

Utilisation du réseau TCP/IP

Michel Hassenforder

- Bibliographie
- Généralités
- Topologie
- Protocoles de base
- API Socket
- Protocoles de services
- Protocoles de routage
- Protocoles applications
- Protocoles avancés
- La multidiffusion

Internetworking with TCP/IP - Volume 1 - Principles, protocols, and Architectures

Douglas E. COMER

Prentice Hall, 4 ème édition.

Internetworking with TCP/IP - Volume 3 - Windows Sockets Version - Client-Server
Programming and Applications

Douglas E. COMER, David L. STEVENS

Prentice Hall.

Internetworking with TCP/IP - Volume 2 - Design, Implementation and Internal
Douglas E. COMER
Prentice Hall.

TCP/IP JumpStart - Internet Protocol Basics
Andrew G. Blank
Sybex.

Illustrated TCP/IP

Matthew G. Naugle

Wiley Computer Publishing, John Wiley & Sons, Inc.

OSPF Design Guide

Sam Halabi

Cisco Systems

Les réseaux TCP/IP

Laurent Toutain

ENST bretagne

Internet Core Protocols : The definitive Guide (chapitre 2)

Eric Hall

O'Reilly

Network security with Open SSL (chapitre 1)

John Viega, Matt Messier, Pravir Chandra

O'Reilly

<http://www.infres.enst.fr/~dax> (multicast - Philippe Dax)

<http://www.commentcamarche.net> (divers)

<http://www.hsc.fr> (ipsec)

<http://www.univ-valenciennes.fr> (Mbone - Guy Bisiaux)

<http://www.ietf.org> (les RFCxxxx.txt sur le net)

<http://www.transtec.fr> (hardware)

<http://www.cisco.com> (VPN)

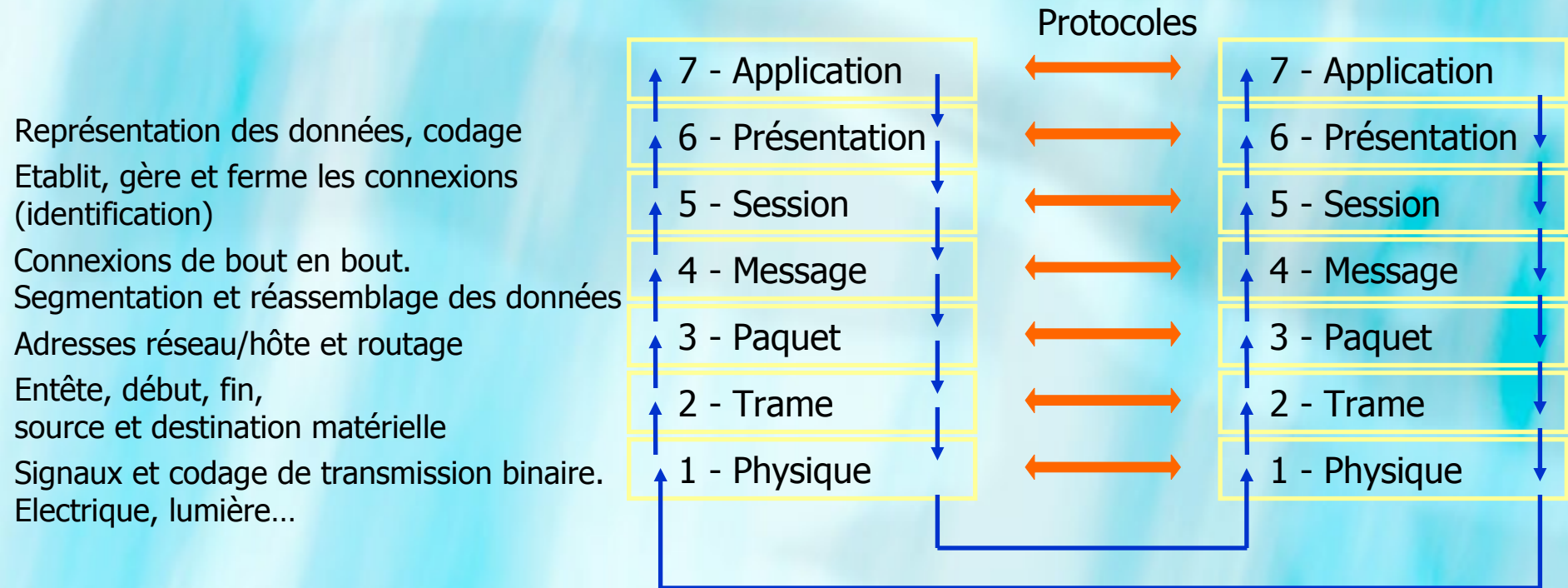
<http://www.urec.cnrs.fr> (routage - Jean Paul Gautier)

<http://www.urec.cnrs.fr> (ipv6, multicast, routage - Bernard Tuy)

<http://grove.circa.ufl.edu/> (XDR)

<http://abcdrfc.free.fr> (traduction française des rfc)

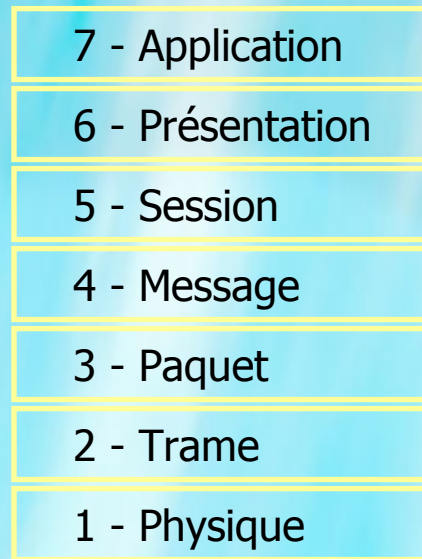
Des pages de Bernard Tuy, André Aoun, Chunlei liu, Laurent Mirtain, Jean Luc Archimbeau, Valéry Frémaux, David Mills



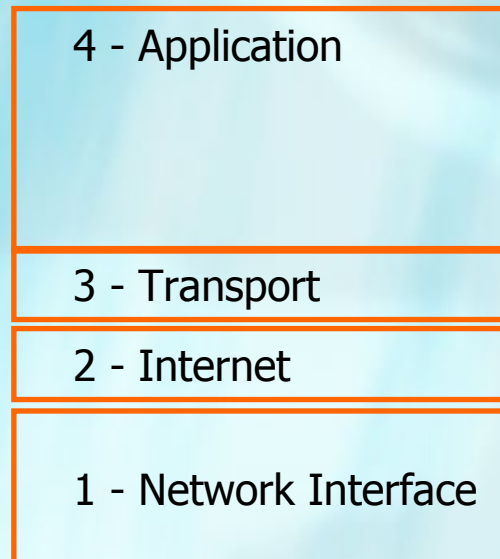
Remarque : un protocole n'est pas obligatoirement lié avec la pile de protocoles des niveaux inférieurs.

Le même protocole de présentation peut être utilisé dans le cadre de plusieurs protocoles de transport.

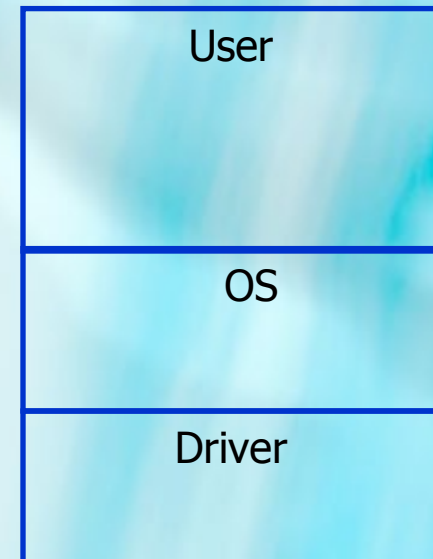
Model OSI



Model TCP/IP



Model OS



Je vais montrer les protocoles qui font partis de la pile TCP/IP
Il y a beaucoup d'autres protocoles (Ethertalk, IPX, ATM, etc.)
Ils peuvent même partager des propriétés (protocoles) avec TCP/IP

- Niveau physique définit
 - Le média
 - Câble coaxial gros / fin (10Base15, 10Base2, etc...)
 - Paires torsadées (100BaseT, 1000BaseT, UTP, FTP, etc...)
 - Fibres optiques (FOIRL, ...)
 - Canal Hertzien
 - Etc..
 - Les signaux électriques
 - Niveau
 - Codage
 - Taux d'erreur
 - La politique d'accès au média (CSMA/CD, etc...)
 - La connectique
- Ce niveau assure la connexion entre deux appareils

- Niveau trame encapsule les données
 - Par de la signalisation de synchronisation
 - Par les adresses : émetteur et récepteur
 - Par une mise en conformité des données
 - Par une insertion de code de détection/correction d'erreurs
- Ce niveau assure la communication entre deux dispositifs

- Niveau paquet définit
 - Les solutions adoptées pour interconnecter
 - des machines proches
 - des machines éloignées (nœuds intermédiaires)
 - L'acheminement d'un flux
 - Fragmentation des données
 - Assemblage ordonné des données
 - La régulation du débit
- Ce niveau assure l'acheminement des données entre deux machines

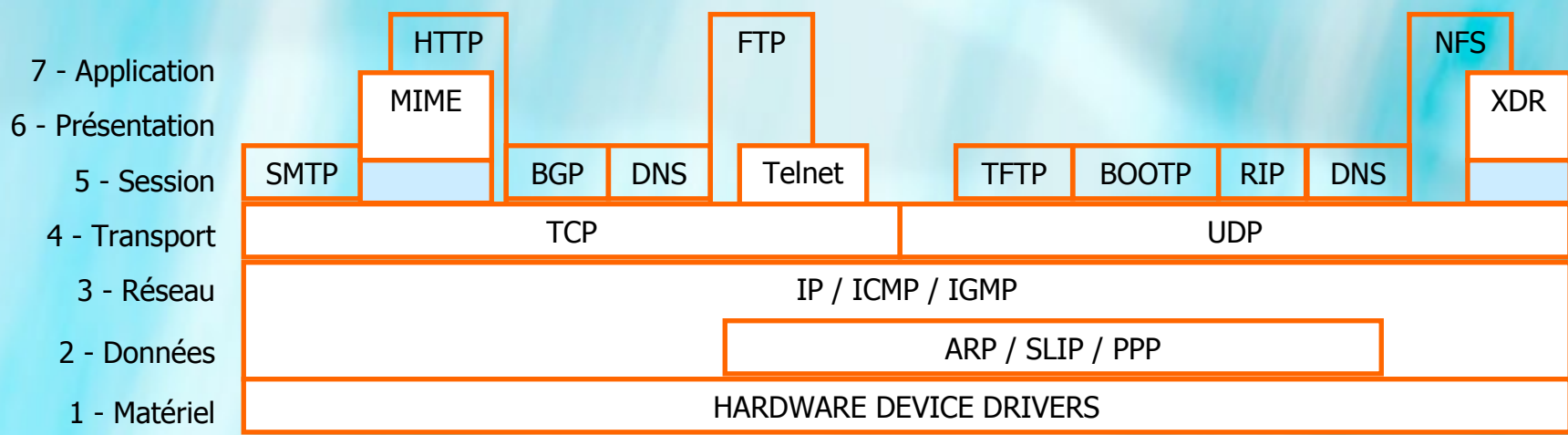
- Niveau message définit
 - Le transport des données utilisateur entre les machines concernées
 - Deux grandes solutions
 - rapide - facile - taille limitée - pas fiable
 - lente - lourde - taille illimitée - fiable
 - ... mais d'autres types de transport arrivent
- Ce niveau assure le transport de bout en bout

- Niveau session définit
 - L'ouverture de session
 - La fermeture de session
 - La suspension de session
 - La reprise de session
 - Etc...
 - Une session étant une collection de messages qui sera échangée entre les machines concernées
- Ce niveau assure une cohérence à un ensemble de messages

- Niveau présentation définit
- Le codage des données
 - Les types de base (entier, flottant, etc...)
 - Les types tableaux (Array, String, ...)
 - Les types composés (structure)
- Problème
 - Manipulation de la valeur entière : 0x0A 0x0B 0x0C 0x0D
 - Motorola utilise le format big endian pour coder les entiers
 - Dans les adresses successives : 0x0A 0x0B 0x0C 0x0D
 - Intel le format little endian pour coder les entiers
 - Dans les adresses successives : 0x0D 0x0C 0x0B 0x0A
- Ce niveau assure l'indépendance des données échangées par rapport aux architectures supports.

- Niveau application définit
- C'est le programme utilisateur qui peut être découpé en familles ... mais est-ce bien raisonnable
 - MHS : Message Handling System (mail, news, etc...)
 - DS : Directory service (DNS, LDAP, ...)
 - FTAM : File Transfer, Access and Management (FTP, ...)
 - VT : Virtual Terminal (telnet, ...)
 - JTM : Job Transfer and Manipulation (shell, SSH, ...)
 - ...

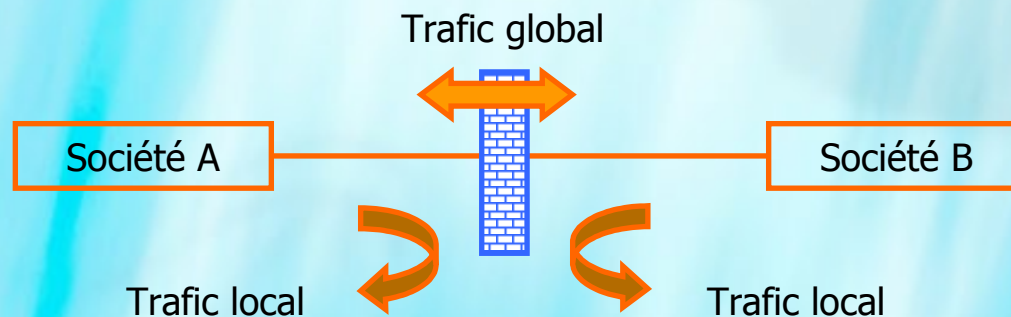
- Le fonctionnement de l'internet est dicté par les RFC
 - RFC = Request For Comment
- N'importe qui peut écrire un RFC (RFC 1543)
 - En anglais, au format texte, avec une mise en page bien définie (RFC 1543)
- Après des grandes discussions des experts et un accord sur le contenu, il y a attribution d'un numéro
- Par la suite, ce document fait référence et ne peut plus être modifié (il n'y a pas de version du RFC)
- Les modifications sont possibles par un nouvel RFC qui ajoute, corrige ou remplace un RFC précédent
- Quand c'est une norme bien établie un RFC peut devenir un STD (Standard Track)
- Les STD ne sont presque jamais référencés, c'est toujours les RFC qui sont utilisés



- Je propose de découper les protocoles en familles
- Protocoles de base (emballage des données)
 - ARP, IP, TCP, UDP, ...
- Protocoles de services (gestion des adresses IP)
 - DNS, BOOTP/DHCP, NTP, ...
- Protocoles de routage (acheminement des données)
 - RIP, OSPF, IGP, EGP, ...
- Protocoles de haut niveau (transfert avancé de données)
 - FTP, SMTP, HTTP, LDAP, ...
- Protocoles liés à la multi diffusion (multi producteur/consommateur)
 - RSVP, DVMRP, PIM, MOSPF, ...

- Répéteur
- HUB, Fanout, concentrateur
- Pont, Bridage
- Routeur
- Switch
- Passerelle
- Class A, B, C, D, E, /n
- Configuration carte réseau

- Ethernet marche par inondation
 - Toutes les machines du même réseau reçoivent tous les paquets.
 - Chaque machine réceptionne tous les paquets : les siens, les autres et même les erreurs.
- Conséquences :
 - Sur une machine, la première opération consiste à filtrer les paquets qui lui sont adressés, les autres paquets sont rejetés.
 - Il faut des dispositifs pour interconnecter les machines et les réseaux.
 - Pour éviter la saturation des réseaux, il faut pouvoir isoler les réseaux.



- **Concentrateur, HUB, fanout**
 - Opère au niveau de la couche 1 (matériel)
 - Permet d'interconnecter entre elles plusieurs machines (4-8-12-16-24-32-48 ports)
 - Permet de changer le support physique (10BT, coaxial, optique, etc.)
- **Répéteur**
 - Opère au niveau de la couche 1 (matériel)
 - Permet de régénérer les signaux
 - Permet de changer le support physique (10BT, coaxial, optique, etc.)
- **Pont, pont filtrant, bridge**
 - Opère au niveau de la couche 2 (données)
 - Permet de changer de format de trames (Adaptation en longueur, organisation, etc.)
 - Le pont filtrant permet d'isoler le trafic interne des différents segments
- **Commutateur/switch**
 - Opère au niveau de la couche 2 (données)
 - Le commutateur permet d'aiguiller les trames sur les bons ports (isolement comme le pont)
- **Routeur/router**
 - Opère au niveau de la couche 3 (réseau)
 - Calcule le meilleur chemin pour faire transiter des trames entre routeurs
 - Réalise le travail d'un switch avec des tables d'autorisations/interdictions en plus.
- **Pare-feu/firewall**
 - Opère au niveau des couches 3, 4 voire 7
 - Autorise ou Interdit le transit de trames entre réseau
 - Réalise un travail comparable au switch mais avec l'intelligence d'un routeur (autorisations/interdictions)
- **Passerelle/gateway**
 - Permet de traduire des trames applicatives d'un format vers un autre (couche 7) ex Appletalk->Ethernet



- Adresse MAC : 48 bits
 - fixée par la carte ethernet
 - Fixée par l'utilisateur
- Exemple : 00:A0:C9:0F:92:A5
 - Le premier bit à 1 signifie adresse multiple (multicast), à 0 adresse unique (standard)
 - Les 3 premiers octets identifient le **fabricant**
 - Les 3 derniers octets identifient le **numéro de série du coupleur**
- Broadcast : FF:FF:FF:FF:FF:FF
- La carte écoute les paquets pour son adresse et le broadcast.
- Rappel : un routeur ne route pas un broadcast à travers ses réseaux.
- Défaut : cette adresse ne permet pas de localiser la machine alors que 33(0)389 33 6970 le permet (France/est/haut rhin/uha/ensisa/lsi)

- Adresse Internet IPv4 : 32 bits (4 milliards) fixée au démarrage
 - Manuel et fixe par l'administrateur de la machine
 - Automatique et fixe par l'administrateur de la machine à l'aide du protocole BOOTP
 - Automatique et dynamique à l'aide du protocole DHCP
 - Exemple : 10.66.13.2
 - Avec des milliers d'ordinateurs, il faut une logique d'attribution des numéros IP (RFC 791)
- Au début de l'internet cela semblait largement suffisant.
 - Année 70
 - Usage militaire
- Maintenant avec des millions d'ordinateurs (voire des milliards)
 - Il faut faire une évolution majeure : IPv6 (128 bits)

- Plusieurs solutions pour segmenter l'internet:
- Segmentation en N réseau de M machines (N et M fixes)
 - Problème des petits réseaux qui n'utilisent pas toutes les adresses
 - Problème des grand réseaux qui ont besoin de beaucoup d'adresses
- Segmentation en 5 classes (A, B, C, D, E) (class full)
 - Selon la valeur de bits particuliers
 - il y a un nombre de réseaux plus ou moins important
 - pouvant contenir un nombre plus ou moins élevé de machines
- Segmentation dynamique en réseau /n (slash n) (classless)
 - Une logique simple (masque de bits) permet de gérer la taille des réseaux et leurs étendues.

Classe A 0xxx xxxx 0xxx xxxx 0xxx xxxx 0xxx xxxx

Classe B 10xx xxxx 0xxx xxxx 0xxx xxxx 0xxx xxxx

Classe C 110x xxxx 0xxx xxxx 0xxx xxxx 0xxx xxxx

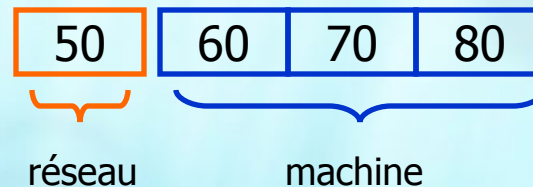
Classe D 1110 xxxx 0xxx xxxx 0xxx xxxx 0xxx xxxx

Classe E 1111 xxxx 0xxx xxxx 0xxx xxxx 0xxx xxxx

En rouge : le numéro de réseau

En jaune : le numéro de machine

- Le premier MSB est à 0
- Le numéro de réseau est codé dans l'octet le plus à gauche.
 - Il y a 127 réseaux différents identifiés par l'octet gauche qui va de la valeur : 1.m.m.m à 127.m.m.m
- Le numéro de machine occupe les trois octets restants.
 - Ce qui autorise 16M machines.
- Exemple :



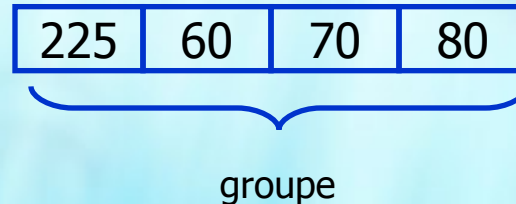
- Les deux premiers MSB sont à 10
- Le numéro de réseau est codé dans les deux octets les plus à gauche.
 - Il y a 16K réseaux différents identifiés par l'octet gauche qui va de la valeur 128.r.m.m à 191.r.m.m
- Le numéro de machine occupe les deux octets restants.
 - Ce qui autorise 64K machines.
- Exemple :



- Les trois premiers MSB sont à 110
- Le numéro de réseau est codé dans les trois octets les plus à gauche.
 - Il y a 2M réseaux différents identifiés par l'octet gauche qui va de la valeur 192.r.r.m à 223.r.r.m.
- Le numéro de machine occupe l'octet le plus à droite.
 - Ce qui autorise 256 machines.
- Exemple :



- Définit dans le RFC 1700, elle permet de faire de la multidiffusion (multicast)
- Les quatre premiers MSB 1110
 - Il y a 1 seul ensemble de groupes identifiés par l'octet gauche qui va de la valeur 224.g.g.g à 240.g.g.g.
- Le numéro de groupe est codé dans les 4 octets (28 bits)
 - Il y a 268M de groupes différents
- Exemple :



- Les quatre premiers MSB sont a 1111
- Ce sont des adresses réservés (Expérimental)
- Elles ne doivent pas exister sur le réseau public
- Exemple :

254	60	70	80
-----	----	----	----

- Les adresses particulières
 - Le loopback (localhost dans le DNS) : 127.0.0.1
 - Le numéro réseau est à 0 : le réseau courant
 - 0.0.0.15 signifie la machine 193.5.5.15 si on est dans le réseau 193.5.5.0
 - Le numéro machine est à 0 : la machine courante
 - 193.5.5.0 signifie la machine locale par exemple 193.5.5.125.
 - Tous les bits du numéro machine est à 1 : toutes les machines dans le réseau courant
 - 193.5.5.255 toutes les machines dans le réseau 193.5.5.0
- Les adresses privées : dans chaque classe il y a des adresses non routables (elles ne vont pas dans le réseau public mais restent dans le réseau local)
 - Classe A : 10.x.x.x soit 1 réseau.
 - Classe B : 172.16.x.x à 172.32.x.x soit 16 réseaux.
 - Classe C : 192.168.0.x à 192.168.255.x soit 256 réseaux.
 - Ex : mon routeur ADSL -> 192.168.1.2

- Exemple :
 - une société possède 150 filiales à interconnecter,
 - dans chaque filiale il y a 100 machines.
- Solutions : 150 classes C ou 1 classe B
- Problèmes :
 - 150 classes C, faut pas rêver, on ne les reçoit pas.
 - 1 classe B, on peut la recevoir, mais
 - l'activité réseau de chaque filiale est reportée sur les autres filiales
 - ou alors il faut des pont filtrant ou des routeurs...
- Les routeurs routent des réseaux...
- Comment découper un réseau trop grand en plusieurs réseaux plus petits.
- Proposition : La notion de sous réseau
 - Le classe B est découpé en sous réseaux interconnectés avec des routeurs locaux
 - L'interconnexion et l'isolation du trafic sont en place
 - Chaque filiale va recevoir un morceau "interne" de la classe B
- Comment faire cette découpe le plus simplement possible : netmask

- Le netmask (RFC 950) permet d'ajouter une notion supplémentaire au réseau et à la machine : le sous réseau.
- Avec un masque de réseau, il est possible de découper une adresse en trois parties : le réseau, le sous réseau et la machine.
- Le réseau est défini par les bits correspondant à la classe du réseau
- Dans le masque de réseau :
 - les bits à 0 identifient la partie machine
 - les bits à 1 qui ne sont pas ceux du réseau identifient la partie sous-réseau

Numéro	10	51	12	1
Masque	255	255	0	0
	Réseau	Sous-réseau	Machine	

- Selon la classe de l'adresse, le masque est standard ou non (sous réseau)
- Masques de réseau standard :
 - classe A 255.0.0.0
 - classe B 255.255.0.0
 - classe C 255.255.255.0
- Masques de réseau non standard :
 - classe A 255.255.0.0 8 bits pour les sous réseaux
 - classe A 255.255. 255.0 16 bits pour les sous réseaux
 - classe B 255.255.240.0 4 bits pour les sous réseaux
 - classe C 255.255.255.192 2 bits pour les sous réseaux
 - classe C 255.255.255.128 1 bits pour le sous réseau

- Un classe A découpé en 256 sous réseaux de 65534 machines... = UHA
 - numéro = 10.59.12.2 avec le netmask = 255.255.0.0
 - net = 10.0.0.0, subnet = 59 et machine = 12.2 (12=mon bureau, 2=2ème ordinateur du bureau)
- Un classe B découpé en 256 réseaux de 256 machines
 - numéro = 161.162.163.14 avec le netmask = 255.255.255.0
 - net = 161.162.0.0, subnet = 163 et machine = 14
- Un classe B découpé en 16 réseaux de 4096 machines
 - numéro = 161.162.163.14 avec le netmask = 255.255.240.0
 - net = 161.162.0.0, subnet = 10 et machine = 3.14
- Un classe C découpé en 4 sous-réseaux de 64 machines
 - numéro = 193.1.2.4 avec le netmask = 255.255.255.192
 - net = 193.1.2.0, subnet = 0 et machine = 4
 - numéro = 193.1.2.69 avec le netmask = 255.255.255.192
 - net = 193.1.2.0, subnet = 1 et machine = 5
 - numéro = 193.1.2.193 avec le netmask = 255.255.255.192
 - net = 193.1.2.0, subnet = 3 et machine = 1

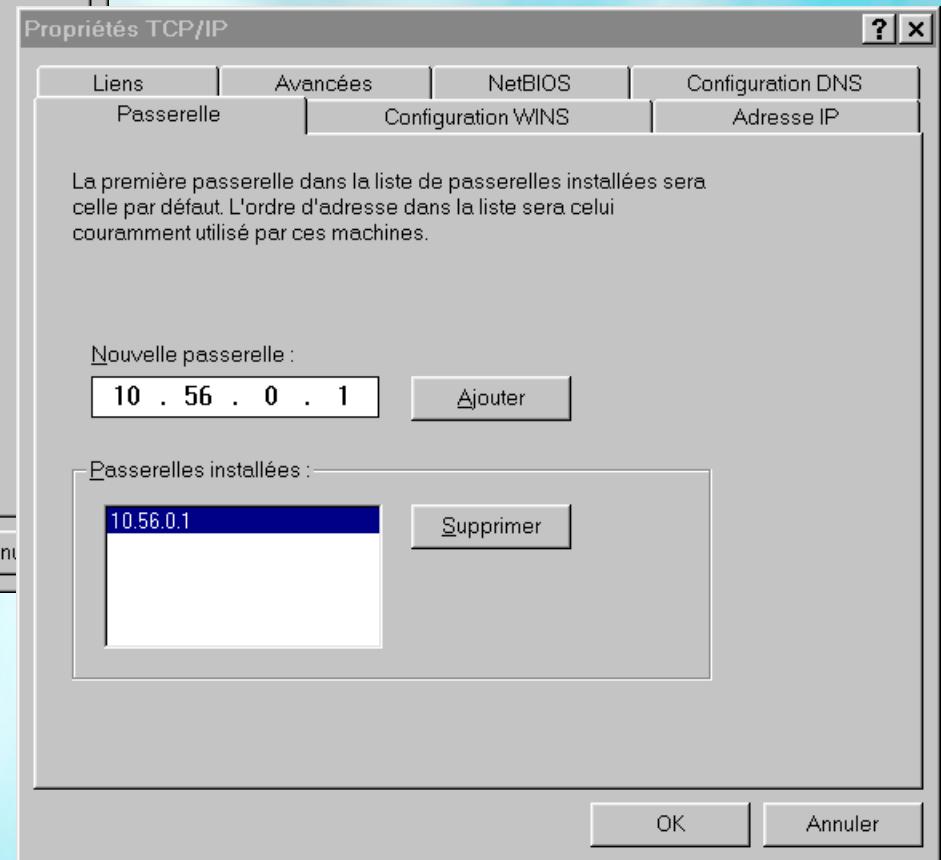
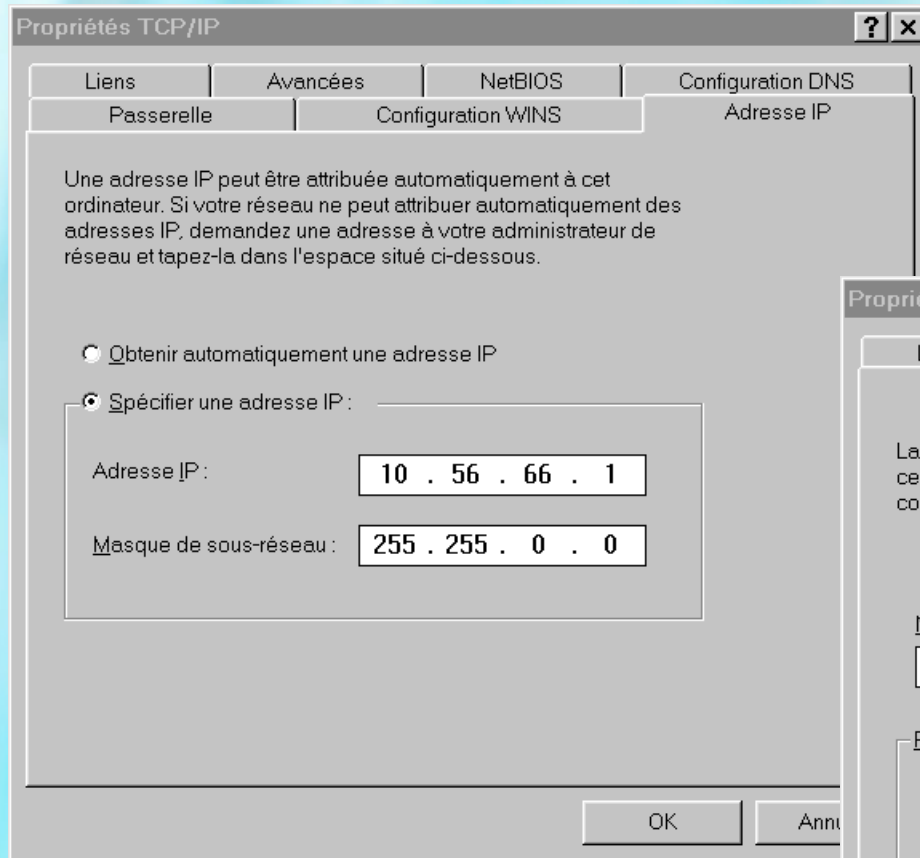
- Par extension, la notion de super réseau a été inventé en appliquant le netmask à la partie réseau.
- Cette notion permet de fusionner facilement des réseaux qui ont des adresses contiguës.

Numéro	193	51	12	1
Masque	255	255	0	0
	Réseau	Super-réseau	Machine	

- Grâce à ce masque, tous les réseaux en 193.51.XXX.0 ont été fusionnés en 1 réseau unique.
- Le super réseau permet de concevoir des réseaux de taille supérieure à 256 machines par agglomération de plusieurs classe C. Le seul prix est d'avoir des adresses contiguës.
- Le super réseau permet de concevoir des réseaux de 1024 machines à partir de 4 classes C fusionnées.
- Exemple :
 - réseaux = 193.1.4.0, 193.1.5.0, 193.1.6.0 et 193.1.7.0
 - netmask = 255.255.252.0 (252=11111100)
 - Il y a un réseau de 1024 machines à partir du réseau 193.1.4.0

- L'extension de ces possibilités et la réunion de ces deux concepts en un seul a donné naissance à la notion de réseaux /n (ou class less)
- Dans cette optique, n représente le nombre de bits pour le numéro de réseau.
 - /8 le ex classe A avec ses 4M machines
 - /24 le ex classe C avec ses 256 machines
 - /30 qui a 2 machines si on exclu le numéro 0 (local) et 3 (broadcast)
- Exemple d'adresses réseau :
- 193.54.8.0/24
- 10.52.12.0/16
- L'intérêt de cette forme d'écriture est sa compacité, propriété très importante pour les routeurs racines de l'internet.

- Si le message est local (dans le même réseau et sous réseau)
 - la carte émettrice se charge de l'acheminement vers la machine destinatrice
- Si le message n'est pas local
 - la carte émettrice ne connaît pas le chemin (route) vers la machine destinatrice
 - La carte émettrice s'adresse à un assistant (router) qui se charge de l'acheminement
- L'adresse "passerelle" permet de définir l'adresse IP de l'assistant
 - Une passerelle doit donc avoir (au moins) deux adresses IP
 - une dans le réseau courant
 - une dans le réseau supérieur
- Astuce technique du routage
 - Le message est expédié à une machine qui a une adresse IP définie, mais l'adresse MAC est celle de la passerelle. Quand la passerelle reçoit le message, elle comprend que ce n'est pas pour elle (pas le bon IP) et alors elle recherche le destinataire.



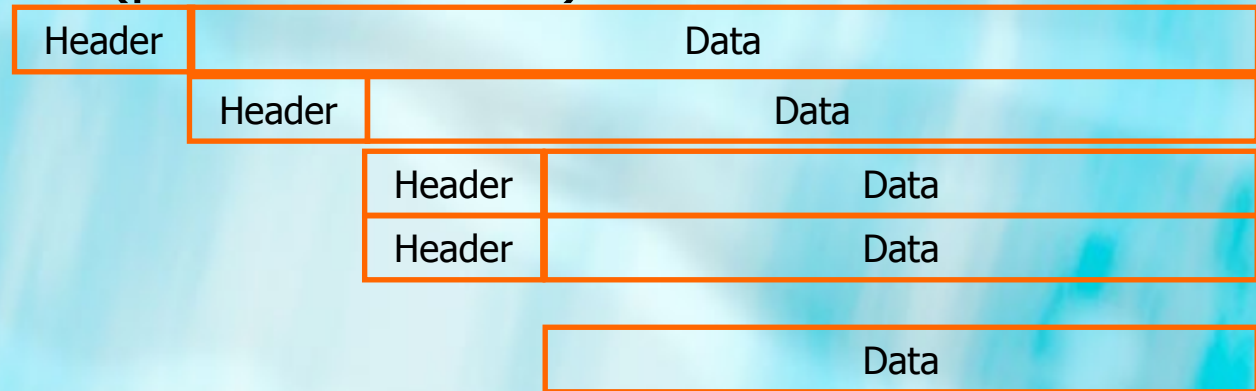
Réseau : 10 (class A)
Sous réseau : 56
Machine : 66.1
Passerelle : 10.56.0.1

- IPv4 est la norme actuelle, qui permet 4M d'adresses, elles ne suffiront jamais
- IPv6 est la future norme, qui code l'adresse IP sur 128 bits
- Exemple :
 - EFCD:AB89:6754:3210:0123:4567:89AB:CDEF
- En fait pour la transition :
 - IPv4 : 192.128.2.100 (C0 A8 02 64)
 - IPv6 : 0:0:0:0:0:0:C0A8:0264 ou plus simple ::C0A8:0264

- Trame ethernet
- Address Resolution Protocol
- Reverse Address Resolution Protocol
- Internet Protocol V4 (V6 dans protocoles avancés)
- Internet Control Message
- User Datagram Protocol
- Transmission Control Protocol

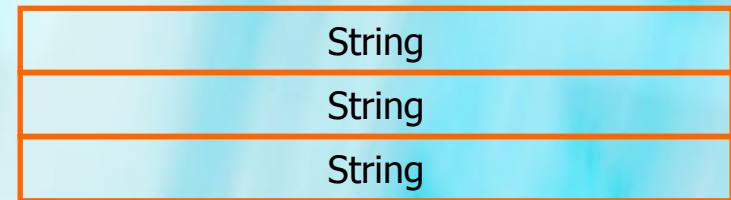
- Protocoles de base (pile de headers)

- trame
- ip
- udp
- tcp
- etc

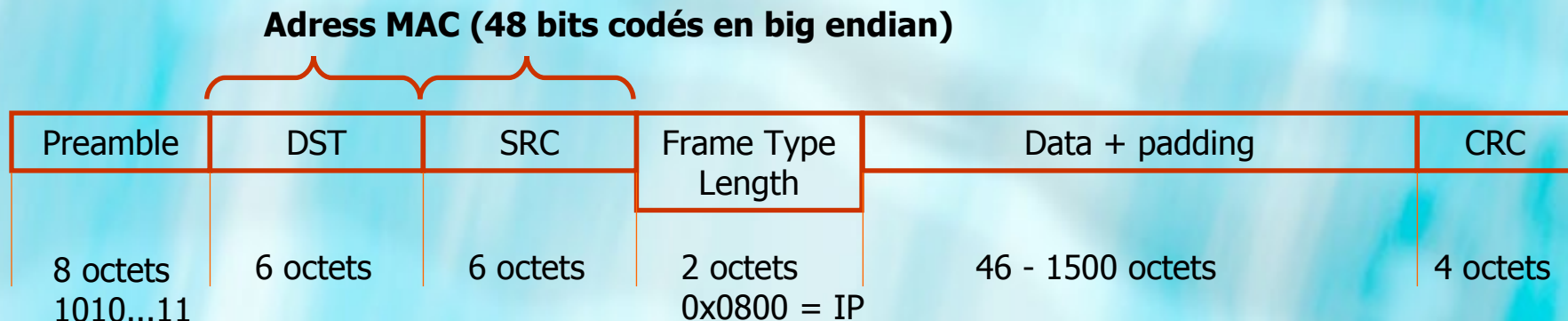


- Protocoles de haut niveau

- telnet
- ftp
- http
- etc



- L'ancienne (et la nouvelle) trame



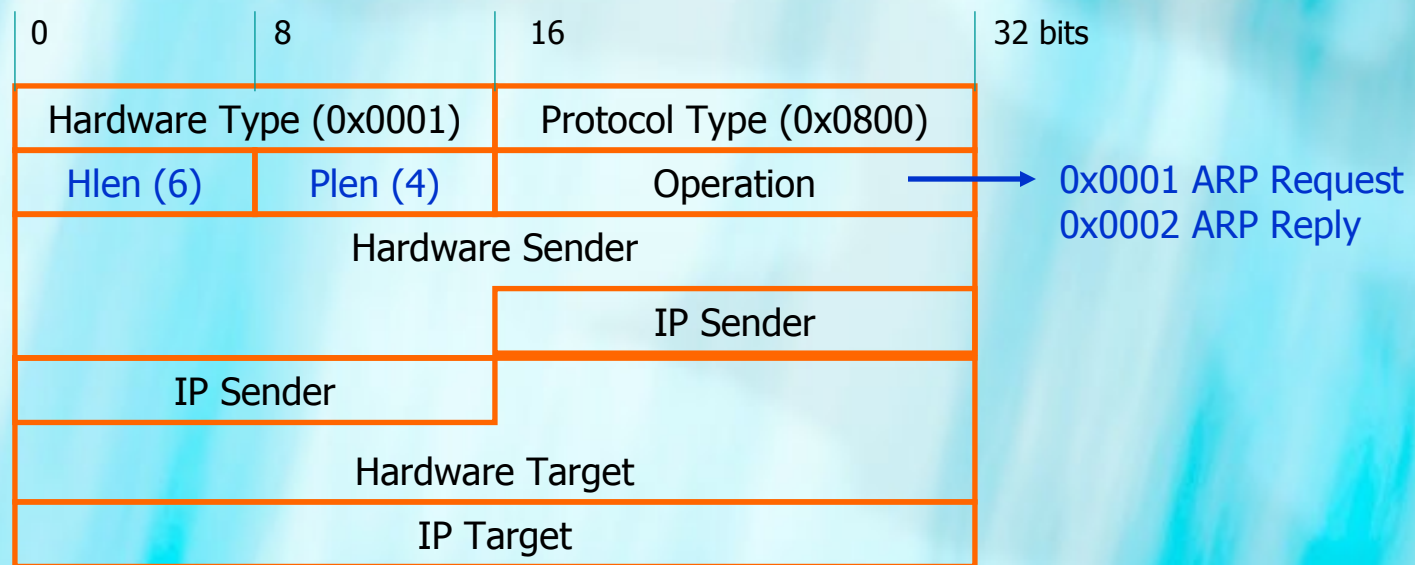
- La trame avancée (VLAN)

- Le champs FrameType/Length devient plus grand :
 - TPID (0x8100 est une valeur spéciale pour indiquer le concept de VLAN)
 - 3 bits pour coder la priorité
 - 1 bit à 0 (Ethernet)
 - 12 bits pour coder le numéro de VLAN
 - 16 bits pour le FrameType/Length (initial)



Minimum 64 octets
Maximum 1518 ou 1522 octets si VLAN

- Ce protocole (ARP) permet d'obtenir le numéro ethernet à partir du numéro Internet
- La machine en recherche émet un paquet de broadcast (FF*)
- Le destinataire répond directement à l'émetteur avec un paquet réponse



Hlen : nb octets de l'adresse MAC
Plen : nb octets de l'adresse IP

0	8	16	32 bits
0x0001		0x0800	
6	4	0x0001	
0x00104B749113			
			193.1.1.2
....			
0x000000000000			
193.1.1.1			

Recherche association IP->Mac

Requête pour 193.1.1.1

Le numéro IP a trouver

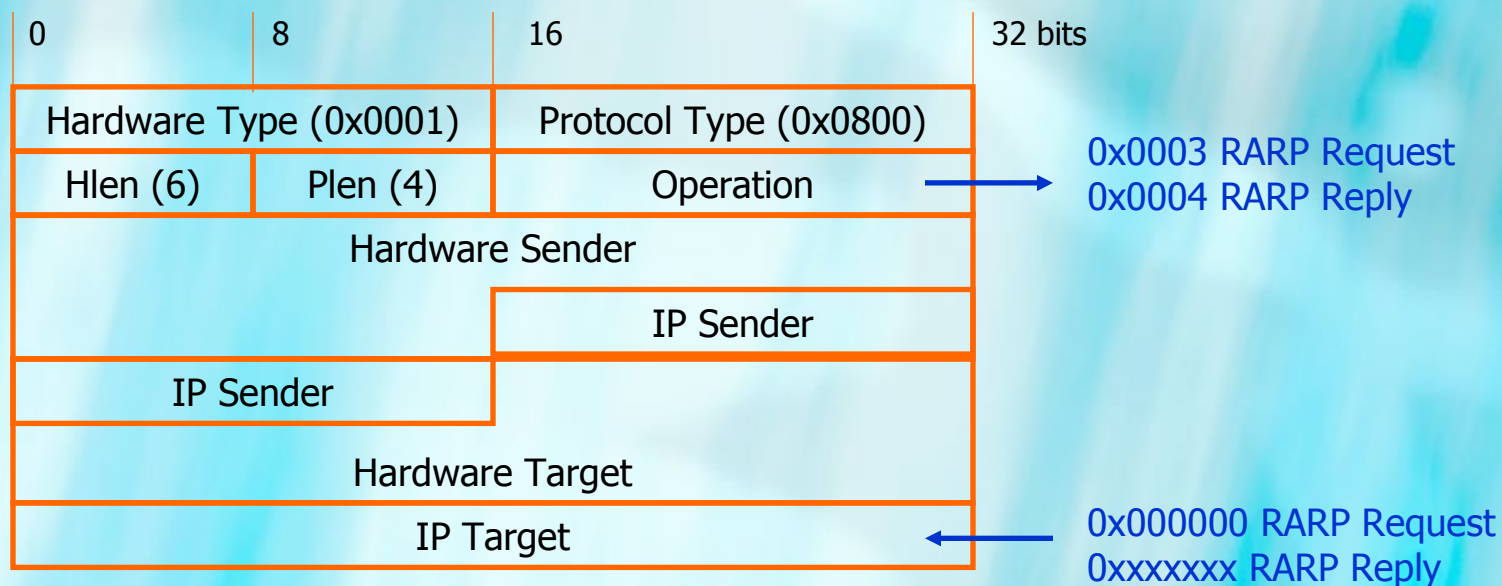
Réponse par la machine
193.1.1.1

Le numéro MAC

0	8	16	32 bits
0x0001		0x0800	
6	4	0x0002	
0x00104B749113			
			193.1.1.2
....			
0x00104B749112			
193.1.1.1			

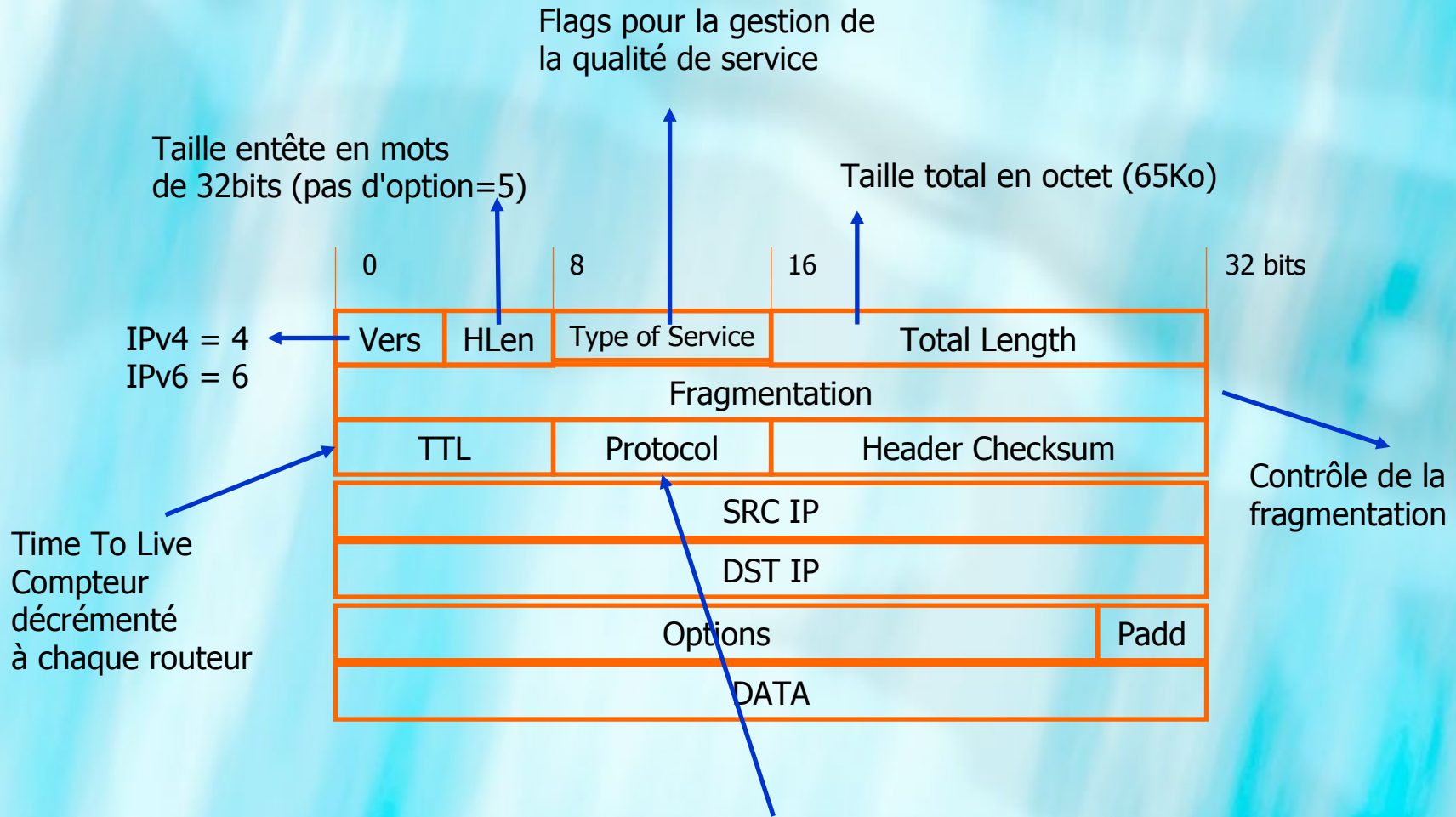
Réponse à la requête

- Ce protocole permet de rechercher le numéro Internet à partir du numéro ethernet



- Ce protocole (IPv4) est chargé de mettre en communication deux machines
- Il doit tenir compte de la taille des paquets
 - Ethernet MTU 1500 octets
 - Token Ring 4Mb/s MTU 4464 octets
 - Token Ring 8Mb/s MTU 8166 octets
 - Token Ring 16Mb/s MTU 17914 octets
 - X25&ISDN MTU 576 octets
- Il doit tenir compte de la localisation des machines
 - même segment
 - deux segments « proches » (il faut passer par un routeur)
 - deux segments « éloignés » (il faut passer par plein de routeur)
- Il doit tenir compte de la qualité de service, car des paquets sont plus importants que d'autres
 - vidéo/audio
 - contrôle temps réel
 - contrôle du trafic





Le numéro de protocole de la couche supérieure : 1=ICMP, 2=IGMP, 6=TCP, 17=UDP,...

