Réseau : introduction

Introduction

Réseau :

- Ensemble de noeuds
- Interconnecté (mais pas forcément tous connectés 2 à 2)
- Pour faire circuler des éléments ou des flux, même entre 2 noeuds non voisins

Ici : Réseaux informatiques, échange de données (séquence de bits ou d'octets, en paquets ou en flux)

Applications :

- Partage de ressources (données, imprimante, machine de calcul...)
- communication (mail, voix, visioconf...)

Exemples de réseaux :

- Réseau local (LAN : local area network)
- Réseau au étendu (WAN)
- "Internet"
- Téléphone cellulaire
- ...

Ce cours : principalement LAN et Internet

Internet

Désigne à la fois :

- Le réseau Internet (interconnexion de réseaux)
- Le nom d'un ensemble de protocoles ("TCP/IP")

Histoire rapide d'Internet

- Fin années 60 : ARPANET : réseau (militaire) censé être résistant aux attaques.
  - inter-universités (en contrat avec le Departement of Defense)
  - Commutation de paquets
  - Invention de TCP/IP pour la communication entre réseaux différents

Années 80 - 90 :

- Ouverture aux universités (CSNET, NSFNET), puis au privé

1989 : WWW (World Wide Web), hyperliens

Problématiques

- Acheminer les données
  - router les données
  - trouver et mettre à jour les routes
  - limiter les congestions
- Assurer que les données arrivent en l'état
  - vérifier s'il y a des erreurs
  - corriger les erreurs
  - transmettre les données dans l'ordre,
  - sans doublons

Objectifs :

- Vitesse de transmission (débit)
- Temps de latence : temps entre l'émission et la réception

Topologie

Réseau $\Leftrightarrow$ graphe connexe
Topologie (type de graphe) :

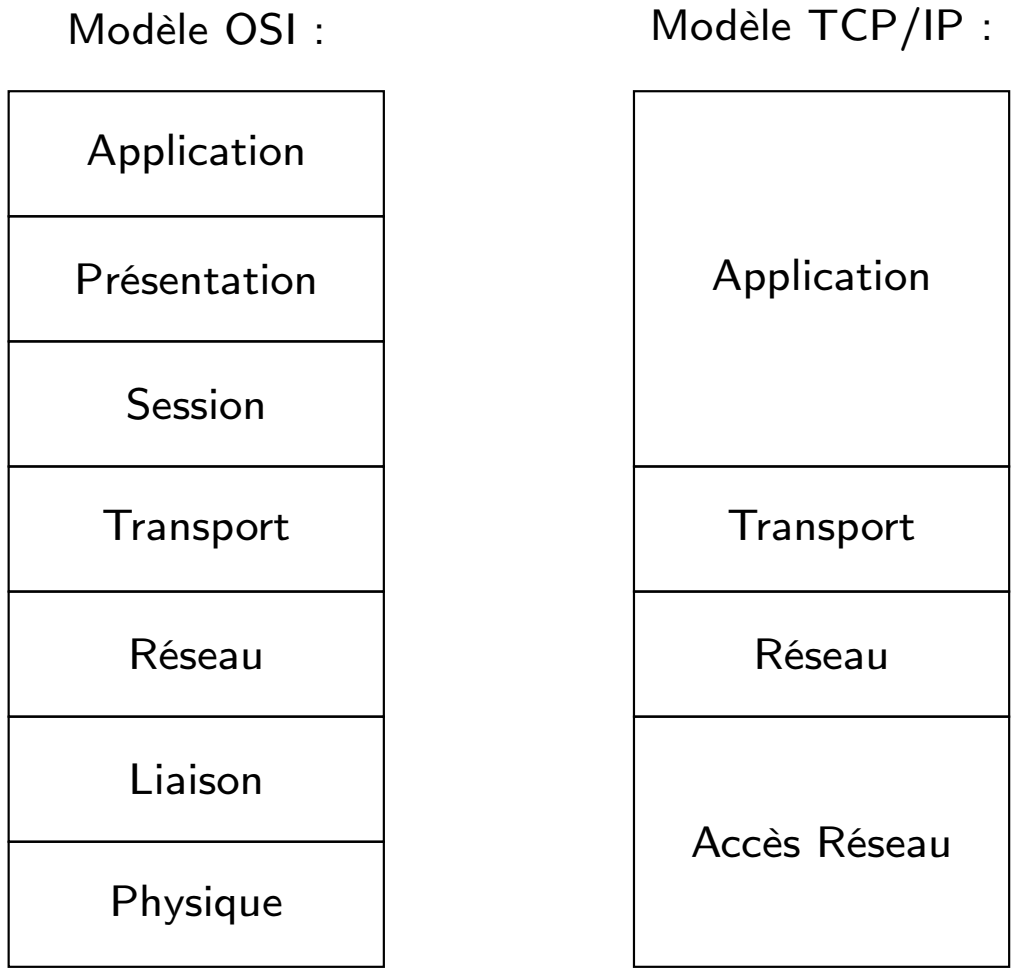
- étoile, arbre (Ethernet)
- cycle (anneau)
- graphe quelconque (Internet...)
- graphe complet
- grille, hypercubes (HPC)...

- homogène (LAN Ethernet)
- hétérogène (Internet)

Modèles d'organisation :

- Modèle Client / serveur (ex : HTTP)
- Modèle pair à pair (P2P)

Modèle(s) par couches



OSI = Open Systems Interconnection

TCP/IP = Transmission Control Protocol / Internet Protocol

Ce cours : principalement TCP/IP

Protocoles de communication

Protocole : Ensemble de règles (convention) pour que deux (ou +) entités puissent communiquer
Cela inclus :

- Format des données
- Signification des données (adresses, somme de contrôle, numéro dans une séquence...)
- "Algorithmes"

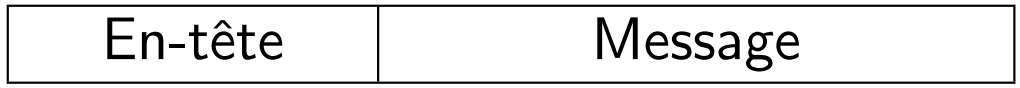
Chaque couche a son/ses protocole(s)

- Accès réseau : Ethernet...
- Réseau : IP (Internet Protocol)
- Transport : TCP, UDP
- Application : HTTP, DNS, IMAP, FTP...

Unités de communication

PDU ("Protocol Data Unit") : l'unité de base manipulé par le protocole

Unité typique :



L'en-tête dépend du protocole, et est spécifié par le protocole

Contient généralement un sous ensemble de :

- Type de "paquet"
- Taille du message
- Somme de contrôle
- Émetteur / Destinataire
- Port de destination
- Information sur le chiffrement
- Numéro de séquence...

Encapsulation

Chaque protocole d'une couche N communique via le/un protocole de la couche $N-1$.
Théoriquement, le protocole $N-1$ ne sait pas ce qu'il transporte

Par exemple le protocole TCP (couche transport) communique via le protocole IP (couche réseau)
Dans ce cas, le message TCP sera encapsulé dans le paquet IP :

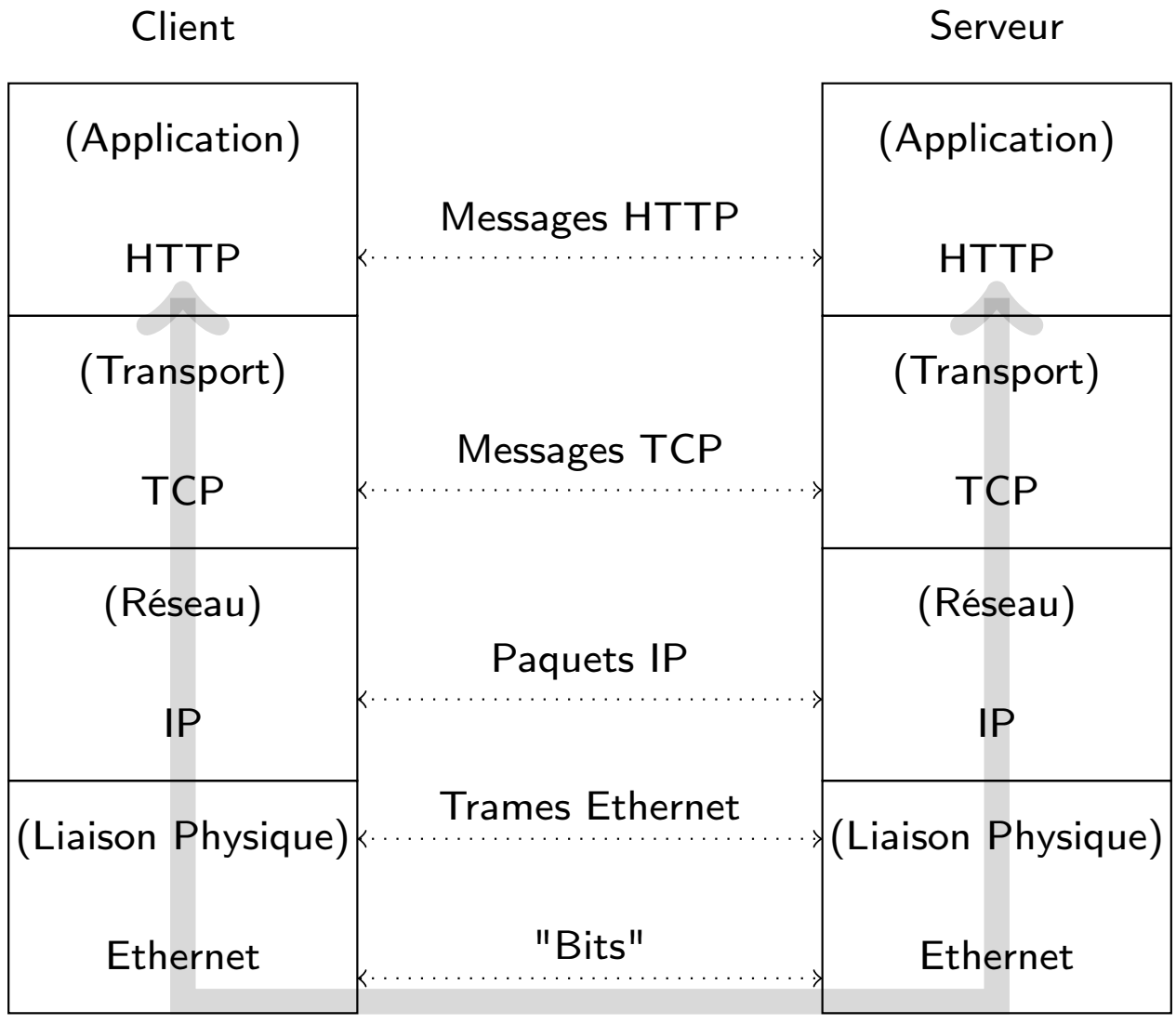
En-tête IP	En-tête TCP	Message TCP

Le protocole IP communique via un protocole Ethernet :

En-tête Ethernet	En-tête IP	En-tête TCP	Message TCP

Fragmentation possible :

- Si un message est trop grand pour être transporté dans une unité du protocole $N-1$, il peut être découpé (si cela est prévu) en plusieurs messages.


Client
Serveur
(Application)
HTTP
(Application)
HTTP
Messages HTTP
(Transport)
TCP
(Transport)
TCP
Messages TCP
(Réseau)
IP
(Réseau)
IP
Paquets IP
(Liaison Physique)
Ethernet
(Liaison Physique)
Ethernet
Trames Ethernet
"Bits"

Couche "Physique" et "Liaison" ("accès réseau")

Couche "Physique" :
Le médium de transport : câble en cuivre, fibre optique, ondes...
PDU : bits

- Comment sont encodés les bits ?
- Perturbations possibles

Couche "Liaison" :

- S'occupe des liaisons point à point.
- Éventuellement, transmet une somme de contrôle pour vérifier l'intégrité (Ethernet : CRC)

PDU : "Trames"
Protocoles : Ethernet, Wifi...

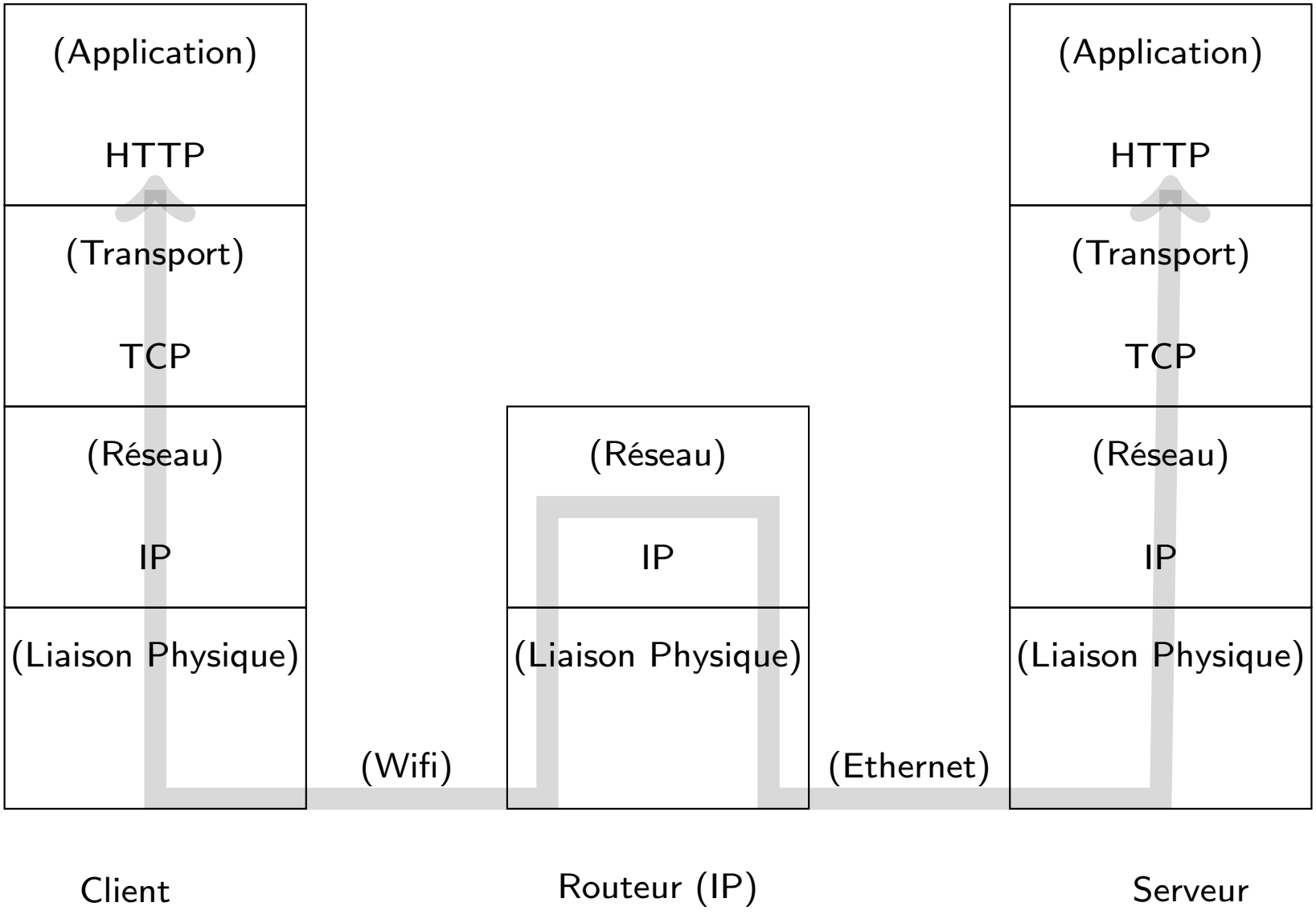
Couche "Réseau"

