# Programmation réseau

Y. Guesnet

Département d'informatique
Université de Rouen

11 février 2020

# Plan

# Plan

# Le cours de programmation réseau

## Objectifs

- Ce premier cours de réseau a pour but de vous présenter les réseaux ainsi que les appels système du système d'exploitation qui permettent de communiquer à travers ces réseaux.
- L'activité consistant à écrire le code prenant en charge la communication réseau à l'aide des appels système est appelée programmation socket.

# Bibliographie



Christophe Blaess.
*Programmation système en C sous Linux, 2e édition*.
Eyrolles, Paris, 2005.


José Dordoigne.
*Réseaux informatiques*.
Éditions ENI, St Herblain, 2005.


Andrew Tanenbaum.
*Réseaux, 4e édition*.
Pearson Education France, Paris, 2008.

# Qu'est-ce qu'un réseau ?

## Réseau

Un réseau d'ordinateurs désigne un ensemble d'ordinateurs autonomes interconnectés au moyen d'une seule technologie.

## Interconnexion

Deux ordinateurs sont interconnectés s'ils sont aptes à échanger des informations.

# Exemples

## Exemples

- Ethernet
- IEEE 802.11 (WiFi : *Wireless Fidelity*)
- ATM (*Asynchronous Transfer Mode*)
- L'internet ?

## Remarque

On associe souvent le nom du protocole utilisé au réseau lui-même.

# Utilisation des réseaux

## Applications

Les réseaux peuvent avoir de multiples applications, citons notamment :

- le partage d'informations (fichiers, base de données) ;
- le partage des ressources physiques (imprimante, scanner, ...) ;
- bureau portable (smartphones, WIFI) ;
- moyen de communication entre individus (messagerie, vidéoconférence, téléphonie IP, ...) ;
- le commerce (passage de commandes, paiements en ligne) ;
- le divertissement (jeux).

# Le modèle client-serveur

## Modèle

Dans le cas de données devant être accédées à distance, on utilise souvent le modèle client-serveur :

- les données sont stockées sur des ordinateurs puissants appelés serveurs ;
- des ordinateurs plus simples, appelés clients, peuvent alors être utilisés pour accéder aux données.

## La communication

En pratique ce modèle met en jeu deux types processus :

- les processus clients envoient des messages au processus serveur et attendent les réponses ;
- pour chaque message, le processus serveur exécute la tâche demandée et envoie la réponse.

# Le modèle peer-to-peer

## Modèle

- Une autre forme de communication consiste à communiquer entre systèmes homologues : le peer-to-peer.
- Dans ce modèle les utilisateurs forment un groupe au sein duquel chacun peut communiquer avec les autres. Il n'existe pas de clients ni de serveurs fixes.

# Plan

# Classification

## Critères de classification

On utilise généralement les deux critères suivants pour caractériser un réseau :

- la technologie de transmission utilisée ;
- sa taille.

# La technologie de transmission

## Deux types de transmission

On distingue généralement deux types de transmission :

- la diffusion ;
- le point-à-point.

# La diffusion

## La diffusion

Un réseau à diffusion (*broadcast*) dispose d'un seul canal de transmission qui est partagé par tous les équipements. Chaque message envoyé est reçu par toutes les machines du réseau.

## L'adresse

Un champ d'adresse permet d'identifier le destinataire. À la réception du message, la machine lit l'adresse et procède au traitement du message si elle est la destinataire ou ignore le message.

# La diffusion

## Plusieurs destinataires

- On peut également adresser un paquet à toutes les destinations en utilisant une adresse spéciale : il s'agit d'une diffusion générale (envoi broadcast).
- On peut également, parfois, adresser un message à un sous-ensemble des machines du réseau : on parle alors de diffusion restreinte (ou diffusion de groupe ou encore diffusion multipoint, diffusion *multicast*).

# Le point-à-point

## Le point-à-point

- Un réseau point-à-point fait intervenir un grand nombre de connexions, chacune faisant intervenir deux machines.
- Une transmission point-à-point entre un expéditeur et un destinataire est appelée envoi unicast.
- Un message peut alors transiter par plusieurs machines avant d'arriver à destination.

## Utilisations

Le système à diffusion sera plutôt utilisé sur des petits réseaux géographiquement limités alors que le système point-à-point le sera plutôt sur de grands réseaux.

# La taille

## Classification en fonction de la taille

On peut classer les systèmes d'interconnexion selon leur taille :

- le réseau personnel (PAN pour *Personal Area Network*) se destine à une personne (réseau sans fil reliant l'ordinateur à la souris et le clavier) ;
- les réseaux de plus grande étendue : les réseaux locaux, métropolitains et longue distance ;
- l'interconnexion de plusieurs réseaux (internet).

La distance entre les machines est une donnée importante car c'est elle qui dictera les techniques à employer.

# La taille

## Tableau récapitulatif

| Distance | Emplacement | |
|---|---|---|
| 1m | $1m^2$ | Réseau personnel |
| 10m | Une salle | Réseau local |
| 100m | Un immeuble | |
| 1km | Un campus | |
| 10km | Une ville | Réseau métropolitain |
| 100km | Un pays | Réseau longue distance |
| 1000km | Un continent | |
| 10000km | Une planète | Internet |

# Les réseaux locaux

## LAN

Les réseaux locaux ou LAN (*Local Area Network*) sont des réseaux privés dont la taille peut atteindre plusieurs kilomètres. Ils se distinguent des autres catégories de réseaux par :

- leur taille (délai de transmission connu) ;
- leur technologie de transmission (de 100 Mbit/s à 10 Gbit/s) ;
- leur topologie (bus : IEEE 802.3 ou éthernet, anneau : IEEE 802.5, ...).

# Les réseaux métropolitains

## MAN

- Les réseaux métropolitains ou MAN (*Metropolitan Area Network*) couvrent une ville.
- L'exemple le plus connu est la télévision par câble.

# Les réseaux longue distance

## WAN

Les réseaux longue distance ou WAN (*Wide Area Network*) s'étendent sur une vaste zone géographique (un pays voire un continent).

## Architecture

- Le réseau regroupe un ensemble d'ordinateurs hôtes.
- Ces hôtes sont reliés par un sous-réseau de communication.
- Le sous-réseau appartient généralement à l'opérateur de télécommunications ou au fournisseur d'accès.
- Les hôtes appartiennent aux clients.

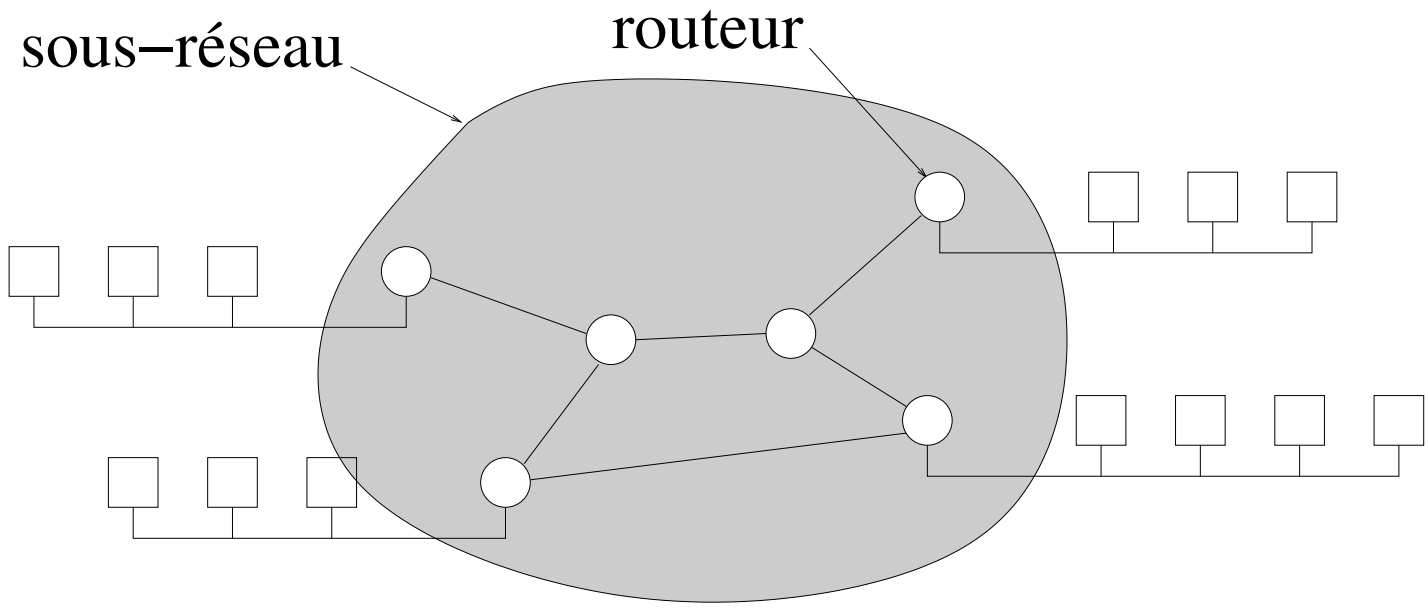
# Le sous-réseau

## Architecture du sous-réseau

Le sous-réseau se compose généralement de :

- lignes de transmissions (fil de cuivre, fibre optique, liaisons radio) ;
- équipements de commutation (les routeurs).

# Le sous-réseau



# Le sous-réseau

## Polysémie

Attention le terme de sous-réseau peut recouvrir deux notions. Il peut s'agir de :

- l'ensemble des routeurs et des lignes de transmission chargés d'acheminer des paquets ;
- l'entité issue du partitionnement à l'aide des adresses d'un réseau.

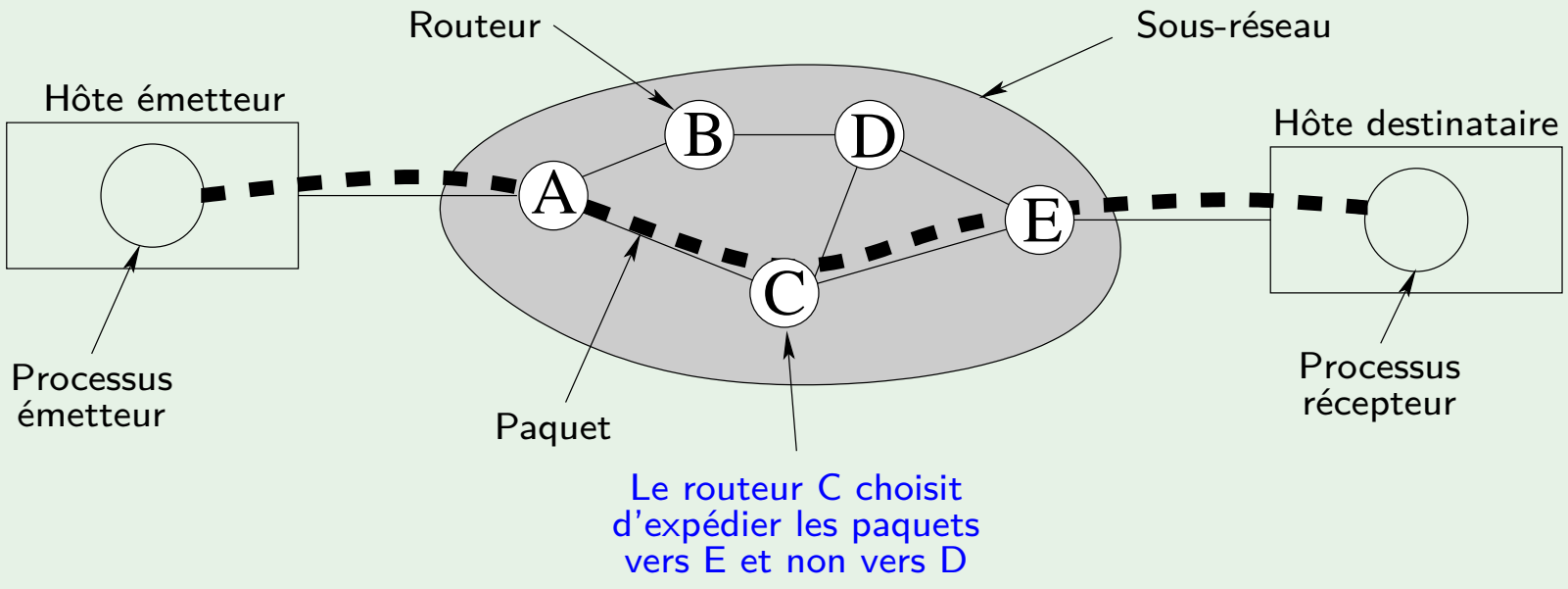
# Le sous-réseau

## Mode de transmission

Quasiment tous les WAN fonctionnent en mode différé (ou commutation de paquet) : lorsqu'un paquet arrive sur un équipement intermédiaire, il est reçu en totalité puis mis en attente en attendant que la ligne de sortie se libère.

# Le sous-réseau

## Flot de paquets




## Routage

La façon dont un routeur détermine la ligne de sortie est appelée un algorithme de routage.

# Les réseaux sans fil

## Classification

On classe généralement les réseaux sans fils en trois catégories :

- l'interconnexion de systèmes (bluetooth) ;
- les LAN sans fils (IEEE 802.11) ;
- les MAN sans fils (BLR : IEEE 802.16).

# Les inter-réseaux

## Définitions

Afin de relier des réseaux différents entre eux, on utilise des passerelles (*gateways*).
Un ensemble de réseaux ainsi reliés s'appelle un inter-réseau.

## Exemples

- L'exemple le plus connu d'inter-réseau est internet.
- Des LAN reliés par l'intermédiaire d'un WAN forment également un inter-réseau.

# Plan

# Abstraction

## Couches

La plupart des réseaux sont organisés en couches situées les unes au-dessus des autres.

- Chaque couche représente un niveau d'abstraction. Le rôle d'une couche est de fournir des services à la couche supérieure tout en lui dissimulant les détails trop techniques de l'implémentation.
- La couche *n* d'une machine dialogue avec la couche *n* d'une autre machine (communication entre pairs).

# Communication

## Définition

Un protocole est une convention acceptée par les parties communicantes sur la façon dont leur dialogue doit se dérouler.