Type of Service

Precedences (3bits)

Type (4 bits)

???

- La précedence permet de contrôler le niveau de priorité du paquet (du moins au plus)
- Ainsi un routeur saturé achemine d'abord les plus prioritaires puis les autres :
 - 0 Normal
 - 1 Priority
 - 2 Immediat
 - 3 Flash
 - 4 Flash override
 - 5 Critical
 - 6 Internetwork Control
 - 7 Network Control
- Le type du service permet de qualifier les besoins associés au paquet
 - 0x00 Normal
 - 0x01 Minimize delay (le routeur utilise le chemin le plus rapide - interactif)
 - 0x02 Maximize throughput (le routeur utilise le chemin avec la plus grande bande passante - débit)
 - 0x04 Maximize reliability (le routeur utilise le chemin qui est le plus fiable - fiabilité)
 - 0x08 Minimize cost (le routeur utilise le chemin avec un coût le moins cher - financier) expérimental
 - 0x10 Maximize security (le routeur utilise le chemin qui maximise la sécurité - militaire) expérimental

Identity	???	Allowed	More	Fragment Offset
----------	-----	---------	------	-----------------

- La fragmentation est gérée à l'aide de quatre champs :
 - Un identificateur de fragment (16 bits)
 - Un flag pour savoir si la fragmentation est autorisée sur ce segment
 - Un flag pour savoir si il y a encore des fragments supplémentaires
 - Un offset de fragment (13 bits) compté par 8 mots de 8 bits

- Exemple d'un paquet de 4464 octets

ID	More	Offset	Length
1310	0	0	4464

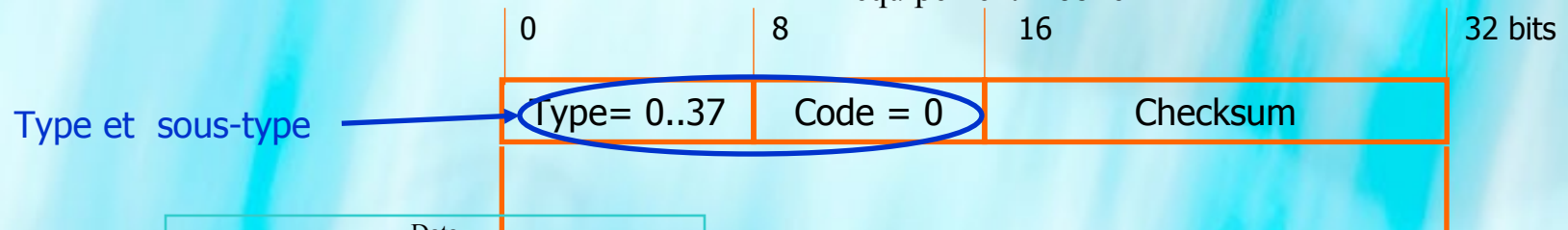
- Découpé en 4 paquets

ID	More	Offset	Length
1310	1	0	1500
1310	1	185	1500
1310	1	370	1500
1310	0	555	24

$$185 = (1500 - 20) / 8$$

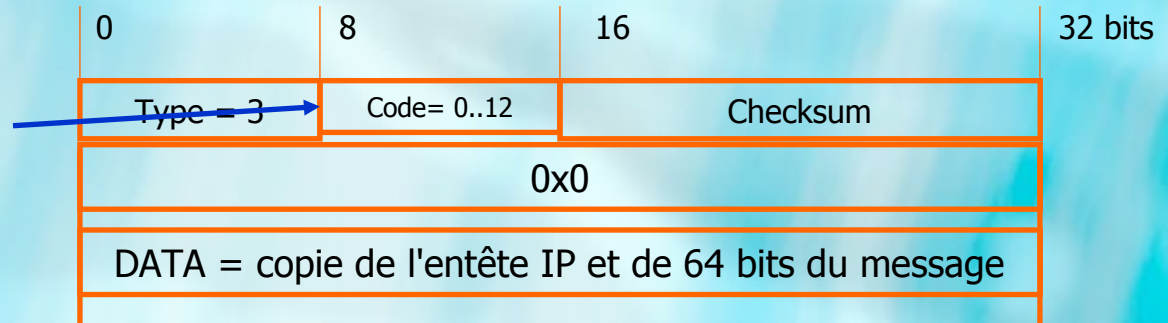
- Internet Control Message Protocol (ICMP - RFC) s'appuie sur IP pour envoyer des données de contrôle.
- Types de message

- | | |
|---|---|
| → 0 Réponse d'écho | 15 Demande d'information |
| 3 Destination inaccessible | 16 Réponse à la demande d'information |
| 4 « Source Quench » (demande de ralentissement) | 17 Demande de « netmask » |
| 5 Redirection | 18 Réponse à la demande de « netmask » |
| → 8 Echo | 30 Traceroute |
| 9 Annonce de routeur | 31 Erreur de conversion des datagrammes |
| 10 Sollicitation de routeur | 32 Redirection d'un équipement mobile |
| 11 TTL expire | 33 Localisation d'un équipement IPv6 |
| 12 Problème de paramétrage | 34 Réponse à la demande de localisation d'un équipement IPv6 |
| 13 Horodatage (Time Stamp) | 35 Demande d'enregistrement d'un équipement mobile |
| 14 Réponse horodatage | 37 Réponse à la demande d'enregistrement d'un équipement mobile |



- Unreachable destination (type 3)

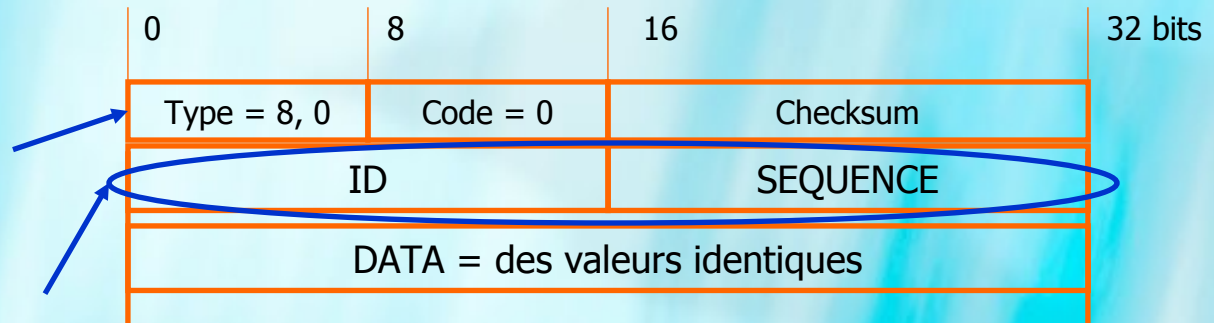
Code de la raison du problème
(0=réseau HS
1=machine,
3=port,
etc)



- echo/reply (types 8 et 0)

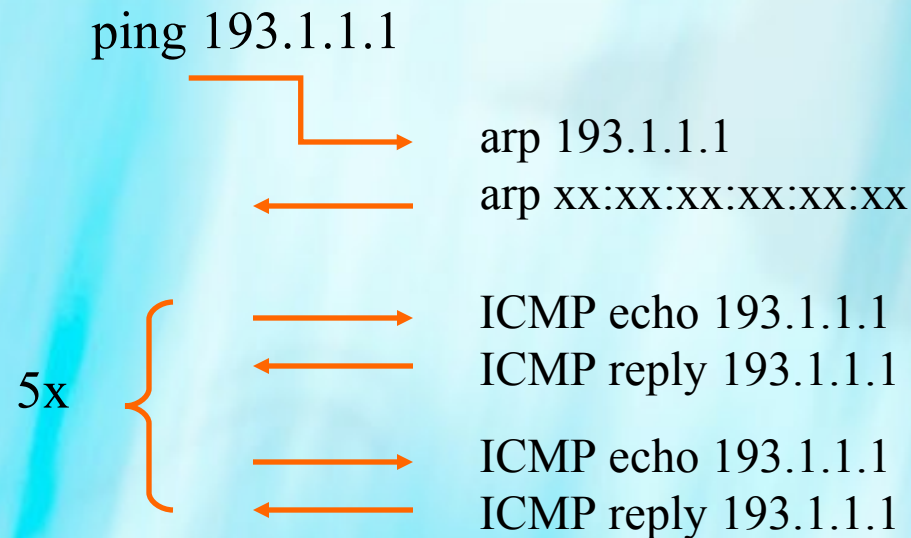
Envoie (8) / Réponse (0)

Identification des paquets



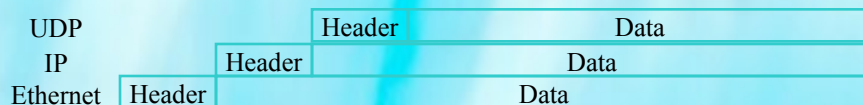
- Cette commande bien connue (Unix, PC, Mac) permet de vérifier la communication entre deux machines.
- Par défaut, cette commande envoie 5 paquets ICMP de requête et attend les réponses. Cette commande chronomètre les temps de transit pour identifier les délais.

- Rq :

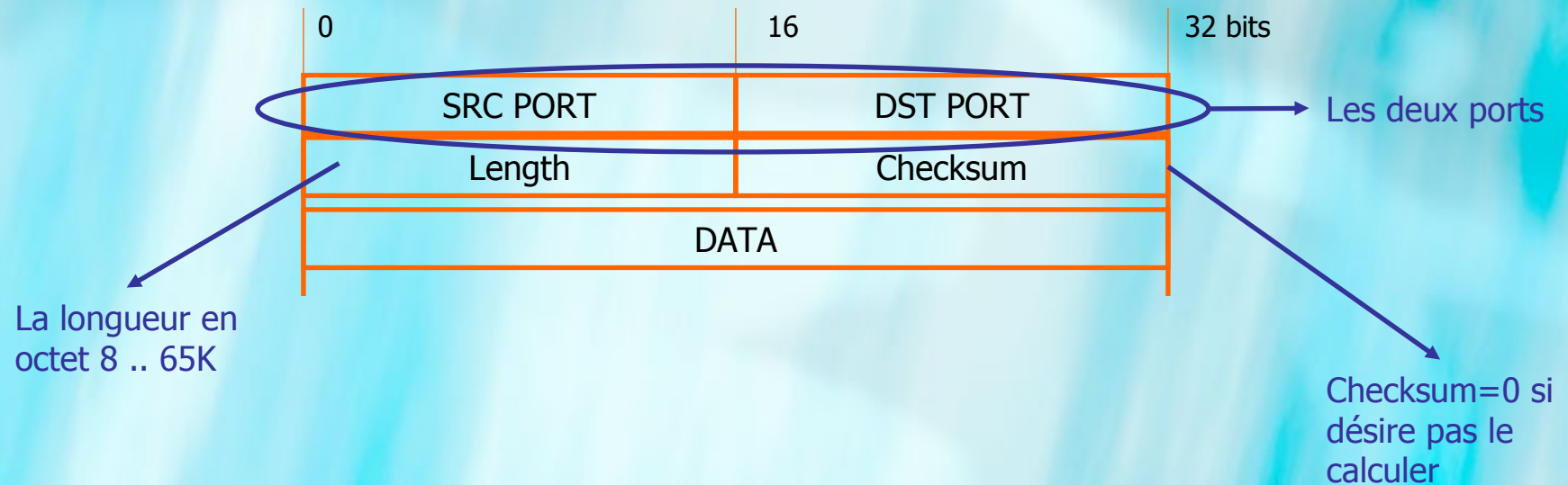


- La couche transport propose en standard deux protocoles différents
 - User Datagram Protocol (UDP) très simple, et léger
 - Transmission Control Protocol (TCP) plus complexe et lourd
 - D'autres protocoles sont à l'étude (à qualité de service variable)
- Les deux protocoles propose le même concept le plus important la notion de PORT :
 - Pour l'instant, ce sont des machines qui communiquent.
 - Une machine a besoin d'ouvrir plusieurs connexions simultanées.
 - Rien ne permet d'identifier une extrémité sur une machine.
 - La notion de PORT permet de relier le message à un programme concerné par le traitement du message.
 - C'est la machine qui aiguille les messages vers le bon programme en utilisant cette notion de PORT.

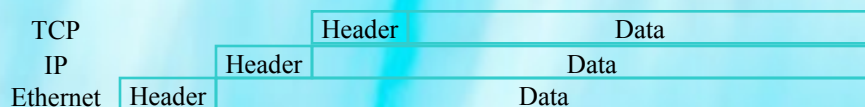
- User Datagram Protocol (UDP) est le premier protocole de la couche transport
- UDP est le plus simple
- UDP est en mode non-connecté
 - Chaque échange réinitialise une connexion et l'interrompt immédiatement
 - Plusieurs messages impliquent plusieurs échanges
- UDP limite la taille maximale (64Ko... Ethernet 1500 octets)
 - Si les données dépassent cette taille il faudra plusieurs messages
- UDP ne garanti pas la bonne réception du message par le destinataire
 - Le protocole n'impose pas d'accusé réception (bonne ou mauvaise) !
- La bande passante du réseau est économisée !
- Métaphore : le journal (unicast) ou la radio (multicast)

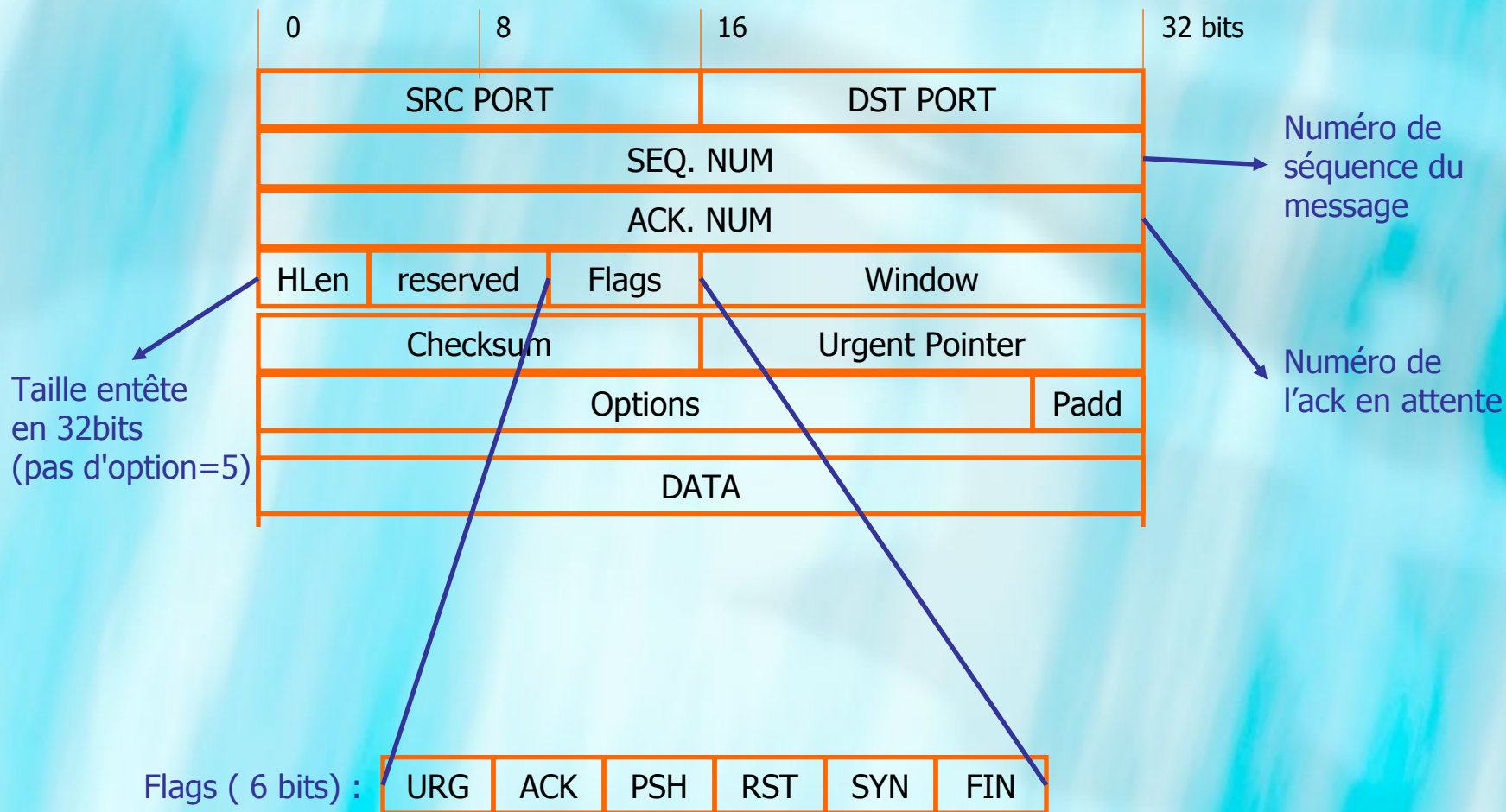


- Deux ports (source et destination) sont nécessaires pour identifier la communication et permettre l'aiguillage interne des messages

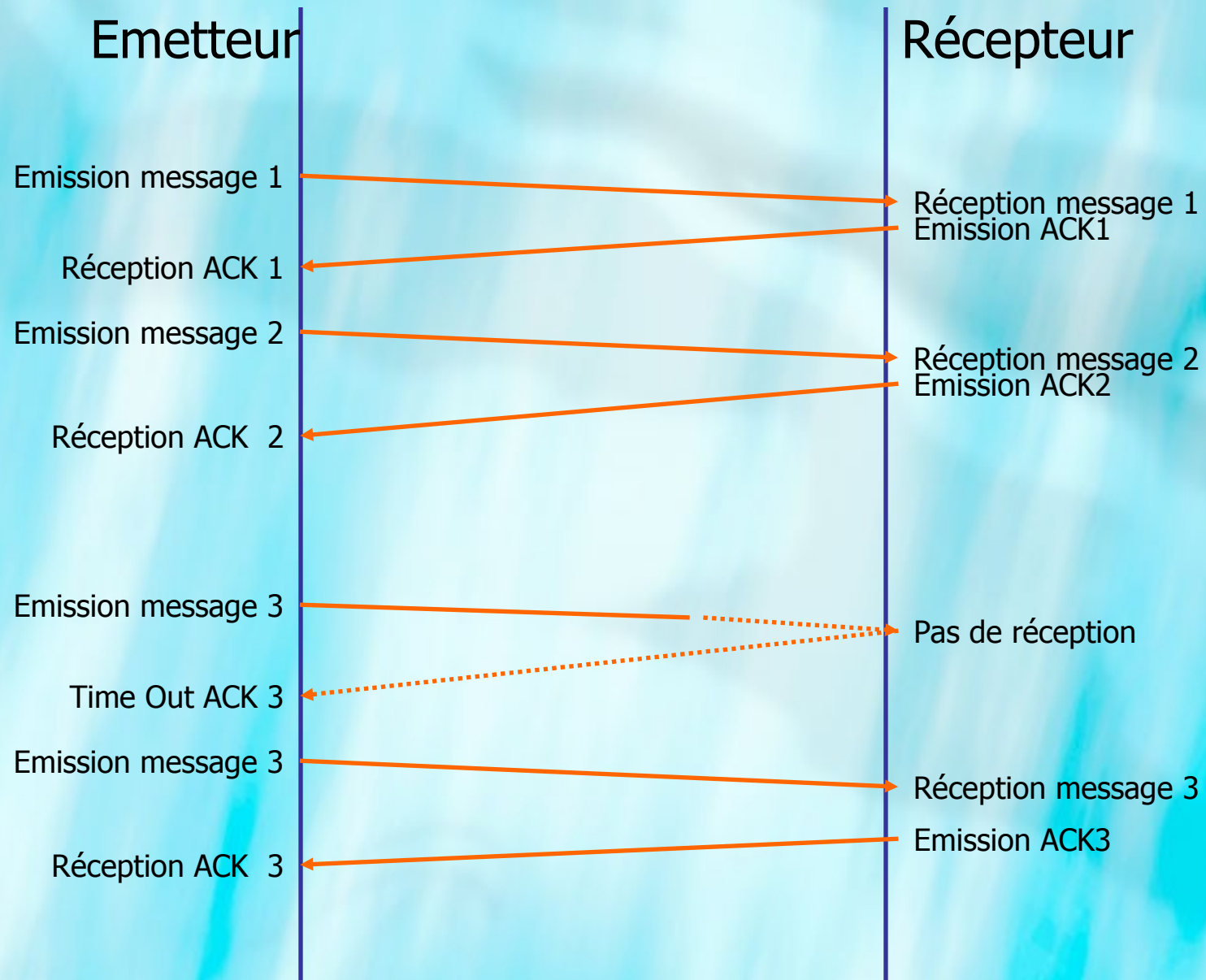


- Transmission Control Protocol (TCP) est le deuxième protocole de la couche transport
- TCP est très complexe
- TCP fonctionne en mode connecté (échanges bidirectionnels en full-duplex) entre les deux intervenants
- TCP peut transmettre une taille infinie... (flux)
- TCP garanti le bon déroulement de la communication
 - Le protocole est lourd et gourmand en bande passante
 - Il y a une poignée de main à l'ouverture, à la fermeture
 - Il y a des accusés de réception à chaque échange
- Métaphore : le téléphone.

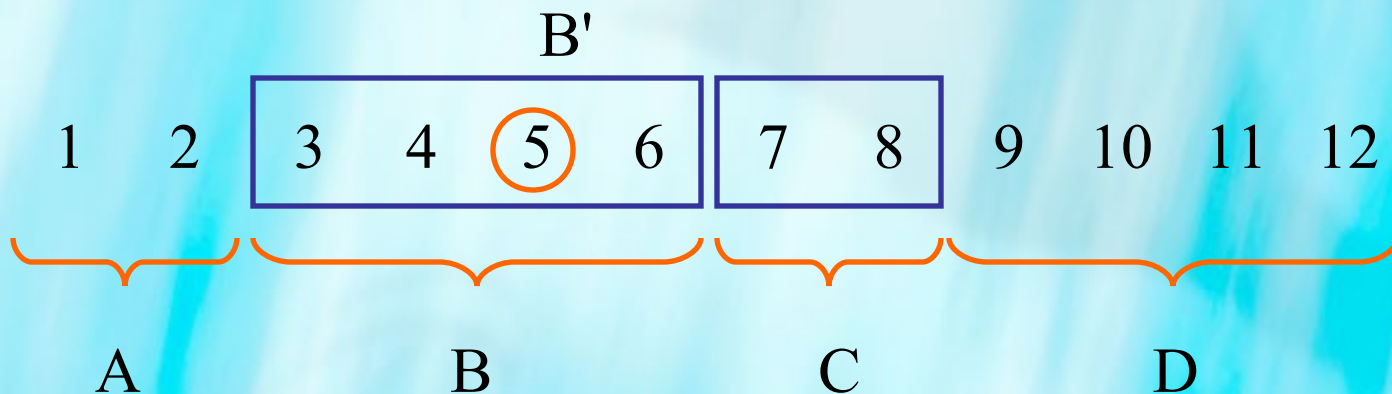




- SYN : ce message est une demande de synchronisation
- ACK : ce message est un acquittement de la synchronisation
- FIN : ce message est une fin de communication
- RST : ce message est un reset de la communication
- URG : ce message est urgent et doit être transmis avant les précédents
- (permet de court-circuiter le flux normal)
- PSH : ce message est émis sans attendre que le tampon d'émission est plein
- (active l'interactivité au prix d'une bande passante plus réduite)



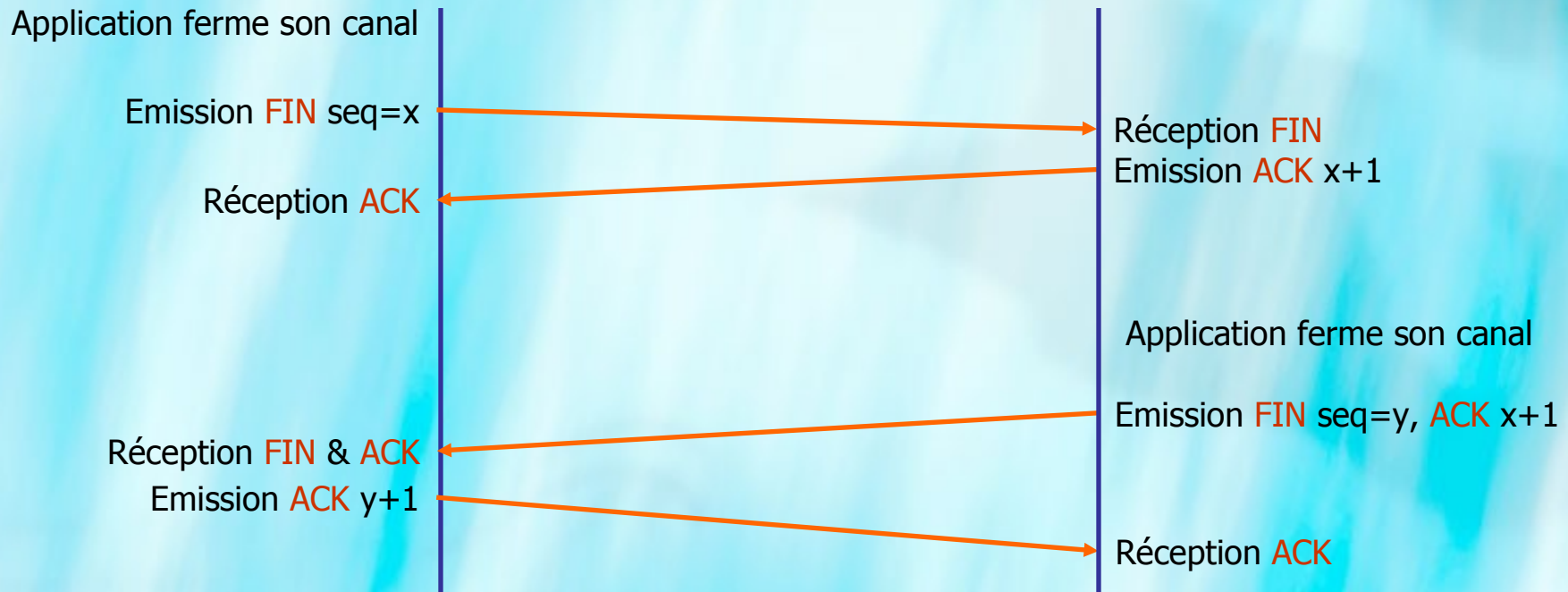
- A : messages émis, ACK arrivés.
- B : messages émis, ACK en attentes.
- B' : message émis, ACK arrivés.
- C : messages en cours d'émission car dans la fenêtre.
- D : messages retardés car hors fenêtre.



Ouverture



Fermeture

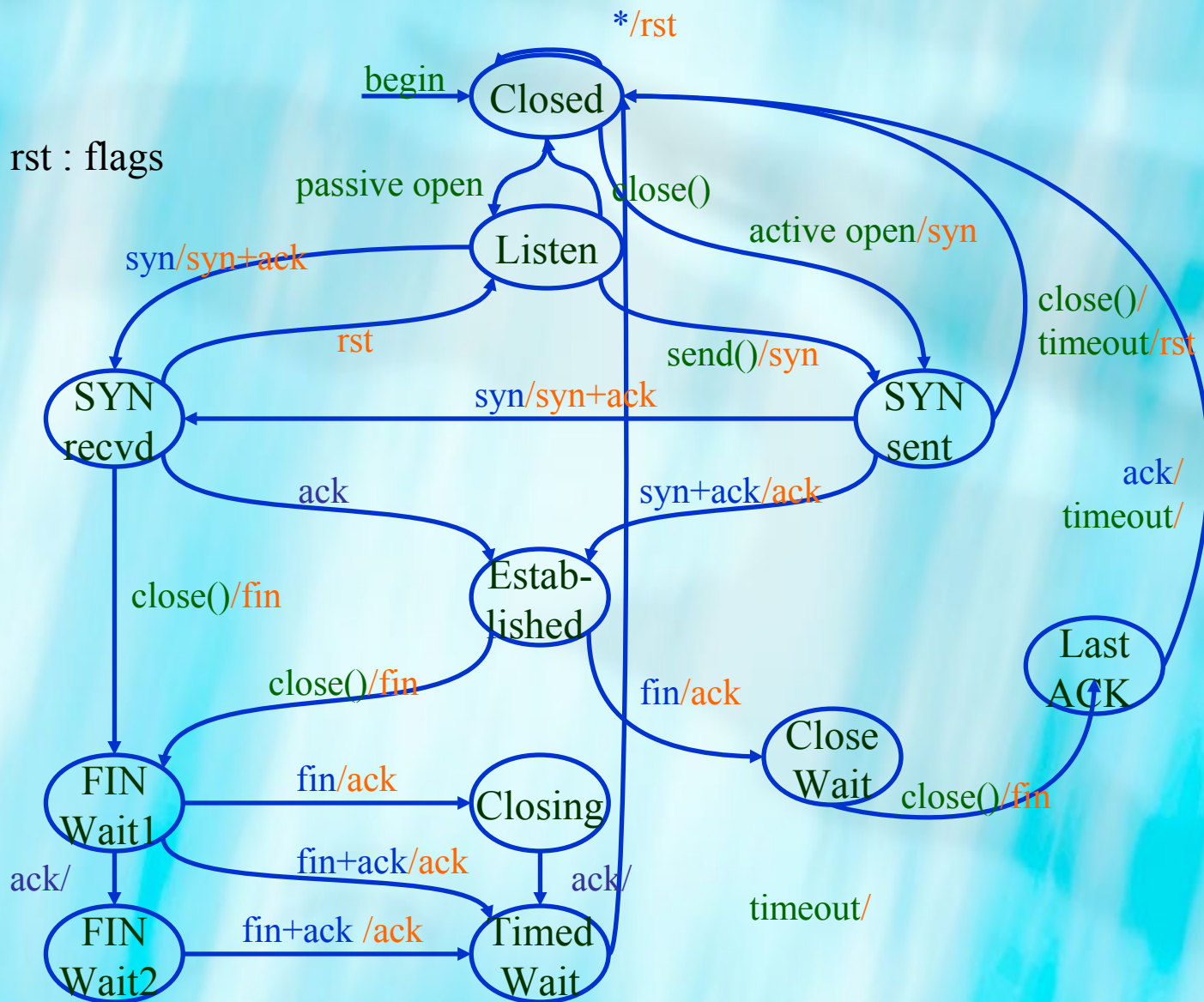


réception

émission

opération

fin, syn, ack, rst : flags



- 1024..65K : Attribués automatiquement par la machine
- 0..1024 : Attribués par une autorité officielle (NIC)

	Nom	TCP	UDP		Nom	TCP	UDP		Nom	TCP	UDP
	echo	7	7	Old DNS	name	42	42		snmp	161	161
/dev/null	discard	9	9	DNS	domain	53	53		snmp-trap		162
	systat	11	11		bootp		67		biff		512
ASCII	daytime	13	13		tftp		69		exec	512	
	netstat	15	15		www	80			login	513	
	qotd	17	17		pop/pop2	109			who		513
noise	chargen	19	19		pop3	110			shell	514	
	ftp-data	20			sunrpc	111	111		syslog		514
	ftp	21			nntp	119			printer	515	
	telnet	23			ntp		123		talk	517	
	smtp	25		μ\$	nbname		137		ntalk	518	
32bits	time	37	37		nbdatagram		138				
	rlp		39		nbssession	139			hmne	5678	1234

Universelle

```
SOCKET socket (int af, int type, int protocol);  
int connect (SOCKET s, const struct sockaddr *name, int namelen);  
int closesocket (SOCKET s);  
int bind (SOCKET s, const struct sockaddr *addr, int namelen);  
int listen (SOCKET s, int backlog);  
SOCKET accept (SOCKET s, struct sockaddr *addr, int *addrlen);  
int recv (SOCKET s, char * buf, int len, int flags);  
int recvfrom (SOCKET s, char * buf, int len, int flags, struct sockaddr *from, int * fromlen);  
int send (SOCKET s, const char * buf, int len, int flags);  
int sendto (SOCKET s, const char * buf, int len, int flags, const struct sockaddr *to, int tolen);  
int shutdown (SOCKET s, int how);  
int getpeername (SOCKET s, struct sockaddr *name, int * namelen);  
int getsockopt (SOCKET s, int level, int optname, char * optval, int *optlen);  
int setsockopt (SOCKET s, int level, int optname, const char * optval, int optlen);
```

Spécifique Windows

```
int WSASStartup(WORD wVersionRequired, LPWSADATA lpWSAData);  
int WSACleanup(void);
```

```
class CAsyncSocket : public CObject {
    CAsyncSocket();

    BOOL Create (UINT nSocketPort = 0, int nSocketType=SOCK_STREAM,
        long lEvent = FD_READ | FD_WRITE | FD_OOB | FD_ACCEPT | FD_CONNECT | FD_CLOSE,
        LPCTSTR lpszSocketAddress = NULL);

    virtual BOOL Accept (CAsyncSocket& rConnectedSocket,
        SOCKADDR* lpSockAddr = NULL, int* lpSockAddrLen = NULL);
    BOOL Bind (UINT nSocketPort, LPCTSTR lpszSocketAddress = NULL);
    BOOL Bind (const SOCKADDR* lpSockAddr, int nSockAddrLen);
    BOOL Connect (LPCTSTR lpszHostAddress, UINT nHostPort);
    BOOL Connect (const SOCKADDR* lpSockAddr, int nSockAddrLen);
    BOOL Listen (int nConnectionBacklog=5);

    virtual void Close ();
    enum { receives = 0, sends = 1, both = 2 };
    BOOL Shutdown (int nHow = sends);

    virtual int Receive (void* lpBuf, int nBufLen, int nFlags = 0);
    int ReceiveFrom (void* lpBuf, int nBufLen, CString& rSocketAddress, UINT& rSocketPort, int nFlags = 0);
    int ReceiveFrom (void* lpBuf, int nBufLen, SOCKADDR* lpSockAddr, int* lpSockAddrLen, int nFlags = 0);
    virtual int Send (const void* lpBuf, int nBufLen, int nFlags = 0);
    int SendTo (const void* lpBuf, int nBufLen, UINT nHostPort, LPCTSTR lpszHostAddress = NULL, int nFlags = 0);
    int SendTo (const void* lpBuf, int nBufLen, const SOCKADDR* lpSockAddr, int nSockAddrLen, int nFlags = 0);

    BOOL SetSockOpt (int nOptionName, const void* lpOptionValue, int nOptionLen, int nLevel = SOL_SOCKET);
    BOOL GetSockOpt (int nOptionName, void* lpOptionValue, int* lpOptionLen, int nLevel = SOL_SOCKET);

    .....
};
```


C'est un extrait expurgé de la doc java

java.net

Class Socket

java.lang.Object

|

+--java.net.Socket

```
public Socket {  
    protected Socket()  
    Socket (InetAddress address, int port)  
    Socket (InetAddress address, int port, InetAddress localAddr, int localPort)  
    Socket (String host, int port)  
    Socket (String host, int port, InetAddress localAddr, int localPort)  
  
    void close ()  
    InputStream getInputStream()  
    OutputStream getOutputStream()  
    void shutdownInput()  
    void shutdownOutput()  
};
```


C'est un extrait expurgé de la doc java

java.net

Class ServerSocket

java.lang.Object

|

+--java.net.ServerSocket

```
public ServerSocket {  
    ServerSocket (int port)  
    ServerSocket (int port, int backlog)  
    ServerSocket (int port, int backlog, InetAddress bindAddr)  
  
    Socket accept ()  
    void close ()  
};
```

Serveur

Client

Création d'une socket
La socket est **attachée** à un protocole/port
Se met en **acceptation** bloquante sur la socket

Création d'une socket
Connexion de la socket au serveur

Accepte la communication et
retourne une socket de travail

Eventuellement **créé** une
tâche spécifique pour
répondre à cette requête

Pour communiquer utilisent
Receive/ReceiveFrom/Send/SendTo

Termine la connexion
Revient à l'étape **Accepte**

Termine la connexion

Fonction qui retourne le numéro IP d'une machine identifiée par son nom

```
struct hostent * gethostbyname(const char * name);
```

```
struct hostent {  
    char    * h_name;           /* official name of host */  
    char    ** h_aliases;      /* alias list */  
    short    h_addrtype;       /* host address type */  
    short    h_length;         /* length of address */  
    char    ** h_addr_list; /* list of addresses */  
};  
#define h_addr h_addr_list[0] /* address, for backward compat */
```

Ce qui est utile :

h_addr ou h_addr_list (le numéro IP prêt à être utilisé sur le réseau (big endian)

h_length la longueur du champs

La fonction retourne NULL si elle ne trouve pas l'adresse de la machine

```
gethostbyname ("www . microsoft . com")
```

```
gethostbyname ("idefix")
```

Fonction qui retourne le numéro d'un protocole identifié par son nom

```
struct protoent * getprotobyname(const char * name);
```

```
struct protoent {  
    char    * p_name;           /* official protocol name */  
    char    ** p_aliases;       /* alias list */  
    short    p_proto;           /* protocol # */  
};
```

ce qui est utile :

p_proto au format des entiers sur un réseau (big endian)

La fonction retourne NULL si ne trouve pas le protocole demandé

```
getprotobyname ("tcp");  
getprotobyname ("udp");
```


Fonction qui retourne le numéro d'un service (port) identifié par son nom

```
struct servent * getservbyname(const char * name, const char * proto);
```

```
struct servent {  
    char    FAR * s_name;           /* official service name */  
    char    FAR * FAR * s_aliases; /* alias list */  
    short   s_port;                /* port # */  
    char    FAR * s_proto;         /* protocol to use */  
};
```

ce qui est utile :

s_port au format des entiers sur un réseau (big endian)

La fonction retourne NULL si ne trouve pas le service (port) demandé

```
getservbyname ("telnet", "tcp");  
getservbyname ("tftp", "udp");  
getservbyname ("hmne", "tcp");
```

Ces fonctions sont absolument nécessaire :

Une machine Sparc, HP, IBM, etc : ces fonctions ne font rien

Une machine PC : les fonctions permutent les octets...

Il ne faut pas les utiliser quand ce n'est pas nécessaire (par exemple les fonctions précédentes ont déjà fait les permutations nécessaires)...

Glossaire : h=Host, n=Network, s=Short, l=long.

Conversions sur les entiers long :

```
u_long htonl (u_long hostlong);
```

```
u_long ntohl (u_long netlong);
```

x86

net

68K



Conversions sur les entiers court :

```
u_short htons (u_short hostshort);
```

```
u_short ntohs (u_short netshort);
```

Conversion chaîne contenant le numéro IP d'une machine -> adresse IP

```
unsigned long inet_addr (const char * cp);
```

```
// "193.1.1.1" -> adresse IP 32 bits au format des entiers sur un réseau (big  
endian)
```

Beaucoup de fonctions référencent une `const struct sockaddr *`

C'est un pointeur vers une structure « opaque ». Cette structure ne stocke rien d'utile.

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

Il faut toujours faire des casts vers les bonne implémentations...

Cette solution permet de créer facilement de nouvelles implémentations liées à de nouveaux protocoles (unix a créé des `sockaddr_ux` et Microsoft des `sockaddr_irda`).

La `sockaddr_in` est l'implémentation de la `sockaddr` pour le monde internet

```
struct sockaddr_in {  
    short  sin_family;  
    u_short sin_port;  
    struct in_addr sin_addr;  
    char    sin_zero[8];  
};
```

`sin_family` -> Numéro de famille : `AF_INET` ou `PF_INET` (selon implémentation)

`sin_port` -> le port (service) utilisé

`sin_addr` -> le numéro IP (32 bits) de la machine ou `INADDR_ANY`


```
WSADATA data;
WSAStartup (0x202, &data);
try {

    struct protoent *ppe = getprotobyname ("udp");
    if (ppe == NULL) throw exception ("Illegal Protocol");

    SOCKET s = socket (AF_INET, SOCK_DGRAM, ppe->p_proto);
    if (s == INVALID_SOCKET) throw exception ("Invalid Socket");

    struct sockaddr_in sin;
    memset (&sin, 0, sizeof(sin));
    sin.sin_family = AF_INET;
    sin.sin_addr.s_addr = INADDR_ANY;
    sin.sin_port = htons (1234);
    if (bind (s, (struct sockaddr *) &sin, sizeof(sin)) == SOCKET_ERROR)
        throw exception ("Bind Error");
    if (listen (s, 50) == SOCKET_ERROR)
        throw exception ("Listen Error");
    ...
    closesocket (s);

}
catch (exception e) {
    cerr << "SOCKET ERROR : " << e.what() << " # " << WSAGetLastError () << endl;
}
WSACleanup ();
```



```
WSADATA data;
WSAStartup (0x202, &data);
try {
    struct protoent *ppe = getprotobyname ("tcp");
    if (ppe == NULL) throw exception ("Illegal Protocol");

    SOCKET s = socket (AF_INET, SOCK_STREAM, ppe->p_proto);
    if (s == INVALID_SOCKET) throw exception ("Invalid Socket");

    struct sockaddr_in sin;
    memset (&sin, 0, sizeof(sin));
    sin.sin_family = AF_INET;

    struct hostent *phe = gethostbyname ("maccaffrey.univ-mulhouse.fr");
    if (phe == NULL) throw exception ("Invalid host name");
    memcpy (&sin.sin_addr, phe->h_addr, phe->h_length);

    struct servent *pse = getservbyname ("hmne", "tcp");
    if (pse == NULL) throw exception ("Invalid Service");
    sin.sin_port = pse->s_port;

    if (connect (s, (struct sockaddr *) &sin, sizeof(sin)) == SOCKET_ERROR)
        throw exception ("Connect Error");

    ...
    closesocket (s);
}
catch (exception e) {
    cerr << "SOCKET ERROR : " << e.what() << " # " << WSAGetLastError () << endl;
}
WSACleanup ();
```

- En principe, il y a 8 stratégies selon le schéma ci-dessous

Avec état interne	
Sans état interne	
Itératif (mono tâche) non connecté (udp)	Itératif (mono tâche) connecté (tcp)
Concurrent (multi tâches) non connecté (udp)	Concurrent (multi tâches) connecté (tcp)

- 1°) Des états (même les caches)
 - Très dangereux, limite le passage à l'échelle, ... à éviter.
- 2°) UDP ou TCP (cela dépend des messages...)
 - UDP à une taille de message limité et n'est pas fiable mais est très léger
 - TCP travaille avec une taille illimité et est fiable mais est très lourd
- 3°) itératif ou concurrent (cela dépend du calcul de la réponse)
 - Le mode itératif est rapide (pas de charge de fonctionnement)
 - Il bloque le serveur pendant la conception de la réponse
 - Le mode concurrent ne bloque pas le serveur
 - Il est plus lent (création de processus, besoin de mémoire)

- mono tâche

Créer la socket d'écoute

Attacher la socket d'écoute au port

Attendre une requête qui sera fournie sur une socket de travail

Traiter la requête à partir des données de la socket de travail

Donner la réponse sur la socket de travail



- multi tâches

Créer la socket d'écoute

Attacher la socket d'écoute au port

Attendre une requête sur la socket de travail

Faire un processus fils

Transmettre la socket de travail au fils



Lire une requête de la socket de travail

Traiter la requête à partir des données

Donner la réponse sur la socket de travail

Mode non connecté (réceptionne d'un émetteur et envoie vers un destinataire connecté au vol)

int recvfrom (SOCKET s, char * buf, int len, int flags, struct sockaddr *from, int * fromlen);

int sendto (SOCKET s, const char * buf, int len, int flags, const struct sockaddr *to, int tolen);

Mode connecté (réception d'un émetteur et envoie vers un destinataire déjà connecté)

int send (SOCKET s, const char * buf, int len, int flags);

int recv (SOCKET s, char * buf, int len, int flags);

Les messages sous UDP gardent leurs limites.