S'occupe de trouver les routes dans le réseau, de les mettre à jour, et router les paquets

Sur Internet : IP (Internet Protocol).
PDU : "Paquets IP"

Deux "variantes" d'IP :

- IPv4 : la plus utilisée actuellement, mais un nombre limité d'adresse possible 2^{32}. Arrive à saturation...
  - exemple d'adresse IPv4 : `192.168.1.79`
- IPv6 : la nouvelle norme. Partiellement en place. 2^{128} adresses
  - adresse IPv6 : `fe80::e6f8:9cff:fe67:ee22`


(Application)
HTTP
(Transport)
TCP
(Réseau)
IP
(Liaison Physique)
(Réseau)
IP
(Liaison Physique)
(Application)
HTTP
(Transport)
TCP
(Réseau)
IP
(Liaison Physique)
(Wifi)
(Ethernet)
Client
Routeur (IP)
Serveur

Couche Transport

Établir et maintenir les connexions, assurer l'arrivée, dans l'ordre des paquets...

Sur Internet : Principalement TCP et UDP.

- TCP : "Transmission Control Protocol"
  - Mode connecté
  - Fiable : détecte les erreurs, les données perdues, assure l'ordre

(Socket TCP : comme les sockets à la fin du cours sur les tubes !)

- UDP : "User Datagram Protocol"
  - Non connecté (plus léger : pas de confirmation de réception)
  - Mais : ne garantie pas la bonne livraison, ni l'ordre

Ces deux protocoles ajoutent un numéro de port : entre 1 et 65535
Généralement un numéro de port $\leftrightarrow$ une application.
HTTP : 80, SSH : 22, DNS : 53...

PDU : Segment TCP, Datagramme UDP...

Couche Application

- Web : HTTP, FTP
- mail : IMAP, POP, SMTP
- session : (Telnet), SSH
- DNS, DHCP
- ...

- Vos programmes !

Suite :

- Les couches en détail
- Programmation IP en POSIX

Couches Accès Réseau

Couche "Physique" : Médium

Support de transmission "guidés" :

- paire de cuivre torsadée
  - Ethernet (actuel) : 4 paires (câble de catégorie 5 "RJ45"...)
  - RTC/xDSL : 1 paire torsadée
- câble coaxial
- fibre optique
- ...

Sans fil :

- Ondes radio (wifi, bluetooth, satellites...)
- Ondes lumineuses

Couche "Physique" : Médium

Sujet à des perturbations (erreurs)

- atténuation (résistance, impuretés), auto-perturbations
- perturbations extérieures

Plus que c'est long/loin, plus qu'il y a de possibilités d'erreur

- débit théorique maximum diminue avec la longueur de la liaison

Pourquoi torsadées ?

- les perturbations électromagnétiques s'annulent

Éventuellement : un blindage en plus.

Couche "Physique" : Encodage des bits

"Modulation numérique" : convertir des bits en signaux analogiques

- Transmission en bande de base
- Modulation d'un signal porteur
  - modulation de fréquence
  - modulation d'amplitude
  - modulation de phase

Transmission en bande de base

Le plus simple (NRZ) :
Paire torsadée :

- Bit = 0 $\rightarrow$ tension positive
- Bit = 1 $\rightarrow$ tension négative

Fibre optique

- Bit = 0 $\rightarrow$ pas de lumière
- Bit = 1 $\rightarrow$ lumière

Problème : si trop de 0 de suite, on s'y perd.

- Codage de Manchester 0 = -+ et 1 =+- (Ethernet "classique")
- Interdire les suites trop longues de 0 ou 1 (réencoder...)

Modulation d'un signal porteur

Pourquoi :

- Multiplexage de fréquences
- Ondes électromagnétiques
- Médium fonctionnant sur une plage de fréquences

Comment :

- Modulation de fréquence (0 : f, 1 : f')
- Modulation d'amplitude
- Modulation de phase

- Possible de coder plus qu'un bit à la fois
- Possible d'associer modulation d'amplitude et de phase

Half / Full Duplex

- (Full-)Duplex : communication dans les deux sens possible en même temps
- Half-Duplex : communication dans les deux sens, mais une à la fois
- Simplex : communication dans un sens

Câble catégorie 5 (Ethernet "RJ45")

- 4 Paires torsadés
- Sert pour Ethernet
- Attention : Ethernet = une famille de normes (10BASE2, 10BASE-T, 100BASE-T, 1000BASE-T...)
  (Ethernet existe aussi sur coaxial et fibre optique).
- L'utilisation des paires diffère selon la norme.
- Améliorations : Cat 5e, Cat 6...
  - spécification plus strictes (résistance, capacité, inductance...)
- RJ45 : nom du connecteur (8P8C)

Couche Liaison

But :

- Interface à la couche réseau
- Contrôler et traiter les erreurs de transmission
- Réguler les flux

Deux types de liaisons :

- point à point : communication entre 2 machines
- diffusion : canal partagé entre plusieurs machines

Couche Liaison

Problèmes :

Comment découper le flux de bits en trames ?
$\rightarrow$ fanions de signalisation de début de trame

Détecter les erreurs

- dues au perturbations extérieures
- dues aux collisions de paquets

$\rightarrow$ utilisation de sommes de contrôle (CRC : Cyclic Redundancy Check)

Contrôle de flux $\rightarrow$ retour d'information

Liaison en mode diffusion

Et quand le médium est partagé ? (Wifi, câble commun à plus de 2 machines)

Au début d'Ethernet : un câble coaxial pour plusieurs machines.
Problèmes :

- Plus de collisions à gérer
- Destinataire de la trame

$\Rightarrow$ Sous-couche MAC (Medium Acces Control)

MAC Ethernet classique

Paquet Ethernet :

Préambule	MAC dest.	MAC source	Type/Longueur	Données	CRC
8	6	6	2	<=1500	4

Adresse MAC :

- 6 octets : 3 premiers pour le constructeur, les 3 derniers pour les cartes construites par le constructeur.
- Théoriquement, chaque carte réseau a une adresse MAC différente.

CRC : somme de contrôle

Trame MAC Wifi (802.11)

Contrôle	Durée	Dest.	Source	Adresse 3	Séquence	Données	CRC
2	2	6	6	6	2	<=2312	4

Ethernet commuté

Ethernet sur un câble coaxial :

- Difficile à rajouter une nouvelle machine
- Une carte réseau défaillante peut bloquer tout le réseau
- Vite encombré.

Évolutions d'Ethernet :

- Topologie en étoile : toutes les machines sont reliées à un *hub*, par une paire torsadée

Puis, pour augmenter la capacité : Inutile d'envoyer les paquets aux machines non destinatrices de la trame !

- Remplacement du *hub* par un *switch* (commutateur) : Ethernet commuté

Ethernet commuté

Le switch doit garder une table adresse MAC / port.

Comment les trouver la bonne bijection ?

Apprentissage a posteriori :

Quand il reçoit une trame de source s, destinataire d par le port p :

- Il associe l'adresse MAC s au port p
- Si le port de l'adresse d n'est pas p, il diffuse la trame sur le port de d
- Si le port de l'adresse d est p, il rejette la trame
- Si il ne sait pas le port de l'adresse d, il diffuse à tout le monde

Marche aussi pour une topologie en arbre !

Ethernet commuté

Les switch jouent un rôle similaire aux routeurs IP.

Pourquoi rajouter une couche "Réseau" ?

Suite :

- Couche Réseau et Transport : TCP/IP

Couche réseau & Protocole IP

Introduction

La couche "liaison" s'occupe des communications point à point.

La couche "réseau" s'occupe de :

- router les informations dans le réseau
- trouver et mettre à jour les routes
- détecter nouveaux liens
- détecter les pertes de liens et problèmes de congestion

Sur internet : protocoles IP

- IPv4 : le plus courant, mais on arrive à bout des adresses disponibles
- IPv6 : le nouveau, partiellement en place

Commutation de paquets vs de circuits

Il existe deux grand types de réseaux :

- Réseaux à commutation de paquets (store-and-forward)
  - Unité de base, un "paquet" : une suite d'octets
  - Le routeur reçoit un paquet, analyse à qui il est destiné, et le renvoie dans la bonne direction
- Réseaux à commutation de circuit
  - Chaque routeur prépare la route en connectant le port d'entrée au port de sortie. Puis l'information passe en flux jusqu'à la fin de la communication
  - Exemple : réseau téléphonique traditionnel

Il est aussi possible d'avoir des réseaux à paquets, où les routes sont prédéfinies à l'avance pour chaque connexion.

Internet : commutation de paquets.
Unité de base du protocole : "paquet IP"

Routage de paquets

Chaque noeud interne du réseau (noeud avec au moins 2 voisins) doit choisir, quand il reçoit un paquet, à qui il doit le transférer. ("Router un paquet").

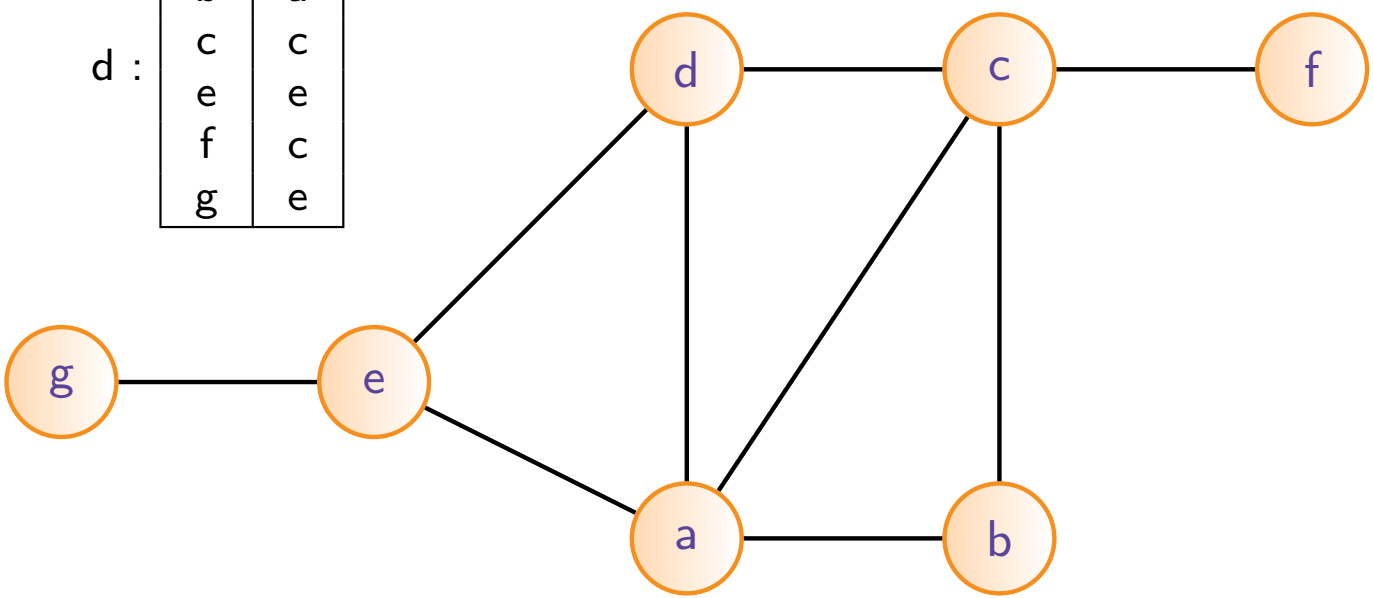
Un noeud interne est souvent appelé un routeur (ordinateur ou matériel spécialisé)

Il possède pour cela une table de routage :