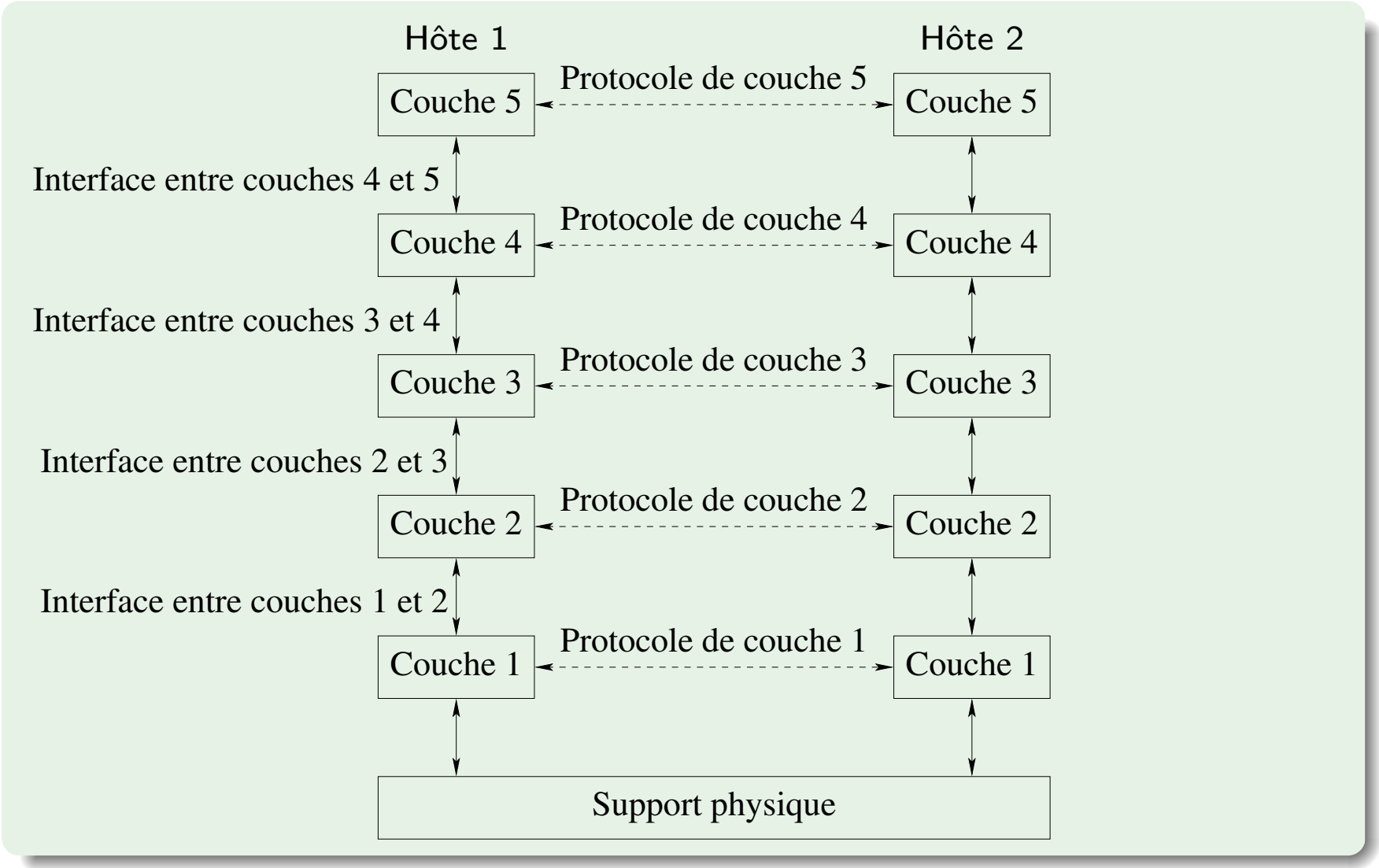
## Transmission

Chaque couche passe les données et les informations de contrôle à la couche inférieure jusqu'à la couche la plus basse. Le support physique se charge alors de la transmission des données.

## Définition

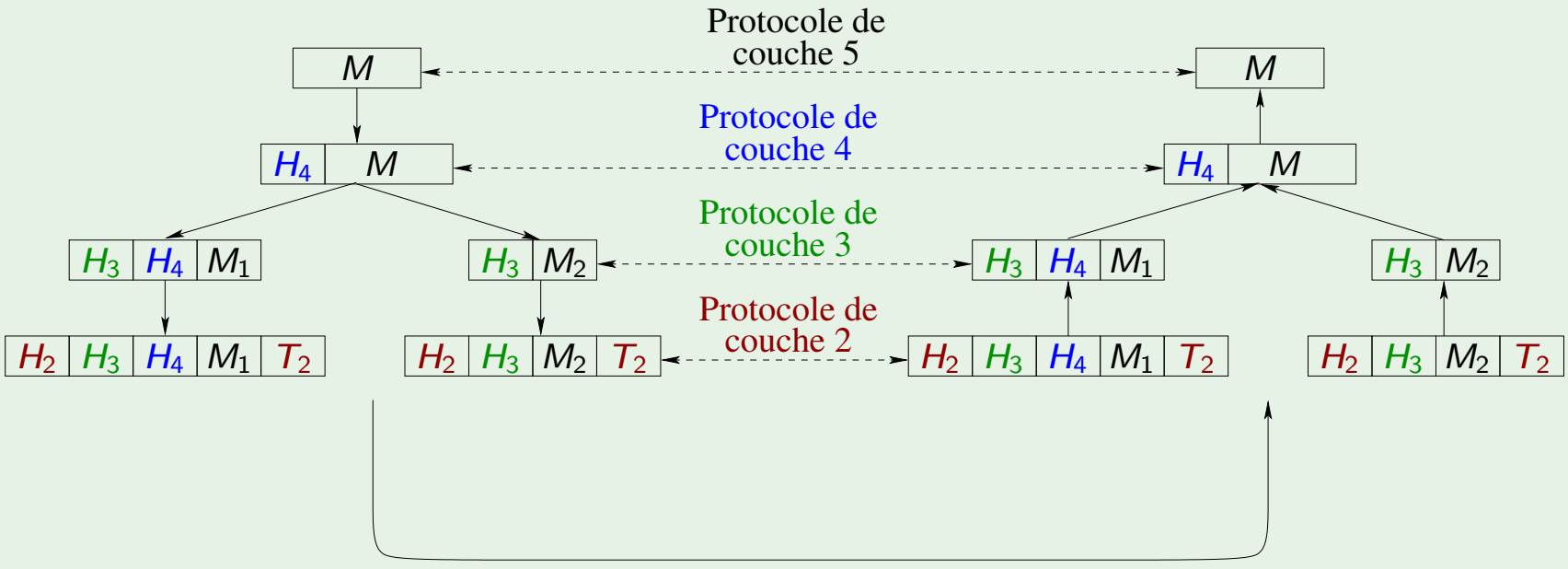
Entre chaque paire de couches adjacentes on trouve une interface qui définit les opérations fondamentales et les services que la couche inférieure offre à la couche supérieure.

# Résumé


Hôte 1
Hôte 2
Couche 5
Protocole de couche 5
Couche 5
Interface entre couches 4 et 5
Couche 4
Protocole de couche 4
Couche 4
Interface entre couches 3 et 4
Couche 3
Protocole de couche 3
Couche 3
Interface entre couches 2 et 3
Couche 2
Protocole de couche 2
Couche 2
Interface entre couches 1 et 2
Couche 1
Protocole de couche 1
Couche 1
Support physique

# La transmission

## Envoi d'un message $M$



# Communication

**Définition**

L'ensemble des couches et des protocoles d'un réseau forment l'architecture réseau.

**Définition**

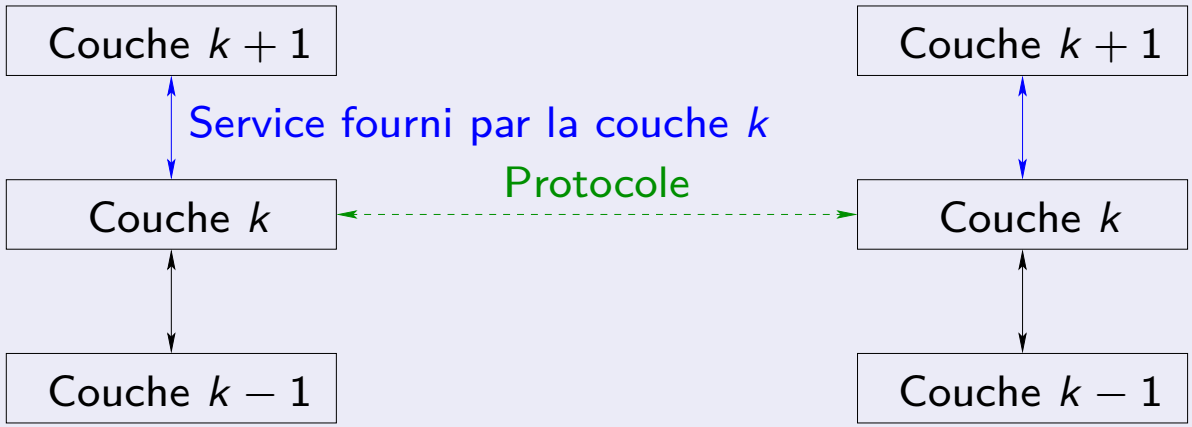
L'ensemble des protocoles utilisés par un système, avec un protocole par couche, est appelé pile de protocoles.

# Services

## Primitives

- Un service est défini par l'ensemble des primitives qu'un processus utilisateur peut employer pour accéder au service.
- Il ne faut pas confondre service et protocole.

## Relation service/protocole



# Services

## Connexions

Les couches peuvent offrir deux types de services aux couches supérieures :

- un service avec connexion nécessite l'établissement d'une connexion, suivie éventuellement d'une négociation, puis de son utilisation et enfin de sa libération ;
- dans un service sans connexion, chaque message est envoyé avec son adresse de destination.

# Connexions

## Qualité de service

- Un service qualifié de fiable ne perd jamais de données (comme lors d'un transfert de fichiers, par exemple).
- Certains services privilégient avant tout la vitesse de transmission, on peut alors autoriser la perte de quelques données (téléphonie, conférences vidéos).

# Connexions

## Définition

- Un service non fiable sans connexion est appelé un service datagramme.
- Le service datagramme acquitté, quant-à lui, permet de s'assurer que les messages ont bien été reçus.

## Définition

Un service demande-réponse permet l'envoi d'un datagramme contenant une demande et, en retour, la réception de la réponse.

# Plan

# Le modèle OSI

## Présentation du modèle

Le modèle de référence OSI (*Open Systems Interconnection*) a été développé par l'ISO (*International Organization for Standardization)* pour permettre la normalisation des protocoles des différentes couches.

- Le modèle OSI se compose de sept couches.
- Le modèle ne spécifie pas les services et les protocoles, il se contente de préciser ce que chaque couche doit faire.
- L'ISO a également produit des protocoles pour chaque couche mais ils sont rarement employés.

# Le modèle OSI

## Principes

Les principes qui ont régi l'élaboration des couches sont les suivants.

- Une couche doit être créée lorsqu'un nouveau niveau d'abstraction est nécessaire.
- Chaque couche doit assurer une fonction bien définie.
- La fonction de chaque couche doit être choisie en visant la définition de protocoles normalisés.
- Les limites d'une couche doivent être fixées de manière à réduire la quantité d'informations devant passer au travers des interfaces.
- Le nombre des couches doit être suffisamment grand pour que les fonctions soit bien réparties mais suffisamment faible pour que l'architecture ne soit pas trop complexe.

# Le modèle OSI

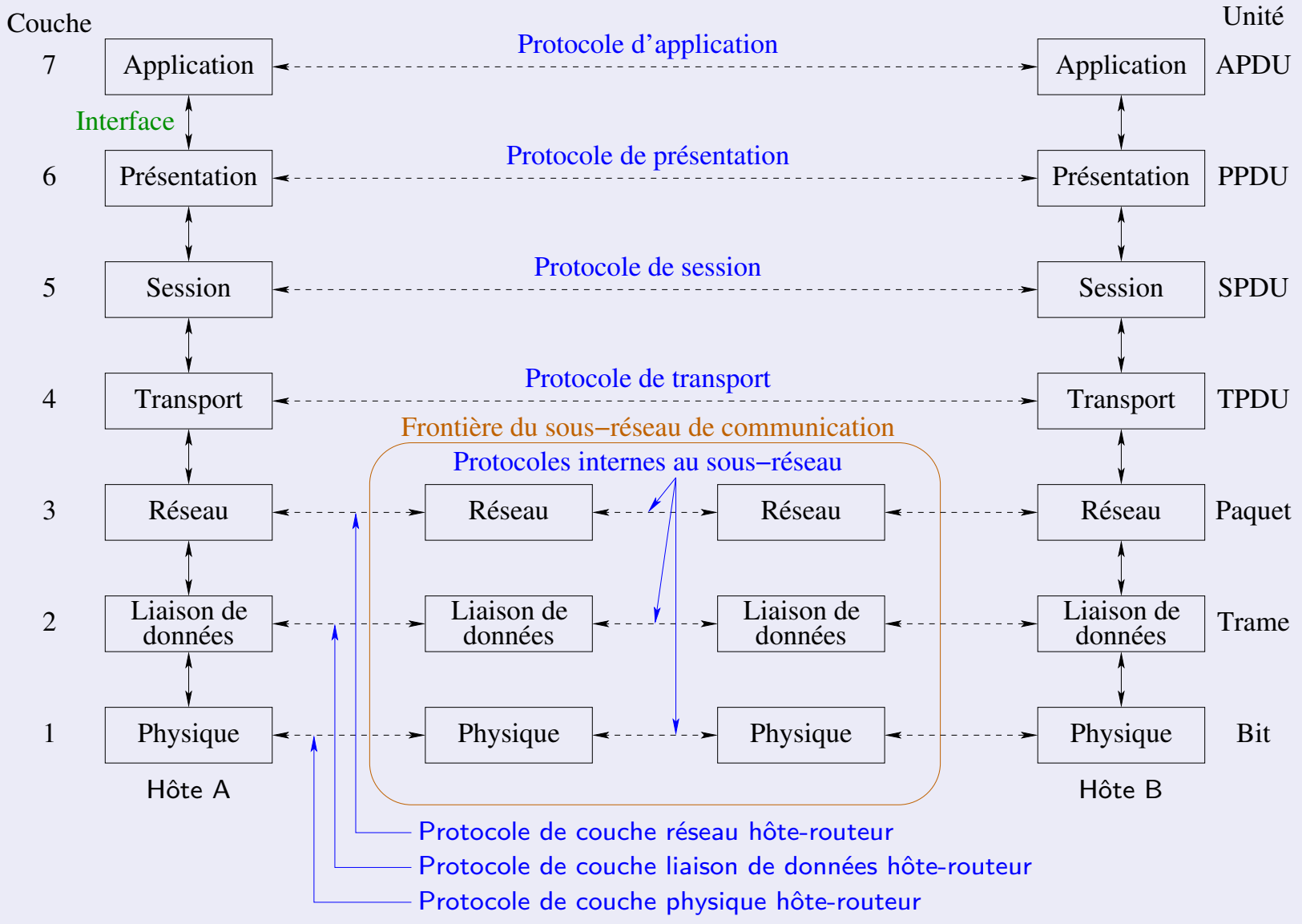
## Concepts

Trois concepts sont au cœur du modèle OSI.

1. Les services (ce que fait une couche).
2. Les interfaces (comment accéder à la couche).
3. Les protocoles (la cuisine interne à la couche).

# Le modèle OSI

## Schéma général



# Le modèle OSI : la couche physique

## La couche physique

La couche physique se charge de la transmission des bits à l'état brut.

## Objectif

- S'assurer qu'un bit à 1 envoyé à une extrémité arrive aussi à 1 à l'autre extrémité.
- La couche physique définit les méthodes d'accès au support, le codage, les topologies supportées, les débits, ...

# La couche physique

## Questions

- Nombre de volt à fournir pour un 1, pour un 0 ?
- Nombre de nanosecondes que doit durer un bit ?
- Possibilité d'envoi dans les deux sens ?
- Établissement/libération d'une connexion ?
- Nombre de broche d'un connecteur et leur rôle ?

## Conception

Les problèmes de conceptions concernent les interfaces mécaniques et électriques, la synchronisation ainsi que le support physique de transmission.