Un message de X octets doit être envoyé en une fois et réceptionné en 1 fois.

Il faut prendre ses précautions (taille des tampons) pour lire le message.

Les messages sous TCP ne gardent pas leurs limites.

Un message de X octets peut être envoyé en N fois dans des formats différents...

Ce même message peut être réceptionné en M fois avec des formats différents...

Exemple : 60 octets écrit avec le send

4 messages de 10 octets + 1 messages de 20 octets à l'émission.

2 messages de 20 octets + 2 messages de 10 octets à la réception.

En fait, un message est réceptionné en entier quand l'émetteur ferme le fichier (streaming)

Pas de vraie synchronisation en terme de compte de paquet,

La fermeture interdit un échange par la suite.

Il faut prendre ses précautions pour lire le message.

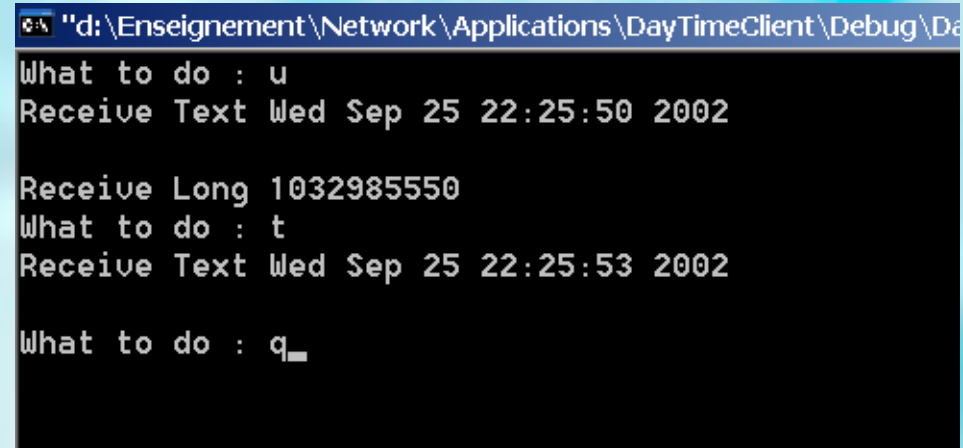
Ma fonction de réception en mode streaming

```
void Reader_TCP (SOCKET s, string & Input) {  
  
    char buffer[256]; // cette taille n'est pas toujours suffisante  
    char *tmp = buffer;  
    int bufferlen = sizeof(buffer);  
    int len;  
  
    do {  
        len = recv (s, tmp, bufferlen, 0);  
        tmp += len;  
        bufferlen -= len;  
    } while (len != SOCKET_ERROR && len != 0 && bufferlen != 0);  
  
    if (len == SOCKET_ERROR)  
        throw exception ("Reader Error");  
    Input.assign (buffer, tmp - buffer);  
}
```

- Les exemples sont construits sur la base suivante :
 - a) Le format des données est soit un entier, soit une chaîne à taille variable.
 - b) On explore exhaustivement (sans volonté de proposer une habitude) toutes les formes d'initialisation, de connexion et d'échange
- A) Un client universel : UDP, TCP, TCP synchronisé.
- B) Un serveur mono-tâche UDP
- C) Un serveur mono-tâche TCP
- D) Un serveur multi-tâches TCP
- E) Un serveur TCP synchronisé
- F) Un serveur TCP et UDP simultanément
- G) Un client TCP avec Microsoft C++
- H) Un client TCP avec Java
- I) Un serveur Web en Java


```
void main(int argc, char* argv[])
{
    WSADATA data;
    WSAStartup (0x202, &data);
    try {
        while (1) {
            cout << "What to do : ";
            char line[256];
            cin.getline(line, sizeof(line));
            switch (line[0]) {
                case 'u' : UDP_Client (); break;
                case 't' : TCP_Client (); break;
                case 's' : TCP_Sync_Client (); break;
                case 'q' : exit (0); break;
            }
        }
    }
    catch (exception e)
    {
        cerr << "SOCKET ERROR : " << e.what() << " # " << WSAGetLastError () << endl;
    }

    WSACleanup ();
}
```



```
"d:\Enseignement\Network\Applications\DayTimeClient\Debug\Da
What to do : u
Receive Text Wed Sep 25 22:25:50 2002

Receive Long 1032985550
What to do : t
Receive Text Wed Sep 25 22:25:53 2002

What to do : q_
```



```
extern SOCKET Create_Socket_Server_UDP (short service);
```

```
extern SOCKET Create_Socket_Server_TCP (short service);
```

```
extern SOCKET Create_Socket_Client_UDP (const char * server, const char * service);
```

```
extern SOCKET Create_Socket_Client_TCP (const char * server, const char * service);
```

```
extern void Reader_UDP_Unconnected (SOCKET s, string & Input, struct sockaddr_in *from);
```

```
extern void Writer_UDP_Unconnected (SOCKET s, const string & Output, struct sockaddr_in *from);
```

```
extern void Reader_UDP_Connected (SOCKET s, string & Input);
```

```
extern void Writer_UDP_Connected (SOCKET s, const string & text);
```

```
extern void Reader_TCP (SOCKET s, string & Input);
```

```
extern void Writer_TCP (SOCKET s, const string & text);
```

```
extern void Synch_Reader_TCP (SOCKET s, string & Input);
```

```
extern void Synch_Writer_TCP (SOCKET s, const string & text);
```

```
extern void Reader_UDP_Connected (SOCKET s, long & Input);
```

```
extern void Writer_UDP_Unconnected (SOCKET s, long Output, struct sockaddr_in *from);
```

Construction de la socket pour le client

```
SOCKET Create_Socket_Client_UDP (const char * serveradress, const char * service) {  
    struct protoent *ppe = getprotobyname("udp");  
    if (ppe == NULL) throw exception ("Illegal Protocol");  
  
    SOCKET s = socket (AF_INET, SOCK_DGRAM, ppe->p_proto);  
    if (s == INVALID_SOCKET) throw exception ("Invalid Socket");  
  
    struct sockaddr_in sin;  
    memset (&sin, 0, sizeof(sin));  
    sin.sin_family = AF_INET;  
    sin.sin_addr.s_addr = inet_addr (serveradress);  
  
    struct servent *pse = getservbyname (service, "udp");  
    if (pse == NULL) throw exception ("Invalid Service");  
    sin.sin_port = pse->s_port;  
  
    if (connect (s, (struct sockaddr *) &sin, sizeof(sin)) == SOCKET_ERROR)  
        throw exception ("Connect Error");  
  
    return s;  
}
```

Procédures de lectures du client

```
void Reader_UDP_Connected (SOCKET s, string & Input) {  
    char buffer[256];  
    int len = recv (s, buffer, sizeof (buffer), 0);  
    if (len == SOCKET_ERROR) throw exception ("String Reader Error");  
    buffer[len] = '\0';  
    Input = buffer;  
}
```

```
void Reader_UDP_Connected (SOCKET s, long & Input) {  
    char buffer[256];  
    int len = recv (s, buffer, sizeof (buffer), 0);  
    if (len == SOCKET_ERROR) throw exception ("Long Reader Error");  
    buffer[len] = '\0';  
    Input = ntohl (* (long *)buffer);  
}
```

Procédure d'écriture du client

```
void Writer_UDP_Connected (SOCKET s, const string & text) {  
    int len = send (s, text.data(), text.size(), 0);  
    if (len == SOCKET_ERROR) throw exception ("Reader Error");  
}
```



```
static void UDP_Client () {  
  
    SOCKET s = Create_Socket_Client_UDP ("127.0.0.1", "hmne");  
  
    Writer_UDP_Connected (s, "");  
  
    string TextInput;  
    Reader_UDP_Connected (s, TextInput);  
    cout << "Receive Text " << TextInput << endl;  
  
    long LongInput;  
    Reader_UDP_Connected (s, LongInput);  
    cout << "Receive Long " << LongInput << endl;  
  
    closesocket (s);  
}
```


Construction de la socket pour le serveur

```
SOCKET Create_Socket_Server_UDP (short service) {  
    struct protoent *ppe = getprotobyname("udp");  
    if (ppe == NULL) throw exception ("Illegal Protocol");  
  
    SOCKET s = socket (AF_INET, SOCK_DGRAM, ppe->p_proto);  
    if (s == INVALID_SOCKET) throw exception ("Invalid Socket");  
  
    struct sockaddr_in sin;  
    memset (&sin, 0, sizeof(sin));  
    sin.sin_family = AF_INET;  
    sin.sin_addr.s_addr = INADDR_ANY;  
    sin.sin_port = htons (service);  
  
    if (bind (s, (struct sockaddr *) &sin, sizeof(sin)) == SOCKET_ERROR)  
        throw exception ("Bind Error");  
  
    return s;  
}
```

Procédure de lecture du serveur en mode non connecté

```
void Reader_UDP_Unconnected (SOCKET s, string & Input, struct sockaddr_in *from) {  
    char buffer [256];  
    int fromlen = sizeof (*from);  
  
    int len = recvfrom (s, buffer, sizeof(buffer), 0, (struct sockaddr *) from, &fromlen);  
    if (len == SOCKET_ERROR) throw exception ("Reader Error");  
    Input = buffer;  
}
```

Procédures d'écritures du serveur en mode non connecté

```
void Writer_UDP_Unconnected (SOCKET s, const string & Output, struct sockaddr_in *from) {  
    int r = sendto (s, Output.data(), Output.size(), 0, (struct sockaddr *) from, sizeof (*from));  
    if (r == SOCKET_ERROR) throw exception ("Writer Error");  
}  
  
void Writer_UDP_Unconnected (SOCKET s, long Output, struct sockaddr_in *from) {  
    long Value = htonl (Output);  
    int r = sendto (s, (const char *) &Value, sizeof(Value), 0, (struct sockaddr *) from, sizeof (*from));  
    if (r == SOCKET_ERROR) throw exception ("Writer Error");  
}
```

```
#include "stdafx.h"
#include "STDEXCPT.h"
#include "winsock.h"
#include "socket.h"
#include <time.h>

void main(int argc, char* argv[]) {
    WSADATA data; WSAStartup (0x202, &data);
    try {
        SOCKET s = Create_Socket_Server_UDP (1234);

        string Input;
        struct sockaddr_in from;
        Reader_UDP_Unconnected (s, Input, &from);

        time_t clock_time;
        time (&clock_time );
        struct tm *current_time = localtime( &clock_time );

        cout << clock_time << endl;
        cout << asctime (current_time) << endl;
        Writer_UDP_Unconnected (s, asctime (current_time), &from);
        Writer_UDP_Unconnected (s, clock_time, &from);
    }
    catch (exception e) {
        cerr << "SOCKET ERROR : " << e.what() << " # " << WSAGetLastError () << endl;
    }
    WSACleanup ();
}
```

Client :

```
d:\Enseignement\Network\Applications\DayTimeClient\D
What to do : u
Receive Text Wed Sep 25 22:45:51 2002

Receive Long 1032986751
What to do :
```

Serveur :

```
d:\Enseignement\Network\Applications\DayTimeUDP\
1032986751
Wed Sep 25 22:45:51 2002

_
```



```
SOCKET Create_Socket_Client_TCP (const char * servername, const char * service) {  
    struct protoent *ppe = getprotobyname("tcp");  
    if (ppe == NULL) throw exception ("Illegal Protocol");  
  
    SOCKET s = socket (AF_INET, SOCK_STREAM, ppe->p_proto);  
    if (s == INVALID_SOCKET) throw exception ("Invalid Socket");  
  
    struct sockaddr_in sin;  
    memset (&sin, 0, sizeof(sin));  
    sin.sin_family = AF_INET;  
  
    struct hostent *phe = gethostbyname (servername);  
    if (phe == NULL) throw exception ("Invalid host name");  
    memcpy (&sin.sin_addr, phe->h_addr, phe->h_length);  
  
    struct servent *pse = getservbyname (service, "tcp");  
    if (pse == NULL) throw exception ("Invalid Service");  
    sin.sin_port = pse->s_port;  
  
    if (connect (s, (struct sockaddr *) &sin, sizeof(sin)) == SOCKET_ERROR)  
        throw exception ("Connect Error");  
  
    return s;  
}
```

```
void Reader_TCP (SOCKET s, string & Input) {
    char buffer[256];
    char *tmp = buffer;
    int bufferlen = sizeof(buffer);
    int len;

    do {
        len = recv (s, tmp, bufferlen, 0);
        tmp += len;
        bufferlen -= len;
    } while (len != SOCKET_ERROR && len != 0 && bufferlen != 0);

    if (len == SOCKET_ERROR)
        throw exception ("Reader Error");
    Input.assign (buffer, tmp - buffer);
}

void Writer_TCP (SOCKET s, const string & text) {
    int len = send (s, text.data(), text.size(), 0);
    if (len == SOCKET_ERROR)
        throw exception ("Writer Error");
}
```

```
static void TCP_Client () {  
    SOCKET s = Create_Socket_Client_TCP ("localhost", "hmne");  
  
    string TextInput;  
    Reader_TCP (s, TextInput);  
    cout << "Receive Text " << TextInput << endl;  
  
    closesocket (s);  
}
```

```
SOCKET Create_Socket_Server_TCP (short service) {  
    struct protoent *ppe = getprotobyname("tcp");  
    if (ppe == NULL) throw exception ("Illegal Protocol");  
  
    SOCKET s = socket (AF_INET, SOCK_STREAM, ppe->p_proto);  
    if (s == INVALID_SOCKET) throw exception ("Invalid Socket");  
  
    struct sockaddr_in sin;  
    memset (&sin, 0, sizeof(sin));  
    sin.sin_family = AF_INET;  
    sin.sin_addr.s_addr = INADDR_ANY;  
    sin.sin_port = htons (service);  
  
    if (bind (s, (struct sockaddr *) &sin, sizeof(sin)) == SOCKET_ERROR)  
        throw exception ("Bind Error");  
  
    if (listen (s, 5) == SOCKET_ERROR)  
        throw exception ("Listen Error");  
  
    return s;  
}
```



```
void main(int argc, char* argv[]) {
    WSADATA data;
    WSAStartup (0x202, &data);
    try {
        SOCKET master = Create_Socket_Server_TCP (5678);

        while (1) {
            SOCKET slave = accept (master, NULL, 0);

            time_t clock_time;
            time (&clock_time );
            struct tm *current_time = localtime( &clock_time );
            cout << asctime (current_time) << endl;

            Writer_TCP (slave, asctime (current_time));
            closesocket (slave);
        }
        closesocket (master);
    }
    catch (exception e) {
        cerr << "SOCKET ERROR : " << e.what() << " # " << WSAGetLastError () << endl;
    }

    WSACleanup ();
}
```

Client :

```
d:\Enseignement\Network\Applications\DayTimeClient\l
What to do : t
Receive Text Wed Sep 25 22:59:35 2002

What to do : t
Receive Text Wed Sep 25 22:59:39 2002

What to do : _
```

Serveur :

```
d:\Enseignement\Network\Applications\DayTimeTCP\D
1032987575
Wed Sep 25 22:59:35 2002

1032987579
Wed Sep 25 22:59:39 2002
```

```
void main(int argc, char* argv[]) {  
    WSADATA data;  
    WSStartup (0x202, &data);  
    try {  
        SOCKET master = Create_Socket_Server_TCP (5678);  
  
        while (1) {  
            SOCKET slave = accept (master, NULL, 0);  
            if (slave == SOCKET_ERROR)  
                throw exception ("accept failed");  
            if (_beginthread ((void (*)(void *)) Service, 0, (void *) slave) < 0)  
                throw exception ("Thread failed");  
        }  
    }  
    catch (exception e)  
    {  
        cerr << "SOCKET ERROR : " << e.what() << " # " << WSAGetLastError () << endl;  
    }  
  
    WSACleanup ();  
}
```

```
static void Service (SOCKET s) {
    try {
        static int thread_id = 1;
        time_t clock_time;
        time (&clock_time );
        struct tm *current_time = localtime( &clock_time );

        cout << "Thread Id " << thread_id++ << endl;
        cout << clock_time << endl;
        cout << asctime (current_time) << endl;

        Writer_TCP (s, asctime (current_time));
    }
    catch (exception e) {
        cerr << "SOCKET ERROR : " << e.what() << " # " << WSAGetLastError () << endl;
    }
    closesocket (s);
}
```


Client :

```
d:\Enseignement\Network\Applications\DayTimeClient\DayTimeClient.exe
What to do : t
Receive Text Wed Sep 25 23:06:27 2002

What to do : t
Receive Text Wed Sep 25 23:06:28 2002

What to do :
```

Serveur :

```
d:\Enseignement\Network\Applications\DayTimeMulti\DayTimeMulti.exe
Thread Id 1
1032987987
Wed Sep 25 23:06:27 2002

Thread Id 2
1032987988
Wed Sep 25 23:06:28 2002
```

```
void Synch_Reader_TCP (SOCKET s, string & Input) {  
    short Size;  
    int len = recv (s, (char *) &Size, sizeof (Size), 0);  
    if (len == SOCKET_ERROR) throw exception ("Reader Error");  
    Size = ntohs (Size);  
    char *buffer = (char *) malloc (Size), *tmp = buffer;  
    int bufferlen = Size;  
  
    while (len != SOCKET_ERROR && bufferlen != 0) {  
        len = recv (s, tmp, bufferlen, 0);  
        tmp += len;  
        bufferlen -= len;  
    }  
    if (len == SOCKET_ERROR) throw exception ("Reader Error");  
    Input.assign (buffer, Size);  
}
```

Longueur	Data
2 octets	L octets

```
void Synch_Writer_TCP (SOCKET s, const string & text) {  
    short Size = htons(text.size());  
    int len = send (s, (char *) &Size, sizeof (Size), 0);  
    if (len == SOCKET_ERROR) throw exception ("Synch Error");  
  
    len = send (s, text.data(), text.size(), 0);  
    if (len == SOCKET_ERROR) throw exception ("Writer Error");  
}
```

```
static void TCP_Sync_Client () {  
    SOCKET s = Create_Socket_Client_TCP ("localhost", "hmne");  
  
    string TextInput;  
  
    Synch_Reader_TCP (s, TextInput);  
    cout << "Receive Text " << TextInput << endl;  
  
    Synch_Writer_TCP (s, "Hello world ");  
  
    Synch_Reader_TCP (s, TextInput);  
    cout << "Receive Text " << TextInput << endl;  
  
    closesocket (s);  
}
```

```
static void Service (SOCKET s) {  
    try {  
        time_t clock_time;  
        time (&clock_time );  
        struct tm *current_time = localtime( &clock_time );  
  
        cout << clock_time << endl;  
        cout << asctime (current_time) << endl;  
  
        Synch_Writer_TCP (s, asctime (current_time));  
  
        string Input;  
        Synch_Reader_TCP (s, Input);  
  
        Synch_Writer_TCP (s, Input + asctime (current_time));  
    }  
    catch (exception e) {  
        cerr << "SOCKET ERROR : " << e.what() << " # " << WSAGetLastError () << endl;  
    }  
    closesocket (s);  
}
```


Client :

```
d:\Enseignement\Network\Applications\DayTimeClient\Debug\DayTime
What to do : s
Receive Text Wed Sep 25 23:38:51 2002

Receive Text Hello world Wed Sep 25 23:38:51 2002

What to do : s
Receive Text Wed Sep 25 23:39:10 2002

Receive Text Hello world Wed Sep 25 23:39:10 2002

What to do : _
```

Serveur :

```
d:\Enseignement\Network\Applications\DayTimeLoop\
1032989931
Wed Sep 25 23:38:51 2002

Hello world
1032989950
Wed Sep 25 23:39:10 2002

Hello world
_
```

```
int main(int argc, char* argv[])
{
    ...
    SOCKET master_tcp = Create_Socket_Server_TCP (5678);
    SOCKET master_udp = Create_Socket_Server_UDP (1234);

    fd_set rfd;
    FD_ZERO (&rfd);

    while (1) {
        FD_SET (master_tcp, &rfd);
        FD_SET (master_udp, &rfd);

        if (select (FD_SETSIZE, &rfd, 0, 0, 0) == SOCKET_ERROR)
            throw exception ("select error");

        if (FD_ISSET (master_tcp, &rfd)) {
            // réponse du serveur sous TCP
        }

        if (FD_ISSET (master_udp, &rfd)) {
            // réponse du serveur sous UDP
        }
    }
    ...
}
```

Client :

```
d:\Enseignement\Network\Applications\DayTimeClient\D
What to do : t
Receive Text Wed Sep 25 23:42:59 2002

What to do : u
Receive Text Wed Sep 25 23:43:01 2002

Receive Long 1032990181
What to do : t
Receive Text Wed Sep 25 23:43:03 2002

What to do : _
```

Serveur :

```
d:\Enseignement\Network\Applications\DayTimeBoth\
TCP
1032990179
Wed Sep 25 23:42:59 2002

UDP
1032990181
Wed Sep 25 23:43:01 2002

TCP
1032990183
Wed Sep 25 23:43:03 2002
```

```
{
    WSADATA data;
    WSAStartup (0x0202, &data);

    CAsyncSocket socket; socket.Create ();

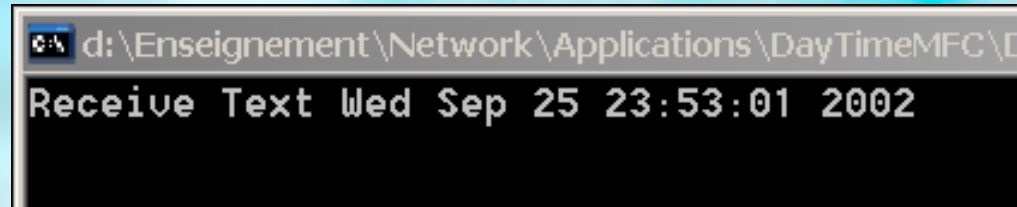
    struct sockaddr_in sin;
    memset (&sin, 0, sizeof(sin));
    sin.sin_family = AF_INET;

    struct hostent *phe = gethostbyname ("localhost");
    if (phe == NULL) throw exception ("Invalid host name");
    memcpy (&sin.sin_addr, phe->h_addr, phe->h_length);

    struct servent *pse = getservbyname ("http", "tcp");
    if (pse == NULL) throw exception ("Invalid Service");
    sin.sin_port = pse->s_port;

    socket.Connect ((struct sockaddr *) &sin, sizeof (sin));
    char Buffer [256];
    int len = socket.Receive (Buffer, sizeof(Buffer), 0);
    Buffer[len] = '\0';
    cout << "Receive Text " << Buffer << endl;

    socket.Close();
    WSACleanup ();
}
```




```
import java.io.*;
import java.net.*;

class DayTimeJava {
    public static void main (String argv[]) {
        try {
            Socket socket = new Socket ("localhost", 5678); //DatagramSocket pour udp
            InputStream input = socket.getInputStream ();
            OutputStream output = socket.getOutputStream ();
            byte [] buffer = new byte [256];

            int length = input.read (buffer, 0, 256);

            System.out.write (buffer, 0, length);
            System.out.println ("");

            socket.close ();
        }
        catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

```
C:\JBuilder6\jdk1.3.1\bin\javaw -classpath
"D:\Enseignement\Network\Applications\DayTimeJava\class
\JBuilder6\jdk1.3.1\jre\lib\jaws.jar;C:\JBuilder6\jdk1.
;C:\JBuilder6\jdk1.3.1\lib\htmlconverter.jar;C:\JBuide
Wed Sep 25 23:59:06 2002
```

Structure du serveur en trois fils d'exécution

Le fil principal regarde le clavier pour la fin

Le fil serveur écoute la socket du www

A la réception d'une requête création d'un fil service

Le fil service créé à la demande exécute la requête

- il doit lire la requête

- il doit identifier la requête (Analyse()->GET/POST/HEAD/...)

- il doit extraire le nom du fichier demandé (GetFilename())

- il doit exécuter la requête prévue (DoGet(), DoHead, ...)

- Puis décède sans attendre permettre de nouvelle requête



Evolutions possibles

ajouter une socket qui lit le clavier et qui traduit les ordres en commande pour le serveur, le serveur se met en écoute multiple.

créer des services dédiés selon la nature de la requête (Get, Post, etc)

```
import java.io.*;

class WebServerApp {
    public static void main (String argv[]) {
        System.out.println ("Web Server Application Started");
        WebServer server = new WebServer (80);
        server.start ();
        try {
            BufferedReader in = new BufferedReader (new InputStreamReader(System.in));
            while (true) {
                System.out.print ("Q to quit : ");
                String buffer;
                buffer = in.readLine();
                if (buffer.length() > 0) {
                    if (buffer.charAt(0) == 'q') {
                        System.out.println ("Web Server Application Halted");
                        System.exit (0);
                    }
                }
            }
        }
        catch (Exception e) {
            e.printStackTrace (System.out);
            System.out.println ("Web Server Application Halted");
        }
    }
}
```



```
import java.io.*;
import java.net.*;

class WebServer extends Thread {
    public WebServer (int _port) {
        Port = _port;
    }

    public void run () {
        try {
            System.out.println ("Web Server Started");
            server = new ServerSocket (Port, 5);
            while (true) {
                Socket worker = server.accept ();
                System.out.println ("Web Service Required");
                WebService service = new WebService (worker);
                service.start ();
            }
        }
        catch (Exception e) { }
        server.close ();
    }

    private int Port;
    private ServerSocket server;
}
```



```
import java.io.*;
import java.net.*;

class WebService extends Thread {
    public WebService (Socket _connection) {
        Connection = _connection;
    }

    private void DoGet (String Filename) throws IOException {
        PrintStream output = new PrintStream (Connection.getOutputStream ());
        output.println ("HTTP/1.0 200 OK");
        output.println ("");
        BufferedReader input = new BufferedReader (new FileReader(new File (Filename)));
        String line;
        while ((line = input.readLine ()) != null) {
            output.println (line);
        }
        input.close ();
    }

    private void DoError () throws IOException {
        PrintStream output = new PrintStream (Connection.getOutputStream ());
        output.println ("HTTP/1.0 404 Not Found");
        output.println ("Content-Type: text/html");
        output.println ("");
        output.println ("<TITLE>Not Found</TITLE><H1> Error 404 - File Not Found </H1>");
    }
}
```

```
public void run () {  
    try {  
        System.out.println ("Web Service Started");  
        BufferedReader input = new BufferedReader (  
            new InputStreamReader(Connection.getInputStream ());  
  
        String request = input.readLine ();  
        switch (Analyse(request))  
        {  
            case GET : DoGet (GetFileName (request)); break;  
            case HEAD : DoHead (); break;  
            case POST : DoPost (); break;  
            default : DoError (); break;  
        }  
        Connection.close ();  
        System.out.println ("Service Ended");  
    }  
    catch (Exception e) {  
        e.printStackTrace();  
    }  
}  
  
private Socket Connection;
```

```
private final int GET  = 1;
private final int HEAD = 2;
private final int POST = 3;
private final int ERROR = 4;
```

```
private int Analyse (String request) {
    try {
        if (request.substring(0, 3).equalsIgnoreCase("GET")) return GET;
        if (request.substring(0, 4).equalsIgnoreCase("HEAD")) return HEAD;
        if (request.substring(0, 4).equalsIgnoreCase("POST")) return POST;
    }
    catch (Exception e) { }
    return ERROR;
}
```

```
private String GetFileName (String request) {
    String Filename;
    try {
        int startindex = request.indexOf (' ')+1;
        if (request.charAt (startindex) == '/') startindex++;
        int endindex = request.indexOf (' ', startindex);
        Filename = request.substring (startindex, endindex);
        if (Filename.equals ("")) Filename = "index.htm";
    }
    catch (Exception e) { Filename = "index.htm"; }
    return Filename;
}
```

Web Server Application Started

Q to quit :

Web Server Started

Web Service Required

Web Service Started

GET /index.htm HTTP/1.1

Accept: image/gif, image/jpeg, */*

Accept-Language: fr

Host: localhost

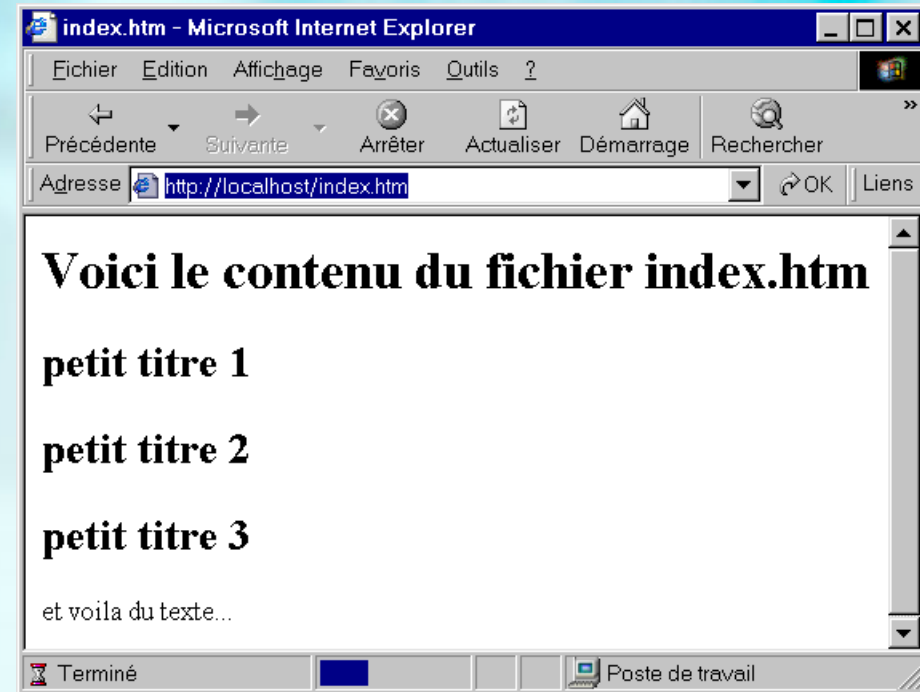
...

Web Service Required

Web Service Started

GET /index.html HTTP/1.1

q



N'existe-t-il pas déjà un protocole adapté à mon application ?

HTTP : les URL c'est plutôt facile à encoder

`http://localhost/rfid.php?state`

`http://localhost/rfid.php?read`

`http://localhost/rfid.php?reset`

FTP, TIME, etc...

Non, alors il faut structurer son application le plus possible selon le mode

Client/Serveur synchrone : le plus facile.

Client/Serveur asynchrone-statefull : un peu moins facile mais dangereux.

Client/Serveur asynchrone-stateless : encore moins facile mais moins dangereux.

Client/Serveur et Serveur/Client : sur la même connexion à éviter !

Client/Serveur synchrone : c'est le plus facile.

Le client envoie une question au serveur et attend la réponse

Le serveur attend une question et fournit une réponse

Client/Serveur asynchrone-statefull : c'est un peu moins facile.

Le client envoie une question (qui pourrait dépendre des questions précédentes) et fait autre chose

Quand il reçoit la réponse, il la comprend grâce à la question sauvegardée

C'est le modèle C/S avec un AFD pour encoder l'état

Le serveur attend une question et fournit une réponse

Client/Serveur asynchrone-stateless : c'est encore moins facile.

Le client envoie une question (qui pourrait dépendre des questions précédentes) et fait autre chose

Quand il reçoit la réponse, il la comprend car les éléments de la question y sont récapitulés

C'est le modèle C/S avec un AFD pour encoder l'état

Le serveur attend une question et fournit une réponse qui inclut les éléments de la question

Niveau le plus bas

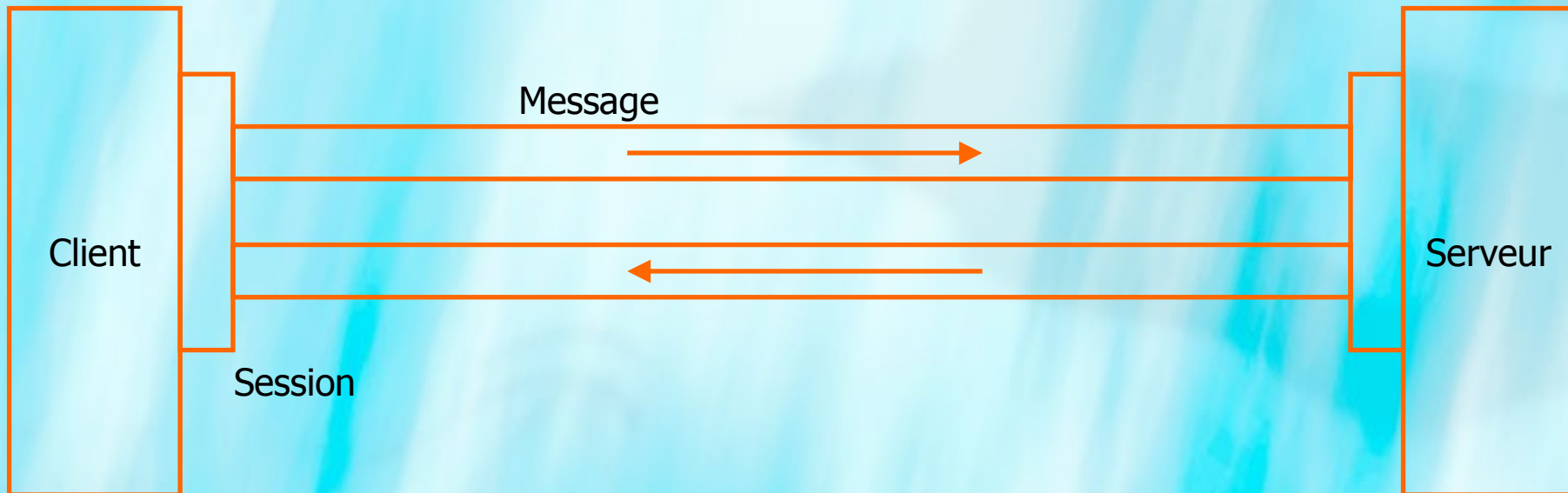
Message représente une question ou une réponse.

Dans l'esprit c'est une entité indissociable (atomique) plutôt petite, mais elle peut aussi contenir un film de 700Mo

Session

C'est l'association de la question et de sa réponse

Il permet de modéliser l'AFD qui est nécessaire quand l'état devient important



Le client et le serveur seront deux applications indépendantes

- 1 fichier commun pour déclarer les éléments commun du protocole

Le client (ou le serveur) doit concevoir des messages (pour les envoyer)

La conception du message consiste à ordonner et choisir une représentation adaptée pour chaque donnée à envoyer

- Entier (byte, short, long, ...)

- String (longueur+texte ou texte +marqueur)

- Tableau (longueur+cellules ou ...)

- Boolean (8, 16 ou 32 bits)

- Discriminant (texte, int, ...)

- 1 classe pour construire un message à partir des données internes

- 1 méthode pour chaque type de message

- 1 classe pour extraire d'un message des données internes

- 1 méthode pour chaque type de message

Le client (ou le serveur) doit manipuler des messages

- 1 classe session pour lire/analyser/calculer/envoyer un message

- 1 méthode pour chaque question/réponse

Conception d'un protocole C/S

121

TP Multimédia 2007-2008

La classe protocole contient

Le numéro du port

Les codes des messages échangés entre partenaires

Aurait pu contenir le nom du serveur, ...

La classe MediaWriter contient

Toutes les méthodes pour encoder les messages

Expédier le message

Des méthodes internes pour organiser l'encodage

Protocol

```
PORT_ID
QUERY_START
QUERY_ANIMATION_LIST
QUERY_ROLE_LIST
QUERY_JOIN
QUERY_LEAVE
...
SUCCESS_START
SUCCESS_ANIMATION_LIST
...
FAIL_START
FAIL_ANIMATION_LIST
...
```

MediaWriter

```
- OutputStream outputStream
- ByteArrayOutputStream baos
- DataOutputStream output

+ createQueryAnimationList ()
+ createQueryRoleList(String)
+ createQueryRessourceList (String, String)
+ createReplyAnimationList(List<String>)
+ createFailedGetRessource(URL)
+ send()
- writeInt (int)
- writeInt (URL)
```

Détails des méthodes de la classe MediaWriter

```
public void createQueryAnimationList() {  
    writeInt (Protocol.QUERY_ANIMATION_LIST);  
}
```

```
public void createQueryRoleList(String animationName) {  
    writeInt (Protocol.QUERY_ROLE_LIST);  
    writeUTF (animationName);  
}
```

```
public void createQueryRessourceList(String animationName, String roleName) {  
    writeInt (Protocol.QUERY_RESSOURCE_LIST);  
    writeUTF (animationName);  
    writeUTF (roleName);  
}
```

```
private void writeInt (int v) {  
    try { output.writeInt(v); } catch (IOException e) { e.printStackTrace(); }  
}
```

```
public void send() {  
    byte [] message = baos.toByteArray();  
    try { outputStream.write(message); outputStream.flush(); } catch (IOException e) { ... }  
}
```

La classe `MediaReader` contient

Réceptionner le message

Toute les méthodes pour décoder les messages

Des méthodes internes pour organiser l'encodage

MediaReader

- `InputStream inputStream`
- `DataInputStream input`

- + `getType () : int`
- + `getParams () : String []`
- + `receive ()`
- `readInt () : int`
- `readUTF () : String`
- ...

Détails des méthodes de la classe MediaWriter

```
private void fillParams(int count) {  
    params = new ArrayList<String> (count);  
    for (int i=0; i<count;++i) params.add(readUTF ());  
}  
  
private int readInt () {  
    try { return inputStream.readInt(); } catch (IOException e) { return 0; }  
}  
  
public void receive() {  
    type = readInt ();  
    switch (type) {  
        case Protocol.QUERY_ANIMATION_LIST : break;  
        case Protocol.QUERY_ROLE_LIST : fillParams (1); break;  
        case Protocol.QUERY_RESSOURCE_LIST : fillParams (2); break;  
        case Protocol.QUERY_JOIN : fillParams (2); break;  
        case Protocol.SUCCESS_ANIMATION_LIST : fillParams(); break;  
        case Protocol.FAIL_ANIMATION_LIST : break;  
        ...  
    }  
}
```


Usage de ces classes :

Une méthode interne du client qui demande la liste des rôles pour une animation donnée

```
protected List<String> getRoleList (String animationName) {  
    try {  
        if (connection == null) connect();  
        MediaWriter writer = new MediaWriter (connection.getOutputStream());  
        writer.createQueryRoleList(animationName);  
        writer.send ();  
        MediaReader reader = new MediaReader (connection.getInputStream());  
        reader.receive ();  
        List<String> reply = null;  
        if (reader.getType() == Protocol.SUCCESS_ROLE_LIST) {  
            reply = reader.getParams();  
        }  
        return reply;  
    } catch (Exception e) {  
        e.printStackTrace();  
        return null;  
    }  
}
```

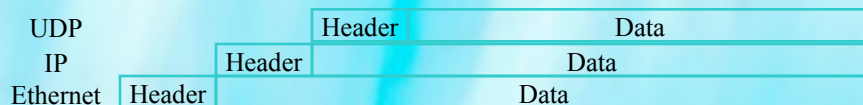
Usage de ces classes :

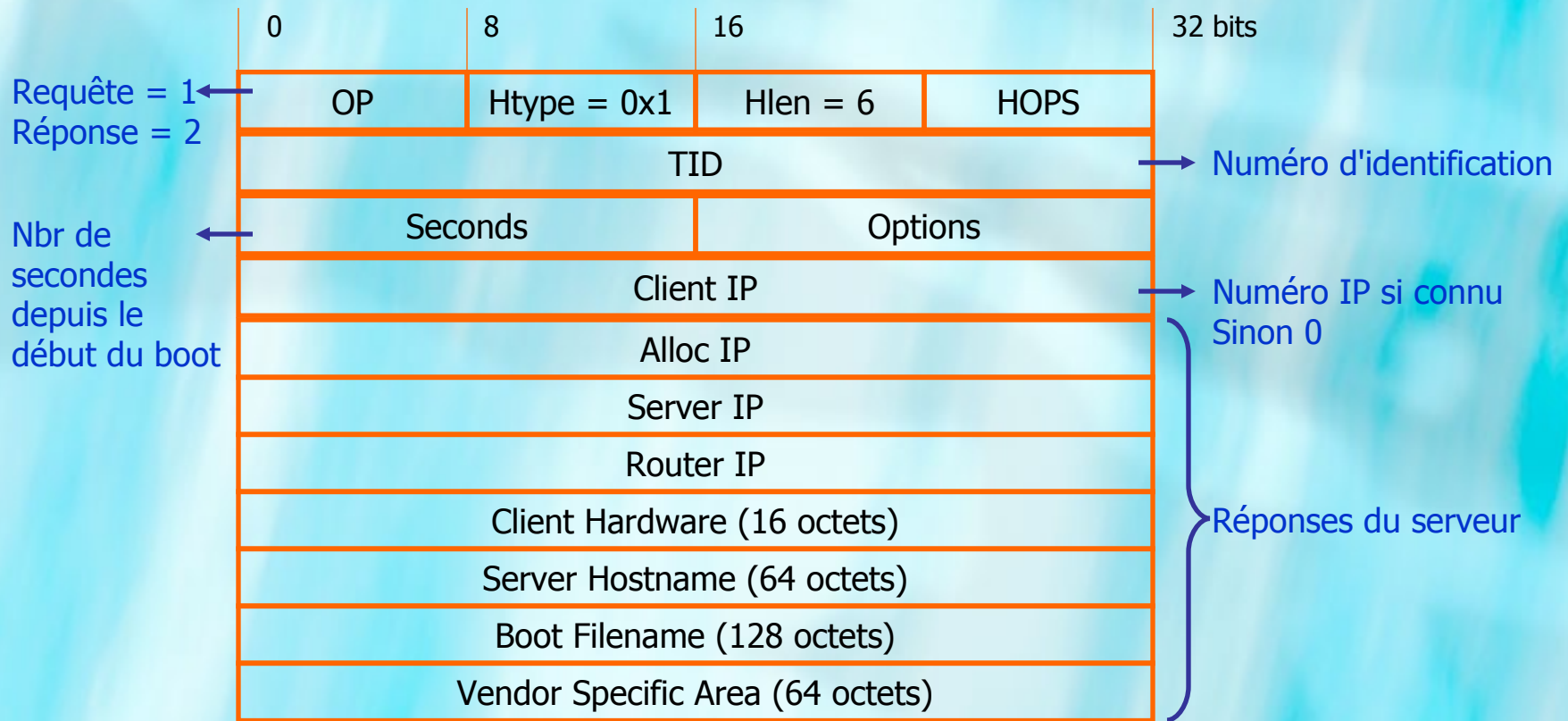
Une méthode interne du serveur qui renvoie la liste des rôles pour une animation donnée