- une table de paires (adresse, voisin)

d :

a	a
b	a
c	c
e	e
f	c
g	e



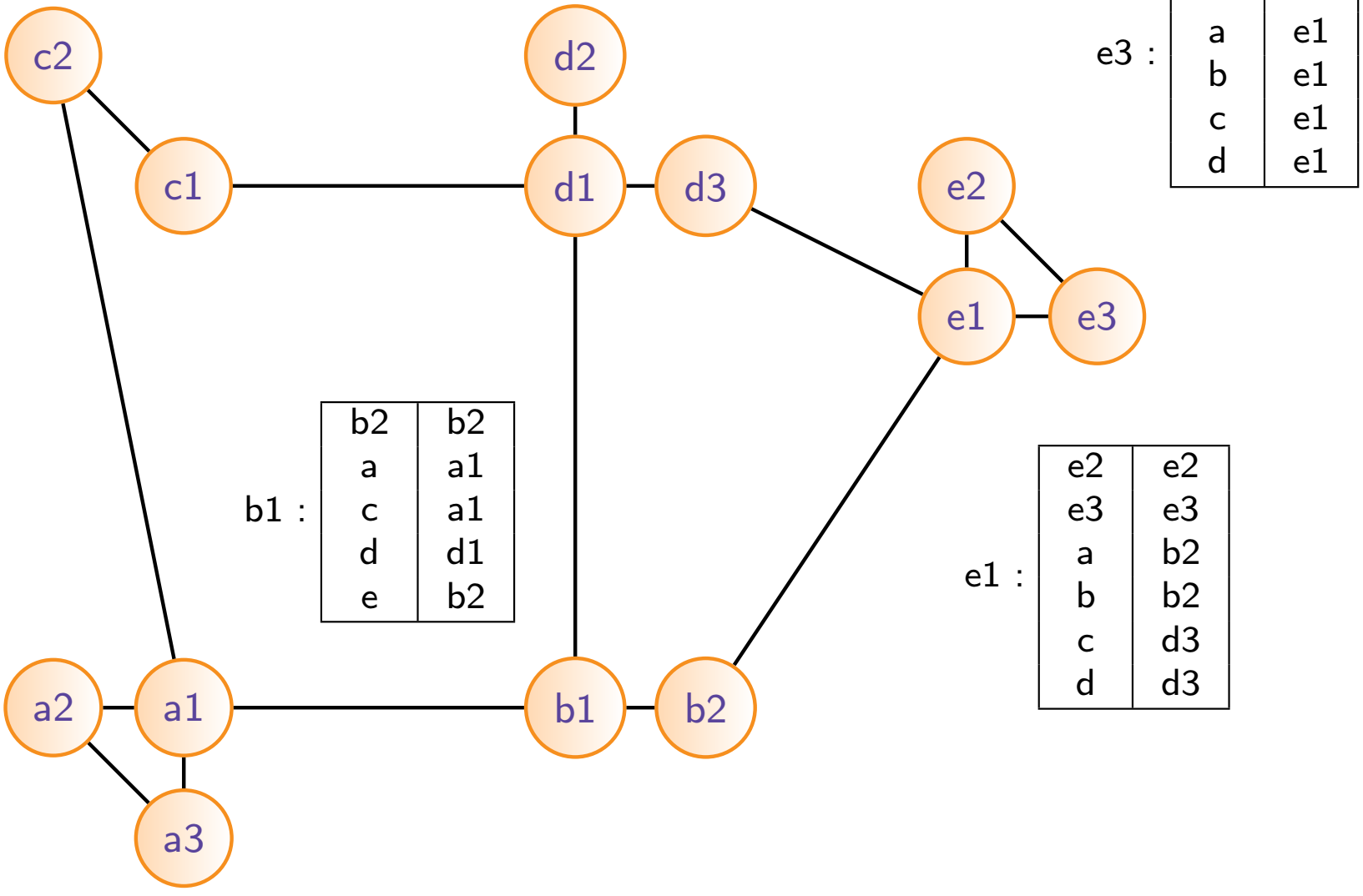
Routage hiérarchique

Problème : la table a autant d'entrées que de noeuds dans le graphe.
Internet IPv4 : $\sim 2^{32} = \sim 4$ milliard...

Solution : Routage hiérarchique
Chaque le réseau est divisé en sous réseaux :

- les adresses sont hiérarchiques, du type "sous-reseau.machine"
- chaque sous réseau S sait comment router ses paquets
- en dehors du sous-réseau S, tous les paquets vers une machine de S est routé vers le même routeur, un point d'entrée de S.

Routage hiérarchique



e3 :

e1	e1
e2	e2
a	e1
b	e1
c	e1
d	e1

b1 :

b2	b2
a	a1
c	a1
d	d1
e	b2

e1 :

e2	e2
e3	e3
a	b2
b	b2
c	d3
d	d3

Trouver les tables

Les tables de routage peuvent être définies statiquement (routage statique), ou dynamiquement

- Statique : L'administrateur de la machine/routeur définit les entrées de la table.
  Cas habituel pour vos machines.
  Problème : ingérable pour les routeurs internes
- Dynamique : Le routeur construit lui même sa table de routage, en partageant des informations avec ses voisins

Comment un routeur peut faire pour remplir/compléter/mettre à jour sa table de routage ?

Routage par inondation

Découverte de routes par inondation :
Si on ne connaît pas la direction pour un noeud, on peut faire une diffusion générale ("broadcast") d'un paquet demandant où se trouve le noeud
Chaque routeur qui reçoit le paquet de demande : soit

- répond si il sait, ou
- renvoie le paquet à tous ses autres voisins

Problème : lourd

Pour éviter de trop encombrer le réseau :

- Chaque requête possède un numéro. On garde mémoire des requêtes déjà renvoyées, pour ne pas le faire une deuxième fois
- Pour chaque demande non satisfaite, on attend un petit moment avant de la renvoyer. Si on reçoit la même demande entre temps d'un autre voisin, on ne la lui retransmettra pas.

Plus court chemin

Si un routeur connaît la topologie du réseau, il peut effectuer un algorithme du plus court chemin (comme l'algorithme de Dijkstra)

$\Rightarrow$ plus court chemin dans un graphe

Avantage : poids sur les liens (distance, prix...)

Problème : il faut connaître toute la topologie du réseau

Vecteur de distance (Bellman-Ford)

- Chaque routeur connaît la distance vers ses voisins. (d_i)
- Distance : nombre de sauts, délai de propagation...
- Chaque routeur maintient (en plus de sa table de routage) une liste de distance estimée à tous les noeuds du réseau ("vecteur")
- Chaque routeur envoie régulièrement à tous ses voisins ce vecteur
- Le routeur met à jour sa table de routage : pour chaque noeud x dont il a connaissance, il route le paquet vers le voisin i qui minimise $d_i + V_i[x]$. (Et met à jour son vecteur de distance en même temps)

Problème : si une route disparaît, l'information mettra du temps à être corrigée... (problème de la valeur infinie)

État de lien

Routage par informations d'état de lien :

- Chaque routeur construit un paquet avec l'ensemble de ses voisins, et la distance
- Le paquet est broadcasté sur tout le réseau (avec un numéro de séquence)
- Chaque routeur connaît donc l'ensemble des noeuds et leurs voisins, et peut reconstruire le graphe
- Les routes sont décidées grâce à un algorithme comme Dijkstra

Contrôle de la congestion

En cas de congestion, il peut y avoir l'effondrement du débit du réseau :

- les émetteurs retransmettent les paquets perdus (ou trop retardés), qui seront à nouveau perdus...

Lors d'une congestion, que faire ?

- augmenter la capacité du lien
- rerouter sur une route moins encombrée
- avertir les sources
- contrôle d'admission
- éliminer des paquets...

Qualité de service (QoS)

Toutes les applications utilisant le réseau n'ont pas les mêmes besoins. Par exemple :

- Visio-conf : demande haute en délai, faible en fiabilité, haute en bande passante
- Mail : demande faible en délai, haute en fiabilité, faible en bande passante

Les algorithmes de routage peuvent considérer plusieurs classes de paquets
Par exemple : on préfère perdre des paquets plutôt que les faire attendre lors des congestions.

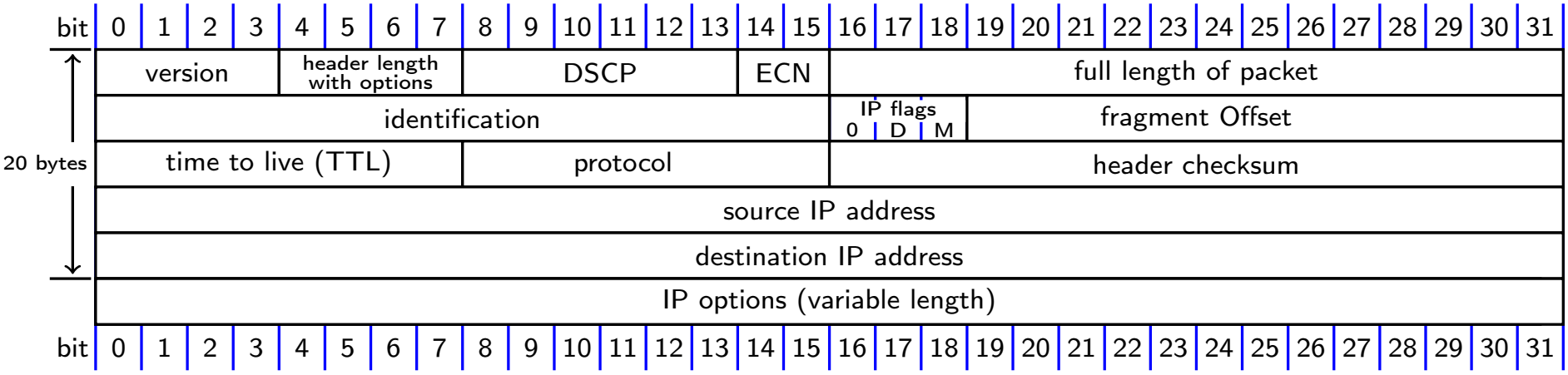
Internet Protocol

Protocole réseau utilisé sur Internet : IP (Internet Protocol). Deux versions :

- IPv4 : le plus utilisé actuellement, mais un nombre d'adresses limité ($< 2^{32}$).
- IPv6 : la nouvelle norme, partiellement en place

Routage de paquets, store-and-forward

Entête IPv4



- Version = 4
- DSCP (Differentiated Services Code Point) : Classe du paquet (QoS)
- ECN (Explicit Congestion Notification)
- Identification / D / M / Framgent Offset : Quand un paquet est fragmenté, tous les fragments qu'un paquet contiennent la même identification. Fragment Offset continent la position du fragment. (D : Don't fragment. M : More fragment)
- TTL : décrémenté à chaque saut (retransmission par un routeur). Quand il atteint 0, le paquet est éliminé (et un message ICMP est envoyé à la source)
- Protocol : TCP=6, UDP=17, ICMP=1...
- Options : Routage strict, enregistrement de la route...

Attention : Big endian !

Adresses IPv4

Une adresse IPv4 = 32 bits

- c-à-d 4 octets, ou
- 4 entiers de 0 à 255.

Notation a.b.c.d. Ex : `192.168.1.1`

ICANN (Internet Corporation for Assigned Names and Numbers) fournit les adresses

Organisées hiérarchiquement en sous-réseaux

Sous réseaux IPv4

Un sous réseau possède une adresse IP i, et un masque m de sous-réseau

Toutes les adresses IP j telles que $j \& m = i$ font parties du sous-réseau.

Masque (ou netmask) : en binaire : suite de k '1', puis de $32-k$ '0'
Notation : ip/k :
Exemple : `192.168.1.0/24` signifie que :

- le masque est `255.255.255.0`
- le sous réseau va de `192.168.1.0` à `192.168.1.255`

Sous réseaux IPv4 : Exemple

- Machine (hôte) d'adresse IP : `147.222.23.42`
- Sur le réseau : `147.222.16.0/20`
- $\Rightarrow$ Masque réseau : 255.255.240.0

Adresse hôte	147								222								23								42							
Adresse hôte	1	0	0	1	0	0	1	1	1	1	0	1	1	1	1	0	0	0	0	1	0	1	1	1	0	0	1	0	1	0	1	0
Masque	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0
Adresse réseau	1	0	0	1	0	0	1	1	1	1	0	1	1	1	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
Adresse réseau	147								222								16								0							
	Champ sous-réseau																				Champ hôte											

Routage simple

Pour voir/configurer les interfaces réseaux et les adresses IP/masque : `ifconfig`.

Routage statique via une passerelle (cas simple de routage)

- Tous les paquets pour les adresses IP dans le réseau sont envoyé directement au destinataire via Ethernet (ou Wifi, ou la couche liaison du réseau)
- Tous les autres paquets sont envoyés à la passerelle (l'adresse IP de l'interface du routeur qui est connecté au reste d'Internet)

Pour voir/configurer les routes sur Linux : `route`

```
Table de routage IP du noyau
Destination   Passerelle   Genmask          ...  Iface
0.0.0.0.      140.77.12.1  0.0.0.0               eth0
140.77.12.0   0.0.0.0      255.255.254.0         eth0
169.254.0.0   0.0.0.0      255.255.0.0           eth0
```

(Règle du "plus grand préfixe commun")

Adresses réservées

Certaines plages d'adresses sont réservées :

- 127.0.0.0/8 : Bouclage interne
- 255.255.255.255/32 : Broadcast local
- 10.0.0.0/8, 172.16.0.0/12, 192.168.0.0/16 : Adresses privées (NAT)
- 169.254.0.0/16 : lien local
- 224.0.0.0/4 : multicast
- 240.0.0.0/4 : réservé pour une utilisation future...
- ...

NAT (Network Address Translation)

Pour éviter de gaspiller les adresses et d'avoir à demander à l'ICANN des adresses pour les machines personnelles ou qui ne nécessitent pas une adresse IP visible de l'extérieur (pas de serveur)

- Assigner à un sous-réseau local une seule adresse IP (`addr`) pour internet

Principe de NAT :

- Les machines à l'intérieur du réseau local ont une adresse IP en `192.168.0.0/16`, `10.0.0.0/8` ou `172.16.0.0/12`.
- Quand elles veulent envoyer un paquet à Internet, elles passent par une passerelle, qui est connectée à Internet via l'adresse `addr`
- La passerelle remplace dans le paquet IP l'adresse du réseau local en `addr` avant d'envoyer le paquet sur Internet
- Quand la passerelle reçoit un paquet depuis internet, elle analyse les entêtes des paquets TCP et UDP, et regarde le port utilisé pour retrouver l'adresse du destinataire dans le réseau.