# Le modèle OSI : la couche liaison de données

## La couche liaison de données

La couche liaison de données fournit une liaison exempte d'erreurs de transmission à la couche supérieure (la couche réseau).

## Trames

- La couche liaison de données décompose les données de l'émetteur en trames de données et envoie les trames en séquence.
- Le service est fiable : le récepteur confirme la bonne réception de chaque trame en envoyant à l'émetteur une trame d'acquittement.

# La couche liaison de données

## Régulation

La couche doit également prendre en charge la régulation de l'émission des données afin d'éviter qu'un récepteur lent soit submergé par un émetteur rapide.

## Diffusion

Pour les réseaux à diffusion, il faut également contrôler l'accès au canal partagé. C'est une sous-couche de la couche liaison de données qui gère ce problème : la sous-couche d'accès au média (MAC).

# Le modèle OSI : la couche réseau

## La couche réseau

La couche réseau contrôle le fonctionnement du sous-réseau.

## Routage

Elle a en charge le routage des paquets de la source vers la destination.

## QoS

Elle gère également la qualité de service : délais, temps de transit, congestion, ...

# La couche réseau

## Interconnexion

La couche réseau doit prendre en charge l'hétérogénéité des réseaux :

- techniques d'adressages pouvant être différentes ;
- paquet pouvant être rejeté car trop gros ;
- protocoles différents, ...

## Diffusion

Sur un réseau à diffusion, la technique de routage étant très simple, la couche réseau est très mince, voire inexistante.

# Le modèle OSI : la couche transport

## La couche transport

La couche transport accepte les données des couches supérieures, les divise en unités plus petites si nécessaire et les transmet à la couche réseau.
Elle doit s'assurer qu'elles sont correctement acheminées.

## Technologies

La couche transport isole les couches supérieures des changements de matériels inévitables.

# La couche transport

## Services

- La couche transport détermine le type de service à fournir aux utilisateurs du réseau (connexion point à point, remise de messages isolés, diffusion de messages, ...).
- Le type de service est déterminé à la connexion.

## Communication directe

La couche transport offre un service de bout-en-bout : c'est la première couche où les machines sources et destinations conversent directement ; dans les couches inférieures, la communication a lieu de proche en proche.

# Le modèle OSI : la couche session

## La couche session

La couche session permet aux utilisateurs de différentes machines d'établir des sessions.
Une session offre divers services, dont :

- la gestion du dialogue (suivi du tour de transmission) ;
- la gestion du jeton (empêche deux participants de tenter la même opération critique en même temps) ;
- la synchronisation (possibilité de reprendre une transmission là où elle s'était interrompue).

# Le modèle OSI : la couche présentation

## La couche présentation

La couche présentation s'intéresse à la syntaxe et à la sémantique des informations transmises.

## Représentation des données

- Elle a en charge la gestion des structures de données abstraites échangées entre des machines qui peuvent avoir des représentations de données différentes.
- Elle définit également un système d'encodage standard.
- Elle permet l'échange de structures de données de plus haut niveau.

# Le modèle OSI : la couche application

## La couche application

La couche application contient une variété de protocoles qui seront utiles aux utilisateurs.

## Protocoles d'application

Par exemple, il existe des protocoles pour consulter des pages web, pour le transfert de fichiers ou encore pour l'échange de mails.

# Le modèle TCP/IP

## Autres modèles

- Il existe d'autres modèles de référence pour les réseaux même s'ils sont peu employés.
- Par exemple, le modèle de référence d'internet est issu du réseau historique ARPAnet. Il s'agit du modèle TCP/IP.

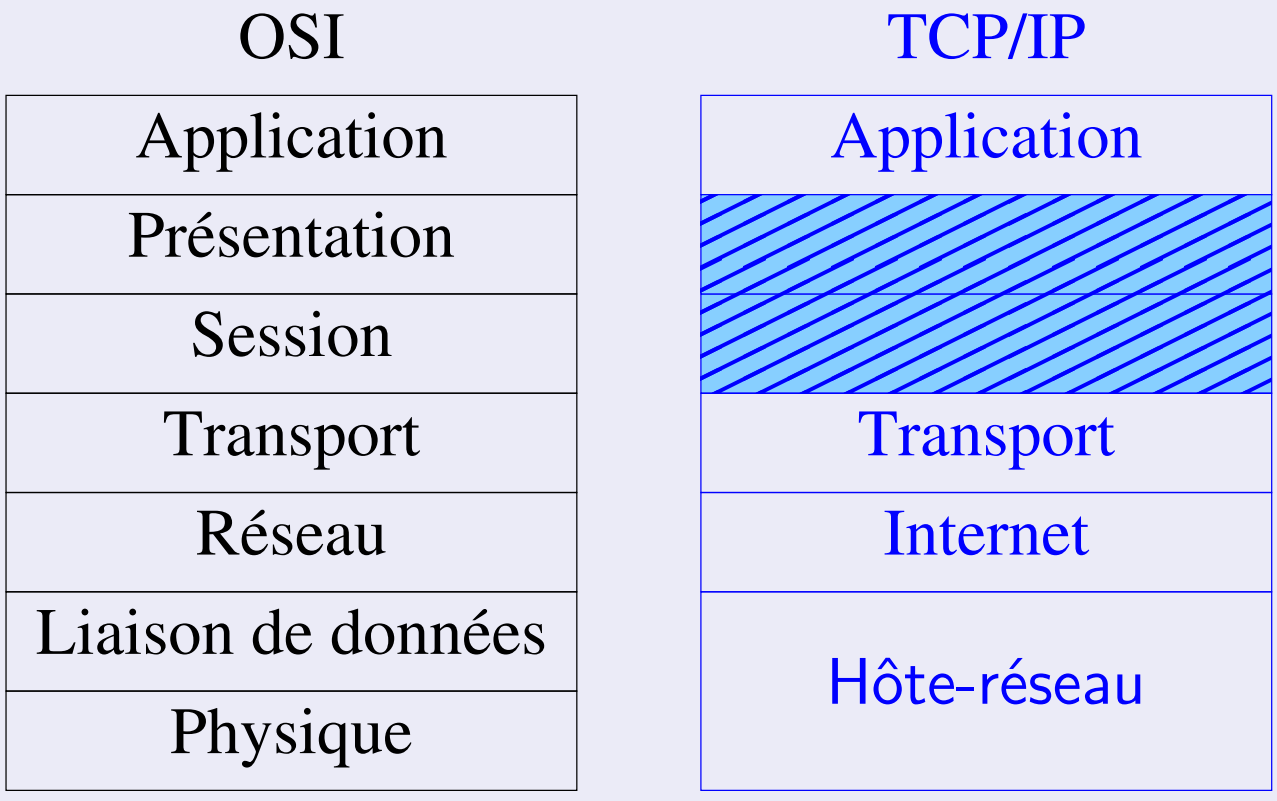
## Le modèle TCP/IP

Le modèle TCP/IP repose sur trois couches. Ses objectifs sont :

- de permettre d'interconnecter de nombreux réseaux de façon transparente ;
- le maintien de la disponibilité du système même en cas de panne d'une partie des équipements ou des lignes de transmission ;
- une grande souplesse au niveau de l'architecture pour permettre l'utilisation d'applications aux exigences très variées.

# Le modèle TCP/IP

## Les modèle TCP/IP et OSI



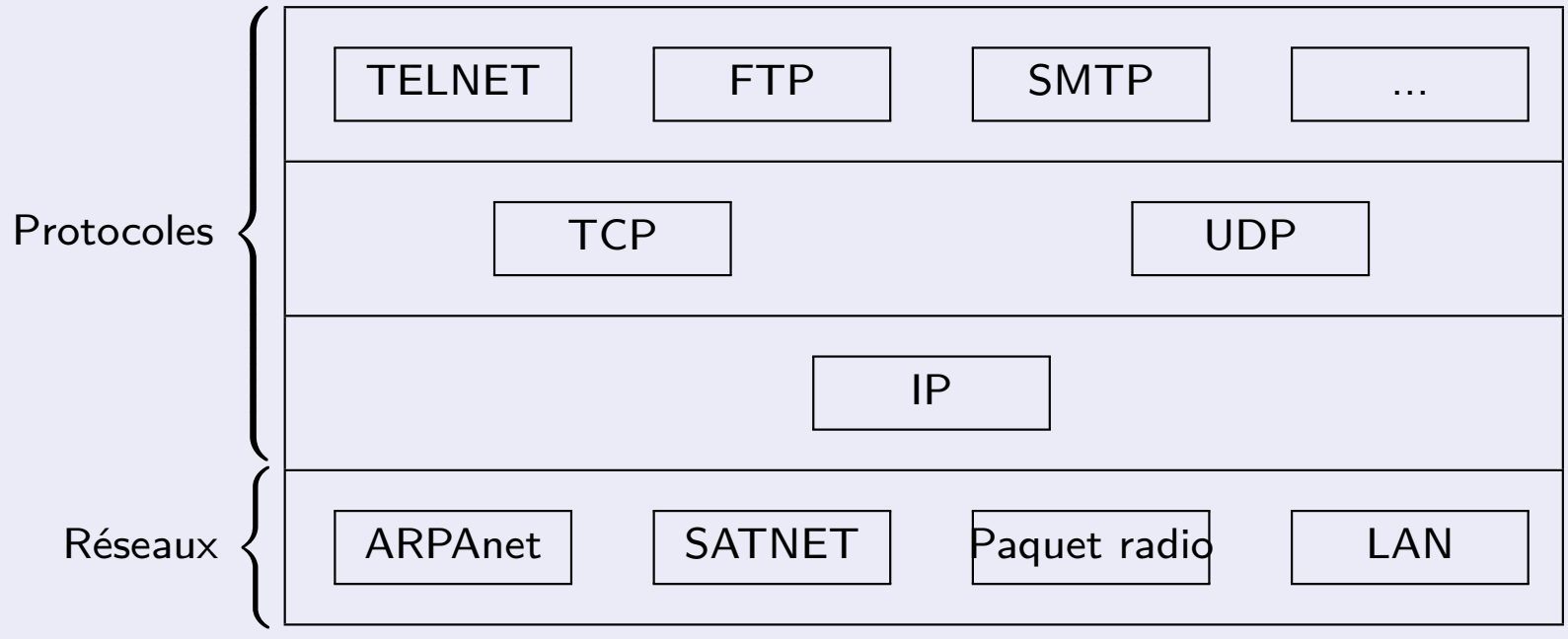
# Le modèle TCP/IP

## Caractéristiques principales

- La couche internet (pour inter-réseau) est une couche d'interconnexion sans connexion. Le réseau est un réseau à commutation de paquets, la couche internet se charge d'acheminer les paquets indépendamment des autres.
- La couche transport permet à deux entités de mener une conversation (comme pour la couche OSI du même nom). Mais contrairement à OSI, cette couche supporte le mode avec connexions et le mode sans connexions.
- La couche application se trouve directement au-dessus de la couche transport.
- La couche hôte-réseau est très flou dans le modèle TCP/IP.

# Le modèle TCP/IP

## Principaux protocoles



# Plan

# La couche application

## Objectifs

La couche application est la dernière couche du modèle OSI (couche 7). C'est la couche où se trouve toutes les applications.
Les couches précédentes se « contentent » de fournir un transport fiable des données.

## Protocoles

Les applications de cette couche s'appuient sur des protocoles.
Nous allons en présenter un : le protocole DNS.
Nous présenterons également une application et les protocoles sous-jacents : la messagerie électronique.

# Plan

# Messagerie électronique

## Architecture

Les systèmes de messagerie électronique (ou courrier électronique) sont généralement constitués de deux sous-systèmes :

- les agents utilisateurs qui permettent aux utilisateurs de lire et d'envoyer des messages ;
- les agents de transfert de messages qui acheminent les messages de la source vers la destination.

# Messagerie électronique

## Services

Les systèmes de messagerie électronique offrent généralement les services suivants :

- composition ;
- acheminement ;
- notification d'envoi (remis, rejeté ou perdu) ;
- affichage ;
- stockage.

De nos jours les systèmes de messagerie offrent également de nombreux autres services (stockage des messages dans des boîtes aux lettres, liste de diffusion, chiffrement, ...).

# Messagerie électronique

## Messages

Les messages sont constitués de plusieurs parties :

- l'enveloppe encapsule le contenu du message. Elle contient les informations utiles au transfert du message (adresse, priorité, ...).
- le contenu du message se compose de deux parties :
  - l'en-tête contient les informations de contrôle pour les agents utilisateurs,
  - le corps se destine au destinataire du message.

# Agent utilisateur

## Les lecteurs de courrier

Nous ne nous attarderons pas sur les agents utilisateurs car vous en avez forcément utilisé à un moment ou un autre (Thunderbird, Outlook, webmail, ...). Ils permettent :

- l'envoi de message ;
- la lecture des messages ;
- et bien d'autres choses (liste de diffusion, carnet d'adresse, correction orthographique, ...).

# Format de message

## RFC 822

Un message se compose :

- d'une enveloppe (RFC 821) ;
- d'un certain nombre de champs d'en-tête ;
- d'une ligne vierge ;
- d'un corps.

# RFCs

## RFC

- Une RFC (*Request For Comments*) est un rapport technique qui est réalisé lorsque le besoin d'une norme se fait sentir.
- Ces documents, réalisés à l'origine par l'IAB (*Internet Activities Board*, puis *Internet Architecture Board*, un comité d'ARPAnet), sont désormais rédigés par des experts techniques puis soumis à l'IETF (*Internet Engineering Task Force*).
- Certaines de ces RFC sont devenues des normes après le processus suivant :

$$\begin{aligned} \textit{RFC} &\longrightarrow \textit{Internet Draft} \\ &\longrightarrow \textit{Proposed Standard} \\ &\longrightarrow \textit{Draft Standard} \\ &\longrightarrow \textit{Internet Standard} \end{aligned}$$

# En-tête

## Champs d'en-tête

Chaque champ d'en-tête se compose (logiquement) d'une seule ligne de texte ASCII de la forme

*nom-du-champ*: *valeur*

- La RFC 822 ne fait pas de distinction claire entre les champs de l'enveloppe et ceux de l'en-tête.
- L'agent utilisateur construit le message qui sera transmis à l'agent de transfert qui utilisera certains des champs de l'en-tête pour construire l'enveloppe.

# Quelques champs d'en-tête relatifs au transport

| En-tête | Description |
|---|---|
| To : | Adresse des destinataires principaux |
| Cc : | Adresse des destinataires secondaires |
| Bcc : | Adresse des destinataires de copie cachée |
| From : | Adresse de messagerie de l'auteur |
| Sender : | Adresse de messagerie de l'emetteur |
| Received : | Ligne ajoutée par chaque agent de transfert le long de l'itinéraire |
| Return-path : | Peut être utilisé pour indiquer un chemin de retour jusqu'à l'émetteur |

# Quelques champs d'en-tête utilisés par les utilisateurs

| En-tête | Description |
|---|---|
| Date : | La date et l'heure à laquelle le message a été envoyé |
| Reply-To : | L'adresse à laquelle les réponses doivent être expédiées |
| Message-Id : | Numéro de référence unique identifiant le message |
| In-Reply-To : | Identifiant du message auquel celui-ci répond |
| References : | D'autres identifiants de messages pertinents |
| Keywords : | Mots clés choisis par l'utilisateur |
| Subject : | Bref résumé du message pour un affichage sur une ligne |
| X-* | en-tête à usage privé |

# MIME

**Exemple**

Un exemple de message minimal est ici

# MIME

## Problème des messages non ASCII

La RFC ne gère que les messages écrits en ASCII. On a donc des problèmes pour gérer :

- des messages écrits dans des langues contenant des accents (français ou allemand, par exemple) ;
- des messages écrits dans un alphabet non latin (l'arabe ou le russe, par exemple) ;
- des messages écrits sans alphabets (les idéogrammes chinois et japonais, par exemple) ;
- des messages ne contenant pas de texte (du son ou des images, par exemple).

