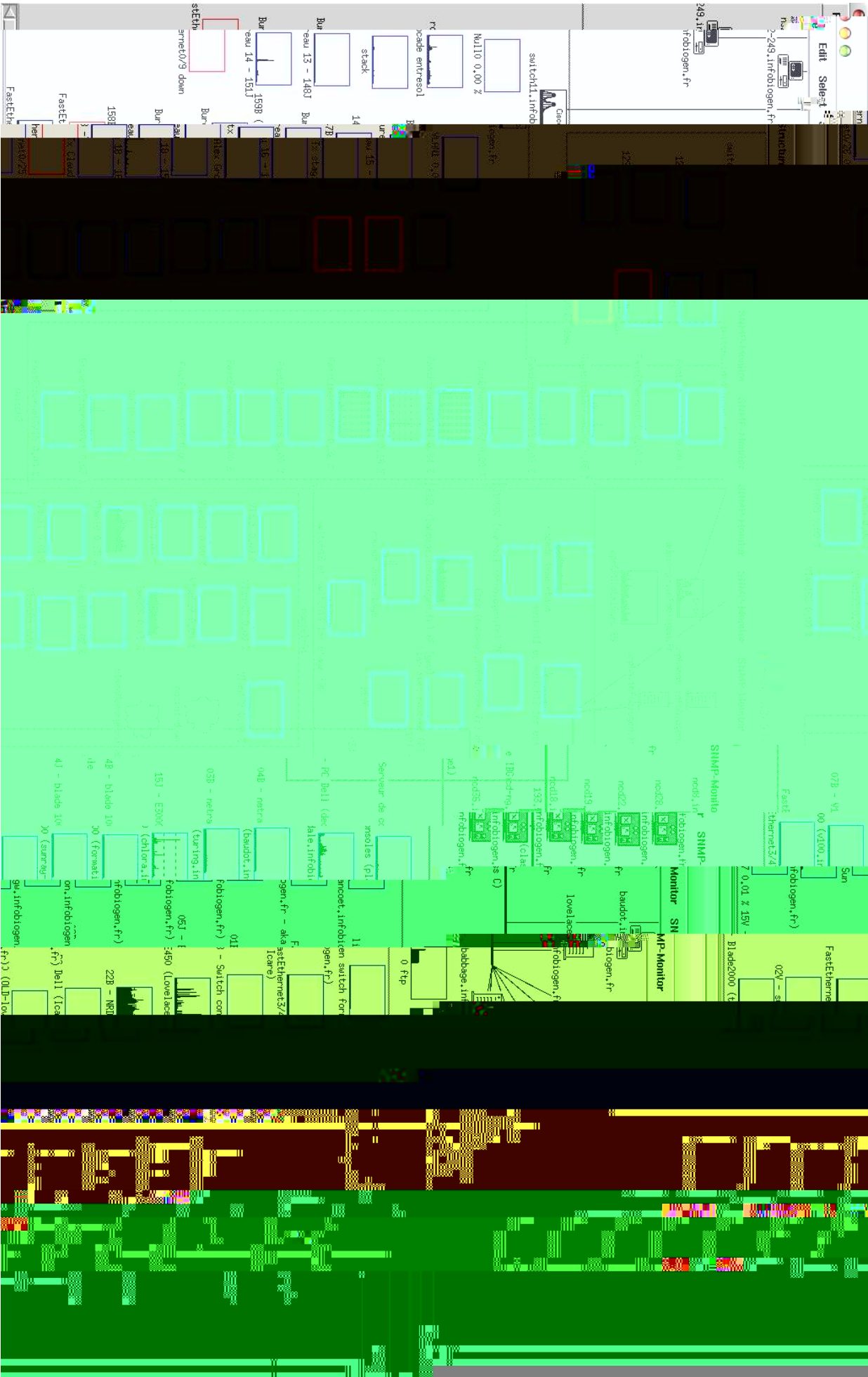


figure XI.06 — Synthèse graphique des compteurs `ifInOctets` et `ifOutOctets` sur 24h

Il existe d'autres MIBs, notamment applicatives comme celle définie par la RFC 1611 qui concerne le serveur de noms et celle définie par la RFC 2790 qui concerne les ressources de l'hôte lui-même (espace disque, charge...) et que l'on peut surveiller également avec l'outil `mrtg`. Il est ainsi aisé de faire des graphiques d'usage de la mémoire, des disques, de la charge de la machine, etc...

<sup>11</sup><http://oss.oetiker.ch/mrtg/>



La page précédente est un exemple d'écran de surveillance d'un réseau aujourd'hui complètement démantelé, et obtenu à l'aide de l'outil open-source tkined!

## 7 Glossaire des acronymes SNMP

**Agent** C'est le logiciel embarqué dans l'hôte réseau, quel qu'il soit. Il fonctionne en mode serveur et est à l'écoute en UDP sur le port 161 (**snmp**). Il est également susceptible d'envoyer des "traps" vers le port 162 du ou des manager(s).

**ASN1** "Abstract Syntax Notation One" est une norme internationale<sup>12</sup> dont la vocation première est la spécification de données utilisées dans les protocoles de communication.

**BER** "Basic Encoding Rules" Méthode d'encodage des valeurs pour tous les types définis dans ASN.1.

**Manager** Le logiciel de supervision. Il interroge les agents dans une relation de type client – serveur dont il assume la partie cliente. Le manager est destinataire des "traps", qu'il réceptionne en UDP sur le port 162 (snmptrap).

**MIB** "Management Information Base". C'est la description de tous les composants logiciels ou matériels surveillés par l'agent. Chaque composant est désigné par son OID. La MIB est écrite à l'aide du langage ASN.1 et selon une SMI bien précise. C'est un arbre, dont les nœuds et les feuilles sont repérés de manière unique par un chiffre.

**NMA** "Network Management Application". Est une autre manière, non spécifique à SNMP, de désigner le manager.

**NME** "Network Management Entity". Est une autre manière, non spécifique à SNMP, de désigner l'agent.

**NMS** "Network Management Software". Synonyme de NMA.

**OID** "Object Identifier". C'est la désignation unique d'un objet dans la structure en arbre de la MIB.

**PDU** "Protocol Data Unit". Il s'agit des paquets réseau, structurés selon le détail du protocole applicatif SNMP.

**RMON** "Remote Network Monitoring". Il s'agit d'un agent particulier dont l'objet est la surveillance du réseau lui-même et non un hôte en particulier. On désigne souvent ces agents sous la terminologie de sonde RMON.

**SMI** "Structure of Management Information". C'est la description du contenu et du formatage des données d'une MIB.

---

<sup>12</sup><http://asn1.elibel.tm.fr/fr/>

**SNMP** “ Simple Network Management Protocol ”. C’est le nom du protocole réseau qui sert à interroger les agents/sondes RMON.

**Trap** C’est un message d’exception envoyé depuis l’agent vers le port 162 du (des) manager(s).

## 8 Liens & Bibliographie

Quelques RFCs minimales et en nombre non exhaustif!

**RFC 1115, 1156, 1157, 1213, 1901 à 1908, 3411 à 3418**

Des urls :

- Pour ASN.1

<http://www.asn1.org/>  
<http://asn1.elibel.tm.fr/>  
<http://www.itu.int/ITU-T/asn1/>

- Pour SNMP lui-même :

<http://www.snmplink.org/>  
<http://www.faqs.org/faqs/snmp-faq/part1/index.html>  
<http://www.faqs.org/faqs/snmp-faq/part2/index.html>  
<http://www.net-snmp.org/wiki/index.php/Tutorials>  
<http://www.cisco.com/en/US/docs/internetworking/technology/handbook/SNM>

- Quelques outils en vrac :

Net-Snmp <http://net-snmp.sourceforge.net/>  
Mrtg <http://oss.oetiker.ch/mrtg/>  
Scotty <http://wwwhome.cs.utwente.nl/~schoenw/scotty/>  
Mbrowse <http://www.kill-9.org/mbrowse/>

Quelques ouvrages de référence :

- William Stalling — “ SNMPv1, SNMPv2, SNMPv3 and RMON 1 and 2 ” (third Edition) — Addison-Wesley 1999.
- Douglas R. Mauro and Kevin J. Schmidt — “ Essential SNMP ” — O’Reilly 2001.
- W. Richard Stevens — “ TCP/IP Illustrated, Volume 1 - The protocols ” — Addison-Wesley

## Quatrième partie

### Sockets BSD et architecture de serveurs





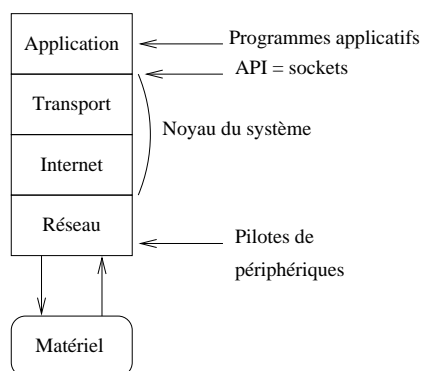
# Chapitre XII

## Généralités sur les sockets de Berkeley

### 1 Généralités

La version BSD 4.1c d'Unix pour VAX, en 1982, a été la première à inclure TCP/IP dans le noyau du système d'exploitation et à proposer une interface de programmation de ces protocoles : les sockets<sup>1</sup>.

Les sockets sont ce que l'on appelle une API ( " Application Program Interface " ) c'est à dire une interface entre les programmes d'applications et la couche transport, par exemple TCP ou UDP. Néanmoins les sockets ne sont pas liées à TCP/IP et peuvent utiliser d'autres protocoles comme AppleTalk, Xérox XNS, etc. . .



*figure XII.01 — Les sockets, une famille de primitives*

Les deux principales API pour Unix sont les sockets Berkeley et les TLI System V. Ces deux interfaces ont été développées en C.

Les fonctionnalités des sockets que nous allons décrire, sont celles apparues à partir de la version 4.3 de BSD Unix, en 1986. Il faut noter que les

---

<sup>1</sup>Pour un historique très intéressant de cette période on pourra consulter <http://www.oreillynet.com/pub/a/network/2000/03/17/bsd.html>

constructeurs de stations de travail comme HP, SUN<sup>2</sup>, IBM, SGI, ont adopté ces sockets, ainsi sont-elles devenues un standard de fait et une justification de leur étude.

Pour conforter ce point de vue il n'est pas sans intérêt d'ajouter que toutes les applications majeures (`named`, `dhcpcd`, `sendmail`, `ftpd`, `apache`,...) "Open Sources" de l'Internet, utilisent cette API.

Enfin, et avant d'entrer dans le vif du sujet, le schéma ci-dessous rappelle les relations entre pile *ARPA*, N° de port et processus.

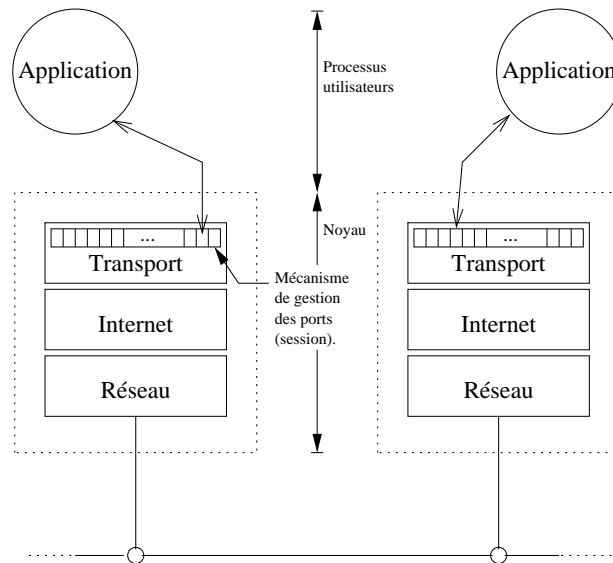


figure XII.02 — Relation pile IP, numéro de port et process ID

## 2 Présentation des sockets

Les créateurs des sockets ont essayé au maximum de conserver la sémantique des primitives systèmes d'entrées/sorties sur fichiers comme `open`, `read`, `write`, et `close`. Cependant, les mécanismes propres aux opérations sur les réseaux les ont conduits à développer des primitives complémentaires (par exemple les notions de connexion et d'adresse IP n'existent pas lorsque l'on a besoin d'ouvrir un fichier!).

Quand un processus ouvre un fichier (`open`), le système place un pointeur sur les structures internes de données correspondantes dans la table des descripteurs ouverts de ce processus et renvoie l'indice utilisé dans cette table. Par la suite, l'utilisateur manipule ce fichier uniquement par l'intermédiaire de l'indice, aussi nommé descripteur de fichier.

Comme pour un fichier, chaque socket active est identifiée par un petit entier appelé descripteur de socket. Unix place ce descripteur dans la même

<sup>2</sup>Les applications natives de ce constructeur utilisent les TLI, par contre il est possible d'utiliser les sockets dans toutes les applications que l'on recompile soi-même, elles sont présentes dans des bibliothèques précisées lors de la compilation des sources

table que les descripteurs de fichiers, ainsi une application ne peut-elle pas avoir un descripteur de fichier et un descripteur de socket de même valeur.

Pour créer une socket, une application utilisera la primitive `socket` et non `open`, pour les raisons que nous allons examiner. En effet, il serait très agréable si cette interface avec le réseau conservait la sémantique des primitives de gestion du système de fichiers sous Unix, malheureusement les entrées/sorties sur réseau mettent en jeu plus de mécanismes que les entrées/sorties sur un système de fichiers, ce n'est donc pas possible.

Il faut considérer les points suivants :

1. Dans une relation du type client-serveur les relations ne sont pas symétriques. Démarrer une telle relation suppose que le programme sait quel rôle il doit jouer.
2. Une connexion réseau peut être du type connectée ou non. Dans le premier cas, une fois la connexion établie le processus origine discute uniquement avec le processus destinataire. Dans le cas d'un mode non connecté, un même processus peut envoyer plusieurs data-grammes à plusieurs autres processus sur des machines différentes.
3. Une connexion est définie par un quintuplet (cf cours TCP page 89) qui est beaucoup plus compliqué qu'un simple nom de fichier.
4. L'interface réseau supporte de multiples protocoles comme XNS, IPX, APPLETalk<sup>3</sup>, la liste n'est pas exhaustive. Un sous système de gestion de fichiers sous Unix ne supporte qu'un seul format.

En conclusion de ce paragraphe on peut dire que le terme *socket* désigne, d'une part un ensemble de primitives, on parle des *sockets de Berkeley*, et d'autre part l'extrémité d'un canal de communication (point de communication) par lequel un processus peut émettre ou recevoir des données. Ce point de communication est représenté par une variable entière, similaire à un descripteur de fichier.

## 3 Étude des primitives

Ce paragraphe est consacré à une présentation des primitives essentielles pour programmer des applications en réseaux. Pour être bien complet il est fortement souhaitable de consulter les pages de manuels associées aux primitives et la documentation citée en fin de chapitre page 273.

### 3.1 Création d'une socket

La création d'une socket se fait par l'appel système `socket`.

---

<sup>3</sup>L'inspection du fichier `/usr/include/sys/socket.h` sous FreeBSD 6.x en explicite une petite quarantaine

```
#include <sys/types.h>    /* Pour toutes les primitives */
#include <sys/socket.h>    /* de ce chapitre il faut */
#include <netinet/in.h>    /* inclure ces fichiers. */
```

```
int socket(int PF, int TYPE, int PROTOCOL);
```

**PF** Spécifie la famille de protocole (“ **P**rotocol **F**amily ”) à utiliser avec la socket. On trouve (extrait) par exemple sur FreeBSD <sup>4</sup> 7.0 :

```
PF_INET      : Pour les sockets IPv4
PF_INET6     : Pour les sockets IPv6
PF_LOCAL     : Pour rester en mode local (pipe)...
PF_UNIX      : Idem AF_LOCAL
PF_ROUTE     : Accès à la table de routage
PF_KEY       : Accès à une table de clefs (IPsec)
PF_LINK      : Accès à la couche “ Link ”
```

Mais il existe d’autres implémentations notamment avec les protocoles<sup>5</sup> :

```
PF_APPLETALK : Pour les réseaux Apple
PF_NS        : Pour le protocole Xerox NS
PF_ISO       : Pour le protocole de l’OSI
PF_SNA       : Pour le protocole SNA d’IBM
PF_IPX       : Protocole Internet de Novell
PF_ATM       : “ Asynchronous Transfert Mode ”
...          : ...
```

Le préfixe PF est la contraction de “ **P**rotocol **F**amily ” On peut également utiliser le préfixe **AF**, pour “ **A**ddress **F**amily ”. Les deux nommages sont possibles ; l’équivalence est définie dans le fichier d’en-tête `socket.h`.

**TYPE** Cet argument spécifie le type de communication désiré. En fait avec la famille PF\_INET, le type permet de faire le choix entre un mode connecté, un mode non connecté ou une intervention directe dans la couche IP :

```
SOCK_STREAM  : Mode connecté           Couche transport
SOCK_DGRAM   : Mode non connecté       Idem
SOCK_RAW     : Dialogue direct avec la couche IP
```

Faut-il préciser que seules les sockets en mode connecté permettent les liaisons “ full-duplex ” ?

**PROTOCOL** Ce troisième argument permet de spécifier le protocole à utiliser. Il est du type UDP ou TCP le plus couramment<sup>6</sup>.

<sup>4</sup>[www.freebsd.org](http://www.freebsd.org)

<sup>5</sup>On en compte 38 sur une machine FreeBSD 7.0 (22/10/2008)

<sup>6</sup>il en existe au moins un autre pour les sockets de type PF\_INET et PF\_INET6, nommé SCTP et qui n’est pas (encore) traité dans ce cours

```

IPPROTO_TCP           : TCP
IPPROTO_SCTP          : SCTP
IPPROTO_UDP           : UDP
IPPROTO_RAW, IPPROTO_ICMP : uniquement avec SOCK_RAW

```

PROTOCOL est typiquement mis à zéro car l'association de la famille de protocole et du type de communication définit explicitement le protocole de transport :

```

PF_INET + SOCK_STREAM  ⇒ TCP = IPPROTO_TCP
PF_INET + SOCK_DGRAM   ⇒ UDP = IPPROTO_UDP

```

C'est une constante définie dans le fichier d'en-têtes `/usr/include/netinet/in.h` et qui reflète le contenu du fichier système `/etc/protocols`.

### 3.1.1 Valeur retournée par socket

La primitive `socket` retourne un entier qui est le descripteur de la socket nouvellement créée par cet appel.

Par rapport à la connexion future cette primitive ne fait que donner le premier élément du quintuplet :

{**protocole**, port local, adresse locale, port éloigné, adresse éloignée}

Si la primitive renvoie -1, la variable globale `errno` donne l'indice du message d'erreur idoine dans la table `sys_errlist`, que la bibliothèque standard sait parfaitement exploiter <sup>7</sup>.

#### Remarque importante :

Comme pour les entrées/sorties sur fichiers, un appel système `fork` duplique la table des descripteurs de fichiers ouverts du processus père dans le processus fils. Ainsi les descripteurs de sockets sont également transmis.

Le bon usage du descripteur de socket partagé entre les deux processus incombe donc à la responsabilité du programmeur.

Enfin, quand un processus a fini d'utiliser une socket il appelle la primitive `close` avec en argument le descripteur de la socket :

```
close(descripteur de socket);
```

Si un processus ayant ouvert des sockets vient à s'interrompre pour une raison quelconque, en interne la socket est fermée et si plus aucun processus n'a de descripteur ouvert sur elle, le noyau la supprime.

<sup>7</sup>cf `man errno` ou la page de manuel de `perror(3)`

## 3.2 Spécification d'une adresse

Il faut remarquer qu'une `socket` est créée sans l'adresse de l'émetteur - comprendre le couple (numéro de port, adresse IP) - ni celle du destinataire. Il y a deux couples à préciser, celui coté client et l'autre coté serveur. La primitive `bind` effectue cette opération pour la socket de l'hôte local.

### 3.2.1 Spécification d'un numéro de port

L'usage d'un numéro de port est obligatoire. Par contre le choix de sa valeur est largement conditionné par le rôle que remplit la socket : client versus serveur.

S'il s'agit d'un serveur, l'usage d'une valeur de port " bien connue " est essentiel pour être accessible systématiquement par les clients (par exemple le port 25 pour un serveur SMTP ou 80 pour un serveur HTTP).

À l'inverse, le codage de la partie cliente d'une application réseau ne nécessite pas une telle précaution (sauf contrainte particulière due au protocole de l'application elle-même) parce que le numéro de port associé à la socket cliente est communiqué au serveur via l'en-tête de la couche de transport choisie, dès la prise de contact par le réseau.

Le serveur utilise alors la valeur lue dans l'en-tête pour répondre à la requête du client, quel que soit le choix de sa valeur initiale. L'établissement de cette valeur par le client peut donc être le résultat d'un automate, éventuellement débrayable.

### 3.2.2 Spécification d'une adresse IP

Pour des raisons évidentes de communication, il est nécessaire de préciser l'adresse IP du serveur avec lequel on souhaite établir un trafic réseau.

Par contre, concernant le choix sa propre adresse IP, c'est à dire celle qui va servir d'adresse pour le retour des datagrammes, un comportement par défaut peut être choisi lors de la construction de la socket, qui consiste à laisser au noyau du système le soin d'en choisir la valeur la plus appropriée.

Pour une machine unix standard mise en réseau, c'est le cas par exemple d'une station de travail, celle-ci possède au moins deux adresses IP : une sur le réseau local et une autre sur l'interface de loopback (cf page 75). La socket est alors associée aux deux adresses IP, voire plus si la machine est du type " multi-homed " (page 44).

On peut également choisir pour sa socket un comportement plus sélectif, consistant à n'écouter que sur une seule des adresses IP de la station.

### 3.2.3 La primitive `bind`

La primitive `bind` effectue ce travail d'associer une socket à un couple (adresse IP, numéro de port) associés dans une structure de type `sockaddr_in`, pour IPv4. Mais la primitive `bind` est généraliste, ce qui

explique que son prototype fasse état d'une structure générique nommée `sockaddr`, plutôt qu'à une structure dédiée d'un protocole particulier (IPv4 ici).

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

`socket` : Usage du descripteur renvoyé par `socket`.  
`myaddr` : La structure qui spécifie l'adresse locale que l'on veut associer à la `socket` préalablement ouverte.  
`addrlen` : Taille en octets de la structure qui contient l'adresse.

`sockaddr` est constituée (dans sa forme POSIX) de deux octets qui rappellent la famille de protocole, suivis de 14 octets qui définissent l'adresse en elle-même.

### 3.2.4 Les structures d'adresses

Avec la présence de plus en plus effective d'IPv6, les implémentations les plus récentes tiennent compte des recommandations de la RFC 3493<sup>8</sup>, ajoutent un champ `sa_len` d'une longueur de 8 bits et font passer de 16 à 8 bits la taille du champ `sa_family` pour ne pas augmenter la taille globale de la structure.

```
struct sockaddr {
    uint8_t    sa_len ;           /* La structure */
    sa_family_t sa_family ;       /* generique */
    char       sa_data[14] ;
};
```

`sa_len` indique taille de la structure en octets, il est présent au même emplacement dans toutes les variantes de cette structure et contient 16 (octets) pour une structure de type `sockaddr_in`, ou 28 octets pour une structure de type `sockaddr_in6` (IPv6).

Pour la famille `PF_INET` (IPv4) cette structure se nomme `sockaddr_in`, et est définie de la manière suivante :

```
struct in_addr {
    unsigned long    s_addr ;      /* 32 bits Internet */
};

struct sockaddr_in {
    uint8_t    sin_len ;          /* Taille de la structure == 16 octets */
    sa_family_t sin_family ;      /* PF_INET (IPv4) */
    in_port_t  sin_port ;        /* Numero de port sur 16 bits / NBO */
    struct in_addr sin_addr ;     /* Adresse IP sur 32 bits / NBO */
    char       sin_zero[8] ;     /* Inutilises */
};
```

---

<sup>8</sup>Basic Socket Interface Extensions for IPv6

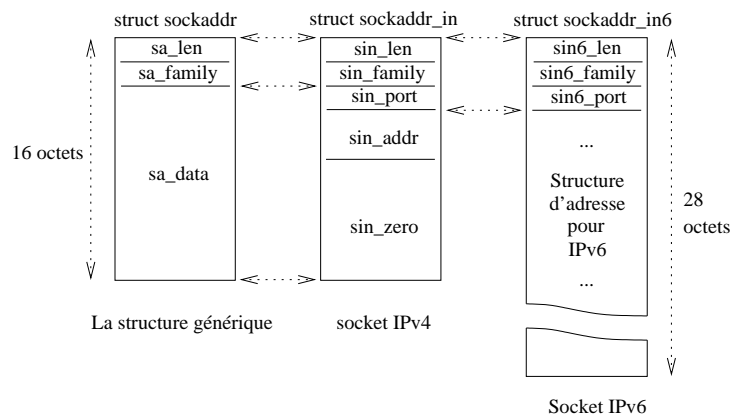


figure XII.03 — Structures d'adresses

La primitive `bind` ne permet pas toutes les associations de numéros de port, par exemple si un numéro de port est déjà utilisé par un autre processus, ou encore si l'adresse internet est invalide.

Trois utilisations typiques de la primitive :

1. En règle général les serveurs fonctionnent avec des numéros de port bien connus (cf `/etc/services`). Dans ce cas `bind` indique au système “ c’est mon adresse, tout message reçu à cette adresse doit m’être renvoyé ”. En mode connecté ou non, les serveurs ont besoin de préciser cette information avant de pouvoir accepter les requêtes des clients.
2. Un client peut préciser sa propre adresse, en mode connecté ou non.
3. Un client en mode non connecté a besoin que le système lui assigne une adresse particulière, ce qui autorise l’usage des primitives `read` et `write` traditionnellement dédiées au mode connecté.

### 3.2.5 Valeur retournée par `bind`

`Bind` retourne 0 si tout va bien, -1 si une erreur est intervenue. Dans ce cas la variable globale `errno` est positionnée à la bonne valeur.

Cet appel système complète l’adresse locale et le numéro de port du quintuplet qui qualifie une connexion. Avec `bind+socket` on a la moitié d’une connexion, à savoir un protocole, un numéro de port et une adresse IP :

{protocole, port local, adresse locale, port éloigné, adresse éloignée}



### 3.3 Connexion à une adresse distante

Prendre l'initiative de l'établissement d'une connexion est typique de la démarche d'un client réseau..

La primitive `connect` permet d'établir la connexion avec une socket distante, supposée à l'écoute sur un port connu à l'avance de la partie cliente. Son usage principal est d'utiliser une socket en mode " connecté ". L'usage d'une socket en mode datagramme est possible mais a un autre sens (voir plus loin) et est moins utilisé.

La primitive `connect` a le prototype suivant :

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

|                 |   |
|-----------------|---|
| <b>sockfd</b>   | : Le descripteur de socket renvoyé par la primitive <code>socket</code> .                       |
| <b>servaddr</b> | : La structure qui définit l'adresse du destinataire, du même type que pour <code>bind</code> . |
| <b>addrlen</b>  | : La longueur de l'adresse, en octets.  |

#### 3.3.1 Mode connecté

Pour les protocoles orientés connexion, cet appel système rend la main au code utilisateur une fois établi le circuit virtuel entre les deux piles TCP/IP. Durant cette phase, des paquets sont échangés comme nous avons pu déjà l'examiner page 89 dans le cas de TCP.

Tant que cette connexion n'est pas complètement établie au niveau de la couche de transport, la primitive `connect` reste en mode noyau, et est donc bloquante vis à vis du code de l'application.

Dans le cas général, les clients n'ont pas besoin de faire appel à `bind` avant d'invoquer `connect`, la définition de la socket locale est complétée automatiquement : le port est attribué automatiquement selon une démarche décrite page 276, et l'adresse IP est l'une de celles de l'interface qu'emprunte le datagramme pour son routage initial<sup>9</sup>.

#### 3.3.2 Mode datagramme

Dans le cas d'un client en mode datagramme, un appel à `connect` n'est pas faux mais il ne sert à rien au niveau protocolaire, il redonne aussitôt la main au code utilisateur. Le seul intérêt que l'on peut y trouver est que l'adresse du destinataire est alors fixée et que l'on peut alors utiliser les primitives `read`, `write`, `recv` et `send`, traditionnellement réservées au mode connecté.

---

<sup>9</sup>Est-il nécessaire de rappeler qu'un interface peut comporter plusieurs adresses IP et qu'il peut y avoir plusieurs interfaces réseau sur un même hôte... ?

### 3.3.3 Valeur retournée par connect :

En cas d'erreur elle renvoie la valeur -1 et positionne la variable globale `errno` à la valeur idoine, par exemple à `ETIMEDOUT`, s'il n'y a pas eu de réponse à l'émission des paquets de synchronisation (cf page 94). Bien d'autres erreurs liées à des problèmes du réseau sont à consulter dans la section **ERRORS** de la page de manuel. Un code 0 indique que la connexion est établie sans problème particulier.

Tous les éléments du quintuplet sont en place :

{protocole, port local, adresse locale, port éloigné, adresse éloignée}

## 3.4 Envoyer des données

Une fois qu'un programme d'application a créé une socket, il peut l'utiliser pour transmettre des données. Il y a cinq primitives possibles pour ce faire :

`send`, `write`, `writenv`, `sendto`, `sendmsg`

### 3.4.1 Envoi en mode connecté

`Send`, `write` et `writenv` fonctionnent uniquement en mode connecté, parce-qu'elles n'offrent pas la possibilité de préciser l'adresse du destinataire.

Les différences entre ces trois primitives sont mineures.

```
ssize_t write(int descripteur, const void *buffer, size_t longueur);
```

Quand on utilise `write`, le `descripteur` désigne l'entier renvoyé par la primitive `socket`. Le `buffer` contient les octets à transmettre, et `longueur` leur cardinal.

Tous les octets ne sont pas forcément transmis d'un seul coup, et ce n'est pas une condition d'erreur. En conséquence il est absolument nécessaire de tenir compte de la valeur de retour de cette primitive, négative ou non.

La primitive `writenv` est sensiblement la même que `write` simplement elle permet d'envoyer un tableau de structures du type `iovec` plutôt qu'un simple buffer, l'argument `vectorlen` spécifie le nombre d'entrées dans `iovector` :

```
ssize_t writenv(int descriptor, const struct iovec *iovector, int
                vectorlen);
```

La primitive `send` à la forme suivante :

```
int send(int s, const void *msg, size_t len, int flags);
```

**s** Désigne l'entier renvoyé par la primitive `socket`.

**msg** Donne l'adresse du début de la suite d'octets à transmettre.

**len** Spécifie le nombre d'octets à transmettre.

**flags** Ce drapeau permet de paramétrer la transmission du data-gramme, notamment si le buffer d'écriture est plein ou si l'on désire, par exemple et avec TCP, faire un envoi en urgence (**out-of-band**) :

|          |   |  |
|----------|---|--|
| 0        | : | Non opérant, c'est le cas le plus courant.   |
| MSG_OOB  | : | Pour envoyer ou recevoir des messages <b>out-of-band</b> .   |
| MSG_PEEK | : | Permet d'aller voir quel message on a reçu sans le lire, c'est à dire sans qu'il soit effectivement retiré des buffers internes (ne s'applique qu'à <b>recv</b> (page 262)). |

### 3.4.2 Envoi en mode datagramme

Les deux autres primitives, **sendto** et **sendmsg** donnent la possibilité d'envoyer un message via une socket en mode non connecté. Toutes deux réclament que l'on spécifie le destinataire à chaque appel.

```
ssize_t sendto(int s, const void *msg, size_t len, int flags, const struct
               sockaddr *to, socklen_t tolen);
```

Les quatre premiers arguments sont exactement les mêmes que pour **send**, les deux derniers permettent de spécifier une adresse et sa longueur avec une structure du type **sockaddr**, comme vu précédemment avec **bind**.

Le programmeur soucieux d'avoir un code plus lisible pourra utiliser la deuxième primitive :

```
ssize_t sendmsg(int sockfd, const struct msghdr *messagestruct, int flags);
```

Où **messagestruct** désigne une structure contenant le message à envoyer sa longueur, l'adresse du destinataire et sa longueur. Cette primitive est très commode à employer avec son pendant **recvmsg** car elle travaille avec la même structure.

## 3.5 Recevoir des données

Symétriquement aux cinq primitives d'envoi, il existe cinq primitives de réception : `read`, `readv`, `recv`, `recvfrom`, `recvmsg`.

### 3.5.1 Reception en mode connecté

La forme conventionnelle `read` d'Unix n'est possible qu'avec des sockets en mode connecté car son retour dans le code utilisateur ne s'accompagne d'aucune précision quant à l'adresse de l'émetteur. Sa forme d'utilisation est :

```
ssize_t read(int descripteur, void *buffer, size_t longueur);
```

Bien sur, si cette primitive est utilisée avec les sockets BSD, le descripteur est l'entier renvoyé par un appel précédent à la primitive `socket`. `buffer` et `longueur` spécifie respectivement le buffer de lecture et la longueur de ce que l'on accepte de lire.

Chaque lecture ne renvoie pas forcément le nombre d'octets demandés, mais peut être un nombre inférieur.

Mais le programmeur peut aussi employer le `readv`, avec la forme :

```
ssize_t readv(int descripteur, const struct iovec *iov, int vectorlen);
```

Avec les mêmes caractéristiques que pour le `readv`.