IPv6

Problème d'IPv4 :

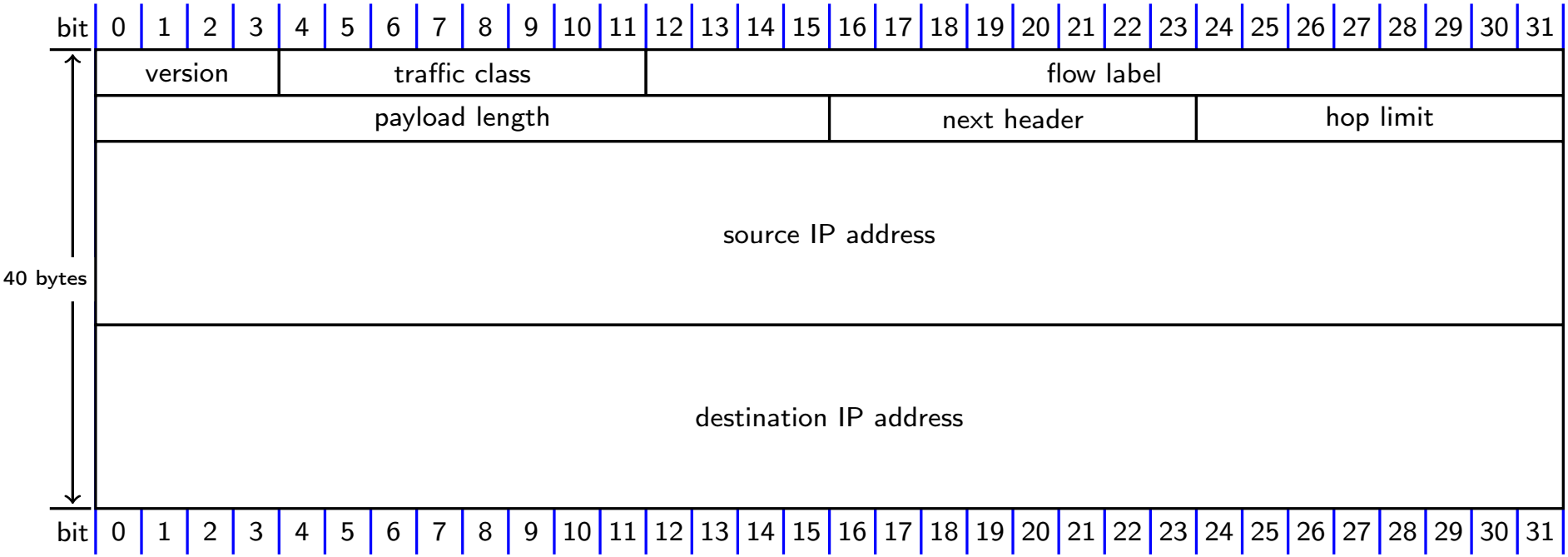
- Seulement 2^{32} adresses
- Tables de routage trop longues, dû à la fragmentation en sous-réseaux trop petits

Solution : IPv6

Changements entre IPv4 et IPv6 :

- Adresses sur 128 bits
- Simplification de l'en-tête
- Suppression de la somme de contrôle
- Suppression de la fragmentation des paquets en cours de route
- IPsec (Internet Protocol Security)
- ...

Entête IPv6



- version = 6
- traffic class = DSCP/ECN de IPv4
- flow label : identification du flux (réservation / QoS)
- hop limit : le nouveau nom du "time to live"
- next header : soit le contenu du prochain entête facultatif (option), soit le protocole
- options : informations sur la fragmentation, routage, chiffrement...

Adresse IPv6

8 blocs de 16 octets.
Les blocs sont écrits en hexadécimal, séparés par ":"

- On peut omettre les 0 de début de blocs
- Un ou plusieurs blocs consécutifs à 0 peuvent être remplacés par "::"

Exemple :
`fe80:0000:0000:0000:028d:99ff:fec1:0078=`
`fe80::28d:99ff:fec1:78`

Protocoles de gestion

La couche IP contient d'autres protocoles (de gestion) :

- ICMP : messages de contrôle IPv4
- ICMPv6 : messages de contrôle IPv6
- ARP/RARP : résolution d'adresse
- DHCP : configuration dynamique
- IGMP
- ...

ICMP

ICMP = Internet Control Message Protocol
Paquets ICMP :

- destination inaccessible
- délai expiré
- demande/envoi d'écho (`ping`)
- problème d'en-tête
- ...

Commandes utilisant les messages ICMP : `ping`, `traceroute`

ARP

ARP = Address Resolution Protocol

Permet, dans un sous-réseau, de trouver la correspondance entre l'adresse IP d'une interface et son adresse MAC. Principe :

- Quand un noeud ne connaît pas l'adresse MAC associée à une adresse IP, il envoie un paquet "broadcast" (à tout le monde) demandant :
- "Je suis (adresse IP)/(adresse MAC). À qui appartient (adresse IP) ?"
- L'ordinateur possédant l'adresse IP répond.

- RARP (Reverse ARP) : demande l'adresse IP à partir de l'adresse MAC

Commande : `arp`

DHCP

DHCP = Dynamic Host Configuration Protocol

Un noeud sur un réseau peut demander, via ce protocole, une adresse IP et la configuration du réseau (Masque, adresse IP de la passerelle, serveurs DNS...)

- Il envoie un message broadcast
- Un serveur DHCP (normalement, un serveur pour tout le sous-réseau) se charge d'assigner les adresses, et de répondre.

Commandes : `dhcpd` / `dhclient`

Algorithmes de routage sur Internet

Internet est un réseau de réseaux. Il n'y a pas un unique algorithme de routage. Chaque sous-réseau peut utiliser le sien.
Plusieurs algorithmes existent.

Pour les "Système Autonome" (FAI, RENATER...)

- RIP (Routing Information Protocol) : vecteurs de distance
- IGRP (Interior Gateway Routing Protocol) : vecteurs de distance
- OSPF (Open Shortest Path First) : état de liens
- IS-IS (Intermediate system to intermediate system) : état de liens

Entre systèmes autonomes : des considérations économiques et politiques s'ajoutent. Par exemple : certains liens sont payants.
BGP (Border Gateway Protocol) : Vecteurs de chemins

Couche "Transport" : TCP et UDP

Introduction

La couche "réseau" s'occupe de router les paquets :

- Il n'y a aucune garantie qu'un paquet arrive...
- Même si des messages de contrôle sont prévu (ICMP), il n'est pas garanti qu'on reçoive une erreur si un paquet n'arrive pas

La couche "transport" :

- s'occupe de rajouter de la fiabilité
  - contrôle que tous les paquets arrivent
  - si un paquet n'arrive pas, elle le renvoie automatiquement
  - contrôle que les paquets arrivent dans l'ordre
  - sinon, elle les remet dans l'ordre avant de passer à la couche suivante ("Application")
- est l'interface qu'on va utiliser dans les applications.
  - c'est la dernière couche dans le "système"
  - c'est celle qu'on va appeler dans nos programmes

Protocoles : sur Internet, principalement TCP et UDP

Interface avec les applications

Une application (et vous) :

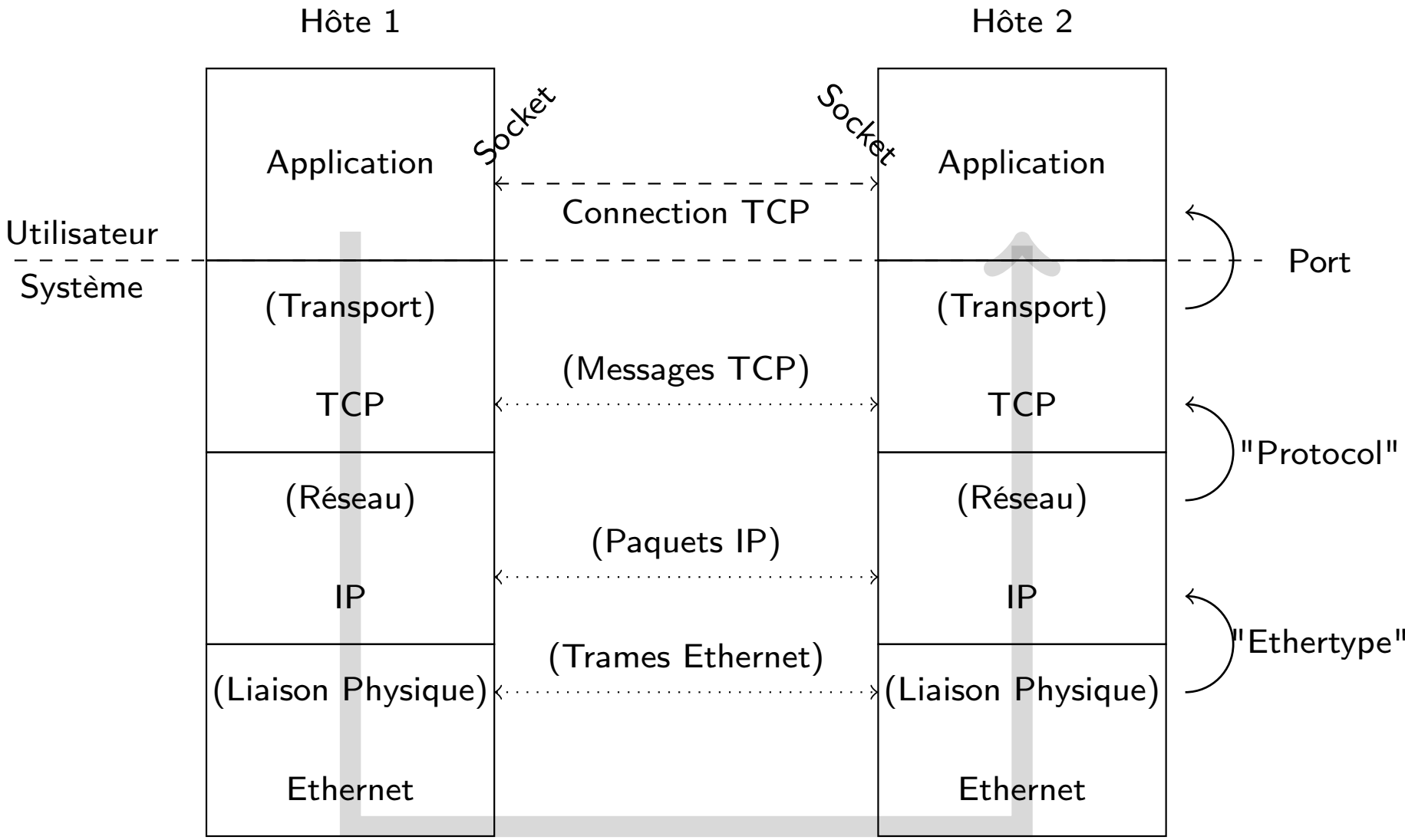
- veut avoir une interface standard pour accéder au réseau
- ne veut pas (en général) avoir à gérer les vérifications (contrôle de somme, qu'un paquet n'a pas été perdu...)

La couche transport s'occupe de cela.

Le système propose des sockets pour les connexions réseaux :

- identifiées par des descripteurs de fichiers
- elles fonctionnent comme des tubes
- bi-directionnelles

Côté réseau, les sockets sont identifiées par leur numéro de port


Hôte 1
Hôte 2
Application
Socket
Socket
Application
Connection TCP
Utilisateur
Système
Port
(Transport)
(Transport)
(Messages TCP)
TCP
TCP
"Protocol"
(Réseau)
(Réseau)
(Paquets IP)
IP
IP
"Ethertype"
(Trames Ethernet)
(Liaison Physique)
(Liaison Physique)
Ethernet
Ethernet

Ports

TCP et UDP rajoutent des numéros de port (un entier entre 1 et 65535).

Ces numéro servent à identifier les applications de la couche supérieure ("Application")

- L'application demande à la couche TCP/UDP d'ouvrir un port (via une socket)
- Quand un segment TCP arrive sur la machine (par la couche "réseau"), la couche TCP/UDP regarde le numéro de port
- Si c'est un numéro associé, le segment sera transmis à l'application correspondante (via la socket)
- Sinon, la couche renvoie un "message" d'erreur

Chaque type de service à un port normalement assigné. HTTP : 80, SSH : 22... (voir `/etc/services`)

Différence TCP/UDP

Sur Internet : TCP et UDP

UDP (User Datagram Protocol)

- non connecté
- somme de contrôle
- pas de garantie :
- le paquet peut ne pas arriver, ou arriver en double
- les paquets peuvent être intervertis
- (Proche des garanties de la couche réseau)

PDU : "Datagrames UDP"

Différence TCP/UDP

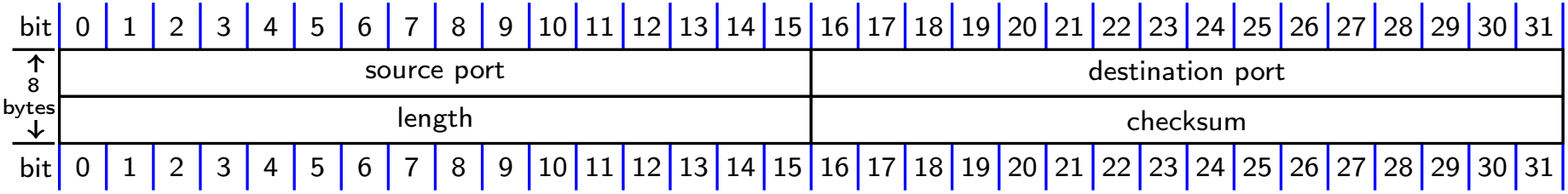
TCP (Transmission Control Protocol)

- connecté
- somme de contrôle
- la couche TCP s'occupe que les paquets arrivent, et arrivent dans l'ordre :
- si un paquet n'arrive pas, ou arrive erroné (mauvaise somme de contrôle), elle se charge elle même de renvoyer le paquet
- si des paquets sont intervertis, elle les remet dans le bon ordre avant de les délivrer à la couche supérieure

PDU : "Segments TCP"

UDP (User Datagram Protocol)

Entête :



Notes (idem pour TCP) :

- Il y a un port destination, mais également un port source (sert à retourner une réponse)
- la somme de contrôle se fait sur une "pseudo-entête IP" (adresse source, adresse destination, protocole et longueur), plus le datagramme UDP (avec le champ checksum à 0)

Rappel : Un datagramme perdu ou erroné ne sera pas automatiquement renvoyé (contrairement à TCP)

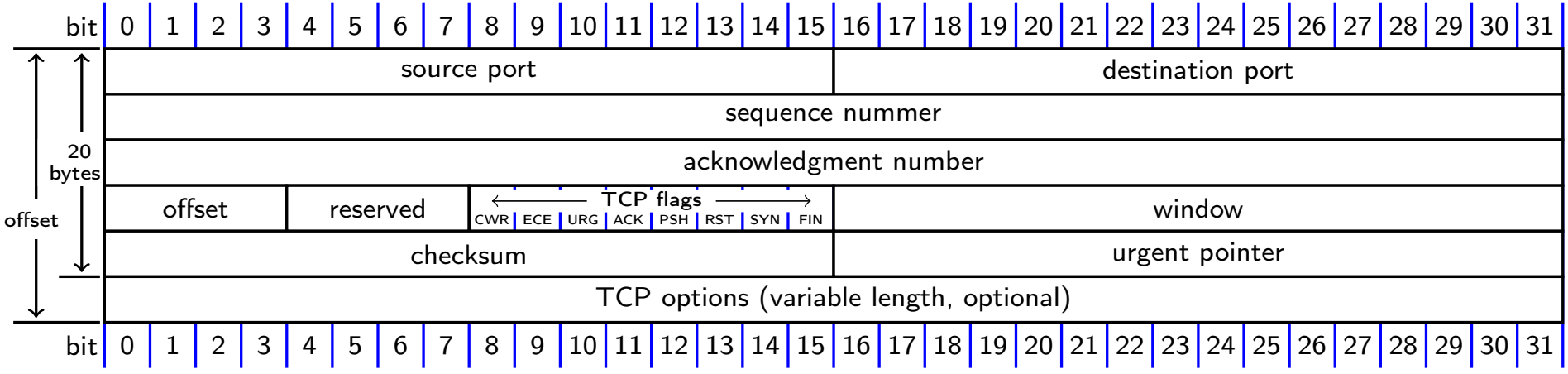
TCP (Transmission Control Protocol)

Mode connecté :

- établissement d'une connexion entre les 2 parties (initiateur/receveur)
- transmission des informations, avec accusés de réceptions du destinataire
- fermeture de la connexion

Note : agrément uniquement entre la source et la destination. Les routeurs/routes n'ont rien à voir avec la connexion TCP !