# MIME

## RFC 1341 et RFC 2045 à 2049

Pour palier à ces problèmes une solution a été proposée : il s'agit de MIME (*Multipurpose Internet Mail Extension*). L'idée est d'ajouter une structure au corps du message et de définir des règles de codage pour les messages non ASCII.

## Nouveaux en-têtes MIME

| **En-tête** | **Description** |
|---|---|
| MIME-Version : | Version du format MIME utilisé |
| Content-Description : | Description du contenu |
| Content-Id : | Identifiant unique |
| Content-Tranfer-Encoding : | Méthode d'encapsulation du corps pour la transmission |
| Content-Type : | Type et format du contenu |

# MIME

## Encodage

Il existe cinq types d'encodage :

- ASCII : le plus simple, codage sur 7 bits, la ligne ne doit pas excéder 1000 caractères ;
- Idem mais sur 8 bits (violation du protocole) ;
- Idem sur 8 bits mais sans limitation de longueur de ligne (codage binaire, viole également le protocole) ;
- base64 : le principe est de diviser chaque groupe de 24 bits en quatre unités de 6 bits envoyées sous la forme d'un caractère. 'A' pour 0, 'B' pour 1, *etc.* La table complète peut être consultée dans la RFC 3548 ;
- quoted-printable : codage sur 7 bits où les caractères supérieurs à 127 sont codés d'un signe = suivi par la valeur du caractère sous la forme de deux chiffres hexadécimaux.

# MIME

## Exemple

- Un exemple de message minimal généré par Thunderbird utilisant des en-têtes MIME est ici
- Un autre exemple créé par le webmail de l'université est ici
- Un exemple de message encodé en base 64 sera présenté plus loin lorsque nous aborderons les messages composés de plusieurs parties.

# MIME

## Types et sous-types de contenu

Le champ *Content-Type* est composé d'un type et d'un sous-type séparés par le caractère '/'. Voici quelques types et sous-types :

| Type | Sous-type | Description |
|---|---|---|
| text | plain | Texte non formaté |
| | enriched | Texte avec des commandes de formatage simple |
| | html | Texte HTML (RFC 2854) |
| | xml | Document XML (RFC 3023) |
| image | gif | Image GIF |
| | jpeg | Image JPEG |
| audio | basic | Son |
| video | mpeg | Vidéo au format MPEG |
| application | octet-stream | Séquence d'octets |
| | postscript | Document PostScript |

# MIME

## Types et sous-types de contenu

| Type | Sous-type | Description |
|---|---|---|
| message | rfc822 | Message MIME RFC 822 |
| | partial | Message découpé pour la transmission |
| | external-body | le message lui-même est à récupérer *via* l'internet |
| multipart | mixed | Parties indépendantes |
| | alternative | Message identique en différents formats |
| | parallel | Parties à visualiser simultanément |
| | digest | Chaque partie est un message RFC 822 complet. |

# MIME

**Exemple**

Des exemples de message composés de plusieurs parties sont ici, ici et ici.

# Le transfert des messages

## Protocole SMTP

- L'envoi du courrier passe par l'établissement d'une connexion TCP par la machine source sur le port 587 (ou 25, ou 465) du serveur de courrier de l'utilisateur. Le démon à l'écoute du port 587 comprend le protocole SMTP (*Simple Mail Transfer Protocol*).
- Le serveur de courrier de l'utilisateur transmettra le message par SMTP au serveur de courrier du destinataire ou éventuellement à un serveur de courrier intermédiaire (*via* le port 25).
- Le serveur de courrier du destinataire stockera le message dans la boîte mail du destinataire.

# Le transfert des messages

## Protocole SMTP

SMTP est un simple protocole ASCII. Après l'établissement de la connexion :

- le serveur envoie un message indiquant son identité et s'il est prêt à recevoir du courrier ;
- si le serveur ne peut recevoir de courrier, le client essaie plus tard ;
- sinon, suit un échange client/serveur où le client :
  - dit bonjour ;
  - indique qui envoie le message ;
  - à qui il est adressé ;
  - envoie le message.

# Un exemple

```
220 osiris.univ-rouen.fr ESMTP
HELO univ-rouen.fr
250 osiris.univ-rouen.fr
MAIL FROM: <yannick.guesnet@univ-rouen.fr>
250 2.1.0 Ok
RCPT TO: <yannick.guesnet@laposte.net>
250 2.1.5 Ok
DATA
354 End data with <CR><LF>.<CR><LF>
From: yannick.guesnet@univ-rouen.fr
To: yannick.guesnet@laposte.net
MIME-Version: 1.0
Message-Id: <0704760941.AA00747@univ-rouen.fr>
Subject: Exemple
```

# Un exemple (suite)

```
Content-Type: multipart/alternative;
  boundary=azertyuiopqsdfghjklm
Preambule ignore

--azertyuiopqsdfghjklm
Content-Type: text/enriched

<bold>bonjour</bold>

--azertyuiopqsdfghjklm
Content-Type: text/html

<!DOCTYPE HTML PUBLIC
  "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
```

# Un exemple (suite)

```
  <head>
    <meta content="text/html; charset=ISO-8859-1"
      http-equiv="Content-Type">
    </head>
  <body bgcolor="#ffffff" text="#000000">
    <b>Bonjour</b><br>
  </body>
</html>

--azertyuiopqsdfghjklm--
.
250 2.0.0 Ok: queued as C26DF36C6A7
QUIT
221 2.0.0 Bye
```

# Un exemple par telnet

## Exemple

- Comme le protocole SMTP est un protocole texte, on pourrait aussi utiliser *telnet* (ou *netcat*) pour envoyer nos messages. Voir l'exemple
- Le mail reçu est alors : mail

## Remarque

Les serveurs utilisent les enregistrements *MX* des enregistrements DNS pour acheminer le courrier (voir plus loin).

# Le transfert des messages

## Protocole ESMTP

On peut également utiliser le protocole ESMTP (*Extended SMTP*) qui est une extension au protocole SMTP. Il faudra alors envoyer EHLO au lieu de HELO pour dire bonjour.

# Remise de message

## Récupérer ses messages

La plupart du temps les messages sont stockés dans une boîte mail située sur une machine du FAI. L'utilisateur doit donc pouvoir les récupérer sur sa machine.
Pour cela on dispose de deux protocoles : POP3 et IMAP.

# Remise de message

## POP3

POP3 (*Post Office Protocole* version 3) est décrit dans la RFC 1939.

## Fonctionnement

Pour utiliser le protocole POP3, on doit établir une connexion sur le port 110 du serveur. Il y a alors trois phases :

1. autorisation ;
2. transactions ;
3. mise à jour.

# POP3

## Autorisation

Durant la phase de connexion, le client envoie son nom d'utilisateur et son mot de passe. Une fois authentifié, il peut lister les messages à l'aide de la commande *LIST*.

## Transactions

Le client peut alors extraire les messages à l'aide de la commande *RETR* et les marquer pour suppression avec la commande *DELE*.

## Mise à jour

Le client émet enfin la commande *QUIT* pour passer en mode mise à jour. Lorsque le serveur a supprimé tous les messages, il envoie une réponse et met fin à la connexion.

# Un exemple

## Exemple

```
+OK hermes.univ-rouen.fr server ready
APOP guesnet kzehfzu7678678jhezjhf87Y8kzdj76d
+OK guesnet's maildrop has 1 message (120 octets)
LIST
+OK 1 message (120 octets)
1 120
.
RETR 1
+OK 120 octets
<le serveur POP3 envoie le message 1>
.
DELE 1
+OK message 1 deleted
QUIT
+OK Cyrus POP3 server signing off (maildrop empty)
```

# Un exemple

**Exemple**

Un autre exemple est disponible ici.

# Remise de message

## IMAP

IMAP (*Internet Message Access Protocol*) est décrit dans la RFC 2060.

## Fonctionnement

- IMAP offre plus de fonctionnalités que POP3. Il part du principe que le courrier reste sur le serveur indéfiniment.
- Il offre des mécanismes étendus pour lire des messages et même des parties de messages. Il permet également de gérer plusieurs boîtes aux lettres sur le serveur.
- Un serveur IMAP est en écoute sur le port 143.

# Plan

# Le DNS

## Contexte

- Les équipements d'un réseau sont accessibles à partir de leur adresse réseau.
- Cette adresse est une adresse numérique difficile à retenir et qui peut être amenée à changer.
- Aussi, il a été introduit un autre système de référence des machines à l'aide de chaînes ASCII.

## Objectifs

Le rôle principal du DNS (*Domain Name System*) est de mettre en correspondance les noms d'hôtes et leurs adresses IP.

# Le DNS

## Exemple

$$\texttt{www.univ-rouen.fr} \Rightarrow \texttt{193.52.144.34}$$

En fait, il s'agit d'un alias :

$$\texttt{193.52.144.34} \Leftrightarrow \begin{cases} \texttt{rp.univ-rouen.fr (CNAME)} \\ \texttt{www.univ-rouen.fr} \end{cases}$$

# Espaces de noms

## Principes du DNS

- Le DNS repose sur un schéma de nommage hiérarchique fondé sur la notion de domaine.
- Le DNS utilise une base de données répartie implémentant ce schéma de nommage.
- L'ICANN (Internet Corporation for Assigned Names and Numbers) est chargée de coordonner l'utilisation de cette base de données.
- En France, l'AFNIC (Association Française pour le Nommage Internet en Coopération) gère le `.fr`

# Espaces de noms

## La hiérarchie du DNS

- DNS est réparti en domaines de premier niveau (les TLD : *Top Level Domain*).
- Les TLD sont de sept types :
  - Un domaine pour la gestion de l'infrastructure de l'internet (`arpa`)
  - Un domaine pour les organisations intergouvernementales (`int`)
  - Génériques (gTLD), plus de 1200 en mars 2018 (`com`, `org`, `net`, dots)
  - Génériques restreints (grTLD : `biz`, `name`, `pro`)
  - Sponsorisés (sTLD : `aero`, `asia`, `gov`, …)
  - Nationaux (ccTLD), environ 260 (`fr`, `us`, `jp`, …)
  - TLD de test (tTLD, sous `test`, n'apparaissent pas sous la racine)

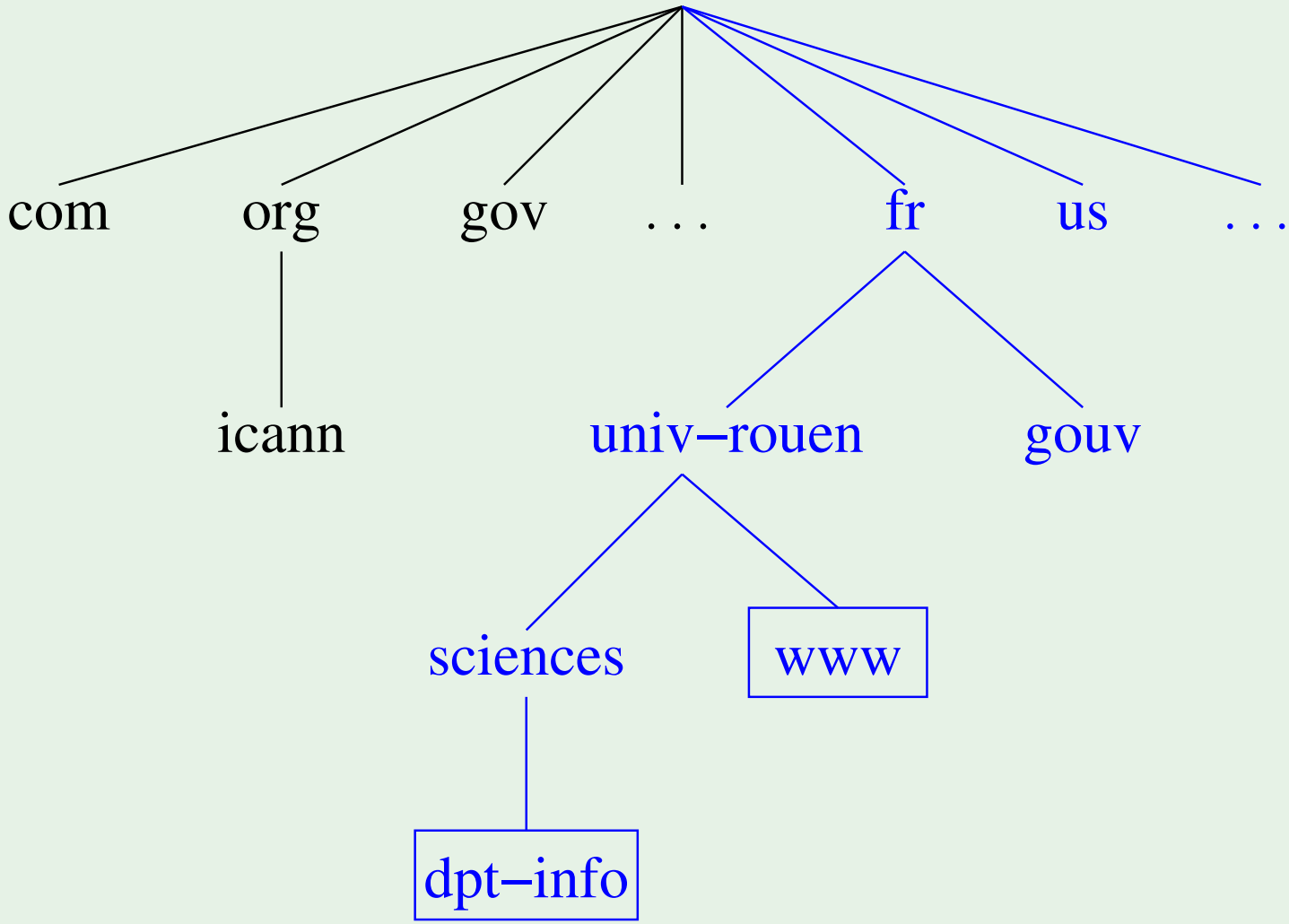
# Espaces de noms

## La hiérarchie du DNS

- Les TLD sont divisés en sous-domaines qui peuvent à leur tour être divisés en sous-domaine, *etc*.
- Les feuilles de l'arbre représentent un hôte ou une entité du réseau.

# Le DNS

## Exemple : `dpt-info.sciences.univ-rouen.fr`



# Le DNS

## Les noms de domaines

- Un nom de domaine est formé de plusieurs composants séparés par un point en progressant vers la racine (sans nom).
- Un nom de domaine peut être absolu ou relatif. Un nom absolu se termine par un point.
- Les noms de domaines sont insensibles à la casse.
- Chaque composant peut atteindre 63 caractères et le nom complet ne doit pas dépasser 255 caractères.
- En réalité, le DNS est un graphe et non un arbre : `ma_societe.fr` et `ma_societe.com` peuvent faire référence au même sous-domaine.