```
public void run() {  
    boolean loop = true;  
    while (loop) {  
        MediaPlayer reader = new MediaPlayer (connection.getInputStream());  
        reader.receive ();  
        process (reader);  
    }  
}  
  
private void process(MediaReader reader) {  
    switch (reader.getType()) {  
        case Protocol.QUERY_ROLE_LIST: processRoleList(reader.getParam(0)); break;  
        ...  
        default : processError (); break;  
    }  
}  
  
private void processRoleList(String animationName) {  
    List<String> reply = document.getRoleList(animationName);  
    MediaWriter writer = new MediaWriter (connection.getOutputStream());  
    writer.createReplyRoleList(reply);  
    writer.send();  
}
```

- BOOT Protocol - Dynamic Host Configuration Protocol
- Trivial File Transfert Protocol
- Domain Name Service
- Simple Network Management Protocol
- Network Time Protocol - Simple Network Time Protocol
- Point to Point Protocol
- Network Adresse Translation
- Mobile IP (Nemo)

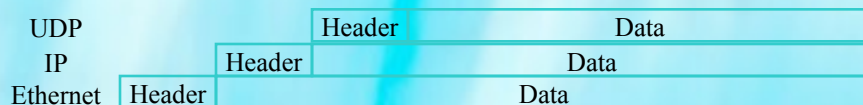
- Boot Protocol (BOOTP) est un protocole conçu initialement pour des machines sans disque ou des terminaux.
- Initialement défini par le RFC 951, puis modifié et enfin clarifié par le RFC 1542
- Ce protocole permet de communiquer automatiquement un numéro IP, mais également les autres paramètres d'une configuration réseau.
- Le protocole est basé UDP.
- Requête en broadcast, puis réponse en unicast si l'adresse MAC est connue sinon en broadcast
- Le client est chargé de gérer :
 - les problèmes de connexion
 - de vérifier la réponse





- Vendor Specific Area est une zone libre qui permet de transmettre des données quelconques.
- Il est recommandé de remplir les 4 premier octets d'un chiffre magique. Si on en a pas, il faut mettre 99.130.83.99 puis 255 et finir avec des 0

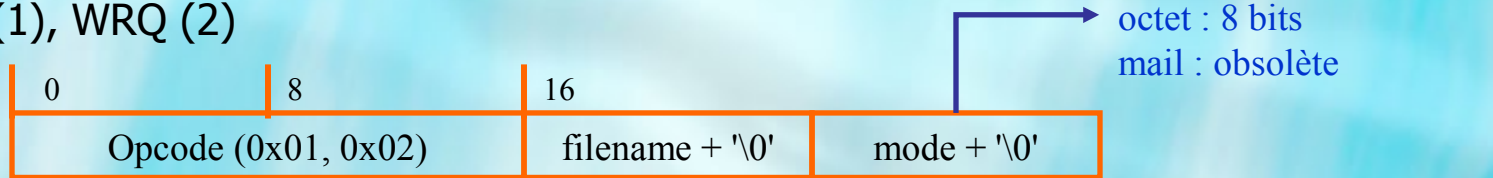
- TFTP est un protocole de transfert de fichier très simple :
- Il est utilisé (principalement) entre deux machines pour télécharger un noyau d'exécution
 - Terminaux X
 - Serveur d'impression
- Il n'y a pas d'identification ni d'authentification des intervenants car ce sont des machines
- Il n'y a que deux commandes :
 - RRQ (Read Request)
 - WRQ (Write Request)
- Le protocole est au dessus de UDP (port 69)



- Un fichier est découpé en bloc de 512 octets
- Associe un numéro de bloc à chaque bloc de 512 octets
- Après chaque émission attends un accusé de réception
- Si accusé valide continue jusqu'à la fin du fichier
- Si erreur ferme la communication
- Si timeout pendant attente ferme la communication

- Il y a 5 paquets TFTP

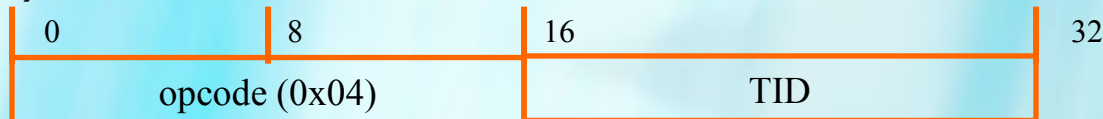
- RRQ (1), WRQ (2)



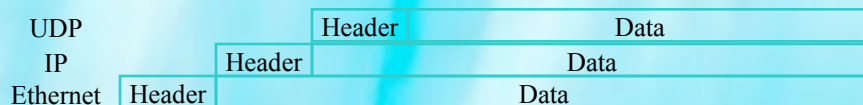
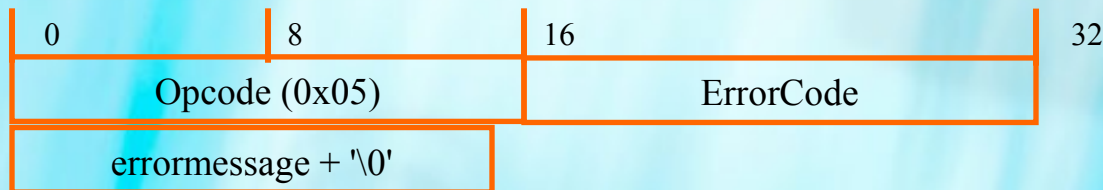
- DATA (3)



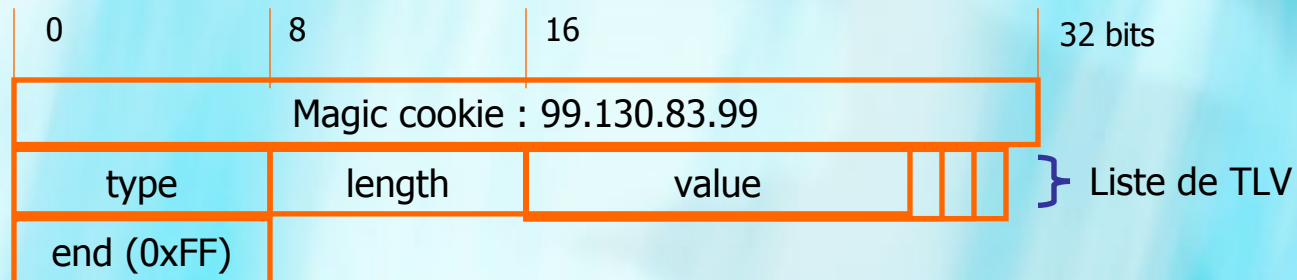
- ACK (4)



- ERROR (5)



- DHCP est défini par le RFC 1541 et ... et terminé par le RFC 2132
- DHCP permet de louer des adresses IP pour une certaine durée
- DHCP est implémenté à l'aide du protocole BOOTP
- DHCP utilise la zone Vendor Specific Area pour transmettre ses données.
- Structure de la zone Vendor Specific Area



- Type : type d'une opération 1..254, 0 = padding, 255 = fin de liste
- Length : longueur du champ Value
- Value : une valeur quelconque à transmettre

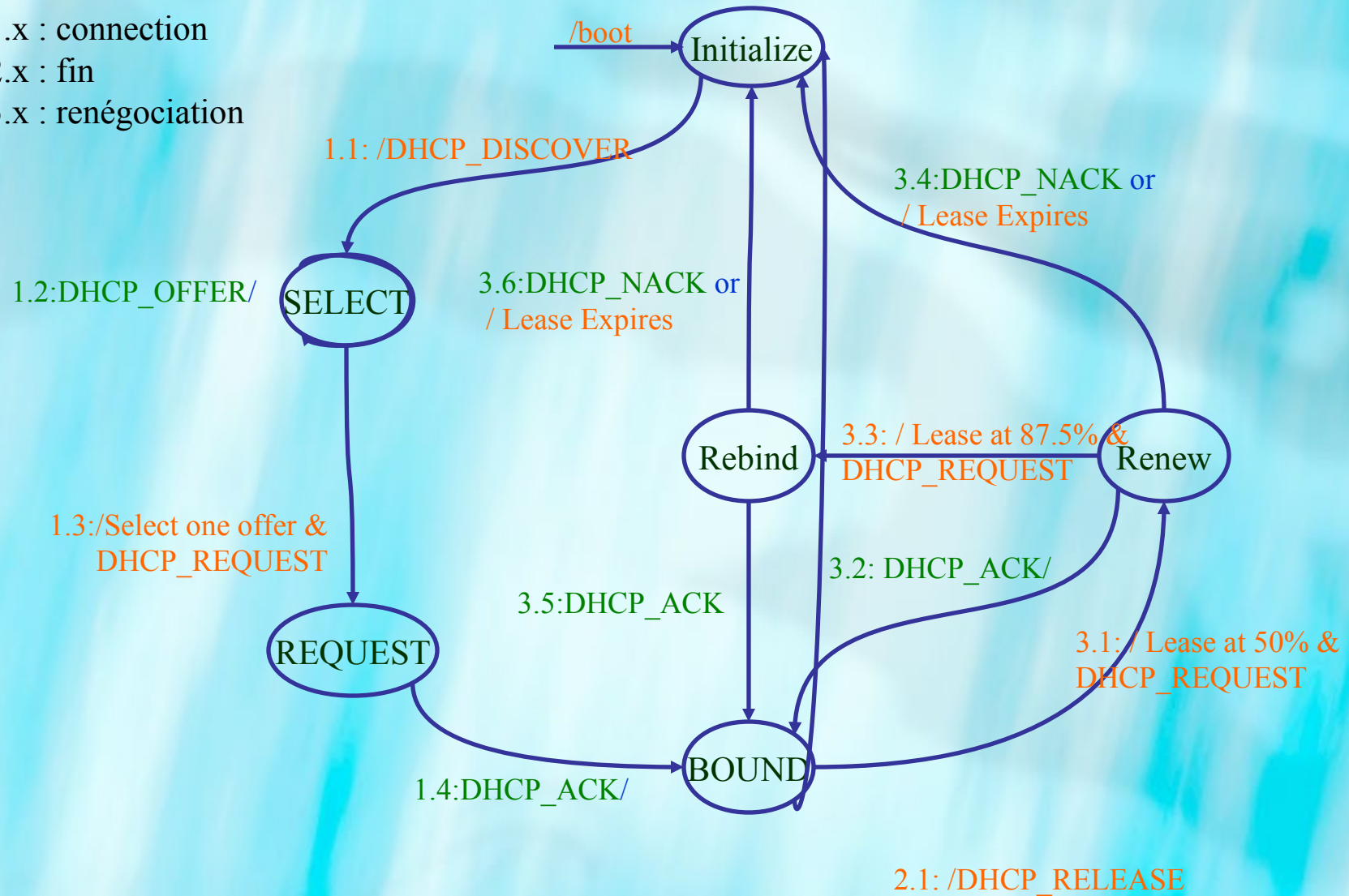
- Le RFC 2132 donne une liste particulièrement longue des types
- DHCP est juste un cas particulier
 - Type = 53
 - Length = 1
 - Value :
 - 1=DHCP_DISCOVER 5=DHCP_ACK
 - 2=DHCP_OFFER 6=DHCP_NACK
 - 3=DHCP_REQUEST 7=DHCP_RELEASE
 - 4=DHCP_DECLINE 8=DHCP_INFORM
 - Ces 8 valeurs permettent de négocier le bail
 - Le bail est re-négociable régulièrement

server / client

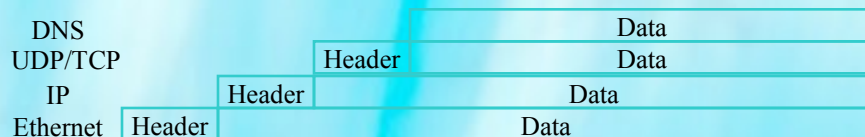
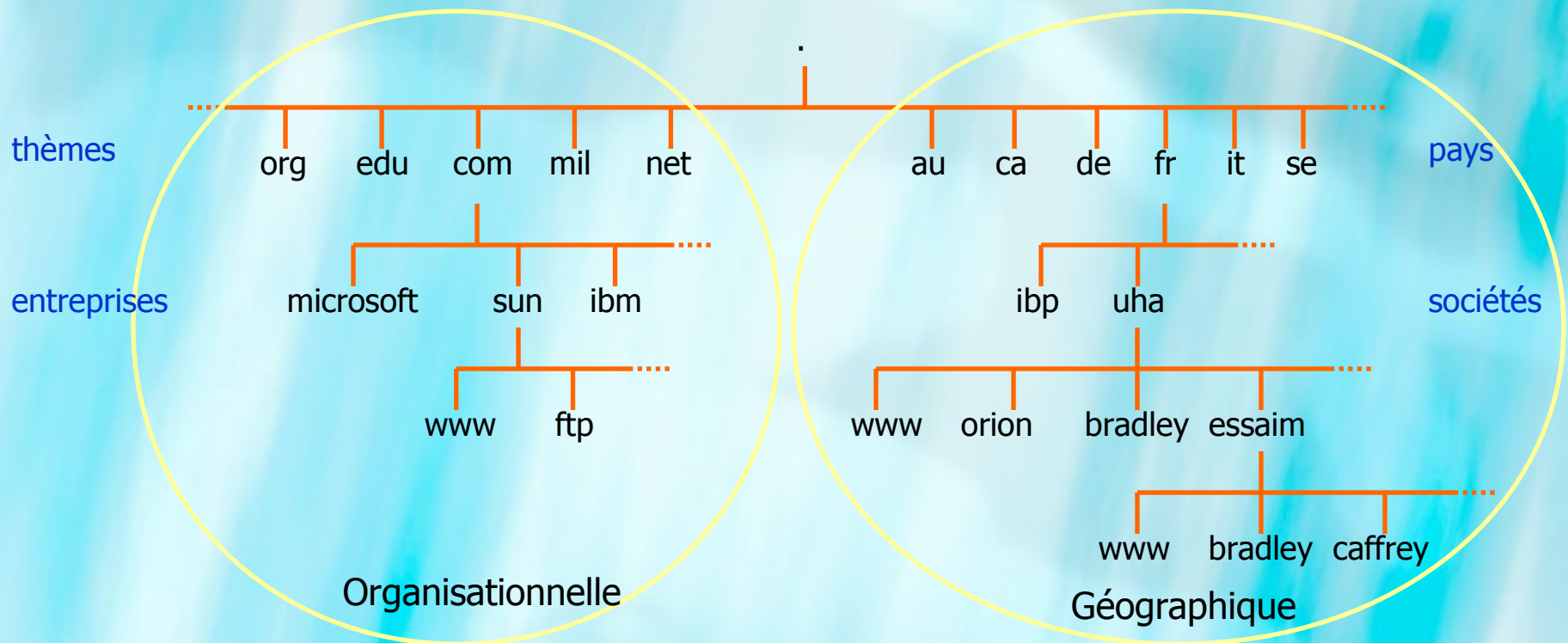
1.x : connection

2.x : fin

3.x : renégociation



- Domain Name Server est un protocole hiérarchique qui permet d'associer un nom à un numéro IP (et inversement)
- Son organisation est décrit par le RFC 1591



- Chaque entité est appelée une zone
- Une zone possède un DNS qui peut être considérée comme complètement indépendant des autres pour ce qui le concerne
- Pour donner l'adresse d'une machine asimov dans la zone uha de la zone fr
 - En absolu, il faut utiliser : asimov.uha.fr
 - En relatif et dans la zone uha.fr : asimov
 - En relatif dans une autre zone : impossible
- La zone est quand même relié à ses proches voisins
 - Le domaine père (uha.fr doit connaître fr)
 - Les domaines fils (fr doit connaître uha)
- Ces liens sont nécessaires quand il faut résoudre une adresse en dehors de la zone exemple :
 - asimov.uha.fr demande www.microsoft.com

- La machine asimov.uha.fr demande l'adresse IP de `www.microsoft.com`
- Une solution pour trouver cette adresse :
 - asimov interroge uha.fr à propos de `www.microsoft.com`
 - uha.fr interroge fr à propos de `www.microsoft.com`
 - fr interroge fr à propos de `com`
 - fr interroge com à propos de `www.microsoft.com`
 - com interroge microsoft à propos de `www`
 - microsoft retourne l'adresse IP de `www`
 - com, puis fr, puis uha.fr retourne l'adresse IP de `www.microsoft.com`
 - asimov reçoit l'adresse IP de `www.microsoft.com`
- Ce n'est pas la seule solution pour déterminer l'adresse
 - Soit c'est le demandeur qui découpe et récupère les morceaux de requête pour obtenir en définitive la valeur cherché
 - Soit c'est les serveurs qui se passent la requête et font ce qu'ils peuvent

- Recherche longue, itérative et répétitive :
 - Imaginer la séquence pour : www.drtgeii.essaim.uha.fr
- Il y a une mise en cache des associations
- Il faut gérer le cache à chaque niveau
- Il faut ajouter les nouvelles associations mais aussi retirer les plus anciennes (moins fiables)
- Et que faire si un serveur DNS tombe en panne. Il faut mettre en place une redondance (en principe 3 serveurs)
- PS : si le cache ARP ne répond pas car l'adresse ethernet du serveur n'est pas connu, il y a une requête ARP avant chaque requête DNS...

- La base de données DNS est structurée en enregistrements de types différents
 - A Host's IP address
 - PTR Host's domain name, host identified by its IP address
 - CNAME Host's canonical name, host identified by an alias domain name
 - MX Host's or domain's mail exchanger
 - NS Host's or domain's name server(s)
 - SOA Indicates authority for the domain
 - TXT Generic text record
 - SRV Service location record
 - RP Responsible person

 - AAAA Host's IPv6 address

- Enregistrement Start Of Authority (SOA)

- `essaim.uha.fr. IN SOA nsp.essaim.uha.fr hassen.mailhost.essaim.uha.fr.(`
- `1567 ;Serial`
- `18000 ;Refresh after 5 hours`
- `3600 ;Retry after 1 hour`
- `604800 ;Expire after 1 week`
- `86400 ;Minimum TTL of 1 day`
- `)`

- Explications :

- Domaine : `essaim.uha.fr`
- Serveur primaire : `nsp.essaim.uha.fr`
- Coupable : `hassen@mailhost.essaim.uha.fr.`
- (...) : informations de datation pour les caches et secondaires

- Name Server (NS)
 - `essaim.uha.fr. IN NS nsp.essaim.uha.fr.`
 - `essaim.uha.fr. IN NS nss1.essaim.uha.fr.`
 - `essaim.uha.fr. IN NS nss2.essaim.uha.fr.`

- Mail Exchanger (MX)
 - `essaim.uha.fr. IN MX 5 mailhost.essaim.uha.fr.`
 - `essaim.uha.fr. IN MX 15 mailhost2.essaim.uha.fr.`
 - `essaim.uha.fr. IN MX 10 mailhost3.essaim.uha.fr.`

- Explications :
 - Domaine : `essaim.uha.fr`
 - Serveurs de noms :
 - `nsp.essaim.uha.fr`
 - `nss1.essaim.uha.fr`
 - `nss2.essaim.uha.fr`
 - Serveurs de mails :
 - `mailhost.essaim.uha.fr` (en premier)
 - `mailhost2.essaim.uha.fr` (en fin)
 - `mailhost3.essaim.uha.fr` (en deuxième)

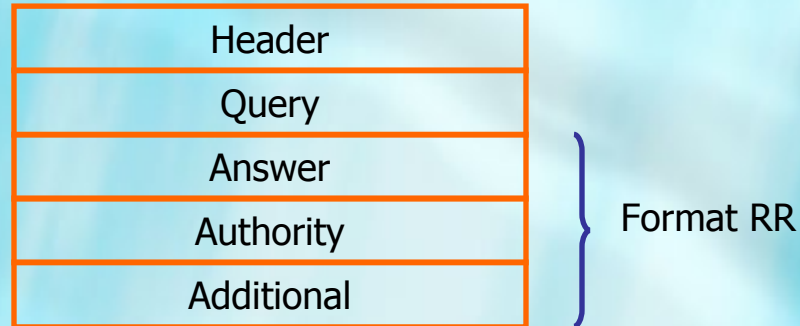
- Adress Record (A)
 - asimov.essaim.uha.fr. IN A 10.58.17.3
 - chanterelle.essaim.uha.fr. IN A 10.56.66.1
 - nsp.essaim.uha.fr. IN A 10.0.9.10
 - nss1.essaim.uha.fr. IN A 10.0.9.11
 - nss2.essaim.uha.fr. IN A 10.0.9.12

- Explications :
 - La machine asimov dans le domaine essaim.uha.fr a l'adresse IP 10.58.17.3

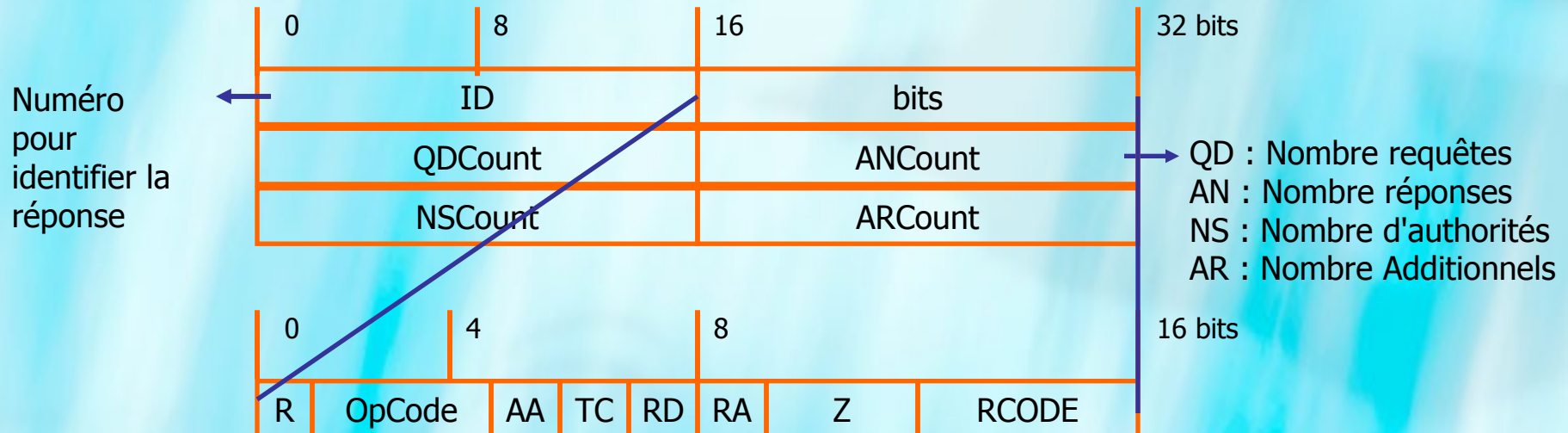
- Canonical Name (CNAME)
 - www.essaim.uha.fr. IN CNAME asimov.essaim.uha.fr.
 - ftp.essaim.uha.fr. IN CNAME chanterelle.essaim.uha.fr.

- Explications :
 - La machine www dans le domaine essaim.uha.fr est aussi la machine asimov dans le même domaine.

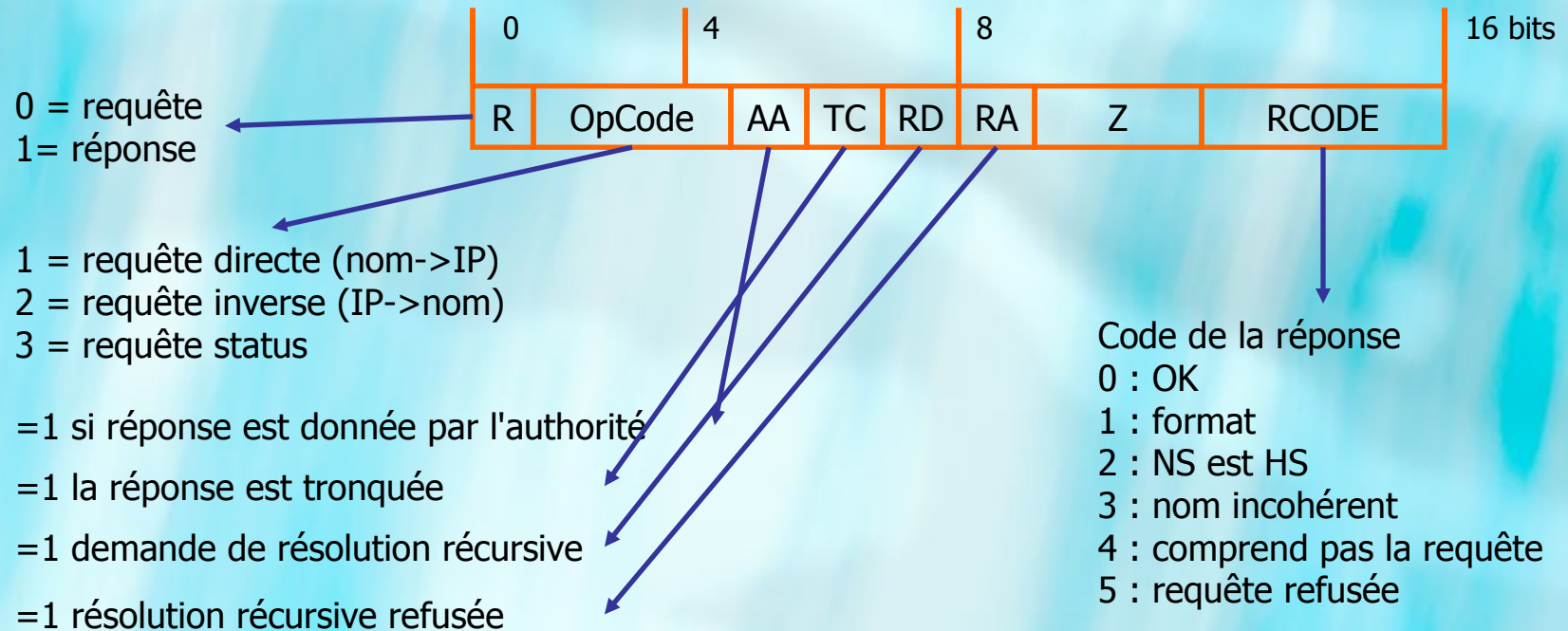
- 5 champs dans un message



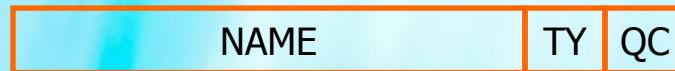
- Structure du header



• Les bits de l'entête



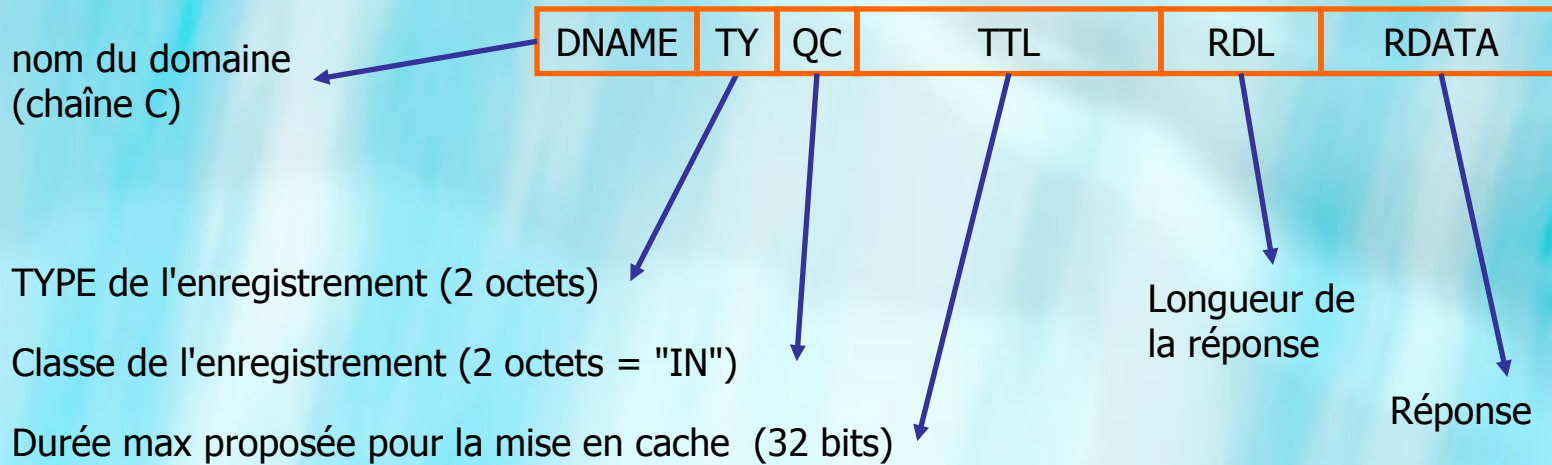
• La requête



NAME : le nom que l'on cherche à identifier au format C

TY = 2 octets représente un type d'enregistrement
QC = 2 octets qui ont la valeur "IN"

- Les réponse au format Resource Record (RR)



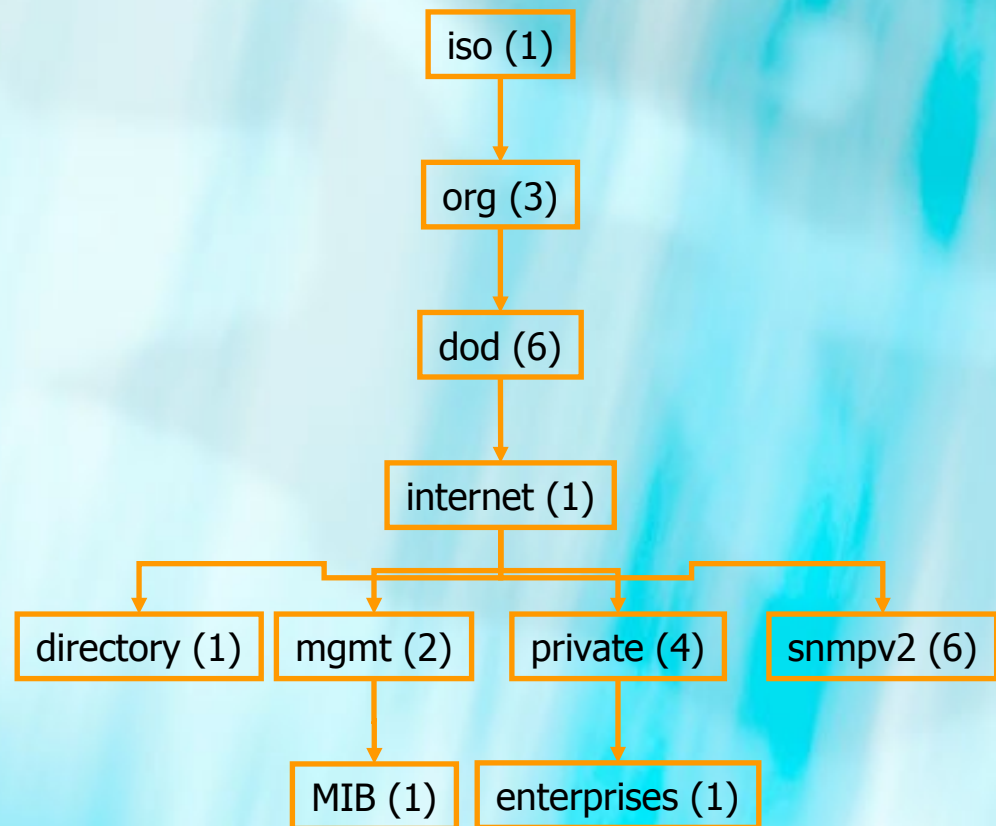
Si demande adresse IP de asimov.uha.fr reçoit un

- TYPE = "A"
- QC = "IN"
- RDL = 4
- RDATA = %0A%3A0%11%03

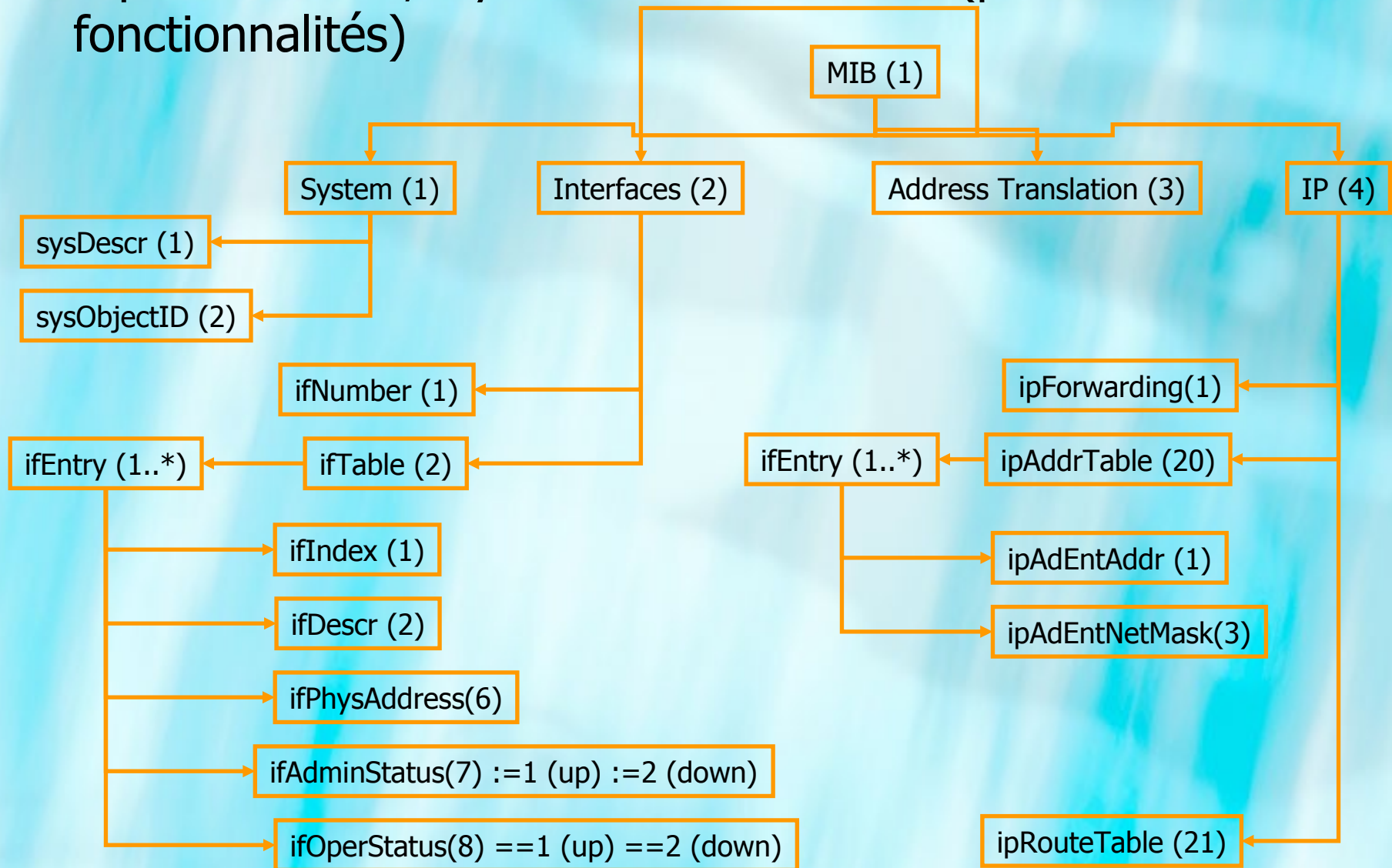
- SNMP - RFC 1065, 1098, ..., 190[2..7]
- SNMP est prévue pour administrer les machines du réseau (routeur, switch, hub, firewall, ...)
- Chaque machine est supervisée par un agent qui stocke les informations dans une MIB (Management Information Base)
- SNMP propose de
 - Lire et d'écrire des données à distance
 - Gérer des alarmes (trap)
 - Piloter des droits d'accès aux données
- L'agent distant écoute le port 161 pour les requêtes
- Le manager écoute le port 162 pour les alarmes

- Le réseau qui pilote le réseau (attention aux hackers !)
- Le manager SNMP place les agents dans des communautés qui ont des droits d'accès différents
 - Public (communauté en lecture seule)
 - Private (communauté en lecture /écriture)
 - XXX (communauté autre)
- Public et Private sont des communautés par défaut
- Les hackers les connaissent, il est recommandé de ne pas les utiliser (no access), mais de créer des communautés avec des noms exotiques...

- Deux types de données
 - Variable simple
 - Table de variables
- Les données sont classées dans un système hiérarchique
- Un agent en clair
 - org.dod.internet.mgmt.mib
- Le même en SNMP
 - 1.3.6.1.2.1



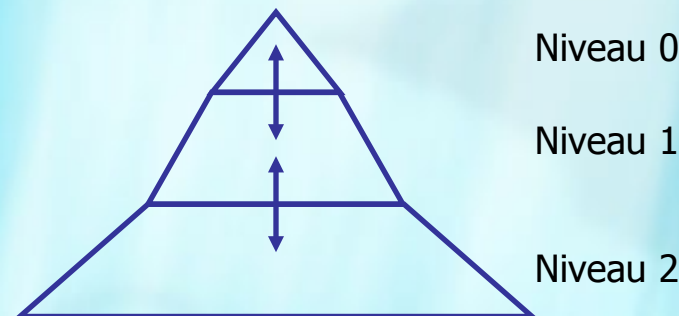
- A partir de MIB, il y a encore des arbres (pour toutes les fonctionnalités)



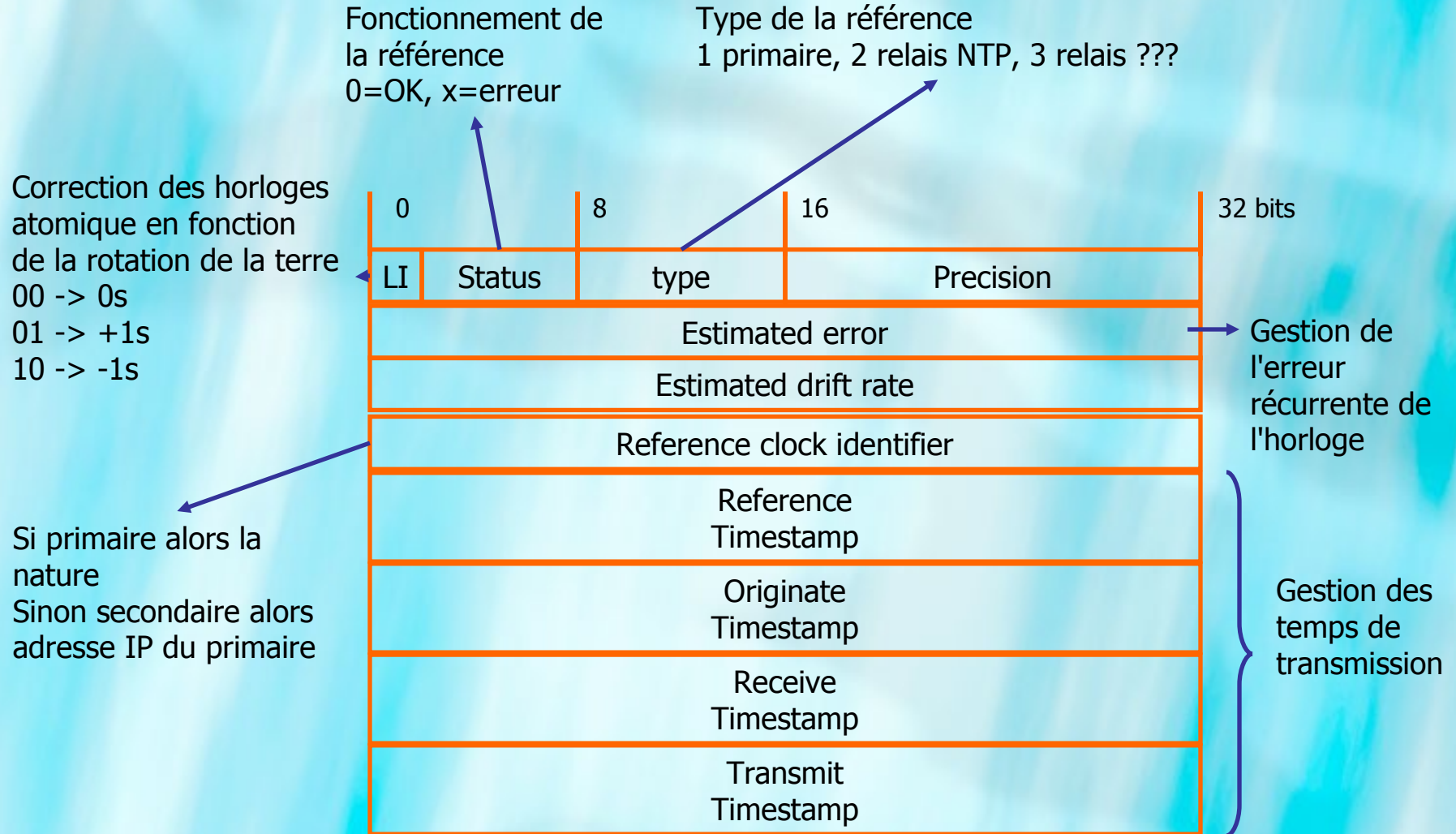
- Logique d'accès à une variable simple :
 - Trois chaînes de numéros :
 - Numéro de MIB : 1.3.6.1.2.1
 - Puis l'adresse de la variable : 1.1 (System.sysDescr)
 - Puis l'identification d'une variable simple : 0
- Logique d'accès à une variable indexée dans une table :
 - Quatre chaînes de numéros :
 - Numéro de MIB : 1.3.6.1.2.1
 - Puis l'adresse de la variable : 4.20 (IP.ipAddrTable)
 - Puis l'identification de l'index : 1..*
 - Puis l'identification du champs : 1 (ipAdEntAddr)
- Les interfaces de commandes semblent savoir travailler sur les deux plans simultanément (textes et nombres)

- Il y a beaucoup de RFC pour décrire les MIB de chaque support
 - ATM (RFC 2515)
 - DecNet (RFC 1559)
 - AppleTalk (RFC 1742)
 - HTTP (RFC 2964)
 - Compte (RFC 2975)
- Ce protocole de contrôle (SNMP) est fortement implanté.
- Ces OID reviennent dans d'autres domaines...

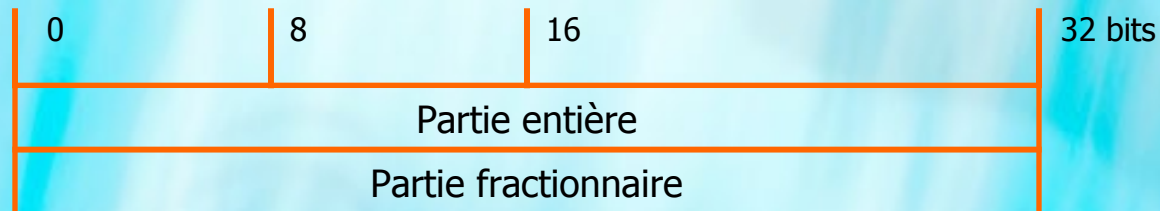
- L'heure exacte est une donnée très importante pour les ordinateurs
 - date de dernière modification dans le cadre de développement coopératif
 - date d'un mail
 - fonction d'exécution automatique (scheduler, crontab, etc)
- Un système de mise à jour de l'heure à partir d'une horloge de référence est en place
- Le système fonctionne sur le principe de strates



- Et chaque entité utilise NTP pour communiquer

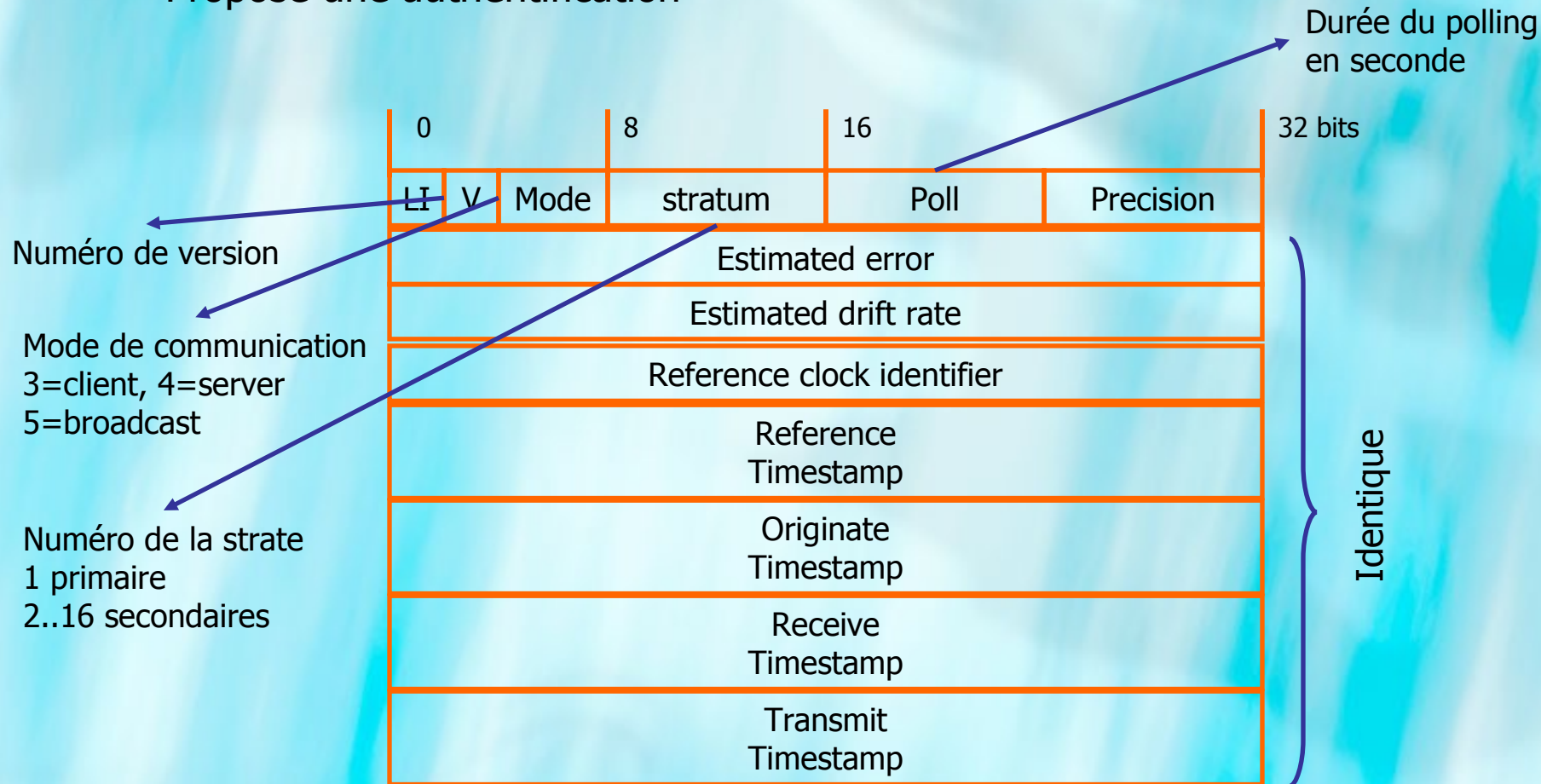


- Il y a 4 temps dans le message
 - Reference timestamp
 - Temps de la dernière synchronisation par le serveur
 - Originate timestamp
 - Temps au moment du départ du message de requête du client
 - Receive timestamp
 - Temps au moment de la réception du message de requête par le serveur
 - Transmit timestamp
 - Temps au moment de la transmission du message de réponse par le serveur
- Format d'un temps
 - Référence : 1 janvier 1970 à 0h 0m 0s
 - Base seconde, maximum : année 2036, minimum : 0.2 ns

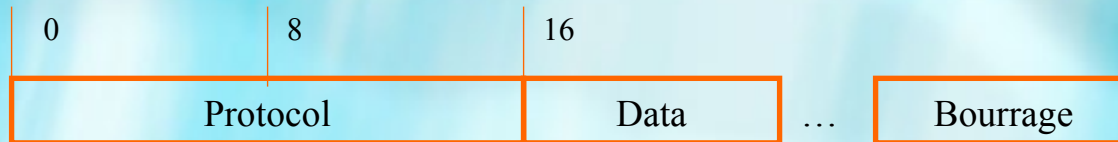


- SNTP est une évolution de NTP

- Intègre plus de strates (16)
- Propose une authentification



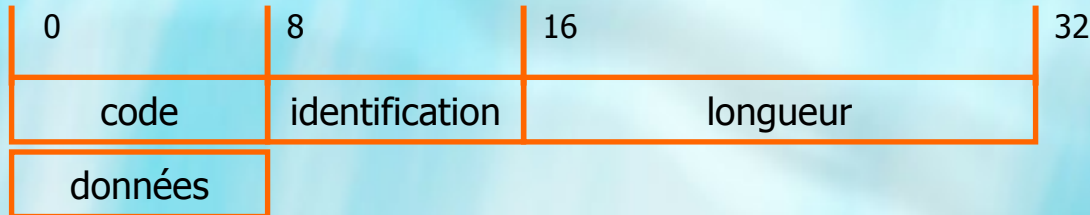
- Point to Point Protocol remplace avantageusement SLIP
- Permet de faire du tunnel, il encapsule un protocole très simplement une trame PPP



- Le numéro du protocole est ... étrange
 - Le LSB du LSO doit être à 1 (impaire)
 - Le LSB du MSO doit être à 0
- Pour un protocole de communication, le champs Data est tout simplement le datagramme du protocole initial
- Bourrage est constitué d'octets pour faire de l'alignement.

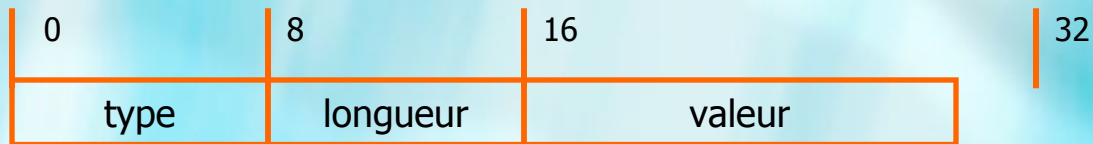
- Bien entendu il y a des numéros de protocole réservés
 - Il y a plusieurs familles de protocole les plus importants :
 - C021 : Link Control Protocol (LCP)
 - C023 : Password Authentication Protocol (PAP)
 - 8021 : IP (Network) Control Protocol (IPCP)
 - 80FB : Compression (Network) Control Protocol (CCP)
 - ... : Network Control Protocol
 - 0021 : IP (une trame IP encapsulée)
- Fonctionnement d'une liaison PPP
 - Configuration du lien négocié à la connexion (paquets LCP)
 - Configuration du réseau négocié à la connexion (paquets NCP)
 - Émission des paquets NCP
 - Puis terminaison (paquets LCP) ou arrêt brutal
- Structure des protocoles de réseau (négociations)
 - Tous les protocoles encapsulés par PPP ont la même structure que LCP (PAP, IPCP, CCP)
 - Le numéro et la signification des codes changent pour chaque protocole
- Structure du protocole IP (Data = trame IP initiale)

- Un paquets LCP a la forme suivante



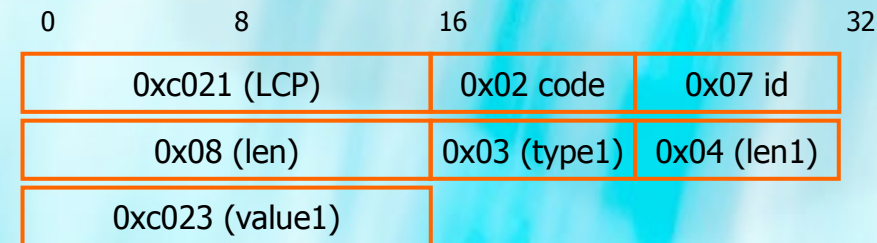
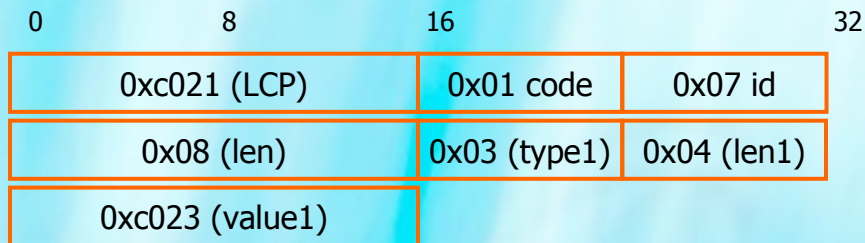
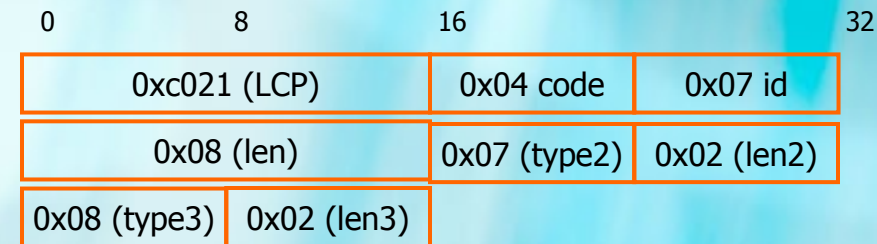
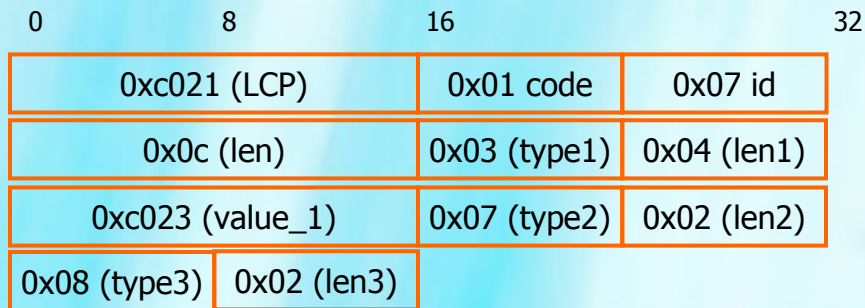
- code :
 - 1 : configure-request : demande de configuration, plusieurs options peuvent être négociées dans la même requête
 - 2 : configure-ack : demande de configuration acceptée, toutes les options négociées sont acceptées
 - 3 : configure-nack : demande de configuration non acceptable, certaines options ne peuvent être exploitées
 - 4 : configure-reject : demande de configuration non acceptable, certaines options ne sont pas négociables
 - 5 : terminate-request : demande de fin de communication
 - 6 : terminate-ack : demande de fin de communication acceptée
 - 7 : code-reject : le code LCD est incompréhensible
 - ...
- identification :
 - Numéro unique qui permet d'identifier la requête et la réponse
- longueur :
 - Nombre d'octets du paquet LCP au complet ! (minimum =4)
- données :
 - Les données : 0 .. longueur-4

- Le champs donnée devient un champs d'option pour la négociation dans le cadre du protocole LCP
- Les options sont au format TLV

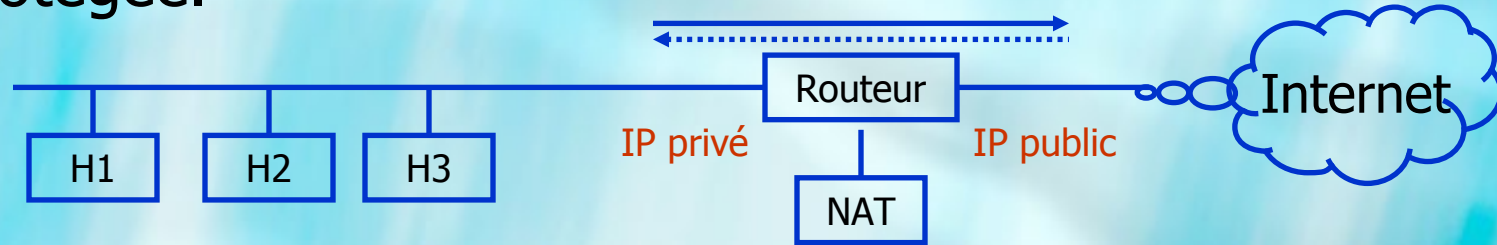


- Type : le numéro de l'option à négocier 1..254
 - Longueur : la longueur complète (minimum = 2)
 - Valeur : les données complémentaires
-
- La liste exhaustive des options n'est pas essentielle

- Exemple de négociation avec des trame LCP
 - Négociation des options : 3 (authentification) , 7 (compression champ protocole) et 8 (compression adresse et contrôle)
 - Rejet des options 7, 8
 - Renégociation de l'option : 3
 - Acceptation de option 3



- NAT est un protocole qui permet de créer une zone protégée.



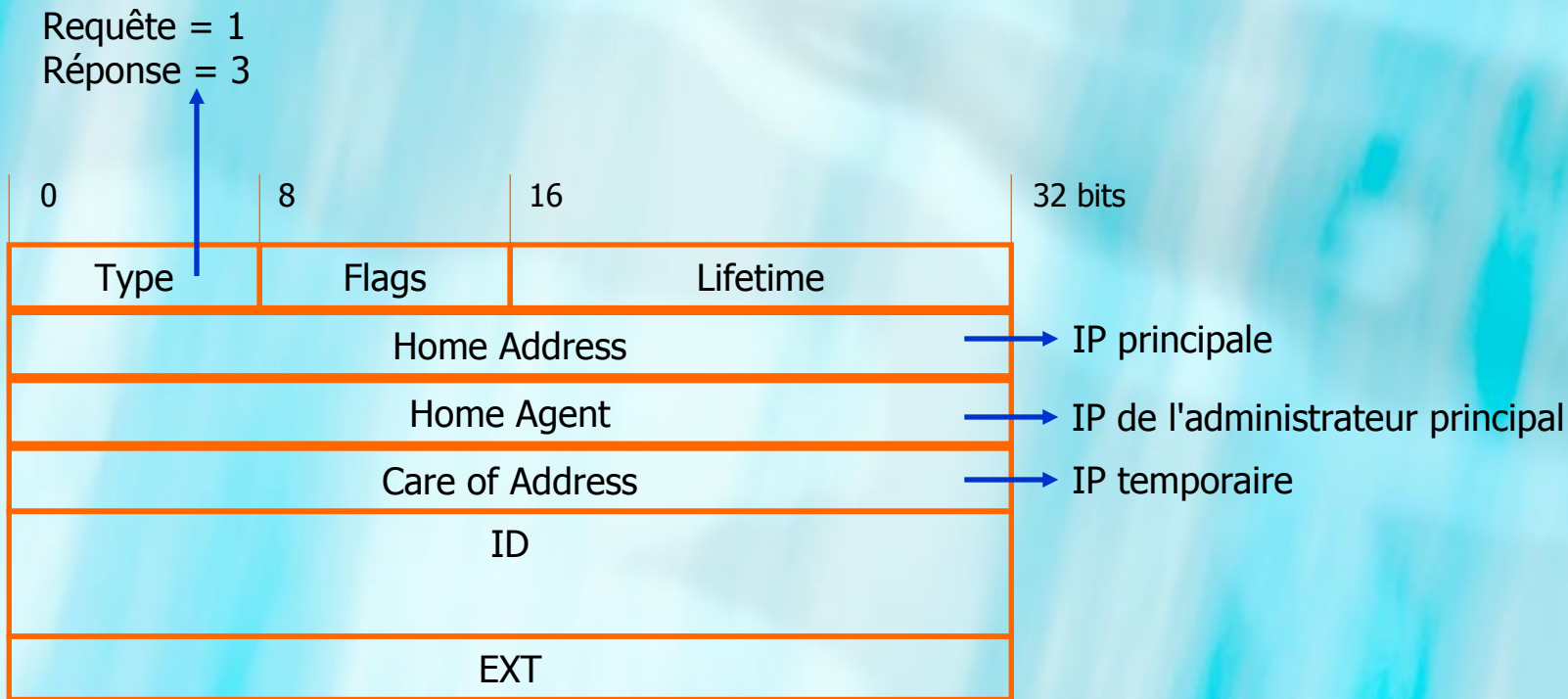
- H1...x des machines avec des IP privées non routables (10.x, ...)
- IP public est un réseau IP classique : 193.1.1.y
- Quand une machine Hx sort du réseau privée, le routeur interroge le NAT qui se charge de traduire l'adresse privée 10.x en une adresse public 193.1.1.y et de mémoriser l'association pour la réponse.

- Pour les machines voyageuses - Deux adresses IP
 - Principale (officielle)
 - Secondaire (temporaire)
- Concerne principalement les routeurs (s'ils disposent des fonctionnalités)

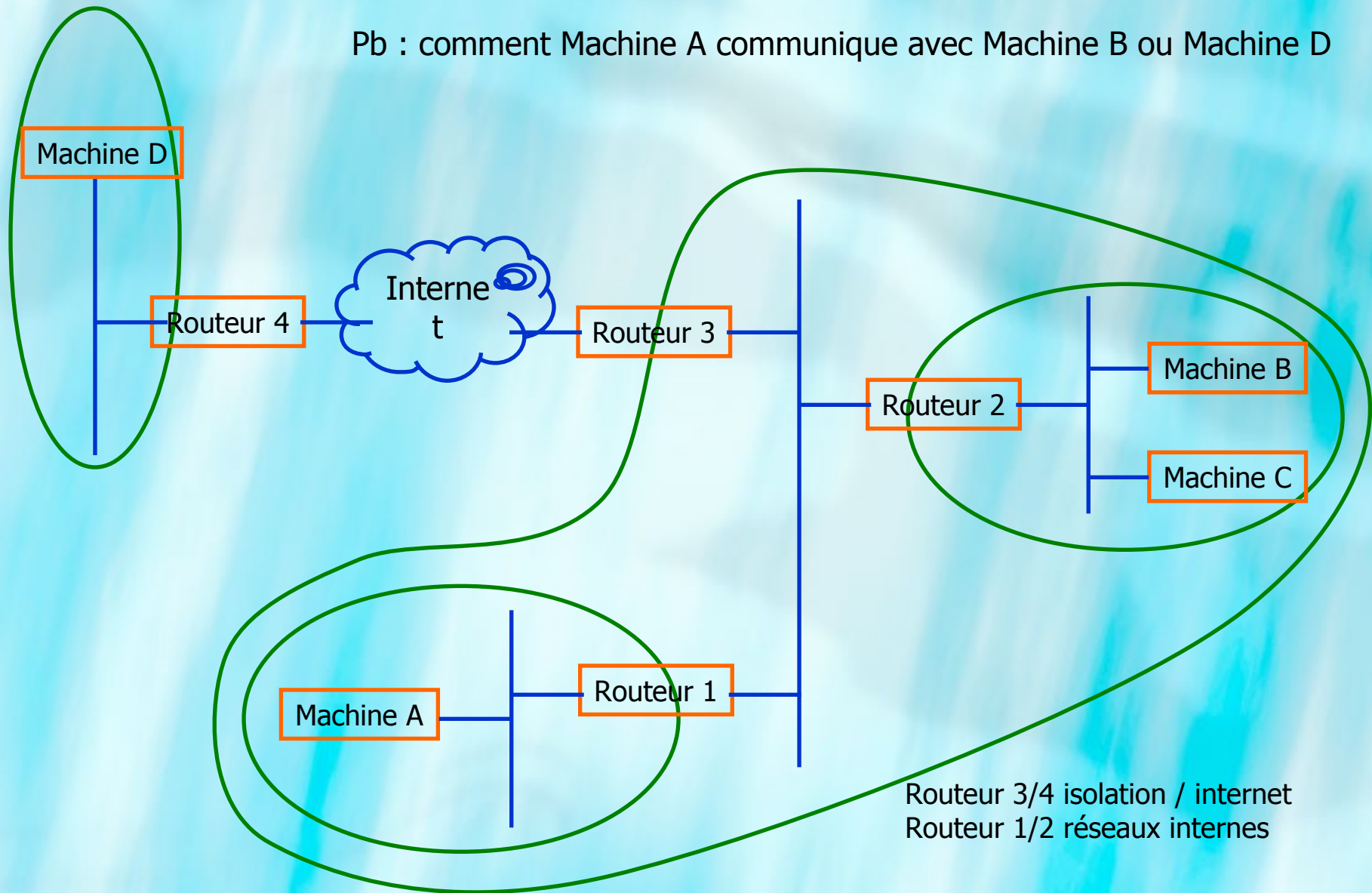


- Quand la machine arrive dans une nouvelle société
- Il informe le routeur temporaire de son arrivé
- Le routeur informe le routeur principal de l'existence d'une de ses machines mobiles