En addition aux deux primitives conventionnelles, il y a trois primitives nouvelles pour lire des messages sur le réseau :

```
ssize_t recv(int s, void *buf, size_t len, int flags);
```

|                    |  |
|--------------------|--|
| <code>s</code>     | : L'entier qui désigne la socket.                                |
| <code>buf</code>   | : Une adresse où l'on peut écrire, en mémoire.                   |
| <code>len</code>   | : La longueur du buffer.   |
| <code>flags</code> | : Permet au lecteur d'effectuer un contrôle sur les paquets lus. |

### 3.5.2 Recevoir en mode datagramme

```
ssize_t recvfrom(int s, void *buf, size_t len, int flags, struct sockaddr  
*from, socklen_t *fromlen);
```

Les deux arguments additionnels par rapport à `recv` sont des pointeurs vers une structure de type `sockaddr` et sa longueur. Le premier contient l'adresse de l'émetteur. Notons que la primitive `sendto` fournit une adresse dans le même format, ce qui facilite les réponses.

La dernière primitive `recvmsg` est faite pour fonctionner avec son homologue `sendmsg` :

```
ssize_t recvmsg(int sockfd, struct msghdr *messagestruct, int flags);
```

La structure `messagestruct` est exactement la même que pour `sendmsg` ainsi ces deux primitives sont faites pour fonctionner de paire.

### 3.6 Spécifier une file d'attente

Imaginons un serveur en train de répondre à un client, si une requête arrive d'un autre client, le serveur étant occupé, la requête n'est pas prise en compte, et le système refuse la connexion.

La primitive `listen` est là pour permettre la mise en file d'attente des demandes de connexions.

Elle est généralement utilisée après les appels de `socket` et de `bind` et immédiatement avant le `accept`.

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

**sockfd** : l'entier qui décrit la socket.

**backlog** : Le nombre de connexions possibles en attente (quelques dizaines). La valeur maximale est fonction du paramétrage du noyau. Sous FreeBSD la valeur maximale par défaut est de 128 (sans paramétrage spécifique du noyau), alors que sous Solaris 10, " There is currently no backlog limit ".

Le nombre de fois où le noyau refuse une connexion est comptabilisé et accessible au niveau de la ligne de commande via le résultat de l'exécution de la commande `netstat -s -p tcp` (chercher " listen queue overflow "). Ce paramètre est important à suivre dans le cas d'un serveur très sollicité.

### 3.7 Accepter une connexion

Accepter une connexion est typique de la démarche d'un serveur sur le réseau.

nous l'avons examiné, un serveur utilise les primitives `socket`, `bind` et `listen` pour se préparer à recevoir les connexions. Il manque cependant à ce trio le moyen de dire au protocole " j'accepte désormais les connexions entrantes ". La primitive `accept` est le chaînon manquant !

Quand le serveur invoque cette primitive, le noyau est prévenu que le processus est en attente d'un événement réseau le concernant. Le retour dans le code de l'application ne fait que sous deux conditions, réception d'une demande de connexion ou réception d'un signal par le processus.

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

Qui s'utilise comme suit :

```
int newsock ;  
newsock = accept(sockfd, addr, addrlen) ;
```

`sockfd` descripteur de la socket, renvoyé par la primitive du même nom.

**addr** Un pointeur sur une structure du type **sockaddr**.

**addrlen** Un pointeur sur un entier.

Quand une connexion arrive, les couches sous-jacentes du protocole de transport remplissent la structure **addr** avec l'adresse du client qui fait la demande de connexion. **Addrlen** contient alors la longueur de cette adresse. Cette valeur peut être modifiée par le noyau lorsque la primitive est utilisée avec des sockets d'autres type pour lesquelles la taille de la structure d'adresse est variable (**sockaddr\_un** pour les sockets locales par exemple), ce qui justifie un pointeur là où nous ne pourrions attendre qu'un simple passage d'argument par valeur.

Puis le système crée une nouvelle socket par clonage de celle transmise et pour laquelle il renvoie un descripteur, récupéré ici dans **newsock**. Par cet artifice, la socket originale reste disponible pour d'éventuelles autres connexions (elle est clonée avant que le quintuplet soit complet).

En conclusion, lorsqu'une demande de connexion arrive, l'appel à la primitive **accept** redonne la main au code utilisateur.

### 3.8 Terminer une connexion

Dans le cas du mode connecté on termine une connexion avec la primitive **close** ou **shutdown**.

```
int close(descripteur) ;
```

La primitive bien connue sous Unix peut être aussi employée avec un descripteur de socket. Si cette dernière est en mode connecté, le système assure que tous les octets en attente de transmission seront effectivement transmis dans de bonnes conditions. Normalement cet appel retourne immédiatement, cependant le kernel peut attendre le temps de la transmission des derniers octets (transparent).

Le moyen le plus classique de terminer une connexion est de passer par la primitive **close**, mais la primitive **shutdown** permet un plus grand contrôle sur les connexions en “ full-duplex ”.

```
int shutdown(int sockfd, int how) ;
```

**Sockfd** est le descripteur à fermer, **how** permet de fermer partiellement le descripteur suivant les valeurs qu'il prend :

- 0** Aucune donnée ne peut plus être reçue par la socket.
- 1** Aucune donnée ne peut plus être émise.
- 2** Combinaison de 0 et de 1 (équivalent de **close**).

Enfin, pour une socket en mode connecté, si un processus est interrompu de manière inopinée (réception d'un signal de fin par exemple), un “ reset ” (voir page 92) est envoyé à l'hôte distant ce qui provoque la fin brutale de la connexion. Les octets éventuellement en attente sont perdus.

## 4 Schéma général d'une session client–serveur

Il est temps de donner un aperçu de la structure d'un serveur et d'un client, mettant en œuvre les APIs vus dans ce chapitre, et de rapprocher les événements réseaux de ceux observables sur le système et dans le processus qui s'exécute.

**Relation client–serveur en mode connecté :**

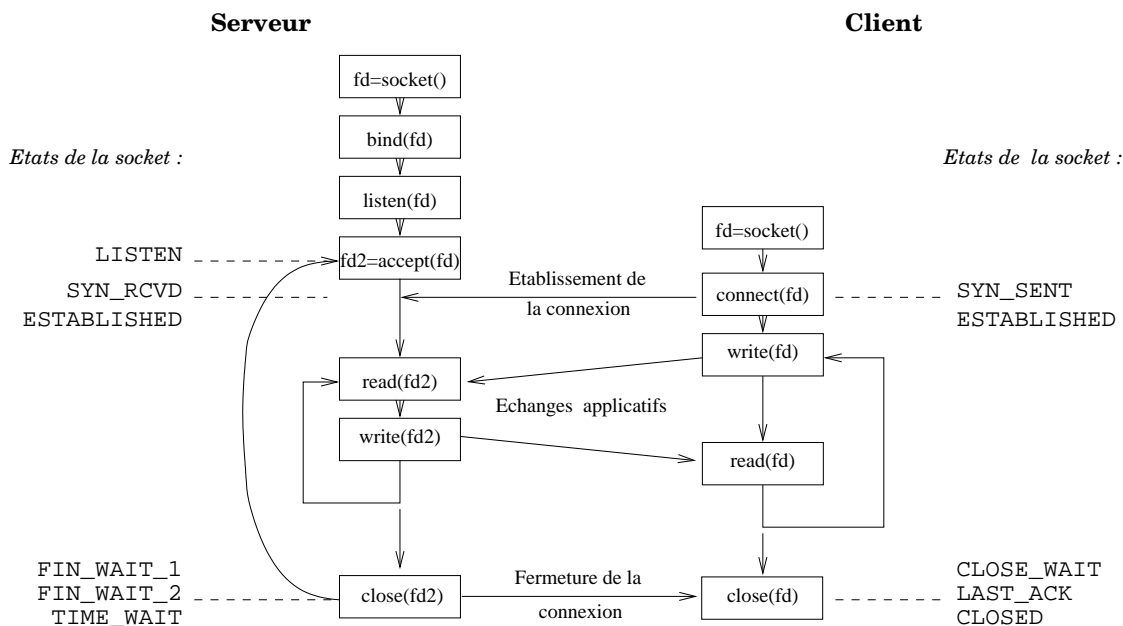


figure XII.04 — Relation client–serveur en mode connecté

Il faut établir une comparaison entre cette figure et les figures VI.03 page 94 et VI.04 page 95. Les sockets coté client ou coté serveur, si elles participent à l'établissement d'un canal de communication symétrique en fonctionnement, ne passent pas par les mêmes états, de leur création jusqu'au recyclage des octets qui les composent.

La RFC 793 précise 11 états pour une socket et la figure ci-dessus les met en situation de manière simplifiée. Ces états peuvent être visualisés avec la commande `netstat -f inet [-a]`, dans la colonne **state** de la sortie.

**LISTEN** La socket du serveur est en attente passive d'une demande de connexion (ouverture passive).

**SYN-SENT** C'est l'état de la socket cliente qui a envoyé le premier paquet de demande d'une connexion avec un flag SYN mais non encore acquitté (ouverture active).

**SYN-RCVD** La socket du serveur a reçu un paquet de demande de connexion, l'acquitte et envoi sa propre demande de connexion. Elle attend l'acquittement de sa demande.

**ESTABLISHED** Les demandes de connexions sont acquittées aux deux extrémités. La connexion est établie. La totalité du trafic TCP applicatif s'effectue dans cet état. Sa durée est indéfinie, la clôture est à l'initiative des applications.

**FIN-WAIT-1** Celui qui est à l'initiative de l'envoi du premier paquet de demande de fin est dans cet état (fermeture active).

**FIN-WAIT-2** On a reçu l'acquittement à la demande de fin de connexion.

**TIME-WAIT** La socket était en **FIN-WAIT-2** et a reçu la demande de fin de la socket distante. On doit attendre le temps suffisant pour être certain que la socket distante a bien reçu l'acquittement (re-émission sinon). Cet état peut donc être long dans le temps,  $2 \times MSL$  précise la RFC 793. Cette constante peut aller de quelques dizaines de secondes à une ou deux minutes selon les implémentations.

**CLOSE-WAIT** La socket était en **ESTABLISHED** et a reçu une demande de fin. Cet état perdure jusqu'à ce que la socket envoie à son tour une demande de fin (fermeture passive).

**CLOSING** Si la réponse à une demande de fin s'accompagne immédiatement de la demande de fin de la socket locale, cet état remplace **FIN-WAIT-1** et **FIN-WAIT-2**.

**LAST-ACK** La dernière demande de fin est en attente du dernier acquittement.

**CLOSED** État de fin. Les octets de la socket vont être recyclés.

L'état **TIME-WAIT** est supporté par celui qui clôt la connexion. Les architectures de serveurs préfèrent une clôture à l'initiative du serveur, ce qui se comprend du point de vue de l'efficacité (rester maître de la durée de la communication), mais le fonctionnement interne du protocole TCP implique ce temps d'attente. Sur un serveur très chargé les sockets dans cet état peuvent être en très grand nombre (des dizaines de milliers...) bloquant ainsi les nouvelles connexions entrantes.

#### Relation client-serveur en mode non connecté :

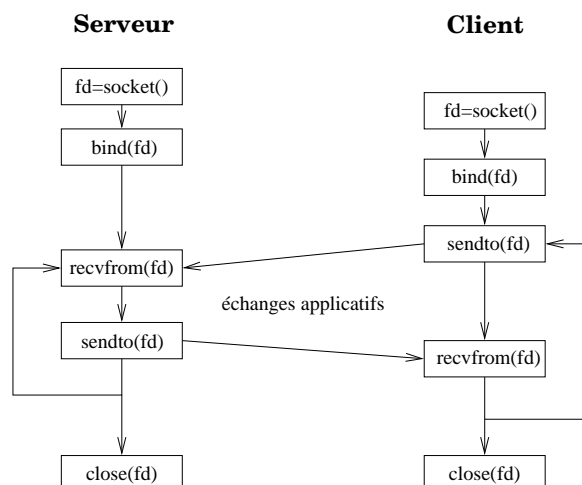


figure XII.05 — Relation client-serveur en mode non connecté



## 5 Exemples de code “ client ”

L’objectif de ces deux exemples est de montrer le codage en C et le fonctionnement d’un client du serveur de date (RFC 867, “ daytime protocol ”) présent sur toute machine unix<sup>10</sup>.

Ce serveur fonctionne en mode connecté ou non, sur le port 13 qui lui est réservé (/etc/services). Ici le serveur est une machine portant l’adresse IP 192.168.52.232. La connaissance de l’adresse IP du client n’est absolument pas utile pour la compréhension de l’exemple.

En mode TCP le simple fait de se connecter provoque l’émission de la chaîne ascii contenant la date vers le client et sa déconnexion.

En mode UDP il suffit d’envoyer un datagramme quelconque (1 caractère) au serveur puis d’attendre sa réponse.

### 5.1 Client TCP “ DTCPcli ”

Exemple d’usage :

```
$ ./DTCPcli 192.168.52.232
Date(192.168.52.232) = Wed Dec 10 20:59:46 2003
```

Une capture des trames réseau échangées lors de l’exécution de cette commande se trouve page 270.

**ligne 29** Déclaration de la structure `saddr` du type `sockaddr_in`, à utiliser avec IPv4. Attention, il s’agit bien d’une structure et non d’un pointeur de structure.

**ligne 35** La variable `sfd`, reçoit la valeur du descripteur de socket. Celle-ci est dédiée au protocole TCP.

**ligne 39** Le champ `sin_family` de la structure `saddr` indique que ce qui suit (dans la structure) concerne IPv4.

**ligne 40** Le champ `sin_port` est affecté à la valeur du numéro de port sur lequel écoute le serveur. Il faut remarquer l’usage de la fonction `htons` (en fait une macro du pré-processeur `cpp`) qui s’assure que ce numéro de port respecte bien le NBO (“ Network Byte Order ”), car cette valeur est directement recopiée dans le champ PORT DESTINATION du protocole de transport employé (voir page 84 pour UDP et page 91 pour TCP).

Nous en dirons plus sur `htons` au chapitre suivant.

Si le processeur courant est du type “ little endian ” (architecture Intel par exemple) les octets sont inversés (le NBO est du type “ big endian ”). Vous croyez écrire 13 alors qu’en réalité pour le réseau vous avez écrit 3328 (0x0D00) ce qui bien évidemment ne conduit pas au même résultat, sauf si un serveur de date écoute également sur le port 3328, non conventionnel donc très peu probable à priori.

---

<sup>10</sup>Son activation est éventuellement à faire à partir du serveur de serveur `inetd`, page 320

En résumé, si le programmeur n'utilise pas la fonction `htons`, ce code n'est utilisable que sur les machines d'architecture "big endian".

```

1  /* $Id: DTCPcli.c 92 2009-02-12 17:39:44Z fla $
2  *
3  * Client TCP pour se connecter au serveur de date (port 13 - RFC 867).
4  * La syntaxe d'utilisation est : DTCPcli <adresse ip sous forme décimale>
5  *
6  */
7
8  #include <stdio.h>
9  #include <stdlib.h>
10 #include <string.h>
11 #include <unistd.h>
12 #include <sys/types.h>
13 #include <sys/socket.h>
14 #include <sys/param.h>
15 #include <netinet/in.h>
16 #include <arpa/inet.h>
17
18 #define USAGE      "Usage:%s adresse IP du serveur\n"
19 #define MAXMSG      1024
20 #define NPORT       13
21
22 int
23 main(int argc, char *argv[])
24 {
25     int          n, sfd ;
26     char          buf[MAXMSG] ;
27     struct        sockaddr_in saddr ;
28
29     if (argc != 2) {
30         (void)fprintf(stderr,USAGE,argv[0]) ;
31         exit(EX_USAGE) ;
32     }
33     if ((sfd = socket(PF_INET,SOCK_STREAM,IPPROTO_TCP)) < 0) {
34         perror("socket") ;
35         exit(EX_OSERR) ;
36     }
37     saddr.sin_family = AF_INET ;
38     saddr.sin_port   = htons(NPORT) ;
39     if (inet_pton(AF_INET,argv[1],&saddr.sin_addr) != 1) {
40         (void)fprintf(stderr,"Address «%s» is not parseable !\n",argv[1]) ;
41         exit(EX_DATAERR) ;
42     }
43     if (connect(sfd,(struct sockaddr *)&saddr,sizeof saddr) < 0) {
44         perror("connect") ;
45         exit(EX_OSERR) ;
46     }
47     if ((n = read(sfd, buf,MAXMSG-1)) < 0) {
48         perror("read") ;
49         exit(EX_OSERR) ;
50     }
51     buf[n] = '\0' ;
52     (void)printf("Date(%s) = %s\n",argv[1],buf) ;
53     exit(EX_OK) ;
54 }
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
2559
2560
2561
2562
25
```

porter un caractère ascii 0, donc interprétable comme le caractère de fin de chaîne du langage C.

C’est pourquoi à cet endroit on ne peut pas employer les habituelles fonctions de la bibliothèque standard (celles qui commencent par `str`).

Ici le problème se complique un peu dans la mesure où l’on dispose au départ d’une adresse IP sous sa forme décimale pointée. La gestion d’erreur protège le code des erreurs de syntaxe à la saisie.

La fonction `inet_pton` gère parfaitement ce cas de figure. Nous en dirons plus à son sujet au chapitre suivant.

**ligne 45** Appel à la primitive `connect` pour établir la connexion avec le serveur distant. Quand celle-ci retourne dans le code du programme, soit la connexion a échoué et il faut traiter l’erreur, soit la connexion est établie. L’échange préliminaire des trois paquets s’est effectué dans de bonnes conditions (page 94).

Du point de vue TCP, les cinq éléments du quintuplet qui caractérisent la connexion sont définis (page 90).

Sur la capture des paquets de la page 270 nous sommes arrivés à la ligne 6, c’est à dire l’envoi de l’acquittement par le client du paquet de synchronisation envoyé par le serveur (ligne 3 et 4).

Il faut noter que bien que nous ayons transmis la structure `saddr` par adresse (caractère `&`) et non par valeur, la primitive `connect` ne modifie pas son contenu pour autant.

Notons également l’usage du “ `cast` ” du C pour forcer le type du pointeur (le prototype de la primitive exige à cet endroit un pointeur de type `sockaddr`).

**ligne 49** Appel à la primitive `read` pour lire le résultat en provenance du serveur, à l’aide du descripteur `sfd`.

Sur la capture d’écran on voit ligne 8 (et 9) l’envoi de la date en provenance du serveur, d’une longueur de 26 caractères.

Ce nombre de caractères effectivement lus est affecté à la variable `n`.

Ce nombre ne peut excéder le résultat de l’évaluation de `MAXMSG - 1`, qui correspond à la taille du buffer `buf` moins 1 caractère prévu pour ajouter le caractère 0 de fin de chaîne.

En effet, celui-ci fait partie de la convention de représentation des chaînes de caractères du langage C. Rien ne dit que le serveur qui répond ajoute un tel caractère à la fin de sa réponse. Le contraire est même certain puisque la RFC 867 n’y fait pas mention.

Remarque : le buffer `buf` est largement surdimensionné compte tenu de la réponse attendue. La RFC 867 ne prévoit pas de taille maximum si ce n’est implicitement celle de l’expression de la date du système en anglais, une quarantaine d’octets au maximum.

**ligne 53** Ajout du caractère de fin de chaîne en utilisant le nombre de caractères renvoyés par `read`.

**ligne 55** La sortie du programme induit une clôture de la socket coté client. Coté serveur elle est déjà fermée (séquence `write + close`) comme on peut le voir ligne 8 (flag FP, page 92) ci-après dans la capture du trafic entre le client et le serveur.

Remarque : rien n'est explicitement prévu dans le code pour établir la socket coté client, à savoir l'association d'un numéro de port et d'une adresse IP. En fait c'est la primitive `connect` qui s'en charge. L'adresse IP est celle de la machine. Le numéro de port est choisi dans la zone d'attribution automatique comme nous l'avons examiné page 85.

Il existe bien entendu une possibilité pour le programme d'avoir connaissance de cette information : la primitive `getsockname`.

```

1 23:03:29.465183 client.2769 > serveur.daytime: S 2381636173:2381636173(0)
2    win 57344 <mss 1460,nop,wscale 0,nop,nop,timestamp 299093825 0> (DF)
3 23:03:29.465361 serveur.daytime > client.2769: S 3179731077:3179731077(0)
4    ack 2381636174 win 57344 <mss 1460,nop,wscale 0,nop,nop,timestamp
5      4133222 299093825> (DF)
6 23:03:29.465397 client.2769 > serveur.daytime: . ack 1 win 57920
7      <nop,nop,timestamp 299093826 4133222> (DF)
8 23:03:29.466853 serveur.daytime > client.2769: FP 1:27(26) ack 1 win 57920
9      <nop,nop,timestamp 4133223 299093826> (DF)
10 23:03:29.466871 client.2769 > serveur.daytime: . ack 28 win 57894
11      <nop,nop,timestamp 299093826 4133223> (DF)
12 23:03:29.467146 client.2769 > serveur.daytime: F 1:1(0) ack 28 win 57920
13      <nop,nop,timestamp 299093826 4133223> (DF)
14 23:03:29.467296 serveur.daytime > client.2769: . ack 2 win 57920
15      <nop,nop,timestamp 4133223 299093826> (DF)

```

### *Trafic “ daytime ” TCP, capturé avec tcpdump*

Un autre exemple d'interrogation, mais avec un autre hôte du même LAN mais sur lequel le service `daytime` n'est pas en fonctionnement :

```

$ ./DTCPcli 192.168.52.232
connect: Connection refused

```

```

1 16:13:21.612274 IP client.57694 > serveur.daytime: S 2248945646:2248945646(0)
2    win 65535 <mss 1460,nop,nop,sackOK,nop,wscale 1,nop,nop,
3      timestamp 360942290 0>
4 16:13:21.612648 IP serveur.daytime > client.57694: R 0:0(0) ack 2248945647 win 0

```

### *Trafic “ daytime ” TCP (reset), capturé avec tcpdump*

L'envoi d'un reset (drapeau R) envoyé par le serveur en guise de réponse est bien visible ligne 4.

## 5.2 Client UDP “ DUDPcli ”

```

1  /* $Id: DUDPcli.c 92 2009-02-12 17:39:44Z fla $
2  *
3  * Client UDP pour se connecter au serveur de date (port 13 - RFC 867).
4  * La syntaxe d'utilisation est : DUDPcli <adresse ip sous forme décimale>
5  *
6  */
7  #include <stdio.h>
8  #include <stdlib.h>
9  #include <string.h>
10 #include <unistd.h>
11 #include <sysexits.h>
12
13 #include <sys/types.h>
14 #include <sys/socket.h>
15 #include <sys/param.h>
16 #include <netinet/in.h>
17 #include <arpa/inet.h>
18
19 #define  USAGE  "Usage:%s adresse IP du serveur\n"
20 #define  MAXMSG 1024
21 #define  NPORT  13
22
23 int
24 main(int argc, char *argv[])
25 {
26     int          n, sfd ;
27     char          buf[MAXMSG] ;
28     struct        sockaddr_in saddr ;
29
30     if (argc != 2) {
31         (void)fprintf(stderr,USAGE,argv[0]) ;
32         exit(EX_USAGE) ;
33     }
34     if ((sfd = socket(PF_INET,SOCK_DGRAM,IPPROTO_UDP)) < 0) {
35         perror("socket") ;
36         exit(EX_OSERR) ;
37     }
38     saddr.sin_family      = AF_INET ;
39     saddr.sin_port        = htons(NPORT) ;      /* Attention au NBO ! */
40     if (inet_pton(PF_INET,argv[1],&saddr.sin_addr) != 1) {
41         (void)fprintf(stderr,"Address «%s» is not parseable !\n",argv[1]) ;
42         exit(EX_DATAERR) ;
43     }
44     buf[0] = '\0' ;
45     if (sendto(sfd,buf,1,0,(struct sockaddr *)&saddr, sizeof saddr) != 1) {
46         perror("sendto") ;
47         exit(EX_OSERR) ;
48     }
49     if ((n=recv(sfd,buf,MAXMSG-1,0)) < 0) {
50         perror("recv") ;
51         exit(EX_OSERR) ;
52     }
53     buf[n] = '\0' ;
54     (void)printf("Date(%s) = %s\n",argv[1],buf) ;
55     exit(EX_OK) ;
56 }

```

*Source du client “ DUDPcli.c ”*

Exemple d'usage :

```

$ ./DUDPcli 192.168.52.232
Date(192.168.52.232) = Wed Dec 10 20:56:58 2003

```

**ligne 34** Ouverture d'une socket UDP, donc en mode non connecté.

**ligne 45** Envoi d'un caractère (NULL) au serveur, sans quoi il n'a aucun moyen de connaître notre existence.

**ligne 38, 39 et 40** Le remplissage de la structure `saddr` est identique à celui de la version TCP.

**ligne 49** Réception de caractères en provenance du réseau.

Il faut remarquer que rien n'assure que les octets lus sont bien en provenance du serveur de date interrogé.

Nous aurions pu utiliser la primitive `recvfrom` dont un des arguments est une structure d'adresse contenant justement l'adresse de la socket qui envoie le datagramme (ici la réponse du serveur).

Le raisonnement sur la taille du buffer est identique à celui de la version TCP.

La capture de trames suivante montre l'extrême simplicité de l'échange en comparaison avec celle de la version utilisant TCP !

```

1
2
3
4  23:23:17.668689 client.4222 > serveur.daytime: udp 1
5  23:23:17.670175 serveur.daytime > client.4222: udp 26
```

*Trafic " daytime " UDP, capturé avec tcpdump*

Un autre essai avec la machine 192.168.52.233 qui ne répond pas plus sur le port 13 en UDP :

```

1  16:29:42.090816 IP client.55822 > serveur.daytime: UDP, length: 1
2  16:29:42.091205 IP serveur > client: icmp 36: serveur udp port daytime
3                                     unreachable
```

*Trafic " daytime " UDP (icmp), capturé avec tcpdump*

Et le code client reste bloqué en lecture, malgré l'envoi d'un code ICMP qui n'est pas interprété par défaut par `recv`... Pour éviter une telle situation de blocage, il faudrait configurer la socket en lui ajoutant un délai au delà duquel elle retourne dans le code du client avec un code spécifique d'erreur<sup>11</sup>.

<sup>11</sup>voir la page de manuel de `setsockopt` assortie du paramètre `SO_RCVTIMEO`

## 6 Conclusion et Bibliographie

En conclusion on peut établir le tableau suivant :

|                               | Protocole           | Adresses locale<br>et N° de port. | Adresse éloignée<br>et N° de port. |
|-------------------------------|---------------------|-----------------------------------|------------------------------------|
| Serveur orienté connexion     | <code>socket</code> | <code>bind</code>                 | <code>listen, accept</code>        |
| Client orienté connexion      | <code>socket</code> |                                   | <code>connect</code>               |
| Serveur non orienté connexion | <code>socket</code> | <code>bind</code>                 | <code>recvfrom</code>              |
| Client non orienté connexion  | <code>socket</code> | <code>bind</code>                 | <code>sendto</code>                |

**RFC 867** “ Daytime Protocol ”. J. Postel. May-01-1983. (Format : TXT=2405 bytes) (Also STD0025) (Status : STANDARD)

**RFC 793** “ Transmission Control Protocol. ” J. Postel. September 1981. (Format : TXT=172710 bytes) (Updated by RFC3168) (Also STD0007) (Status : STANDARD)

**RFC 3493** “ Basic Socket Interface Extensions for IPv6 ”. R. Gilligan, S. Thomson, J. Bound, J. McCann, W. Stevens. February 2003. (Format : TXT=82570 bytes) (Obsoletes RFC2553) (Status : INFORMATIONAL)

Pour en savoir davantage, outre les pages de `man` des primitives citées dans ce chapitre, on pourra consulter les documents de référence suivants :

- Stuart Sechrest — “ An Introductory 4.4BSD Interprocess Communication Tutorial ” — Re imprimé dans “ Programmer’s Supplementary Documents ” — O’Reilly & Associates, Inc. — 1994<sup>12</sup>
- W. Richard Stevens — “ Unix Network Programming ” — Prentice All — 1990
- W. Richard Stevens — “ Unix Network Programming ” — Second edition — Prentice All — 1998
- W. Richard Stevens – Bill Fenner, Andrew M. Rudoff — “ Unix Network Programming ” — Third Edition — Addison Wesley — 2003
- Douglas E. Comer – David L. Stevens — “ Internetworking with TCP/IP – Volume III ” (BSD Socket version) — Prentice All — 1993
- Stephen A. Rago — “ Unix System V Network Programming ” — Addison–Wesley — 1993

Et pour aller plus loin dans la compréhension des mécanismes internes :

- W. Richard Stevens — “ TCP/IP Illustrated Volume 2 ” — Prentice All — 1995
- McKusick – Bostic – Karels – Quateman — “ The Design and implementation of the 4.4 BSD Operating System ” — Addison–Wesley — 1996

---

<sup>12</sup>On peut également trouver ce document dans le répertoire `/usr/share/doc/psd/20.ipctut/` des OS d’inspiration Berkeley 4.4





# Chapitre XIII

## Compléments sur les sockets Berkeley

### 1 Réserveation des ports

Au chapitre précédent nous avons utilisé la primitive `bind` pour assigner une adresse à une socket, dans ce paragraphe nous précisons comment choisir le numéro de port qui va bien, selon le type d'application envisagé. Nous avons déjà examiné ce point dans les grandes lignes page 85.

Il y a deux manières d'assigner un N° de port à une socket :

1. Le processus spécifie le numéro. C'est typiquement ce que fait un serveur. On suppose bien évidemment que les clients sont au courant ou qu'ils disposent d'un mécanisme qui peut les renseigner (cf cours sur les RPC).
2. Le processus laisse le système lui assigner automatiquement un numéro de port. C'est typiquement ce que fait un client, sauf cas contraire exigé par le protocole d'application (cf cours sur les “ remote execution ”).

En règle générale le développeur d'application ne s'attribue pas au hasard un (ou plus) numéro de port. Il doit respecter quelques contraintes comme ne pas utiliser les ports déjà attribués. Ceux-ci figurent dans une RFC particulière. La dernière en date est la RFC 1700 [Reynolds & Postel 1994] au paragraphe “ WELL KNOWN PORT NUMBERS ”. Plus simplement, sur toute machine Unix à jour, une liste de ces ports se trouve dans le fichier `/etc/services`<sup>1</sup>.

Codé sur deux octets non signés, le numéro de port permet 65536 possibilités de 0 à 65535. Cette échelle est fragmentée de deux manières, l'ancienne ou la nouvelle méthode. Toutes les applications sur tous les systèmes d'exploitation n'ont pas encore adopté la nouvelle approche, les deux vont donc cohabiter un certain temps, ne serait-ce qu'à cause d'applications plus anciennes non encore mises à jour...

---

<sup>1</sup><http://www.iana.org/assignments/port-numbers> pour se tenir au courant des évolutions de cette liste

## 1.1 Réserve de port — Ancienne méthode

**Port N° 0** Ce numéro n'est pas utilisable pour une application, c'est une sorte de "jocker" qui indique au système que c'est à lui de compléter automatiquement le numéro (voir plus loin **de 1024 à 5000**).

**Port de 1 à 255** Pour utiliser cette zone il faut avoir les droits du `root` à l'exécution (`UID = 0`) pour que le `bind` ne retourne pas une erreur. Les serveurs "classiques" (`domain`, `ftp`, `smtp`, `telnet`, `ssh`, `http`, `snmp`...) se situent tous dans cette partie.

**Ports de 256 à 511** Jadis considéré comme une "réserve" des serveurs officiels commencent à s'y installer, faute de place dans la zone précédente. Il faut également avoir un `UID = 0` pour utiliser un numéro de port dans cette zone.

**Port de 512 à 1023** Une fonction `rresvport` permet l'attribution automatique d'un numéro de port dans cette zone, pour des applications ayant un `UID = 0`. Par exemple, c'est dans cette zone qu'`inetd` (cf cours sur les serveurs) attribue des ports automatiquement pour les outils en "r" de Berkeley (`rlogin`, `rcp`, `rexec`, `rdist`,...).

**Port de 1024 à 5000** Zone d'attribution automatique par `bind`. Lorsque l'utilisateur (non `root`) utilise 0 comme numéro, c'est le premier port libre qui est employé. Si tous les utilisateurs peuvent s'attribuer "manuellement" un numéro dans cette zone, il vaut mieux éviter de le faire, la suivante est prévue pour cela.

**5001 à 65535** Zone "libre" attention cependant car de très nombreux serveurs y ont un port réservé, et pas des moindres comme le serveur `X11` sur le port 6000 !

## 1.2 Réserve de port — Nouvelle méthode

**Port N° 0** Ce numéro n'est pas utilisable pour une application, c'est une sorte de "jocker" qui indique au système que c'est à lui de compléter automatiquement le numéro (voir plus loin **de 49152 à 65535**).

**Port de 1 à 1023** Pour utiliser cette zone il faut avoir les droits du `root` à l'exécution pour que le `bind` ne retourne pas une erreur. Les serveurs "classiques" (`domain`, `ftp`, `smtp`, `telnet`, ...) se situent tous dans cette partie.

**Port de 1024 à 49151** est la zone des services enregistrés par l'IANA et qui fonctionnent avec des droits ordinaires.

**Port de 49152 à 65535** est la zone d'attribution automatique des ports, pour la partie cliente des connexions (si le protocole n'impose pas une valeur particulière) et pour les tests de serveurs locaux.