# Le DNS

## Exemple de résolution de noms

- Avec *host* :

```
[Nannick@localhost ~]$ host www.univ-rouen.fr.
www.univ-rouen.fr is an alias for wattrelos.univ-rouen.fr.
wattrelos.univ-rouen.fr has address 10.0.128.44
```

- Avec *nslookup* :

```
[Nannick@localhost ~]$ nslookup www
Server: 10.0.130.11
Address: 10.0.130.11#53

www.univ-rouen.fr canonical name = wattrelos.univ-rouen.fr.
Name: wattrelos.univ-rouen.fr
Address: 10.0.128.44
```

# Les ressources

## Enregistrements de ressources

On associe à chaque domaine un ensemble d'enregistrements de ressources (*resource records*). Lorsqu'on communique un nom de domaine à un serveur DNS, on reçoit en retour des enregistrements de ressources associés au nom.

## Structure d'un enregistrement

Un enregistrement se compose de cinq éléments :

- Nom de domaine : domaine auquel s'applique cet enregistrement.
- Durée de vie : indication sur la stabilité de l'enregistrement (en seconde, par exemple 86400 ou 60).
- Classe : `IN` pour les informations concernant internet.
- Type et valeur : le type de l'enregistrement ainsi que la valeur associée..

# Les différents type d'enregistrements

## Principaux types d'enregistrement (IPv4)

| Type | Valeur |
|---|---|
| `SOA` | Informations sur le serveur primaire de la zone du serveur de noms |
| `A` | Adresse IP (entier de 32 bits) d'un hôte |
| `MX` | Hôte prenant le courrier électronique |
| `NS` | Nom d'un serveur de nom |
| `CNAME` | Nom de l'hôte correspondant à l'alias |
| `PTR` | Adresse IP (pour les recherches inverses, *reverse lookup*) |

# Exemples avec dig

## Interrogation du DNS avec dig

```
[user@host ~]$ dig www.univ-rouen.fr

; <<>> DiG 9.7.1 <<>> www.univ-rouen.fr
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 57687
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 2,
                                AUTHORITY: 1, ADDITIONAL: 1

;; QUESTION SECTION:
;www.univ-rouen.fr.              IN      A

;; ANSWER SECTION:
www.univ-rouen.fr.      3600    IN      CNAME   rp.univ-rouen.fr.
rp.univ-rouen.fr.       3600    IN      A       193.52.144.34

...
```

# Exemples avec dig

## Interrogation du DNS avec dig

```
[user@host ~]$ dig -x 193.52.144.34

<<>> DiG 9.7.1 <<>> -x 193.52.144.34
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 25399
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1,
                                AUTHORITY: 1, ADDITIONAL: 1

;; QUESTION SECTION:
;34.144.52.193.in-addr.arpa.      IN      PTR

;; ANSWER SECTION:
34.144.52.193.in-addr.arpa. 3600 IN      PTR     rp.univ-rouen.fr.

...
```

# Exemples avec dig

## Interrogation du DNS avec dig

```
[user@host ~]$ dig MX univ-rouen.fr
; <<>> DiG 9.7.1 <<>> MX univ-rouen.fr
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 33427
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 2,
                                AUTHORITY: 1, ADDITIONAL: 2

;; QUESTION SECTION:
;univ-rouen.fr.          IN   MX

;; ANSWER SECTION:
univ-rouen.fr.   3600   IN   MX   100 cata-smtp.univ-rouen.fr.
univ-rouen.fr.   3600   IN   MX   10 osiris.univ-rouen.fr.

...
```

# Exemples avec dig

## Interrogation du DNS avec dig

```
[user@host ~]$ dig MX univ-rouen.fr

; <<>> DiG 9.10.4 <<>> MX univ-rouen.fr
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 24757
;; flags: qr rd ra; QUERY: 1, ANSWER: 4,
                                    AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 1460
;; QUESTION SECTION:
;univ-rouen.fr.          IN   MX

;; ANSWER SECTION:
univ-rouen.fr.   3600   IN   MX   10 mxc.relay.renater.fr.
univ-rouen.fr.   3600   IN   MX   10 mxd.relay.renater.fr.
univ-rouen.fr.   3600   IN   MX   10 mxb.relay.renater.fr.
univ-rouen.fr.   3600   IN   MX   10 mxa.relay.renater.fr.
```

# Exemples avec `dig`

## Interrogation du DNS avec `dig`

On peut voir l'échange de données *via* `wireshark` : capture

# Les serveurs de noms

## Zones

L'espace de noms DNS est divisé en zones distinctes.
Chaque zone contient une partie de l'arbre et des serveurs de noms relatifs à la zone.

## Les serveurs de noms d'une zone

En général, une zone contient un serveur de noms principal (serveur primaire) qui obtient les informations à partir de son disque dur et des serveurs secondaires qui obtiennent leurs informations à partir du serveur primaire.

# Les serveurs de noms

## Les serveurs racines

- Il existe également des serveurs racine (une dizaine) et chacun d'eux connait l'adresse de tous les serveurs de domaine de premier niveau.
- Une machine connaissant l'adresse d'un serveur racine peut effectuer n'importe quelle recherche.
- Au démarrage d'un serveur DNS, celui-ci charge dans son cache les adresses des serveurs racines à partir d'un fichier de configuration.

# Requêtes récursive

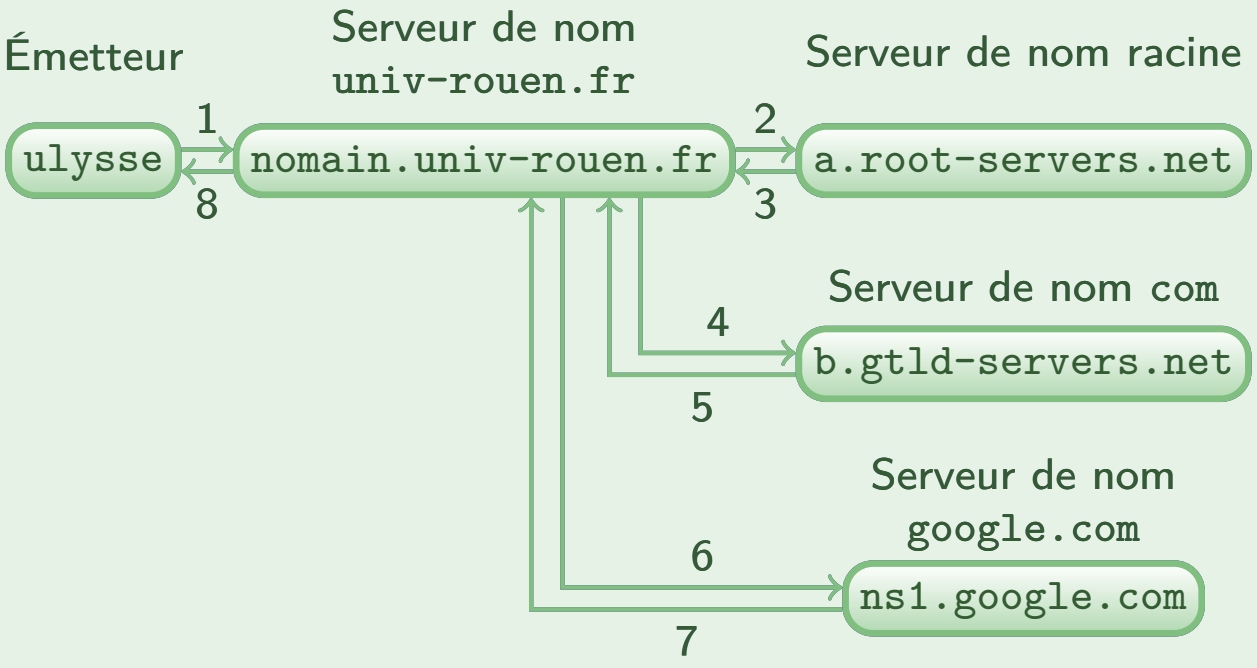
## Résolution d'un nom

Lorsqu'un solveur (par exemple `dig`) reçoit une requête :

- il la transmet à l'un des serveurs de noms locaux (*/etc/resolv.conf*) ;
- si le domaine tombe dans la juridiction du serveur alors il retourne les enregistrements de ressources officiels (`autoritative records`) ;
- si le serveur possède l'information dans son cache alors il retourne les enregistrements correspondants ;
- sinon, il envoie un message de requête au serveur de premier niveau du domaine concerné ;
- lorsqu'il reçoit la réponse, il la met dans son cache pour la durée de vie indiquée par le champ de l'enregistrement.

# Requêtes récursive

## Résolution d'un nom : www.google.com



# Résolution d'un nom

## Requête itérative

- Sur l'exemple précédent le serveur `nomain.univ-rouen.fr` réalise une requête récursive (il fournit la réponse complète).
- On aurait également pu lui demander de réaliser une requête itérative (comme le font les serveurs des domaines `com` et `google.com` : ils fournissent la meilleure réponse possible).

## DNS *forwarders*

L'exemple précédent est simplifié : dans la réalité les serveurs DNS de l'université utilisent des « DNS forwarders » pour essayer de résoudre les adresses extérieures au domaine `univ-rouen` avant de s'adresser aux serveurs de premiers niveaux.

# Utilisation de l'API C

## Résolution DNS

La recherche d'une adresse IP à partir d'un nom de machine et *vice versa* se fera à partir des fonctions :

```
int getaddrinfo(const char *node, const char *service,
                const struct addrinfo *hints,
                struct addrinfo **res);
int getnameinfo(const struct sockaddr *sa, socklen_t salen,
                char *host, size_t hostlen,
                char *serv, size_t servlen, int flags);
```

# Utilisation de l'API C

## Exemples

Nous attendrons le chapitre sur la couche transport pour détailler l'utilisation des fonctions précédentes. Nous nous contentons pour l'instant de donner deux exemples d'utilisation de ces fonctions.

- Résolution d'une adresse IP à partir d'un nom de machine (`getaddrinfo`)
- Résolution d'un nom de machine à partir d'une adresse IP (`getnameinfo`)

# Plan

# La couche transport

## Principes

- La couche transport fournit à l'utilisateur (processus de la couche application) un service de transport efficace, fiable et économique.
- La couche utilise des services mis à disposition par la couche réseau.
- Le logiciel et/ou le matériel qui assure cette fonction est appelé l'entité de transport.
- L'entité de transport peut se trouver dans la carte réseau, dans le noyau, dans une bibliothèque d'application ou encore dans un processus utilisateur.

# La couche transport

## Services

- La couche transport propose deux types de service : avec ou sans connexion.
- Les connexions passent par trois phases : établissement, transfert et libération.
- La couche transport permet de masquer les disparités qui peuvent exister entre les différents types de services réseau en proposant une uniformisation des procédures à l'utilisateur.

# La couche transport

## Fournisseurs et utilisateurs

- On appelle les quatre niveaux inférieurs du modèle OSI les fournisseurs de la couche transport
- Les couches supérieures sont appelées les utilisatrices de la couche transport.

# La couche transport

## Interface

- Pour permettre aux utilisateurs d'accéder au service transport, la couche transport doit en fournir une interface.
- Chaque service transport possède sa propre interface. Nous présenterons ici l'interface des sockets de Berkeley.

# Plan

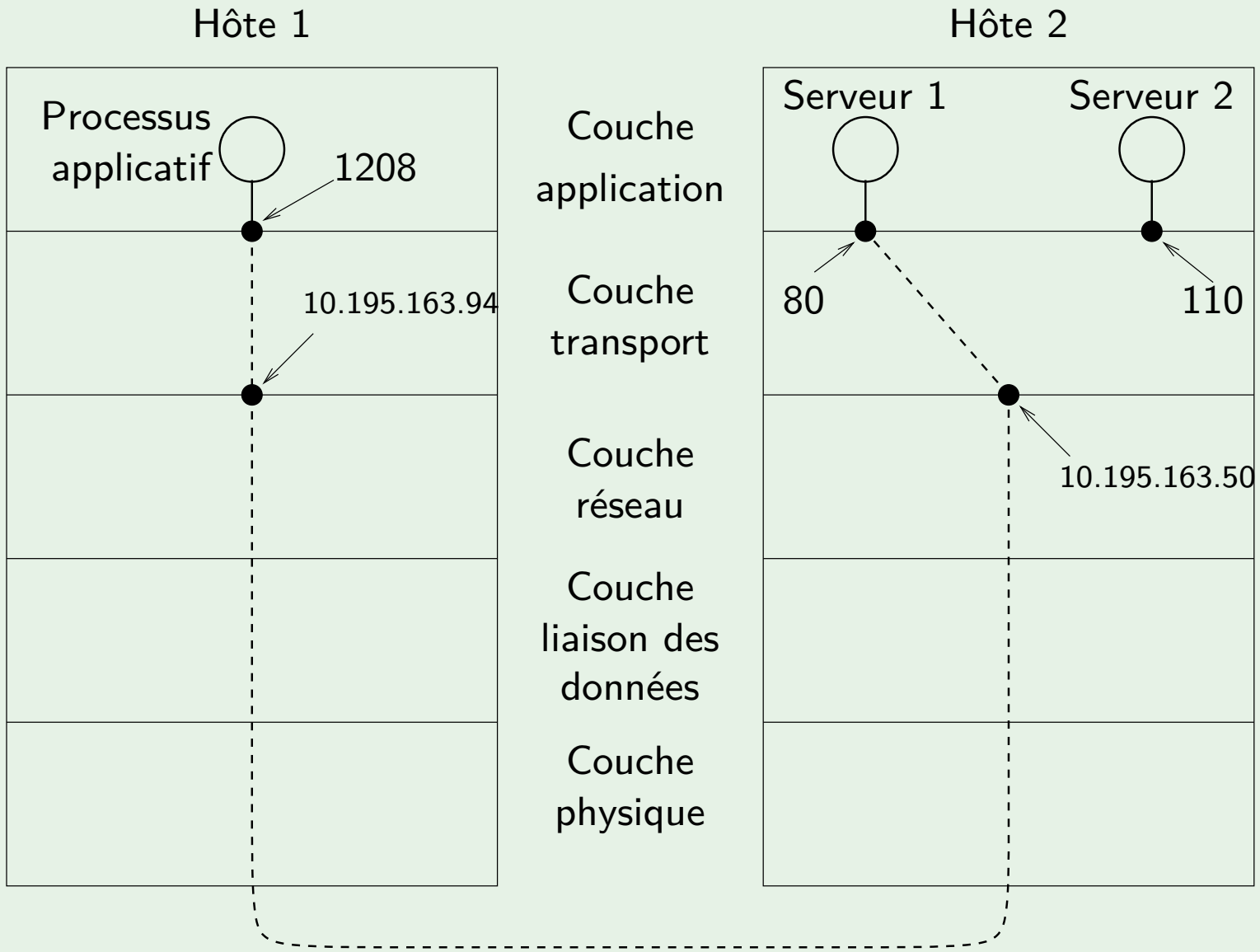
# Adressage

## Points terminaux

- Lorsqu'un processus veut établir une connexion ou envoyer des données sans connexion, il doit spécifier à qui il veut s'adresser.
- La méthode normale consiste à définir des adresses de transport pour lesquels les processus peuvent être à l'écoute d'éventuelles demandes de connexion.
- Nous appellerons ces point terminaux des ports comme c'est le cas sur internet. Plus généralement on parle de TSAP (*Transport Service Access Point*).
- Dans la couche réseau il existe également des points terminaux : ce sont, par exemple, les adresses IP. On parle plus généralement de NSAP.