Entête TCP



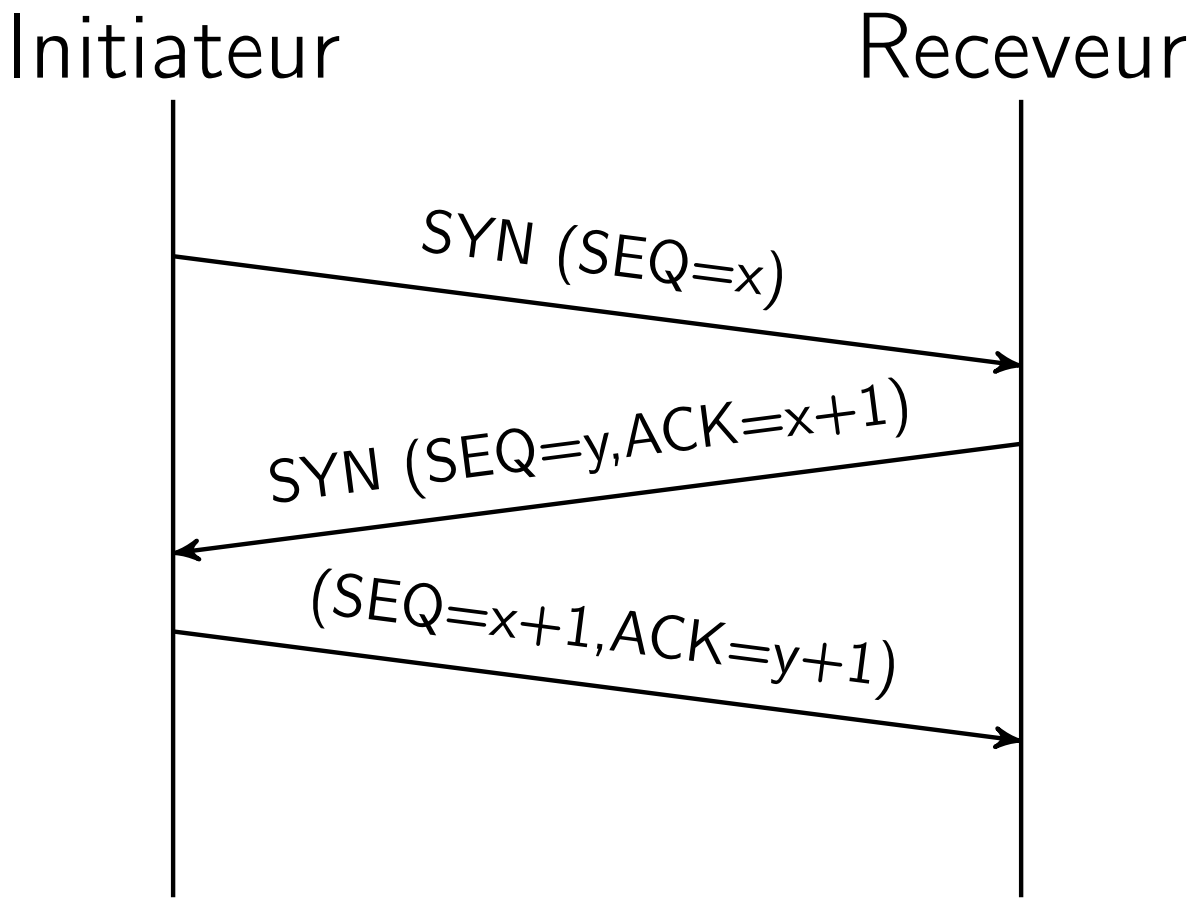
- SYN = synchronize : établissement d'une connexion
- ACK = acknowledge (accusé de réception présent)
- FIN : fermeture d'une connexion
- CWR/ECE : signalisation de congestion

TCP : Établissement de la connexion

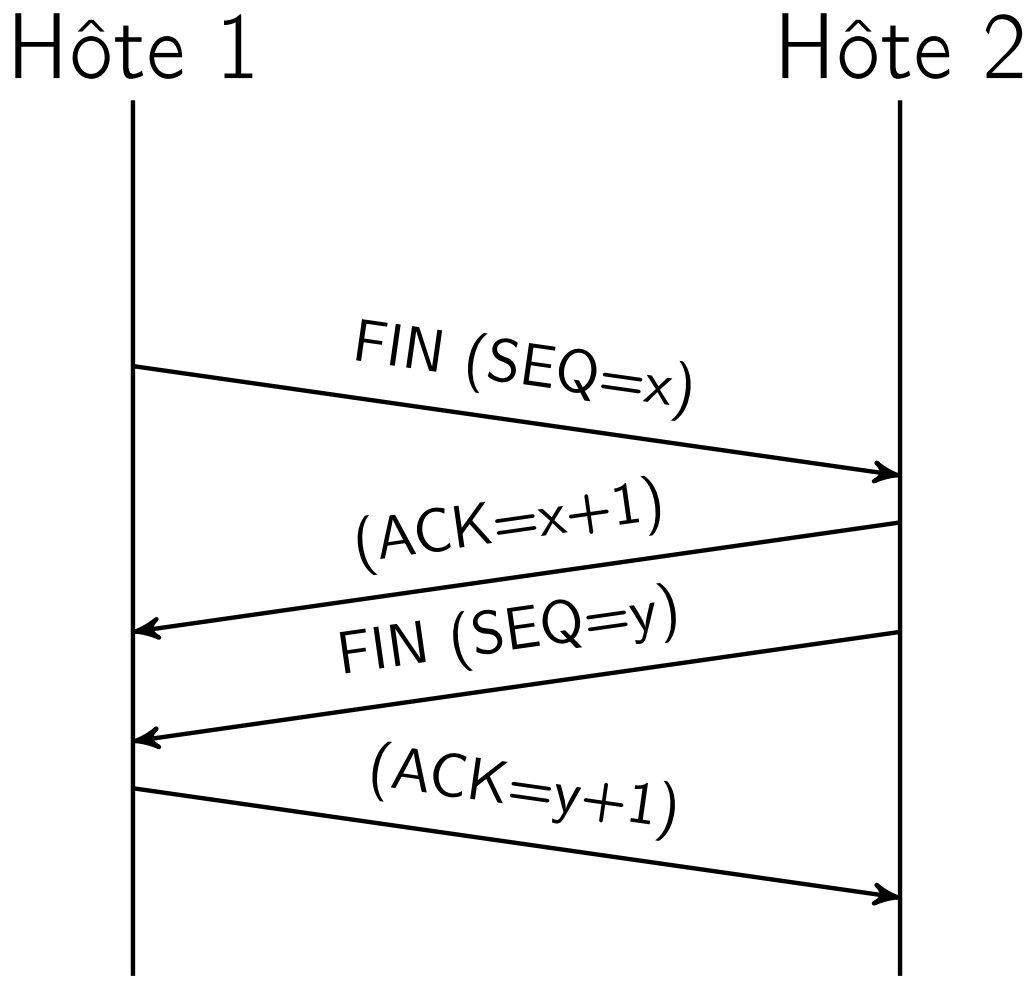
En 3 temps ("three-way-handshake")

- L'initiateur envoie au receveur un paquet SYN avec un numéro de séquence x (un nombre aléatoire)
- Le receveur envoie à l'initiateur un paquet SYN+ACK avec un numéro de séquence y (un nombre aléatoire), et l'accusé de réception $= x + 1$
- L'initiateur envoie au receveur un paquet ACK avec l'accusé de réception $= y + 1$