## 2 Ordre des octets sur le réseau

Nous reprenons ici un point déjà évoqué page 48 :

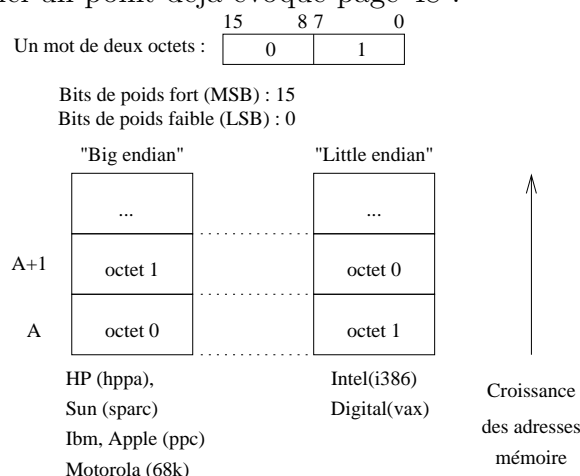


figure XIII.01 — Ordre des octets sur le réseau

Le problème de l'ordre des octets sur le réseau est d'autant plus crucial que l'on travaille dans un environnement avec des architectures hétérogènes.

La couche *Réseau* (page 30) ne transforme pas les octets de la couche *Internet* (page 30) qui elle même ne modifie pas ceux de la couche de *Transport*<sup>2</sup> (page 29).

Pour cette raison, le numéro de port inscrit dans l'en-tête *TCP* (vs *UDP*) de l'émetteur est exploité tel quel par la couche de transport du récepteur et donc il convient de prendre certaines précautions pour s'assurer que les couches de même niveau se comprennent.

D'un point de vue plus général, les réseaux imposent le " poids fort " avant le " poids faible ", c'est le " Network Byte Order ". Les architectures qui travaillent naturellement avec cette représentation n'ont donc théoriquement pas besoin d'utiliser les fonctions qui suivent, de même pour les applications qui doivent dialoguer avec d'autres ayant la même architecture matérielle. Néanmoins écrire du code " portable " consiste à **utiliser ces macros dans tous les cas**<sup>3</sup> !

Pour se souvenir de ces fonctions, il faut connaître la signification des quatre lettres utiles :

|          |             |  |
|----------|-------------|--|
| <b>s</b> | " short "   | Entier court sur 16 bits, un numéro de port par exemple. |
| <b>l</b> | " long "    | Entier long sur 32 bits, une adresse IP par exemple.     |
| <b>h</b> | " host "    | La machine sur laquelle s'exécute le programme.          |
| <b>n</b> | " network " | Le réseau sur lequel on envoie les données.              |

<sup>2</sup>Nous n'aborderons pas ici la question de la transmission de données hétérogènes au niveau applicatif, elle sera examinée dans le cours sur les *XDR*

<sup>3</sup>Pour les machines qui respectent naturellement le *NBO*, comme les stations *HP* (processeur risc) ou *SUN* (processeur sparc), *IBM* (processeur ppc) ces fonctions sont des macros " vides " contrairement à toutes les architectures de type *i386*

D'où les prototypes :

```
#include <sys/types.h>
u_long  htonl  (u_long); /* host to network --- long */
u_short htons  (u_short); /* host to network --- short */
u_long  ntohl  (u_long); /* network to host --- long */
u_short ntohs  (u_short); /* network to host --- short */
```

Par exemple, pour affecter le numéro de port 13 (service “daytime”) au champ `sin_port` d'une structure de type `sockaddr_in` :

```
saddr.sin_port = htons(13);
```

Cette écriture est valable quelle que soit l'architecture sur laquelle elle est compilée. S'il avait fallu se passer de la macro `htons` sur une architecture Intel (“little endian”), pour l'affectation du même numéro de port, il eut fallu écrire :

```
saddr.sin_port = 0x0D00; /* 0D hexadécimal == 13 décimal */
```

### 3 Opérations sur les octets

Dans le même ordre d'idée qu'au paragraphe précédent, les réseaux interconnectent des architectures hétérogènes et donc aussi des conventions de représentation des chaînes de caractères différentes. Pour être plus précis, le caractère NULL marqueur de fin de chaîne bien connu des programmeurs C, n'est pas valide partout, voire même est associé à une autre signification !

En conséquence, pour toutes les fonctions et primitives qui lisent et écrivent des octets sur le réseau, les chaînes de caractères sont toujours associées au nombre de caractères qui les composent.

Le corollaire est que les fonctions “classiques” de manipulation de chaînes en C (`strcpy`, `strcat`, ...) ne sont à utiliser qu'avec une extrême prudence.

Pour copier des octets d'une zone vers une autre il faut utiliser `bcopy`, pour comparer deux buffers, `bcmp`, enfin pour mettre à zéro (remplir d'octets NULL) une zone, `bzero`.

```
#include <string.h>
void bcopy (const void *src, void *dst, size_t len);
int  bcmp  (const void *b1, const void *b2, size_t len);
void bzero (const void *b, size_t len);
```

**bcopy** Attention, `len` octets de `src` sont copiés dans `dst` et non l'inverse, comme dans `strcpy`.

**bcmp** Compare `b1` et `b2` et renvoie 0 si les `len` premiers octets sont identiques, sinon une valeur non nulle qui n'est pas exploitable vis à vis d'une quelconque relation d'ordre (à la différence de `strcmp` qui suppose les caractères dans la table *ASCII*).

**bzero** Met des octets NULL (0) `len` fois à l'adresse `b`.

Il existe des outils similaires, issus du système V : `memcpy`, `memcmp`, `memset`, ...

## 4 Conversion d'adresses

La plupart du temps, par exemple dans une forme de saisie pour la configuration d'un appareil, les adresses IP sont fournies par l'utilisateur dans le format "décimal pointé", or la structure d'adresse (`sockaddr_in`) a besoin d'un entier non signé sur 32 bits qui respecte le NBO. Une conversion est donc nécessaire pour passer d'une représentation à une autre.

### 4.1 Conversion d'adresse - IPv4 seul

La fonction `inet_addr` convertit une adresse décimale pointée en un entier long non signé et qui respecte le NBO. La fonction `inet_ntoa` effectue le travail inverse. Ces deux fonctions ne sont valables que pour les adresses 32 bits d'IPv4 et sont présentes dans la majeure partie des codes réseaux. Dans le cadre d'un nouveau développement on leur préférera les fonctions décrites après.

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

in_addr_t  inet_addr  (char *); /* in_addr_t == unsigned long */
char *     inet_ntoa  (struct in_addr);
```

Remarque : Ces deux fonctions ne sont pas symétriques, le fait que `inet_addr` ne renvoie pas une structure du type `in_addr` n'est pas une incohérence mais est dû au fait que la structure `in_addr` était prévue au départ pour évoluer.

Exemple d'utilisation :

```
struct sockaddr_in saddr ;
if ((saddr.sin_addr.s_addr = inet_addr("138.195.52.130")) != INADDR_NONE)
    printf("Adresse IP = %s\n",inet_ntoa(saddr.sin_addr)) ;
else
    printf("Erreur sur l'argument de 'inet_addr' !\n" );
```

Il faut juste noter qu'en cas d'erreur la fonction `inet_addr` renvoie la constante `INADDR_NONE`.

### 4.2 Conversion d'adresse - Compatible IPv4 et IPv6

Avec les mêmes fichiers d'en-tête, les deux nouvelles fonctions de conversion :

```
const char *  inet_ntop  (int af, const void *src, char *dst, size_t size);
int           inet_pton  (int af, const char *src, void *dst);
```

Le **p** signifie "presentation", comprendre lisible par l'humain, alors que le **n** signifie "numeric", c'est à dire compréhensible par le noyau (entier qui respecte le NBO). Donc **ntop** convertit le format système vers l'humain et **pton** effectue le travail inverse.

Du fait de leur compatibilité entre les familles de protocoles, ces fonctions sont un peu plus compliquées à l'usage : Il faut préciser `PF_INET` ou `PF_INET6`.

Exemple d'utilisation :

```
struct sockaddr_in saddr ;
if (inet_pton(PF_INET,"138.195.52.130",&saddr.sin_addr) != 1)
    (void)fprintf(stderr,"L'adresse n'est pas exploitable !\n") ;
else {
    char adr[INET_ADDRSTRLEN] ; /* 16 == 4 x 3(digits) + 3 (.) + 1 (NULL) */
    printf("Adresse IP = %s\n",inet_ntop(PF_INET,&saddr.sin_addr,adr,sizeof(adr))) ;
}
```

Il faut noter que le code de retour de la fonction `inet_pton` peut prendre les valeurs -1, 0 ou 1. 1 signifie que transformation l'adresse transcodée est utilisable.

Le code de retour de `inet_ntop` est soit NULL si la conversion a échoué, ou un pointeur sur la chaîne affichable. Ici, par construction, on suppose que la conversion sera toujours réussie.

## 5 Conversion hôte – adresse IPv4

Se pose régulièrement le problème de convertir un nom symbolique en une adresse IP et inversement. L'origine de cette information dépend de la configuration de la station de travail : c'est un serveur de noms (DNS), c'est un fichier (`/etc/hosts`) ou tout autre système de propagation des informations (*NIS*...). Dans tous les cas l'information arrive à un programme via une entité nommée le *resolver*, qui unifie les sources d'informations.

Les paragraphes 5.1, 5.2 (p. 282), 6.1 (p. 282) et 6.2 (p. 284) présentent une approche traditionnelle, seulement valable avec IPv4 alors que le paragraphe 7 (p. 285) expose une démarche beaucoup plus récente et adaptée également à IPv6. L'écriture de nouveaux codes ne devraient faire appel qu'à cette nouvelle api.

### 5.1 Une adresse IP à partir d'un nom d'hôte

```
#include <netdb.h>
struct hostent * gethostbyname (char *name);
struct hostent {
    char *h_name ;          /* Le nom officiel */
    char **h_aliases ;      /* Tableau de synonymes */
    int h_addrtype ;        /* PF_INET pour ARPA */
    int h_length ;          /* Long. de l'adresse */
    char **h_addr_list ;    /* Adresses possibles */
} ;
#define h_addr h_addr_list[0]
```

la macro `h_addr` sert à assurer la compatibilité avec les premières versions dans lesquelles il n'y avait qu'une seule adresse IP possible par hôte.

Le nom “ officiel ” s’oppose aux noms synonymes. Par exemple, soit une machine officiellement baptisée `pasglop.mon-domain.fr` ; si pour répondre au besoin d’une certaine application l’administrateur du réseau lui donne le surnom `www.mon-domain.fr`, celui-ci sera considéré comme un “ alias ” vis à vis du nom officiel et donc lu dans `h_aliases`. (voir page 185)

```

1  /*
2   * $Id: gethostbyname.c 46 2007-12-03 19:39:16Z fla $
3   * Exemple d'utilisation de la fonction "gethostbyname".
4   */
5  #include <stdio.h>
6  #include <stdlib.h>
7  #include <sys/types.h>
8  #include <sys/socket.h> /* AF_INET */
9  #include <netinet/in.h> /* struct in_addr */
10 #include <netdb.h> /* gethostbyname */
11 #include <arpa/inet.h> /* inet_ntoa */
12 #define DIT_FAMILY(x) (x)==AF_UNSPEC?"AF_UNSPEC":(x)==AF_UNIX?"AF_UNIX":\
13 (x)==AF_INET?"AF_INET":(x)==AF_INET6?"AF_INET6": "other..."
14 #define USAGE "Usage:%s liste de machines distantes\n"
15
16 void
17 impnet(struct in_addr **list)
18 {
19     struct in_addr *adr ;
20     while ((adr = *list++))
21         (void)printf("Adresse Internet : %s\n", inet_ntoa(*adr)) ;
22 }
23
24 int
25 main(int argc, char *argv[])
26 {
27     register char *ptr ;
28     register struct hostent *pth ;
29     if (argc < 2) {
30         (void)fprintf(stderr, USAGE, argv[0]) ;
31         exit(EX_USAGE) ;
32     }
33     while (--argc > 0) {
34         ptr = *++argv ;
35         if (!(pth = gethostbyname(ptr))) {
36             (void)fprintf(stderr, "%s : hôte inconnu !\n", ptr) ;
37             exit(EX_SOFTWARE) ;
38         }
39         (void)printf("Nom officiel : %s\n", pth->h_name) ;
40         while ((ptr = *(pth->h_aliases)) != NULL) {
41             (void)printf("alias : %s\n", ptr) ;
42             pth->h_aliases++ ;
43         }
44         (void)printf("Type d'adresse : %s\n", DIT_FAMILY(pth->h_addrtype)) ;
45         if (pth->h_addrtype == PF_INET)
46             impnet((struct in_addr **)pth->h_addr_list) ;
47         else
48             (void)printf("Type d'adresse non reconnu !\n") ;
49     }
50     exit(EX_OK) ;
51 }
52

```

*gethostbyname.c*

La fin du tableau de pointeurs est marquée par un pointeur NULL.

La liste des adresses est un tableau de pointeurs, le marqueur de fin de liste est également un pointeur NULL. Chaque adresse est une zone de `h_length` octets (cf fonction `impnet` dans l'exemple ci-après).

Le programme d'usage qui suit affiche toutes les informations contenues dans cette structure pour les hôtes dont le nom est passé en argument (le code source de cet exemple, `gethostbyname.c`, est à la page suivante).

```
$ gethostbyname gw-sio.sio.ecp.fr srv-sio.sio.ecp.fr
Nom officiel      : gw-sio.sio.ecp.fr
Type d'adresse    : AF_INET
Adresse Internet  : 138.195.52.33
Adresse Internet  : 138.195.53.1
Adresse Internet  : 138.195.10.52
Nom officiel      : srv-sio.sio.ecp.fr
Type d'adresse    : AF_INET
Adresse Internet  : 138.195.52.130

$ gethostbyname anna.sio.ecp.fr
anna.sio.ecp.fr : hote inconnu !
```

## 5.2 Un nom d'hôte à partir d'une adresse IP

le problème inverse se résout avec la fonction `gethostbyaddr`. La définition du prototype se trouve au même endroit que précédemment, la fonction renvoie un pointeur sur une structure du type `hostent`.

```
#include <netdb.h>

struct hostent * gethostbyaddr (char *addr, int len, int type);
    addr    : Pointe sur une structure du type in_addr
    len     : Est la longueur de addr
    type    : PF_INET quand on utilise la pile ARPA
```

## 6 Conversion N° de port – service

Couramment les ports bien connus sont donnés par leur nom plutôt que par leur valeur numérique, comme par exemple dans les sorties de la commande `tcpdump`.

### 6.1 Le numéro à partir du nom

Un tel programme a besoin de faire la conversion symbolique — numérique, la fonction `getservbyname` effectue ce travail. L'utilisateur récupère un pointeur sur une structure du type `servent`, NULL dans le cas d'une impossibilité. La source d'informations se trouve dans le fichier `/etc/services`.

```
#include <netdb.h>
```



```

struct servent * getservbyname (char *name, char *proto);

struct servent {
    char      *s_name ;
    char      **s_aliases ;
    int       s_port ;
    char      *s_proto ;
} ;

```

**s\_name** Le nom officiel du service.

**s\_aliases** Un tableau de pointeurs sur les aliases possibles. Le marqueur de fin de tableau est un pointeur à NULL.

**s\_port** Le numéro du port (il respecte le *Network Byte Order*).

**s\_proto** Le nom du protocole à utiliser pour contacter le service (*TCP* vs *UDP*).

Voici un programme de mise en application de la fonction, le code source de l'exemple, `getservbyname.c` se trouve à la page suivante.

```
$ getservbyname domain
```

Le service domain est reconnu dans /etc/services

protocole :tcp - N° de port :53

```
$ getservbyname domain udp
```

Le service domain est reconnu dans /etc/services

protocole :udp - N° de port :53

```

1  /*
2   * $Id: getservbyname.c 47 2007-12-03 19:41:04Z fla $
3   * Exemple d'utilisation de la fonction "getservbyname".
4   */
5  #include <stdio.h>
6  #include <stdlib.h>
7  #include <sysexits.h>
8  #include <sys/types.h>
9  #include <netinet/in.h> /* Pour "ntohs" */
10 #include <netdb.h>      /* Pour "getservbyname" */
11
12 #define USAGE "Usage:%s <nom de service> [<nom de protocole>]\n"
13 #define MSG1  "Le service %s est reconnu dans /etc/services\nprotocole :%s - N° de port :%d\n"
14 #define MSG2  "Le service %s (%s) est introuvable dans /etc/services !\n"
15
16 int
17 main(int argc, char *argv[])
18 {
19     struct servent *serv ;
20
21     if (argc < 2) {
22         (void)fprintf(stderr, USAGE, argv[0]) ;
23         exit(EX_USAGE) ;
24     }
25     if ((serv = getservbyname(argv[1], argv[2]?argv[2]:"tcp")))
26         (void)printf(MSG1, serv->s_name, serv->s_proto, ntohs(serv->s_port)) ;
27     else
28         (void)printf(MSG2, argv[1], argv[2]?argv[2]:"") ;
29     exit(EX_OK) ;
30 }

```

*getservbyname.c*

## 6.2 Le nom à partir du numéro

Symétriquement la fonction `getservbyport` effectue le travail inverse. Elle renvoie aussi un pointeur sur une structure `servent`, NULL dans le cas d'une impossibilité.

```
#include <netdb.h>
struct servent * getservbyport (int port, char *proto);
```

Exemples d'usage :

```
$ getservbyport 53
```

Le port 53 correspond au service "domain" (protocole tcp).

```
$ getservbyport 53 udp
```

Le port 53 correspond au service "domain" (protocole udp).

Exemple de programmation :

```
1  /*
2   * $Id: getservbyport.c 2 2007-09-12 20:00:17Z fla $
3   * Exemple d'utilisation de la fonction "getservbyport".
4   */
5  #include <stdio.h>
6  #include <stdlib.h>      /* Pour "atoi".          */
7  #include <sysexits.h>
8  #include <sys/types.h>
9  #include <netinet/in.h> /* Pour "ntohs"          */
10 #include <netdb.h>      /* Pour "getservbyport" */
11
12 #define USAGE      "Usage:%s <numéro de port> [<nom de protocole>]\n"
13 #define MSG1       "Le port %d correspond au service \"%s\" (protocole %s).\n"
14 #define MSG2       "Le port %s (%s) est introuvable dans /etc/services !\n"
15
16 int
17 main(int argc, char *argv[])
18 {
19     struct servent *serv ;
20     if (argc < 2) {
21         (void)fprintf(stderr, USAGE, argv[0]) ;
22         exit(EX_USAGE) ;
23     }
24     if ((serv = getservbyport(atoi(argv[1]), argv[2]?argv[2]:"tcp"))
25         (void)printf(MSG1, ntohs(serv->s_port), serv->s_name, serv->s_proto) ;
26     else
27         (void)printf(MSG2, argv[1], argv[2]?argv[2]:"") ;
28     exit(EX_OK) ;
29 }
```

*getservbyport.c*

## 7 Getaddrinfo, pour IPv4 et IPv6

Les apis des deux paragraphes qui précèdent (`gethostbyname` et `getservbyname` et leur symétrique) sont des standards de facto et ce depuis le début des années 80. On les trouve sur toutes les variantes d'Unix et même au delà, ce qui a participé à une grande portabilité du code écrit qui les utilise.

L'arrivée d'IPv6 et de sa probable très longue cohabitation avec IPv4 oblige à modifier les habitudes de programmation au profit d'une nouvelle approche, que ces concepteurs souhaitent aussi stable et largement répandue que la précédente. L'écriture de tout nouveau code devrait s'appuyer sur cette nouvelle API, définie dans la RFC 3493.

La nouvelle fonction `getaddrinfo` de la libc ne se contente pas seulement de synthétiser `gethostbyname` et `getservbyname` en une seule fonction, elle banalise également l'usage d'IPv6.

La démarche est plus concise (une fonction au lieu d'une), et la manipulation des adresses IP est rendue plus aisée en retour, puisque que la structure de données utilisée par la fonction contient directement une structure d'adresse conforme à la famille de protocole utilisée, `sockaddr_in` pour IPv4, directement utilisable par exemple avec les primitives `bind`, `connect`,... On peut songer par comparaison au champ `h_addr_list` de la structure `hostent` (page 280) qui ne contient que les adresses IP.

### 7.1 La fonction `getaddrinfo`

La fonction `getaddrinfo` combine les fonctionnalités de `gethostbyname` et `getservbyname` pour les protocoles IPv4 et IPv6. Son prototype est donc le reflet de sa relative complexité, par contre son fonctionnement est très logique, il découle de celui des deux fonctions qu'elle remplace.

#### 7.1.1 Prototype de `getaddrinfo`

```
#include <netdb.h>
int getaddrinfo(const char *hostname, const char *servname,
                const struct addrinfo *hints, struct addrinfo **res);
```

Comme il y a beaucoup d'informations à transmettre et à recevoir, tout (ou presque) s'effectue via une nouvelle structure de données nommée `addrinfo`. Elle apparaît en troisième argument (informations fournies à l'appel) et quatrième (dernier) argument, le résultat.

Si cette fonction ne retourne pas une valeur nulle (0), le code d'erreur est à exploiter à l'aide d'une fonction spécialisée, `gai_strerror` qui ajoute une bonne dizaine de messages d'erreurs supplémentaires spécialisés.

### 7.1.2 Description des arguments

Les deux premiers arguments, `hostname` ou `servname`, sont soit des chaînes de caractères, soit un pointeur nul. Il ne peuvent pas être tous les deux nuls.

**hostname** Les valeurs acceptables sont soit un nom d'hôte valide ou une adresse IP exprimée sous sa forme décimale pointée.

**servname** est soit un nom, soit un numéro de service présent dans `/etc/services`.

**hints** La structure est optionnelle (NULL dans ce cas) et permet de piloter finement le comportement de la fonction. L'explication de son usage passe par un survol de la constitution de sa structure.

**res** Le dernier argument est un pointeur de pointeur sur une structure du même type que `hints`. C'est par ce moyen que la fonction va renvoyer le résultat construit, en modifiant la valeur du pointeur (il nous faut passer l'adresse du pointeur puisque sa valeur va changer, d'où le pointeur de pointeur).

La fonction alloue la mémoire nécessaire pour le stockage des données, au programme appelant de la restituer au système. Il dispose à cet effet d'une fonction spécialisée : `freeaddrinfo`.

### 7.1.3 La structure `addrinfo`

Voici les membres de la structure `addrinfo`, définis dans le fichier d'entêtes `netdb.h` :

```
struct addrinfo {
    int      ai_flags;          /* AI_PASSIVE, AI_CANONNAME, AI_NUMERICHOST */
    int      ai_family;        /* PF_xxx */
    int      ai_socktype;      /* SOCK_xxx */
    int      ai_protocol;      /* 0 or IPPROTO_xxx for IPv4 and IPv6 */
    socklen_t ai_addrlen;      /* length of ai_addr */
    char     *ai_canonname;    /* canonical name for hostname */
    struct    sockaddr *ai_addr; /* binary address */
    struct    addrinfo *ai_next; /* next structure in linked list */
};
```

Signalons tout de suite au lecteur le dernier membre de la structure, `ai_next`. Il est du même type que celui de la structure elle-même (structure auto-référente en langage C) ce qui signifie qu'il peut pointer vers une autre structure du même type. Il s'agit alors d'une liste chaînée<sup>4</sup> afin de retourner une information multiple, comme peut l'être par exemple la réponse d'une requête DNS (type A ou PTR en l'occurrence), ou la liste des protocoles

<sup>4</sup>Type Abstrait de Données (TAD) très répandu, cf : [http://fr.wikipedia.org/wiki/Liste\\_chaine](http://fr.wikipedia.org/wiki/Liste_chaine)

prévus pour un service donné. La fin de la liste est marquée par un pointeur de valeur nulle (NULL).

La structure `hints` doit être mise à zéro avant chaque usage. Les quatre premiers champs sont utilisés à l'appel, les autres éléments sont à zéro lors de la transmission à la fonction.

**ai\_family** Pour indiquer la famille de protocole utilisée. C'est le même argument que celui qu'on utilise en première position avec la primitive `socket`, cf page 253.

Il peut prendre la valeur `PF_UNSPEC` quand le protocole n'est pas fixé.

**ai\_socktype** Pour préciser le type de socket demandé, c'est à dire le mode connecté ou datagramme. C'est le deuxième argument de la primitive `socket`.

Si la valeur est laissée à 0 cela signifie que n'importe quel protocole est accepté.

**ai\_protocol** Pour préciser le nom du protocole. C'est le troisième argument de `socket`. Même remarque que précédemment concernant la valeur 0.

**ai\_flags** Ce drapeau est éventuellement une combinaison de valeurs binaires assemblées avec l'opérateur `|` du langage C (ou inclusif).

**AI\_ADDRCONFIG** Seules les adresses (IPv4 ou IPv6) configurées sur le système local sont retournées.

**AI\_ALL** Combiné avec **AI\_V4MAPPED** donne toutes les adresses IPv4 et IPv6. Sans effet si **AI\_V4MAPPED** n'est pas présent.

**AI\_CANONNAME** Si l'information est accessible (DNS...) le champ `ai_canonname` de la première structure `res` pointe sur le nom canonique de `hostname`.

**AI\_NUMERICHOST** Précise que `hostname` doit être traité comme une adresse IP délivrée sous sa forme numérique, c'est à dire décimale pointée pour IPv4.

**AI\_NUMERICSERV** Indique que si `servname` est un port donné sous forme numérique (la chaîne encode la représentation du nombre, comme par exemple "23").

**AI\_PASSIVE** Sert pour l'attribution automatique du numéro de port de la structure d'adresse, `sockaddr_in` pour IPv4, accessible via le pointeur générique `sockaddr`.

**AI\_V4MAPPED** Utilisé avec IPv6 (**AF\_INET6**).

#### 7.1.4 En résumé

Il y a 6 variables à configurer, non toutes utiles selon les utilisations. Il est évident que certaines des nombreuses possibilités offertes par la combinatoire ne sont pas consistante. Le bon sens doit prédominer et le test de retour de la fonction être toujours exploité...!

### 7.1.5 Exemple d'usage à la place de gethostbyname

Cet exemple remplace celui de la page 280. Le code n'est pas plus simple.

```

1  /*
2  * $Id$
3  * Exemple d'utilisation de la fonction "getaddrinfo" avec IPv4
4  * et en remplacement de 'gethostbyname'
5  */
6  #include <stdio.h>
7  #include <stdlib.h>
8  #include <sysexits.h>
9  #include <strings.h>
10 #include <sys/types.h>
11 #include <sys/socket.h> /* AF_INET */
12 #include <netinet/in.h> /* struct in_addr */
13 #include <netdb.h> /* getaddrinfo */
14 #include <arpa/inet.h> /* inet_ntop */
15
16 #define USAGE "Usage: %s liste de machines distantes\n"
17 #define DIT_FAMILY(x) ((x)==AF_UNSPEC?"AF_UNSPEC":(x)==AF_UNIX?"AF_UNIX":\
18 (x)==AF_INET?"AF_INET":(x)==AF_INET6?"AF_INET6":"other...")
19
20 int
21 main(int argc, char *argv[])
22 {
23     char *pt ;
24     int ret ;
25     char buf[INET_ADDRSTRLEN] ;
26     struct addrinfo profil, *lstres, *lstres0 ;
27     if (argc < 2) {
28         (void)fprintf(stderr, USAGE, argv[0]) ;
29         exit(EX_USAGE) ;
30     }
31     while (--argc > 0) {
32         pt = *++argv ;
33         bzero(&profil, sizeof(profil)) ;
34         profil.ai_flags = AI_CANONNAME ;
35         profil.ai_family = AF_INET ;
36         profil.ai_socktype = SOCK_DGRAM ;
37         if ((ret=getaddrinfo(pt, NULL, &profil, &lstres0)) {
38             (void)fprintf(stderr, "%s", gai_strerror(ret)) ;
39             exit(EX_SOFTWARE) ;
40         }
41         (void)printf ("Nom officiel : %s\n", lstres0->ai_canonname) ;
42         (void)printf ("Type d'adresse : %s\n", DIT_FAMILY(lstres0->ai_socktype)) ;
43         for (lstres=lstres0; lstres!=NULL; lstres=lstres->ai_next)
44             (void)printf ("Adresse Internet : %s\n", \
45                          inet_ntop(lstres->ai_socktype,
46                                     &((struct sockaddr_in*)lstres->ai_addr)->sin_addr,
47                                     buf, sizeof buf));
48     }
49     freeaddrinfo(lstres0) ; /* Ne sert à rien ici ! */
50     exit(EX_OK) ;
51 }

```

*getaddrinfo\_1.c*

**Ligne 13** Il faut inclure ce fichier d'en-têtes pour avoir le prototype de `getaddrinfo`.

**Ligne 26** Déclaration d'une structure de type `addrinfo` et de deux pointeurs du même type.

**Ligne 31** On décrémente avant de faire le test. Donc quand `argc` passe par 0 on sort de la boucle. La dernière valeur utilisée de `argc` est 1.

**Ligne 32** `pt` pointe sur `argv[1]`, puis sur `argv[2]`, etc... On utilise `argc` valeurs très exactement.

**Lignes 33** Mise à zéro de tous les bits de la structure

**Lignes 34, 35 et 36** On veut les noms canoniques, pour IPv4. L'usage de `SOCK_DGRAM` est un artifice pour éviter d'avoir deux réponses, une avec TCP et l'autre avec UDP.

**Ligne 37** Ne pas oublier de conserver le code de retour pour pouvoir éventuellement l'exploiter à l'aide de la fonction `gai_strerror`, comme **ligne 38**.

**Ligne 41 et 42** Affichage des informations de la première structure (`lstres0`).

**Ligne 43** Boucle `for` pour explorer tous les éléments de la liste chaînée. La condition d'arrêt est de rencontrer un pointeur nul.

**Ligne 44, 45, 46 et 47** Affichage de l'adresse IP extraite de la structure d'adresse, et mise en forme par la fonction `inet_ntop`.

Notez l'utilisation des éléments de la structure d'information pour compléter les arguments d'appel de `inet_ntop`. Il faut utiliser un cast (`struct sockaddr_in *`) afin d'accéder au champ `sin_addr` de la structure d'adresse IPv4. La structure `ai_addr` est générique et n'y donne pas accès.

**Ligne 50** Restitution de la mémoire allouée, pour l'exemple parce que le noyau va de toute manière recycler toute la mémoire du processus lors de l'opération de fin provoquée **ligne 51**.

### 7.1.6 Exemple d'usage à la place de getservbyname

Cet exemple remplace celui de la page 282

```

1  /*
2  * $Id$
3  * Exemple d'utilisation de la fonction "getaddrinfo" avec IPv4
4  * et en remplacement de 'getservbyname'
5  */
6  #include <stdio.h>
7  #include <stdlib.h>
8  #include <sysexits.h>
9  #include <strings.h>
10 #include <sys/types.h>
11 #include <sys/socket.h> /* AF_INET */
12 #include <netdb.h> /* getaddrinfo */
13
14 #define USAGE "Usage:%s <nom de service>\n"
15
16 int
17 main(int argc, char *argv[])
18 {
19     int ret ;
20     struct addrinfo profil,*lstres,*lstres0 ;
21     struct protoent *myproto ;
22     if (argc < 2) {
23         (void)fprintf(stderr,USAGE,argv[0]) ;
24         exit(EX_USAGE) ;
25     }
26     bzero(&profil,sizeof(profil)) ;
27     profil.ai_family = AF_INET ;
28     if ((ret=getaddrinfo(NULL,argv[1],&profil,&lstres0))) {
29         (void)fprintf(stderr,"%s\n",gai_strerror(ret)) ;
30         exit(EX_SOFTWARE) ;
31     }
32     for (lstres=lstres0;lstres !=NULL;lstres=lstres->ai_next) {
33         myproto = getprotobynumber(lstres->ai_protocol) ;
34         (void)printf("Port:%d - Protocole:%s\n",\
35                     ntohs(((struct sockaddr_in*)lstres->ai_addr)->sin_port),\
36                             myproto->p_name) ;
37     }
38     freeaddrinfo(lstres0) ; /* Ne sert à rien ici ! */
39     exit(EX_OK) ;
40 }

```

*getaddrinfo\_2.c*

**Ligne 27** On précise IPv4.

**Ligne 33** Usage de la fonction `getprotobynumber` dont le prototype est décrit au paragraphe 8 page 291 et qui sert à retrouver la valeur symbolique d'un protocole, connaissant son codage numérique (fichier `/etc/protocols`).

**Ligne 34, 35 et 36** Affichage du numéro de port et du nom du protocole. Notez l'usage de la fonction `ntohs` pour présenter les octets du numéro de port dans " le bon ordre " !

### 7.1.7 En résumé

...



## 8 Conversion nom de protocole – N° de protocole

Les fonctions `getservbyname` et `getservbyport` délivrent un nom de protocole, donc un symbole.

Lors de la déclaration d'une `socket` le troisième argument est numérique, il est donc nécessaire d'avoir un moyen pour convertir les numéros de protocoles (`IPPROTO_UDP`, `IPPROTO_TCP`, `IPPROTO_IP`, `IPPROTO_ICMP`, ...) en symboles et réciproquement.

Le fichiers `/etc/protocols` contient cette information, et la paire de fonctions `getprotobyname` et `getprotobynumber` l'exploitent.

```
#include <netdb.h>
struct protoent *getprotobyname (const char *name) ;
struct protoent *getprotobynumber (int proto) ;
struct protoent {
    char *p_name ;
    char **p_aliases ;
    int p_proto ;
} ;
```

**p\_name** Le nom officiel du protocole.

**p\_aliases** La liste des synonymes, le dernier pointeurs est NULL.

**p\_proto** Le numéro du protocole, dans `/etc/services`.