# Adressage

## Schémas



# Adressage

## Communication des ports

L'un des problèmes soulevé est : comment connaître le port sur lequel écoute un serveur ?

- Une solution est apportée par les serveurs offrant des services connus. On peut alors dresser la liste de tous les services connus (comme dans le fichier `/etc/services` par exemple).
- On peut aussi utiliser un serveur de noms qui permet d'obtenir le TSAP à partir du nom du service souhaité.

# Adressage

**Exemple**

Voici un exemple de recherche d'un numéro de port à partir d'un nom de service qui utilise la fonction« fourre-tout »`getaddrinfo` : voir le source.

# Adressage

**Remarque**

Il existe des « super-serveurs »(on parlera plutôt de serveur de processus) qui écoutent un ensemble de ports et lancent les serveurs à la demande (comme *xinetd* ou *systemd*).

# Plan

# Les sockets

## Principes

- Les sockets (« prise », parfois traduit par point de communication) sont apparues dans BSD 4.2 (*Berkeley Software Distribution*) en 1983.
- Après initialisation, les sockets se manipulent à l'aide d'un descripteur similaire aux descripteurs de fichiers.
- On pourra donc manipuler les sockets comme on manipule les fichiers (*cf* cours de système du premier semestre).
- La seule complication apparait donc lors de l'initialisation de la socket.

## Remarque

Sous UNIX, vous pouvez afficher les sockets présentes sur votre machine à l'aide de la commande `netstat -a`.

# Les sockets

## Schéma général d'un client

Pour communiquer à l'aide d'une socket un client suivra les étapes suivantes :

1. Création de la socket
2. Si besoin, connexion au serveur
3. Envoi/réception des messages
4. Fermeture de la socket

# Les sockets

## Création d'une socket

Le premier appel système à utiliser est l'appel *socket* défini dans `#include <sys/socket.h>` :

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

Cet appel crée une socket et renvoie le descripteur associé.

## Paramètre *domain*

Le paramètre *domain* indique le domaine de communication, cela influe sur la famille de protocoles qu'on peut employer. Dans ce cours, nous aborderons les domaines suivants (mais il en existe d'autres) :

- *AF_UNIX* pour la communication locale ;
- *AF_INET* et *AF_INET6* pour les protocoles IPv4 et IPv6.

# Les sockets

## Paramètre *type*

Le paramètre *type* indique le type de la socket, parmi ceux-ci nous utiliserons :

- *SOCK_STREAM* mode connecté ;
- *SOCK_DGRAM* mode sans connexion ;

# Les sockets

## Paramètre *protocol*

Le paramètre *protocol* indique le protocole à utiliser pour la communication. Par exemple :

| Domain | Type | Protocole |
|---|---|---|
| AF_INET | SOCK_STREAM | IPPROTO_TCP |
| AF_INET | SOCK_DGRAM | IPPROTO_UDP |

- Généralement il n'y a qu'un seul protocole par paire domaine/type mais rien n'interdit qu'il en existe plusieurs.
- On peut indiquer une valeur nulle pour utiliser le protocole par défaut.

# Les sockets

**Exemple**

Voici un exemple de création de socket internet en mode non connecté (protocole UDP) : source.

**Remarque**

Si on utilise la commande `netstat -a --udp` on ne voit pas la socket apparaître dans la liste. Pourquoi ?

# Les sockets

## Appel système `socket`

En fait l'appel système `socket` se contente de réserver un numéro de descripteur pour la socket (ou $-1$ en cas d'échec). Il n'y a aucun dialogue réseau ni échange d'information avec le noyau.

## Adressage

Arrivé à ce point, si on veut communiquer avec les sockets il faut être capable de leur affecter des adresses (couche réseau et transport). Et c'est là que les choses se compliquent...

# Les sockets

## Adresses génériques

Quel que soit le domaine de la socket, son adresse est représentée par une structure générique `struct sockaddr` :

```
struct sockaddr {
  sa_family_t sa_family;
  char        sa_data[];
}
```

C'est cette structure qui sera utilisée dans les différents appels systèmes manipulant les sockets.

# Les sockets

## Adresses spécifiques

En réalité chaque domaine de communication définit sa propre structure d'adresse qui sera de la forme :

```
struct sockaddr_domain {
  sa_family_t sdomain_family;
  ... /* champs spécifiques au domaine */
}
```

# Les sockets

## Adresses génériques

Lorsqu'on ne connait pas le type d'adresse que l'on va manipuler, on peut utiliser la structure *sockaddr_storage* qui :

- est assez grande pour accueillir tout type d'adresse spécifique ;
- est alignée de façon à ce que tout pointeur sur celle-ci peut être converti en un pointeur spécifique quelconque ;
- contient le champ *sa_family_t ss_family*.

# Exemple d'adresse : les sockets Unix

## Adresses pour les sockets UNIX

Une adresse de socket de domaine UNIX peut-être un nom de fichier (il existe également des adresses abstraites). Elle est représentée par la structure *sockaddr_un* :

```
struct sockaddr_un {
  sa_family_t sun_family;
  char        sun_path[UNIX_PATH_MAX];
};
```

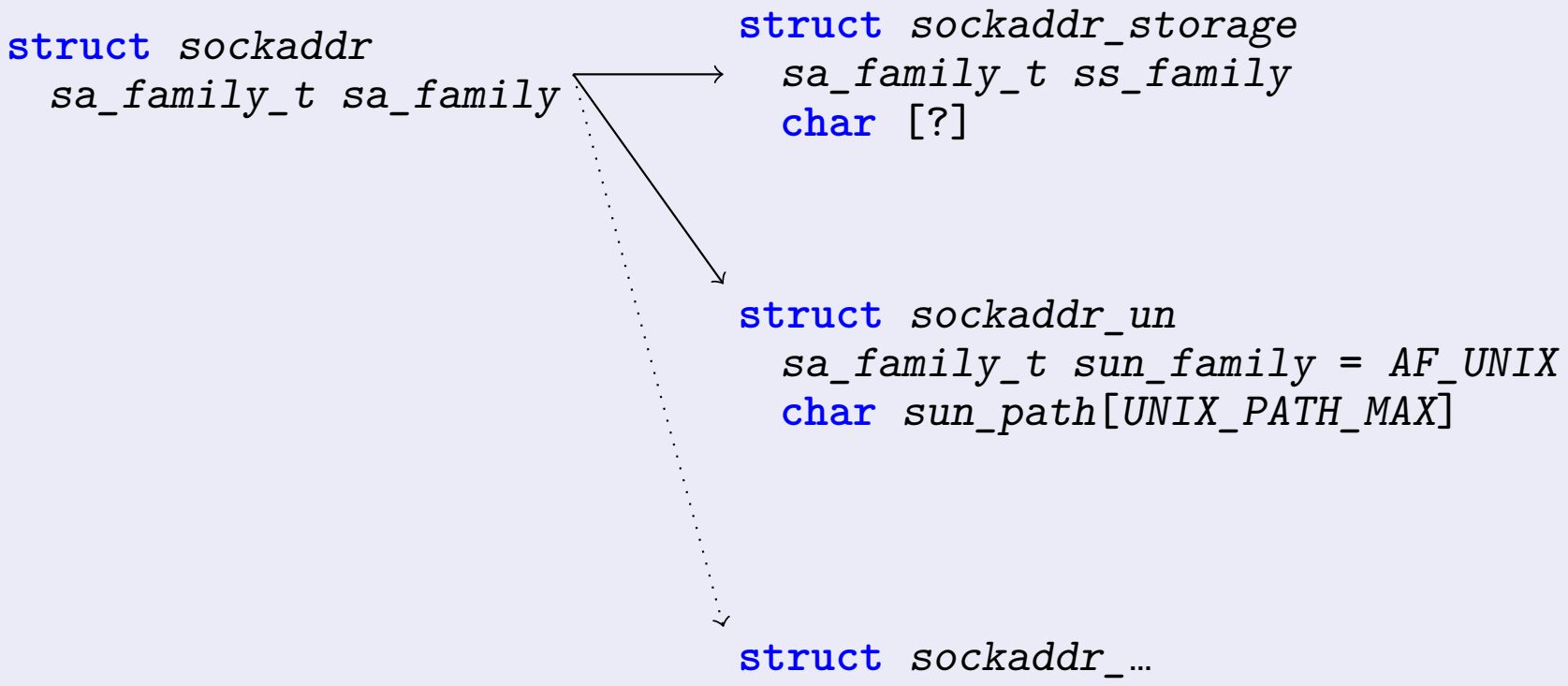
où *sun_family* vaut toujours *AF_UNIX* et *sun_path* est un chemin vers un fichier. Le fichier correspondant sera alors de type socket.

## Remarque

Les sockets UNIX sont un cas particulier où on n'utilise que la couche transport sans descendre dans la couche réseau.

# Les structures d'adresses

## Résumé



# Exemple d'adresse : les sockets Unix

**Exemple**

Un exemple de création d'adresse de socket UNIX : ici.

**Remarque**

Pour avoir une description des sockets UNIX *via* le `man`, consultez la page `unix` de la section 7 :

```
# man 7 unix
```

# Communiquer à travers des sockets : mode non connecté

## Appel système *sendto*

On peut envoyer des données à travers une socket à l'aide de *sendto* :

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

- *sockfd* est le descripteur de la socket.
- Les données à envoyer sont indiquées à l'aide de *buf* et de *len* qui indique la taille de *buf*.
- *flags* permet de spécifier des options d'envoi.
- L'adresse de destination est spécifiée grâce à *dest_addr* et *addrlen* qui contient la taille effective de la structure *addr*.
- *sendto* retourne le nombre d'octets envoyés ($-1$ en cas d'erreur).

## Un premier exemple : un client en mode non connecté (source)

```
int main(int argc, char* argv[]) {
  // Création de la socket
  int s;
  if ((s = socket(AF_UNIX, SOCK_DGRAM, 0)) == -1)
    // Erreur...

  // Adresse où envoyer le message
  struct sockaddr_un addr;
  memset(&addr, '\0', sizeof(struct sockaddr_un));
  addr.sun_family = AF_UNIX;
  strncpy(addr.sun_path, "my_socket_unix",
          sizeof(addr.sun_path) - 1);

  // Envoi du message
  if (sendto(s, "bonjour", 8, 0,
             (struct sockaddr*)&addr,
             sizeof(struct sockaddr_un )) == -1)
    // Erreur...

  return EXIT_SUCCESS;
}
```

# Les serveurs

## Schéma général d'un serveur en mode non connecté

Un serveur en mode non connecté suivra les étapes suivantes :

1. Création de la socket
2. Affectation d'une adresse à la socket
3. Réception/Envoi des messages
4. Fermeture de la socket

# Affectation d'adresse

## Appel système *bind*

On souhaite parfois affecter une adresse à notre socket (pour qu'on puisse nous contacter, c'est même obligatoire). Pour cela on utilise l'appel système *bind* :

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

# Affectation d'adresse

## Socket UNIX

- Pour les socket UNIX, l'adresse d'une socket est un chemin dans le système de fichier.
- L'appel système *bind* invoqué sur une socket UNIX crée un fichier de type socket.
- Il faudra alors penser à supprimer ce fichier lorsqu'on n'en aura plus besoin (*unlink*).

# Affectation d'adresse

## Exemple

Un exemple d'utilisation de *bind* pour les sockets UNIX : ici.

- On peut vérifier que la socket est bien créée à l'aide de la commande

```
netstat -a --unix|grep mon_adresse.socket
```

- Si on interrompt le programme (avec un *SIGINT*, par exemple) puis qu'on le relance on aura le message suivant :

```
bind: Address already in use
```

# Communiquer à travers des sockets : mode non connecté

## Appel système *recvfrom*

On peut recevoir des données à l'aide de l'appel-système *recvfrom* :

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

- Lit les données sur la socket *sockfd*. Ces données seront stockées dans le tampon *buf* de taille *len* (les données surnuméraires pouvant être perdues). Par défaut, le processus est placé en attente s'il n'y a pas de données à lire. *flags* permet de transmettre des options de réception.
- *recvfrom* retourne le nombre d'octets lus (-1 en cas d'erreur).
- Si *addr* est non *NULL* alors *addrlen* doit indiquer sa taille et alors *addr* contiendra en sortie l'adresse de l'emetteur (si elle est disponible) et *addrlen* la taille effectivement occupée.

## Un (tout petit) serveur en mode non connecté

```
int main(int argc, char* argv[]) {
  // Création de la socket
  int s;
  if ((s = socket(AF_UNIX, SOCK_DGRAM, 0)) == -1)
    // Erreur...

  // On affecte une adresse à notre socket
  struct sockaddr_un addr;
  memset(&addr, '\0', sizeof(struct sockaddr_un));
  addr.sun_family = AF_UNIX;
  strncpy(addr.sun_path, "my_socket_unix",
          sizeof(addr.sun_path) - 1);
  if (bind(s, (struct sockaddr*) &addr, sizeof(addr))
      == -1)
    // Erreur...
```

```
  // On reçoit un message
  char buf[BUF_SIZE];
  buf[BUF_SIZE - 1] = 0;
  struct sockaddr_un addr_from;
  socklen_t addr_len = sizeof(addr_from);
  if (recvfrom(s, buf, sizeof(buf) - 1, 0,
      (struct sockaddr*)&addr_from, &addr_len) == -1)
    // Erreur...

  // On affiche le message
  if (addr_len == 0)
    strcpy(addr_from.sun_path, "unnamed");
  printf("msgv received from %s: %s\n",
      addr_from.sun_path, buf);

  // On supprime la socket en quittant
  if (unlink("my_unix_socket") == -1)
    // Erreur...
  return EXIT_SUCCESS;
}
```

## Un (tout petit) serveur en mode non connecté

Le source est ici.

## Remarques

- Avec le client précédent, le serveur ne connait pas l'adresse de l'émetteur du message. On pourrait utiliser le client suivant afin que le serveur sache qui lui envoie le message : source.
- Que se passe-t-il si on utilise l'adresse serveur `"\0adresse.socket"` au lieu de `"adresse.socket"` (il s'agit d'une extension Linux) ?

# Le mode connecté

## Le mode connecté

Un serveur en mode connecté fera intervenir deux types de sockets :

- La socket d'écoute sera en attente des demandes de connexions de la part des clients.
- Les sockets de services correspondent chacune à une connexion d'un client avec le serveur. La communication entre les deux entités passe par cette socket.

# Le mode connecté

## Algorithme général d'un serveur en mode connecté

L'algorithme général d'un serveur en mode connecté ressemblera à celui-ci :

- Création de la socket (`socket`, déjà vu)
- Affectation d'une adresse (`bind`, déjà vu)
- Déclarer la socket comme acceptant des connexions (`listen`)
- Attendre des connexions (`accept`)
  - À chaque nouvelle connexion, traiter la connexion en créant éventuellement un nouveau thread ou un nouveau processus
  - Se remettre en attente d'une nouvelle connexion

# Déclarer une socket d'écoute

## L'appel système *listen*

On peut déclarer une socket comme acceptant des connexions (il s'agira alors de la socket d'écoute) grâce à l'appel système *listen* :

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

L'appel système *listen* marque la socket *sockfd* comme étant une socket d'écoute (on emploie aussi le terme de socket passive). *backlog* définit le nombre maximal de connexions qui peuvent être placées en attente de traitement.