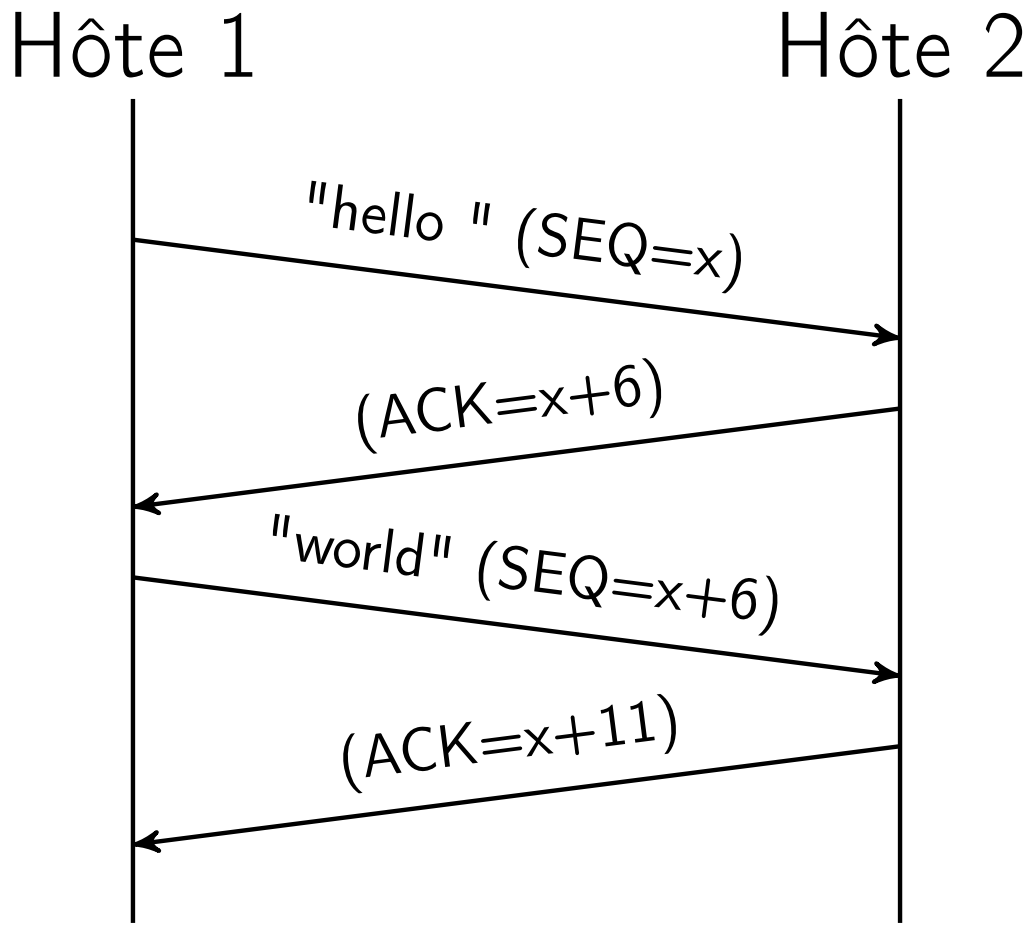
TCP : Établissement de la connexion



TCP : Fermeture de la connexion

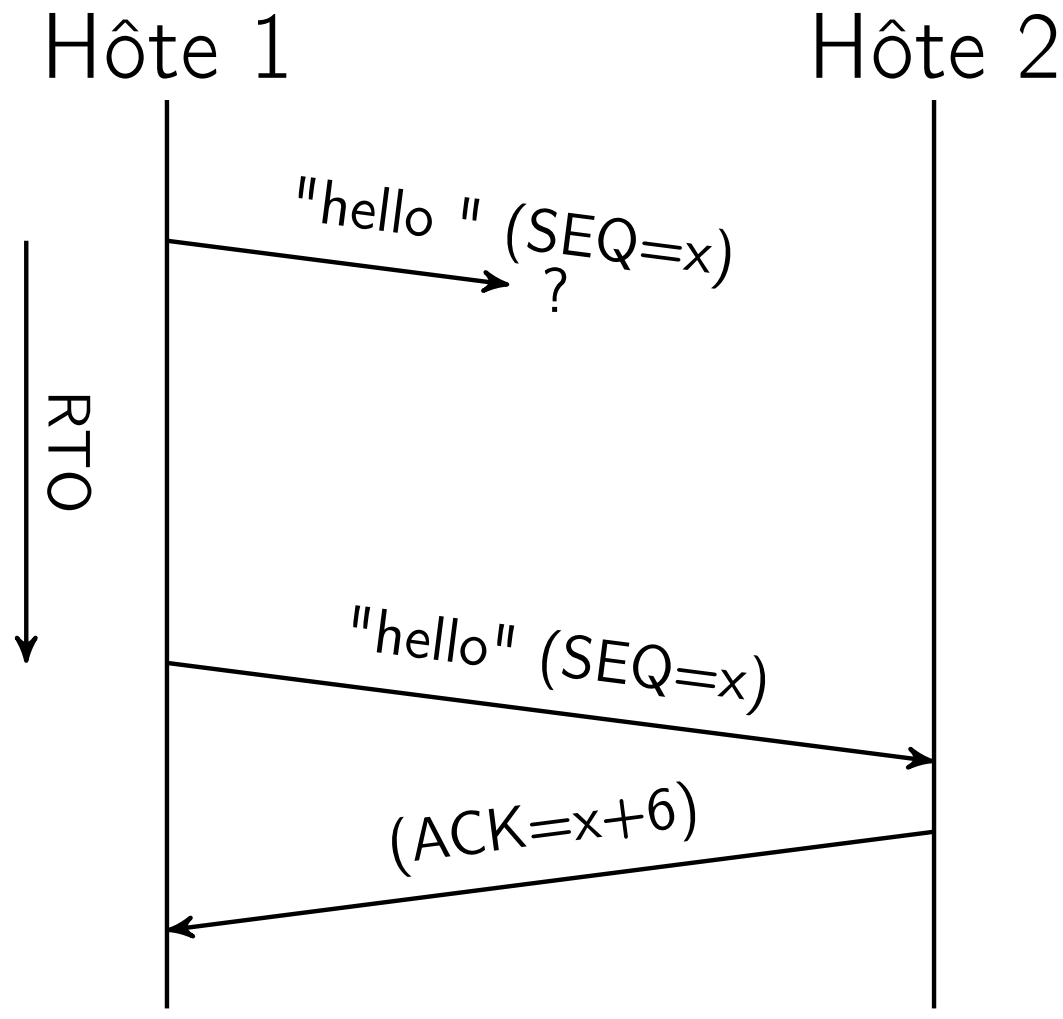


TCP : Transmission des informations



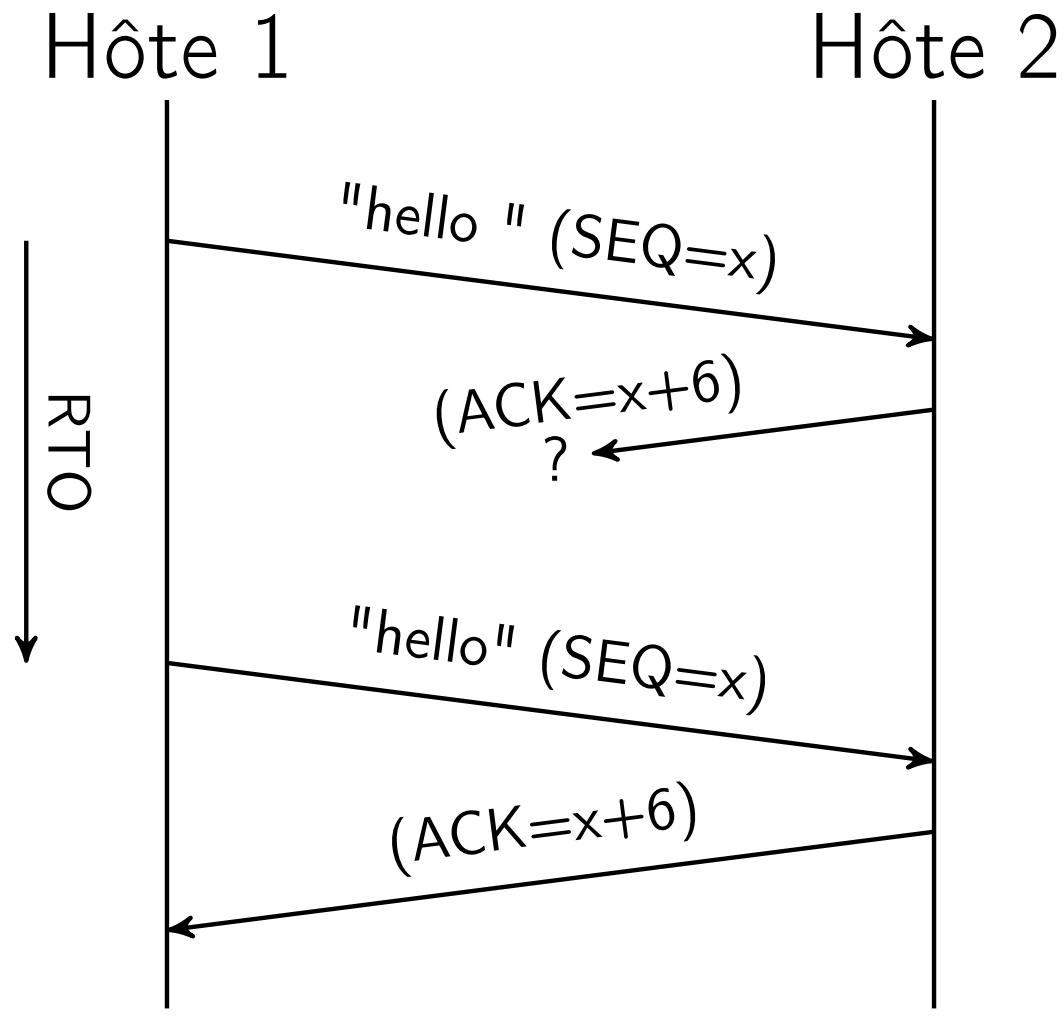
TCP : Transmission des informations

Perte de paquet ?



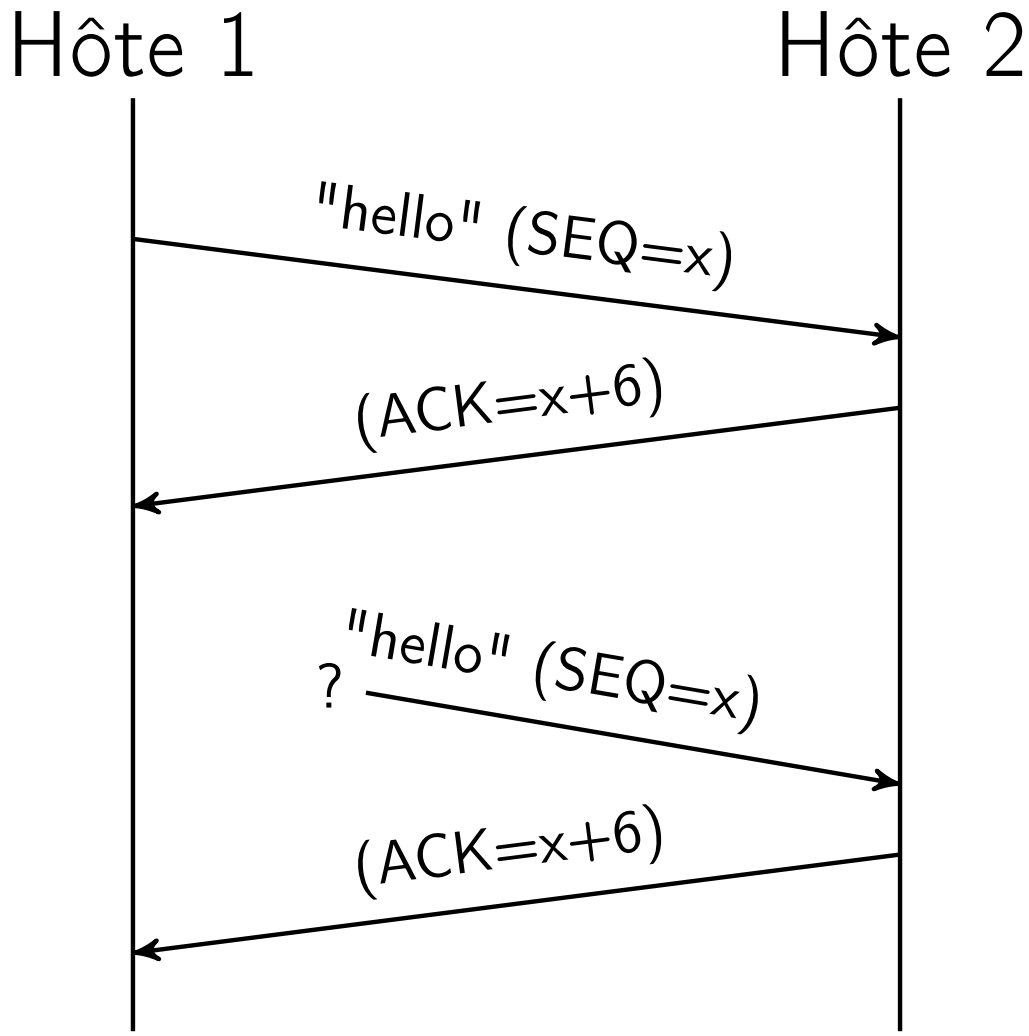
TCP : Transmission des informations

Perte de paquet ?



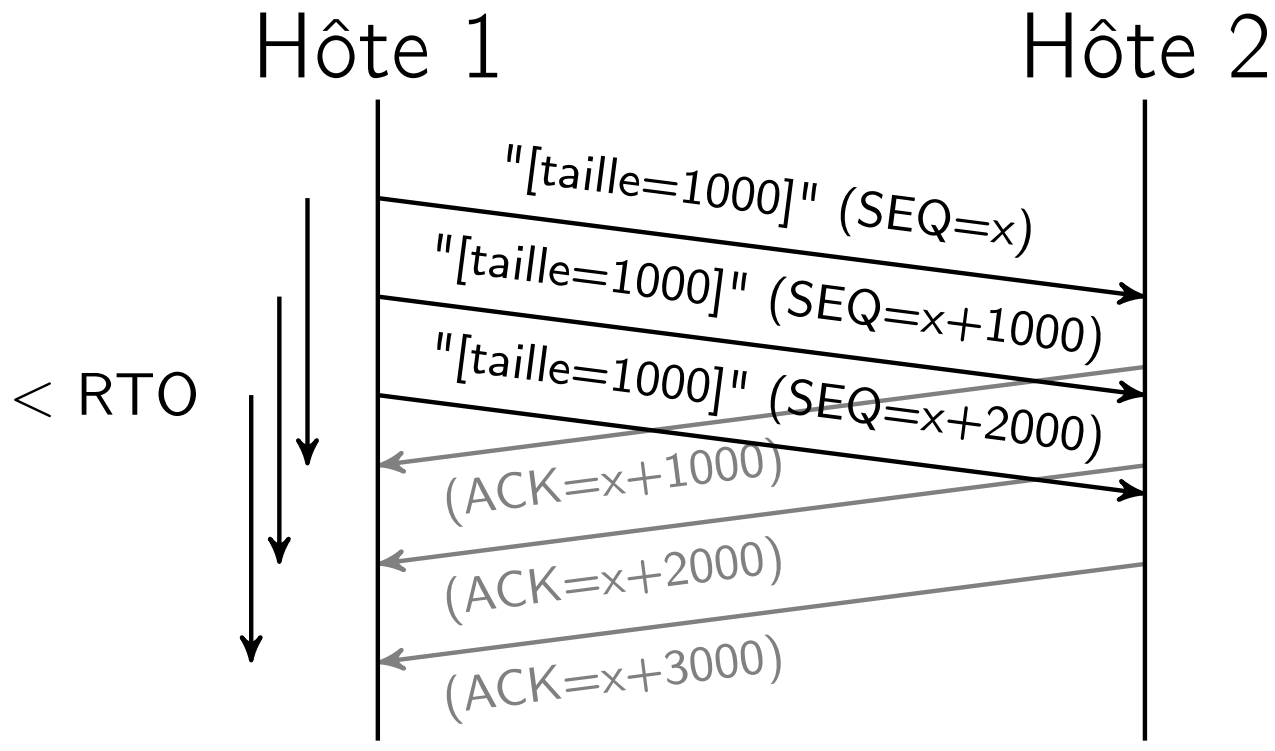
TCP : Transmission des informations

Duplication de paquet ?

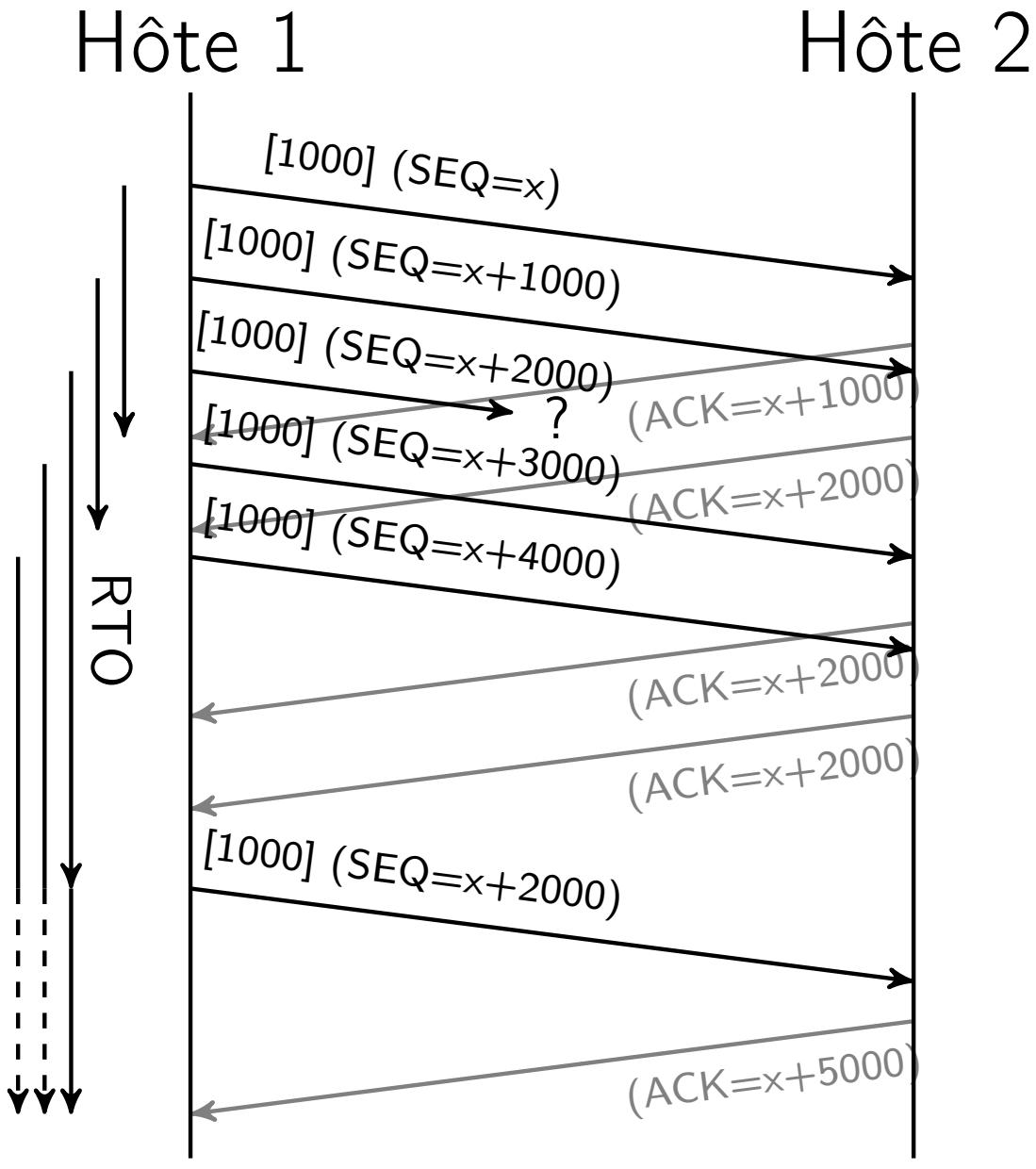


TCP : Transmission des informations

Problème : lent !