```
1  /*
2  * $Id: getprotobyname.c 2 2007-09-12 20:00:17Z fla $
3  * Exemple d'utilisation de la fonction "getprotobyname".
4  */
5  #include <stdio.h>
6  #include <sysexits.h>
7  #include <netdb.h> /* Pour getservbyname */
8
9  #define USAGE "Usage:%s <nom du protocole>\n"
10 #define MSG1 "Le protocole %s est reconnu dans /etc/protocols – N° : %d\n"
11 #define MSG2 "Le protocole %s est introuvable dans /etc/protocols !\n"
12
13 int
14 main(int argc, char *argv[])
15 {
16     struct protoent *proto ;
17
18     if (argc < 2) {
19         (void)fprintf(stderr, USAGE, argv[0]) ;
20         exit(EX_USAGE) ;
21     }
22     if (proto = getservbyname(argv[1]))
23         (void)printf(MSG1, proto->p_name, proto->p_proto) ;
24     else
25         (void)printf(MSG2, argv[1]) ;
26     exit(EX_OK) ;
27 }
```

*getprotobyname.c*

Le source qui précède donne un exemple de programmation de la fonction `getprotobyname`. Usage de ce programme :

```
$ getprotobyname ip
Le protocole ip est reconnu dans /etc/protocols - N° : 0
$ getprotobyname tcp
Le protocole tcp est reconnu dans /etc/protocols - N° : 6
$ getprotobyname vmtcp
Le protocole ipv6 est reconnu dans /etc/protocols - N° : 41
```

## 9 Diagnostic

Chaque retour de primitive devrait être soigneusement testé, le code généré n'en est que plus fiable et les éventuels dysfonctionnements plus aisés à détecter.

Quelques unes des erreurs ajoutées au fichier d'en-tête `errno.h`

| erreur          | Description de l'erreur                     |
|-----------------|---|
| ENOTSOCK        | Le descripteur n'est pas celui d'une socket |
| EDESTADDRREQ    | Adresse de destination requise              |
| EMSGSIZE        | Le message est trop long                    |
| EPROTOTYPE      | Mauvais type de protocole pour une socket   |
| ENOPROTOOPT     | Protocole non disponible                    |
| EPROTONOSUPPORT | Protocole non supporté                      |
| ESOCKTNOSUPPORT | Type de socket non supporté                 |
| EOPNOTSUPP      | Opération non supportée                     |
| EAFNOSUPPORT    | Famille d'adresse non supportée             |
| EADDRINUSE      | Adresse déjà utilisée                       |
| EADDRNOTAVAIL   | L'adresse ne peut pas être affectée         |
| ENETDOWN        | Réseau hors service                         |
| ENETUNREACH     | Pas de route pour atteindre ce réseau       |
| ENETRESET       | Connexion coupée par le réseau              |
| ECONNABORTED    | Connexion interrompue                       |
| ECONNRESET      | Connexion interrompue par l'hôte distant    |
| ENOBUFS         | Le buffer est saturé                        |
| EISCONN         | La socket est déjà connectée                |
| ENOTCONN        | La socket n'est pas connectée               |
| ESHUTDOWN       | Transmission après un <b>shutdown</b>       |
| ETIMEDOUT       | " time-out " expiré                         |
| ECONNREFUSED    | Connexion refusée                           |
| EREMOTERELEASE  | L'hôte distant a interrompue sa connexion   |
| EHOSTDOWN       | L'hôte n'est pas en marche                  |
| EHOSTUNREACH    | Pas de route vers cet hôte                  |

## 10 Exemples de mise en application

### 10.1 Ancienne méthode (usage de gethostbyname)

Deux premiers exemples de fonctions de connexion à un serveur : `tcp_open` et `udp_open`. Toutes les deux renvoient un descripteur de `socket` prêt à l'emploi, ou -1 en cas d'erreur (le message d'erreur doit être généré par les fonctions elles mêmes). Ces deux fonctions sont basées sur l'usage de l'api `gethosbyname`, en voici les prototypes :

```
int tcp_open(char *host, char *service, int port) ;
int udp_open(char *host, char *service, int port, int conn) ;
```

Description des arguments :

**host** Une chaîne de caractères qui est l'adresse de l'hôte distant. Cette adresse est soit sous forme décimale pointée, soit c'est un nom symbolique. Elle ne peut pas être nulle.

**service** Si cette chaîne n'est pas nulle, elle contient le nom symbolique du service distant sur lequel il faut se connecter.

**port** Le numéro du port distant. S'il est négatif alors **service** est obligatoirement non nulle. Dans le cas où **service** est non nulle et `port > 0`, cette dernière valeur l'emporte sur celle trouvée dans le fichier `/etc/services`.

**conn** Cet argument n'a d'usage que pour la fonction `udp_open`. Égal à un, il précise que la socket créée est dédiée au serveur **host** (avec un `bind`), ie on pourra employer `send` et `recv` au lieu de `sendto` et `recvfrom`.

```
1  /* $Id: open_tcp1.c 2 2007-09-12 20:00:17Z fla $
2  *
3  *  Fonction "tcp_open" + exemple d'utilisation avec "daytime"
4  *  et "time".
5  */
6  #include      <stdio.h>
7  #include      <unistd.h>
8  #include      <string.h>
9  #include      <errno.h>
10 #include      <sysexits.h>
11 #include      <sys/types.h>
12 #include      <sys/socket.h>
13 #include      <netinet/in.h>
14 #include      <netdb.h>
15 #include      <arpa/inet.h>
16
17 #define        USAGE      "Usage:%s <nom de machine distante>\n"
18
19 struct sockaddr_in  tcp_serv_addr ; /* Adresse Internet du serveur. */
20 struct servent      tcp_serv_info ; /* Infos de "getservbyname". */
21 struct hostent      tcp_host_info ; /* Infos de "gethostbyname". */
```

```

1  int
2  tcp_open(char *host,char *service,int port)
3  {
4      int                fd ;
5      unsigned long      inaddr ;
6      struct hostent      *hp ;
7      struct servent      *sv ;
8
9      bzero ((char *)&tcp_serv_addr, sizeof tcp_serv_addr) ;
10     tcp_serv_addr.sin_family = AF_INET ;
11     if (*service) {
12         if (!(sv = getservbyname(service,"tcp"))) {
13             (void)fprintf(stderr,"tcp_open:service inconnu:%s/tcp\n",service) ;
14             return -1 ;
15         }
16         tcp_serv_info = *sv ;
17         if (port > 0)
18             tcp_serv_addr.sin_port = htons(port) ;
19         else
20             tcp_serv_addr.sin_port = sv->s_port ;
21     }
22     else {
23         if (port <= 0) {
24             (void)fprintf(stderr,"tcp_open:numéro de port non spécifié !\n" ) ;
25             return -1 ;
26         }
27         tcp_serv_addr.sin_port = htons(port) ;
28     }
29
30     if ((inaddr = inet_addr(host)) != INADDR_NONE) { /* netid.hostid ? */
31         bcopy((char *)&inaddr,(char *)&tcp_serv_addr.sin_addr,sizeof inaddr)
32         tcp_host_info.h_name = (char *)NULL ;
33     }
34     else {
35         if (!(hp = gethostbyname(host))) {
36             (void)fprintf(stderr,"tcp_open: erreur de nom de machine : %s : %s\n",
37                 host,strerror(errno)) ;
38             return -1 ;
39         }
40         tcp_host_info = *hp ;
41         bcopy(hp->h_addr,(char *)&tcp_serv_addr.sin_addr,hp->h_length) ;
42     }
43
44     if ((fd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
45         (void)fprintf(stderr,"tcp_open: impossible de créer la socket !\n" ) ;
46         return -1 ;
47     }
48     if (connect(fd,(struct sockaddr *)&tcp_serv_addr,sizeof tcp_serv_addr)<0)
49         (void)fprintf(stderr,"tcp_open: impossible de se connecter !\n" ) ;
50     (void)close(fd) ;
51     return -1 ;
52 }
53
54 return fd ;
55 }
56
57 int
58 main(int argc,char *argv[])
59 {
60     int sfd ;
61     int n ;
62     unsigned long temps ;
63     char buf[256] ;
64

```

```

1      if (argc < 2) {
2          (void)fprintf(stderr,USAGE,argv[0]) ;
3          exit (EX_USAGE) ;
4      }
5
6      /*
7       * Connexion au serveur de date
8       */
9      if ((sfd = tcp_open(argv[1],"daytime",0)) < 0)
10         exit(EX_SOFTWARE) ;
11      if ((n=read(sfd,(void *)buf, sizeof(buf)-1))< 0)
12         perror("read/daytime") ;
13      else {
14         buf[n]='\0' ;
15         (void)printf("Date(%s)=%s",argv[1],buf) ;
16      }
17      (void)close(sfd) ;
18
19      /*
20       * Connexion au serveur de temps
21       */
22      if ((sfd = tcp_open(argv[1],"",37)) < 0)
23         exit (EX_SOFTWARE) ;
24      if (read(sfd,(void *)&temps, sizeof temps) < 0)
25         perror("read/port 37") ;
26      else
27         (void)printf("Temps(%s)=%lu\n",argv[1],ntohl(temps)) ;
28
29      exit (EX_OK) ;
30  }

```

*open\_tcp.c*

Exemple d'usage :

```

$ ./open_tcp localhost
Date   : Sun Dec  2 16:12:57 2001
Temps  : 3216294777

```

Remarque : Pour transmettre la structure d'adresse du serveur à la fonction appelante, La fonction `udp_open` complète une structure globale.

```

1  /* $Id: open_udp1.c 2 2007-09-12 20:00:17Z fla $
2   *
3   * Fonction "udp_open" + exemple d'utilisation avec "daytime"
4   * et "time".
5   */
6  #include <stdio.h>
7  #include <unistd.h>
8  #include <string.h>
9  #include <errno.h>
10 #include <sysexits.h>
11 #include <sys/types.h>
12 #include <sys/socket.h>
13 #include <netinet/in.h>
14 #include <netdb.h>
15 #include <arpa/inet.h>
16 #define USAGE "Usage:%s <nom de machine distante>\n"
17 struct sockaddr_in udp_serv_addr ; /* Adresse Internet du serveur. */
18 struct sockaddr_in udp_cli_addr ; /* Adresse Internet du client. */
19 struct servent udp_serv_info ; /* Infos de "getservbyname". */

```

```

1  int
2  udp_open(char *host,char *service,int port,int conn)
3  {
4      int fd ;
5      unsigned long inaddr ;
6      struct hostent *hp ;
7      struct servent *sv ;
8
9
10     bzero ((char *)&udp_serv_addr, sizeof udp_serv_addr) ;
11     udp_serv_addr.sin_family = AF_INET ;
12     if (*service) {
13         if (!(sv = getservbyname(service,"udp"))) {
14             (void)fprintf(stderr,"udp_open:service inconnu:%s/udp\n",service) ;
15             return -1 ;
16         }
17         udp_serv_info = *sv ;
18         if (port > 0)
19             udp_serv_addr.sin_port = htons(port) ;
20         else
21             udp_serv_addr.sin_port = sv->s_port ;
22     }
23     else {
24         if (port <= 0) {
25             (void)fprintf(stderr,"udp_open:numéro de port non spécifié !\n") ;
26             return -1 ;
27         }
28         udp_serv_addr.sin_port = htons(port) ;
29     }
30
31     if ((inaddr = inet_addr(host)) != INADDR_NONE) { /* netid.hostid ? */
32         bcopy ((char *)&inaddr,(char *)&udp_serv_addr.sin_addr,sizeof inaddr) ;
33         udp_host_info.h_name = (char *)NULL ;
34     }
35     else {
36         if (!(hp = gethostbyname(host))) {
37             (void)fprintf(stderr,"udp_open:erreur de nom de machine:%s:%s\n",
38                 host,strerror(errno)) ;
39             return -1 ;
40         }
41         udp_host_info = *hp ;
42         bcopy(hp->h_addr,(char *)&udp_serv_addr.sin_addr,hp->h_length) ;
43     }
44     if ((fd = socket(AF_INET, SOCK_DGRAM, 0)) < 0) {
45         (void)fprintf(stderr,"udp_open: impossible de créer la socket !\n") ;
46         return -1 ;
47     }
48     bzero ((char *)&udp_cli_addr,sizeof udp_cli_addr) ;
49     udp_cli_addr.sin_family = AF_INET ;
50     udp_cli_addr.sin_addr.s_addr = htonl(INADDR_ANY) ;
51     udp_cli_addr.sin_port = htons(0) ;
52     if (bind(fd,(struct sockaddr *)&udp_cli_addr,sizeof udp_cli_addr) < 0) {
53         (void)fprintf(stderr,"udp_open:erreur avec l'adresse locale !\n") ;
54         (void)close(fd) ;
55         return -1 ;
56     }
57     if (conn == 1) { /* Figer l'adresse du serveur. */
58         if (connect(fd,(struct sockaddr *)&udp_serv_addr,
59             sizeof udp_serv_addr)<0) {
60             (void)fprintf(stderr,"udp_open: connexion impossible avec %s !\n",\
61                 host)
62             (void)close(fd) ;
63             return -1 ;
64         }
65     }
66     return fd ;
67 }

```

```

1
2  int
3  main(int argc,char *argv[])
4  {
5      int sfd ;
6      int n ;
7      unsigned long temps ;
8      char buf[256] ;
9
10     if (argc < 2) {
11         (void)fprintf(stderr,USAGE,argv[0]) ;
12         exit (EX_USAGE) ;
13     }
14
15     /*
16      * Connexion au serveur de date
17      */
18     if ((sfd = udp_open(argv[1],"daytime",0,1)) < 0)
19         exit(EX_SOFTWARE) ;
20     if (send(sfd,(void *)" ",1,0) < 0)
21         perror("send") ;
22     else
23         if ((n=recv(sfd,(void *)buf, sizeof buf,0)) < 0)
24             perror("recv") ;
25         else {
26             buf[n]='\0' ;
27             (void)printf("Date(%s)=%s",argv[1],buf) ;
28         }
29     (void)close(sfd) ;
30
31     /*
32      * Connexion au serveur de temps
33      */
34     if ((sfd = udp_open(argv[1],"",37,0)) < 0)
35         exit(EX_SOFTWARE) ;
36
37     n = sizeof udp_serv_addr ;
38     if (sendto(sfd,(void *)" ",1,0,(void *)&udp_serv_addr, n) < 0)
39         perror("sendto") ;
40     else
41         if (recvfrom(sfd,(void *)&temps,sizeof temps,0,\
42                     (void *)&udp_serv_addr, &n) < 0)
43             perror("recvfrom") ;
44         else
45             (void)printf("Temps(%s)=%lu\n",argv[1],ntohl(temps)) ;
46     exit(EX_OK) ;
47 }

```

*open\_udp.c*

Exemple d'usage :

```

$ ./open_udp localhost
Date  : Sun Dec  2 16:12:17 2001
Temps : 3216294737

```

## 10.2 Nouvelle méthode (usage de getaddrinfo)

```

1  /* $Id$
2  *
3  *  Fonction "sock_open" + exemple d'utilisation avec "daytime"
4  *  et "time" en UDP et TCP.
5  */
6  #include <stdio.h>
7  #include <unistd.h>
8  #include <stdlib.h>
9  #include <sysexits.h>
10 #include <strings.h>
11 #include <sys/types.h>
12 #include <sys/socket.h>
13 #include <netinet/in.h>
14 #include <netdb.h>
15 #include <arpa/inet.h>
16
17 #define          USAGE      "Usage:%s <host distant>\n"
18
19 int
20 sock_open(char *host,char *service,int socktype)
21 {
22     int ret ;
23     int mysoc ;
24     struct addrinfo profil, *res, *lstres0 ;
25     bzero(&profil,sizeof(profil)) ;
26     profil.ai_family = AF_INET ;
27     profil.ai_socktype = socktype ;
28     profil.ai_flags = AI_PASSIVE ;
29     if ((ret = getaddrinfo(host,service,&profil,&lstres0))) {
30         (void)fprintf(stderr,"%s",gai_strerror(ret)) ;
31         return -1 ;
32     }
33     for (res=lstres0;res!=NULL;res=res->ai_next) {
34         if ((mysoc=socket(res->ai_family,res->ai_socktype,
35                          res->ai_protocol)) < 0)
36             continue ;
37         if (connect(mysoc,res->ai_addr,res->ai_addrlen) == 0)
38             break ; /* Gotcha ! */
39         (void)close(mysoc) ; /* inutilisable */
40         mysoc=-1 ;
41     }
42     freeaddrinfo(lstres0) ;
43     if (mysoc < 0)
44         (void)fprintf(stderr,"Unable to connect to %s:%s\n",host,service) ;
45     return mysoc ;
46 }
47
48 int
49 main(int argc,char *argv[])
50 {
51     int sfd ;
52     int n ;
53     unsigned long temps ;
54     char buf[256] ;
55
56     if (argc < 2) {
57         (void)fprintf(stderr,USAGE,argv[0]) ;
58         exit (EX_USAGE) ;
59     }

```

Cette nouvelle version combine `tcp_open` et `udp_open` en un seul source. La fonction `sock_open` génère un descripteur de socket prêt à l'emploi, comme



précédemment. Le main appelle quatre fois cette nouvelle fonction avec les mêmes hypothèses de travail que pour les anciennes versions.

```

1      /*
2      * Connexion au serveur de date avec UDP (socket dédiée)
3      */
4      if ((sfd = sock_open(argv[1], "daytime", SOCK_DGRAM)) >= 0) {
5          if (send(sfd, (void *) " ", 1, 0) < 0)
6              perror("send - Daytime") ;
7          else
8              if ((n=recv(sfd, (void *)buf, sizeof(buf)-1, 0)) < 0)
9                  perror("recv - Daytime") ;
10             else {
11                 buf[n]='\0' ;
12                 (void)printf("Date/udp/%s=%s", argv[1], buf) ;
13             }
14             (void)close(sfd) ;
15     }
16     /*
17     * Connexion au serveur de temps avec UDP (socket dédiée)
18     */
19     if ((sfd = sock_open(argv[1], "37", SOCK_DGRAM)) >= 0) {
20         if (send(sfd, (void *) " ", 1, 0) < 0)
21             perror("send - Time") ;
22         else
23             if (recv(sfd, (void *)&temps, sizeof temps, 0) < 0)
24                 perror("recv - Time") ;
25             else
26                 (void)printf("Temps/udp/%s=%u\n", argv[1], ntohl(temps)) ;
27             (void)close(sfd) ;
28     }
29     /*
30     * Connexion au serveur de date avec TCP
31     */
32     if ((sfd = sock_open(argv[1], "daytime", SOCK_STREAM)) >= 0) {
33         if ((n=read(sfd, (void *)buf, sizeof(buf)-1)) < 0)
34             perror("read - Daytime") ;
35         else {
36             buf[n]='\0' ;
37             (void)printf("Date/tcp/%s=%s", argv[1], buf) ;
38         }
39         (void)close(sfd) ;
40     }
41     /*
42     * Connexion au serveur de temps avec TCP
43     */
44     if ((sfd = sock_open(argv[1], "37", SOCK_STREAM)) >= 0) {
45         if (read(sfd, (void *)&temps, sizeof temps) < 0)
46             perror("read - Time") ;
47         else
48             (void)printf("Temps/tcp/%s=%u\n", argv[1], ntohl(temps)) ;
49         (void)close(sfd) ;
50     }
51
52     exit(EX_OK);
53 }

```

Exemple d'usage :

```

Date/udp/srv-sio=Mon Nov  3 19:19:11 2008
Temps/udp/srv-sio=3434725151
Date/tcp/srv-sio=Mon Nov  3 19:19:11 2008
Temps/tcp/srv-sio=3434725151

```

## 11 Conclusion et bibliographie

Outre les pages de manuel (`man`) des primitives et fonctions rencontrées, le lecteur pourra consulter avec grand profit les ouvrages suivants :

**RFC 1700** J. Reynolds, J. Postel, “ ASSIGNED NUMBERS ”, 10/20/1994. (Pages=230) (Format=.txt) (Obsoletes RFC1340) (STD 2)

Consultez surtout le site <http://www.iana.org/>

**RFC 3493** Basic Socket Interface Extensions for IPv6. R. Gilligan, S. Thomson, J. Bound, J. McCann, W. Stevens. February 2003. (Format : TXT=82570 bytes) (Obsoletes RFC2553) (Status : INFORMATIONAL)

Et des ouvrages de références :

- Samuel J. Leffler, Robert S. Fabry, William N. Joy, Phil Lapsley — “ An Advanced 4.4BSD Interprocess Communication Tutorial ” — CSRG University of California, Berkeley Berkeley, California 94720. Ce document est réédité dans “ Programmer’s Supplementary Documents ” éditeur O’Reilly, ou sous forme de fichier ascii dans le répertoire :  
`/usr/share/doc/psd/21.ipc/paper.ascii.gz`
- W. Richard Stevens — “ Unix Network Programming ” — Prentice All — 1990
- W. Richard Stevens — “ Unix Network Programming ” — Second edition — Prentice All — 1998
- W. Richard Stevens – Bill Fenner – Andrew M. Rudoff — “ Unix Network Programming ” — Third edition, volume 1 — Prentice All — 2004
- Douglas E. Comer – David L. Stevens — “ Internetworking with TCP/IP – Volume III ” (BSD Socket version) — Prentice All — 1993

# Chapitre XIV

## Éléments de serveurs

Dans ce chapitre nous abordons quelques grands principes de fonctionnement des logiciels serveurs. D’abord nous tentons de résumer leurs comportements selon une typologie en quatre modèles génériques, puis nous examinons quelques points techniques remarquables de leur architecture logicielle comme la gestion des tâches multiples, des descripteurs multiples, le fonctionnement en arrière plan (les fameux “ daemon ”), la gestion des logs...

Enfin nous concluons ce chapitre avec une présentation très synthétique du “ serveur de serveurs ” sous Unix, c’est à dire la commande `inetd`, suivie d’une lecture commentée d’un petit code en langage C qui s’inspire de son fonctionnement, pour mieux comprendre sa stratégie !

### 1 Type de serveurs

L’algorithme intuitif d’un serveur, déduit des schémas (revoir la page 265) d’utilisation des sockets, pourrait être celui-ci :

1. Créer une socket, lui affecter une adresse locale avec un numéro de port connu des clients potentiels.
2. Entrer dans une boucle infinie qui accepte les requêtes des clients, les lit, formule une réponse et la renvoie au client.

Cette démarche, que nous pourrions qualifier de naïve, ne peut convenir qu’à des applications très simples. Considérons l’exemple d’un serveur de fichiers fonctionnant sur ce mode. Un client réseau qui s’y connecte et télécharge pour 10 Go de données accapare le serveur pendant un temps significativement long, même au regard des bandes passantes modernes. Un deuxième client réseau qui attendrait la disponibilité du même serveur pour transférer 1Ko aurait des raisons de s’impatier !

#### 1.1 Serveurs itératif et concourant

Un serveur itératif (“ iterative server ”) désigne une implémentation qui traite une seule requête à la fois.

Un serveur concourant (“ concurrent server ”) désigne une implémentation capable de gérer plusieurs tâches en apparence simultanées. Attention, cette fonctionnalité n’implique pas nécessairement que ces tâches concourantes doivent toutes s’exécuter en parallèle. . .

Dans cette première approche purement algorithmique nous n’abordons pas la mise en œuvre technique, le paragraphe 2 s’y consacrera !

D’un point de vue conceptuel, les serveurs itératifs sont plus faciles à concevoir et à programmer que les serveurs concourants, mais le résultat n’est pas toujours satisfaisant pour les clients. Au contraire, les serveurs concourants, s’ils sont d’une conception plus savante, sont d’un usage plus agréable pour les utilisateurs parceque naturellement plus disponibles.

## 1.2 Le choix d’un protocole

La pile ARPA nous donne le choix entre TCP et UDP. L’alternative n’est pas triviale. Le protocole d’application peut être complètement bouleversé par le choix de l’un ou de l’autre. Avant toute chose il faut se souvenir des caractéristiques les plus marquantes de l’un et de l’autre.

### 1.2.1 Mode connecté

Le mode connecté avec TCP est le plus facile à programmer, de plus il assure que les données sont transmises, sans perte.

Par contre, Il établit un circuit virtuel bi-directionnel dédié à chaque client ce qui monopolise une socket, donc un descripteur, et interdit par construction toute possibilité de “ broadcast ”.

L’établissement d’une connexion et sa terminaison entraîne l’échange de 7 paquets. S’il n’y a que quelques octets à échanger entre le client et le serveur, cet échange est un gaspillage des ressources du réseau.

Il y a plus préoccupant. Si la connexion est au repos , c’est à dire qu’il n’y a plus d’échange entre le client et le serveur, rien n’indique à celui-ci que le client est toujours là ! TCP est silencieux si les deux parties n’ont rien à s’échanger<sup>1</sup>.

Si l’application cliente a été interrompue accidentellement<sup>2</sup>, rien n’indique au serveur que cette connexion est terminée et il maintient la socket et les buffers associés. Que cette opération se répète un grand nombre de fois et le serveur ne répondra plus, faute de descripteur disponible, voire de mémoire libre au niveau de la couche de transport (allocation au niveau du noyau, en fonction de la mémoire totale et au démarrage de la machine) !

---

<sup>1</sup>Nous verrons au chapitre suivant comment on peut modifier ce comportement par défaut

<sup>2</sup> crash du système, retrait du réseau, . . .

### 1.2.2 Mode datagramme

Le mode datagramme ou “ non connecté ” avec UDP hérite de tous les désagréments de IP, à savoir perte, duplication et désordre introduit dans l’ordre des datagrammes.

Pourtant malgré ces inconvénients UDP reste un protocole qui offre des avantages par rapport à TCP. Avec un seul descripteur de socket un serveur peut traiter un nombre quelconque de clients sans perte de ressources due à de mauvaises déconnexions. Le “ broadcast ” et le “ multicast ” sont possibles.

Par contre les problèmes de fiabilité du transport doivent être gérés au niveau de l’application. Généralement c’est la partie cliente qui est en charge de la réémission de la requête si aucune réponse du serveur ne lui parvient. La valeur du temps au delà duquel l’application considère qu’il doit y avoir réémission est évidemment délicate à établir. Elle ne doit pas être figée aux caractéristiques d’un réseau local particulier et doit être capable de s’adapter aux conditions changeantes d’un internet.

## 1.3 Quatre modèles de serveurs

Deux comportements de serveurs et deux protocoles de transport combinés induisent quatre modèles de serveurs :

|                           |                        |
|---------------------------|------------------------|
| Itératif<br>Data-gramme   | Itératif<br>Connecté   |
| Concourant<br>Data-gramme | Concourant<br>Connecté |

*figure XIV.01*

La terminologie “ **tâche esclave** ” employée dans les algorithmes qui suivent se veut neutre quant au choix technologique retenu pour les implémenter. Ce qui importe c’est leur nature concourante avec la “ **tâche maître** ” qui les pilote.

**Algorithme itératif - Mode data-gramme :**

1. Créer une socket, lui attribuer un port connu des clients.
2. Répéter :
  - Lire une requête d’un client,
  - Formuler la réponse,
  - Envoyer la réponse, conformément au protocole d’application.

**Critique :**

Cette forme de serveur est la plus simple, elle n'est pas pour autant inutile. Elle est adaptée quand il y a un tout petit volume d'information à échanger et en tout cas sans temps de calcul pour l'élaboration de la réponse. Le serveur de date " daytime " ou le serveur de temps " time " en sont d'excellents exemples.

**Algorithme Itératif - Mode connecté :**

1. Créer une socket, lui attribuer un port connu des clients.
2. Mettre la socket à l'écoute du réseau, en mode passif.
3. Accepter la connexion entrante, obtenir une socket pour la traiter.
4. Entamer le dialogue avec le client, conformément au protocole de l'application.
5. Quand le dialogue est terminé, fermer la connexion et aller en 3).

**Critique :**

Ce type de serveur est peu utilisé. Son usage pourrait être dédié à des relations clients/serveurs mettant en jeu de petits volumes d'informations avec la nécessité d'en assurer à coup sûr le transport. Le temps d'élaboration de la réponse doit rester court.

Le temps d'établissement de la connexion n'est pas négligeable par rapport au temps de réponse du serveur, ce qui le rend peu attractif.

**Algorithme concourant - Mode datagramme :****Maître :**

1. Créer une socket, lui attribuer un port connu des clients.
2. Répéter :
  - Lire une requête d'un client
  - Créer une tâche esclave pour élaborer la réponse.

**Esclave :**

1. Recevoir la demande du client,
2. Élaborer la réponse,
3. Envoyer la réponse au client, conformément au protocole de l'application,
4. Terminer la tâche.

**Critique :**

Si le temps d'élaboration de la réponse est rendu indifférent pour cause de création de processus esclave, par contre le coût de création de ce processus fils est prohibitif par rapport à son usage : formuler une seule réponse et l'envoyer. Cet inconvénient l'emporte généralement sur l'avantage apporté par le " parallélisme " .

Néanmoins, dans le cas d'un temps d'élaboration de la réponse long par rapport au temps de création du processus esclave, cette solution se justifie.

**Algorithme concourant - Mode connecté :****Maître :**

1. Créer une socket, lui attribuer un port connu des clients.
2. Mettre la socket à l'écoute du réseau, en mode passif.
3. Répéter :
  - Accepter la connexion entrante, obtenir une `socket` pour la traiter,
  - Créer une tâche esclave pour traiter la réponse.

**Esclave :**

1. Recevoir la demande du client,
2. Amorcer le dialogue avec le client, conformément au protocole de l'application,
3. Terminer la connexion et la tâche.

**Critique :**

C'est le type le plus général de serveur parce-qu'il offre les meilleurs caractéristiques de transport et de souplesse d'utilisation pour le client. Il est sur-dimensionné pour les " petits " services et sa programmation soignée n'est pas toujours à la portée du programmeur débutant.



## 2 Technologie élémentaire

De la partie algorithmique découlent des questions techniques sur le “comment le faire”. Ce paragraphe donne quelques grandes indications très élémentaires que le lecteur soucieux d’acquérir une vraie compétence devra compléter par les lectures indiquées au dernier paragraphe ; la Bibliographie du chapitre (page 325). Notamment il est nécessaire de consulter les ouvrages de **W. R. Stevens** pour la partie système et **David R. Butenhof** pour la programmation des threads.

La suite du texte va se consacrer à éclairer les points suivants :

1. Gestion des “ tâches esclaves ” (paragraphe 2.1, 2.2, 2.3, 2.4)
2. Gestion de descripteurs multiples (paragraphe 2.5, 2.6)
3. Fonctionnement des processus en arrière plan ou “ daemon ” (paragraphe 3)

### 2.1 Gestion des “ tâches esclaves ”

La gestion des “ tâches esclaves ” signalées dans le paragraphe 1 induit que le programme “ serveur ” est capable de gérer plusieurs actions concourantes, c’est à dire qui ont un comportement qui donne l’illusion à l’utilisateur que sa requête est traitée dans un délai raisonnable, sans devoir patienter jusqu’à l’achèvement de la requête précédente.

C’est typiquement le comportement d’un système d’exploitation qui ordonnance des processus entre-eux pour donner à chacun d’eux un peu de la puissance de calcul disponible (“ time-sharing ”).

La démarche qui paraît la plus naturelle pour implémenter ces “ tâches esclaves ” est donc de tirer partie des propriétés mêmes de la gestion des processus du système d’exploitation.

Sur un système Unix l’usage de processus est une bonne solution dans un premier choix car ce système dispose de primitives (APIs) bien rodées pour les gérer, en particulier `fork()`, `vfork()` et `rfork()`.

Néanmoins, comme le paragraphe suivant le rappelle, l’usage de processus fils n’est pas la panacée car cette solution comporte des désagréments. Deux autres voies existent, non toujours valables partout et dans tous les cas de figure. La première passe par l’usage de processus légers ou “ threads ” (paragraphe 2.3), la deuxième par l’usage du signal `SIGIO` qui autorise ce que l’on nomme la programmation asynchrone (paragraphe 2.4).

Pour conclure il faut préciser que des tâches esclaves ou concourantes peuvent s’exécuter dans un ordre aléatoire mais pas nécessairement en même temps. Cette dernière caractéristique est celle des tâches parallèles. Autrement dit, les tâches parallèles sont toutes concourantes mais l’inverse n’est pas vrai. Concrètement il faut disposer d’une machine avec plusieurs processeurs pour avoir, par exemple, des processus (ou des “ threads kernel ”, si elles sont supportées) qui s’exécutent vraiment de manière simultanée donc sur

des processeurs différents. Sur une architecture mono-processeur, les tâches ne peuvent être que concourantes !

## 2.2 fork, vfork et rfork

Il ne s'agit pas ici de faire un rappel sur la primitive `fork()` examinée dans le cadre du cours sur les primitives Unix, mais d'examiner l'incidence de ses propriétés sur l'architecture des serveurs.

Le résultat du `fork()` est la création d'un processus fils qui ne diffère de son père que par les points suivants :

1. Le code de retour de `fork` : 0 pour le fils, le pid du fils pour le père
2. Le numéro de processus (pid) ainsi que le numéro de processus du processus père (ppid)
3. Les compteurs de temps (`utime`, `stime`, ...) qui sont remis à zéro
4. Les verrous (`flock`) qui ne sont pas transmis
5. Les signaux en attente non transmis également

Tout le reste est doublonné, notamment la “ stack ” et surtout la “ heap ” qui peuvent être très volumineuses et donc rendre cette opération pénalisante voire quasi rédhibitoire sur un serveur très chargé (des milliers de processus et de connexions réseaux).

Si le but du `fork` dans le processus fils est d'effectuer un `exec` immédiatement, alors il est très intéressant d'utiliser plutôt le `vmfork`. Celui-ci ne fait que créer un processus fils sans copier les données. En conséquence, durant le temps de son exécution avant le `exec` le fils partage strictement les mêmes données que le père (à utiliser avec précaution). Jusqu'à ce que le processus rencontre un `exit` ou un `exec`, le processus père reste bloqué (le `vmfork` ne retourne pas).

En allant plus loin dans la direction prise par `vmfork`, le `rfork`<sup>3</sup> autorise la continuation du processus père après le `fork`, la conséquence est que deux processus partagent le même espace d'adressage simultanément. L'argument d'appel du `rfork` permet de paramétrer ce qui est effectivement partagé ou non. `RFMEM`, le principal d'entre eux, indique au noyau que les deux processus partagent tout l'espace d'adressage.

Si cette dernière primitive est très riche de potentialités<sup>4</sup>, elle est également délicate à manipuler : deux (ou plus) entités logicielles exécutant le même code et accédant aux mêmes données sans précaution particulière vont très certainement converger vers de sérieux ennuis de fonctionnement si le déroulement de leurs opérations n'est pas rigoureusement balisé.

En effet, le souci principal de ce type de programme multi-entités est de veiller à ce qu'aucune de ses composantes ne puisse changer les états de

---

<sup>3</sup>`clone()` sous Linux

<sup>4</sup> d'ailleurs l'implémentation actuelle des **threads** sous Linux emploie cette primitive avec des avantages et beaucoup d'inconvénients par rapport à ce que prévoit la norme Posix et à la gestion des processus

sa mémoire simultanément. Autrement dit, il faut introduire presque obligatoirement un mécanisme de sémaphore qui permette à l’une des entités logicielles de verrouiller l’accès à telle ou telle ressource mémoire pendant le temps nécessaire à son usage.

Cette opération de “ verrouillage ” elle-même pose problème, parceque les entités logicielles peuvent s’exécuter en parallèle (architecture multi-processeurs) et donc il est indispensable que l’acquisition du sémaphore qui protège une ressource commune soit une opération atomique, c’est à dire qui s’exécute en une fois, sans qu’il y ait possibilité que deux (ou plus) entités logicielles tentent avec succès de l’acquérir. C’est toute la problématique des **mutex**<sup>5</sup>.

## 2.3 Processus légers, les “ threads ”

Les processus légers ou “**threads**” sont une idée du milieu des années 80. La norme Posix a posé les bases de leur développement durable en 1995 (Posix 1.c), on parle dans ce cas des **pthread**s.

L’idée fondatrice des threads est de ne pas faire de **fork** mais plutôt de permettre le partage de l’espace d’adressage à autant de contextes d’exécution du même code<sup>6</sup> que l’on souhaite.

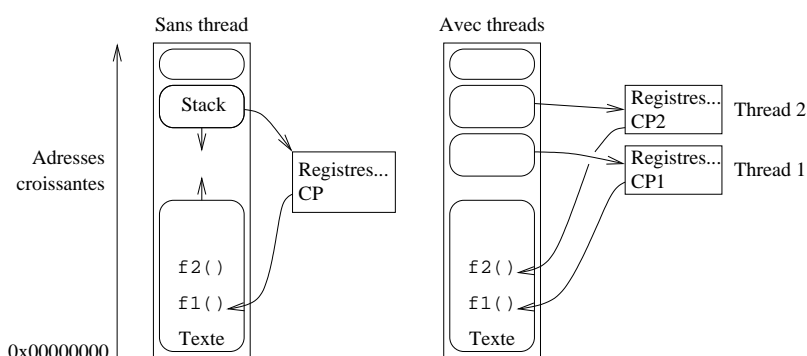


figure XIV.02

Au lieu de créer un nouveau processus on crée une nouvelle thread, ce qui revient (en gros) à ajouter un nouveau contexte d’exécution sur la pile système dans le processus. L’usage de **mutex** (cf paragraphe 2.2) est fortement recommandé pour sérialiser les accès aux “ sections critiques ” du code.

Sur une machine ayant une architecture mono-processeur, le premier type de threads est suffisant, mais dès que la machine est construite avec une architecture smp<sup>7</sup> ou cmt<sup>8</sup> (ce qui est de plus en plus le cas avec la banalisation

<sup>5</sup> “ mutual exclusion ”

<sup>6</sup> ordre de grandeur de quelques dizaines

<sup>7</sup> “ Symmetric Multi Processor ”

<sup>8</sup> “ Chip Multithreading ” - [http://developers.sun.com/solaris/articles/app\\_perf\\_cmt.html](http://developers.sun.com/solaris/articles/app_perf_cmt.html)

des configurations à plusieurs processeurs chacun étant lui-même composé de plusieurs cœurs) l'usage de threads gérables par le noyau devient beaucoup plus intéressant car il utilise au mieux les ressources de la machine : un même processus pourrait avoir deux threads, une s'exécutant sur chacun des deux processeurs (ou plus bien entendu, s'il y a plus de processeurs).

Le principe étant posé, on distingue plusieurs familles d'implémentation.

D'un coté il y a les threads “ **user land** ” c'est à dire qui sont complètement gérées par le processus utilisateur et de l'autre les threads “ **kernel** ”, qui sont gérées par le noyau. Ces dernières threads sont supportées par les constructeurs de machines à architectures parallèles, traditionnellement Sun (Solaris), Ibm (Aix), et Compaq (ex Digital, avec True64) et plus récemment Hewlett-Packard avec la version 11.xx d'HP-UX. Le problème est très complexe et chaque constructeur développe ses propres stratégies.

Du coté des OS libres le problème a stagné un peu pendant des années car il monopolise beaucoup de programmeurs de haut niveau, non toujours disponibles pour des tâches au long court... Néanmoins la famille des BSD (FreeBSD et NetBSD principalement) bénéficie depuis peu d'une gestion opérationnelle des threads.

Les threads Linux utilisent `rfork` qui est simple et très efficace. Cette approche n'est pas satisfaisante car chaque thread est exécutée dans un processus différent (pid différent donc) ce qui est contraire aux recommandations POSIX, d'une part, et d'autre part ne permet pas d'utiliser les règles de priorité définies également par POSIX. Une application avec un grand nombre de threads prend l'avantage sur les autres applications par le fait qu'elle consomme en temps cumulé bien plus que les autres processus mono-thread.

Les threads de FreeBSD sont devenues très efficaces et performantes depuis la version 7 du système, à l'issue d'un travail de longue haleine dont l'historique se trouve sur cette page <http://www.freebsd.org/smp/>.

### Conclusion :

Les threads **user land** ne s'exécutent que sur un seul processeur quelle que soit l'architecture de la machine qui les supporte. Sur une machine de type smp/cmt il faut que le système d'exploitation supporte les threads **kernel** pour qu'un même processus puisse avoir des sous-tâches sur tous les processeurs existants.

## 2.4 Programmation asynchrone

Les paragraphes qui précèdent utilisent un processus ou une thread pour pouvoir effectuer au moins deux tâches simultanément : écouter le réseau et traiter une (ou plusieurs) requête(s). Dans le cas d'un serveur peu sollicité il tout à fait envisageable de mettre en œuvre une autre technique appelée “ programmation asynchrone ”.

La programmation asynchrone s'appuie sur l'usage du signal, **SIGIO** (**SIGPOLL** sur système V), ignoré par défaut, qui prévient le processus d'une activité sur un descripteur.

La gestion des entrées/sorties sur le descripteur en question est alors traitée comme une exception, par un “ handler ” de signaux.

Le signal **SIGIO** est ignoré par défaut, il faut demander explicitement au noyau de le recevoir, à l'aide d'un appel à la primitive **fcntl**. Une fois activé, il n'est pas reçu pour les mêmes raisons selon le protocole employé :

### UDP :

- Arrivée d'un paquet pour la socket
- Une erreur

### TCP :

- Une demande de connexion (attente sur un **accept**) qui arrive
- Une déconnexion
- Une demi-déconnexion (**shutdown**)
- Arrivée de données sur une socket
- Fin de l'émission de données (buffer d'émission vide) sur une socket
- Une erreur

Où l'on voit que cette technique, du moins en TCP, ne peut être envisagée pour que pour des serveurs peu sollicités. Un trop grand nombre d'interruptions possibles nuit à l'efficacité du système (changements de contexte). De plus la distinction entre les causes du signal est difficile à faire, donc ce signal en TCP est quasi inexploitable.

### Conclusion :

La dénomination “ programmation asynchrone ” basée seulement sur l'usage du signal **SIGIO** (versus **SIGPOLL**) est abusive. Pour être vraiment asynchrones, ces opérations de lecture et d'écriture ne devraient pas être assujetties au retour des primitives **read** ou **write**<sup>9</sup>. Cette technique permet l'écriture du code de petits serveurs basé sur le protocole UDP (En TCP les causes de réception d'un tel signal sont trop nombreuses) sans **fork** ni **thread**.

---

<sup>9</sup>La norme POSIX permet un tel comportement avec les primitives **aio\_read** et **aio\_write**

## 2.5 La primitive select

Un serveur qui a la charge de gérer simultanément plusieurs sockets (serveur multi-protocoles par exemple, comme `inetd...`) se trouve par construction dans une situation où il doit examiner en même temps plusieurs descripteurs (il pourrait s'agir aussi de tubes de communication).

Il est absolument déconseillé dans cette situation de faire du `polling`. Cette activité consisterait à examiner chaque descripteur l'un après l'autre dans une boucle infinie qui devrait être la plus rapide possible pour être la plus réactive possible face aux requêtes entrantes. Sous Unix cette opération entraîne une consommation exagérée des ressources cpu, au détriment des autres usagers et services.

La primitive `select` (4.3 BSD) surveille un ensemble de descripteurs, si aucun n'est actif le processus est endormi et ne consomme aucune ressource cpu. Dès que l'un des descripteurs devient actif (il peut y en avoir plusieurs à la fois) le noyau réveille le processus et l'appel de `select` rend la main à la procédure appelante avec suffisamment d'information pour que celle-ci puisse identifier quel(s) descripteur(s) justifie(nt) son réveil !

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

int select (int maxfd, fd_set *readfs,
            fd_set *writefs,
            fd_set *exceptfs,
            struct timeval *timeout) ;

FD_ZERO(fd_set *fdset) ;           /* Tous les bits a zero.          */
FD_SET(int fd, fd_set *fdset) ;    /* Positionne 'fd' dans 'fdset' */
FD_CLR(int fd, fd_set *fdset) ;    /* Retire 'fd' de 'fdset'       */
FD_ISSET(int fd, fd_set *fdset) ;  /* Teste la presence de 'fd'    */

struct timeval /* Cf "time.h" */
{
    long tv_sec ;           /* Nombre de secondes.          */
    long tv_usec ;         /* Nombre de micro-secondes.    */
} ;
```