# Déclarer une socket d'écoute

**Exemple**

- Un exemple de création de socket d'écoute est disponible ici.
- On peut vérifier que la socket est bien une socket d'écoute à l'aide de la commande `netstat -a`.
- On peut s'assurer de l'utilité du paramètre `backlog` en lançant plusieurs clients

# Attendre des connexions

## L'appel système *accept*

On peut placer le processus en attente de connexion grâce à l'appel système *accept* :

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

L'appel système extrait la première connexion en attente sur la socket d'écoute *sockfd*.
*addr* et *addrlen* ont la même signification que pour *recvfrom*.
Par défaut, s'il n'y a pas déjà de connexion en attente alors le processus est placé en attente passive de connexion.

# Attendre des connexions

## Socket de service

- La valeur de retour de `accept` est le descripteur d'une nouvelle socket (une socket de service) qui servira à traiter les échanges avec le client qui a initié la demande de connexion.
- La socket d'écoute reste disponible pour traiter les demandes de connexions qui vont suivre.

## Un (tout petit) serveur en mode connecté (source)

```
int main(int argc, char* argv[]) {
  // Création de la socket et affectation d'adresse :
  // idem que le mode non connecté
  int s;
  ...

  // Déclarer la socket comme une socket d'écoute
  if (listen(s, 10) == -1)
    // Erreur...
```

```
  // On récupère les signaux du type
  // SIGQUIT, SIGTERM, ...
  // afin de quitter proprement le serveur
  connect_signals();

  // Boucle d'attente de connexions
  int service; // descripteur de la socket de service
  while ((service = accept(s, NULL, NULL)) != -1) {
    process_connexion(service);
  }

  // Erreur ...
  return EXIT_FAILURE;
}
```

# Le mode connecté

## Le client en mode connecté

Pour qu'un client puisse se connecter, il devra utiliser l'appel système *connect* :

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

Cet appel connecte la socket *sockfd* à l'adresse indiquée par *addr* (de longueur *addrlen*).

## Manipulation des descripteurs

Une fois connectée, une socket (cliente ou de service) peut être manipulée à l'aide de son descripteur comme n'importe quel fichier.

## Un (petit) client en mode connecté (source)

```
int main(int argc, char* argv[]) {
  // Création de la socket : idem mode non connecté
  int s;
  ...

  // Connexion
  struct sockaddr_un addr;
  memset(&addr, '\0', sizeof(struct sockaddr_un));
  addr.sun_family = AF_UNIX;
  strncpy(addr.sun_path, "my_socket_unix",
          sizeof(addr.sun_path) - 1);
  if (connect(s, (struct sockaddr*) &addr,
      sizeof(addr)) == -1)
    // Erreur...

  // Envoi du message
  char buf[] = "bonjour";
  if (write(s, buf, 8) == -1)
    // Erreur...

  return EXIT_SUCCESS;
}
```

# Fin de vie d'une socket

## Fermeture d'une socket

- Les sockets peuvent être libérées à l'aide de l'appel système *close* commun à tous les descripteurs.
- Cependant, dans le cas d'une socket en mode connectée, il se peut que cette dernière ne soit pas libérée aussitôt car le protocole sous-jacent peut nécessiter des échanges entre le client et le serveur supplémentaires pour fermer la connexion afin d'avoir un transport fiable des données.
- Utiliser l'appel système *bind* sur une adresse qui appartenait à une socket qui vient d'être fermée peut donc renvoyer l'erreur *EADDRINUSE*.

# Fin de vie d'une socket

## Appel système *shutdown*

Les sockets étant bidirectionnelles (on parle de connexion full-duplex), on peut contrôler plus finement quel type d'utilisation de la socket on souhaite libérer à l'aide de l'appel *shutdown* :

```
int shutdown(int sockfd, int how);
```

Cette fonction termine tout ou partie de la connexion sur la socket *sockfd*. Si *how* vaut

- *SHUT_RD* alors la réception est désactivée ;
- *SHUT_WR* ce sera l'émission qui sera désactivée ;
- *SHUT_RDWR* alors la socket est totalement fermée.

# Fin de vie d'une socket

## Exemple d'utilisation de *shutdown*

On peut imaginer le cas d'utilisation suivant de *shutdown* :

- Le client envoie des données à un serveur qui ne connait pas la longueur des données qu'il va recevoir ;
- Une fois les données envoyées le client ferme la socket en écriture ;
- Lorsque le serveur détecte la fin de l'envoi des données (*read* retourne 0) il peut les traiter et envoyer la réponse ;
- Le client peut lire la réponse car sa socket n'est pas fermée en lecture.

# Plan

# La couche transport internet

## Protocoles

La couche transport de l'internet dispose de deux protocoles principaux :

- le protocole UDP qui est un protocole sans connexion ;
- le protocole TCP qui est un protocole orienté connexion.

# La couche transport internet

## Protocole UDP

Le protocole UDP (*User Datagram Protocol*) est un protocole de transport en mode non connecté.
UDP est défini dans la RFC 768.

## Ce que UDP ne fait pas

UDP ne gère pas :

- le contrôle de flux ;
- le contrôle d'erreur ;
- la retransmission après réception d'un segment erroné.

# Protocole UDP

## Ce qu'est UDP

UDP est une interface pour le protocole IP équipé d'une fonction de multiplexage exploitant les ports.

## À quoi ça sert ?

UDP est utilisé (entre autre) :

- pour les RPC (*Remote Procedure Call* ou appel de procédures à distance) ;
- pour les échanges en temps réel, en particulier le protocole RTP (*Real-Time Transport Protocol*) fonctionne sur UDP.

# Protocole UDP

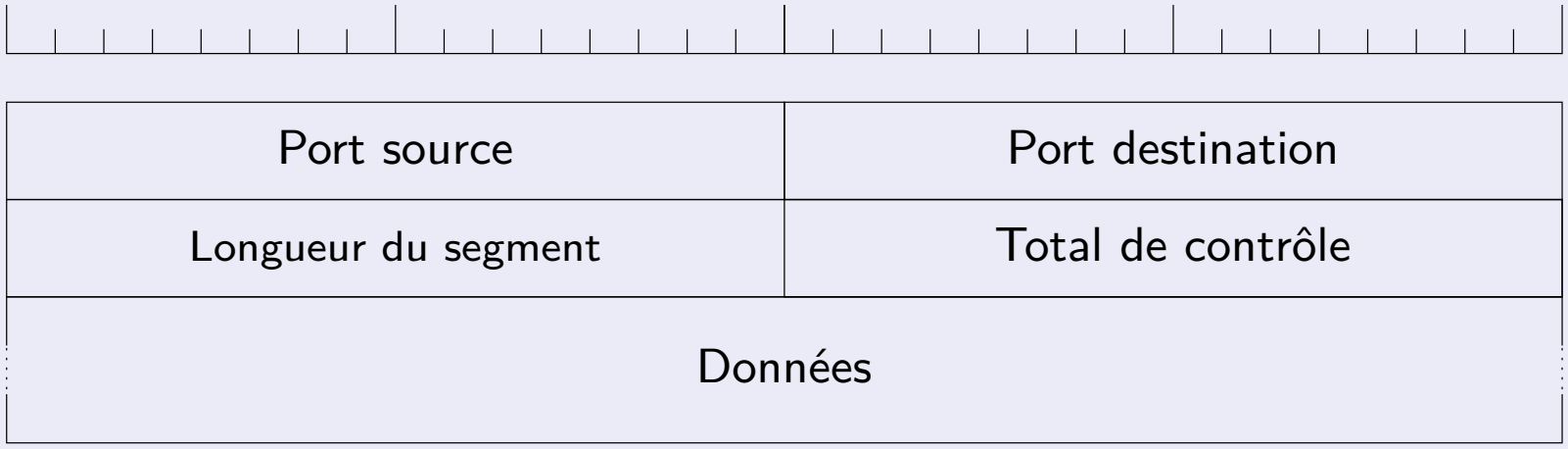
## Segment UDP

Un segment UDP comprend un en-tête de 8 octets suivi de la charge utile. L'en-tête comprend :

- Le port source (2 octets) ;
- Le port destination (2 octets) ;
- La longueur du segment en-tête comprise (2 octets) ;
- Une somme de contrôle permettant de contrôler l'intégrité du segment.

# Protocole UDP

## Segment UDP




## Remarque

La somme de contrôle prend aussi en compte une partie de l'en-tête IP, ce qui est une des raisons pour lesquelles le modèle TCP/IP ne suit pas parfaitement le modèle OSI.

## Exemple

Un exemple de datagramme est ici.

# La couche transport internet

## Protocole TCP

UDP ne permet pas d'assurer une remise séquentielle et fiable. Pour cela on utilise le protocole TCP (*Transmission Control Protocol*).

## Principes de TCP

- TCP est un protocole conçu pour traiter de bout en bout de manière fiable des données sur un ensemble de réseaux non fiables.
- TCP est conçu pour s'adapter dynamiquement aux variations des propriétés des composants constituant le réseau (topologie, largeur de bande, retard de transmission, taille maximale des paquets, ...).

# Le protocole TCP

## Norme

TCP est défini par la norme RFC 793.
Il est corrigé par les normes RFC 1122 et RFC 1323.

## Fonctionnement

L'entité de transport TCP

1. Accepte des flux de données utilisateurs.
2. Elle les segmente en unités ne dépassant pas 64Ko (en pratique c'est souvent des paquets de 1 460 octets pour pouvoir les rentrer dans une trame Ethernet avec en-tête IP et TCP).
3. Envoie chaque unité sous forme de datagramme IP.
4. À la réception les paquets IP contenant des données TCP sont remis à l'entité TCP qui reconstruit le message.

# Le protocole TCP

## Fiabilité

- La couche IP ne garantissant pas la bonne remise des datagrammes, l'entité TCP doit gérer un timer et retransmettre, si nécessaire, les données perdues.
- De même, les datagrammes peuvent arriver dans le désordre : TCP doit donc être capable de les ré-assembler en messages correctement ordonnés.

# Interface TCP

## Adresse TCP

- Un service TCP fonctionne à l'aide de deux sockets connectées (une côté client et une autre côté serveur).
- Nous avons déjà vu que pour un service TCP/IP, chaque socket possède une adresse constituée de l'adresse IP (NSAP) de la machine hôte et un port TCP local à la machine (TSAP).
- Le port est un nombre sur 16 bits.
- Une connexion est identifiée par le couple formé par les deux sockets connectées.
- Ainsi, une socket peut-être utilisée pour plusieurs connexion.

# Interface TCP

## Adresse TCP

- Les ports de numéro inférieur à 1024 sont appelés ports réservés (*well-known port*) et sont utilisés par les services standards. Ces numéros de port sont assignés par l'IANA (*Internet Assigned Numbers Authority*).
- Les ports de numéros compris entre 1024 et 49151 sont appelés ports enregistrés. Il sont généralement utilisés par les processus clients mais ils peuvent également être utilisés par des services non enregistrés par l'IANA.
- Les ports de numéros supérieurs à 49151 sont appelés ports dynamiques ou encore ports privés et correspondent à des connexions TCP éphémères de processus clients.

# Quelques ports réservés par TCP

## Services standard

| Port | Protocole | Utilisation |
|---|---|---|
| 21 | FTP | Transfert de fichiers |
| 23 | Telnet | Ouverture de session distante |
| 25 | SMTP | E-mail |
| 69 | TFTP | Protocole de transfert de fichiers |
| 79 | Finger | Recherche d'informations sur l'utilisateur |
| 80 | HTTP | World Wide Web |
| 110 | POP-3 | Accès e-mail à distance |
| 119 | NNTP | News Usenet |

# Services standard

## Lancement automatique

- Afin d'éviter de devoir lancer tous les serveurs au démarrage de la machine, on préfère parfois proposer un démon (`inetd` pour *InterNET Daemon* par exemple) qui écoute différents ports afin d'éviter d'encombrer la mémoire de démons inactifs.
- Lorsqu'une connexion se présente le démon lance le serveur correspondant et le laisse gérer la requête.
- L'entrée et la sortie standard sont redirigées vers la socket, le serveur lancé par `inetd` n'a donc plus besoin de s'occuper des aspects réseaux.

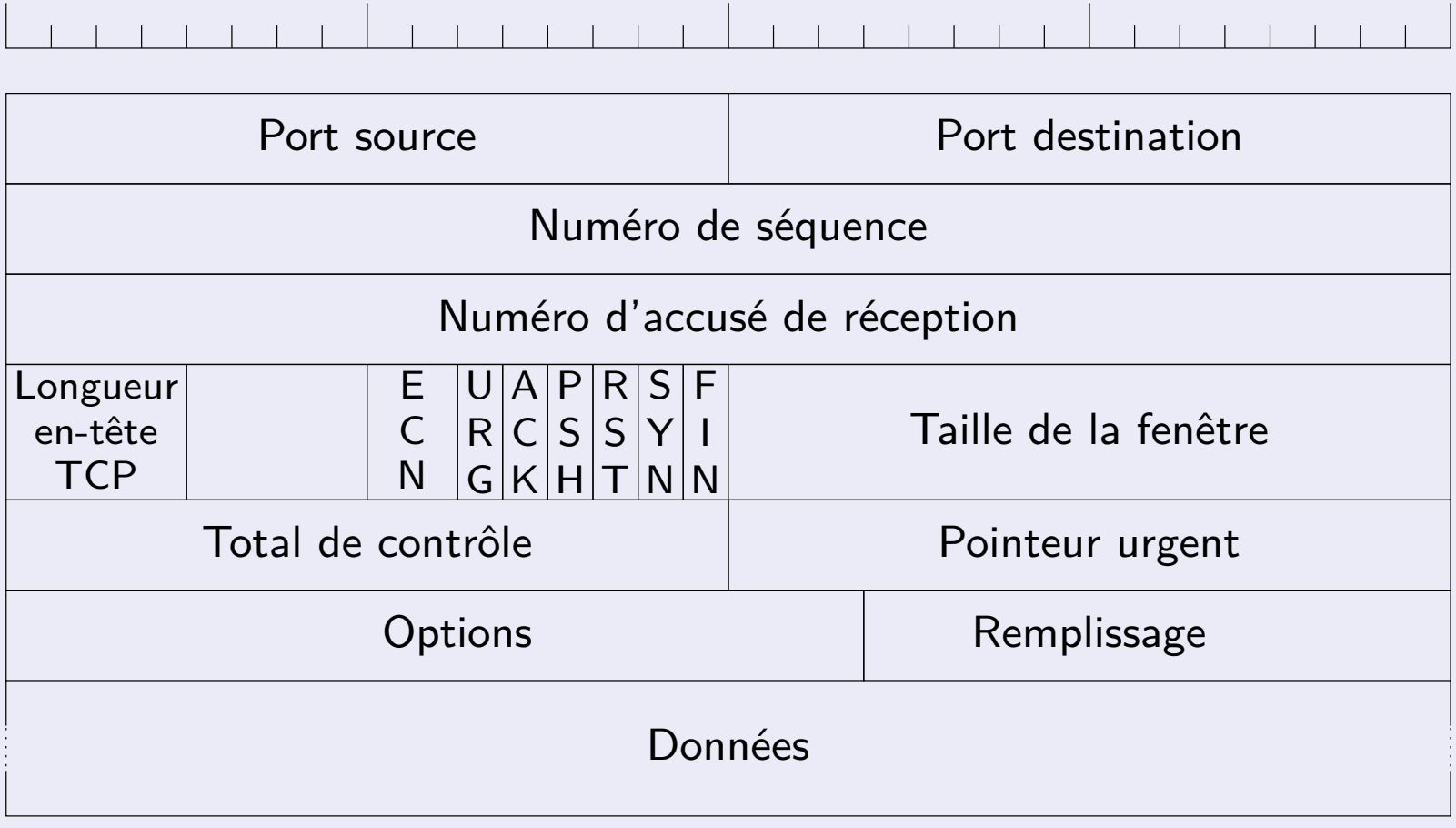
# Services standard

## Connexion TCP

- Toutes les connexions TCP sont bidirectionnelles en mode point-à-point. Ainsi TCP ne permet pas la multidiffusion (*multicasting*), ni la diffusion générale (*broadcasting*).
- Une connexion TCP correspond à un flux d'octets et non de messages : la délimitation des messages n'est pas conservée (comme lorsqu'on écrit dans un fichier).