- Les deux routeurs mettent en place un tunnel entre eux
- Le tunnel consiste à traduire les adresses pour cette machine spécifique

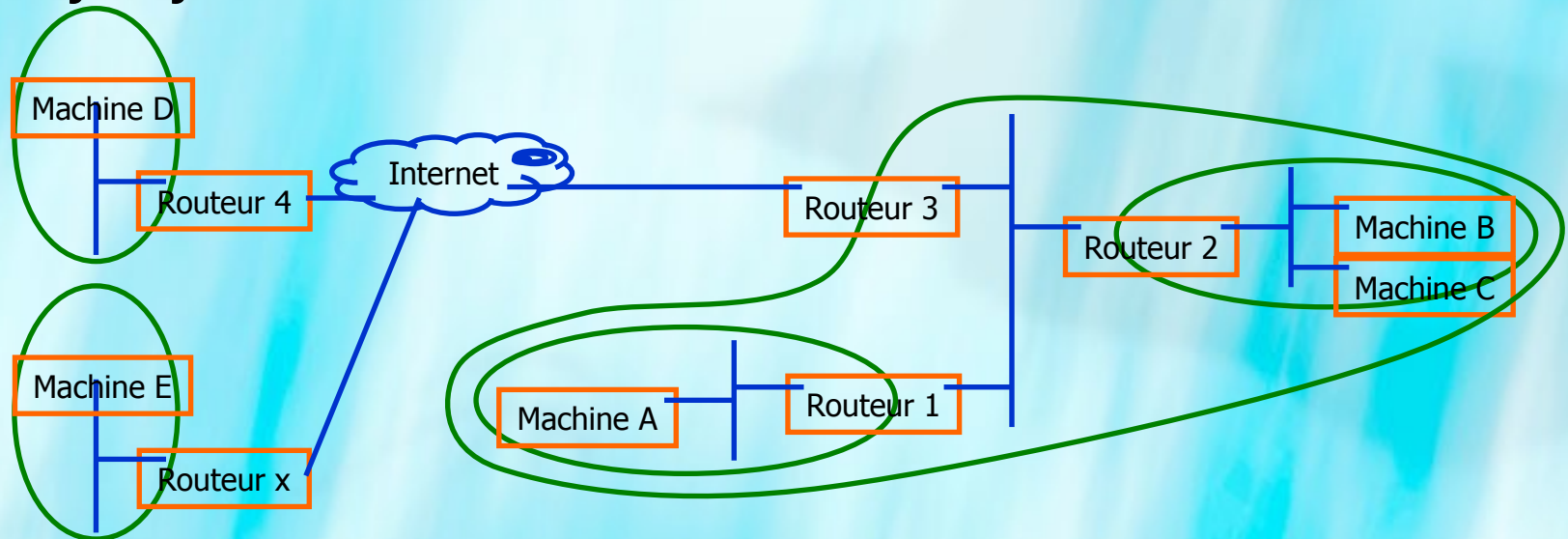
- Format d'une requête envoyée sur le port UDP 437



Pb : comment Machine A communique avec Machine B ou Machine D



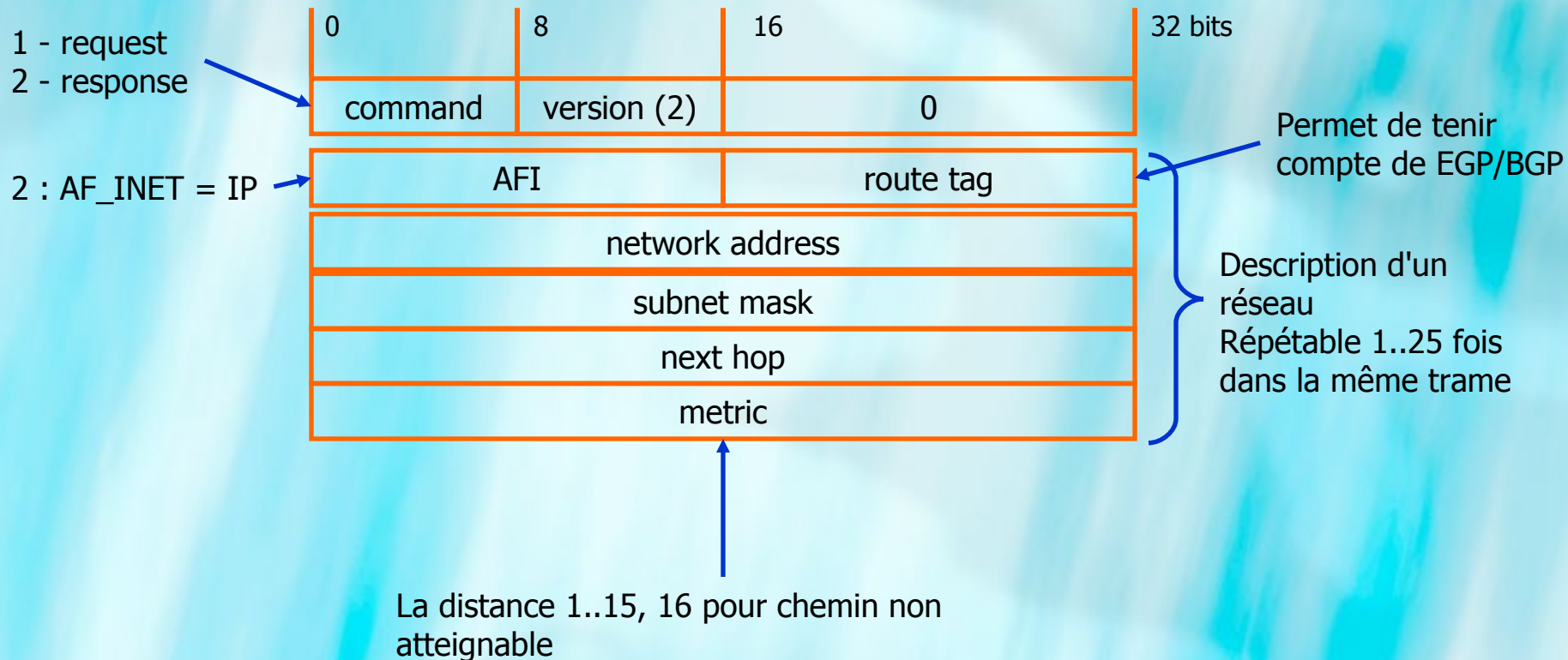
- Avec le jeu des adresses de passerelles par défaut
 - MachineA envoie ses paquets au Routeur 1
 - Routeur 1 envoie les paquets au Routeur 2 (comment il savait ?)
 - Routeur 2 envoie les paquets à la Machine B
 - MachineA envoie ses paquets au Routeur 1
 - Routeur 1 envoie les paquets au Routeur 3 (et pourquoi celui-ci ?)
 - Routeur 3 envoie les paquets au Routeur 4 (et pourquoi celui-ci ?)
 - Routeur 4 envoie les paquets à la Machine D
- Et si je rajoute un nouveau Routeur x ???

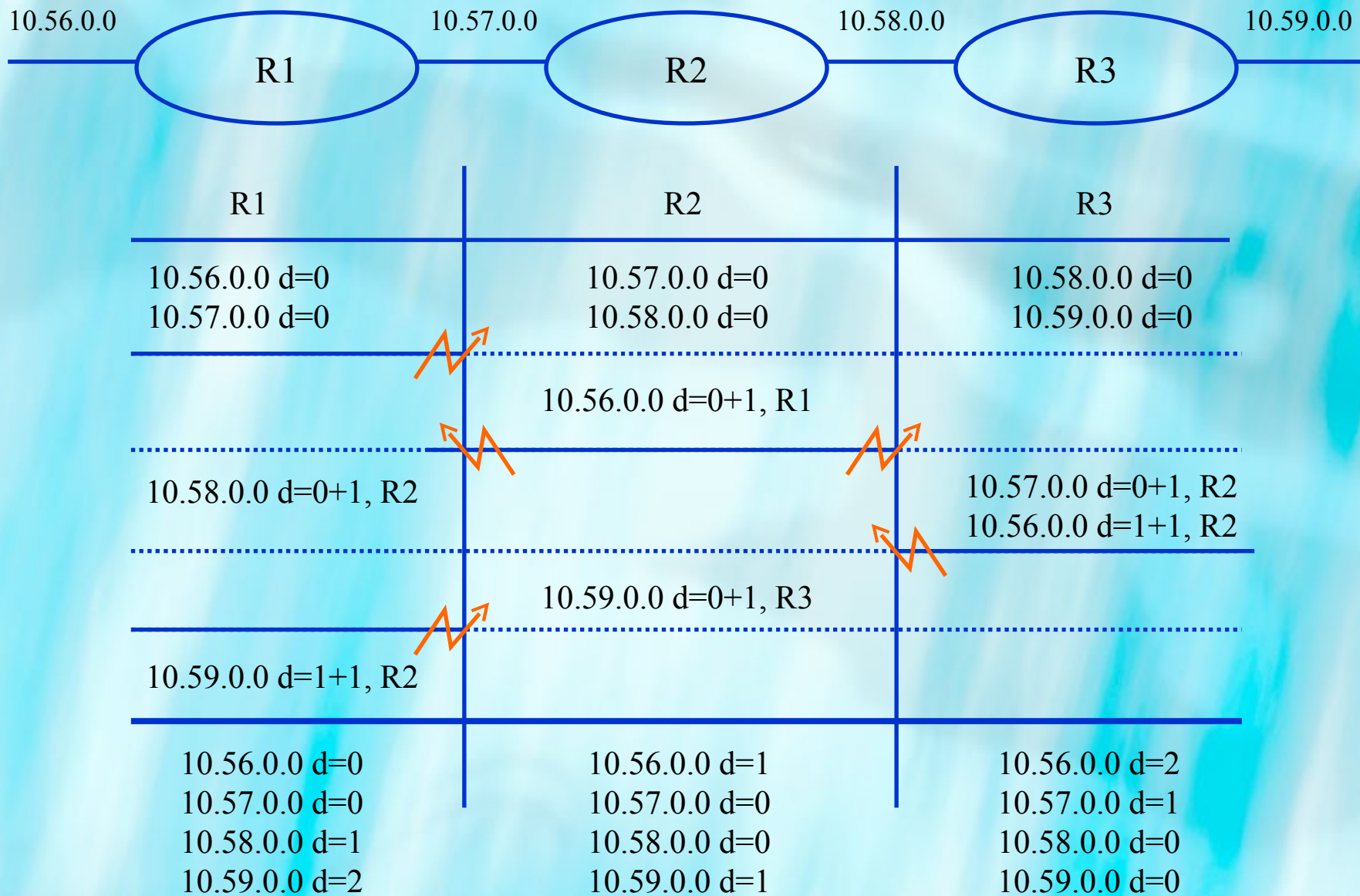


- La passerelle par défaut est une information suffisante pour un ordinateur, mais pas pour les routeurs
- Il faut que les routeurs parlent entre eux pour se donner les tables de routage
- Nécessité d'un protocole d'échange entre routeurs
- Il y a deux famille de protocoles : interior et exterior
- Interior protocol (à l'intérieur d'un site)
 - RIP - Routing Information Protocol - RFC 1058, 1387, RFC 1388
 - OSPF - Open Shortest Path First - RFC 1583
 - IGRP - Interior Gateway Routing Protocol (propriétaire)
- Exterior protocol (entre grands sites)
 - EGP - Exterior Gateway Protocol - RFC 904, 911
 - BGP - Border Gateway Protocol - RFC 1105, 1163, 1267, 1654, 1655

- Utilise l'algorithme de Bellman-Ford
 - Chaque routeur calcule la distance qui le sépare de ses voisins
 - La distance est le nombre de routeur à franchir -> 15 max
 - Chaque routeur diffuse la collection de réseaux et de distances sur les réseaux auxquels il a accès
 - Diffusion sur requête ou sur timer de 30s en principe
- RIP versions 1
 - Diffusion par broadcast
- RIP versions 2
 - Diffusion par Multicast 224.0.0.9
 - Intégration des netmasks dans le protocole
 - Intégration des routages extérieurs
 - Intégration de l'authentification

- Format d'une trame de donnée RIP v2 (existe une trame d'authentification)



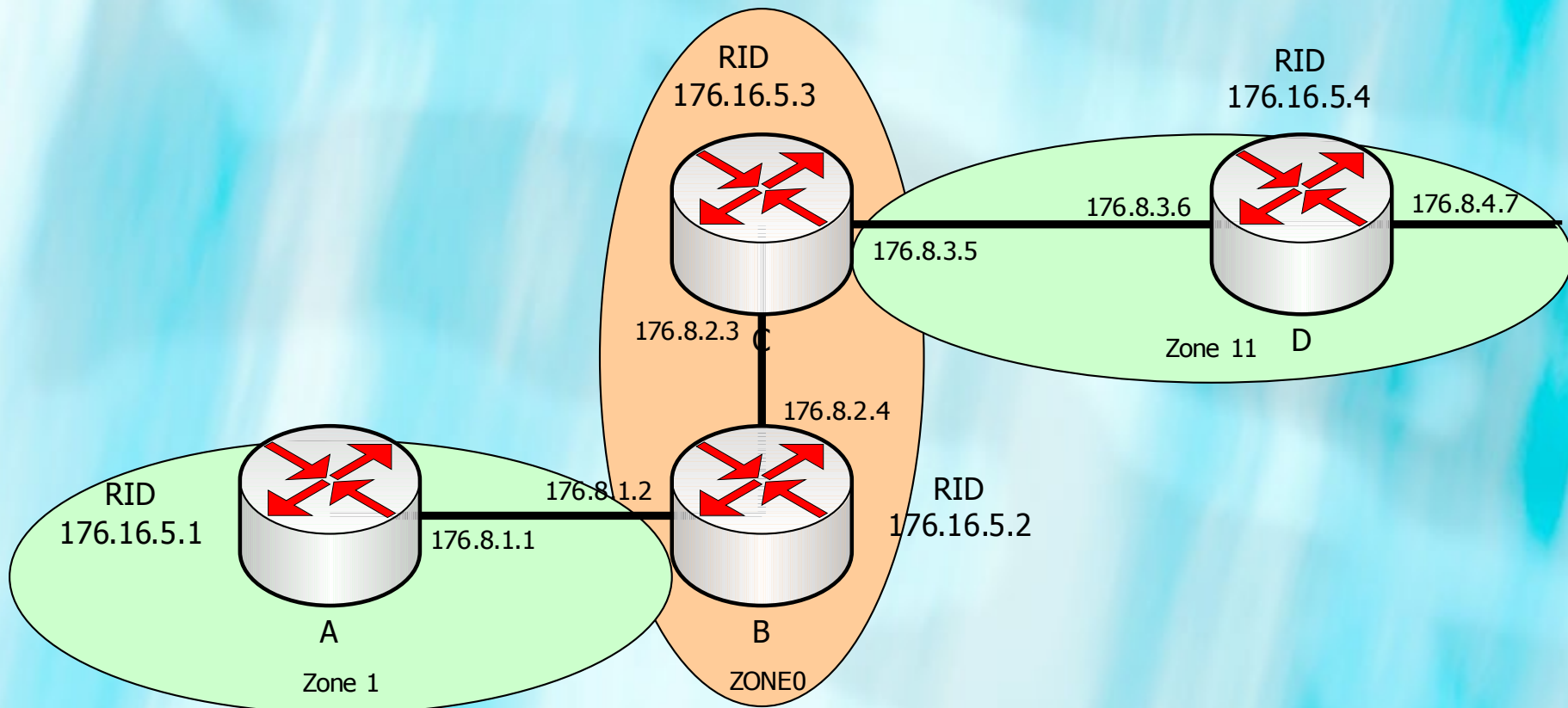


- RIP

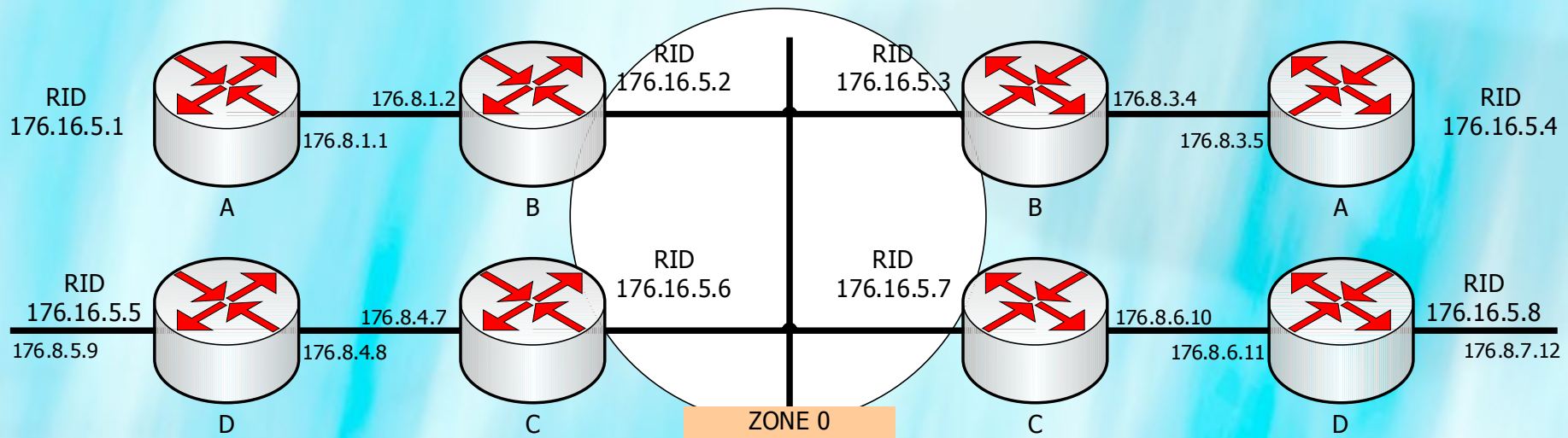
- Très simple à réaliser
- Ne gère pas la qualité des chemins
- Ne peut pas s'étendre sur une grande échelle (internet)
- N'est plus prévu pour s'étendre sur une grande échelle

- OSPF

- Introduire la notion de zones
- Toutes les zones sont interconnectées sur un backbone
- Utiliser la qualité des chemins et pas le nombre de sauts
- Diffusion automatique ou sur demande par multicast
- Election du routeur OSPF
 - La plus haute adresse IP
 - L'adresse de retour de boucle (loopback)



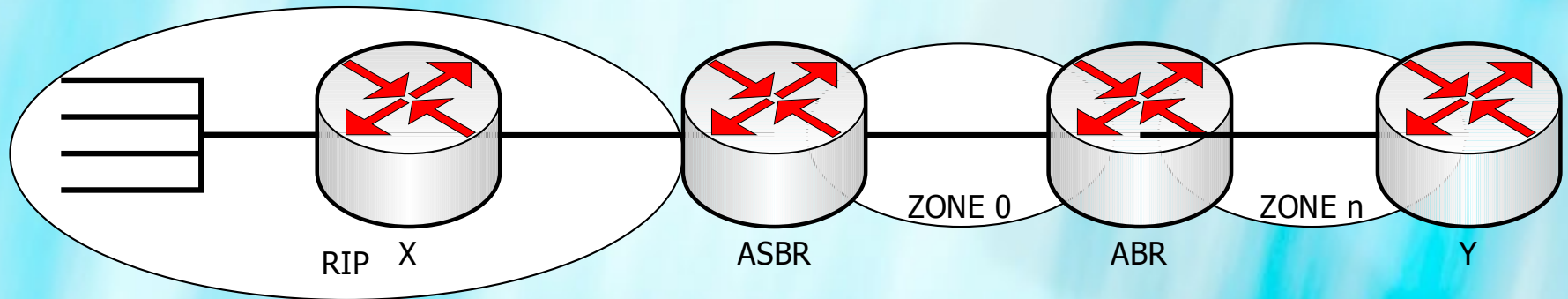
- Obligation de connecter toutes les zones sur la zone 0 (backbone)
- Limitation de l'expansion de OSPF



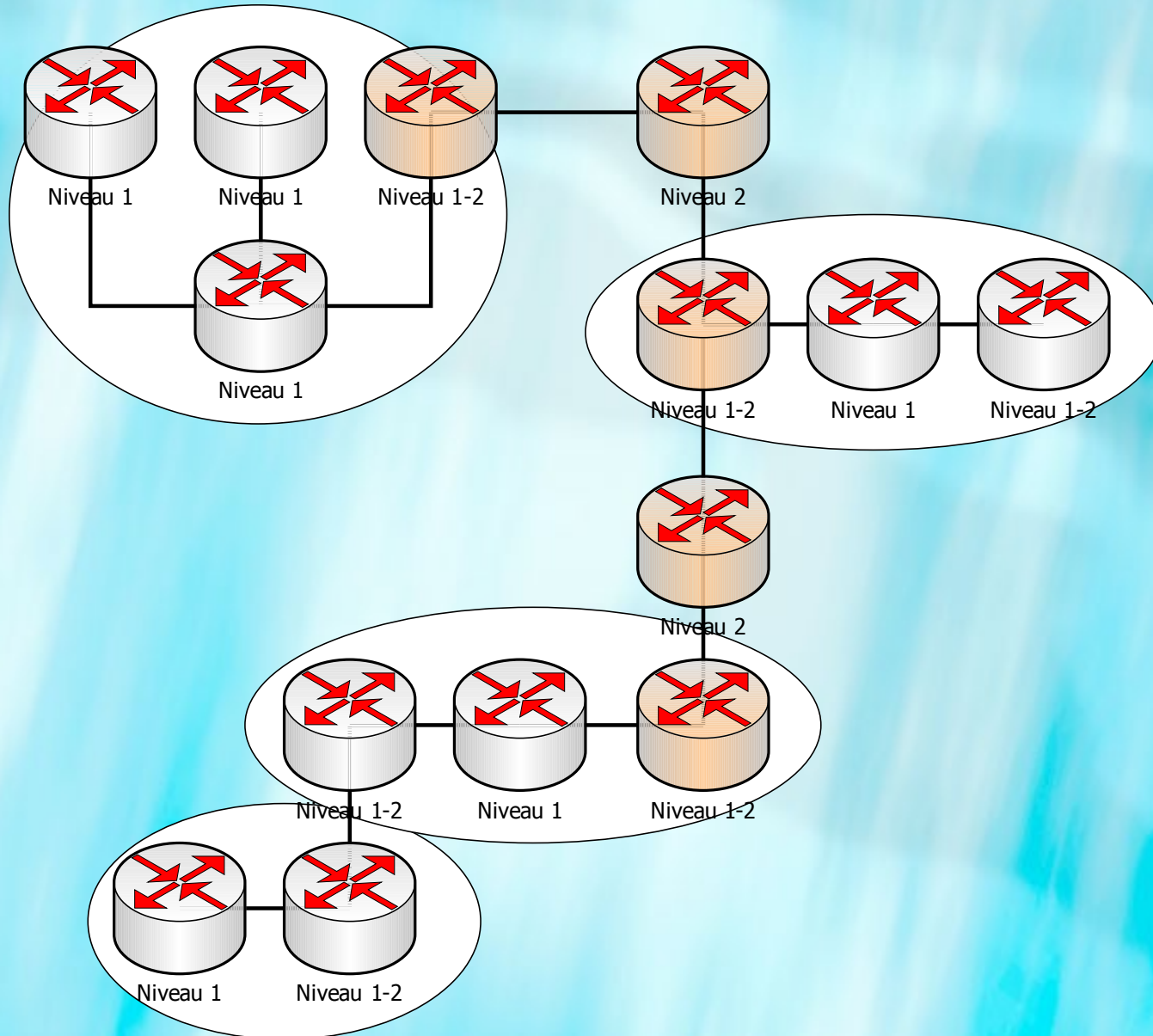
- Métrique de la route
 - Uniquement la bande passante
 - $\text{Coût} = 100\,000\,000 / \text{bande_passante}$

Interface	Bande passante	Coût
Ethernet	10 Mb	10
Fast Ethernet	100 Mb	1
Gigabit Ethernet	1000 Mb	1
T1	1.544 Mb	64

- Trois types de routeurs
 - Routeur de frontière (ABR)
 - Zone x vers la zone 0
 - Routeur de frontière de système autonome (ASBR)
 - Intégration de RIP dans OSPF
 - Routeur interne
 - Une zone isolée

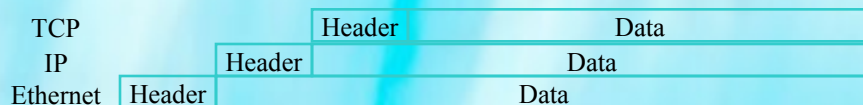
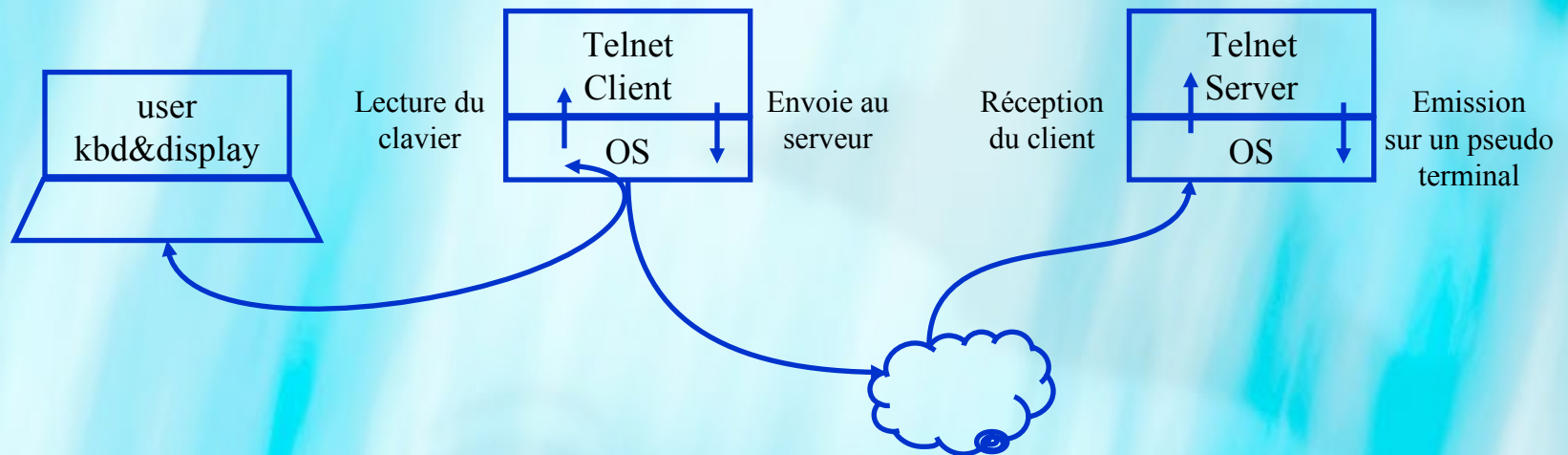


- Plus de zone 0 pour le backbone
- Interconnexions plus simples
- A nouveau trois types de routeurs
 - Niveau 1 \Leftrightarrow Intrazone \Leftrightarrow RI (OSPF)
 - Niveau 1-2 \Leftrightarrow Intrazone et Interzone \Leftrightarrow ABR (OSPF)
 - Niveau 2 \Leftrightarrow Interzone \Leftrightarrow
- Backbone est formée par la chaîne de routeurs de niveau 2
- Métriques
 - 10 par défaut ...
 - Il faut l'affecter manuellement avec des contraintes incroyables
 - [0..63] pour le niveau 1
 - [0..1024] pour le niveau 2
 - Il est possible de travailler avec des métriques plus larges (wide)
 - 24 bits pour le niveau 1
 - 32 bits pour le niveau 2



- Telnet
- FTP
- SMTP
- POP/IMAP
- HTTP
- URI
- MIME
- LDAP
- IRC
- Napster

- Telnet est une application qui permet de simuler la connexion d'un terminal via le réseau
- La transparence des OS est assurée par le Network Virtual Terminal (NVT)
- Il y a toujours un NVT à chaque extrémité



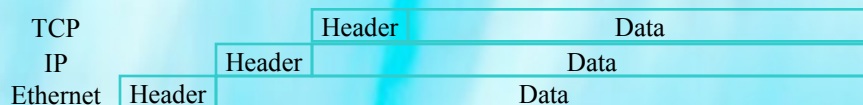
- Le NVT est un composant bidirectionnel
 - Sortie "terminal"
 - Entrée "clavier"
 - Supporte un mode d'urgence ou de re-synchronisation sur le réseau
 - Le NVT spécifie et standardise les caractères de contrôle
 - Négocie les fonctionnalités à la connexion

- Le NVT spécifie et standardise les caractères de contrôle
 - NULL (0) sans effet
 - CR LF est la seule forme du saut de ligne
 - CR (13) déplace le curseur en début de ligne uniquement
 - LN (10) déplace le curseur vers le bas uniquement
 - Pour ne pas avoir d'ambiguïté, il est recommandé de faire CR (13) NULL (0) pour avoir l'effet d'un retour chariot
 - BS (8) recule le curseur d'une cellule
 - BELL (7) signal audible
 - HT (9) tabulation horizontale
 - VT (11) tabulation verticale
 - FF (12) saut de page
 - IP (255+244) interruption du processus distant
 - EC (255+247) efface le dernier caractère
 - EL (255+248) efface la dernière ligne

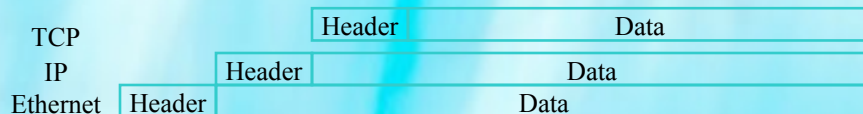
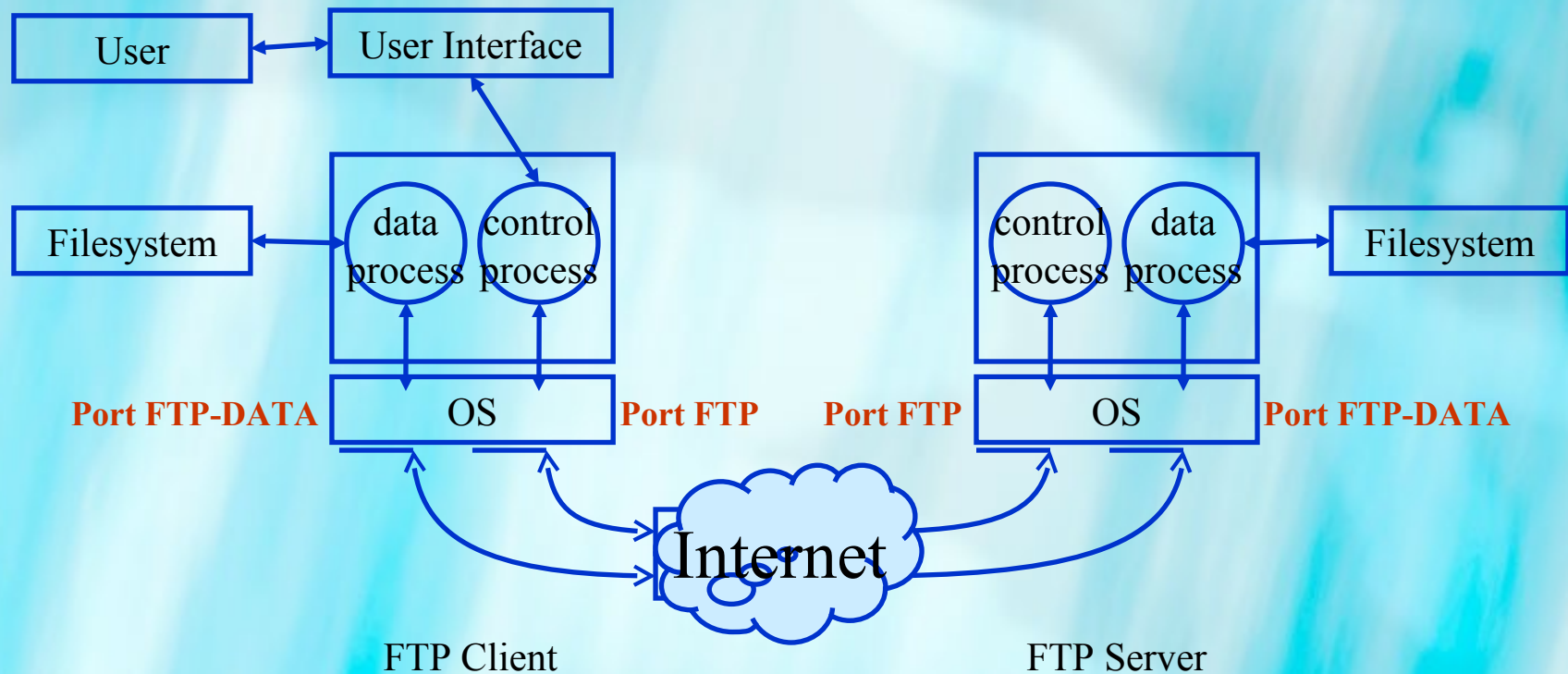
- Le NVT sait envoyer des ordres de contrôles urgent :
 - Envoyer un IP (interruption du processus distant) sans vider les tampons ...
- Processus client :
 - Fabrique un message Interrupt Process (IP) (255 + 244)
 - Envoie le message dans un datagram avec le bit URG (urgence)
 - Fabrique un message Data Mark (DM) (255 + 242)
 - Envoie le message normalement
- Processus serveur :
 - Réceptionne un message urgent
 - Vide le tampon jusqu'à l'occurrence d'un message DM

- Le NVT sait négocier les fonctionnalités :
 - En début de connexion émission de messages de négociation
- Codes pour la négociation
 - IAC (255) : Interpret next octet As Command
 - Suivie de l'un des quatre :
 - DON'T (254) : n'utilise pas impérativement une option
 - DO (253) : utilise impérativement une option
 - WON'T (252) : demande si l'option est disponible
 - WILL (251) : infirme la disponibilité d'une option
 - Suivie du numéro d'option (1 octet)
- Attention :
 - DO/DON'T est acquitté par WILL/WON'T
 - WILL/WON'T est acquitté par DO/DON'T

- Le RFC 959 est le standard track mais ... il y en a eu beaucoup et même plus (42 avant, et après ?)
- FTP est le premier protocole de partage de données
- FTP permet de transférer des gros fichiers entre machines sous contrôle de l'utilisateur
- FTP est indépendant du système de fichiers du serveur
- FTP est sûr car il utilise un protocole sûr



- Le modèle de traitement FTP est complexe (le plus complexe pour l'instant)



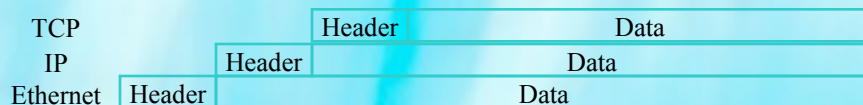
- Chaque port est piloté par un NVT (celui de Telnet)
- Le NVT de FTP est plus complet que celui de telnet, il supporte :
 - Des conversions de représentation (ASCII, EBCDIC, IMAGE, etc.)
 - Des conversions de format (ligne, page, enregistrement, etc.)
 - Des modes de transmission (block, ligne, comprimé, etc.)
- Le programme FTP peut être
 - Client actif connecté à un serveur
 - Client passif qui attend une connexion

- Liste de commandes
 - Impossible de la donner (33)
- Commandes de contrôle d'accès
 - gérer l'authentification du client (USER, PASS, ...)
- Commandes de transfert
 - configurer le mode de transfert (IMAGE, ASCII, ...)
- Commandes de service
 - réalise le transfert (RETR, STOR, DELE, MKD, LIST, ...)

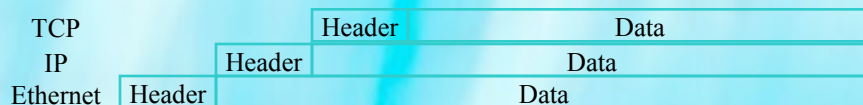
- Le format d'échange utilisateur -> serveur est la ligne de caractères
 - COMMANDE OPTIONS
 - Où commande est composé au plus de 4 caractères
- Le format d'échange serveur -> utilisateur est une ligne de caractères
 - CODE MESSAGE
 - Où code est un nombre à trois chiffres qui donne le code d'erreur de la commande. Ce nombre est vital pour l'interprète de commande FTP
 - Un espace sépare le code du message
 - Où message est un texte qui donne le message d'erreur de la commande. Ce texte est ignoré par l'interprète de commande FTP mais peut être utile pour l'utilisateur
- Ou il peut être un ensemble de ligne de caractères
 - CODE-MESSAGE1
 - MESSAGE2
 - ...
 - CODE MESSAGE_n
 - Un moins '-' sépare le code du message
 - Et la fin du message est indiquée par une répétition du code en début de ligne

- Le format des codes d'erreur est codifié
 - 1yz réponse positive préliminaire
 - 2yz réponse positive définitive
 - 3yz réponse positive intermédiaire (en attente de données complémentaires)
 - 4yz réponse négative préliminaire
 - 5yz réponse négative définitive
- La nature des erreurs est également codifiée
 - x0z problème de syntaxe
 - x1z information
 - x2z connexion
 - x3z authentification et compte
 - x4z non spécifié
 - x5z système de fichier

- FTP assure une session via une connexion type login sur la machine distante.
 - C:\>ftp chanterelle
 - Connected to chanterelle.univ-mulhouse.fr.
 - 220 chanterelle FTP server (SunOS 5.6) ready.
 - User (chanterelle.univ-mulhouse.fr:(none)): hassen
 - 331 Password required for hassen.
 - Password:
 - 230 User hassen logged in.
 - ftp>
- L'authentification se fait avec le login/passwd habituel de la machine
- Cela signifie que la couche session est intégrée à FTP... Sous forme de commandes spécifiques (c'est déjà ça)... Dommage qu'elle ne soit pas dans la bonne couche...

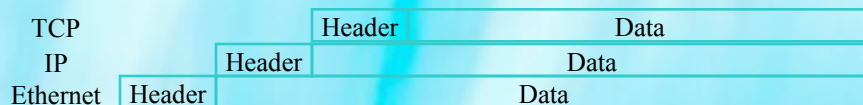


- FTP propose des commandes pour naviguer
 - ftp> dir
 - 200 PORT command successful.
 - 150 ASCII data connection for /bin/ls (10.58.17.2,1249) (0 bytes).
 - total 1036
 - drwxr-x--- 7 hassen prof 1024 Nov 13 16:38 .
 - drwxr-xr-x 10 root root 512 Nov 7 14:25 ..
 - drwxr-xr-x 2 hassen prof 512 Oct 1 09:25 Comptes
 - -rw-r--r-- 1 hassen prof 138427 Nov 13 16:38 rosi.zip
 - 226 ASCII Transfer complete.
 - 902 octets reçus dans 0,00 seconds (902000,00 Kbytes/sec)
 - ftp> cd Comptes
 - 250 CWD command successful.

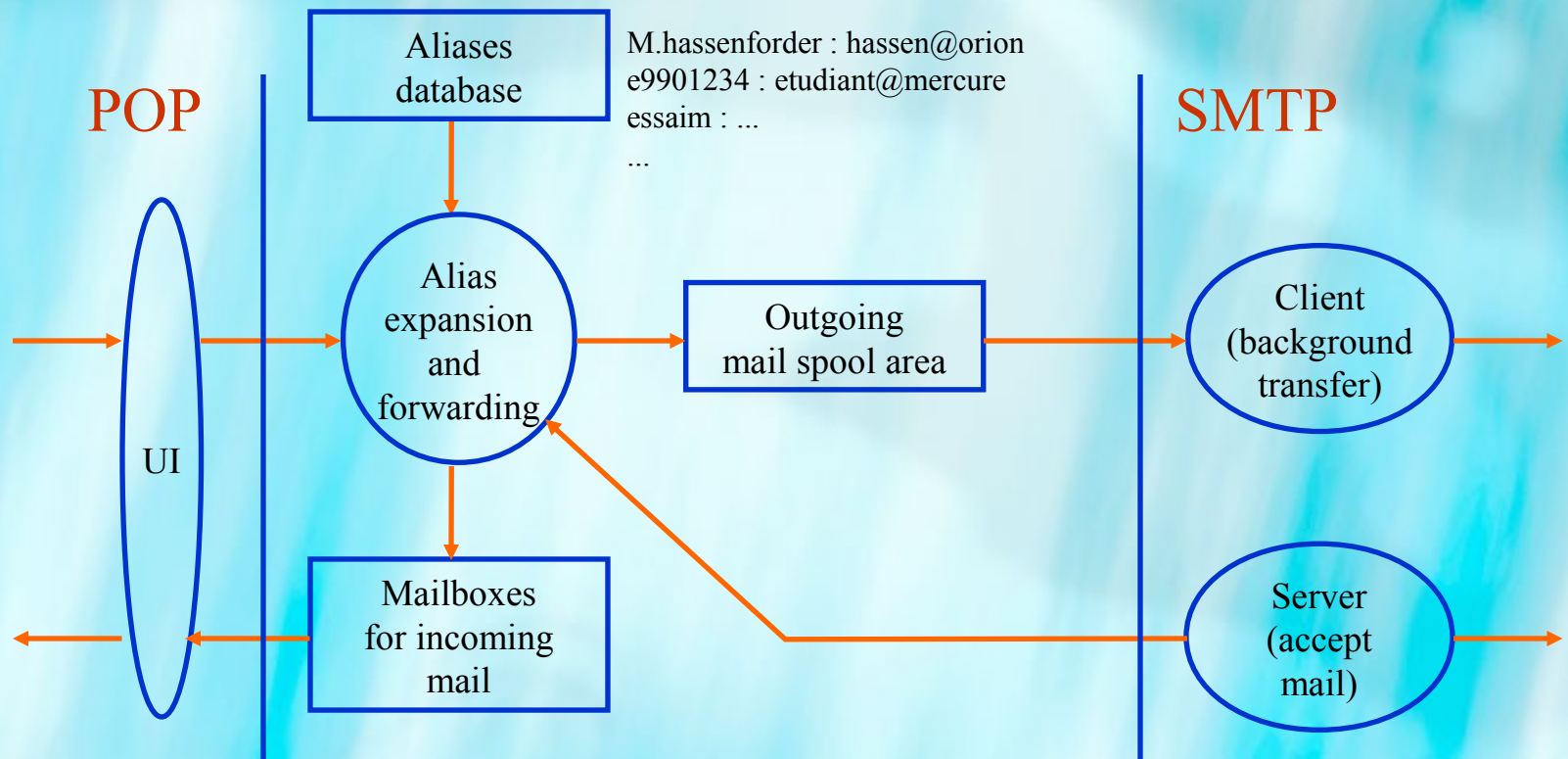


- FTP et des commandes de transfert

- ftp> get 3a.txt
- 200 PORT command successful.
- 150 ASCII data connection for 3a.txt (10.58.17.2,1251) (450 bytes).
- 226 ASCII Transfer complete.
- 503 octets reçus dans 0,01 seconds (50,30 Kbytes/sec)
- ftp> put 3a.txt 333.txt
- 200 PORT command successful.
- 150 ASCII data connection for 333.txt (10.58.17.2,1252).
- 226 Transfer complete.
- 503 octets envoyés dans 0,01 seconds (50,30 Kbytes/sec)
- ftp> image
- 200 Type set to I
- ftp>



- Deux protocoles qui permettent d'échanger du courrier
 - SMTP entre deux serveurs
 - POP/IMAP entre client et serveur



- La logique du protocole est complètement basée sur FTP
 - Voilà pourquoi FTP était important...
- Dialogue
 - Orienté ligne avec une commande sur 4 caractères et des options
 - La réponse est un code sur 3 chiffres et un texte sur une ligne
- Commandes (les plus importantes)
 - HELO obsolète et remplacé par EHLO
 - EHLO nom-machine-complet (coucou je suis la machine ...)
 - MAIL FROM: courriel_expéditeur (j'envoie du courrier de ...)
 - RCPT TO : courriel_destinataire (j'envoie du courrier vers ...)
 - DATA (voilà le texte jusqu'à une ligne qui commence par un point '.')
 - QUIT (fin de session)
 - TURN (échange les rôles : serveur / client)
 - Les deux formes de HELO et EHLO permet d'identifier la version de SMTP

- (S) 220 orion.uha.fr Simple Mail Transfert Service Ready
- (C) HELO papyrus.essaim.uha.fr
- (S) 250 orion.uha.fr
- (C) MAIL FROM:<m.hassenforder@papyrus.essaim.uha.fr>
- (S) 250 OK
- (C) RCPT TO:<qwerty@uha.fr>
- (S) 550 No such user here
- (C) RCPT TO:<webmaster@uha.fr>
- (S) 250 OK
- (C) DATA
- (S) 354 Start mail input; end with <CR><LF>.<CR><LF>
- (C) blabla le texte du mail
- (C) <CR><LF>.<CR><LF>
- (S) 250 OK
- (C) QUIT
- (S) 221 uha.fr Service closing transmission channel

- La logique du protocole est complètement basé sur FTP
 - Voilà pourquoi FTP était important...

- Il y a la logique de la réponse qui est un peu spéciale
 - +OK text pour une commande qui se passe bien
 - -ERR text pour une commande en erreur

- Les commandes ... facile
 - USER user_name (nom d'utilisateur)
 - PASS passwd (mot de passe)
 - LIST (liste des numéros de messages non effacés)
 - RETR number (extraction du message numéro number)
 - DELE number (destruction du message numéro number)
 - QUIT (fin de session)

- (S) **+OK** POP3 orion.univ-mulhouse.fr v2001.78rh server ready
- (C) **USER** hassen
- (S) **+OK** User name accepted, password please
- (C) **PASS** laporte
- (S) **+OK** Mailbox open, 2 messages
- (C) **STAT**
- (S) **+OK** 2 3781
- (C) **LIST**
- (S) **+OK** Mailbox scan listing follows
- (S) 1 1963
- (S) 2 1818
- (S) .
- (C) **RETR** 1
- (S) **+OK** 1963 octets
- (S) ... le texte du message
- (S) .
- (S)
- (C) **DELE** 1
- (S) **+OK** Message deleted
- (C) **LIST**
- (S) **+OK** Mailbox scan listing follows
- (S) 2 1818
- (S) .
- (C) **QUIT**
- (S) **+OK** Sayonara

Return-Path: <c.petitjean@uha.fr>
Received: from dnsinterne.uha.fr (dnsinterne.uha.fr [10.200.9.10])
by orion.univ-mulhouse.fr (8.11.6/8.11.6) with SMTP id g916rKp11791
for <hassen@orion.univ-mulhouse.fr>; Tue, 1 Oct 2002 08:53:20 +0200
Received: from orion.univ-mulhouse.fr (...)
by dnsinterne.uha.fr (8.11.6/8.11.0) with SMTP id g916rKd28929
for <m.hassenforder@uha.fr>; Tue, 1 Oct 2002 08:53:20 +0200
Received: from uha.fr (yotta.univ-mulhouse.fr [10.58.45.8])
by orion.univ-mulhouse.fr (8.11.6/8.11.6) with ESMTP id g916rJp11786
for <m.hassenforder@uha.fr>; Tue, 1 Oct 2002 08:53:19 +0200
Message-ID: <3D994651.3040701@uha.fr>
Date: Tue, 01 Oct 2002 08:53:05 +0200
From: Cyrille PETITJEAN <c.petitjean@uha.fr>
Reply-To: c.petitjean@uha.fr
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; fr-FR; rv:1.0.0) Gecko/20020530
X-Accept-Language: fr-fr, en-us, en
MIME-Version: 1.0
To: Michel Hassenforder <m.hassenforder@uha.fr>
Subject: Re: test
References: <3D9945B6.1070703@uha.fr>
Content-Type: text/plain; charset=ISO-8859-1; format=flowed
Content-Transfer-Encoding: 8bit
X-MIME-Autoconverted: from quoted-printable to 8bit by orion.univ-mulhouse.fr id g916rKp11791
Status: O

cout << "hello world" << endl;

Le saut de ligne pour séparer les options du texte

Michel Hassenforder a écrit:

> salut, peux tu me faire un reply SVP

- La logique du protocole est complètement basé sur FTP
 - Voilà pourquoi FTP était important...
- Dialogue client -> serveur : Orienté ligne en quatre parties (http1.0)
 - Requête (sur une ligne)
 - Options (champs optionnel les capacités du client - négociation...)
 - Marqueur (une ligne vide)
 - Corps (Optionnel)
- Dialogue serveur -> client : Orienté ligne en quatre parties (http1.0)
 - Ligne code réponse (HTTP/1.0 ### Message) où ### code numérique
 - Entête (des informations complémentaires - négociation...)
 - Marqueur (une ligne vide)
 - Corps (Optionnel)

- Il y a trois requêtes classiques et deux pas toujours implémentées :
- GET URI PROTOCOL
 - Demande le contenu de l'URI en utilisant le protocole (HTTP/1.0 ou HTTP/1.1)
 - En retour, le corps contiendra le fichier complet
- HEAD URI PROTOCOL
 - Demande l'entête de l'URI en utilisant le protocole (HTTP/1.0 ou HTTP/1.1)
 - En retour, le corps contiendra la balise HEAD du fichier HTML (les balises META)
- POST URI PROTOCOL
 - Demande le contenu de l'URI en utilisant le protocole (HTTP/1.0 ou HTTP/1.1)
 - Le corps du message contient des données complémentaires (formulaire)
- PUT URI PROTOCOL
 - Dépose le contenu de l'URI en utilisant le protocole (HTTP/1.0 ou HTTP/1.1)
 - Le corps du message contient le contenu du fichier
- DELETE URI PROTOCOL
 - Suppression de l'URI

- Options classiques

- Content-Type: text/html
- Expires: now
- Accept: */*
- Accept-Language: fr
- Accept-Encoding: gzip, deflate
- User-Agent: Mozilla/4.0 (compatible; MSIE 5.0; Windows 98; DigExt)
- Referer URL du lien à partir duquel la requête a été effectuée
- Content-Length Longueur du corps de la requête