TCP : Transmission des informations



TCP : Transmission des informations

Pour limiter les ACK inutiles, il est possible :

- d'envoyer des ACK et des données en même temps
- d'attendre avant d'envoyer les ACK (et les cumuler)

Combien de temps attentre avant de retransmettre un segment ?
(RTO = Retransmission TimeOut)

- se baser sur le temps "moyen" des aller-retours
  (RTT = Round-Trip Time)
- se baser sur l'ecart type des aller-retours

Combien de temps attendre entre l'emission de deux segments ?

- l'émetteur essaye des temps de plus en plus courts, jusqu'à ce qu'il y a signalement d'une congestion, ou perte de segments.

Fenêtre TCP

La couche transport renvoie les acquittements :

- à la réception des données,
- et non pas quand l'application lit les données dans la socket

Problème : il se peut que l'application ne lise pas assez vite les données.

Solution : la couche transport dispose d'un "tampon" (ou "fenêtre")

Le récepteur envoie avec l'acquittement la taille maximum d'octets à envoyer après l'octet acquitté (la place restante dans le tampon)

- L'émetteur se mettra en attente si la fenêtre est vide (ou trop petite)
- l'écriture sera bloquante (ou échouera, si la socket est non bloquante) du coté de l'application de l'émetteur

TCP ou UDP ?

Quand utiliser TCP :

- Quand on privilégie la simplicité
- Quand on privilégie la fiabilité

Quand utiliser UDP :

- Quand on privilégie la vitesse
- Si la perte de paquet pas très importante (streaming, voix)
- Si on gère d'un autre moyen les problèmes de transfert

Généralement pour vos applications : TCP

Couche "application"

Couche "application"

Généralement en espace utilisateur

- Multitude d'applications utilisant Internet
- vos programmes

DNS

DNS = Domain Name System

Traduit des noms en adresse IP. Exemple :
`www.ens-lyon.fr` $\rightarrow$ `140.77.167.5`

- Organisation hirearchique des noms et serveurs
- Architecture client/serveur
- Ports : 53 (TCP et UDP)
- Serveur : `bind` (Berkeley Internet Name Daemon),...
- Client : dans l'OS, `nslookup`
- Sous Linux les IP des serveurs de nom à consulter : dans `/etc/resolv.conf`

DNS

Organisation hiérarchique :

- un serveur "racine" traduit "fr", et redirige vers un serveur qui traduit les ".fr"
- le serveur pour "fr" traduit "ens-lyon.fr", et redirige vers un serveur qui traduit les ".ens-lyon.fr"
- le serveur pour "ens-lyon.fr" traduit "www.ens-lyon.fr" vers un numéro IP

- 13 serveur racines (A-M), gérés par l'ICANN (Internet Corporation for Assigned Names and Numbers)
- .fr géré par AFNIC (Association française pour le nommage Internet en coopération)
- ens-lyon.fr géré par l'ENS de Lyon

DNS

exemple de fichier de configuration :

```
$TTL    86400 ; 24 hours could have been written as 24h or 1
    d
; $TTL used for all RRs without explicit TTL value
$ORIGIN example.com.
@  1D  IN  SOA ns1.example.com. hostmaster.example.com. (
                                2002022401 ; serial
                                3H ; refresh
                                15 ; retry
                                1w ; expire
                                3h ; nxdomain ttl
                               )
       IN  NS     ns1.example.com. ; in the domain
       IN  NS     ns2.smokeyjoe.com. ; external to domain
       IN  MX  10 mail.another.com. ; external mail provider
; server host definitions
ns1    IN  A      192.168.0.1  ;name server definition
www    IN  A      192.168.0.2  ;web server definition
ftp    IN  CNAME  www.example.com.  ;ftp server definition
; non server domain hosts
bill   IN  A      192.168.0.3
fred   IN  A      192.168.0.4
```

DNS

Notes :

- Les serveurs font "cache" (les données ont une durée de vie).
- Il y a généralement plusieurs serveurs de nom pour un domaine. Il y a généralement un serveur primaire (maître) et des secondaires (esclaves).
- Un nom peut avoir plusieurs IP associées ("round robin", pour répartir la charge)

Web et HTTP

WWW (Word Wide Web) : système hypertexte (contenus reliés par des hyperliens) sur internet, utilisant généralement le protocole HTTP.

HTTP (Hypertext transfert protocol)

- Client/serveur. Protocole : TCP, Port : 80
- Serveur HTTP : Apache...
- Client (Navigateur) : Firefox, Chrome...

Web et HTTP

Le serveur HTTP permet de récupérer des "pages" (avec une organisation hiérarchique).

Les pages sont identifiées par une URL (Uniform Resource Locator)

`http://serveur.org/chemin/page`

Généralement, ces pages sont au format HTML (Hypertext Markup Language).

Une page est soit :

- statique (càd provient d'un fichier sur le disque du serveur)
- dynamique : résultat de l'exécution d'un programme/script :
  - PHP
  - CGI...

Web et HTTP

Le client :

- récupère les pages identifiées par leur URL
- affiche les pages HTML, en utilisant les informations de mise en page
- exécute les scripts de la page (javascript, AJAX)

Web et HTTP

Protocole HTTP (au travers d'une connexion TCP) :

Requête HTTP :

```
GET /chermin/page.html HTTP/1.1
Host: www.serveur.com
```

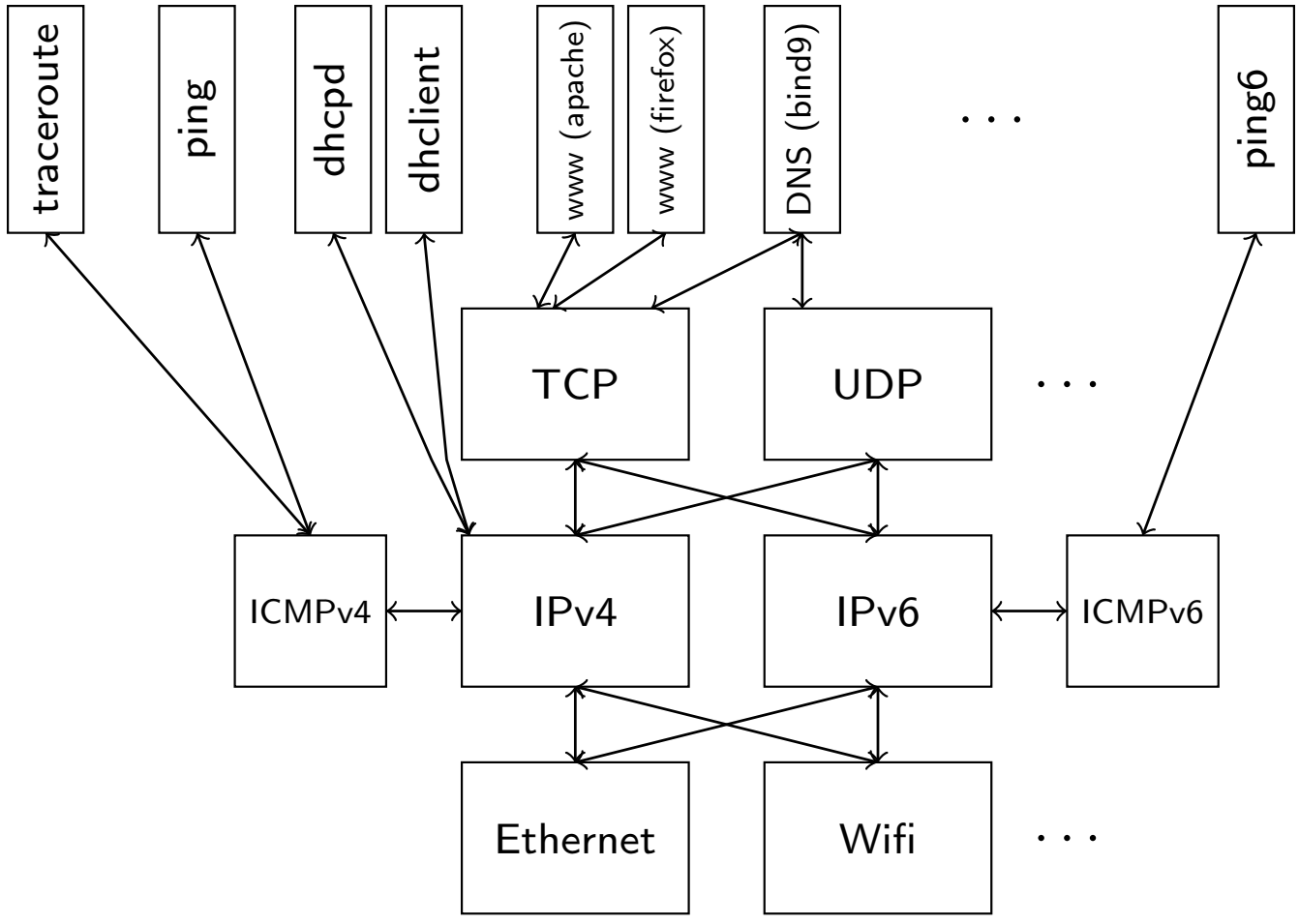
Réponse HTTP :

```
HTTP/1.1 200 OK
Date: Wed, 27 Apr 2016 14:33:32 GMT
Server: Apache/2.4.10 (Debian)
Vary: Accept-Encoding
Content-Length: 1535
Content-Type: text/html; charset=UTF-8

<!DOCTYPE html>
<html><head><title>Titre</title>
...
</head>
<body>
...
</body>
```

Autres

- HTTPS : HTTP sur SSL (Secure Sockets Layer) ou TLS (Transport Layer Security) Port : 443
- FTP (File Transfert Protocol)
- SMTP (Simple Mail Transfert Protocol)
- Récupération des courriers sur un serveur :
  - IMAP (Internet Message Access Protocol)
  - POP (Post Office Protocol)
- SSH (Secure Shell) :
  - remplace telnet (non sécurisé)


traceroute
ping
dhcpd
dhclient
www (apache)
www (firefox)
DNS (bind9)
. . .
ping6
TCP
UDP
. . .
ICMPv4
IPv4
IPv6
ICMPv6
Ethernet
Wifi
. . .

Sockets & programmation réseau POSIX

Sockets

Socket : point de communication bidirectionnel

La communication peut être :

- une communication réseau
- entre deux processus de la même machine
- entre un processus et le noyau...

Une connexion réseau :

- 2 sockets
- = les deux points finaux de la connexion

Attention :

- une socket n'est pas forcement pour une communication réseau
- une communication réseau n'est pas forcement une communication par le protocole IP

Vue générale

- créer une socket : `socket`
- associer une socket : `bind`

TCP / Côté "serveur" :

- écouter sur une socket : `listen`
- accepter une connexion : `accept`

TCP Côté "client" :

- se connecter : `connect`

Côté "client" et "serveur" :

- envoyer un message : `write`, `send`, `sento`
- recevoir un message : `read`, `recv`, `recvfrom`
- fermer une socket : `shutdown` et `close`

Créer une socket :

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

Renvoie un descripteur de fichier (-1 si erreur)

Domaine : ("domaine de communication", familles de protocoles)

- AF_UNIX : socket local (voir chapitre tubes)
- AF_INET : Internet IPV4
- AF_INET6 : internet IPV6
- AF_PACKET : paquets liaison (ethernet...)
- autres réseaux ou réseaux obsolètes (IPX, X25, AppleTalk...)

Créer une socket :

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

Type :

- SOCK_STREAM : par flot, connecté, bidirectonnel, fiable
  pour AF_INET(6), c'est TCP
- SOCK_DGRAM : par paquets, non connecté, non fiable
  pour AF_INET(6), c'est UDP
- SOCK_RAW : accès réseau "brut"
- SOCK_SEQPACKET : paquets, connecté, bidirectonnel, fiable
  pour AF_INET(6), protocole SCTP (en cours de déploiement)

Protocole :

- Pour AF_INET ou AF_INET6, c'est le numéro du protocole
  dans le paquet IP (TCP=6, UDP=17)
- Si un unique protocole existe dans le domaine/type,
  `protocole` peut être à zéro

Créer une socket :

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

Si domaine = AF_INET (IPv4) ou AF_INET6 (IPv6)

- TCP : type = SOCK_STREAM,
  protocole = 0 ou IPPROTO_TCP (=6)
- UDP : type = SOCK_DGRAM,
  protocole = 0 ou IPPROTO_UDP (=17)
- pour construire un paquet IP "brut" :
  type = SOCK_RAW, protocole > 0 (champ "protocol" dans le paquet IP)
  (et il faut les droits qui vont avec...)

Créer une socket :

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

Erreurs possibles

- EPERM : opération non permise (exemple : AF_INET/SOCK_RAW pour utilisateur lambda)
- ESOCKTNOSUPPORT : type non supporté (exemple : AF_INET/SOCK_SEQPACKET)
- EPROTONOSUPPORT : protocole non supporté
- ...

RAPPEL : Mode "flot" (STREAM)

Mode flot : les envois successifs d'informations s'additionnent.
Il n'y a pas de "séparations" entre elles.

Exemple :

- `write(in,"ABC",3)`
  - le tampon contient "ABC"
- `write(in,"123",3)`
  - le tampon contient "ABC123"
- `read(out,bf,4)`
  - renvoie 4, et bf contient "ABC1"
  - le tampon contient "23"
- `read(out,bf,4)`
  - renvoie 2, et bf contient "23"
  - le tampon est vide
- `read(out,bf,4)`
  - bloque jusqu'à ce qu'un processus écrive dans la socket...