Le type `fd_set` est décrit dans `<sys/types.h>`, ainsi que les macros `FD_XXX`.

Le prototype de `select` est dans `<sys/time.h>`.

La primitive `select` examine les masques `readfs`, `writefs` et `exceptfs` et se comporte en fonction de `timeout` :

- Si `timeout` est une structure existante (pointeur non nul), la primitive retourne immédiatement après avoir testé les descripteurs. Tous les champs de `timeout` doivent être à 0 ( " polling " dans ce cas).
- Si `timeout` est une structure existante (pointeur non nul), et si ses champs sont non nuls, `select` retourne quand un des descripteurs est

prêt, et en tout cas jamais au delà de la valeur précisée par `timeout` (cf `MAXALARM` dans `<sys/param.h>`).

- Si `timeout` est un pointeur `NULL`, la primitive est bloquante jusqu'à ce qu'un descripteur soit prêt (ou qu'un signal intervienne).

Remarque : `select` travaille au niveau de la micro-seconde, ce que ne fait pas `sleep` (seconde), d'où un usage possible de timer de *précision*.

**readfs** descripteurs à surveiller en lecture.

**writefs** descripteurs à surveiller en écriture.

**exceptfs** Ce champ permet de traiter des événements exceptionnels sur les descripteurs désignés. Par exemple :

- Données *out-of-band* sur une `socket`.
- Contrôle du statut sur un pseudo-tty maître.

**maxfd** prend à l'appel la valeur du plus grand descripteur à tester, plus un. Potentiellement un système BSD (4.3 et versions suivantes) permet d'examiner jusqu'à 256 descripteurs.

A l'appel, le programme précise quels sont les descripteurs à surveiller dans **readfs**, **writefs** et **exceptfs**.

Au retour, la primitive précise quels sont les descripteurs qui sont actifs dans les champs **readfs**, **writefs** et **exceptfs**. Il convient donc de conserver une copie des valeurs avant l'appel si on veut pouvoir les réutiliser ultérieurement. La primitive renvoie -1 en cas d'erreur (à tester systématiquement) ; une cause d'erreur classique est la réception d'un signal (`errno==EINTR`).

La macro `FD_ISSET` est utile au retour pour tester quel descripteur est actif et dans quel ensemble.

Le serveur de serveurs `inetd` (page 4) est un excellent exemple d'utilisation de la primitive.

## 2.6 La primitive poll

La primitive `poll` (System V) permet la même chose que la primitive `select`, mais avec une approche différente.

```
#include <poll.h>
int
poll(struct pollfd *fds, unsigned int nfds, int timeout);

struct pollfd {
    int fd ;          /* Descripteur de fichier */
    short events ;    /* Evenements attendus */
    short revents ;   /* Evenements observes */
} ;
```

La primitive retourne le nombre de descripteurs rendus disponibles pour effectuer des opérations d'entrée/sortie. -1 indique une condition d'erreur. 0 indique l'expiration d'un délai ( " time-out " ).

**fds** est un pointeur sur la base d'un tableau de **nfds** structures du type `struct pollfd`.

Les champs **events** et **revents** sont des masques de bits qui paramètrent respectivement les souhaits du programmeur et ce que le noyau retourne.

On utilise principalement :

**POLLIN**

**POLLOUT**

**POLLERR**

**POLLHUP**

**nfds** Taille du vecteur.

**timeout** Est un compteur de millisecondes qui précise le comportement de `poll` :

- Le nombre de millisecondes est positif strictement. Quand le temps prévu est écoulé, la primitive retourne dans le code de l'utilisateur même si aucun événement n'est intervenu.
- Le nombre de millisecondes est **INFTIM** (-1), la primitive est bloquante.
- 0. La primitive retourne immédiatement.

On s'aperçoit immédiatement que la valeur du paramètre de `timeout` n'est pas compatible ni en forme ni en comportement entre `select` et `poll`.



## 3 Fonctionnement des daemons

Sous Unix les serveurs sont implémentés le plus souvent sous forme de *daemons*<sup>10</sup>. La raison principale est que ce type de processus est le plus adapté à cette forme de service, comme nous allons l'examiner.

### 3.1 Programmation d'un *daemon*

Les *daemons* sont des processus ordinaires, mais :

- ils ne sont pas rattachés à un terminal particulier (ils sont en “ arrière plan ”);
- ils s'exécutent le plus souvent avec les droits du “ super-utilisateur ”, voire, mieux, sous ceux d'un pseudo-utilisateur sans mot de passe ni shell défini.
- ils sont le plus souvent lancés au démarrage du système, lors de l'exécution des shell-scripts de configuration (par exemple à partir de `/etc/rc`);
- ils ne s'arrêtent en principe jamais (sauf bien sûr avec le système!).

La conception d'un *daemon* suit les règles suivantes :

1. Exécuter un `fork`, terminer l'exécution du père et continuer celle du fils qui est alors adopté par `init` (traditionnellement c'est le processus N° 1). Le processus fils est alors détaché du terminal, ce que l'on peut visualiser avec un `ps -auxw` (versus `ps -edalf` sur un système V) en examinant la colonne TT : elle contient ??;
2. Appeler la primitive `setsid` pour que le processus courant devienne “ leader ” de groupe (il peut y avoir un seul processus dans un groupe);
3. Changer de répertoire courant, généralement la racine (/) ou tout autre répertoire à la convenance de l'application;
4. Modifier le masque de création des fichiers `umask = 0` pour que le troisième argument de `open` ne soit pas biaisé par la valeur du `umask` lorsque cette primitive sert aussi à créer des fichiers;
5. Fermer tous les descripteurs devenus inutiles, et en particulier 0, 1 et 2 (entrée et sorties standards n'ont plus de sens pour un processus détaché d'un terminal).

le source ci-après est un exemple de programmation de daemon, les appels à la fonction `syslog` font référence à un autre daemon nommé `syslogd` que nous examinons au paragraphe suivant.

---

<sup>10</sup>Si l'on en croit la première édition de “ UNIX System Administration Handbook ”, Nemeth, Synder & Seebass, pp 403-404 : “Many people equate the word 'daemon' with the word 'demon' implying some kind of Satanic connection between UNIX and the underworld. This is an egregious misunderstanding. 'Daemon' is actually a much older form of 'demon'; daemons have no particular bias towards good or evil, but rather serve to help define a person's character or personality.”

```

1  /* $Id: diable.c 92 2009-02-12 17:39:44Z fla $
2  *
3  * Diablotin : exemple de démon miniature...
4  */
5
6  #include      <stdio.h>
7  #include      <stdlib.h>
8  #include      <unistd.h>
9  #include      <sys/types.h>
10 #include      <sys/stat.h>
11 #include      <syslog.h>
12 #include      <errno.h>
13
14 int
15 main()
16 {
17     switch (fork()) {
18         case -1 :                /* erreur du "fork".                */
19             perror("fork") ;
20             exit (1) ;
21
22         case 0 :                /* Le futur "demon".                */
23             (void)printf("Je suis infernal, je me transforme en demon !\n") ;
24             (void)setsid() ;    /* Devenir chef de groupe.                */
25             (void)chdir("/") ; /* Repertoire de travail.                */
26             (void)umask(0) ;
27             (void)close(0) ;
28             (void)close(1) ;
29             (void)close(2) ;
30             openlog("diablotin", LOG_PID | LOG_NDELAY, LOG_USER) ;
31             syslog(LOG_INFO, "Attention, je suis un vrai 'daemon'...\n") ;
32             (void)sleep(1) ;
33             (void)syslog(LOG_INFO, "Je me tue !\n") ;
34             closelog() ;
35             exit(EX_OK) ;
36
37         default :
38             exit(EX_OK) ;
39     }
40 }

```

*diable.c*

## 3.2 Daemon syslogd

Du fait de leur fonctionnement détaché d'un terminal, les daemons ne peuvent plus délivrer directement de message par les canaux habituels (`perror...`). Pour pallier à cette déficience un daemon est spécialisé dans l'écoute des autres daemons (écoute passive :), il s'agit de `syslogd`<sup>11</sup>.

Pour dialoguer avec ce daemon un programme doit utiliser les fonctionnalités que le lecteur trouvera très bien décrites dans “`man syslog`”, sinon le paragraphe 3.4 en donne un aperçu rapide.

La *figure XIV.3* suivante schématise le circuit de l'information dans le cas d'une utilisation de `syslogd`.

Le fichier `/etc/syslog.conf` est le fichier standard de configuration du daemon `syslogd`. Il est constitué de lignes de deux champs : un déclencheur

<sup>11</sup>Ce rôle stratégique lui vaut d'être lancé le premier et d'être stoppé le dernier

(*selector*) et une action. Entre ces deux champs un nombre quelconque de tabulations.

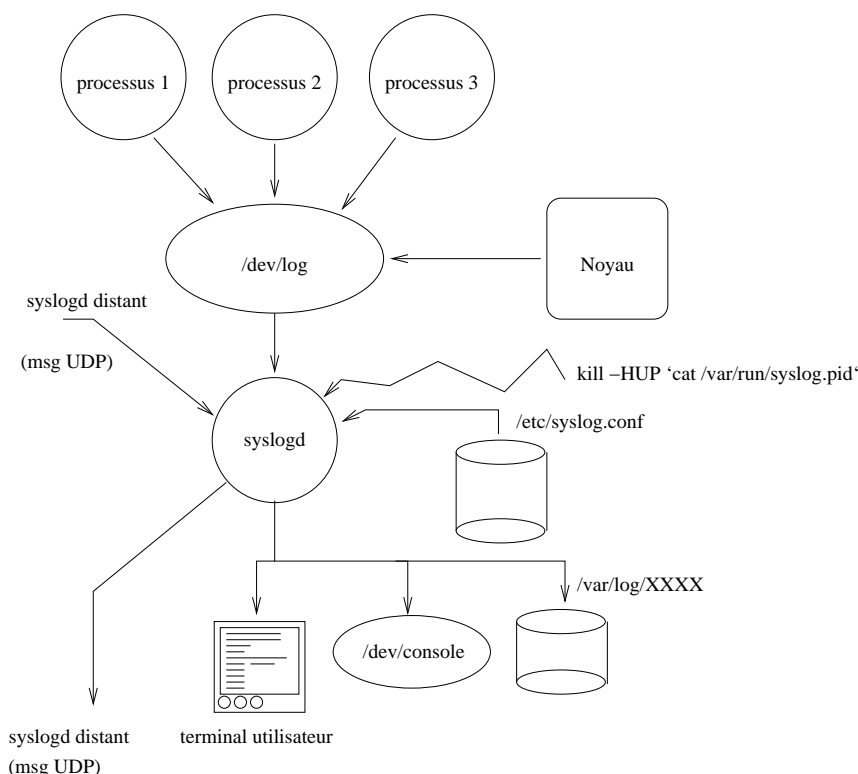


figure XIV.03

Si les conditions du déclencheur sont remplies l'action est exécutée, plus précisément :

**Le déclencheur** est un filtre qui associe un type de daemon avec un niveau de message. Par exemple `mail.debug` signifie les messages de niveau `DEBUG` pour le système de routage du courrier.

Les mots clefs possibles pour le type de daemon sont `auth`, `authpriv`, `cron`, `daemon`, `kern`, `lpr`, `mail`, `news`, `syslog`, `user`, `uucp`, et `local0` à `local17`. Une étoile (\*) à la place, signifie n'importe quel mot clef.

Le niveau de message est l'un des mots clefs suivants : `emerg`, `alert`, `crit`, `err`, `warning`, `notice`, et `debug`. Une étoile (\*) signifie n'importe lequel. Un point (.) sépare les deux parties du filtre, comme dans `mail.debug`.

Dans les syslog plus évolués l'administrateur a la possibilité de dérouter tous les messages contenant un nom de programme (`!nom_du_prog`) ou un nom de machine (`+nom_de_machine`)

**L'action** est soit :

- Un fichier désigné par un chemin absolu, comme `/var/log/syslog`.
- Une liste de logins d'utilisateurs, comme `root,fla...`
- Un nom de machine distante (`@machine.domaine.fr`)
- Tous les utilisateurs connectés avec une étoile \*.

### 3.3 Fichier syslog.conf

Exemple de fichier /etc/syslog.conf :

```
*.err;kern.debug;auth.notice;mail.crit      /dev/console
*.notice;kern.debug;lpr.info;mail.crit      /var/log/messages
mail.info                                    /var/log/maillog
lpr.info                                    /var/log/lpd-errs
cron.*                                       /var/cron/log
*.err                                        root
*.notice;auth.debug                         root
*.alert                                     root
*.emerg                                     *
*.info                                      |/usr/local/bin/traitinfo
!diablotin
*.*                                          /var/log/diablotin.log
```

Résultat de l'exécution de diablotin sur la machine glups, et dans le fichier /var/log/diablotin.log :

```
...
Jan 27 18:52:02 glups diablotin[20254]: Attention, je suis un vrai 'daemon'...
Jan 27 18:52:03 glups diablotin[20254]: Je me tue !
...
```

### 3.4 Fonctions syslog

Les prototypes et arguments des fonctions :

```
#include <syslog.h>

void openlog(const char *ident, int logopt, int facility) ;
void syslog(int priority, const char *message, ...) ;
void closelog(void) ;
```

Comme dans l'exemple de “ diablotin ”, un programme commence par déclarer son intention d'utiliser le système de log en faisant appel à la fonction `openlog` :

**logopt** Donne la possibilité de préciser où le message est envoyés et dans quelle condition.

**facility** Est l'étiquette par défaut des futurs messages envoyés par `syslog`.

| logopt | description |
|--------|-------------|
|--------|-------------|

|            |   |
|------------|---|
| LOG_CONS   | Écriture sur /dev/console.                                      |
| LOG_NDELAY | Ouverture immédiate de la connexion avec <code>syslogd</code> . |
| LOG_PERROR | Écriture d'un double du message sur <code>stderr</code> .       |
| LOG_PID    | Identifier chaque message avec le <code>pid</code> .            |

| facility     | description   |
|--------------|---|
| LOG_AUTH     | Services d'authentification.                            |
| LOG_AUTHPRIV | Idem ci-dessus.   |
| LOG_CRON     | Le daemon qui gère les procédures <code>batch</code> .  |
| LOG_DAEMON   | Tous les daemons du système, comme <code>gated</code> . |
| LOG_KERN     | Messages du noyau.                                      |
| LOG_LPR      | Messages du gestionnaire d'imprimante.                  |
| LOG_MAIL     | Messages du gestionnaire de courrier.                   |
| LOG_NEWS     | Messages du gestionnaire de " news ".                   |
| LOG_SYSLOG   | Messages du daemon <code>syslogd</code> lui-même.       |
| LOG_USER     | Messages des processus utilisateur (default).           |
| LOG_UUCP     | Messages du système de transfert de fichiers.           |
| LOG_LOCAL0   | Réservé pour un usage local.                            |

Puis chaque appel à la fonction `syslog` est composé d'un message (généré par l'application) et d'un code de priorité, composé d'un niveau d'urgence précisé par le tableau ci-dessous (niveaux décroissants) et d'une étiquette optionnelle, prise dans le tableau ci-dessus ; elle prime alors sur celle précisée lors du `openlog`.

| priority    | description                           |
|-------------|---------------------------------------|
| LOG_EMERG   | Une condition de " panic system ".    |
| LOG_ALERT   | Intervention immédiate requise.       |
| LOG_CRIT    | Problèmes de matériels                |
| LOG_ERR     | Erreurs.                              |
| LOG_WARNING | Messages d'avertissement.             |
| LOG_NOTICE  | Messages qui ne sont pas des erreurs. |
| LOG_INFO    | Informations sans conséquence.        |
| LOG_DEBUG   | Messages pour le debug.               |

Enfin le `closelog` matérialise la fin d'utilisation de ce système dans le code.

## 4 Exemple de “ daemon ” inetd

Dans cette partie nous allons étudier un serveur de serveurs nommé `inetd` qui est un très bel exemple pour conclure ce chapitre.

Ce chapitre pourra se prolonger par la lecture du code source C d'`inetd`.

### 4.1 Présentation de inetd

Sous Unix on peut imaginer facilement que chacun des services réseaux offerts soient programmés comme un daemon, avec une ou plusieurs sockets, chacun surveillant son ou ses ports de communication.

Un tel fonctionnement existe, généralement repéré par le vocabulaire “stand alone”. Avec cette stratégie, chaque service comme “ftp”, “rlogin”, ou encore “telnet” fait l’objet d’un processus daemon (“daemon”).

Avant la version 4.3 de BSD, c’est comme cela que tous les services fonctionnaient. Le problème est que pour faire fonctionner les services de base du réseau on devait maintenir en mémoire (primaire en “ram” ou secondaire sur la zone de “swap”) un grand nombre de processus souvent complètement inutiles à un instant donné, simplement au cas où...

L’inconvénient de cette stratégie est la consommation importante de ressources surtout avec le nombre croissant des services réseaux “de base”. De plus, on peut remarquer que lancés au démarrage de la machine, tous ces processus effectuent des opérations similaires (cf 3), seuls diffèrent les traitements propres aux serveurs eux-mêmes c’est à dire ceux qui relèvent du protocole de l’application.

La version 4.3 de BSD a apporté une simplification en introduisant une nouvelle notion, celle de serveur de serveurs : “The Internet superserver — `inetd`”. C’est un daemon que peuvent utiliser tous les serveurs TCP/UDP.

`Inetd` fournit essentiellement deux services principaux :

1. Il permet à un seul processus (celui d'`inetd`) d’attendre de multiples demandes de connexions au lieu d’avoir 1 processus par type de connexion. Cette stratégie réduit d’autant le nombre de processus.
2. Il simplifie l’écriture des serveurs eux-mêmes, puisqu’il gère toute la prise en charge de la connexion. Les serveurs lisent les requêtes sur leur entrée standard et écrivent la réponse sur leur sortie standard.

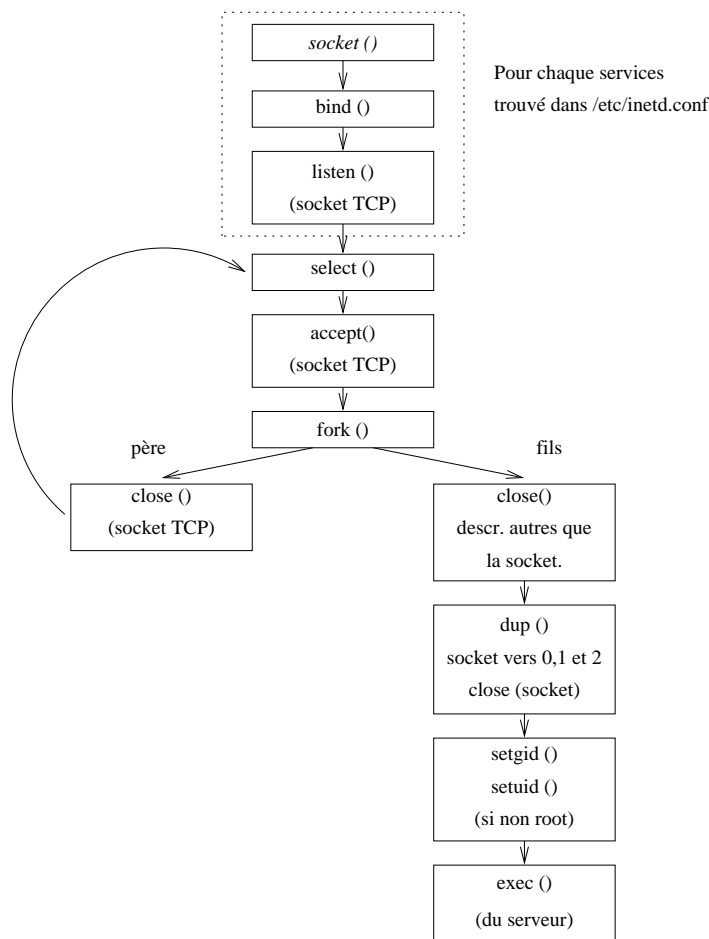
`Inetd` est un serveur parallèle en mode connecté ou data-gramme. De plus il combine des caractéristiques particulières, puisqu’il est également multi-protocoles et multi-services. Un même service peut y être enregistré et accessible en udp comme en tcp. Bien sûr cela sous entend que le programmeur de ce service ait prévu ce fonctionnement.

Le prix à payer pour une telle souplesse est élevé, `inetd` invoque `fork` puis `exec` pour pratiquement tous les services qu’il offre (cf lecture de code).

Sur les Unix à architecture Berkeley, `inetd` est invoqué au démarrage de la machine, dans les scripts de lancement, `/etc/rc` par exemple. Dès le début de son exécution il se transforme en daemon (cf paragraphe IV.5.3) et lit un

fichier de configuration généralement nommé `/etc/inetd.conf`. Ce fichier est en ASCII, il est lisible normalement par tous, cependant, sur certains sites et pour des raisons de sécurité, il peut ne pas l'être.

La *figure XIV.04* montre l'architecture générale (très simplifiée) de fonctionnement.



*figure XIV.04*

Le fichier `/etc/inetd.conf` est organisé de la manière suivante :

- Un `#` en début de ligne indique un commentaire, comme pour un shell-script.
- Les lignes vides ne sont pas prises en compte.
- Les lignes bien formées sont constituées de 7 champs. Chaque ligne bien formée décrit un serveur.

Description des champs :

1. Le nom du service, qui doit également se trouver dans le fichier `/etc/services`. C'est grâce à lui que `inetd` connaît le numéro de port à employer
2. Le type de `socket`, connectée (`stream`) ou non (`dgram`).

3. Le protocole qui doit être `tcp` ou `udp` et doit en tout cas se trouver dans le fichier `/etc/protocols`. Ce dernier fichier donne une correspondance numérique aux différents protocoles.
4. `wait` ou `nowait` suivant que le serveur est itératif ou parallèle.
5. Le nom du propriétaire (pour les droits à l'exécution). Le plus souvent c'est `root`, mais ce n'est pas une règle générale.
6. Le chemin absolu pour désigner l'exécutable du serveur.
7. Les arguments transmis à cet exécutable lors du `exec`, il y en a 20 au maximum dans les implémentations Berkeley de `inetd` (certaines re-écritures, comme celle d'HP, limitent ce nombre).

## 5 Exemple de code serveur

L'exemple qui suit est le code en langage C d'un serveur d'écho multi protocoles, c'est à dire qui fonctionne avec TCP et UDP simultanément sur un même numéro de port pour les deux protocoles. La contrainte est que l'usage du serveur pour l'un des protocoles n'empêche pas l'accès au serveur pour l'autre protocole.

Ce serveur offre également le choix de travailler en mode itératif ou en mode parallèle. Cette alternative est pilotée à partir de la ligne de commande, donc au lancement du serveur (option `-n` ou `-w`).

Il est intéressant de remarquer que le cœur du serveur est construit autour de l'usage de la primitive `select` pour gérer l'écoute sur des sockets multiples, ici au nombre de deux.

D'un point de vue plus général ce serveur reprend l'architecture globale du serveur de serveur `inetd` mais le simplifiant à l'extrême, c'est à dire sans gestion du fichier de configuration, et sans gestion des limites.

### 5.1 Guide de lecture du source `serv2prot.c`

Le source de cet exemple se trouve à l'Annexe A, page 367. Le programme `'serv2prot'` le lance avec les options suivantes :

- p** numéro du port
- n** mode concourant
- w** mode itératif

La fonction `'main'` (ligne 64 à 178) contient la structure principale du programme.