```
GET /index.html HTTP/1.1
```

Ce fichier n'existe pas

```
HTTP/1.1 400 Bad Request
Date: Wed, 28 Nov 2001 13:47:01 GMT
Server: Apache/1.3.0 (Unix)
Connection: close
Content-Type: text/html

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<HTML><HEAD>
<TITLE>400 Bad Request</TITLE>
</HEAD><BODY>
<H1>Bad Request</H1>
Your browser sent a request that
this server could not understand.<P>
</BODY></HTML>
```

```
GET /index.htm HTTP/1.1
Accept: image/gif, image/jpeg, application/msword, ..... */*
Accept-Language: fr
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 5.5; Windows NT 4.0)
Host: localhost
Connection: Keep-Alive
```

Le fichier existe

```
HTTP/1.0 200 OK
Date: Wed, 28 Nov 2001 13:47:01 GMT
Server: Apache/1.3.0 (Unix)
Connection: close
Content-Type: text/html

<HTML>
  <HEAD> <TITLE>index.htm</TITLE> </HEAD>
  <BODY>
    <H1> Voici le contenu du fichier index.htm</H1>
    <H2> petit titre 1 </H2>
    et voila du texte...
  </BODY>
</HTML>
```

- Depuis l'arrivé du Web, nous avons eu des
- URI Uniform Resource Identifiers (RFC 3986)
- URL Uniform Resource Locator (RFC 3986)
- URN Uniform Resource Name (RFC 3986)
- IRI Internationalized Resource Identifiers (RFC 3987)
- Pourquoi faire, y a t-il des différences... (voir un document du W3C)
- Comment c'est codé ?

- Initialement :

- URI c'est la généralisation des URL et URN (il aurait pu en voir d'autres).
- URL était une partie des URI qui donnait une localisation fixe sur le réseau.
- URN était une partie des URI qui donnait une localisation indépendante de la localisation sur le réseau.

- Aujourd'hui

- URI et URL sont interchangeable la notion de schéma a permis de fusionner les représentations.
- URN est un schéma particulier

- Exemples

- <ftp://ftp.is.co.za/rfc/rfc1808.txt>
- <http://www.ietf.org/rfc/rfc2396.txt>
- <mailto:John.Doe@example.com>
- [ldap://\[2001:db8::7\]/c=GB?objectClass=one](ldap://[2001:db8::7]/c=GB?objectClass=one)
- <news:comp.infosystems.www.servers.unix>
- <tel:+1-816-555-1212>
- <telnet://192.0.2.16:80/>
- <urn:oasis:names:specification:docbook:dtd:xml:4.1.2>
- [about:](#)

pct-encoded	::=	%[0-9a-fA-F] [0-9a-fA-F]	[ER]
delims	::=	[\$&'()*+,,;=]	[ER]
unreserved	::=	[a-zA-Z0-9._~ -]	[ER]
text	::=	unreserved pct-encoded delims	[EBNF]

URI	::=	scheme ":" hier-part ["?" query] ["#" fragment]	[EBNF]
-----	-----	---	--------

scheme	::=	[a-Za-z][a-Za-z0-9+.-]*	[ER]
--------	-----	-------------------------	------

hier-part	::=	"/" authority ("/" char*)* "/" [char+ ("/" char*)*] char+ ("/" char*)* ε	[EBNF]
-----------	-----	--	--------

authority	::=	[userinfo "@"] host [":" port]	[EBNF]
-----------	-----	------------------------------------	--------

char	::=	text ":" "@"	[EBNF]
------	-----	------------------	--------

userinfo	::=	(text ":")*	[EBNF]
----------	-----	-----------------	--------

host	::=	IPv4address reg-name "[" (IPv6address / IPvFuture) "]"	[EBNF]
------	-----	--	--------

IPv4address	::=	[0-9]+ "." [0-9]+ "." [0-9]+ "." [0-9]+	[ER]
-------------	-----	---	------

reg-name	::=	(text)*	[EBNF]
----------	-----	-----------	--------

port	::=	[0-9]+	[ER]
------	-----	--------	------

query	::=	(text ":" "@" "/" "?") *	[EBNF]
-------	-----	------------------------------------	--------

fragment	::=	(text ":" "@" "/" "?") *	[EBNF]
----------	-----	------------------------------------	--------

	Schéma	Autorité	Chemin	requête	fragment
ftp://ftp.is.co.za/rfc/rfc1808.txt	ftp	ftp.is.co.za	rfc/rfc1808.txt		
http://www.ietf.org/rfc/rfc2396.txt	http	www.ietf.org	rfc/re2396.txt		
http://hassen:xxx@www.essaim.uha.fr	http	hassen:xxx@www.essaim.uha.fr			
http://www.essaim.uha.fr:8080/cours/XML.html	http	www.essaim.uha.fr:8080	cours/XML.html		
http://www.essaim.uha.fr/index.html#xml	http	www.essaim.uha.fr	index.html		xml
mailto:John.Doe@example.com	mailto	John.Doe@example.com			
ldap://[2001:db8::7]/c=GB?objectClass?one	ldap	2001:db8::7	/	c=GB?objectClass?one	
news:comp.infosystems.www.servers.unix	news	comp.infosystems.www.servers.unix			
tel:+1-816-555-1212	tel	+1-816-555-1212			
telnet://192.0.2.16:80/	telnet	192.0.2.16:80			
urn:oasis:names:specification:docbook:dtd:xml:4.1.2	urn	oasis:names:specification:docbook:dtd:xml:4.1.2			
about:	about				

	UserInfo	Host	Port
ftp://ftp.is.co.za/rfc/rfc1808.txt		ftp.is.co.za	
http://www.ietf.org/rfc/rfc2396.txt		www.ietf.org	
http://hassen:xxx@www.essaim.uha.fr	hassen:xxx user:passwd probablement	www.essaim.uha.fr	
http://www.essaim.uha.fr:8080/cours/XML.html		www.essaim.uha.fr	8080
http://www.essaim.uha.fr/index.html#xml		www.essaim.uha.fr	
mailto:John.Doe@example.com	John.Doe	example.com	
ldap://[2001:db8::7]/c=GB?objectClass?one		2001:db8::7	
news:comp.infosystems.www.servers.unix		comp.infosystems.www.servers.unix	
tel:+1-816-555-1212		+1-816-555-1212	
telnet://192.0.2.16:80/		192.0.2.16:80	
urn:oasis:names:specification:docbook:dtd:xml:4.1.2		oasis:names:specification:docbook:dtd:xml:4.1.2	
about:			

- Le transport de données (complexes) est devenu de plus en plus complexe
- Il faut prévenir le plus vite possible l'application de la nature de la donnée
- Multipurpose Internet Mail Extensions MIME
- MIME permet d'encoder le type, l'encodage du document.
- La version MIME : MIME-Version: [0-9]+. [0-9]+
- Le type du document : Content-Type: Type/Subtype
- L'encodage : Content-Transfer-Encoding: Mecanisme
- L'identité selon MIME : Content-ID:ID (unique sur le monde)
- Une description informelle : Content-Description: .*
- Autres ...: Content-.*
-

- Le type du document est déclarée par : Content-Type: Type/Subtype
 - type/subtype C'est le format normalisé
 - type/subtype+xml C'est un format additionnel qui permet de préciser le codage xml du document (en fait le + est autorisé dans le nom de type, c'est juste une interprétation sémantique du subtype).
- Type est composé des valeurs
 - text | image | audio | video | application | message | multipart | <ietf-token> | <x-token>
- Text/plain ou text/richtext ou text/enriched
 - Du texte standard qui peut être lu et affiché sans interprète.
- Image/jpeg ou Image/gif ou Image/png ...
 - Pour des images qu'il est impossible d'afficher sans décodeur
- Audio/basic
 - Pour des sons qu'il faut obligatoirement jouer sur une carte son.
- Video/mpeg ...
 - Pour des videos qu'il est impossible d'afficher sans matériel adapté (codec hard ou soft)
- Application/octet-stream ou Application/???
 - Pour des documents qui sont attachés à des logiciels (office, etc.)
- Multipart/mixed ou Multipart/alternative ou Multipart/Parallele ou Multipart/Digest
 - Pour un document composite (ensemble, alternative, composition parallèle, résumé).
- Message/partial ou Message/RFC822
 - Message...
- Ietf-token
 - Pour des nouveaux types qui peuvent être enregistrés à l'IETF
- x-token
 - Pour des nouveaux types propriétaires ils commencent par x- ou X-

- L'encodage : Content-Transfer-Encoding: Mechanisme
- Mechanisme est composé des valeurs
 - 7bit | 8bit | binary | quoted-printable | base64 | <ietf-token> | <x-token>
- 7bit, 8bit et binary
 - Codage standard des octets
- quoted-printable
 - C'est un codage qui permet de représenter les caractères non 7bits en caractères 7bits.
 - En principe les caractères sont codés par un "=" suivi du code hexa en majuscule
 - Un code ASCII simple (33 à 126 sauf 61) peut ne être encodé
 - Les lignes font au plus 76 caractères, s'il faut découper des lignes longues il faut mettre un '=' en fin de ligne
- Base64
 - C'est un codage qui permet de représenter les caractères non 7bits en caractères 7bits.
 - Les lignes font au plus 76 caractères, s'il faut découper des lignes longues il faut mettre un '=' en fin de ligne
 - Ce codage est basé sur une table de translation qui permet de représenter 6 bits binaire sous un caractère 7bit équivalent.

▪ Binaire	Base64
▪ 0-25	A-Z
▪ 26-51	a-z
▪ 52-61	0-9
▪ 62	+
▪ 63	/
▪ Padding	=

- Lightweight Directory Access Protocol est :
 - Un modèle d'information (le type des données)
 - Un modèle de nommage (l'organisation des données)
 - Un modèle fonctionnel (accès à l'information)
 - Un modèle de sécurité (protection des données)
 - Un modèle de réplication (duplication des données entre serveurs)
 - Un protocole et un format d'échange
 - Une API pour les applications

- **Modèle de données**
 - Arbre de nœuds appelés entrée (Directory Service Entry)
 - Racine (Directory Information Tree)
 - Chaque entrée est un objet du monde réel
 - Chaque entrée contient des attributs qui stockent des valeurs
 - Un attribut est typé : bin, ces (case exact string), cis (case ignore string), ...
 - Les attributs ont des super-types et des sous-types
- **Les classes de d'objets**
 - Une classe est le moule qui décrit un futur objet
 - Nom (distinguished name : dn)
 - OID (voir SNMP c'est la même logique (1.3.6.1.4.1 entreprises)
 - Attributs obligatoires
 - Attributs optionnels
 - Types (dans ce cas l'héritage simple et multiple est possible)
 - Les fonctions de comparaisons et de classifications sur chaque attribut
- **La description des classes constituant un annuaire s'appelle un schéma**

- Exemple d'entrée LDAP en LDIF :

- dn : cn=hassenforder, ou=essaim, dc=uha, dc=fr
- objectClass: top
- objectClass: person
- objectClass: organizationalPerson
- objectClass: inetOrgPerson
- cn: hassenforder
- gn: michel
- mail: m.hassenforder@uha.fr
- uid: hassen
- telephoneNumber : (0)3 89 33 69 44

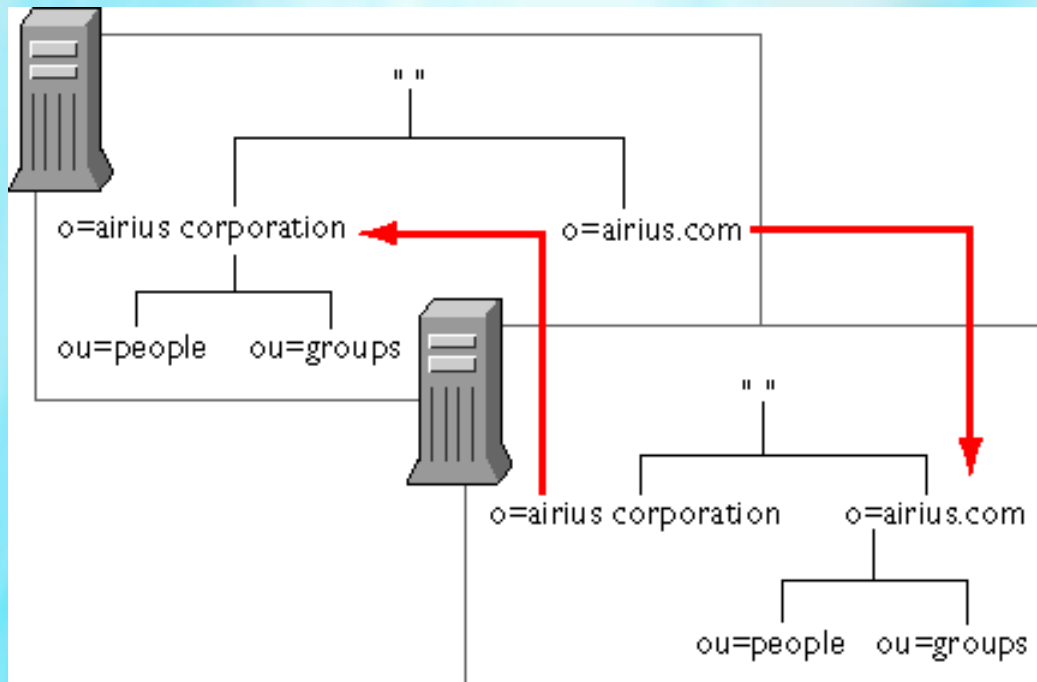
Entrée décrite sous la forme
du nom, dans un département,
dans un domaine internet

Les 4 pères qui vont définir le
type et les attributs
obligatoires et facultatifs

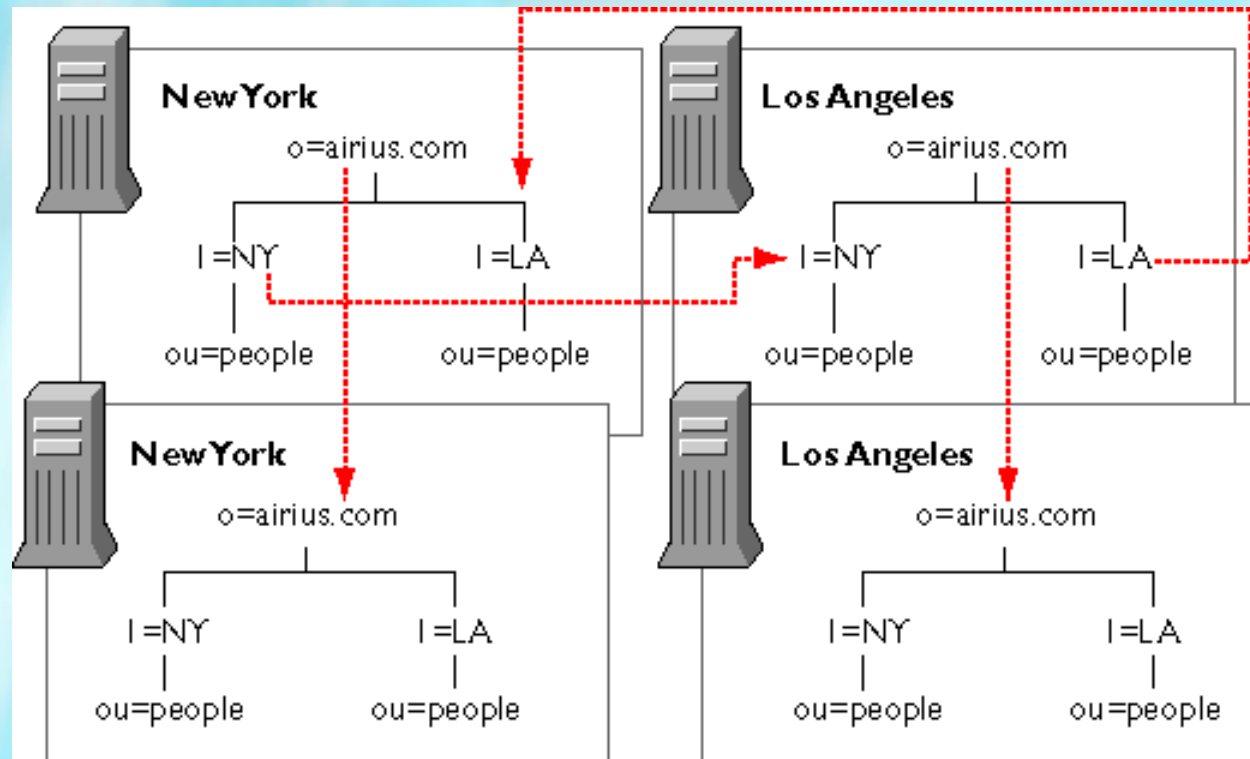
Des attributs divers

- Opérations de base (via le web...)
 - Search, Compare
 - Add, Modify, Delete, Rename
 - Bind, UnBind, Abandon, Extended
- Quelques paramètres de recherche et de comparaison
 - Base object (l'endroit de début de recherche)
 - Scope (base=local, one=1 niveau de profondeur, sub= le sous arbre)
 - Filter (algorithme de correspondance : =, ~=, >, &, |, !, etc)
- La recherche peut se faire via une URL (simple...)
 - `ldap://... /base_dn?attributes?scope?filter`
 - Base_dn : nom de début de la recherche
 - Attributes : les attributs qu'il faut traiter
 - Scope : base, one, sub.
 - Filter : le filtre de recherche (tous par défaut)

- Il faut concevoir le système de nommage
 - Structure de l'arbre
- Il faut concevoir le schéma
 - Les données, leurs types, etc
- Il faut concevoir la topologie (LDAP est distribué)



- Il faut concevoir la topologie (LDAP est redondant)



- IRC est défini par le RFC 1459 et mise à jour par les RFC 2810 à 2813
- La base est une communauté de serveurs
- Un client est connecté à un serveur
 - Avec un pseudonyme (9 caractères)
- Les serveurs se communiquent les clients
 - Pseudonyme
 - Nom de la machine client
 - Nom de l'utilisateur client
 - Nom du serveur de connection
- Les serveurs proposent des canaux de discussion
 - Un canal est un groupe de personnes
 - Un canal est créé avec le premier client et détruit avec le dernier

- Un canal est identifié par un nom (50 caractères) qui débute par :
 - # : canal global, le créateur est l'opérateur du canal
 - & : canal purement local au serveur
 - ! : la création est subordonnée à une demande au serveur
 - + : canal global sans mode de fonctionnement
- Un canal a des propriétés (modes):
 - a : anonyme (le pseudo de l'émetteur d'un message est masqué)
 - m : modéré (l'opérateur du canal modère les messages)
 - p : privée (le canal n'est pas listé par les commandes de recherche)
 - s : secret (le canal n'est pas listé dans aucune commande)
 - etc...

- Message IRC :
 - Une ligne unique de 510 caractères en trois parties
 - [":" prefix ' '] command [params]
 - prefix : nom_serveur ou pseudo...
 - params : liste de paramètres
 - command : la commande à réaliser
- Réponse IRC
 - [prefix] 3 chiffres et un texte
- Commandes
 - Il y en a beaucoup mais elles ressemblent à toutes les autres
 - NICK, PASS, USER, (identification d'un utilisateur)
 - JOIN, LIST, NAMES, KICK (joindre, liste canaux, liste connectés, éjection)
 - PRIVMSG (envoi d'un message !)

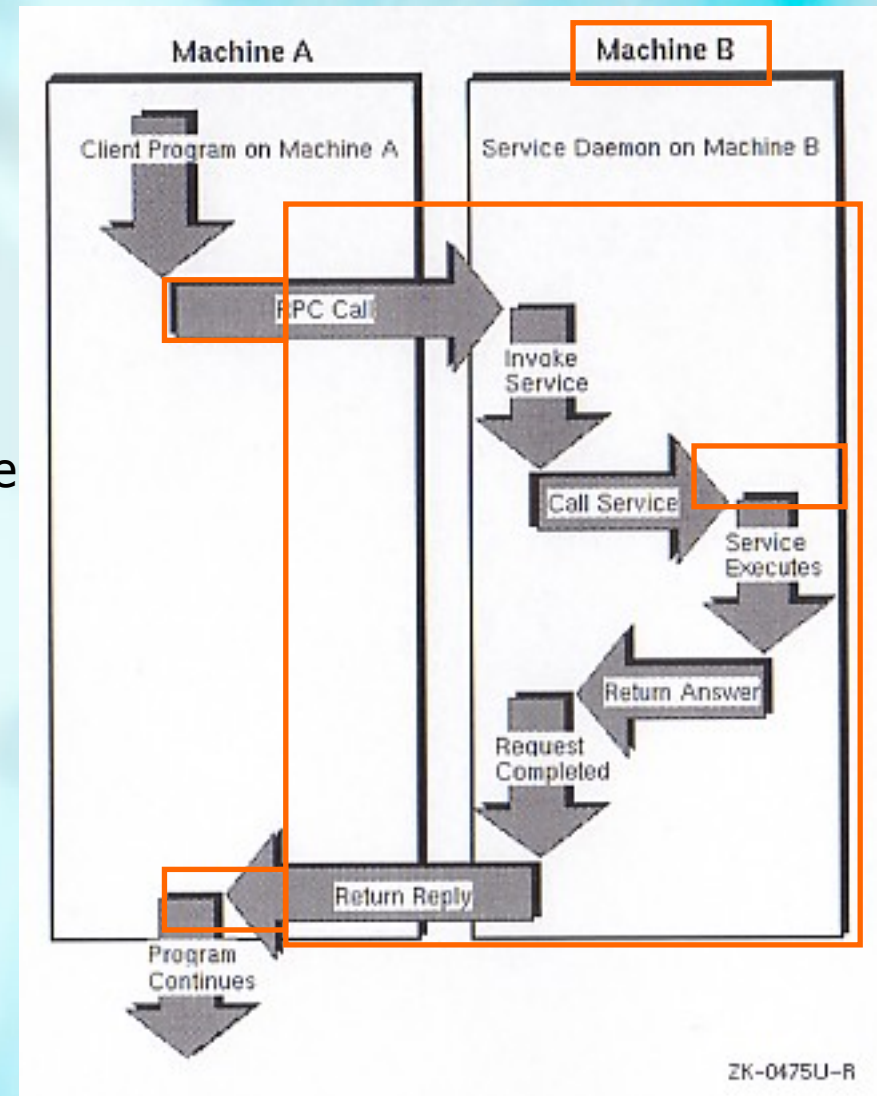
- Le protocole qui se cache derrière Napster est très simple
- Les messages sont symétriques
- Il n'y a qu'un format de message

Length	Type	Data
2 octets	2 octets	Length octets

- Le seul hic : les deux entiers sont en little endian (origine PC...)
 - 0x0001 est codé 0x01 0x00
 - 0x8000 est codé 0x00 0x80
- Type définit le type du message ... 146 messages différents

- eXternal Data Representation
- Remote Procedure Call
- IPSecure
- Secure Socket Layer / Transport Layer Security
- STCP - Stream Control Transmission Protocol
- IPv6
- Temps réel
 - Realtime Transport Protocol
 - Realtime Transport Control Protocol
 - Realtime profile
 - Resource Reservation Protocol

- Remote Procedure Call
- eXternal Data Representation
- Dans cadre il faut :
 - Identifier le serveur,
 - Identifier le paquetage, la procédure
 - Encoder les données d'entrée
 - Appeler la procédure distante
 - Décoder des données de sortie



- Besoin des prototypes `rpc/xdr.h` (ces fichiers n'existent pas sur PC)
 - `#include <rpc/xdr.h>`
- Création d'un flux XDR brut
 - `XDR xdrstream;`
- Initialisation en mode encodage/décodage et association d'un buffer pour un travail en mémoire :
 - `char buffer [MAX_SIZE];`
 - `xdrmem_create (&xdrstream, buffer, sizeof(buffer), XDR_ENCODE);`
 - `xdrmem_create (&xdrstream, buffer, sizeof(buffer), XDR_DECODE);`
- Initialisation pour un travail sur un flux C (FILE*)
 - `xdrstdio_create (&xdrstream, stdout, XDR_ENCODE);`

- Pour chaque type à transférer il n'y a qu'une opération à définir, c'est le flux qui décide le sens du transfert en fonction du sens du flux.
- Modèle : `bool_t xdr_<type> (XDR *xdrs, <type> *object);`
- Les opérations standards :
 - `bool_t xdr_char (XDR *xdrstream, char *object);`
 - `bool_t xdr_short (XDR *xdrstream, short *object);`
 - `bool_t xdr_int (XDR *xdrstream, int *object);`
 - `bool_t xdr_long (XDR *xdrstream, long *object);`
 - `bool_t xdr_u_short (XDR *xdrstream, unsigned short *object);`
 - `bool_t xdr_u_int (XDR *xdrstream, unsigned int *object);`
 - `bool_t xdr_u_long (XDR *xdrstream, unsigned long *object);`
 - `bool_t xdr_float (XDR *xdrstream, float *object);`
 - `bool_t xdr_double (XDR *xdrstream, double *object);`

l'utilisation est très simple : (encodage)

```
#include <rpc/xdr.h>
```

```
#define MAX_SIZE 1024
```

```
main ()
```

```
{
```

```
    XDR xdrstream1;
```

```
    char buffer [MAX_SIZE];
```

```
    xdrmem_create (&xdrstream1, buffer, sizeof(buffer), XDR_ENCODE);
```

```
    long ie = 20;
```

```
    xdr_long (&xdrstream1, &ie);
```

```
    float fe = 3.1415;
```

```
    xdr_float (&xdrstream1, &fe);
```

```
    char ce = '!';
```

```
    xdr_char (&xdrstream1, &ce);
```

l'utilisation est très simple : (decodage)

```
XDR xdrstream2;  
xdrmem_create (&xdrstream2, buffer, sizeof(buffer), XDR_DECODE);
```

```
long id; // il y aura 20  
xdr_long (&xdrstream2, &id);  
float fd; // il y aura 3.1415;  
xdr_float (&xdrstream2, &fd);  
char cd; // il y aura '!'  
xdr_char (&xdrstream2, &cd);
```