RAPPEL : Mode paquet (DGRAM)

Au contraire du mode flot (STREAM), chaque information envoyée constitue une entité indivisible.
Exemple :

- `write(in,"ABC",3)`
  - la file de messages contient "ABC"
- `write(in,"123",3)`
  - la file contient "ABC","123"
- `read(out,bf,10)`
  - renvoie 3, et bf contient "ABC"
  - la file contient "123"
- `read(out,bf,10)`
  - renvoie 3, et bf contient "123"
  - la file vide
- `read(out,bf,4)`
  - bloque jusqu'à ce qu'un processus écrive dans la socket...

Si le tampon n'est pas assez grand, la fin est perdue !

Attacher une socket

`socket()` permet de créer une socket, mais elle n'est par défaut associée à rien (ni adresse, ni port).

Pour l'associer, il faut utiliser :
`int bind(int fd, struct sockaddr *addr, int addrlen)`

- `fd` : descripteur de fichier associé à une socket
- `addr` : pointeur sur une structure `sockaddr_*`, qui contient l'adresse et le port de la machine locale
- `addrlen` : taille de la structure addr
- renvoie 0 (OK), ou -1 (erreur)

Attacher une socket

`addr` dépend du domaine de communication. Chaque domaine à sa structure `sockaddr` (et sa taille). D'où l'intérêt de `addrlen`.

- AF_INET : sockaddr_in
- AF_INET6 : sockaddr_in6

Attacher une socket

```
struct sockaddr_in {
  sa_family_t     sin_family; /* AF_INET */
  in_port_t       sin_port;   /* port */
  struct in_addr sin_addr;    /* adresse IPv4 */
};

/* Adresse Internet */
struct in_addr {
  uint32_t        s_addr;     /* adresse */
};
```

Attacher une socket

```
struct sockaddr_in6 {
 sa_family_t     sin6_family;   /* AF_INET6 */
 in_port_t       sin6_port;     /* port */
 uint32_t        sin6_flowinfo; /* info flux */
 struct in6_addr sin6_addr;     /* adresse IPv6 */
 uint32_t        sin6_scope_id; /* Scope ID */
};

struct in6_addr {
 unsigned char   s6_addr[16];   /* adresse */
};
```

Attacher une socket

- sin_family : le domaine de communication de la socket
- sin_port : le port (TCP ou UDP)
  - si 0 : attachée à un port libre.
- (Rappel : un machine peut avoir plusieurs adresses !)
- sin_addr : l'adresse de la machine
  - INADDR_ANY : toutes les adresses possibles de la machine

Note : `bind()` est optionnel. Si on effectue un `listen()` ou un `connect()` sur une socket non affectée, elle sera affectée automatiquement sur un port libre.
$\rightarrow$ OK pour les clients, mais problématique pour les serveurs...

Connexion (TCP / initiateur de connexion)

```
int connect(int fd, struct sockaddr *addr, int addrlen)
```

- `fd` : descripteur de fichier associé à une socket
- `addr` : pointeur sur une structure `sockaddr_*`, qui contient l'adresse et le port de la machine distante
- `addrlen` : taille de la structure addr
- renvoie 0 (OK), ou -1 (erreur)

Connexion (TCP / initiateur de connexion)

Exemple client simple (web)

Écouter un port (TCP / receveur de connexion, "serveur")

Rappel : il peut y avoir plusieurs connexions sur un même port : C'est la paire (adresse/port hôte 1, adresse/port hôte 2) qui identifie une connexion

Pour écouter un port :

`int listen(int fd, int backlog)`

- `fd` : descripteur de fichier associé à une socket
- `backlog` : nombre maximum de connexions en attente

Accepter une connexion (TCP / receveur, "serveur")

`int accept(int fd, struct sockaddr *addr, int *addrlen)`
Permet de prendre connaissance des nouvelles connexions

- `fd` : descripteur de fichier associé à une socket
- `addr` : pointeur vers une structure `sockaddr_*` où sera copié l'adresse de l'initiateur de la connexion.
- `addrlen` : est un pointeur sur un entier
  - elle contient la taille maximum de la structure pointée par `addr`
  - au retour de la fonction, elle contiendra sa taille effective
- renvoie un nouveau descripteur de fichier (ou -1 si erreur)
  - c'est sur ce nouveau descripteur qu'on fera nos opérations d'envoi et d'écoute

Par défaut, `accept` est bloquant. Pour avoir le caractère non bloquant : `fcntl+O_NONBLOCK`, `select` ou `poll`.

Accepter une connexion (TCP / receveur, "serveur")

Si on veut gérer plusieurs connexions en même temps, il faut faire attention au caractère bloquant

Solutions possibles :

- utiliser plusieurs threads : un thread par connexion
- utiliser `select` ou `poll`
- passer le descripteur de fichier en mode non bloquant (et trouver une solution pour éviter les attentes actives...)

Exemple serveur simple (web)

Envoi/réception (TCP)

Les opérations d'envoi de message et de lecture peuvent se faire comme à l'accoutumé avec `write` et `read`.

Mais il existe des commandes spécifiques, avec des options en plus :
`int send(int fd, void *buffer, size_t len, int options)`

- fd, buffer, len, retour : comme dans write
- options :
  - MSG_MORE : "more to come". ne pas envoyer directement le paquet, attendre la suite.
  - MSG_OOB : Out-of-band (données "urgentes")
  - MSG_DONTWAIT : non bloquant
  - MSG_DONTROUTE : ne pas router le paquet
  - MSG_NOSIGNAL : pas de signal SIGPIPE si la connexion est fermée
  - MSG_CONFIRM ...

Note : `write(fd,buff,len)` est équivalent à
`send(fd,buff,len,0)`

Envoi/réception (TCP)

`int recv(int fd, void *buffer, size_t len, int options)`

- fd, buffer, len, retour : comme dans read
- options :
  - MSG_PEEK : ne pas enlever les données du tampon de réception
  - MSG_OOB : récupère les données Out-of-band (données "urgentes")
  - MSG_ERRQUEUE : récupérer les données de la queue d'erreurs
  - MSG_DONTWAIT : non bloquant
  - ...

Note : `read(fd,buff,len)` est équivalent à `recv(fd,buff,len,0)`

Envoi/réception (UDP)

- `connect()` sur une socket UDP (DGRAM) définit l'adresse/port où les datagrammes sont envoyés par défaut, et la seule adresse d'où les datagrammes sont acceptés.
- (pas de `listen()`/`accept()` en UDP !)

Méthode alternative :

```
ssize_t sendto(int sockfd, const void *buf,
               size_t len, int flags,
               const struct sockaddr *dest_addr,
               socklen_t addrlen);
```

Permet d'envoyer directement un datagramme l'adresse `dest_addr`, sans faire de `connect` préalable.

Envoi/réception (UDP)

```
ssize_t recvfrom(int sockfd, void *buf,
                 size_t len, int flags,
                 struct sockaddr *src_addr,
                 socklen_t *addrlen);
```

Comme `recv()`, et l'adresse source du datagramme sera copiée dans `src_addr`.

`send(sockfd, buf, len, flags)`
est équivalent à :
`sendto(sockfd, buf, len, flags, NULL, 0)`

SOCK_RAW, AF_PACKET...

Pour construire un paquet "brut" :

- par exemple ICMP (utilisé par `ping`)

```
socket(AF_INET, SOCK_RAW, IPPROTO_ICMP)
```

- un packet IP brut :

```
socket(AF_INET, SOCK_RAW, IPPROTO_RAW)
```

- un paquet Ethernet :

```
socket(AF_PACKET,SOCK_DGRAM, htons(ETH_P_IP))
```

Attention, il faut des droits particuliers :

```
$ getcap /bin/ping
/bin/ping = cap_net_raw+ep
```

Fermer une socket

`int shutdown(int sockfd, int how)`

Rappel : dans une socket TCP, chaque hôte peut indépendamment signaler la fin de ses envois.

`how` :

- `SHUT_RD` : fermeture de la réception
- `SHUT_WR` : fermeture de l'émission
- `SHUT_RDWR` : fermeture des deux directions

Puis `close()` pour fermer la socket !

`htons`...

La représentation des entiers n'est pas forcément la même sur la machine et sur internet :

- Par exemple, les x86 ont une représentation en Little endian (petit-boutiste)
- La représentation dans les packets internet est en Big endian (grand-boutiste)

Il existe des fonctions de conversion :

- `htons()` (Host TO Network Short) :
  entier 16 bits : représentation machine $\Rightarrow$ représentation réseau
- `htonl()` (Host TO Network Long) :
  entier 32 bits : représentation machine $\Rightarrow$ représentation réseau
- `ntohs()` (Network TO Host Short) :
  entier 16 bits : représentation réseau $\Rightarrow$ représentation machine
- `ntohl()` (Network TO Host Long) :
  entier 32 bits : représentation réseau $\Rightarrow$ représentation machine

inet_pton

```
#include <arpa/inet.h>
int inet_pton(int af, const char *src, void *
    dst);
```

Converti une adresse (IPv4 ou IPv6) du format texte au format binaire

- `af` : AF_INET ou AF_INET6
- `src` : la chaîne de caractère de l'adresse
- `dst` : un pointeur sur `struct in_addr` ou `struct in6_addr`

(Remplace `inet_aton()`, qui fonctionne que pour IPv4)

Opération inverse : `inet_ntop()`

Résolution de noms DNS

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

int getaddrinfo(const char *node,
                const char *service,
                const struct addrinfo *hints,
                struct addrinfo **res);

void freeaddrinfo(struct addrinfo *res);

const char *gai_strerror(int errcode);
```

Traduit les noms et/ou services.

Résolution de noms DNS

Paramètres de `getaddrinfo` :

- `node` : (si non NULL) le nom de la machine (DNS)
- `service` : (si non NULL) le nom du service (voir `/etc/services`)
- `hints` : (si non NULL) pointe sure une structure addrinfo qui contient les critères de la recherche
- `res` : où sera copiée la liste chaînée des résultats.

Pourquoi une liste ? Un nom peut avoir plusieurs translations. Par exemple IPv4 et IPv6...

Résolution de noms DNS

`getaddrinfo` renvoie 0 si OK, sinon il renvoie un code d'erreur qui peut être transformé en texte par `gai_strerror()`

`freeaddrinfo()` détruit la liste chaînée des résultats

Fonction inverse : `getnameinfo()`

Ancienne fonction (obsolète) : `gethostbyname()`

Résolution de noms DNS

```
struct addrinfo {
  int              ai_flags;     // options
  int              ai_family;    // AF_*
  int              ai_socktype;  // SOCK_*
  int              ai_protocol;  // 0,6,17...
  socklen_t        ai_addrlen;
  struct sockaddr *ai_addr;
  char            *ai_canonname;
  struct addrinfo *ai_next;
};
```

Résolution de noms DNS

```
struct addrinfo *res,*p;
int err=getaddrinfo(av[1],NULL,NULL,&res);
if(err) printf("erreur␣%s\n",gai_strerror(err));
p=res;
while(p) {
 char hostname[NI_MAXHOST];
 err=getnameinfo(p->ai_addr,p->ai_addrlen,hostname,
     NI_MAXHOST,NULL,0,NI_NUMERICHOST);
 if(err)
  printf("erreur␣%s\n",gai_strerror(err));
 else
  printf("hostname:␣%s\n", hostname);
 p=p->ai_next;
}
freeaddrinfo(res);
```

Option des sockets

```
int getsockopt(int sockfd, int level, int optname,
               void *optval, socklen_t *optlen)
int setsockopt(int sockfd, int level, int optname,
               const void *optval, socklen_t optlen)
```

Permet de lire/modifier les options d'une socket

Par exemple, pour désactiver l'"algorithme de Nagle", qui fait attendre qu'il y ait assez de données avant d'envoyer un packet :

```
int one = 1;
setsockopt(fd, SOL_TCP,
           TCP_NODELAY, &one, sizeof(one));
```

voir `man 7 socket`, `man 7 ip`, `man 7 tcp`...

Administration réseau sous Linux (vue rapide)

ifconfig

Liste et configure les interfaces réseau. Exemples :

Afficher toutes les interfaces

```
ifconfig -a
```

Définir l'adresse IP de eth0 (ethernet)

```
ifconfig eth0 up 192.168.0.1/24
```

Changer l'adresse MAC d'une interface :

```
ifconfig eth0 hw ether ef:42:03:17:a5:6f
```

iwlist/iwconfig

Liste les informations et configure les interfaces réseau sans-fil.
Exemples :

Lister les réseaux sans fil :

```
iwlist wlan0 scanning
```

Connexion à un point d'accès ouvert

```
iwconfig wlan0 essid nom_reseau
```

route

Liste et administre la table de routage statique

Afficher les routes :

```
route
```

Ajout d'une route vers un hôte :

```
route add 192.168.2.4 gw 192.168.1.2
```

Ajout d'une route vers un sous-réseau :

```
route add -net 192.168.3.0/24 gw 192.168.1.5
```

Afficher les connexions, voir les paquets...

- `netstat` : Affiche les connexions réseau, statistiques...
- `iptraf` : Affiche les connexions réseau, statistiques... en interactif
- `wireshark` : Analyseur de paquets interactif
- `ngrep` : Fait une recherche (grep) sur les paquets réseau

autres commandes utiles

Vérification du réseau :

- `ping`
- `traceroute`

DNS

- `host`
- `nslookup`

Utilitaires

- `nc` : "TCP/IP swiss army knife"
  (attention : informations transitent en clair)
- `ssh` : copies de fichier par le réseau, proxy, redirection de port entre différentes machines (chiffré)

iptables

Outil d'administration du filtrage de paquets IP

Afficher les filtres :

```
iptables -L -v -n
```

Ajouter un filtre (ignorer de tout paquet reçu d'un sous réseau)

```
iptables -A INPUT -s 142.17.0.0/16 -j DROP
```

rejet de tout paquet TCP reçu avec port de destination 22

```
iptables -A INPUT -p tcp --dport 22 -j REJECT
```

- Énormément de filtres possibles
- Fonctionne avec des listes. On peut choisir de rejeter tout par défaut, sauf les paquets qui matchent...