**Ligne 77** Boucle de lecture des arguments de la ligne de commande. L'option `-p` a besoin d'un argument (le `#` de port) dont la lecture est effectuée ligne 80 (usage de la fonction `atoi` pour transformer la chaîne de caractères en entier).



**Ligne 102** Ouverture d'une socket UDP utilisant le port `nport` lu sur la ligne de commande

**Ligne 103** Même chose que ligne 102 mais avec une socket TCP.

**Ligne 104** C'est le majorant de `sudp` et `stcp` (pour `select`).

**Ligne 106** Mise à zéro de tous les bits de la variable `lect` (`fd_set`)

**Ligne 107** Ajout du descripteur `udp`

**Ligne 108** Ajout du descripteur `tcp`

**Ligne 110** Mise en place de la prise en compte des signaux de type `SIGCHLD`. C'est la fonction `PasDeZombi` qui est appelée.

**Ligne 111** Mise en place de la prise en compte du signal de fin, ici un `SIGHUP`. Appel de la fonction `FinCanonique` dans ce cas.

**Ligne 113** Entrée de la boucle principale et infinie du serveur

**Ligne 114** Recopie dans `alire` des descripteurs à surveiller en lecture

**Ligne 116** Appel de la primitive `select`, sans time-out, donc bloquante indéfiniment (cad jusqu'à l'arrivée d'une demande de `cnx`)

**Ligne 118** Si on arrive à cette ligne c'est qu'un signal a interrompu la primitive. Le résultat du test est `VRAI` si la primitive a été interrompu par un signal (par exemple `SIGCHLD`), le `'continue'` permet de retourner à l'évaluation de la condition de sortie de boucle immédiatement. Sinon il s'agit d'une erreur non contournable, affichage d'un message et sortie.

**Ligne 124** `select` a renvoyé le nombre de descripteurs qui justifient son retour en "user land". Ce nombre est 1 ou 2 au maximum (seulement 2 sockets à surveiller). On boucle jusqu'à épuisement du nombre de descripteurs à examiner.

**Ligne 125** `FD_ISSET` permet de tester si la socket `stcp` est active. Si oui alors on passe à la ligne 127...

**Ligne 127** Appel de `accept` pour la socket `tcp`. Il faut noter qu'on ne tient pas compte de l'adresse du client réseau (deuxième et troisième argument). `sock` contient le descripteur de la socket vers le client.

**Ligne 133** Idem que ligne 125 mais pour la socket UDP.

**Ligne 138** Usage de la primitive `getpeername` pour obtenir l'adresse de la socket du client (adresse IP + numéro de port).

**Ligne 142** Usage des fonctions `inet_ntoa` et `ntohs` pour afficher l'adresse IP et le port du client qui se connecte.

**Ligne 144** Il s'agit d'une étiquette, point d'entrée du `goto` qui se situe ligne 148.

**Ligne 145** On tente de lancer le service demandé, à exécuter dans un processus fils.

**Ligne 147** En cas d'erreur, si le `fork` a été interrompu par un signal, par exemple `eaSIGCHLD`, on effectue un saut inconditionnel à l'étiquette `retry` signalée ligne 144. Sinon c'est une vraie erreur à traiter !

**Ligne 151** Il s'agit du code exécuté dans le processus fils. `intcp==VRAI` s'il s'agit de la socket TCP. Fermeture des sockets devenues inutiles (c'est `sock` qui est utile).

**Ligne 155** Invocation la fonction qui gère l'écho en TCP

**Ligne 158** Fermeture de la socket TCP inutile. La socket UDP est indispensable.

**Ligne 159** Invocation de la fonction qui gère l'écho en UDP

**Ligne 161** Sortie du code pour les processus fils

**Ligne 162** Il s'agit du code exécuté dans le processus père. Si le mode de fonctionnement est itératif la socket en question (TCP vs UDP) doit être retirée des descripteurs à surveiller. Elle y sera remise lorsque le processus fils qui traite la session en cours sera terminé (cf fonction `PasDeZombi` ligne 184).

**Ligne 165** Si on vient de traiter la socket TCP on fait le ménage avant la prochaine boucle : fermeture de `sock` devenu inutile, retrait de `stcp` de `alire` et conservation d'une trace du pid.

**Ligne 175** on décrémente le nombre de descripteurs à examiner.

**Ligne 177** Fin de la boucle principale commencée ligne 124.

**Ligne 171** Conservation du pid du fils UDP et suppression de `sudp` de `alire`.

La fonction `PasDeZombi` est le `handler` pour les signaux de type `SIGCHLD`, envoyés par le noyau au processus père dès que l'un de ses fils fait `exit`.

**Ligne 194** Usage de la primitive `wait3` qui permet de faire une attente non bloquante (c'est justifié dans la mesure où on a reçu un `SIGCHLD`) de la mort d'un fils. Chaque appel renvoie le pid d'un processus fils mort, s'il n'y a plus de processus fils mort à examiner le retour est négatif. C'est la condition de sortie de boucle.

**Ligne 195** Si on entre dans ce test, la variable `pid` contient le pid du fils terminé et le mode de fonctionnement est itératif.

**Ligne 197** Pour la socket TCP on remet `stcp` dans les descripteurs à surveiller

**Ligne 202** Pour la socket UDP on remet `sudp` dans les descripteurs à surveiller

**Ligne 207** Certains OS ont besoin que l'on repositionne le handler de signaux à chaque réception du signal. Ce n'est pas le cas des BSD.

**Ligne 215** `FinCanonique` est appelée sur réception du signal de fin `SIGHUP`. C'est la sortie inconditionnelle du programme.

Les fonctions `OuvrirSocketUDP` et `OuvrirSocketTCP` sont une reformulation de ce qui a déjà été examiné précédemment.

Les fonctions `TraiterTCP` et `TraiterUDP` ne présentent pas de difficulté de lecture.

## 6 Bibliographie

### **Pour la partie architecture/configuration des serveurs :**

- W. Richard Stevens — “ Unix Network Programming ” — Prentice All — 1990
- W. Richard Stevens — “ Unix Network Programming ” — Volume 1 & 2 — Second edition — Prentice All — 1998
- W. Richard Stevens — “ Advanced Programming in the UNIX Environment ” — Addison-Wesley — 1992
- W. Richard Stevens – Bill Fenner – Andrew M. Rudoff — “ Unix Network Programming ” — Third edition, volume 1 — Prentice All — 2004
- Douglas E. Comer – David L. Stevens — “ Internetworking with TCP/IP – Volume III ” (BSD Socket version) — Prentice All — 1993
- Stephen A. Rago — “ Unix System V Network Programming ” — Addison-Wesley — 1993
- Man `Unix` de `inetd`, `syslog`, `syslogd` et `syslog.conf`.

### **Pour la programmation des threads :**

- David R. Butenhof — “ Programming with POSIX Threads ” — Addison-Wesley — 1997
- Bradford Nichols, Dirsk Buttlar & Jacqueline Proulx Farell — “ Pthreads programming ” – O'Reilly & Associates, Inc. — 1996

### **Et pour aller plus loin dans la compréhension des mécanismes internes :**

- McKusick, Bostik, Karels, Quaterman — “ The Design and implementation of the 4.4 BSD Operating System ” — Addison Wesley — 1996
- Jim Mauro, Richard McDougall — “ Solaris Internals ” — Sun Microsystems Press — 2001
- Uresh Vahalia — “ Unix Internals, the new frontiers ” — Prentice Hall — 1996



# Chapitre XV

## Anatomie d'un serveur Web

### 1 Le protocole HTTP

**ATTENTION CE CHAPITRE N'A PAS FAIT L'OBJET D'UNE REVISION DEPUIS DE NOMBREUSES ANNÉES. LES INFORMATIONS CONTENUES Y SONT JUSTES MAIS PASSABLEMENT OBSOLÈTES!** Ce document est une présentation succincte du protocole HTTP 1.0 et du serveur Apache qui l'utilise.

Le protocole HTTP<sup>1</sup> est le protocole d'application utilisé pour véhiculer, entres autres, de l'HTML<sup>2</sup> sur l'Internet.

C'est le protocole le plus utilisé en volume depuis 1995, devant FTP, NNTP et SMTP ; il accompagne l'explosion de l'utilisation du système global d'information "World-Wide Web".

Depuis 1990, date d'apparition du "Web", le protocole HTTP évolue doucement mais surement. Il est longtemps resté sous forme de "draft". La première version déployée largement a été la 1.0, définie dans la RFC 1945 de mai 1996. Depuis le début du mois de janvier 1997 est apparue la version 1.1, deux fois plus volumineuse pour tenir compte des nouvelles orientations de l'usage du service.

Aujourd'hui ce protocole est tellement répandu que pour bon nombre de nouveaux utilisateurs du réseau, l'Internet c'est le "web" !

Techniquement, l'usage de ce protocole se conçoit comme une relation entre un client et un serveur. Le client, appelé génériquement un "browser", un "User Agent", ou encore *butineur de toile*, interroge un serveur connu par son "url"<sup>3</sup> dont la syntaxe est bien décrite dans la RFC 1738.

Par exemple la chaîne de caractères <http://www.sio.ecp.fr/> est une url ; il suffit de la transmettre en tant qu'argument à un quelconque outil d'exploration et celui-ci vous renverra (si tout se passe comme prévu !) ce qui est prévu sur le serveur en question pour répondre à cette demande (car il s'agit bien d'une requête comme nous le verrons plus loin dans ce chapitre).

---

<sup>1</sup>"Hypertext Transfer Protocol"

<sup>2</sup>"Hypertext Markup Language" — Consulter les "technical reports and publications" du site : <http://www.w3.org/pub/WWW/TR/>

<sup>3</sup>"Uniform Resource Locator"

Le serveur, supposé à l'écoute du réseau au moment où la partie cliente l'interroge, utilise un port connu à l'avance. Le port 80 est dédié officiellement au protocole http<sup>4</sup>, mais ce n'est pas une obligation (cette décision est prise à la configuration du serveur). L'url qui désigne un serveur peut contenir dans sa syntaxe le numéro de port sur lequel il faut l'interroger, comme dans :

<http://www.sio.ecp.fr:11235/>.

## 1.1 Exemple d'échange avec http

Le transport des octets est assuré par TCP<sup>5</sup> et le protocole est "human readable", ce qui nous autorise des essais de connexion avec le client tcp à tout faire : **telnet** ! Bien entendu on pourrait utiliser un "browser" plus classique, mais celui-ci gérant pour nous le détail des échanges il ne serait plus possible de les examiner.

```
$ telnet localhost 80
Trying...
Connected to localhost.
Escape character is '^]'.
GET / HTTP/1.0
```

Ce qui est tapé par l'utilisateur et la réponse du serveur.

La requête, suivie d'une ligne vide.

```
HTTP/1.1 200 OK
Date: Fri, 01 Mar 2002 10:59:06 GMT
Server: Apache/1.3.23 (Unix)
Last-Modified: Sat, 10 Nov 2001 16:13:02 GMT
ETag: "1381-8b-3bed520e"
Accept-Ranges: bytes
Content-Length: 79
Connection: close
Content-Type: text/html
```

Enfin la réponse du serveur, que l'on peut décomposer en trois parties :

1. Un code de retour (HTTP)
2. Un en-tête MIME
3. Des octets, ici ceux d'une page écrite en HTML.

```
<HTML>
<HEAD>
<TITLE>Ceci est un titre</TITLE>
</HEAD>
<BODY>
</BODY>
</HTML>
Connection closed by foreign host.
```

Notons également la déconnexion à l'initiative du serveur, en fin d'envoi de la page HTML.

## 1.2 Structure d'un échange

L'exemple qui précède est typique d'un échange entre le client et le serveur : une question du client génère une réponse du serveur, le tout lors d'une connexion TCP qui se termine lors de l'envoi du dernier octet de la réponse (clôture à l'initiative du serveur).

<sup>4</sup><http://www.iana.org/assignments/port-numbers>

<sup>5</sup>page 89

Le serveur ne conserve pas la mémoire des échanges passés, on dit aussi qu'il est sans état, ou "stateless".

La question et la réponse sont bâties sur un modèle voisin : le message HTTP.

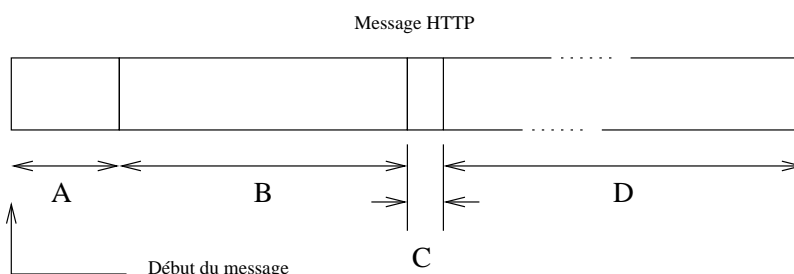


figure XV.01

Les parties A, B et C forment l'en-tête du message, et D le corps.

- A** La première ligne du message, est soit la question posée ("request-line"), soit le statut de la réponse ("status-line").
- o La question est une ligne terminée par CRLF, elle se compose de trois champs :

**Une méthode** à prendre dans GET, HEAD, ou POST.

**GET** Plus de 99% des requêtes ont cette méthode, elle retourne l'information demandée dans l'URL (ci-dessous).

**HEAD** La même chose que GET, mais seul l'en-tête du serveur est envoyé. Utile pour faire des tests d'accessibilité sans surcharger la bande passante. Utile également pour vérifier de la date de fraîcheur d'un document (information contenue dans l'en-tête).

**POST** Cette méthode permet d'envoyer de l'information au serveur, c'est typiquement le contenu d'un formulaire rempli par l'utilisateur.

**Une ressource** que l'on désigne par une URL<sup>6</sup>.

Par exemple <http://www.site.org/>.

**La version du protocole**, sous forme HTTP-Numéro de version.

Par exemple HTTP/1.1 !

- o La réponse. Cette première ligne n'est que le statut de la réponse, les octets qui la détaillent se trouvent plus loin, dans le corps du message. Trois champs la composent, elle se termine par CRLF :

**La version du protocole**, sous forme HTTP-Numéro de version, comme pour la question.

<sup>6</sup> "Uniform Resource Locator", consulter la RFC 1630

**Statut** C'est une valeur numérique qui décrit le statut de la réponse. Le premier des trois digits donne le sens général :

- 1xx N'est pas utilisé "Futur Use"
- 2xx Succès, l'action demandée a été comprise et exécutée correctement.
- 3xx Redirection. La partie cliente doit reprendre l'interrogation, avec une autre formulation.
- 4xx Erreur coté client. La question comporte une erreur de syntaxe ou ne peut être acceptée.
- 5xx Erreur coté serveur. Il peut s'agir d'une erreur interne, due à l'OS ou à la ressource devenue non accessible.

**Phrase** C'est un petit commentaire ("Reason-Phrase") qui accompagne le statut, par exemple le statut 200 est suivi généralement du commentaire "OK" !

**B** C'est une partie optionnelle, qui contient des informations à propos du corps du message. Sa syntaxe est proche de celle employée dans le courrier électronique, et pour cause, elle respecte aussi le standard MIME<sup>7</sup>. Un en-tête de ce type est constitué d'une suite d'une ou plusieurs lignes (la fin d'une ligne est le marqueur CRLF) construite sur le modèle :

**Nom\_de\_champ** : *Valeur\_du\_champ* CRLF

Éventuellement le marqueur de fin de ligne peut être omis pour le séparateur " ;".

Exemple d'en-tête MIME :

```
Date: Fri, 01 Mar 2002 10:59:06 GMT
Server: Apache/1.3.23 (Unix)
Last-Modified: Sat, 10 Nov 2001 16:13:02 GMT
ETag: "1381-8b-3bed520e"
Accept-Ranges: bytes
Content-Length: 79
Connection: close
Content-Type: text/html
```

**Date** : C'est la date à laquelle le message a été envoyé. Bien sûr il s'agit de la date du serveur, il peut exister un décalage incohérent si les machines ne sont pas synchronisées (par exemple avec XNTP).

**Server** : Contient une information relative au serveur qui a fabriqué la réponse. En générale la liste des outils logiciels et leur version.

**Content-type** : Ce champ permet d'identifier les octets du corps du message.

**Content-length** : Désigne la taille (en octets) du corps du message, c'est à dire la partie D de la *figure XV.1*.

<sup>7</sup>"Multipurpose Internet Mail Extension", consulter la RFC 1521



**Last-modified** : Il s'agit de la date de dernière modification du fichier demandé, comme l'illustre le résultat de la commande 11 (voir aussi la coïncidence de la taille du fichier et la valeur du champ précédent).

```
-rw-r--r--  1 web  doc   139 Nov 10 17:13 index.html
```

**ETag** : C'est un identificateur du serveur, constant lors des échanges. C'est un moyen de maintenir le dialogue avec un serveur en particulier, par exemple quand ceux-ci sont en grappe pour équilibrer la charge et assurer la redondance.

**C** Une ligne vide (CRLF) qui est le marqueur de fin d'en-tête. Il est donc absolument obligatoire qu'elle figure dans le message. Son absence entraîne une incapacité de traitement du message, par le serveur ou par le client.

**D** Le corps du message. Il est omis dans certains cas, comme une requête avec la méthode **GET** ou une réponse à une requête avec la méthode **HEAD**.

C'est dans cette partie du message que l'on trouve par exemple les octets de l'HTML, ou encore ceux d'une image...

Le type des octets est intimement lié à celui annoncé dans l'en-tête, plus précisément dans le champ **Content-Type**.

Par exemple :

**Content-Type** : `text/html`  $\Rightarrow$  Le corps du message contient des octets à interpréter comme ceux d'une page écrite en HTML.

**Content-Type** : `image/jpg`  $\Rightarrow$  Le corps du message contient des octets à interpréter comme ceux d'une image au format `jpeg`

## 2 URIs et URLs

Le succès du “web” s’appuie largement sur un système de nommage des objets accessibles, qui en uniformise l’accès, qu’ils appartiennent à la machine sur laquelle on travaille ou distants sur une machine en un point quelconque du réseau (mais supposé accessible). Ce système de nommage universel est l’**url** (“Uniform Resource Locator” — RFC 1738) dérivé d’un système de nommage plus général nommé **uri** (“Universal Resource Identifier” — RFC 1630).

La syntaxe générale d’un(e) url est de la forme :

**<scheme> :<scheme-specific-part>**

Succintement la “scheme” est une méthode que l’on sépare à l’aide du caractère “:” d’une chaîne de caractères ascii 7 bits dont la structure est essentiellement fonction de la “scheme” qui précède et que l’on peut imaginer comme un argument.

Une “scheme” est une séquence de caractères 7bits. Les lettres “a” à “z”, les chiffres de “0” à “9”, le signe “+”, le “.” et le “-” sont admis. Majuscules et minuscules sont indifférenciés.

Exemples de “schemes” : **http**, **ftp**, **file**, **mailto**, ... Il en existe d’autres (cf la RFC) non indispensables pour la compréhension de cet exposé.

Globalement une url doit être encodée en ascii 7 bits<sup>8</sup> sans caractère de contrôle (c’est à dire entre les caractères 20 et 7F), ce qui a comme conséquence que tous les autres caractères doivent être encodés.

La méthode d’encodage transforme tout caractère non utilisable directement en un triplet formé du caractère “%” et de deux caractères qui en représente la valeur hexadécimale. Par exemple l’espace (20 hex) doit être codé **%20**.

Un certain nombre de caractères, bien que théoriquement représentables, sont considérés comme non sûrs (“unsafe”) et devraient être également encodés de la même manière que ci-dessus, ce sont :

% < > " # { } | \ ^ ~ [ ] ‘

Pour un certain nombre de “schemes” (**http**...) certains caractères sont réservés car ils ont une signification particulière. Ce sont :

; / ? : @ = &

Ainsi, s’ils apparaissent dans l’url sans faire partie de sa syntaxe, ils doivent être encodés.

### 2.1 Scheme http

Une url avec la “scheme” **http** bien formée doit être de la forme :

---

<sup>8</sup>man ascii sur toute machine unix

`http://$<$host$>$:$<$port$>$/$<$path$>?$<$searchpath$>$`

“path” et “searchpath” sont optionnels.

**host** C’est un nom de machine ou une adresse IP.

**port** Le numéro de port. S’il n’est pas précisé, la valeur 80 est prise par défaut.

**path** C’est un sélecteur au sens du protocole **http**.

**searchpath** C’est ce que l’on appelle la “query string”, autrement dit la chaîne d’interrogation.

À l’intérieur de ces deux composantes, les caractères `/ ; ?` sont réservés, ce qui signifie que s’ils doivent être employés, ils doivent être encodés pour éviter les ambiguïtés.

Le `?` marque la limite entre l’objet interrogeable et la “query string”. À l’intérieur de cette chaîne d’interrogation le caractère `+` est admis comme raccourci pour l’espace (ascii 20 hex). Il doit donc être encodé s’il doit être utilisé en tant que tel.

De même, à l’intérieur de la “query string” le caractère `=` marque la séparation entre variable et valeur, le `&` marque la séparation entre les couples **variable = valeur**.

Exemple récapitulatif :

`http://www.google.fr/search?q=cours+r%E9seaux&hl=fr&start=10&sa=N`

Notez le “é” codé `%E9`, c’est à dire le caractère de rang  $14 \times 16 + 9 = 233$ . On peut également observer quatre variables **q**, **hl**, **start** et **sa** dont la signification peut être partiellement devinée, mais dont le remplissage reste à la charge du serveur en question.

Le rôle de la chaîne “**search**” est celui de ce que l’on appelle une CGI ou “Common Gateway Interface”, c’est à dire un programme qui effectue le lien entre le serveur interrogé et des programmes d’application, ici une recherche dans une base de données. Nous examinons succinctement le principe de fonctionnement de tels programmes page 347.

### 3 Architecture interne du serveur Apache

Cette partie se consacre à l'étude du fonctionnement du serveur Apache<sup>9</sup>.

Pour comprendre les mécanismes internes d'un tel serveur il est absolument nécessaire de pouvoir en lire le code source, cette contrainte exclu les produits commerciaux.

Au mois de mars 2002, d'après le "Netcraft Web Server Survey"<sup>10</sup> le serveur le plus utilisé (55%) est très majoritairement celui du projet Apache.

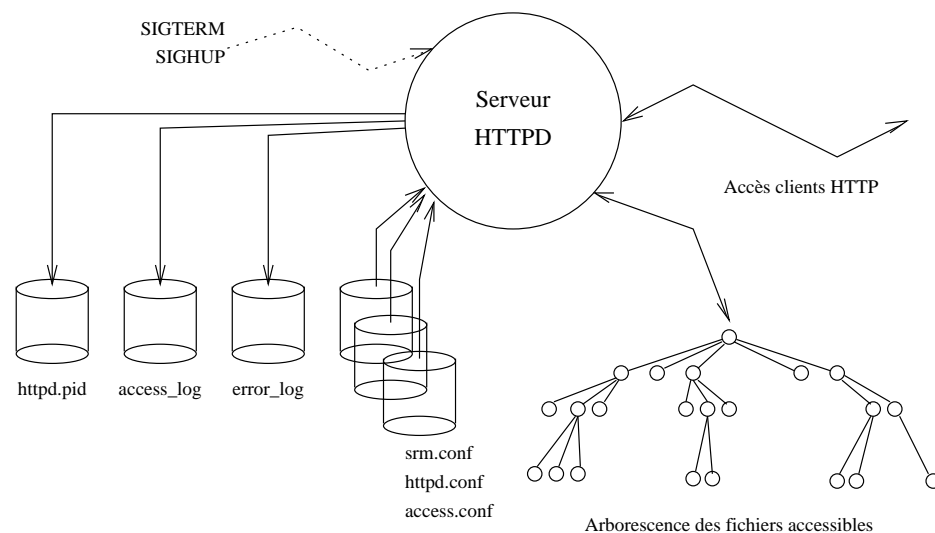
D'après ses auteurs, le serveur Apache est une solution de continuité au serveur du NCSA<sup>11</sup>. Il corrige des bugs et ajoute de nombreuses fonctionnalités, particulièrement un mécanisme d'API pour permettre aux administrateurs de sites de développer de nouveaux modules adaptés à leurs besoins propres.

Plus généralement, tout ce qui n'est pas strictement dans les attributions du serveur (gestion des processus, gestion mémoire, gestion du réseau) est traité comme un module d'extension. Le fichier `apache_1.3.23/htdocs/manual/misc/API.html` de la distribution standard apporte des précisions, le serveur <http://www.apacheweek.com/> pointe également sur grand nombre de documents très utiles.

Le serveur Apache introduit également la possibilité de serveurs multi-domaines (domaines virtuels), ce qui est fort apprécié des hébergeurs de sites.

#### 3.1 Environnement d'utilisation

La figure XV.2 qui suit, synthétise l'environnement d'utilisation.



<sup>9</sup> "Apache HTTP server project", <http://www.apache.org>, <http://www.apacheweek.com/> est également une source intéressante d'informations

<sup>10</sup> <http://www.netcraft.com/survey/>

<sup>11</sup> "National Center for Supercomputing Applications" – Université de l'Illinois, aux États Unis

Le serveur se met en œuvre simplement. La compilation fournit un exécutable, `httpd`, qui, pour s'exécuter correctement, a besoin des trois fichiers ASCII de configuration : `srn.conf`, `access.conf`, `httpd.conf`. C'est en fait dans celui-ci que sont effectués l'essentiel des ajustements locaux à la configuration standard.

Lors de l'exécution, trois fichiers sont modifiés<sup>12</sup> :

`httpd.pid` (`PidFile`) contient le "process ID" du "leader" de groupe ; à utiliser avec le signal `SIGHUP` (cf *figure XV.2*) pour provoquer la re-lecture et le re-démarrage "à chaud" du serveur, ou avec `SIGTERM` pour mettre fin à son activité.

`access_log` (`CustomLog`) Qui contient le détail des accès clients. Ce fichier peut devenir très volumineux.

`error_log` (`ErrorLog`) Qui contient le détail des accès infructueux et des éventuels problèmes de fonctionnement du serveur.

Le "daemon" `httpd` est soit du type (`ServerType`) "standalone" ou si son invocation est occasionnelle, à la demande piloté par `inetd` (cf page 4).

Il démarre son activité par ouvrir le port (`Port`) désigné dans la configuration, puis s'exécute avec les droits de l'utilisateur `User` et du groupe `Group`. Sa configuration se trouve dans le répertoire `conf` sous-répertoire de `ServerRoot`. Les fichiers accessibles par les connexions clientes, eux, se situent dans le répertoire `DocumentRoot` qui en est la racine, c'est à dire vue comme "/" par les browsers clients.

Le fichier `httpd.pid` contient un numéro de processus "leader" de groupe car en fait le serveur Apache, dès son initialisation, démarre un certain nombre de processus, autant de serveurs capables de comprendre les requêtes des clients. En voici un exemple :

|                              |     |   |
|------------------------------|-----|---|
| <code>MinSpareServers</code> | 5   | C'est le nombre minimum d'instances du serveur (non compris le processus maître) en attente d'une connexion. S'il en manque, elles sont créées.         |
| <code>MaxSpareServers</code> | 10  | C'est le nombre maximum d'instances du serveur (non compris le processus maître) en attente d'une connexion. S'il y en a de trop elles sont supprimées. |
| <code>StartServers</code>    | 5   | C'est le nombre minimum d'instances du serveur (non compris le processus maître) au démarrage.  |
| <code>MaxClients</code>      | 150 | C'est le nombre maximum de clients (requêtes HTTP) simultanés. Cette constante peut être augmentée en fonction de la puissance de la machine.           |

Un processus joue le rôle de régulateur, du point de vue Unix c'est un processus chef de groupe ("leader"). La commande `ps` permet de visualiser une situation opérationnelle :

<sup>12</sup>Entre parenthèses le nom de la variable du le fichier `httpd.conf` qui permet d'en modifier le chemin

```

web 17361 2794 0 Mar 13 ? 0:00 /usr/local/bin/httpd -d /users/web
root 2794 1 0 Feb 23 ? 0:06 /usr/local/bin/httpd -d /users/web
web 17363 2794 0 Mar 13 ? 0:00 /usr/local/bin/httpd -d /users/web
web 17270 2794 0 Mar 13 ? 0:01 /usr/local/bin/httpd -d /users/web
web 17362 2794 0 Mar 13 ? 0:00 /usr/local/bin/httpd -d /users/web
web 17401 2794 0 Mar 13 ? 0:00 /usr/local/bin/httpd -d /users/web
web 17313 2794 0 Mar 13 ? 0:00 /usr/local/bin/httpd -d /users/web
web 17312 2794 0 Mar 13 ? 0:01 /usr/local/bin/httpd -d /users/web
web 17355 2794 0 Mar 13 ? 0:00 /usr/local/bin/httpd -d /users/web
web 17314 2794 0 Mar 13 ? 0:01 /usr/local/bin/httpd -d /users/web

```

Ici il y a 9 instances du serveurs et le processus maître, qu'il est aisé de reconnaître (2794) car il est le père des autres processus (ce qui n'implique pas qu'il en est le chef de groupe, mais la suite de l'analyse nous le confirmera).

La commande `netstat -f inet -a | grep http` ne donne qu'une ligne :

```
tcp          0      0 *.http          *.*             LISTEN
```

Cela signifie qu'aucune connexion n'est en cours sur ce serveur, et qu'une seule "socket" est en apparence à l'écoute du réseau. C'est une situation qui peut sembler paradoxale eu égard au nombre de processus ci-dessus, le code nous fournira une explication au paragraphe suivant.

La commande `tail -1 logs/access.log` fournit la trace de la dernière requête :

```
www.chezmoi.tld - - [01/Mar/2002:17:13:28 +0100] "GET / HTTP/1.0" 200 79
```

Il s'agit de la trace de notre exemple d'interrogation du début de ce chapitre !

## 3.2 Architecture interne

**Attention, ce paragraphe concerne la version 1.1.1 du logiciel. Le fonctionnement de la version courante, 1.3.23, reste similaire mais le code ayant beaucoup changé, les numéros de lignes sont devenus de fait complètement faux.**

Ce qui nous intéresse plus particulièrement pour expliquer le fonctionnement du serveur se trouve dans le répertoire `src/`, sous répertoire du répertoire principal de la distribution :

```

$ ll apache_1.1.1/
total 19
-rw----- 1 fla users 3738 Mar 12 1996 CHANGES
-rw----- 1 fla users 2604 Feb 22 1996 LICENSE
-rw----- 1 fla users 3059 Jul 3 1996 README
drwxr-x--- 2 fla users 512 Feb 7 22:14 cgi-bin
drwxr-x--- 2 fla users 512 Feb 7 22:14 conf
drwxr-x--- 2 fla users 512 Feb 7 22:14 htdocs
drwxr-x--- 2 fla users 2048 Feb 7 22:14 icons
drwxr-x--- 2 fla users 512 Jul 8 1996 logs
drwxr-x--- 2 fla users 2048 Mar 12 10:42 src
drwxr-x--- 2 fla users 512 Feb 7 22:15 support

```

Dans ce répertoire qui compte au moins 75 fichier (`wc -l *.c` ⇒ 27441 lignes) nous nous restreignons aux fichiers essentiels décrits dans le “README” soit encore une petite dizaine d’entre eux (`wc -l` ⇒ 6821 lignes) : `mod_cgi.c`, `http_protocol.c`, `http_request.c`, `http_core.c`, `http_config.c`, `http_log.c`, `http_main.c`, `alloc.c`

Dans un premier temps nous allons examiner le fonctionnement de la gestion des processus, du mode de relation entre le père et ses fils.

Puis, dans un autre paragraphe, nous examinerons plus particulièrement ce qui se passe lors d’une connexion, avec le cas particulier de l’exécution d’une CGI<sup>13</sup> qui comme son nom l’indique est le programme qui effectue l’interface entre le serveur HTTP et un programme d’application quelconque. Dans la suite de ce document le terme “cgi” employé seul désigne ce type d’application.

### 3.2.1 Gestion des processus

Au commencement est le `main`, et celui-ci se trouve dans le fichier `http_main.c`, comme il se doit !

```

...
1035 pool *pconf;                /* Pool for config stuff */
1036 pool *ptrans;                /* Pool for per-transaction stuff */
...
1472 int
1473 main(int argc, char *argv[])
1474 {
...
1491     init_alloc();
1492     pconf = permanent_pool;
1493     ptrans = make_sub_pool(pconf);

```

La fonction `init_alloc` appelle `make_sub_pool` qui initialise un intéressant mécanisme de buffers chaînés, utilisé tout au long du code dès lors qu’il y a besoin d’allouer de la mémoire.

```

...
1523     setup_prelinked_modules();

```

Les différents modules du serveurs sont ajoutés dans une liste chaînée.

```

1524
1525     server_conf = read_config (pconf, ptrans, server_confname);
1526
1527     if(standalone) {
1528         clear_pool (pconf);    /* standalone_main rereads... */

```

---

<sup>13</sup>“Common Gateway Interface”

```

1529         standalone_main(argc, argv);
1530     }
1531     else {
1532         ...
1580     }
1581     exit (0);
1582 }
```

“Standalone” est à 0 si le serveur est invoqué depuis `inetd` <sup>14</sup>. Son mode de fonctionnement le plus efficace reste avec ce paramètre à 1 (voir le cours sur `inetd`), que nous supposons ici.

La fonction `standalone_main` (ligne 1362) prépare le processus à son futur rôle de serveur. Pour bien comprendre de cette fonction, il faut imaginer qu'elle est invoquée au démarrage, et “à chaud”, pour lire et relire la configuration.