```
}
```

- Les chaînes de caractères sont également prévues :
 - `bool_t xdr_string (XDR *xdrstream, char **object, int lenmax);`
- Les tableaux d'octets sans contraintes :
 - `bool_t xdr_bytes (XDR *xdrstream, char **object, u_int *len, u_int lenmax);`
- Les enregistrements (il suffit de définir la procédure ...) exemple :
 - `struct point { float x, y; };`
 - `bool_t xdr_point (XDR *xdrstream, point *object) {`
 - `return xdr_float (xdrstream, p->x) && xdr_float (xdrstream, p->y);`
 - `}`
- Les tableaux d'objets :
 - `bool_t xdr_array (`
 - `XDR *xdrstream,`
 - `char **object,`
 - `u_int *len,`
 - `u_int lenmax,`
 - `u_int cellsize,`
 - `bool_t (* xdr_cell (XDR *, cell *))`
 - `);`

- Exemple de transfert de points

- `int psize;`
- `point [3] points1;`
- `char *p1 = (char *) points1;`
- `xdr_array (&xdrstream, &p1, &psize, 3, sizeof(point), xdr_point);`
- `char *p2 = NULL;`
- `psize = 0;`
- `xdr_array (&xdrstream, &p2, &psize, 3, sizeof(point), xdr_point);`
- `points [] points2 = (points []) p2;`
- `// les points sont dans p2 alloué dynamiquement...`

- En C++ selon μ Soft
 - CArchiveFile est une classe qui sérialise les données à la μ Soft
 - Peut être qu'il a fait le bon choix
- En Java
 - DataInputStream et DataOutputStream sont deux classes qui sérialisent correctement les données à la Java
 - Ils ont fait le bon choix pour les types de base et portable, mais pour des types plus complexes (p.e. String) chais pas !

- L'idée principale est d'appeler
 - une PROCEDURE (identifiée par son numéro)
 - contenue dans un PROGRAMME (identifié par un numéro)
 - qui est dans une version donnée
- Quand il y a numéro, il y a réservation et organisme
 - 0x00000000 -> 0x20000000 réservés pour Sun (!!!!)
 - 0x20000000 -> 0x40000000 libres
 - 0x40000000 -> 0xffffffff réservés
- Exemple de numéro réservé (/etc/rpc sur unix)
 - rpcbind 100000 portmap sunrpc rpcbind portmapper
 - rstatd 100001 rstat rup perfmeter
 - rusersd 100002 rusers
 - nfs 100003 nfsprog
 - mountd 100005 mount showmount

- Il y a 4 façons d'utiliser les RPC
- Haut niveau
 - Des fonctions préenregistrées et redéfinies en C dans la librairie locale
 - `int getrpcport (char * host, long program, long version, long protocol);`
 - `void rwall (char *host, char *message);`
 - ...
- Moyen niveau
 - On fabrique à la main tous les éléments...
- Rpcgen
 - On utilise un outil (rpcgen) pour fabriquer les squelettes
- Bas niveau
 - On retombe dans la socket de base

- On va faire un programme qui permet de réaliser :
 - `struct paire { float x, y; };`
 - `float add (paire);`
 - `paire sqrts (paire);`
- La procédure `xdr_paire` :
 - `bool_t xdr_paire (XDR *xdrstream, paire *p) {`
 - `return xdr_float (xdrstream, p->x) && xdr_float (xdrstream, p->y);`
 - `}`
- Les procédures de calculs :

<ul style="list-style-type: none">▪ <code>char *add (paire *p) {</code><ul style="list-style-type: none">• <code>static float res;</code>• <code>res = p->x + p->y;</code>• <code>return (char *) &res;</code>▪ <code>}</code>	<ul style="list-style-type: none">▪ <code>paire * sqrts {</code><ul style="list-style-type: none"><code>static paire res;</code><code>res.x = sqrt (p->x);</code><code>res.y = sqrt (p->y);</code><code>return (char *) &res;</code>▪ <code>}</code>
---	--

- Mise en place du serveur :

- `main () {`
 - `registerrpc (ARITH_PROG, ARITH_VERS, ADD_PROC, add, xdr_paire, xdr_float);`
 - `registerrpc (ARITH_PROG, ARITH_VERS, SQUARE_ROOTS_PROC, sqrts, xdr_paire, xdr_paire);`
 - `svc_run ();`
- `}`

- Exécution par le client :

- `main () {`
 - `paire p1, p2;`
 - `p1.x = 1; p1.y = 2;`
 - `callrpc (host, ARITH_PROG, ARITH_VERS, SQUARE_ROOTS_PROC,`
 - `xdr_paire, &p1, xdr_paire, &p2);`
 - `// résultats dans p2;`
- `}`

- Fabrique un fichier arith.x
 - `typedef struct paire { float x, float y } paire;`
 - `program ARITH_PROG {`
 - `version ARITH_VERS {`
 - `float add (paire) = 1`
 - `paire sqrts (paire) = 2`
 - `} = 1;`
 - `} = 0x20000001;`
- Utilise rpcgen qui génère :
 - `arith.h` -> le fichier qui définit les `#define ARITH_...`
 - `arith_clnt.c` -> le fichier qui décrit le squelette coté client
 - `arith_svc.c` -> le fichier qui décrit le squelette coté server

- Le fichier arith.c sur le serveur :
 - `float * add_1 (paire *p) {`
 - `static float res;`
 - `res = p->x + p-> y;`
 - `return &res;`
 - `}`

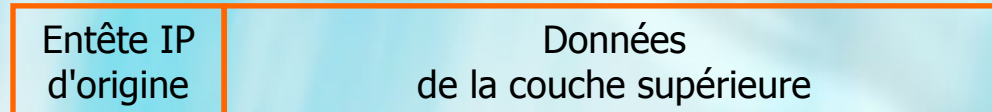
 - `paire * sqrts_1 (paire *p) {`
 - `static paires res;`
 - `res.x = p->x;`
 - `res.y = p-> y;`
 - `return &res;`
 - `}`
- Compilation, édition des liens, c'est fini.
- Le serveur répond sous "udp", "tcp", en serveur autonome ou sous inetd.

- Le client

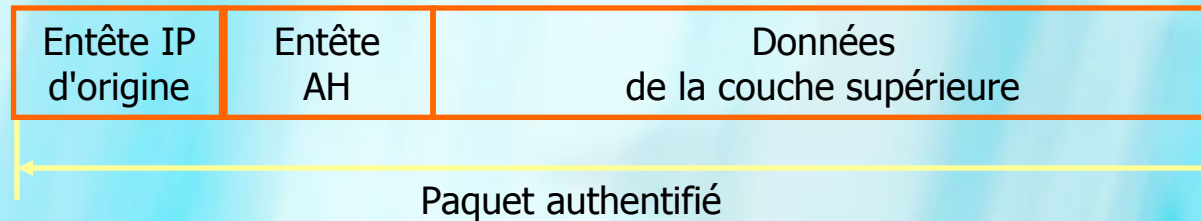
- `main () {`
 - `CLIENT *cl = clnt_create (host, ARITH_PROG, ARITH_VERS, "tcp");`
 - `paire p;`
 - `p.x = 10;`
 - `p.y = 20;`
 - `float *f = add_1 (&p, cl);`
 - `if (f == NULL) fprintf (stderr, "error\n");`
 - `else printf (stdout, "%g\n", *f);`
 - `paire *q = sqrts_1 (&p, cl);`
 - `if (q == NULL) fprintf (stderr, "error\n");`
 - `else printf (stdout, "%g %g\n", q->x, q->y);`
- `}`

- Protocole indépendant qui permet de garantir
 - Le contrôle d'accès
 - L'intégrité des données
 - L'authentification
 - La confidentialité
 - L'anti rejeu
- Deux protocoles sont employés
 - IP authentication Header (AH) : intégrité et authentification
 - Encapsulating Security Payload (ESP) : confidentialité
- Deux mode de fonctionnement
 - Mode transport : la sécurité est appliquée sur les données des couches hautes
 - Mode tunnel : un paquet IP est ré-encapsulé dans une trame IP
- Association de sécurité :
 - Indice de paramètre de sécurité (SPI)
 - Adresse IP destination
 - Identifiant du protocole de sécurité (AH ou ESP)

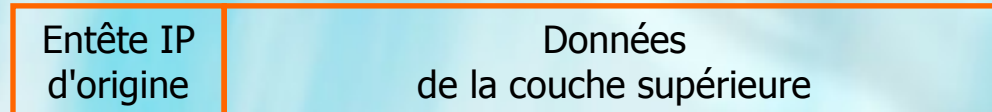
- Paquet avant application de AH



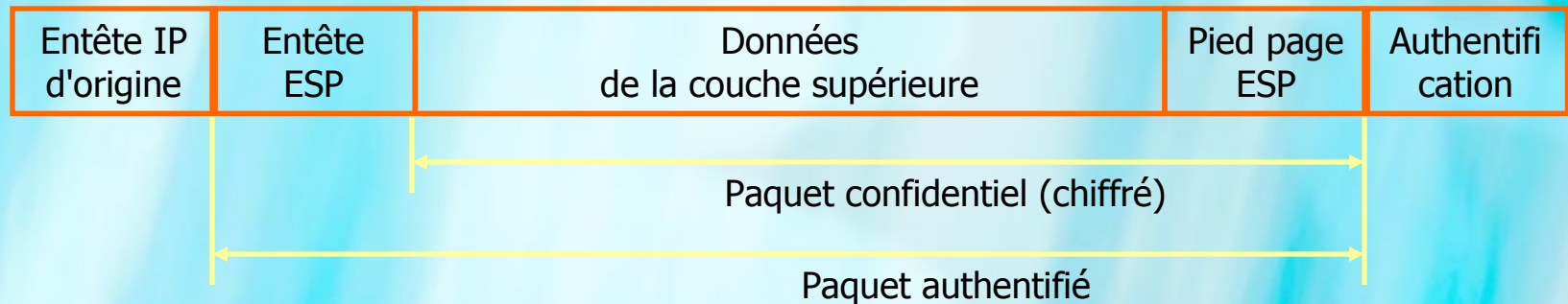
- Paquet après application de AH



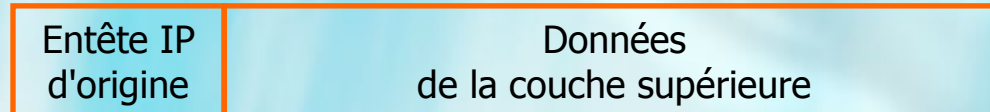
- Paquet avant application de ESP



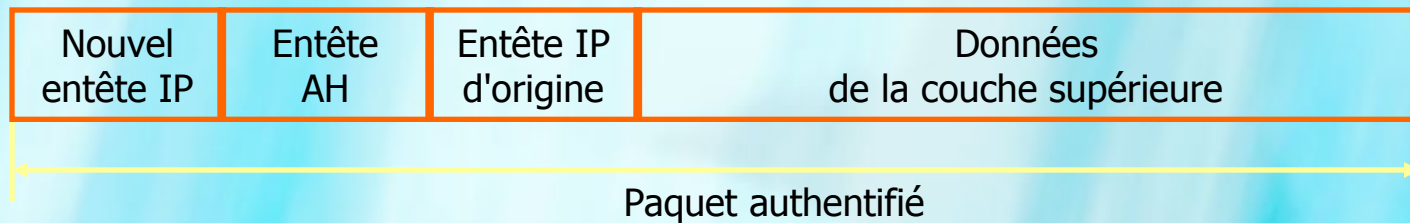
- Paquet après application de ESP



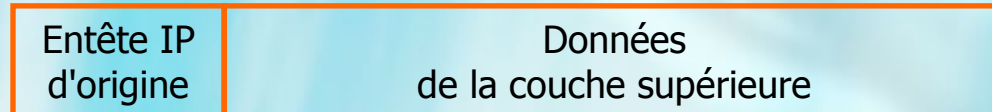
- Paquet avant application de AH



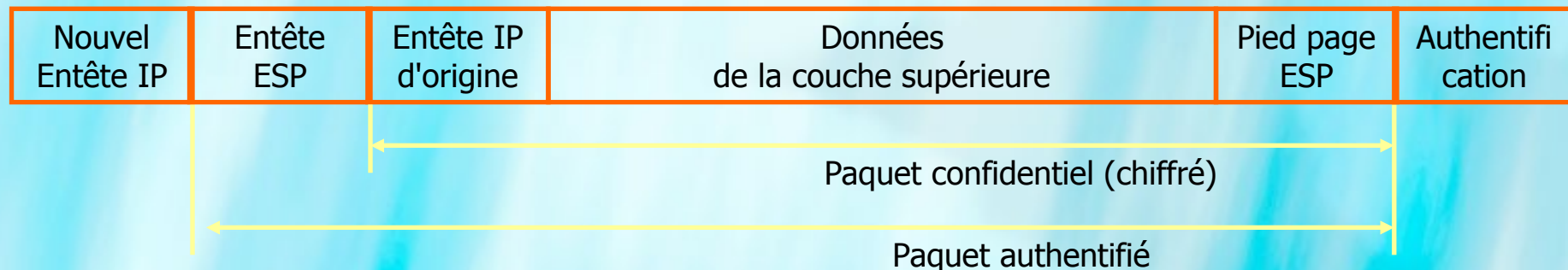
- Paquet après application de AH



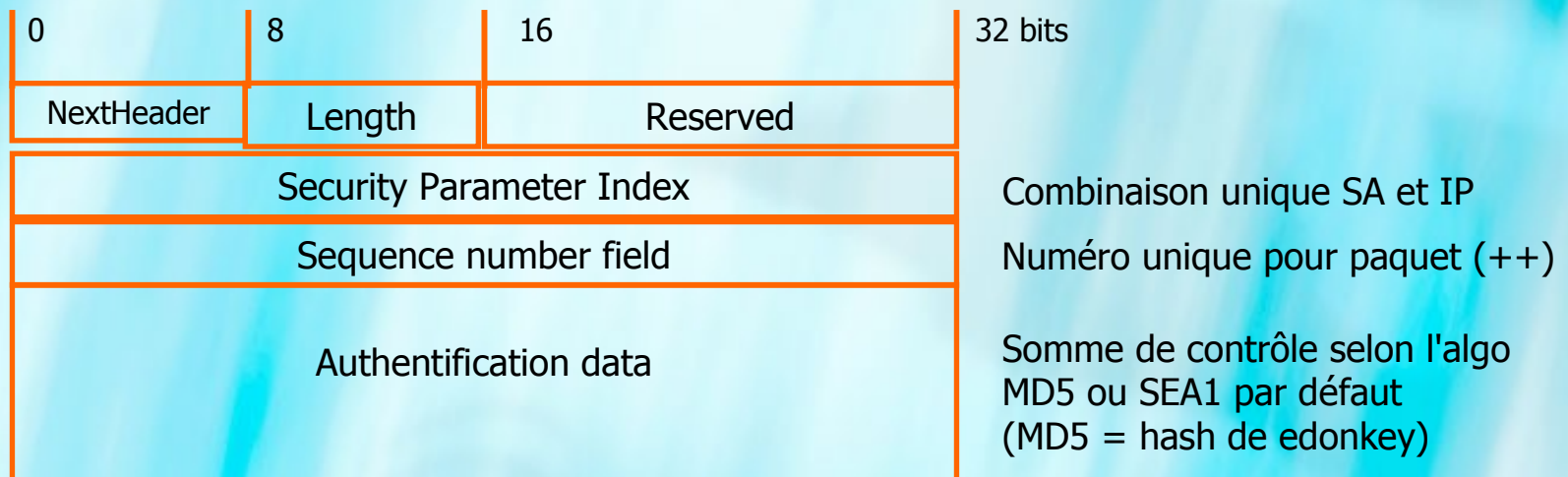
- Paquet avant application de ESP



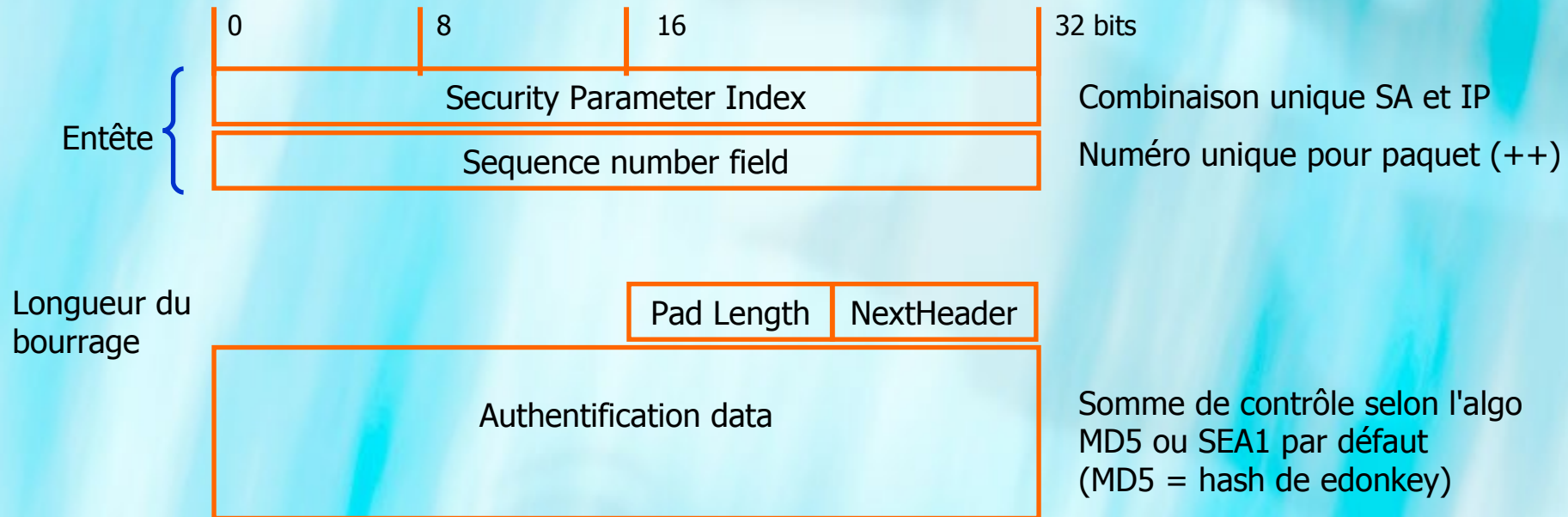
- Paquet après application de ESP



- Intégrité des données
- Authentification de l'origine
- Protection contre le replay
- Le calcul des données d'authentification utilise tous les champs (IP et couche supérieure) qui peuvent avoir une valeur prévisible à l'arrivée



- Confidentialité
- Intégrité des données
- Authentification de l'origine
- Protection contre le re-jeu

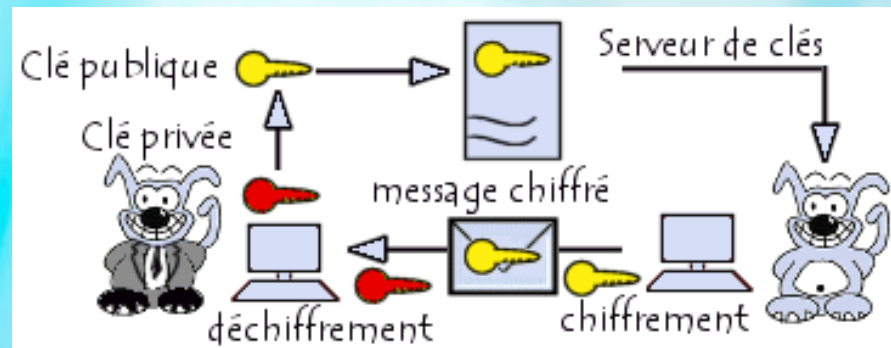


- Il reste le problème des clefs de cryptage
- Ce problème est crucial car c'est lui qui propose les garanties
- Il y a plusieurs solutions proposées par IPSec
- En principe il y aura un échange de clé public et clé privée
- L'échange se fait à l'aide d'un des deux protocoles (indépendant de IPSec)
 - ISAKMP
 - IKE

- Secured Socket Layer est un protocole de Netscape
- Transport Layer Security est la version normalisée par IETF
- SSL et/ou TLS sont complètement transparents
- Le principe est de négocier le cryptage à l'ouverture de la connexion (éventuellement renégocié par la suite)
- Cette négociation peut coûter cher en ressources
 - Mémoire
 - Calcul
 - etc...
- Je peux suggérer la consultation du livre
 - Network Security with OpenSSL (le chapitre 1 est sur le net)

• Modèle de transaction sécurisée par SSL (extrait de CCM)

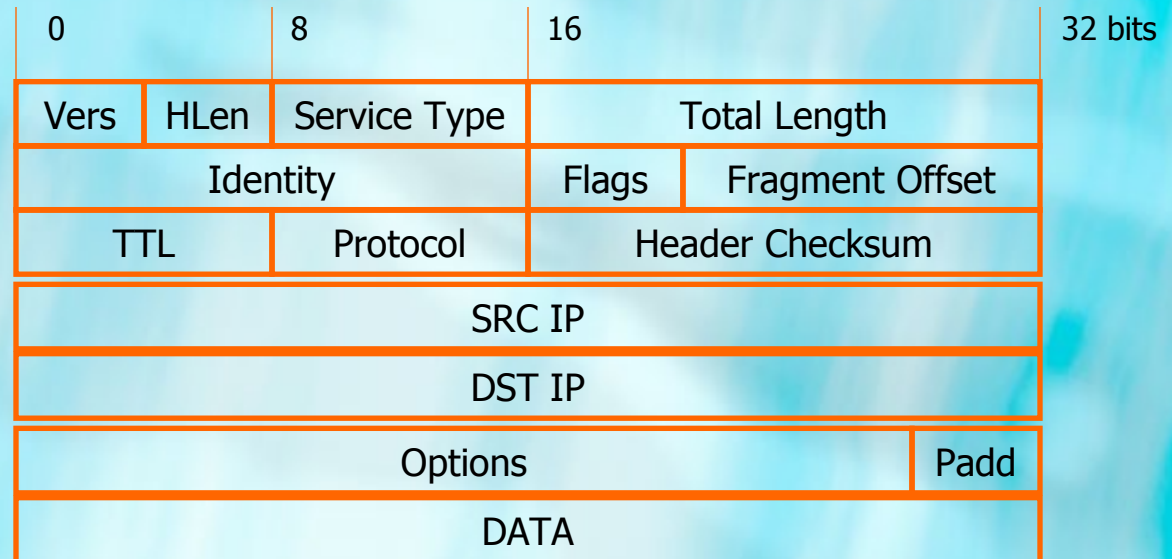
- Le client se connecte au serveur sécurisé par SSL et lui demande de s'authentifier. Le client envoie également la liste des cryptosystèmes qu'il supporte, triée par ordre décroissant selon la longueur des clés.
- Le serveur à réception de la requête envoie un certificat au client, contenant la clé publique du serveur, signée par une autorité de certification (CA), ainsi que le nom du cryptosystème le plus haut dans la liste avec lequel il est compatible (la longueur de la clé de chiffrement - 40 bits ou 128 bits - sera celle du cryptosystème commun ayant la plus grande taille de clé).
- Le client vérifie la validité du certificat (donc l'authenticité du serveur), puis crée une clé secrète aléatoire, chiffre cette clé à l'aide de la clé publique du serveur, puis lui envoie le résultat (la clé de session).
- Le serveur est en mesure de déchiffrer la clé de session avec sa clé privée. Ainsi, les deux entités sont en possession d'une clé commune dont ils sont seuls connaisseurs. Le reste des transactions peut se faire à l'aide de clé de session, garantissant l'intégrité et la confidentialité des données échangées.



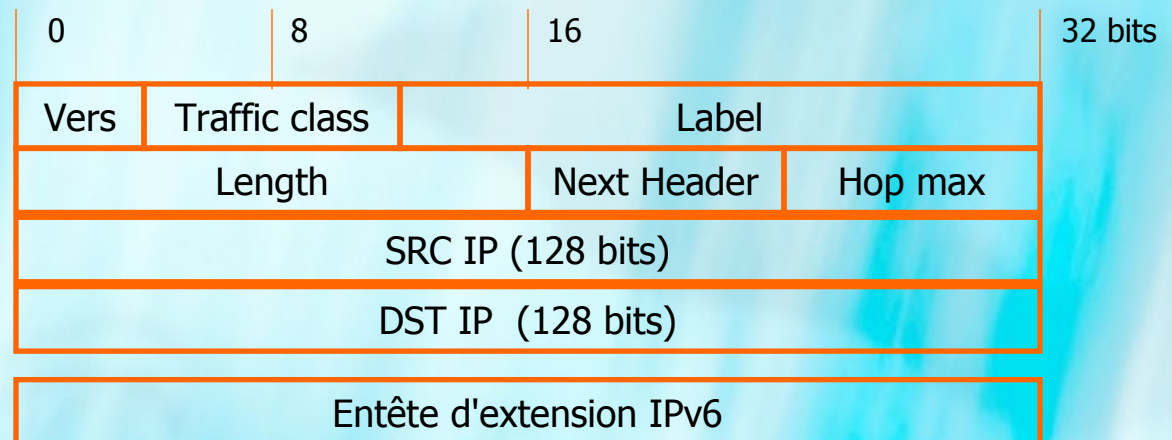
- TCP est très bon, mais trop rigide
- UDP est très simple, mais trop simple
- SCTP
 - Permet la connexion de plusieurs flux indépendant vers la même machine
 - Chaque flux émet/reçoit à son rythme
 - Chaque paquet est numéroté à l'expédition
 - Régulièrement le récepteur envoie un accusé de réception global (pour un ensemble de paquets)
 - Quand des trous sont détectés, l'émetteur peut tenter de les combler
 - Le récepteur peut expliquer qu'il n'a plus besoin du paquet
 - Si un flux se bloque car il y a un trou, les autres ne se bloquent pas

- Avec la saturation actuelle et prévisible des adresses IP il faut prévoir un nouvel adressage
- Une autre solution aurait été de mettre un routeur sur une adresse unique (domicile) ou un ensemble d'adresses pour une entreprise plus grande
- IPv6 ou Ipng ou IP new generation est née
- L'esprit de IPv6 est de :
 - Simplifier le datagramme IP
 - Mettre toute les fioritures nécessaires à un protocole quelconque dans des "entêtes d'extensions"

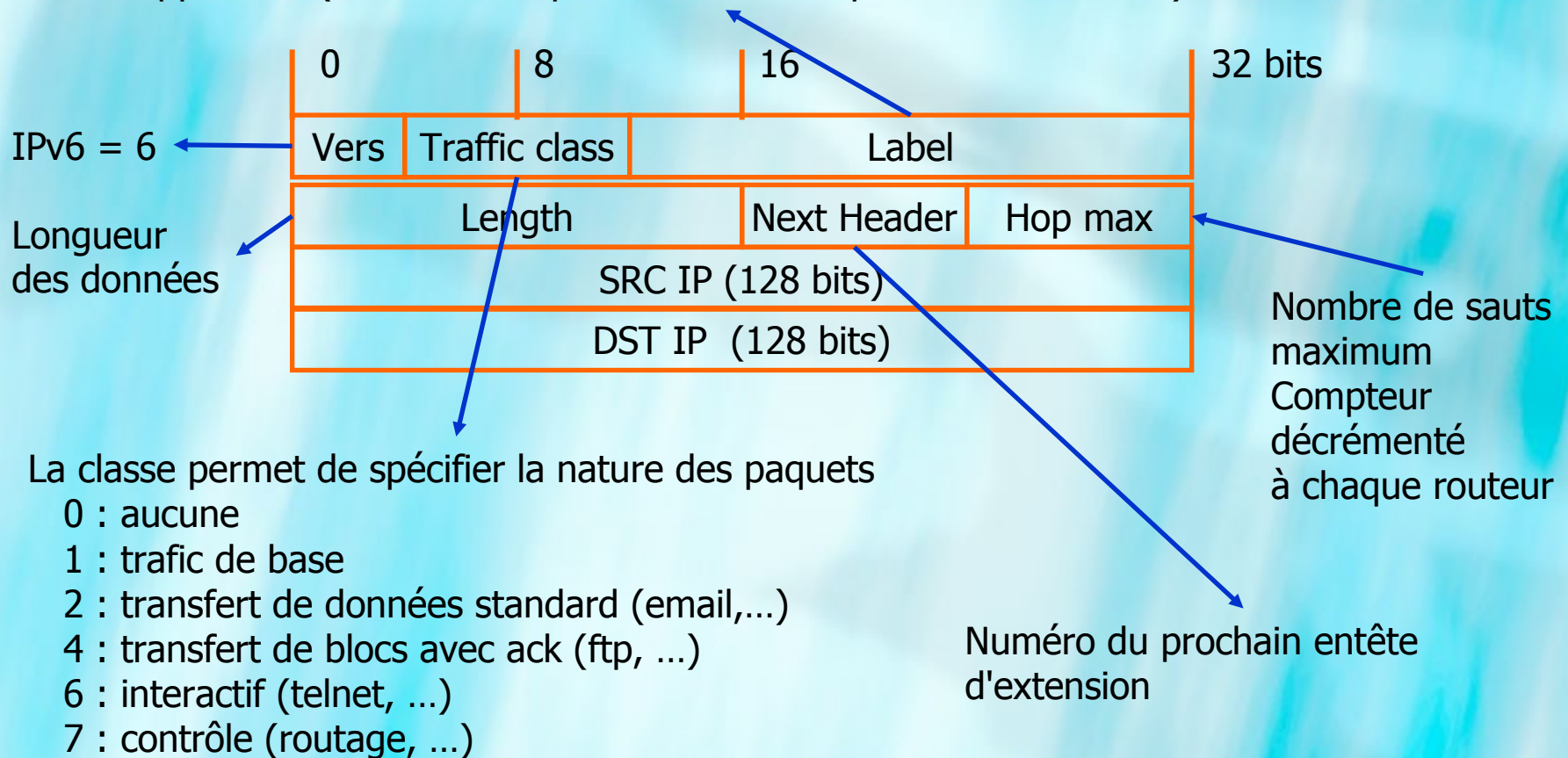
- Rappel en IPv4



- Et en IP v6



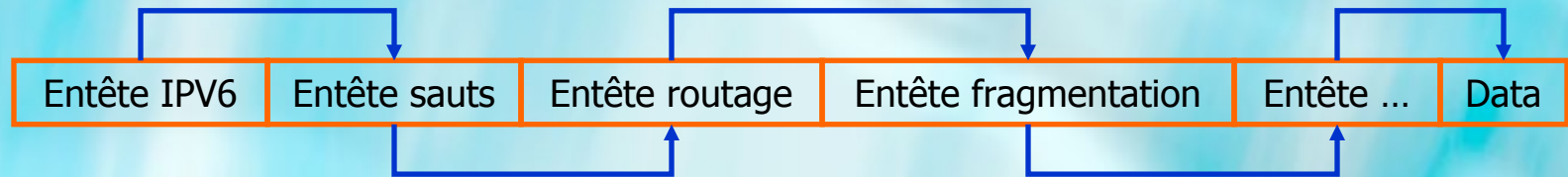
Le label du paquet permet d'étiqueter le paquet dans le flot auquel il appartient (référence la qualité de service qui lui sera attachée)



Adressage géographique !
 10^{23} adresses par m² de la surface terrestre

- De nombreux entêtes d'extension sont prévues :
 - Index désignation
 - 0 Options saut après saut
 - 4 IP
 - 6 TCP
 - 17 UDP
 - 43 Routage
 - 44 Fragmentation
 - 45 Routage inter domaine
 - 46 Réserve de ressources
 - 50 Encapsulation de charge utile sécurisée
 - 51 Authentification
 - 58 ICMP
 - 59 plus rien
- Ils se succèdent dans un ordre car les dispositifs intermédiaires peuvent les traiter au fur et à mesure et éventuellement simplifier le paquet

- Les entêtes d'extension sont cascades



- La solution généralisée pour chaque entête :



- Chaque extension gère le champs données comme il veut
 - Fixe,
 - TLV,
 - Fixe+longueur variable,
 - etc.

- Trois sortes d'adresses :

- Unicast une interface le paquet avec cette adresse arrive sur cette interface
- Anycast un ensemble d'interfaces, le paquet pour cette adresse arrive sur l' interface la plus proche
- Multicast un ensemble d'interfaces, le paquet pour cette adresse arrive sur toutes les interfaces

- Plus de broadcast (le multicast est plus performant)

- Représentation sous la forme de nombre hexa par groupe de 16 bits

- FEDC:BA98:7654:3210:FEDC:BA98:7654:3210
- FEDC::3210
- ::0A01:4203

- Type des adresses :

Address type	Binary prefix	IPv6 notation
Unspecified	00...0 (128 bits)	::/128
Loopback	00...1 (128 bits)	::1/128
Multicast	11111111	FF00::/8
Link-local unicast	1111111010	FE80::/10
Site-local unicast	1111111011	FEC0::/10
Global unicast	001...	2000::/3
Unassigned	le reste	(85% le l'espace disponible)

- Unspecified (:::/128)
 - Ne pas affecter, elle permet de le déposer dans une trame en attendant que l'adresse arrive (BOOTP, ...)
- Loopback (:::1/128)
 - Ne pas affecter, elle permet de spécifier l'interface courante.
- Link-local
 - C'est une adresse privée. Le routeur ne route pas. Le numéro d'interface est de 64 bits
- Site-local
 - C'est une adresse privée. Seul le routeur qui voit la machine utilise cette adresse. Le numéro d'interface est de 64 bits. Le numéro de sous-réseau est de 54 bits.
- IPv4 in IPv6 (:::0:0000:a.b.c.d)
 - Cette adresse peut représenter une interface IPv4 qui passe dans un tunnel IPv6
 - :::0 représente 80 bits à 0, puis 0x0000 et enfin 'abcd' le numéro IPv4
- IPv4only in IPv6 (:::0:FFFF:a.b.c.d)
 - Cette adresse peut représenter une interface IPv4 encodée en IPv6 qui communique qu'avec du IPv4
 - :::0 représente 80 bits à 0, puis 0xFFFF et enfin 'abcd' le numéro IPv4

- Global unicast

- 2000:: allouées avant l'arrivée 'commerciale' d'IPv6
- 2001:: allouées depuis 2001
- Adresse standard pour une interface
- Elle était découpé en six parties (TLA, SLA, NLA, etc)
- Maintenant elle est découpée en trois parties
 - N bits pour le préfixe du réseau (48)
 - M bits pour le sous-réseau (16)
 - P bits pour le numéro de l'interface (64)
- Le numéro de l'interface est le mélange entre
 - Des indicateurs (2)
 - Le numéro de compagnie (14)
 - Le numéro ethernet (48)

- Multicast
 - Adresse découpée en quatre parties
 - 8 bits pour le préfixe multicast (0xFF)
 - 4 bits pour des flags
 - 4 bits pour la portée
 - 112 bits pour le numéro de groupe
 - Préfixe multicast
 - FF
 - Flags
 - 3 MSB à 0
 - LSB 0=adresse affectée par l'IANA, 1=adresse temporaire
 - Portée
 - 1 locale à l'interface
 - 2 locale au lien
 - 4 locale aux liens
 - 5 locale au site
 - 8 locale à une organisation
 - 15 globale
- Le RFC 3849 demande une adresse non routable pour la documentation
 - :2001:DB8::/32

- Longueur des paquets IPv6 demande des paquets de 1280 octets au minimum
- PPP peut poser des problèmes car le MTU peut être faible, très faible
- Un tunnel sous IP a besoin d'octets supplémentaires pour opérer
- Les sommes de contrôles des couches supérieures pensées en IPv4 (@32 bits maintenant @128bits) ne sont plus correctes

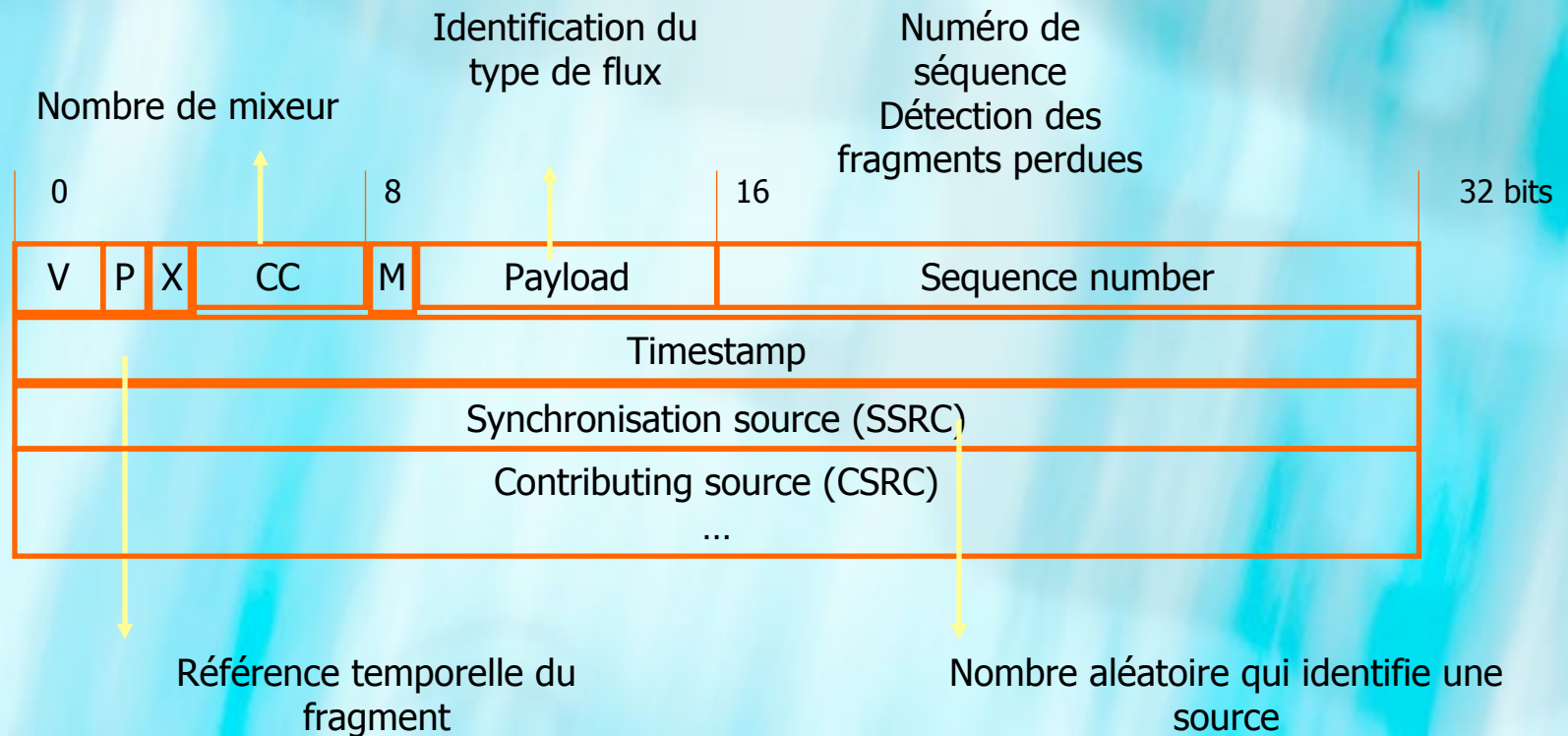
- Le temps réel sur le réseau internet est particulier
- L'internet est basé sur le "best effort" : on fait au mieux...
- Dans ces conditions, faire du Temps Réel n'est pas facile.
- Et pourtant il faut proposer des solutions sinon l'internet va disparaître.
- But : proposer des fonctions de transport de bout en bout pour les applications temps réel (Temps réel est synonyme de vidéo)
 - Realtime Transport Protocol - RFC 1889
 - Profile videoconference - RFC 1890
 - Profile H.261 video streams - RFC 2032
 - Profile video JPEG - RFC 2035
 - Profile video mpeg1 & mpeg2 - RFC 2038

- Une vidéo conférence :
 - C'est un flux vidéo (images/secondes)
 - C'est un flux audio (échantillon/secondes)
 - C'est une synchronisation des deux flux
- RTP propose une solution :
 - Pour dater les composants des deux (n) flux
 - Pour numéroter les fractions des composants des flux
 - Détecter les pertes de paquets
 - Détecter les arrivées retardées des paquets
- De plus le RFC ne traite pas tous les formats possible, mais laisse la porte ouverte à des formats définis sous la forme de RFC spécifique (payload).
- Néanmoins, la standardisation va permettre de mettre en place des négociation sur les formats et les encodages.



RTP pour transporter les flux
RTCP pour contrôler la qualité des flux
RSVP pour réserver des ressources (bande passante)

- A la base le flux est découpé en segments de durée fixe (20ms)
- Chaque segment est émis dans un paquet RTP avec entête



- L'arrivée et le départ de participants dans la conférence
 - Construire une table des participants et de la qualité de service
- RTCP prend ces informations en compte à l'aide de 5 messages
 - Receiver Report : paquet émis par des participants passifs, il y a une liste d'indicateurs sur la qualité des paquets réceptionnés
 - Sender Report : paquet émis par des participants actifs, même données avec en plus des informations liées à l'émission
 - BYE : paquet émis lors du départ d'un participant
 - SDES : Description de la source (CSRC)
 - APP : extension...
- Les paquets RTCP sont réguliers (et revalident les compteurs des présences)
- Les paquets RTCP ne doivent pas dépasser 5% du débit de la session (faut pas que le contrôle consomme la bande passante)

- Dans le cas d'une conférence à trois
 - Deux participants sont à haut débit
 - Un participant est à bas débit
- A l'aide du protocole RTCP le participant bas débit est détecté, des décisions peuvent être prises :
 - L'objectif est de conserver les caractéristiques optimales pour les deux participants à haut débit et d'adapter les flux pour le participant à débit limité
 - Couper le flux (vidéo) uniquement pour le participant à débit faible
 - Changer l'encodage d'un flux pour limiter le coût de transport
- L'adaptation se fait en intercalant des composants spéciaux
 - Translator
 - Mixer

- Translator

- Manipule des flux de données différents
- Si il retransmets les paquets RTP avec des encodage différents, il re-numérote les paquets
- Indétectable

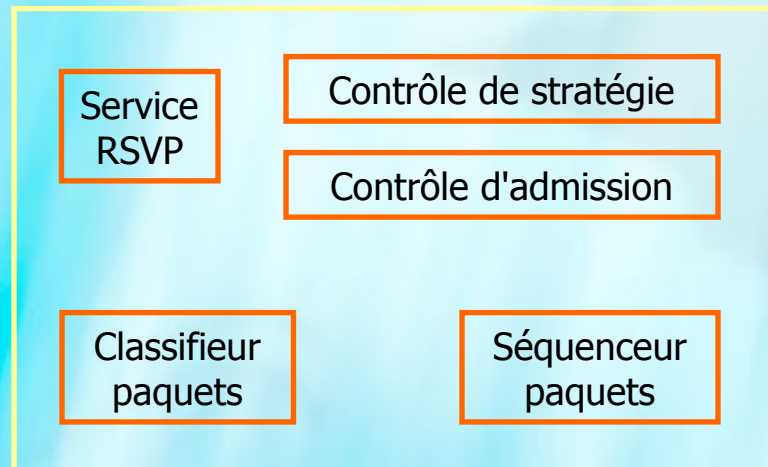
- Mixer

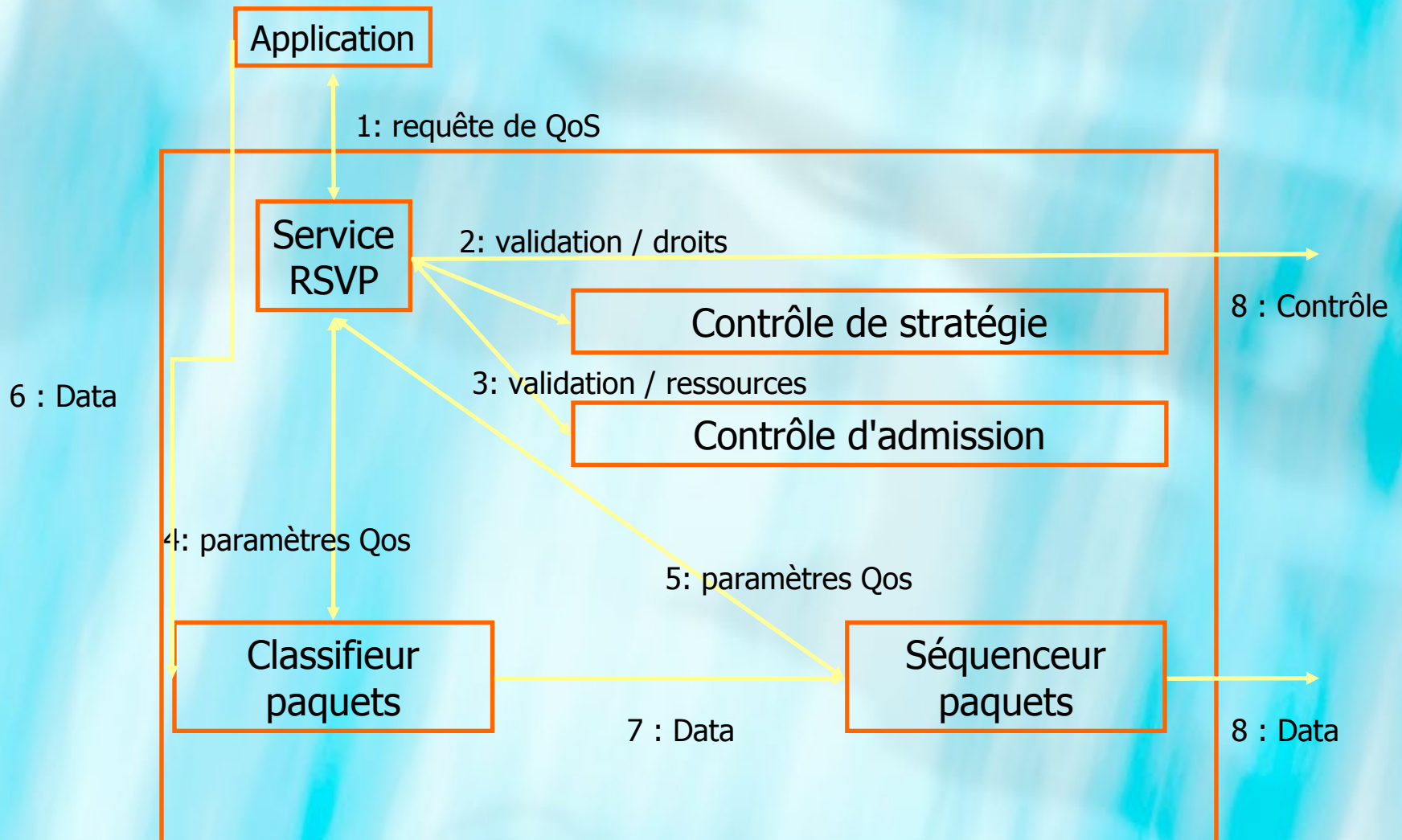
- Combine les flux de sources multiples
- Fabrique un flux qui est resynchronisé
- Les différentes sources sont dans la liste des contributeurs du paquet

- Les profiles (payload) définissent dans les RFC :
 - Les types (audio, video,
 - Les caractéristiques selon le type
- Pour un format audio
 - Les fréquences d'échantillonnage (44100, 24000, etc.)
 - La durée
 - Les canaux (mono, stéréo, quadri, THX, etc.)
 - Les formats (PCMA, PCMI, GSM, etc.)

- Sans une réservation de ressources le temps réel est impossible
- RSVP utilise deux types de message
 - PATH qui va de l'émetteur vers le récepteur, il permet de montrer le chemin et les besoins
 - RESV qui va du récepteur vers l'émetteur, il met en place la réservation

- La réservation de ressource utilise 5 modules :
 - Service RSVP : un service qui opère sur la machine locale
 - Contrôle de stratégie : un module qui valide les droits de la demande
 - Contrôle d'admission : un module qui valide le niveau de qualité de la demande
 - Classifieur de paquets : un module qui va classer les paquets en fonction de la route et de la QoS
 - Séquenceur paquets : un module qui va émettre les paquets dans le bon ordre



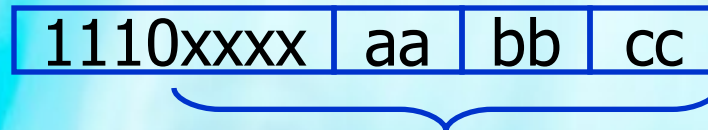


- Internet doit constamment s'adapter
- Internet propose deux catégories de transport
 - IntServ : Integrated Service
 - Gestion de flots indépendants les uns des autres
 - Gestion pour les routeurs entre grandes entites
 - Trois classes de services : Service garanti, charge contrôlée, au mieux.
 - DiffServ : Differentiated Service
 - Regroupement de plusieurs flots dans un super flot
 - Solution pour les routeurs intérieurs
 - Tous les deux proposent trois classes de services
 - Service garanti, (RSVP)
 - Charge contrôlée,
 - Au mieux.

- Le réseau devait avoir la notion de diffusion multiple
 - Journaux, vidéo, conférences, films, etc.
- Une diffusion multiple sérieuse pose les problèmes
 - Elle doit se faire sans augmenter le nombre de paquets sur le réseau
 - Si pas de réplication de paquet alors il ne doit pas avoir de serveur
 - S'il n'y a pas serveur, il n'y a plus de clients
 - Comment une diffusion multiple traverse les routeurs
 - Problème de l'inondation des réseaux
- Il faut créer une nouvelle notion fédératrice
 - La notion de client/serveur n'est plus acceptable
 - Le notion de groupe "social" est proposée

- Unicast ou 1 vers 1
 - C'est le modèle standard
- Multicast ou 1 vers N
 - Diffusion vers plusieurs correspondants (radio, télévision, ghost, *box, etc.)
- Multicast ou N vers 1
 - Collation de données de sources multiples (gestion, supervision, etc.)
- Multicast ou N vers M
 - Communication simultanée entre plusieurs correspondants (vidéo, conférence, jeux, etc.)
- Les protocoles de transport sous-jacents seront toujours sous UDP
 - Pas d'accusé de réception
 - Gestion des flux par la couche supérieure au transport (session, présentation application)

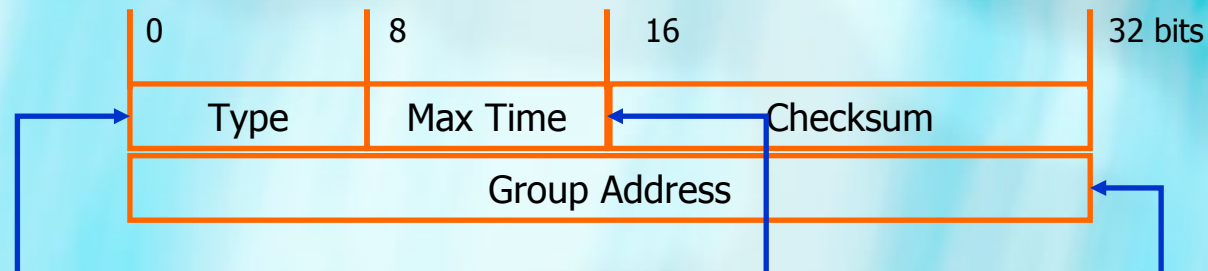
- Il faut créer une nouvelle notion fédératrice
 - La notion de client/serveur n'est plus acceptable
 - La notion de groupe "social" est proposée
- Caractéristiques d'un groupe :
 - Un groupe à une taille illimité !!!
 - Un groupe peut être joint ou quitté à tous moment
 - Un groupe ne propose pas d'identifier ses membres
- Matérialisation d'un groupe
 - Chaque groupe "social" est traduit en un groupe réseau
 - Utilise une adresse IP dans la classe D
 - Classe D : 224 -> 239 : 268M d'adresses donc de groupes



Numéro du groupe

- La carte réseau doit écouter les paquets d'un groupe
- Une trame est identifiée par son émetteur et son destinataire
 - Comment une adresse ethernet unique peut adresser plusieurs machine ?
- Une adresse ethernet virtuelle est utilisée :
 - 0x01 0x00 0x5E + les 23 bits de poids faibles du numéro de groupe
- Une carte réseau devient de plus en plus complexe...
 - Elle doit écouter
 - Son adresse ethernet
 - L'adresse de broadcast
 - Les adresses de groupe de multicast convertie en ethernet
 - L'adresse du groupe de surveillance (224.0.0.1) convertie en ethernet

- Il faut un protocole qui se charge de répertorier
 - les groupes
 - les membres des groupes
- Internet Group Management Protocole - RFC 2236

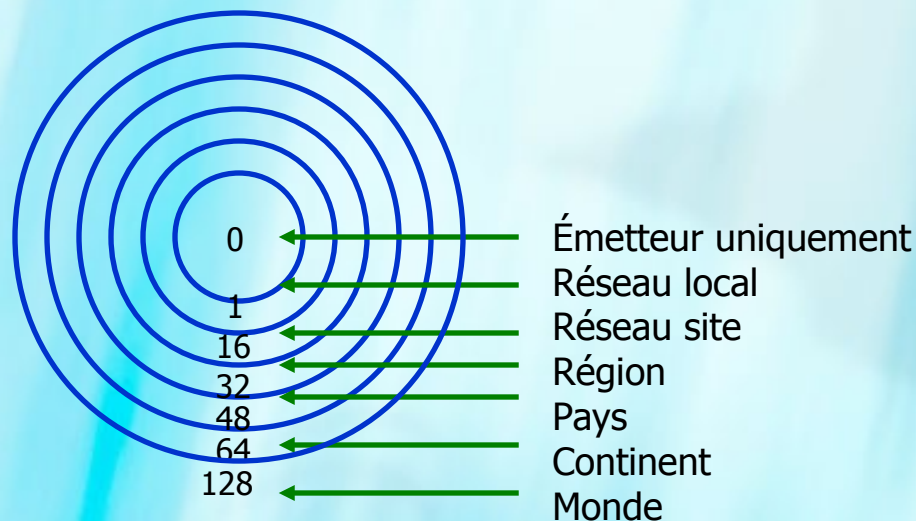


0x11 Query : liste des groupes actifs
0x16 Membership report : rejoindre un groupe
0x17 Leave group : quitter un groupe

Durée d'attente maximum (Query)

L'adresse du groupe concerné

- Des groupes de portée internationale
 - Inutilisable et comment on fait pour le privée
- Les groupes ont une portée variable
 - Il suffit de gérer le TTL de chaque paquet IP (Time To Live)
 - Le TTL est décrémenté à chaque passage de routeur
 - Selon la configuration du routeur des seuils de passage sont mis en place



- 224.0.0.0 -> 224.0.0.255 : adresses administratives non routées
 - 224.0.0.0 Base Address (Reserved)
 - 224.0.0.1 All Systems on this Subnet
 - 224.0.0.9 RIP2 Routers
- 224.0.19.0 -> 224.0.19.63 : Wald Disney Company
- 224.1.0.0 -> 224.1.255.255 : ST Multicast Groups
- 224.2.0.0 -> 224.2.255.255: Multimedia Conference Calls
- 239.0.0.0 -> 239.255.255.255 adresses locales filtrées par le routeur de frontière
- 239.192.0.0 -> 239.251.255.255: adresse locale filtré par l'organisation
- 239.255.0.0 -> 239.255.255.255: adresse locale filtrée par le site

- L'API pour faire de la multidiffusion c'est ... la socket !
- Il faut juste ajouter des primitives pour :
 - Changer le TTL
 - Joindre un groupe
 - Quitter un groupe
 - Avoir plusieurs applications en lecture et écriture sur la même socket sur le même PC
- La primitive existe :
 - `int setsockopt (SOCKET s, int level, int optname, const char *optval, int optlen);`
 - level : couches concernées (SOL_SOCKET, IPPROTO_IP, IPPROTO_TCP, ...)
 - optname : nom de l'option
 - optval : pointeur opaque sur une structure décrivant les paramètres de l'option
 - Optlen : taille de la structure

- Changer le TTL

- `int ttl=1;`
- `if (setsockopt (sockm, IPPROTO_IP, IP_MULTICAST_TTL,`
- `(const char *) &ttl, sizeof(ttl)) == SOCKET_ERROR)`
- `throw exception ("Invalid multicast TTL value");`

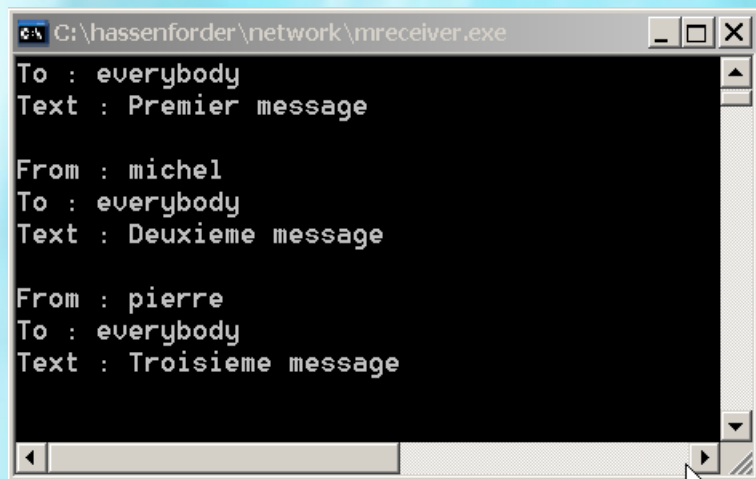
- Joindre un groupe

- `struct ip_mreq mreq;`
- `mreq.imr_multiaddr.s_addr = inet_addr(group);`
- `mreq.imr_interface.s_addr = htonl(INADDR_ANY);`
- `if (setsockopt(sockm, IPPROTO_IP, IP_ADD_MEMBERSHIP,`
- `(char *) &mreq, sizeof(mreq)) == SOCKET_ERROR)`
- `throw exception ("unjoinable group");`

- Avoir plusieurs applications en lecture et écriture sur la même socket sur le même PC

- `int dummy = 1;`
- `if (setsockopt(sock, SOL_SOCKET, SO_REUSEADDR,`
- `(char *) &dummy, sizeof(dummy)) == SOCKET_ERROR)`
- `throw exception ("no multiple application");`

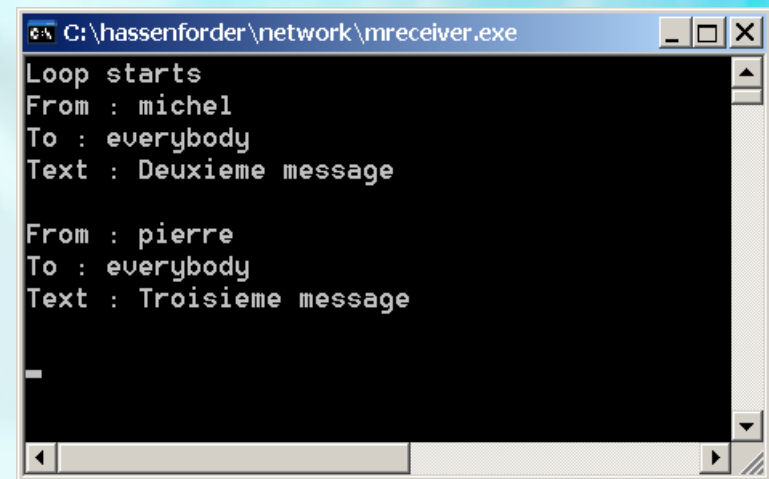
Récepteur qui attend des messages



```
C:\hassenforder\network\mreceiver.exe
To : everybody
Text : Premier message

From : michel
To : everybody
Text : Deuxieme message

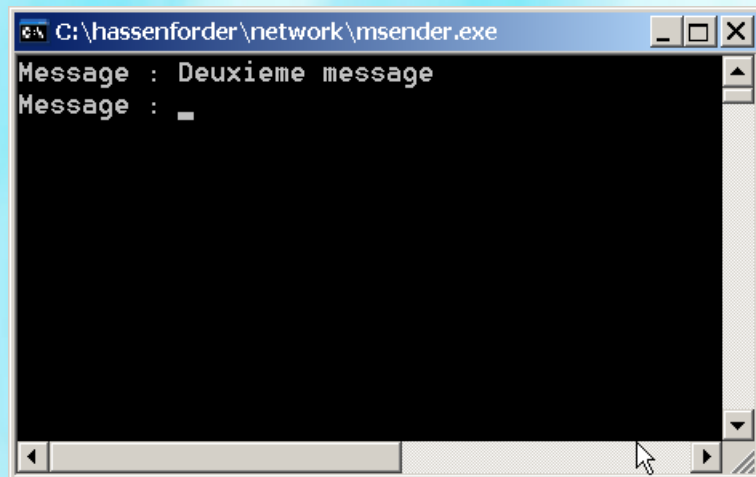
From : pierre
To : everybody
Text : Troisieme message
```



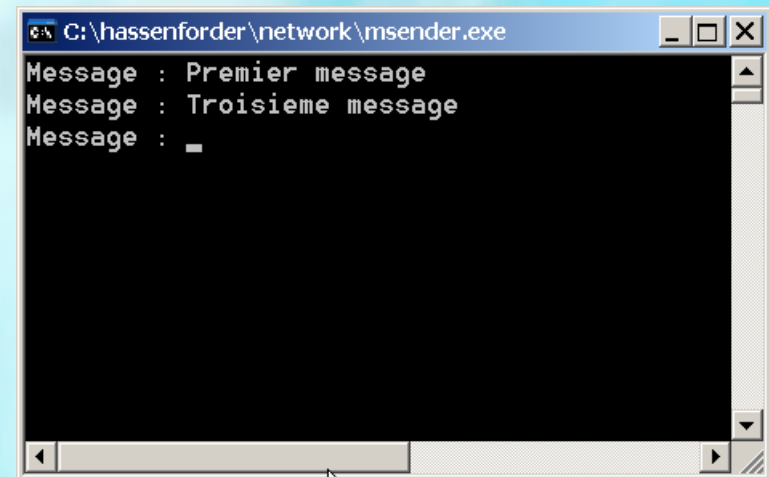
```
C:\hassenforder\network\mreceiver.exe
Loop starts
From : michel
To : everybody
Text : Deuxieme message

From : pierre
To : everybody
Text : Troisieme message

-
```



```
C:\hassenforder\network\msender.exe
Message : Deuxieme message
Message : -
```



```
C:\hassenforder\network\msender.exe
Message : Premier message
Message : Troisieme message
Message : -
```

Emetteur qui envoie une simple chaîne

```
int error;
SOCKET sockm;
if ((sockm = socket(PF_INET, SOCK_DGRAM, 0)) < 0)
    throw exception ("Invalid Socket");
int re_use = 1;
error = setsockopt(sockm, SOL_SOCKET, SO_REUSEADDR, (char *) &re_use, sizeof(re_use));
if ( error == SOCKET_ERROR) throw exception ("no multiple application");

struct sockaddr_in srm;
memset(&srm, 0, sizeof(srm));
srm.sin_family = AF_INET;
srm.sin_port = htons (49001);
srm.sin_addr.s_addr = INADDR_ANY;
if (bind(sockm, (struct sockaddr *)&srm, sizeof(srm)) == SOCKET_ERROR)
    throw exception ("Invalid Socket");

struct ip_mreq mreq;
mreq.imr_multiaddr.s_addr = inet_addr("224.137.194.129");
mreq.imr_interface.s_addr = htonl(INADDR_ANY);
error = setsockopt(sockm, IPPROTO_IP, IP_ADD_MEMBERSHIP, (char *) &mreq,
sizeof(mreq));
if (error == SOCKET_ERROR) throw exception ("Unjoinable group");
```

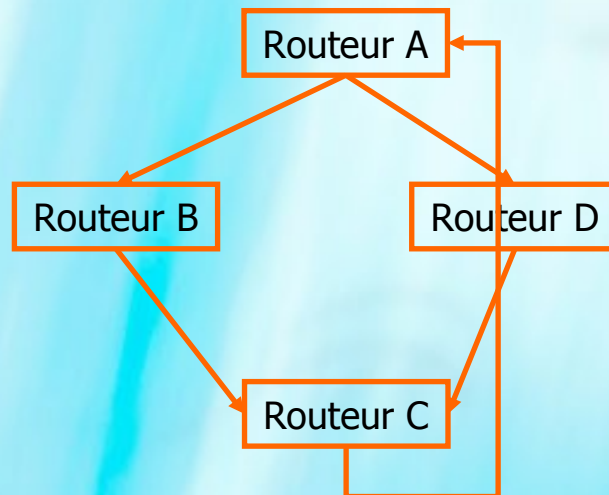


```
cout << "Loop starts" << endl;
for (;;) {
    char message [256];
    int len = sizeof (srm);
    int cnt = recvfrom(sockm, message, sizeof(message), 0, (struct sockaddr *) &srm, &len));
    if (cnt == SOCKET_ERROR )
        throw exception ("Receive fail");
    message[cnt] = '\0';
    cout << message << endl;
}
```

```
int error;  
SOCKET sockm;  
if ((sockm = socket(PF_INET, SOCK_DGRAM, 0)) == SOCKET_ERROR)  
    throw exception ("Invalid Socket");  
int ttl = 1;  
error = setsockopt (sockm, IPPROTO_IP, IP_MULTICAST_TTL, (const char *) &ttl, sizeof(ttl));  
if ( error == SOCKET_ERROR) throw exception (« Multicast TTL");  
  
struct sockaddr_in stm;  
memset(&stm, 0, sizeof(stm));  
stm.sin_family = AF_INET;  
stm.sin_port = htons(49001);  
stm.sin_addr.s_addr = inet_addr("224.137.194.129");
```

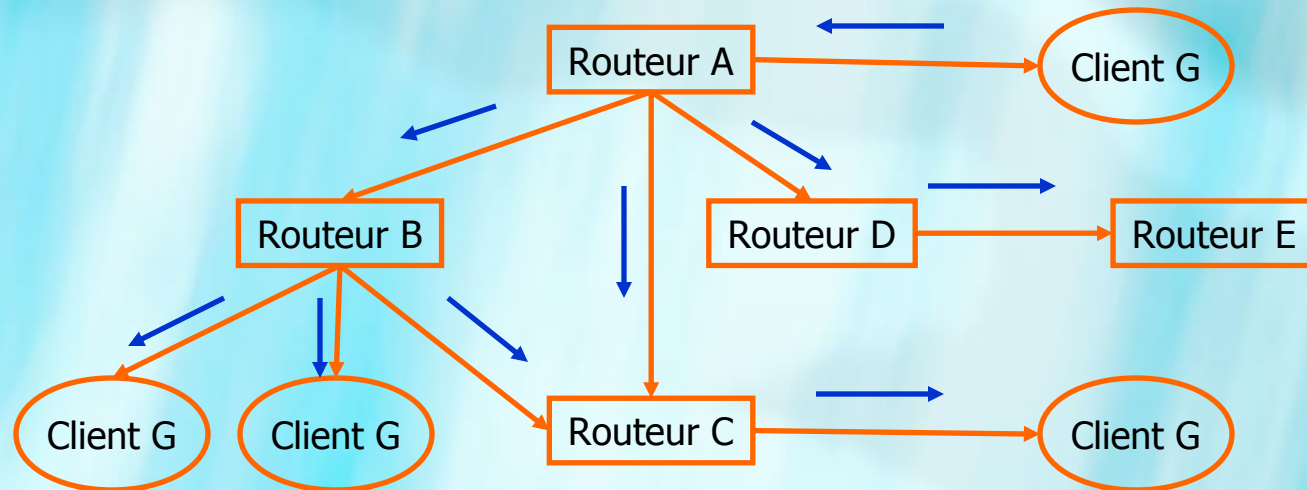
```
char mylogin[64]; strcpy(mylogin, argv[1]);
for (;;) {
    cout << "Message : ";
    char message [256];
    cin.getline (message, sizeof(message));
    if (message[0] == '\0') throw exception ("end of application");
    ostringstream tmp;
    tmp << "From : " << mylogin << endl;
    tmp << "To : everybody" << endl;
    tmp << "Text : ";
    tmp << message << endl;
    string & text = tmp.str();
    error = sendto (sockm, text.data(), text.size(), 0, (struct sockaddr *) &stm, sizeof(stm));
    if ( error == SOCKET_ERROR) throw exception ("sendto fail");
}
```

- Le routage de groupe de diffusion multiple normaux à portée internationale est un énorme problème
 - Le message n'a pas de destination connue
 - Donc il doit aller partout, sur tout les réseaux
 - Un message unique inonde le monde entier
 - Un flux pourrait saturer les routeurs
 - Que faire avec des routeurs rebouclés
 - Le routeur C va recevoir 2 fois le même message
 - Le routeur A reçoit son message



- Il faut mettre en place une politique de limitation
 - Deux politiques sont actuellement en place selon la densité d'abonnés
 - Politique dense :
 - inondation et élagage
 - DVMRP, PIM-DM, MOSPF
 - Politique clairsemé :
 - Greffe et élagage
 - PIM-SM et CBT

- Politique d'inondation par le premier paquet
- Les routeurs ré-émettent les messages sur toutes les interfaces sauf celle qui a réceptionné le message

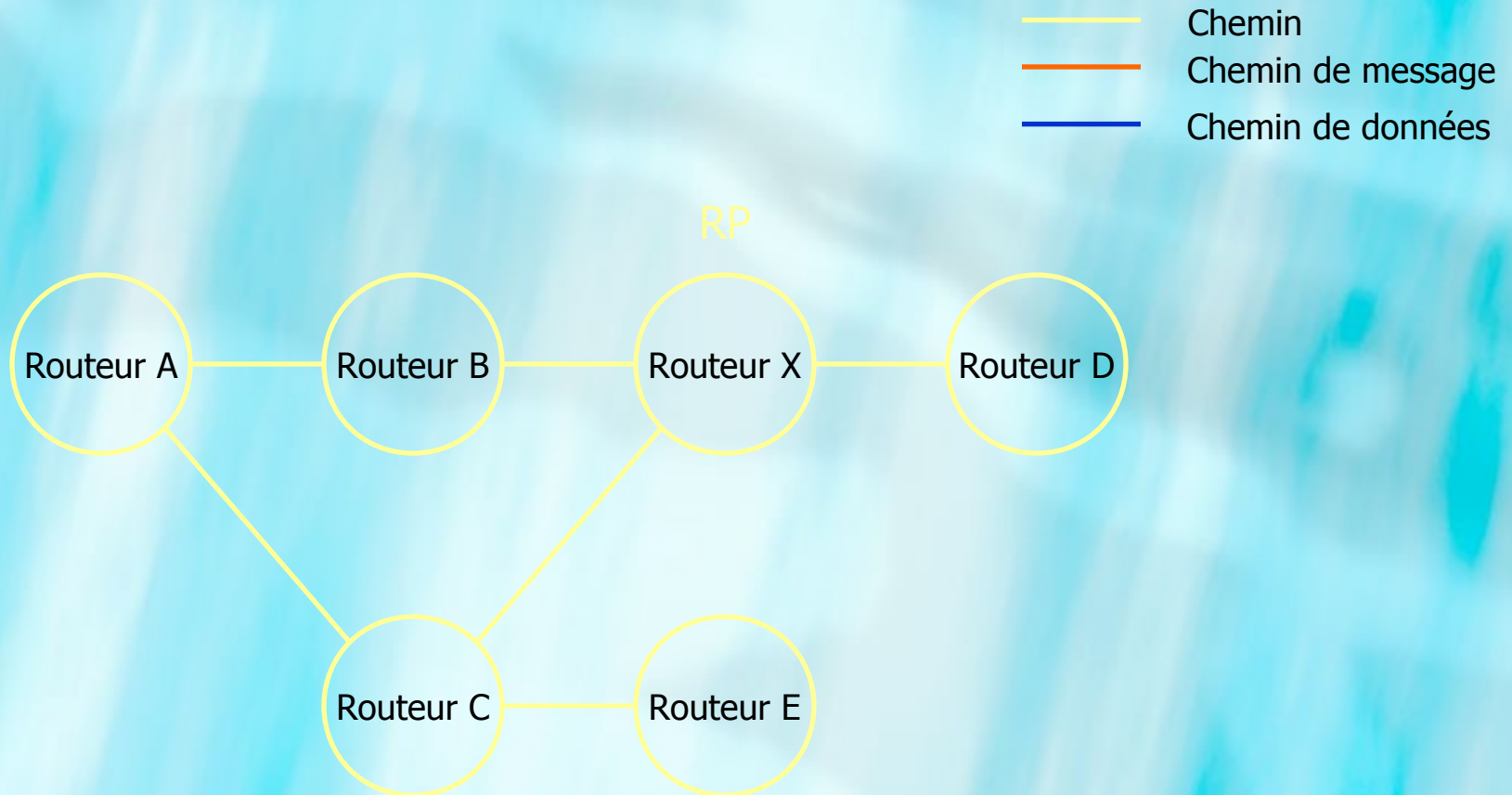


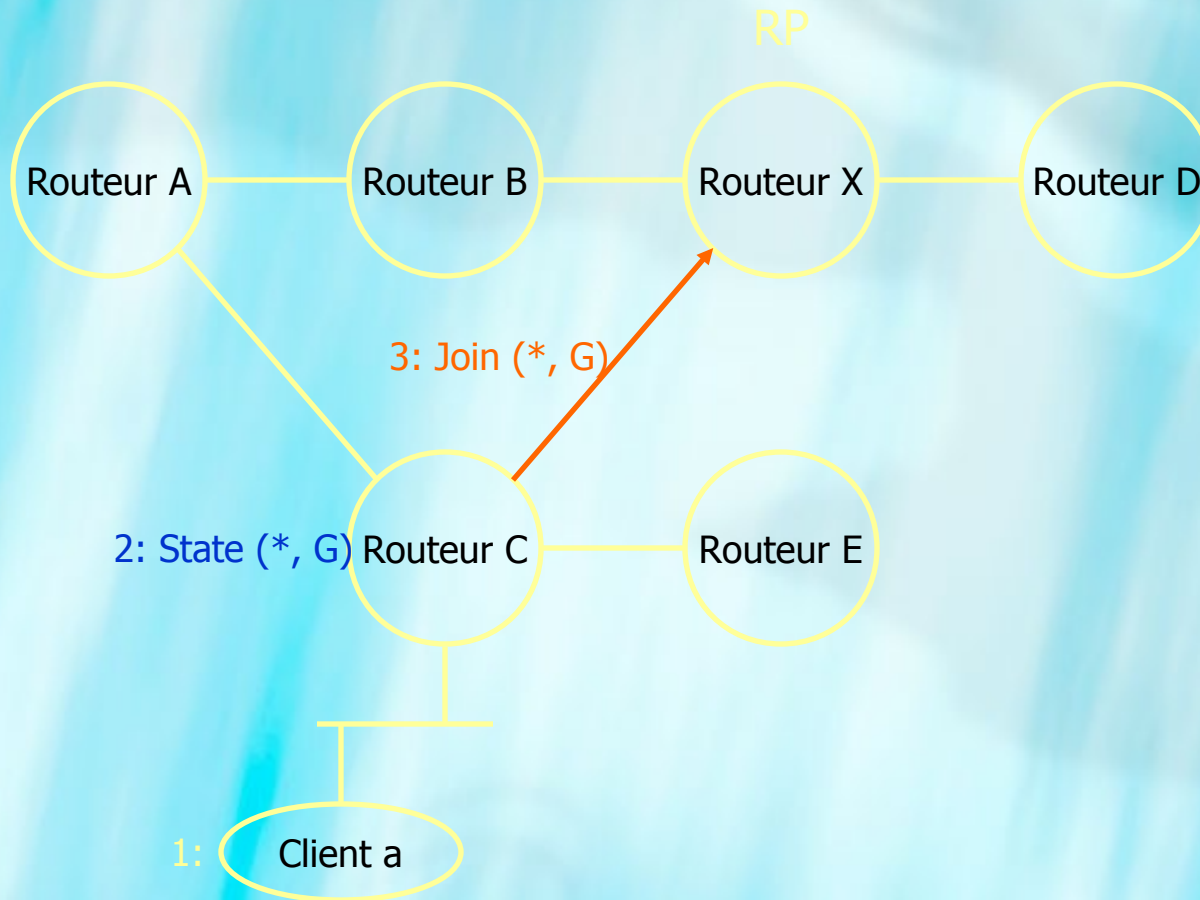
- Les routeurs D et E reçoivent un message pour rien
- Le routeur C reçoit deux messages, un pour rien

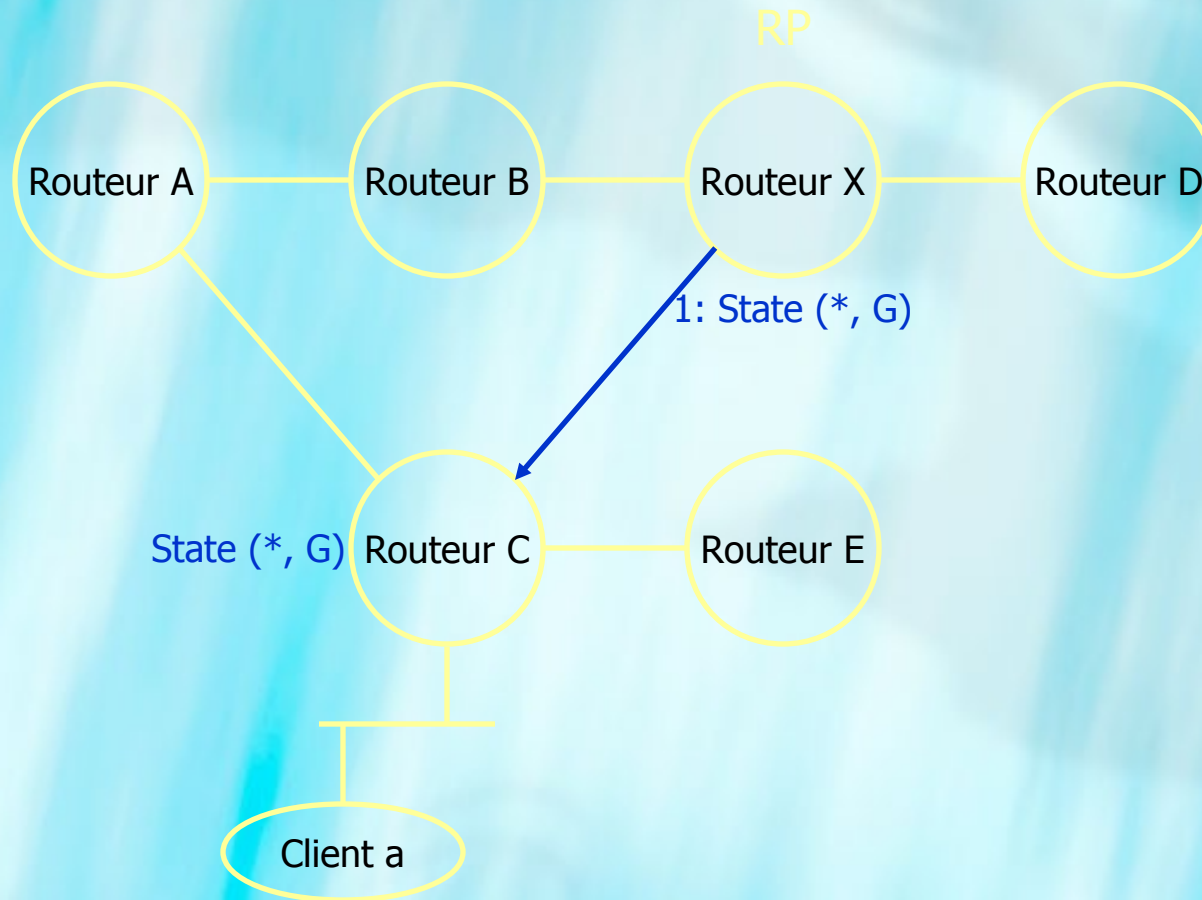
- Le routeur C jette le deuxième message car il sait l'avoir déjà traité
 - En fait la gestion peut être plus complexe car elle dépend du coût d'acheminement des messages.
- Le routeur E ne sait pas quoi faire avec ce message
 - Le message a pollué l'interface, les prochains seront pires
 - Le routeur E va envoyer un message élagage à l'interface sur le routeur D
 - Le routeur D neutralise la diffusion sur l'interface vers le routeur E
 - Comme il n'y a plus de groupe G sur le routeur D
 - Le routeur D va envoyer un message élagage à l'interface sur le routeur A
 - Le routeur A neutralise la diffusion sur l'interface vers le routeur D
- Les messages prochains ne seront transmis qu'aux routeurs essentiels
- L'élagage semble durer 2 heures avant de revenir dans un état stable

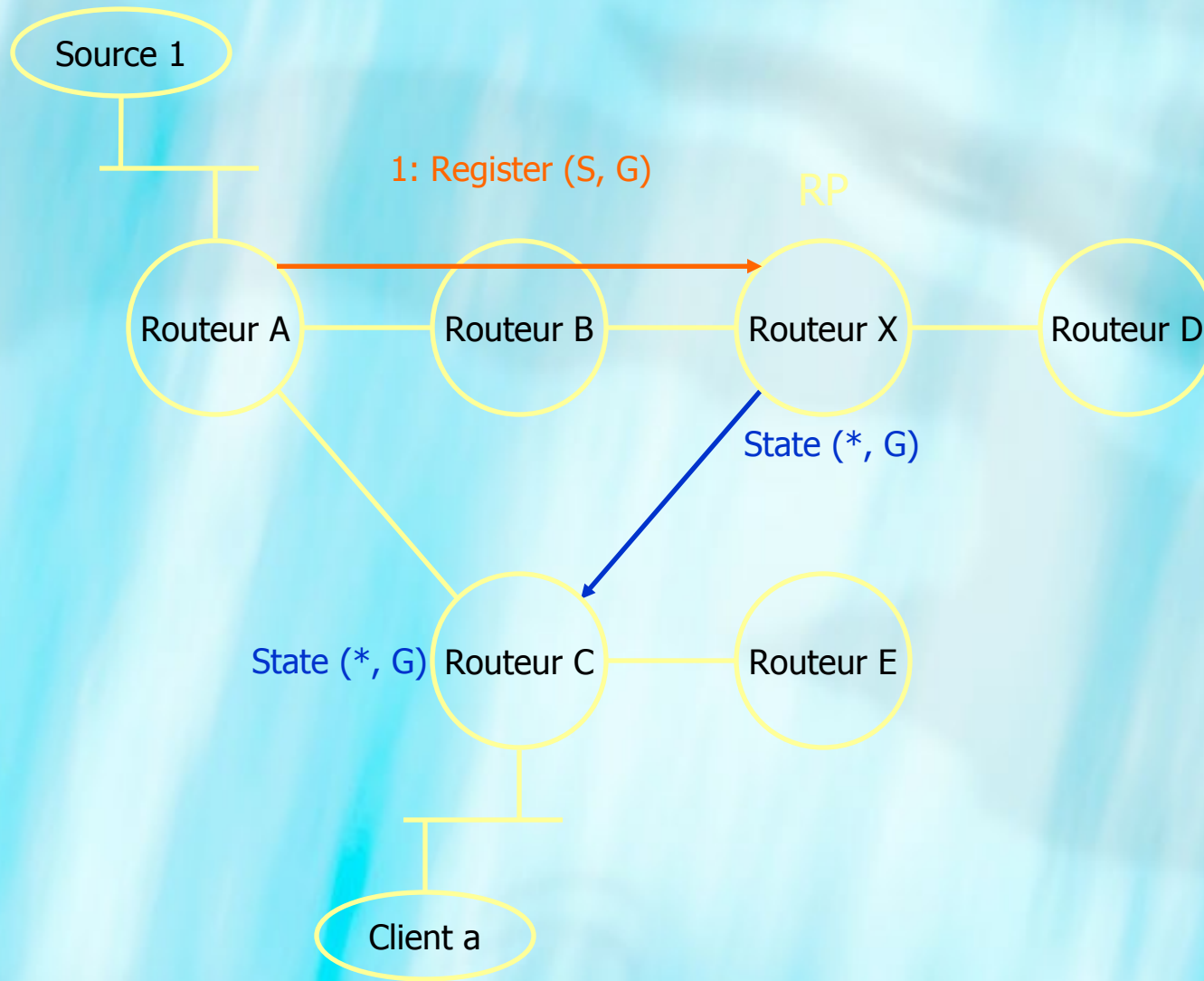
- Si un nouveau client du groupe G apparaît sur le routeur E
 - Par un message IGMP bien entendu
- Le routeur peut être dans trois états
 - L'élagage a été fait mais il y a peu de temps
 - Le routeur envoie un message greffe sur le routeur supérieur (D) pour re-solliciter des messages pour le groupe
 - Le routeur reconnecte l'interface
 - L'élagage est très ancien, mais d'autres interfaces tournent encore
 - Le routeur ajoute l'interface à la liste des interfaces concernées par le groupe
 - Le flux descendant est toujours disponible
 - L'élagage est très ancien et il n'y a plus d'activité
 - Le routeur inonde les interfaces et reprend le processus du début
- Si un client se déconnecte
 - Les routeurs élaguent récursivement au fur et mesure les interfaces qui ne sont plus concernées par le groupe
- Problème : DVMRP est coûteux en terme de routage internationale

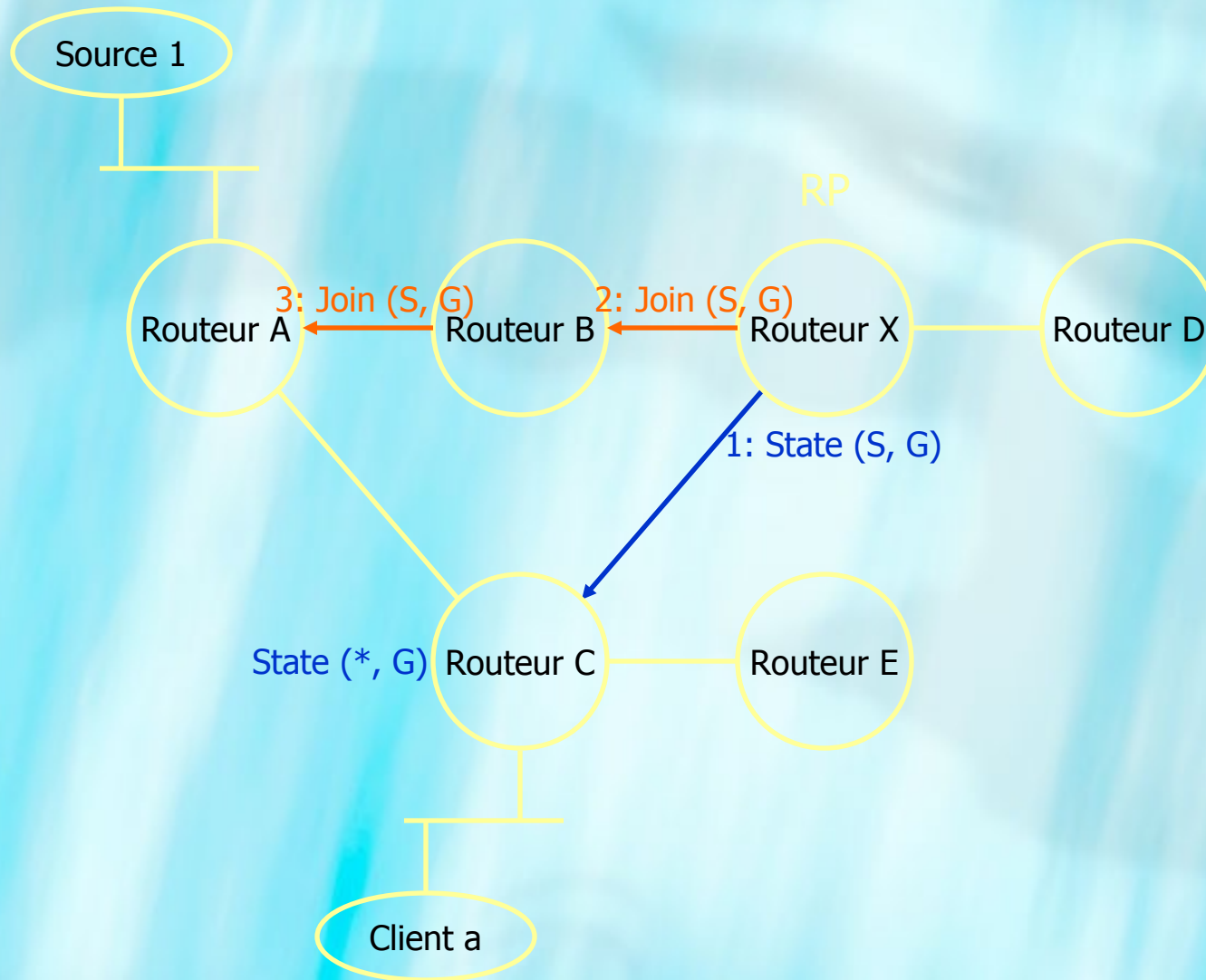
- Protocol Independent Multicast - RFC 2362
- Indépendant des tables de routage
- Utilise les routes unicast existantes
- Deux modes :
 - Dense Mode (PIM-DM), remplace DVMRP
 - Sparse Mode (PIM-SM)
- Philosophie
 - Le RP est la racine de la diffusion multiple
 - Les routeurs sources s'enregistrent auprès d'un point de rendez-vous (RP)
 - Les routeurs destinations s'enregistrent auprès d'un point de rendez-vous (RP)
 - Les communications se font entre routeurs sources et destinations
 - Il n'y a plus de multicast mais du multicast dans un tunnel unicast

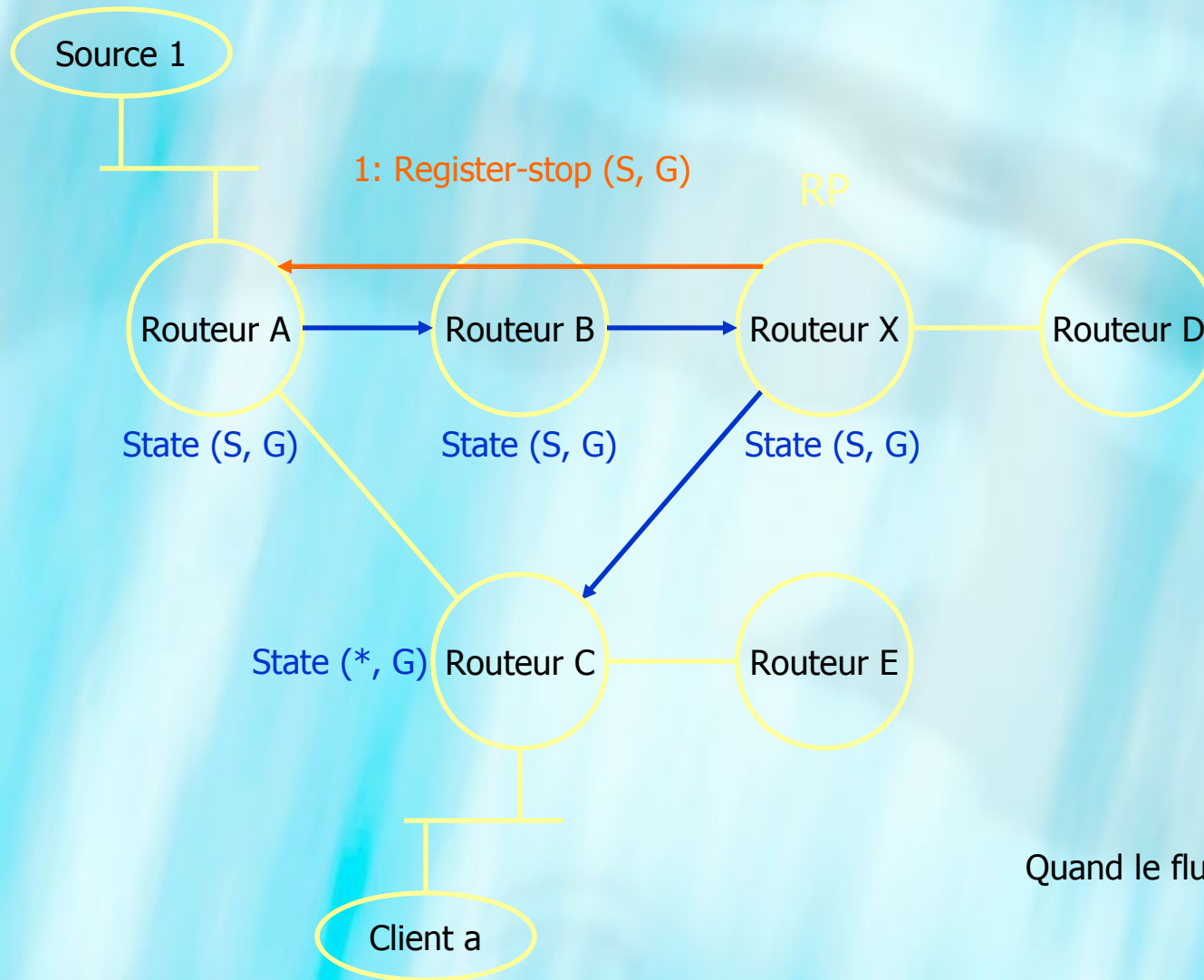












Quand le flux A-B-X est en place