**Ligne 1369** , la variable `one_process = 0` (sinon le processus est en mode debug) occasionne l'appel à la fonction `detach` (ligne 876). Celle-ci transforme le processus en “leader” de groupe avec la succession bien connue `fork + setsid` (pour plus de détails, voir le cours sur les daemons).

**Ligne 1374** , la fonction `sigsetjmp` enregistre la position de la pile et l'état du masque d'interruption (deuxième argument non nul) dans `restart_buffer`. Tout appel ultérieur à `siglongjmp` forcera la reprise de l'exécution en cette ligne.

**Ligne 1377** On ignore tout signal de type `SIGHUP`, cela bloque toute tentative cumulative de relecture de la configuration.

**Ligne 1382** (`one_process = 0`) on envoie un signal `SIGHUP` à tous les processus du groupe. Cette disposition n'est utile que dans le cas d'un re-démarrage “à chaud”. La variable `pgrp` est initialisée par la fonction `detach` (ligne 876), elle contient le PID du chef de groupe.

L'intérêt d'avoir un groupe de processus est que le signal envoyé à son “leader” est automatiquement envoyé à tous les processus y appartenant.

Chaque processus qui reçoit ce signal, s'il est en mesure de le traiter, appelle la fonction `just_die` qui exécute un `exit(0)` (ligne 943). Donc tous les fils meurent, sauf le chef de groupe.

**Ligne 1390** , l'appel à la fonction `reclaim_child_processes()` effectue autant de `wait` qu'il y avait de processus fils, pour éviter les zombies.

**Ligne 1398** Relecture des fichiers de configuration.

**Ligne 1400** `set_group_privs` (ligne 960) change le “user id” et le “group id” si ceux-ci ont été modifiés.

---

<sup>14</sup>Cette alternative est décidée dans le fichier `httpd.conf`, configuration du paramètre `ServerType`



**Ligne 1401** `accept_mutex_init` (ligne 166) fabrique un fichier temporaire (`/usr/tmp/htlock.XXXXXX`), l'ouvre, réduit son nombre de lien à 0, de telle sorte qu'il sera supprimé dès la fin du processus. Ce fichier est le verrou d'accès à la socket principale, comme nous l'examinerons un peu plus loin.

**Ligne 1402** `reinit_scoreboard` (ligne 596) Cette fonction remet à zéro, ou crée (`setup_shared_mem` ligne 432) la zone de mémoire commune entre le chef de groupe et ses processus fils. Cette zone mémoire est, soit un segment de mémoire partagée (IPC du système V), soit de la mémoire rendue commune par un appel à la primitive `mmap` (Le choix est effectué par `configure` qui analyse les possibilités du système, avant compilation du serveur).

La taille de cette zone est de `HARD_SERVER_LIMIT × sizeof(short_score)` octets, la structure `short_score`, définie dans `scoreboard.h`, recueille les statistiques d'utilisation de chaque serveur et son statut (par exemple `SERVER_READY`, `SERVER_DEAD...`). C'est par ce moyen que le serveur maître contrôle les serveurs fils.

`HARD_SERVER_LIMIT` définit le nombre maximum de connexions actives, donc d'instances de serveur, à un instant donné. En standard cette valeur est 150, elle peut être augmentée dans le fichier de configuration `httpd.conf` (voir ci-dessus au paragraphe II.1)

Enfin la *figure XV.4* montre son rôle stratégique.

**Ligne 1413** (on suppose `listeners = NULL`), après avoir initialisé une structure d'adresse, appel à la fonction `make_sock` (ligne 1298). Celle-ci crée et initialise une socket, et, détail important, appelle par deux fois `setsockopt`, ce qui mérite un commentaire :

```
...      ...
1312      if((setsockopt(s, SOL_SOCKET,SO_REUSEADDR,(char *)&one,sizeof(one)))
1313          == -1) {
...      ...
1318      if((setsockopt(s, SOL_SOCKET,SO_KEEPALIVE,(char *)&keepalive_value,
1319          sizeof(keepalive_value))) == -1) {
...      ...
```

`SO_REUSEADDR` Indique que la règle d'exclusivité suivie par `bind(2)` pour l'attribution d'un port ne s'applique plus : un processus peut se servir d'un même port pour plusieurs usages différents (comme par exemple le client `ftp` qui attaque les port 21 et 20 du serveur avec le même port local), voire même des processus différents (c'est le cas ici) peuvent faire un `bind` avec le même port sans rencontrer la fatidique erreur 48 ("Address already in use")!

Vue des clients HTTP, le serveur est accessible uniquement sur le port 80, ce que nous avons remarqué au paragraphe II.1 (`netstat`) sans l'expliquer, voilà qui est fait !

**SO\_KEEPALIVE** Indique à la couche de transport, ici TCP, qu'elle doit émettre à interval régulier (non configurable) un message à destination de la socket distante. Que celle-ci n'y réponde pas et la prochaine tentative d'écriture sur la socket se solde par la réception d'un **SIGPIPE**, indiquant la disparition de la socket distante (voir plus loin la gestion de ce signal dans la fonction `child_main`, ligne 1139).

**Ligne 1430** `set_signals` (1007) prévoit le comportement lors de la réception des signaux :

**SIGHUP** Appel de `restart`  
**SIGTERM** Appel de `sig_term`

**Ligne 1438** et la suivante, création d'autant de processus fils qu'il en est demandé dans le fichier de configuration (`StartServers`). La fonction `make_child` (1275) appelle `fork`, puis dans le fils modifie le comportement face aux signaux **SIGHUP** et **SIGTERM** (`just_die` appelle `exit`) avant d'exécuter `child_main`.

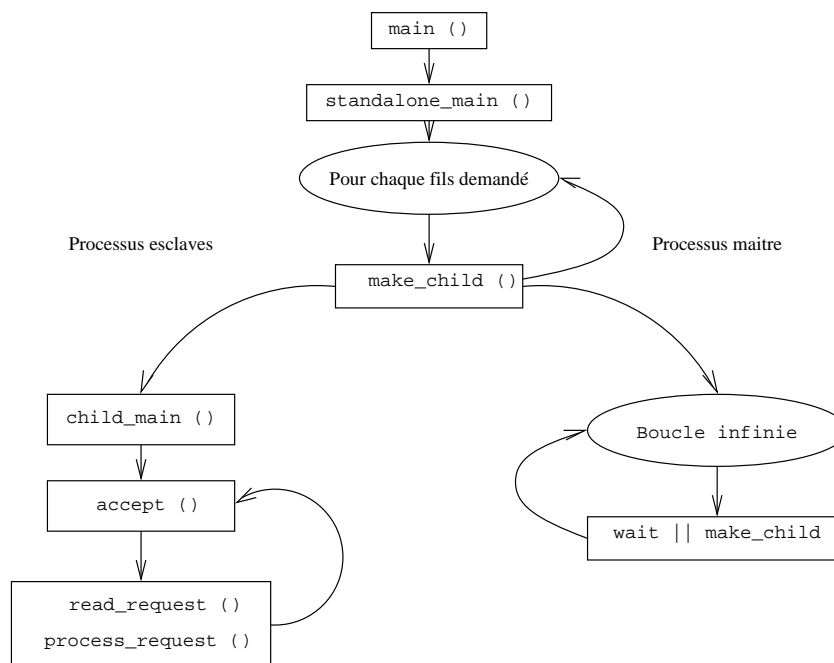


figure XV.03

Arrivés à ce stade, il nous faut analyser l'attitude des deux types de processus.

### Le processus maître

**Ligne 1444** démarre une boucle infinie de surveillance. Seule la réception et le traitement d'un signal peut l'interrompre.

**Ligne 1458** Ici, si le nombre de serveurs disponibles est inférieur au nombre minimal requis, il y a régénération de ceux qui manquent avec la fonction `make_child`

### Les processus esclaves

**Ligne 1139** La gestion de ces processus, autant de serveurs Web opérationnels, débute avec l'appel de la fonction `child_main`.

**Ligne 1167** Début de la boucle infinie qui gère cette instance du serveur. Au cours d'une boucle le serveur gère l'acceptation d'une requête et son traitement.

**Ligne 1174** Gestion du SIGPIPE donc des clients qui déconnectent avant l'heure !

**Ligne 1180** Si le nombre de serveurs libres (`count_idle_servers`) est supérieur à la limite configurée, ou si

**Ligne 1182** le nombre de requêtes traitées par ce processus a atteint la limite `max_requests_per_child`, le processus s'arrête de lui-même. C'est l'auto-régulation pour libérer des ressources occupées par un trop grand nombre de processus serveurs inutiles.

**Ligne 1190** L'appel à la fonction `accept_mutex_on` verrouille l'accès à la ressource définie précédemment avec `accept_mutex_init` (ligne 166). Ce verrouillage est bloquant et exclusif. C'est à dire qu'il faut qu'un autre processus en déverrouille l'accès pour que le premier processus sur la liste d'attente (gérée par le noyau Unix) y accède.

Ce fonctionnement est assuré suivant la version d'Unix par la primitive `flock` ou par la primitive `fcntl`.

Les sémaphore du Système V (`semget`, `semctl`, `semop`...) assurent la même fonctionnalité, en plus complet, ils sont aussi plus complexes à mettre en œuvre.

Cette opération est à rapprocher de l'option `SO_REUSEADDR` prise ligne 1312. Il faut éviter que plusieurs processus serveurs disponibles ne répondent à la requête. Il n'y a qu'un seul processus prêt à répondre à la fois et dès que le `accept` (ligne 1217) retourne dans le code utilisateur la ressource est déverrouillée (on suppose toujours `listeners = 0`).

**Ligne 1221** La fonction `accept_mutex_off` déverrouille l'accès à la socket.

**Ligne 1245** `read_request` lit la requête du client, donc un message HTTP.

**Ligne 1247** `process_request` fabrique la réponse au client, donc un autre message HTTP.

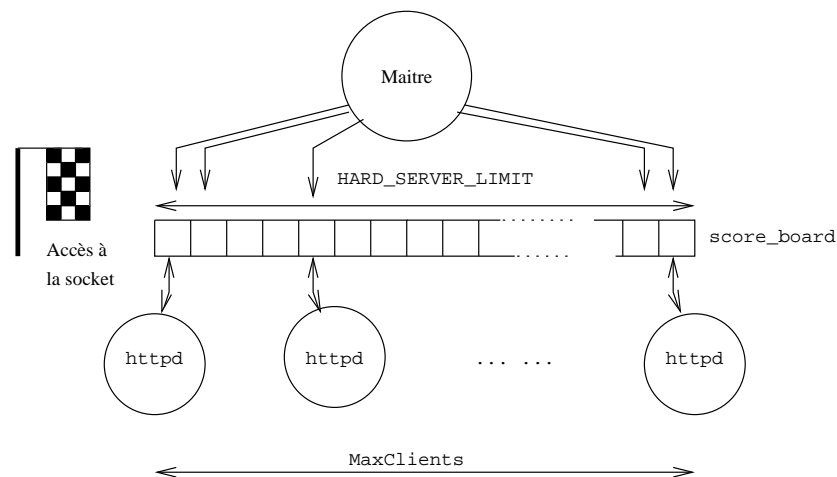


figure XV.04

### 3.2.2 Prise en main des requêtes

Le fichier concerné par la lecture de la requête est `http_protocol.c`.

La lecture de la première ligne du message HTTP est assurée par la fonction `read_request_line`, ligne 329<sup>15</sup>.

La lecture de l'en-tête MIME est assurée par la fonction `get_mime_headers`, ligne 356. Attention, seule l'en-tête lue le corps du message dans le cas de la méthode POST est lu plus tard, lors du traitement du message, par le programme d'application (CGI).

Le fichier concerné par la formulation de la réponse est `http_request.c` et la fonction principale `process_request`, ligne 772. Celle-ci appelle en cascade `process_request_internal`, ligne 684.

Cette dernière effectue un certain nombre de tests avant de traiter effectivement la requête. Parmi ceux-ci on peut relever,

**Ligne 716** La fonction `unescape_url` (`util.c`, ligne 593) assure le décodage des caractères réservés et transcodés comme il est spécifié par la RFC 1738.

**Ligne 723** La fonction `getparents` filtre les chemins ("pathname") qui prêtent à confusion.

**Ligne 768** La fonction `invoke_handler` est le point d'entrée dans le traitement de la requête. Cette fonction (`http_config.c`, ligne 267) invoque le programme (module) qui se charge de fabriquer la réponse, au vue du contenu du champ `content_type` de la requête. Si celui est inexistant, comme dans notre exemple du paragraphe I, il est positionné par défaut à la valeur `html/text`.

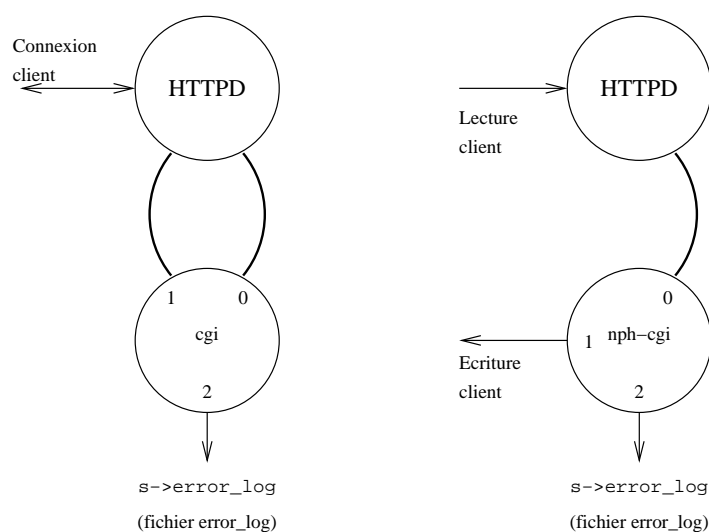
<sup>15</sup>La méthode, l'URL, et le protocole demandé

### 3.2.3 Deux types de CGI

Pour la suite nous supposons que la prise en main de la requête est faite par le module “cgi”, défini dans le fichier `mod_cgi.c`. Le point d’entrée est la fonction `cgi_handler` (ligne 207), c’est à dire celle qui appelée par `invoke_handler`, vue au paragraphe ci-dessus.

La lecture du code permet de déduire qu’il y a deux types de CGI, la distinction est faite par le nom de la cgi elle-même.

La *figure XV.5* résume les 2 situations possibles d’exécution d’une CGI.



*figure XV.05*

Si le nom de l’exécutable commence par `nph`<sup>16</sup> le comportement du serveur http change. Dans le cas ordinaire (nom quelconque) les données transmises depuis le client vers la cgi et réciproquement, passent par le processus httpd et via un tube non nommé (“pipe”).

Dans le cas d’une cgi “nph”, seules les données lues depuis le client (par exemple avec la méthode POST) transitent par le processus httpd, la réponse, elle, est émise directement, ce qui améliore les performances en évitant une séquence lecture/écriture inutile. Ce comportement est justifié dès que de gros volumes d’octets sont à transmettre au client (de grandes pages HTML, des images. ...).

Attention, dans ce dernier cas, c’est à la CGI de fabriquer l’intégralité du message HTTP, y compris l’en-tête MIME. A elle également de gérer la déconnexion prématurée du client (SIGPIPE).

Ces deux modes de fonctionnement ne sont pas clairement documentés, en fait il s’agit d’une caractéristique du serveur du *CERN*, maintenue pour assurer sans doute la compatibilité avec les applicatifs déjà écrits. Il n’est pas assuré que cette possibilité existera dans les futures versions du serveur

<sup>16</sup> “Non Parse Header”

Apache, notamment celles qui utiliseront la version 1.1 d'HTTP.

Examinons le code du fichier `mod_cgi.c` :

```

207 int cgi_handler (request_rec *r)
208 {
209     int np;
210     ...
222     np = !(strcmp(argv0,"np-",4));
223     ...

```

La variable `np` vaut 1 si la cgi est de ce type.

```

...     ...
251     add_common_vars (r);
...     ...

```

Ici on commence à fabriquer l'environnement d'exécution de la cgi. Cette fonction (fichier `util_script.c`, ligne 126) complète les variables qui ne dépendent pas du contenu de la requête, par exemple `SERVER_SOFTWARE`, `REMOTE_HOST`,...

```

...     ...
277     if (!spawn_child (r->connection->pool, cgi_child, (void *)&cld,
278                       np ? just_wait : kill_after_timeout,
279                       &script_out, np ? NULL : &script_in)) {
280         log_reason ("couldn't spawn child process", r->filename, r);
281         return SERVER_ERROR;
282     }
...     ...

```

L'appel de cette fonction provoque la création d'un processus fils, celui qui finalement va exécuter la cgi. Il faut remarquer le deuxième argument qui est un pointeur de fonction (`cgi_child`), et le sixième qui est nul dans le cas d'une cgi du type "np".

`script_in` et `script_out` sont respectivement les canaux d'entrée et sortie des données depuis et vers le processus qui exécute la cgi. Il paraît donc logique que dans le cas d'une cgi de type "np" `script_in` soit nul. Un mécanisme non encore analysé duplique la sortie de ce processus vers le client plutôt que vers le processus serveur.

Nous continuons la description du point de vue du processus père, donc `httpd`.

Ligne 295 et les suivantes, jusqu'à la ligne 332, le processus lit ce que le client lui envoie, si la méthode choisie est du type POST. Le contenu est renvoyé vers le processus fils, sans transformation :

```
311          if (fwrite (argsbuffer, 1, len_read, script_out) == 0)
312              break;
...      ...
335      pfclose (r->connection->pool, script_out);
```

Il est donc clair que c'est à celui-ci d'exploiter ce qu'envoie le client, par exemple le résultat d'une forme de saisie.

```
337      /* Handle script return... */
338      if (script_in && !nph) {
...      ...
373          send_http_header(r);
374          if(!r->header_only) send_fd (script_in, r);
375          kill_timeout (r);
376          pfclose (r->connection->pool, script_in);
377      }
```

Ce test traite le cas d'une cgi normale, dont la sortie est lue par le serveur, puis renvoyée au client (ligne 374).

Examinons maintenant comment se prépare et s'exécute le processus fils :

```
101 void cgi_child (void *child_stuff)
102 {
...      ...
126      add_cgi_vars (r);
127      env = create_environment (r->pool, r->subprocess_env);
```

Ces deux dernières lignes préparent l'environnement du futur processus fils. Il s'agit du tableau de variables accessibles depuis la variable externe `environ`, pour tout processus. La fonction `add_cgi_vars` (fichier `util_script.c`, ligne 192) ajoute, entres autres, les variables `REQUEST_METHOD` et `QUERY_STRING` à l'environnement.

Cette dernière variable joue un rôle majeur dans la transmission des arguments à la cgi quand la méthode choisie est GET. En effet, dans ce cas, le seul moyen pour le client d'envoyer des arguments à la cgi est d'utiliser l'URL, comme par exemple dans :

<http://monweb.chez.moi/cgi-bin/nph-qtp?base=datas&mot=acacia&champ=.MC>

La syntaxe de l'URL prévoit le caractère “?” comme séparateur entre le nom et ses arguments. Chaque argument est ensuite écrit sous la forme :

nom = valeur

Les arguments sont séparés les uns des autres par le caractère “&”.

```
135     error_log2stderr (r->server);
136
...
138     if (nph) client_to_stdout (r->connection);
```

Ligne 135, la sortie d'erreur est redirigée vers le fichier `error_log`, et ligne 138, la sortie standard du processus, c'est la socket, donc envoyée directement vers le client !

```
184     if ((!r->args) || (!r->args[0]) || (ind(r->args, '=') >= 0))
185         execle(r->filename, argv0, NULL, env);
186     else
187         execve(r->filename, create_argv(r->pool, argv0, r->args), env);
```

Puis finalement on exécute la cgi ! Bien sûr, si le programme va au delà de la ligne 187, il s'agit d'une erreur...



## 4 Principe de fonctionnement des CGIs

### 4.1 CGI — Méthode GET, sans argument

Dans ce paragraphe nous examinons ce qui se passe lors de l'exécution d'une CGI très simple (shell script, le source suit) que l'on suppose placée dans le répertoire idoine, pour être exécutée comme lors de la requête suivante :

```
$ telnet localhost 80
Trying ...
Connected to localhost.
Escape character is '^]'.
GET /cgi-bin/nph-test-cgi HTTP/1.0
```

La requête tapée par l'utilisateur.

```
HTTP/1.0 200 OK
Content-type: text/plain
Server: Apache/1.1.1
```

L'en-tête du message HTTP renvoyé par le serveur. La ligne de statut est générée par la cgi car il s'agit d'une cgi de type `nph-`

```
CGI/1.0 test script report:
```

```
argc is 0. argv is .
```

```
SERVER_SOFTWARE = Apache/1.1.1
SERVER_NAME = www.chezmoi.tld
GATEWAY_INTERFACE = CGI/1.1
SERVER_PROTOCOL = HTTP/1.0
SERVER_PORT = 80
REQUEST_METHOD = GET
SCRIPT_NAME = /cgi-bin/nph-test-cgi
QUERY_STRING =
REMOTE_HOST = labas.tresloin.tld
REMOTE_ADDR = 192.168.0.1
REMOTE_USER =
CONTENT_TYPE =
CONTENT_LENGTH =
Connection closed by foreign host.
```

Le corps du message. Ces octets sont générés dynamiquement par le programme `nph-test-cgi`.

Et voici le source de cet exemple (une modification du script de test livré avec Apache) :

```
#!/bin/sh

echo HTTP/1.0 200 OK
echo Content-type: text/plain
echo Server: $SERVER_SOFTWARE
echo

echo CGI/1.0 test script report:
echo

echo argc is $#. argv is "$*".
echo
```

Remarquez la fabrication de l'en-tête MIME, réduite mais suffisante. Le `echo` seul génère une ligne vide, celle qui marque la limite avec le corps du message.

```

echo SERVER_SOFTWARE = $SERVER_SOFTWARE
echo SERVER_NAME = $SERVER_NAME
echo GATEWAY_INTERFACE = $GATEWAY_INTERFACE
echo SERVER_PROTOCOL = $SERVER_PROTOCOL
echo SERVER_PORT = $SERVER_PORT
echo REQUEST_METHOD = $REQUEST_METHOD
echo QUERY_STRING = $QUERY_STRING
echo REMOTE_HOST = $REMOTE_HOST
echo REMOTE_ADDR = $REMOTE_ADDR
echo REMOTE_USER = $REMOTE_USER
echo CONTENT_TYPE = $CONTENT_TYPE
echo CONTENT_LENGTH = $CONTENT_LENGTH

```

Toutes ces variables sont celles de l'environnement généré par le module `cgi.handler` avant d'effectuer le `exec`.

## 4.2 CGI — Méthode GET, avec arguments

Examinons maintenant un cas plus compliqué, avec des arguments transmis, ou “query string”. Celle-ci est transmise à la cgi via la variable d'environnement `QUERY_STRING`. C'est à la cgi de l'analyser puisque son contenu relève de l'applicatif et non du serveur lui-même. Essayons de coder la cgi de l'exemple :

<http://www.chezmoi.tld/cgi-bin/query?base=datas&mot=acacia&champ=.MC>

Première version :

```

#!/bin/sh
echo Content-type: text/plain
echo
echo "QUERY_STRING=>"$QUERY_STRING "<="
exit 0

```

L'interrogation avec un `telnet` produit la sortie :

```
QUERY_STRING=>base=datas&mot=acacia&champ=.MC<=
```

Se trouve très facilement sur les sites `ftp` le source d'un outil nommé `cgiparse`<sup>17</sup>, parfaitement adapté à l'analyse de la chaîne transmise. D'où la **deuxième version** :

```

#!/bin/sh

CGIBIN=~web/cgi-bin
BASE='$CGIBIN/cgiparse -value base'
MOT='$CGIBIN/cgiparse -value mot'
CHAMP='$CGIBIN/cgiparse -value champ'

```

Cette partie du code montre l'analyse du contenu de la variable `QUERY_STRING` avec l'outil `cgiparse`.

<sup>17</sup>Il est dans la distribution standard du serveur `httpd-3.0.tar.gz` du *CERN*

```
echo Content-type: text/plain
echo
```

```
echo "BASE=$BASE"
echo "MOT=$MOT"
echo "CHAMP=$CHAMP"

exit 0
```

Et là, la fabrication du message renvoyé. La cgi renvoie ses octets via un tube au serveur, c'est donc celui-ci qui se charge de fabriquer un en-tête MIME.

Puis le résultat de l'exécution :

```
BASE=datas
MOT=acacia
CHAMP=.MC
```

### 4.3 CGI — Méthode POST

La méthode POST autorise un client à envoyer au serveur une liste de couples `variable=valeur`. Chaque couple est séparé des autres par une fin de ligne, c'est à dire (CR,LF).

Cette méthode est bien adaptée à la transmission d'informations collectées coté client dans une forme<sup>18</sup>de saisie, et dont le volume est variable.

La liste des couples est écrite dans le corps du message, par le programme client, et ne fait pas partie de l'URL, comme c'est le cas pour la méthode GET. Il y a évidemment une limite au nombre maximum d'octets envoyé par le client, c'est le serveur qui en fixe la valeur<sup>19</sup>. Du coté du client, il faut prévenir le serveur, dans l'en-tête du message HTTP, de la taille du corps du message. Cela s'effectue avec le champ `Content-length` :.

L'exemple suivant ne fait que renvoyer la liste des couples lus, vers le client. Attention il ne fonctionne qu'avec `telnet`.

```
$ telnet www.chezmoi.tld 80
Trying 192.168.0.2...
Connected to www.chezmoi.tld.
Escape character is '^]'.
POST /cgi-bin/test-post HTTP/1.0
Content-length:14
```

Le corps du message fait bien 14 caractères si on compte la fin de ligne (CR+LF).

```
areuh=tagada
HTTP/1.0 200 OK
Date: Mon, 24 Mar 1997 14:41:26 GMT
Server: Apache/1.1.1
Content-type: text/plain

REQUEST_METHOD = POST
CONTENT_LENGTH = 14
Couple lu : areuh=tagada
Connection closed by foreign host.
```

La réponse du serveur liste les couples lus, ici un seul! La variable globale `REQUEST_METHOD` pourrait être utilisée pour adapter le comportement de la cgi en fonction de la méthode demandée.

<sup>18</sup>Voir le "tag" FORM de l'HTML

<sup>19</sup>HUGE\_STRING\_LEN qui vaut 8192 en standard, définie dans le fichier `httpd.h`

Et voici le source de la cgi :

```
#!/bin/sh
#
# Ce script teste la methode POST
#
echo Content-type: text/plain
echo
echo REQUEST_METHOD = $REQUEST_METHOD
echo CONTENT_LENGTH = $CONTENT_LENGTH
while read l
do
    echo "Couple lu :" $l
done
exit 0
```

#### 4.4 Ecriture d'une CGI en Perl

Voir cours de *Jean-Jacques Dhenin*...

## 5 Conclusion – Bibliographie

Rien n'est constant, tout change...Et il faut être constamment à l'affût de la nouveauté dans ce domaine très réactif qu'est celui du "World Wide Web".

Les quelques références bibliographique qui suivent illustrent ce cours, mais il est évident qu'elles sont bien insuffisantes pour couvrir tous les aspects que le terme "web" sous-entend !

**RFC 1521** N. Borenstein, N. Freed, "MIME (Multipurpose Internet Mail Extensions) Part One : Mechanisms for Specifying and Describing the Format of Internet Message Bodies", 09/23/1993. (Pages=81) (Format=.txt, .ps) (Obsoletes RFC1341) (Updated by RFC1590)

**RFC 1590** J. Postel, "Media Type Registration Procedure", 03/02/1994. (Pages=7) (Format=.txt) (Updates RFC1521)

**RFC 1630** T. Berners-Lee, "Universal Resource Identifiers in WWW : A Unifying Syntax for the Expression of Names and Addresses of Objects on the Network as used in the World-Wide Web", 06/09/1994. (Pages=28) (Format=.txt)

**RFC 1738** T. Berners-Lee, L. Masinter, M. McCahill, "Uniform Resource Locators (URL)", 12/20/1994. (Pages=25) (Format=.txt)

**RFC 1945** T. Berners-Lee, R. Fielding, H. Nielsen, "Hypertext Transfer Protocol – HTTP/1.0", 05/17/1996. (Pages=60) (Format=.txt)

**RFC 2068** R. Fielding, J. Gettys, J. Mogul, H. Frystyk, T. Berners-Lee, "Hypertext Transfer Protocol – HTTP/1.1", 01/03/1997. (Pages=162) (Format=.txt)

**TCP/IP Illustrated Volume 3** W. Richard Stevens – Addison-Wesley – Janvier 1996.



## Cinquième partie

### Index général & Annexes





# Index

- états de liens, 111
- aio\_read, 311
- aio\_write, 311
- close, 252, 255
- exec, 308
- fork, 255, 307, 315
- open, 252
- read, 252
- rfork, 307
- socket, 140, 253
- vfork, 307, 308
- write, 252
- 1000BaseT, 17
- 100BaseT, 16, 17
- 10Base2, 15, 15--16
- 10Base5, 15, 15
  - prise vampire, 15
- 10BaseT, 16
  - RJ45, 16
- 127.0.0.1, 37
- 224.0.0.5, 122
- 224.0.0.6, 122
- 802.11, 9
- 802.2, 9
- 802.3, 9
  
- accept, 263--264
- Accounting management, 221
- active open, voir ouverture
  - active
- adresse de loopback, 37
- adresse Ethernet, voir
  - Ethernet
- adresse IEEE 802.3, 11
- adresse IP, voir IP
- adresse IP privée, 34
- adresse IP publique, 34
- adresse physique, 11
  
- AF, 254, voir Address Family
- AFNIC, voir Association
  - Française pour le
  - Nommage Internet en
  - Coopération
- AfriNIC, 34
- agent, voir SNMP
- algorithme Bellman-Ford, 111, 113, 120
- Algorithme concourant - Mode connecté, 306
- Algorithme concourant - Mode datagramme, 305
- algorithme de routage, voir routage
- Algorithme Itératif - Mode connecté, 305
- Algorithme itératif - Mode data-gramme, 303
- alias IP, 44, 137
- API, voir Application Program Interface
- APNIC, 34
- Application Program Interface, 251
- ARIN, 34
- ARP, 12, 55, 60, 109
  - Fonctionnement, 55
  - Format du datagramme, 57
  - HARDWARE TYPE, 57
  - HLEN 1, 57
  - HLEN 2, 57
  - OPERATION, 57, 58
  - PROTOCOL TYPE, 57
  - Proxy arp, 56, 58
  - SENDER ADR, 57
  - SENDER HA, 57
  - TARGET ADR, 57

- TARGET HA, 57
- arp, voir commande
- Arpanet, 25, 47, 110, 121, 189
- arpwatch, voir commande
- AS, 71, 110--111
- ASN.1, 229--231, 247
- Association Française pour
  - le Nommage Internet en
  - Coopération, 3
- Authentication Header, 144
- Autonomous systems, voir AS
- base de données distribuée,
  - 169
- bases whois, 168
- Basic Encoding Rules, voir BER
- bcmp, 278
- bcopy, 278
- BER, 231, 247
- Berkeley Internet Name Domain,
  - 186
- BGP, 71, 110
- BGP-4, 40
- big endian, 268, 277
- BIND, voir Berkeley Internet
  - Name Domain
- bind, 256--258, 260
- Bind Operations Guide, 187
- BOG, voir Bind Operations
  - Guide
- BOOTP, 109
- Border Gateway Protocol, voir
  - BGP
- broadcast, 20
- BSD 4.1c, 251
- BSD 4.3, 252, 320
- bzero, 278
- cache dns, 174
- CARP, 12, 137
- CCITT, 229--231
- chroot, 138
- CIDR, voir Classless
  - InterDomain Routing
- circuit virtuel, 89, 90, 302
- Clamav, 209
- Classless InterDomain Routing,
  - 40
- close, 264
- closelog, 318
- CMIP, 224
- CMIS, 224
- commade
  - arpwatch, 223
- commande
  - arp
    - a, 56
    - an, 232
  - cvs, 213
  - gated, 67
  - httptunnel, 140
  - ifconfig, 141
  - inetd, 301, 312, 320--322
  - init, 315
  - ipf, 154
  - ipfw, 154
  - m4, 212
  - mailq, 208
  - mbrowse, 244
  - mrtg, 245
  - mutt, 191
  - named, 186
  - natd, 150, 159
  - netstat
    - p tcp, 229
    - rn, 68
    - s -p icmp, 232
    - s -p ip, 232
    - s -p tcp, 232
    - s -p udp, 232
  - newaliases, 207
  - nsupdate, 179
  - ntop, 226
  - ping, 142, 169
  - procmail, 209, 210
  - ps, 315
  - rcs, 213
  - route, 67, 141
  - routed, 67, 74, 119
  - sendmail, 195
  - snmpd, 238

- snmpdelta, 238
- snmpget, 238, 242
- snmpgetnext, 238, 242
- snmpset, 238, 243
- snmpstable, 238, 243
- snmptranslate, 238
- snmptrapd, 238
- snmpwalk, 238, 244
- sshd, 140
- syslogd, 154, 315
- tcpdump, 142, 270, 272, 282
- tkined, 225, 247
- traceroute, 169
- Common Address Redundancy Protocol, voir CARP
- Common Management Information Protocol, voir SMIP
- Common Management Information Service, voir CMIS
- Community-based SNMPv2, 226
- commutateur, 20--22
  - ISH, 21
  - VLAN, 22
- Commutation de paquets, 4
- concentrateur, 18--19
- concurrent server, voir serveur concourant
- Configuration and name management, 221
- congestion avoidance, 101
- congestion window, 101
- connect, 259
- CRLF, 191
- CSMA/CA, 9
- CSMA/CD, 9, 19
- Cyclades, 25, 121
- daemon, 144, 315--316
- Darpa, 25
- DATA, 198
- datagramme, 44
- Dave Clark, 89
- David H. Crocker, 191
- descripteur de socket, 253
- DHCP, 109, 177
- diablotin, 318
- Differentiated Services
  - CodePoints, voir DSCP
- Diffie-Hellman, 179
- DNS, 230, voir serveur de noms
- dns dynamique, 167
- DNSBL, voir Domain Name Services BlackList
- DNSSEC, 178
- domain completion, 170
- Domain Name Services
  - BlackList, 204
- domaine, 167
- Douglas E. Comer, 166
- DSCP, 50
- dynamic update, 179
- dyndns, voir dns dynamique
- ECN, 50
- Edsger W. Dijkstra, 73, 112
- EGP, 71, 110, 232
- EINTR, 313
- en-tête
  - 802.2/802.3, 13
  - ARP, 57
  - Ethernet, 11
  - ICMP, 60
  - IGMP, 63
  - IP, 49
  - RARP, 57
  - TCP, 91
  - UDP, 84
- Encapsulating Security Payload, 144
- enveloppe d'un mail, 190
- Ethernet, 9, 9--11
  - adresse, 11
  - adresse d'unicast, 13
  - adresse de broadcast, 13
  - adresse de multicast, 13
  - collision, 10
  - en-tête, 11
  - fin, voir 10Base2
  - format d'une trame, 10
  - paires torsadées, voir 10BaseT
  - RJ45, voir 10BaseT

- standard, voir 10Base5
- thick, voir 10Base5
- thin, voir 10Base2
- transceiver, 9
- Twisted Pair, voir 10BaseT
- Explicit Congestion
  - Notification, voir ECN
- Extensible Markup Language,
  - voir XML
- Exterior Gateway Protocol,
  - voir EGP, voir EGP
- eXternal Data Representation,
  - voir Xdr
- FAI, voir Fournisseur d'Accès Internet
- Fault management, 221
- fcntl, 311
- FD\_CLR, 312
- FD\_ISSET, 312, 313
- FD\_SET, 312
- FD\_ZERO, 312
- fenêtres glissantes, 98
- Fibre optique, 16
- fichier
  - .forward, 209
  - .procmailrc, 209
  - /etc/bootptab, 58
  - /etc/host.conf, 173
  - /etc/hosts, 165, 171, 280
  - /etc/inetd.conf, 321
  - /etc/mail/access, 208
  - /etc/mail/aliases, 207
  - /etc/mail/local-host-names, 208
  - /etc/mail/mailertable, 208
  - /etc/mail/sendmail.cf, 207
  - /etc/mail/submit.cf, 207
  - /etc/mail/userdb, 208
  - /etc/mail/virtusertable, 208
  - /etc/nsswitch, 173
  - /etc/protocols, 50, 122, 140, 255, 291, 322
  - /etc/rc, 315
  - /etc/rc.firewall, 155
  - /etc/resolv.conf, 170, 170--171
  - /etc/services, 28, 85, 195, 258, 267, 275, 321
  - /etc/syslog.conf, 317
  - /var/log/syslog, 317
  - /var/spool/mqueue, 208
  - named.boot, 182, 186
  - named.conf, 182, 186
  - named.root, 175
  - resolv.conf, 173
  - sendmail.cf, 212
  - submit.cf, 212
  - syslog.conf, 318
- FIFO, voir pile FIFO
- Firefox, 26
- Fournisseur d'Accès Internet, 34
- FQDN, voir Fully Qualified Domain Name, 184
- frame, voir trame
- full duplex, 91, 254
- Fully Qualified Domain Name, 169
- gated, voir commande
- gateway, voir passerelle
- generic tunnel interface, 140
- gethostbyaddr, 282
- gethostbyname, 170, 173, 280
- getprotobyname, 291
- getprotobynumber, 291
- getservbyname, 283, 291
- getservbyport, 284, 291
- gif, voir generic tunnel interface
- HELO, 197
- HUB, 16
- hub, voir concentrateur
- IAB, 230
- IANA, 28, 41, 43, 86, 231
- ICANN, voir Internet Corporation for Assigned Names and Numbers

- ICMP
  - Echo Reply, 224
  - Echo Request, 224
- IEEE, 9
- IETF, 40
- IGP, 71, 110
- IMAP, 211--212
- in-addr.arpa, 176
- inaddr.arpa, 184, 175--184
- index\_addr, 279
- inet\_ntoa, 279
- inet\_ntop, 279
- inet\_pton, 279
- Institute of Electrical and Electronics Engineers, voir IEEE
- Interface de loopback, 75
- Interior Gateway Protocol, voir IGP
- Internet Activity Board, voir IAB
- Internet Corporation for Assigned Names and Numbers, 34, 43
- Internet Key Exchange, 144
- Internet Software Consortium, 3, 186
- inverse queries, voir question inverse
- IP, 47
  - adresse, 33--44
    - CIDR, 40--41
    - classe A, 35
    - classe B, 35
    - classe C, 35
    - classe expérimentale, 36
    - de broadcast, 37, 41
    - multicast, 13, 42--44
    - sous-réseaux, 38--39
    - unicast, 36
  - All-subnets-directed broadcast, 41
  - DESTINATION ADDRESS, 50
  - DSCP/ECN, 50
  - FLAG, 54
  - FLAGS, 50, 53
    - Don't Fragment bit, 52
    - More fragment, 53
  - FRAGMENT OFFSET, 50, 53
  - fragmentation, 52, 83
  - HEADER CHECKSUM, 50, 54
  - HLEN, 49
  - ICMP, 30, 59
    - CHECKSUM, 60
    - CODE, 60, 62
    - Destination Unreachable, 61
    - Echo Reply, 61
    - Echo Request, 61
    - Format des messages, 60
    - Redirect, 62, 68
    - router advertisement, 73
    - Router solicitation, 62
    - router sollicitation, 73
    - Source Quench, 50, 62
    - Time exceeded, 62
    - TYPE, 60
  - IDENTIFICATION, 50, 53, 54
  - IGMP, 30, 43, 63
    - protocole, 64
  - Limited broadcast, 41, 114, 120
  - MTU, 47, 47, 49, 51, 52, 83, 99, 139
  - multicast, 114
  - Net-directed broadcast, 41
  - OFFSET, 54
  - OPTIONS, 51
  - PADDING, 51
  - PROTOCOL, 50, 82, 122
  - Réassemblage, 53
  - SOURCE ADDRESS, 50
  - Subnet-directed broadcast, 41
  - TOTAL LENGTH, 49, 53, 54
  - TTL, 50, 62, 65
  - TYPE OF SERVICE, 50
  - VERS, 49
- IP aliasing, voir alias ip
- IP payload compression, 144

- IPFIREWALL, 154
- IPFIREWALL\_VERBOSE, 154
- IPsec, 143--147
  - AH, voir Authentication Header
  - ESP, voir Encapsulating Security Payload
  - IKE, voir Internet Key Exchange
  - IPcomp, voir IP payload compression
  - mode transport, 145
  - mode tunnel, 145
  - SA, voir Security Association
  - SPI, voir Security Parameter Index
- ipv6, 147, 166, 292
- IS-IS, 40, 71
- ISC, voir Internet Software Consortium
- ISN, voir Initial Sequence Number
- ISO 3166, 168
- ISO 8824, 229
- ISO 8825, 231
- iterative server, voir serveur itératif
- jail de FreeBSD, 138
- Jon Postel, 25, 81, 89
- Jonathan B. Postel, 195
- KAME, 147
- Konqueror, 26
- LACNIC, 34
- LAN, 3, 7--8
- LDA, voir Local Delivery Agent
- libc, 170, 173, 186
- Link State Packet, voir LSP
- Link-state request, voir LSR
- Link-state update, voir LSU
- LIR, voir Local Internet Registry
- listen, 263
- little endian, 268, 277
- Local Delivery Agent, 200
- Local Internet Registry, 35
- Louis Pouzin, 25
- LSP, voir routage
- LSR, voir routage
- LSU, voir routage
- machine virtuelle, 138
- MAIL, 198
- Mail Submit Agent, 199
- Mail Transfer Agent, 200
- Mail User Agent, 199
- mailing-list, 189
- MAN, 8
- Management Information Base,
  - voir MIB
- manager, voir SNMP
- master, voir serveur principal
- Maximum Transfer Unit, 47
- Mbone, 65
- md5, 179
- memcmp, 278
- memcpy, 278
- memset, 278
- message, 44
- MIB, 224, 227, 247, 228--247
- MIB-2, 225, 232
- mibs vendor, 228
- mibs vendors, 225
- milter, 209
- milter-greylist, 209
- MIME, 212
- MIMEDefang, 209
- mode
  - connecté, 90, 254, 260, 262
  - datagramme, 83, 90, 254, 260
- mode connecté, 302
- mode datagramme, 303
- Mosaic, 26
- Mozilla, 26
- MSA, voir Mail Submit Agent
- MTA, voir Mail Transfer Agent
- MTU, voir Maximum Transfer Unit

- MUA, voir Mail User Agent  
multi-homed, 44  
multicast, 20  
    224.0.0.1, 42, 64, 73  
    224.0.0.2, 42, 73  
    224.0.0.22, 42  
    224.0.0.255, 65  
    224.0.0.5, 42  
    224.0.0.9, 42  
    adresse MAC, 43  
    groupe, 42  
    IGMP, voir IP IGMP  
mutex, 309
- nœud, 167  
name daemon control program,  
    187  
name server control utility,  
    187  
NAPT, voir Network Address  
    Port Translation  
NAT, voir Network Address  
    Translation  
National Science Foundation,  
    26  
NBO, voir network Byte Order  
NCP, 47  
ndc, voir name daemon control  
    program  
NET-SNMP, 238  
Netscape, 26  
netstat, voir commande  
Network Address Port  
    Translation, 149  
Network Address Translation,  
    149  
network Byte Order, 267  
network byte order, 48, 277,  
    283  
Network Control Protocol, voir  
    NCP  
Network Information Center,  
    166, 168  
Network Management  
    Application, voir NMA  
Network Management Entity,  
    voir NMS  
Network Management System,  
    voir NMS  
NIC, voir Network Information  
    Center  
NIS, 171, 280  
NMA, 222, 247  
NME, 222, 247  
NMS, 222  
nommage absolu, 169  
nommage relatif, 169  
notify, 182  
NSF, voir National Science  
    Foundation  
numéro de port, 81, 252, 267,  
    275  
numéro de service, voir numéro  
    de port
- Objet Identifier, voir OID  
OID, 230--231, 247  
open mail relay, 202  
Open Shortest Path First, voir  
    OSPF  
openlog, 318  
orderly release, 95  
Organizationally Unique  
    Identifier, voir OUI  
OSI, 230  
    7 couches de l', 5  
    application, 5  
    donnée, 9  
    données, 5, 14  
    LLC, 14  
    MAC, 14  
    physique, 5  
    présentation, 5, 226, 229  
    réseau, 5  
    session, 5, 30  
    transport, 5  
OSPF, 40, 50, voir routage  
OUI, 11  
ouverture active, 94  
ouverture passive, 94

- paquet, 44
- passerelle, 8, 22--23, 44
  - routeur, 22
- passive open, voir ouverture passive
- Paul Baran, 4
- PDU, 247
- Performance management, 222
- PF, 254, voir Protocol Family
- pile ARPA, 28, 143, 302
- pile FIFO, 83
- poids faible, 277
- poids fort, 277
- Point to Point Protocol, 55, 210
- poll, 314
- POLLERR, 314
- POLLHUP, 314
- POLLIN, 314
- polling, 312
- POLLOUT, 314
- pont, 19--20
- POP, voir Post Office Protocol, 210--211
- port, voir numéro de port
- Post Office Protocol, 210
- PPP, voir Point to Point Protocol, 118, voir Point to Point Protocol
- primary server, voir serveur principal
- Protocol Data Unit, voir PDU
- querie reverse, voir question inverse
- question inverse, 175
- quintuplet, 90, 255, 258
- QUIT, 198
- répéteur, 17--18
- réseau d'interconnexion, 140
- réseau virtuel, 44
- Réseaux IP européen, 34
- RARP, 12, 58, 60
  - bootp, 58
  - dhcp, 58
- RCPT, 198, 202
- read, 262
- readv, 262
- recv, 262
- recvfrom, 262
- recvmsg, 262
- Regional Internet Registries, 34
- relay mail, 200, 201
- Remote Monitoring, voir RMON
- remote procedure call, 275
- repeater, voir répéteur
- Requests For Comments, 27
- resolver, 170, 170--171, 173, 280
- Resource Record, voir RR, 205
- RFC, voir Requests For Comments
- RFC 1025, 92
- RFC 1028, 224
- RFC 1034, 169, 175
- RFC 1035, 83, 182
- RFC 1042, 9
- RFC 1112, 63
- RFC 1155, 224, 226, 228
- RFC 1156, 224, 226, 232
- RFC 1157, 224, 226
- RFC 1213, 225, 226, 229, 232
- RFC 1611, 245
- RFC 1631, 148
- RFC 1700, 36, 42, 43, 86
- RFC 1878, 38
- RFC 1901, 226
- RFC 1902, 226
- RFC 1903, 226
- RFC 1904, 226
- RFC 1905, 226
- RFC 1906, 226
- RFC 1907, 226
- RFC 1908, 226
- RFC 1918, 34, 140, 149
- RFC 1919, 148
- RFC 1939, 210
- RFC 2136, 179
- RFC 2144, 144



- RFC 2328, 73
- RFC 2364, 139
- RFC 2401, 143, 144
- RFC 2402, 144
- RFC 2405, 144
- RFC 2406, 144
- RFC 2409, 144
- RFC 2451, 144
- RFC 2476, 199
- RFC 2516, 139
- RFC 2535, 179
- RFC 2845, 179
- RFC 2930, 179
- RFC 3168, 50
- RFC 3411, 226
- RFC 3412, 226
- RFC 3413, 226
- RFC 3414, 226
- RFC 3415, 226
- RFC 3416, 226
- RFC 3417, 226
- RFC 3418, 226
- RFC 3501, 211
- RFC 768, 81
- RFC 791, 25, 47
- RFC 793, 89, 265
- RFC 821, 195
- RFC 822, 189, 191
- RFC 826, 55
- RFC 867, 267
- RFC 894, 9
- RFC 896, 100
- RFC 903, 58
- RFC 922, 41
- RFC 950, 38, 59
- RFC 1256, 73
- RFC 1700, 275
- RFC 3493, 257
- RIP, voir routage
- RIP-2, 40
- RIPE, voir Réseaux IP européen
- RIR, voir Regional Internet Registries
- RMON, 225
- rndc, voir name server control
  - utility
- root name server, voir serveur racine
- round trip time, 93, 97
- routage, 66--74
  - algorithme de, 70
  - classless, 40
  - découverte de routeurs, 73
  - direct, 66, 109
  - dynamique, 71, 109
  - indirect, 66, 109
- OSPF, 73, 111, 121--134
  - adjacencies & neighbors, 130
  - adjacency database, 131
  - Area border routers, 129
  - Autonomous system
    - boundary routeurs, 129
  - Backbone routers, 129
  - Backup designated router, 129
  - coût des liens, 127
  - DataBase Description
    - paquet, 131
  - Designated router, 129
  - Down, 130
  - Exchange, 131
  - ExStart, 130
  - flooding, 124, 126
  - Hello, 131
  - hiérarchie de routeurs, 127
  - Init, 130
  - Internal routers, 128
  - link-state database, 124
  - Loading, 131
  - LSAck, 132
  - LSP, 124
  - LSR, 131, 132
  - LSU, 131, 132
  - plus court chemin, 127
  - protocole HELLO, 131
  - Two-way, 130
- redirection, 74
- RIP, 72, 111, 113--120

- chemin le plus court, 113
- métrique, 113
- poisoned reverse, 116
- split horizon, 116
- Triggered updates, 117
- statique, 69, 109
- table de, 67
- routed, voir commande
- Routing Information Protocol, voir RIP
- Routing policy, 110
- RPC, voir remote procedure call
- RR, 182
  - A, 182, 184
  - CNAME, 185
  - HINFO, 185
  - KEY, 179, 185
  - MX, 182, 184, 205
  - NS, 182, 183--184
  - PTR, 182, 184
  - SOA, 167, 175, 182, 183
  - TXT, 185
  - WKS, 185
- RS232, 52
- RTT, voir round trip time
- s-mail, 190
- sans fil, 9, 19
- secondary server, voir serveur secondaire
- Security Association, 145
- Security management, 222
- Security Parameter Index, 145
- select, 312--313
- send, 260--261
- sendmsg, 260--261
- sendto, 260--261
- Serial Line IP, 52
- serveur concourant, 302
- serveur de noms, 280
- serveur de serveurs, voir inetd
- serveur itératif, 301
- serveur principal, 167, 175
- serveur racine, 175
- serveur secondaire, 167, 175
- setsid, 315
- SGMP, 224
- shutdown, 264
- SIGIO, 307, 311
- SIGPOLL, 311
- Simple Gateway Monitoring Protocol, voir SGMP
- Simple Mail Transfer Protocol, 195
- slave, voir serveur secondaire
- sliding windows, voir fenêtres glissantes
- SLIP, voir Serial Line IP
- slow start, 101
- SMI, 125, 226, 228, 247
- SMTP, voir Simple Mail Transfer Protocol
- SNMP, 18, 223--227
  - agent, 223, 225, 234, 247
  - communauté, 235
  - GetBulkRequest, 236
  - GetNextRequest, 236
  - GetRequest, 236
  - GetResponse, 236
  - InformRequest, 236
  - manager, 223, 225, 234, 247
  - PDU, 237
  - PDU(textbf, 235
  - rmon, 225--226, 247
  - SNMPv1, 224, 226
  - SNMPv2c, 226
  - SNMPv3, 226, 237
  - Trap, 236
  - trap, 223, 234, 248
- SO\_LINGER, 96
- socket, 253--255
  - IPPROTO\_ICMP, 255, 291
  - IPPROTO\_IGMP, 255
  - IPPROTO\_IP, 291
  - IPPROTO\_RAW, 255
  - IPPROTO\_TCP, 255, 291
  - IPPROTO\_UDP, 255, 291
  - PF\_APPLETALK, 254
  - PF\_ATM, 254

- PF\_INET, 254
- PF\_INET6, 254
- PF\_IPX, 254
- PF\_ISO, 254
- PF\_KEY, 254
- PF\_LINK, 254
- PF\_LOCAL, 254
- PF\_NS, 254
- PF\_ROUTE, 254
- PF\_SNA, 254
- PF\_UNIX, 254
- SOCK\_DGRAM, 254
- SOCK\_RAW, 254
- SOCK\_STREAM, 254
- source
  - gethostbyname.c, 282
- spam, 201--204
- Spanning Tree Protocol, 20
- static nat, 152
- STP, voir Spanning Tree Protocol
- Structure of Management Information, voir SMI
- structure sockaddr, 257
- structure sockaddr\_in, 258
- subnet address, voir Adresse de sous-réseau
- supernet, 40
- switch, voir commutateur
- syslog, 187, 223, voir commande syslogd, 318
- syslogd, 316--317
- table de routage, voir routage
- TCP, 89
  - ACKNOWLEDGEMENT NUMBER, 91
  - CHECKSUM, 92
  - CODE, 92
    - ACK, 91, 92
    - ACNOWLEDGMENT NUMBER, 92
    - FIN, 92
    - PUSH, 92
    - RST, 92, 264
    - SYN, 92, 99
    - URGENT POINTER, 92
  - DESTINATION PORT, 91
  - Initial Sequence Number, 91
  - OFFSET, 91
  - OPTIONS, 92
    - mss, 93, 99
    - nop, 93
    - timestamp, 93
  - PADDING, 93
  - RESERVED, 92
  - SEQUENCE NUMBER, 91, 95
  - SOURCE PORT, 91
  - URGENT POINTER, 92
  - WINDOW, 92, 99
- the internet superserver, voir inetd
- threads kernel, 310
- threads user land, 310
- three-way handshake, 95
- time sharing, 307
- TKERY, voir Transaction Key
- TKEY, 178
- TLD, voir top levels domains
- TLI, voir Transport Layer Interface
- top levels domains, 168
- trame, 44
- Transaction Key, 179
- Transaction SIGNature, 179
- transfert de zone, 182
- Transport Layer Interface, 251
- trap, voir SNMP
- TSIG, 178, voir Transaction SIGNature
- Tunnel IP, 139
- UDP, 81, 118
  - CHECKSUM, 84
  - DESTINATION PORT, 84
  - MESSAGE LENGTH, 84
  - SOURCE PORT, 84
- umask, 315
- Université de Berkeley, 26
- UUCP, 195
- Variable Length Subnet Mask, voir VLSM
- vecteur de distances, 111

Virtual Private Network, 143  
VLSM, 122  
VPN, voir Virtual Private  
Network, 145

W. Richard Stevens, 100  
WAN, 8  
wifi, voir sans fil  
wireless, voir sans fil  
write, 260--261  
writev, 260--261  
wscale, 92, 93

XdR, 5  
XML, 5

zone, 167  
zone de Solaris, 138  
zone reverse, 176

# Annexe A

## Programme serv2prot.c

```
1  /* $Id: serv2prot.c 134 2009-02-27 16:38:44Z fla $
2  *
3  * Exemple de serveur d'écho, parallèle et multiprotocole (TCP, UDP).
4  *
5  * Compiler avec -DBAVARD pour des commentaires sur les états du serveur.
6  * Le serveur se stoppe avec un SIGHUP.
7  *
8  * Compiler avec -DBSD sur les OS BSD.
9  *
10 */
11 #include <stdio.h>          /* De toute façon !      */
12 #include <errno.h>          /* Idem                  */
13 #include <stdlib.h>         /* Pour "atoi".         */
14 #include <unistd.h>         /* Pour "getopt".       */
15 #include <strings.h>        /* Pour "bcopy".        */
16 #include <signal.h>         /* Pour "signal".       */
17 #include <time.h>           /* Pour "select".       */
18 #include <sys/exits.h>      /* Codes de retour      */
19 #include <sys/wait.h>       /* Pour "wait".         */
20 #include <sys/types.h>
21 #include <sys/socket.h>
22 #include <netinet/in.h>
23 #include <arpa/inet.h>      /* Pour "inet_ntoa"     */
24 #include <netdb.h>          /* Pour "getservbyname" */
25
26 extern      char    *optarg ;
27 extern      int     optind, opterr ;
28
29 int         OuvrirSocketUDP(char *,int) ;
30 int         OuvrirSocketTCP(char *, int, int) ;
31 int         PortParLeNom(char *,char *) ;
32 void        TraiterTCP(int) ;
33 void        TraiterUDP(int) ;
34 void        PasDeZombi(int) ;
35 void        FinCanonique(int) ;
36
37 #define max(a,b)          (a > b ? a : b)
38 #define USAGE             "Usage:%s -p <numéro de port> [-n|-w]\n\t-n : serveur
    parallèle\n\t-w : serveur itératif (défaut)\n"
```

```

39 #define VRAI      (1)
40 #define FAUX      (0)
41 #define MAXQ      5
42
43 #ifdef BSD
44 #define CAST      fd_set *
45 #define CAST2     struct rusage *
46 #else
47 #define CAST      int *
48 #define CAST2     int *
49 #endif
50
51 #ifdef BAVARD
52 #define PRINTF     (void)printf
53 #else
54 #define PRINTF     if (0) (void)printf
55 #endif
56
57 int      sudp, stcp ;           /* Descrip. de socket. */
58 pid_t    tcp_pid, udp_pid ;    /* ceux des fils. */
59 int      iteratif ;           /* VRAI:serv. itératif. */
60 fd_set   lect, alire ;        /* Pour le 'select'. */
61 struct sockaddr_in sclient ;   /* Pour le getpeername */
62
63 int
64 main(int argc, char *argv[])
65 {
66     int c ;                    /* Brouillon */
67     int nport ;                /* N°port du serveur. */
68     int ndes ;                 /* Retour de 'select'. */
69     int sock ;                 /* Pour le 'accept'. */
70     int intcp
71     pid_t pid ;                /* Pour le "fork". */
72     socklen_t slen ;
73
74     nport = -1 ;               /* Non configuré. */
75     iteratif = VRAI ;          /* Itératif par défaut. */
76     opterr = 0 ;               /* cf "man 3 getopt" */
77     while ((c=getopt(argc, argv, "p:nw")) != EOF) {
78         switch (c) {
79             case 'p' :         /* Numéro du port. */
80                 nport = atoi(optarg) ;
81                 break ;
82             case 'w' :         /* 'wait' = Itératif. */
83                 iteratif = VRAI ;
84                 break ;
85             case 'n' :         /* 'nowait' = Parallèle.*/
86                 iteratif = FAUX ;
87                 break ;
88             default :           /* Erreur !! */
89                 (void)fprintf(stderr, USAGE, argv[0]) ;
90                 exit(EX_USAGE) ;
91         }
92     }

```

```

93     if (nport < 0) {
94         (void)fprintf(stderr,USAGE,argv[0]) ;
95         exit(EX_USAGE) ;
96     }
97     if (iteratif==VRAI)
98         PRINTF("***Début du server itératif\n") ;
99     else
100         PRINTF("***Début du server parallèle\n") ;
101
102     sudp = OuvrirSocketUDP("", nport) ;
103     stcp = OuvrirSocketTCP("", nport, MAXQ) ;
104     c     = max(sudp,stcp) + 1 ;          /* Rang bit + à gauche. */
105
106     FD_ZERO(&lect) ;                      /* cf "<sys/types.h>" */
107     FD_SET(sudp,&lect) ;
108     FD_SET(stcp,&lect) ;
109
110     (void) signal(SIGCHLD, PasDeZombi) ; /* Mort d'un proc. fils */
111     (void) signal(SIGHUP, FinCanonique) ; /* Fin "propre". */
112
113     while (VRAI) {                        /* Tourne toujours ! */
114         alire = lect ;
115         PRINTF("***Lecture bloquante du 'select'\n") ;
116         if ((ndes = select(c,(CAST)&alire,(CAST)0,(CAST)0,\
117                             (struct timeval *)0)) <
118                             0) {
119             if (errno == EINTR)           /* A cause d'une inter. */
120                 continue ;
121             perror("select") ;
122             exit(EX_OSERR) ;
123         }
124         while (ndes) {                    /* ndes >= 0 */
125             if (FD_ISSET(stcp,&alire)) {
126                 PRINTF("***Sélection de l'entrée TCP\n") ;
127                 if ((sock=accept(stcp,(struct sockaddr *)0,(socklen_t
128                                     *)0))<0) {
129                     perror("accept") ;
130                     exit(EX_OSERR) ;
131                 }
132                 intcp = VRAI ;
133             }
134             else if (FD_ISSET(sudp,&alire)) {
135                 PRINTF("***Sélection de l'entrée UDP\n") ;
136                 sock = sudp ;
137                 intcp = FAUX ;
138             }
139             if (getpeername(sock,(struct sockaddr *)&sclient,&slen) < 0)
140                 perror("Getpeername") ;
141             else
142                 PRINTF("***Nouveau client : %s:%d\n",\
143                         inet_ntoa(sclient.sin_addr),
144                         ntohs(sclient.sin_port)) ;
145         }
146     }
147     retry:

```

```

144         pid=fork() ;
145         switch (pid) {
146             case -1 :                /* Erreur.          */
147                 if (errno==EINTR) goto retry ;
148                 perror("fork") ;
149                 exit(EX_OSERR) ;
150             case 0 :                /* Fils.          */
151                 if (intcp==VRAI) {
152                     (void)close(sudp) ;
153                     (void)close(stcp) ;
154                     TraiterTCP(sock) ;
155                 }
156                 else {
157                     (void)close(stcp) ;
158                     TraiterUDP(sock) ;
159                 }
160                 exit(EX_OK) ;
161             default :                /* Père.          */
162                 if (iteratif==VRAI)
163                     FD_CLR(intcp==VRAI?stcp:sudp,&lect) ;
164                 if (intcp==VRAI) {
165                     (void)close(sock) ;
166                     FD_CLR(stcp,&alire) ;
167                     tcp_pid = pid ;
168                 }
169                 else {
170                     udp_pid = pid ;
171                     FD_CLR(sudp,&alire) ;
172                 }
173             }
174         ndes— ;
175     }
176 }
177 }
178
179 /*
180  * PasDeZombi : Gestion des processus fils qui se terminent.
181  */
182 void
183 PasDeZombi(int nsig)
184 {
185     pid_t pid ;
186     int status ;
187
188     PRINTF("***On a reçu un SIGCHLD\n") ;
189     /*
190      * WNOHANG : évite que wait3 soit bloquant, même si des fils sont
191      *          encore en activité (retour 0).
192      */
193     while ((pid=wait3 (&status, WNOHANG, (CAST2)0)) > 0)
194         if (iteratif==VRAI)
195             if (pid==tcp_pid) {
196                 FD_SET(stcp,&lect) ;
197                 PRINTF ("***L'entrée TCP est réactivée\n") ;

```



---

```

198             break ;
199         }
200         else if (pid==udp_pid) {
201             FD_SET(sudp,&lect) ;
202             PRINTF("***L'entrée UDP est réactivée\n") ;
203             break ;
204         }
205 #ifndef BSD
206     (void)signal(SIGCHLD, PasDeZombi) ;           /* Selon OS      */
207 #endif
208 }
209
210 /*
211  * FinCanonique : On passe par là en cas de fin normale.
212  */
213 void
214 FinCanonique(int nsig)
215 {
216     PRINTF("***Signal SIGHUP reçu - Fin du serveur !\n") ;
217     exit(EX_OK) ;
218 }
219
220 /*
221  * Serveur d'écho, TCP.
222  */
223 void
224 TraiterTCP(int des)
225 {
226     int n ;
227     char buf[1024] ;
228
229     PRINTF("***On entre dans TraiterTCP, chaîne lue :\n");
230
231     while ((n = read(des,buf,sizeof buf)) > 0) { /* == 0 -> EOF */
232         buf[n] = '\0' ;
233         PRINTF("***On renvoie (TCP) %s",buf) ;
234         if (write (des,buf,n) < 0)
235             perror ("write - TCP") ;
236     }
237     PRINTF("\n") ;
238     PRINTF("*** Déconnexion de %s:%d\n",inet_ntoa(sclient.sin_addr),\
239                                           ntohs(sclient.sin_port))
240 }
241
242 /*
243  * Serveur d'écho, UDP.
244  */
245 void
246 TraiterUDP(int des)
247 {
248     char buf[BUFSIZ] ;
249     int n ;
250     socklen_t laddr ;

```

```

251     struct sockaddr adr ;
252
253     PRINTF("***On entre dans TraiterUDP, chaîne lue :\n");
254     laddr = sizeof adr ;
255     if ((n = recvfrom(des,buf,sizeof buf,0,(struct sockaddr
        *)&adr,&laddr)) < 0)
256         return ;
257     buf[n] = '\0' ;
258     PRINTF("***On renvoie (UDP) :%s\n",buf) ;
259     (void)sendto(des,buf,n,0,(struct sockaddr *)&adr,laddr) ;
260 }
261
262 /*
263  * OuvrirSocketUDP : Ouvre une socket UDP et renvoie le descripteur
264  */
265 int
266 OuvrirSocketUDP(char *nserv,int nport)
267 {
268     struct sockaddr_in saddr ;
269     int sd ;
270     int np = htons(nport) ;
271
272     if (np < 0)
273         if ((np = PortParLeNom(nserv,"udp")) < 0)
274             return -1 ;
275     if ((sd=socket(PF_INET, SOCK_DGRAM, 0)) < 0) {
276         perror("socket - SOCK_DGRAM") ;
277         exit(EX_OSERR) ;
278     }
279
280     bzero((char*)&saddr, sizeof saddr) ;
281     saddr.sin_family = PF_INET ;
282     saddr.sin_port = np ;
283
284     if (bind(sd,(struct sockaddr *)&saddr,sizeof saddr) < 0) {
285         perror("bind - SOCK_DGRAM") ;
286         exit(EX_OSERR) ;
287     }
288     return sd ;
289 }
290
291 /*
292  * OuvrirSocketTCP : Ouvre une socket TCP et renvoie le descripteur
293  */
294 int
295 OuvrirSocketTCP(char *nserv,int nport,int queue)
296 {
297     struct sockaddr_in saddr ;
298     int sd ;
299     int np = htons(nport) ;
300
301     if (np < 0)
302         if ((np=PortParLeNom(nserv,"tcp")) < 0)
303             return -1 ;

```

---

```
304     if ((sd = socket (PF_INET, SOCK_STREAM, 0)) < 0) {
305         perror("socket - SOCK_STREAM") ;
306         exit(EX_OSERR) ;
307     }
308     bzero ((char*)&sadr, sizeof sadr) ;
309     sadr.sin_family = PF_INET ;
310     sadr.sin_port   = np ;
311
312     if (bind(sd,(struct sockaddr *)&sadr,sizeof sadr) < 0) {
313         perror("bind - SOCK_STREAM") ;
314         exit(EX_OSERR) ;
315     }
316     if (listen(sd,queue) < 0) {
317         perror("listen - SOCK_STREAM") ;
318         exit(EX_OSERR) ;
319     }
320     return sd ;
321 }
322
323 int
324 PortParLeNom(char *nserv, char *nprot)
325 {
326     struct servent      *serv ;
327
328     if (getservbyname (nserv,nprot) == NULL)
329         return -1 ;
330
331     return serv->s_port ;           /* Respecte le NBO.      */
332 }
```