# Protocole TCP

## Segment TCP



# Protocole TCP

## Segment TCP

- Longueur de l'en-tête est la taille de l'en-tête en mots de 32 bits
- Les options permettent, par exemple, de spécifier la taille maximale des paquets
- Le bit URG sert à indiquer où se trouvent d'éventuelles données urgentes
- Le bit PSH indique qu'on ne souhaite pas que les données soient placées dans un tampon avant d'être transmises à l'application
- Le bit RST indique qu'il s'est produit une erreur et que l'émetteur souhaite réinitialiser la connexion
- Le champ taille de la fenêtre permet la gestion dynamique de la taille de la fenêtre (voir cours de master)
- Les autres champs seront vus plus loin

# Protocole TCP

## Exemple d'un échange TCP

Un exemple de datagramme est ici.

# Plan

# Adresses internet pour les sockets

## Adresses

Une adresse de socket de domaine *AF_INET* est représentée par la structure *sockaddr_in* :

```
struct sockaddr_in {
  sa_family_t    sin_family;
  in_port_t      sin_port;
  struct in_addr sin_addr;
};
```

où *sin_family* vaut toujours *AF_INET*. *sin_addr* et *sin_port* correspondent, respectivement, à l'adresse IP et au numéro de port.

## Remarque

Il existe également des adresses IPv6 (domaine *AF_INET6*) que nous ne détaillerons pas ici.

# Little et Big-Endian

## Représentation

- Les protocoles internet utilisent une représentation en Big-Endian (octet de poids fort en premier).
- Cependant rien n'assure que votre machine utilise cette représentation ni que l'unité atomique de votre machine soit l'octet.
- POSIX propose un ensemble de fonctions sur les entiers permettant de passer de la représentation machine à la représentation réseau (*network byte order*).

# Little et Big-Endian

## Fonctions de conversion

Les fonctions permettant de passer un entier de la représentation réseau (***n**etwork*) à la représentation de la machine (***h**ost*) et *vice versa* sont :

```
uint32_t htonl(uint32_t hostlong);
uint16_t htons(uint16_t hostshort);

uint32_t ntohl(uint32_t netlong);
uint16_t ntohs(uint16_t netshort);
```

## Exemple pour les numéros de port

Ainsi pour remplir le numéro de port d'une adresse, on utilisera :

```
// addr est de type struct sockaddr_in
// numero est de type int
addr.sin_port = htons(numero);
```

# Manipulation des adresses IP

## Structure `struct in_addr`

Nous avons vu que les sockets utilisent une structure `struct in_addr`. Il s'agit d'une structure qui contient un champ :

```
struct in_addr {
  uint32_t        s_addr;
};
```

## Structure `struct in_addr`

Il existe des macros représentant des adresses IPv4 spécifiques :

- *INADDR_LOOPBACK* désigne l'adresse de la boucle locale (`127.0.0.1`).
- *INADDR_ANY* désigne toutes les interfaces réseaux locales (`0.0.0.0`).
- *INADDR_BROADCAST* désigne n'importe quel hôte (`255.255.255.255`).

# Manipulation des adresses IP

**Exemple**

- Un premier exemple de création d'une adresse pour un serveur : ici
- On peut vérifier que le serveur est bien en écoute avec la commande `netstat -a --tcp`.

# Manipulation des adresses IP

## Adresses IPv4

Une adresse IPv4 est constituée de quatre octets. On la représente généralement à l'aide de la notation pointée : c'est-à-dire en écrivant les valeurs décimales des octets séparées par des points (par exemple `192.168.0.1`).
Il nous faut donc un moyen pour passer de la notation pointée à la représentation interne des adresses et *vice versa*.

# Manipulation des adresses IP

## *inet_pton*

On peut utiliser la fonction *inet_pton* pour passer d'une adresse IPv4 en notation pointée en une adresse manipulable par le réseau :

```
int inet_pton(int af, const char *src, void *dst);
```

- *af* indique la famille d'adresse (*AF_INET* pour IPv4).
- *src* est une adresse en notation pointée pour IPv4.
- *dst* est l'adresse en représentation réseau (de type `struct in_addr` pour IPv4).

La fonction retourne 1 si elle réussit, 0 si *src* n'est pas valide et -1 si *af* ne contient pas une famille d'adresse valable.

# Manipulation des adresses IP

**Exemple**

Un exemple d'utilisation de *inet_pton* ici.

# Manipulation des adresses IP

*inet_ntop*

Pour passer d'une adresse IPv4 représentée selon l'ordre des octets du réseau en une adresse en notation pointée, nous pourrons utiliser :

```
#include <arpa/inet.h>

const char *inet_ntop(int af, const void *src,
                      char *dst, socklen_t size);
```

- *af* contient la famille d'adresse (*AF_INET* pour nous).
- *src* contient un pointeur sur l'adresse réseau (une `struct` *in_addr* pour IPv4).
- La fonction retourne le résultat dans *dst* qui est une chaîne de taille maximale *size*.

# Manipulation des adresses IP

**Exemple**

L'exemple précédent contenait également un exemple d'utilisation de *inet_ntop* ici.

**Remarque**

- Les fonctions *inet_pton* et *inet_ntop* permettent aussi de manipuler les adresses IPv6.
- Le « *p* »dans le nom des fonctions précédentes signifie « présentation ».

# Manipulation avancée des adresses IP

*getaddrinfo*

La fonction *getaddrinfo* permet la manipulation d'adresses et de services réseau. Cette fonction est une boîte à outils permettant de réaliser plusieurs actions.

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

- *node* est une chaîne de caractères contenant soit une adresse IP au format numérique, soit un nom de machine.
- Cette fonction fonctionne avec IPv4 ou IPv6. De plus elle est ré-entrante.
- Elle retourne une liste de structures *addrinfo* correspondant à l'hôte *node* et au service *service*.

# Manipulation avancée des adresses IP

**`struct addrinfo`**

La structure `struct addrinfo` contient les champs suivants :

```
int              ai_flags;
int              ai_family;
int              ai_socktype;
int              ai_protocol;
socklen_t        ai_addrlen;
struct sockaddr *ai_addr;
char            *ai_canonname;
struct addrinfo *ai_next;
```

Cette structure est utilisée pour récupérer des adresses mais également pour transmettre des indications à `getaddrinfo`.

# Manipulation avancée des adresses IP

## Paramètre *hints*

Le paramètre *hints* contient des indications sur ce que doit renvoyer la fonction *getaddrinfo* :

- *ai_family* : indique la famille des adresses désirées (*AF_INET* ou *AF_INET6*). *AF_UNSPEC* indique n'importe quel type d'adresse.
- *ai_socktype* : indique le type de socket désirée (*SOCK_STREAM* ou *SOCK_DGRAM*). 0 indique n'importe quel type de socket.
- *ai_protocol* : indique le protocole des adresses de socket (par exemple *IPPROTO_TCP* ou *IPPROTO_UDP*). 0 indique n'importe quel type de protocole.
- *ai_flags* : options supplémentaires (*OU* binaire).

Tous les autres champs doivent-être à 0 ou à *NULL*.

# Manipulation avancée des adresses IP

## Paramètre *hints*

- Si *ai_flags* contient *AI_NUMERICHOST* alors la fonction ne cherchera pas à résoudre les noms de machine.
- Si *ai_flags* contient *AI_NUMERICSERV* alors la fonction ne cherchera pas à résoudre les noms de services.
- Si *ai_flags* contient *AI_PASSIVE* et que *node* est *NULL* alors les adresses retournées contiendront *INADDR_ANY* ou *IN6ADDR_ANY_INIT* comme adresse IP (adresse joker).
- Si *ai_flags* ne contient pas *AI_PASSIVE* et que *node* est *NULL* alors les adresses retournées contiendront *INADDR_LOOPBACK* ou *IN6ADDR_LOOPBACK_INIT*.
- Si *service* est *NULL*, le numéro de port des adresses de socket renvoyées n'est pas initialisé.
- ...

# Manipulation avancée des adresses IP

## Exemple

- Nous avons déjà vu un exemple d'utilisation de *getaddrinfo* ici.
- Nous avons également utilisé *getaddrinfo* pour retrouver le port d'un service dont on connait le nom ici.
- Lorsqu'on souhaitera transformer une adresse et un port en une structure manipulable par le réseau nous pourrons, par exemple, utiliser la fonction *build_inet_address* donnée en exemple ici.

# Manipulation avancée des adresses IP

## Remarque : fonctions diverses

Les exemples précédents utilisent :

- *gai_strerror* pour afficher un message d'erreur si *getaddrinfo* retourne une erreur.
- *freeaddrinfo* pour libérer la liste chaînée d'adresses remplie par *getaddrinfo*.

# Manipulation avancée des adresses IP

## *getnameinfo*

La fonction inverse de *getaddrinfo* est *getnameinfo* :

```
int getnameinfo(
  const struct sockaddr *sa, socklen_t salen,
  char *host, size_t hostlen,
  char *serv, size_t servlen, int flags);
```

- *sa* pointe vers une structure *sockaddr_in* ou *sockaddr_in6*.
- *host* ou *serv* peuvent valoir *NULL* si l'information correspondante n'est pas souhaitée. Sinon, les variables pointent vers une chaîne allouée par l'appelant (et de taille *len).

# Manipulation avancée des adresses IP

*getnameinfo*

*flags* influe sur le comportement de la fonction :

- *NI_NAMEREQD* : produit une erreur si le nom d'hôte n'a pu être trouvé.
- *NI_DGRAM* : indique que le service est plutôt à rechercher pour l'UDP (pertinent, par exemple, pour les ports 512-514 où les services diffèrent en fonction du protocole).
- *NI_NOFQDN* : pour les hôtes locaux, renvoie seulement le nom de l'hôte (et non le nom complètement qualifié).
- *NI_NUMERICHOST* : renvoie le format numérique de l'hôte.
- *NI_NUMERICSERV* : renvoie le format numérique du service.

# Manipulation avancée des adresses IP

**Exemple**

- Nous avons déjà vu un exemple d'utilisation de *`getnameinfo`* ici.
- Un exemple de synthèse est présenté sous la forme d'un mini client/serveur UDP dans les sources ici et ici.

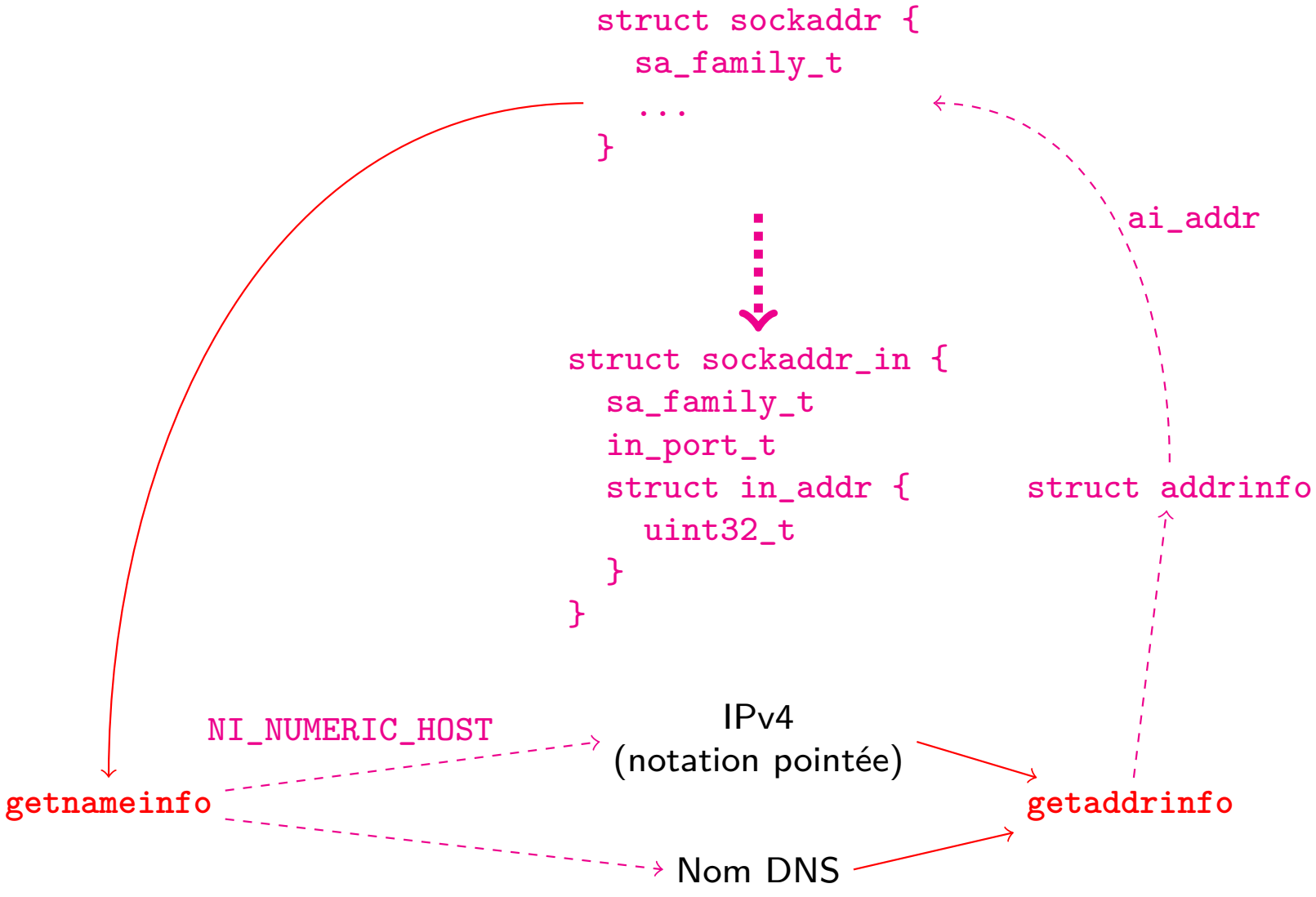
# Manipulation des adresses IP

## Autres fonctions

En recherchant des exemples sur le net, vous trouverez d'autres fonctions de manipulation des adresses IP. Mais attention :

- Les fonctions *gethostbyname* et *gethostbyaddr* sont obsolètes (et même supprimées dans POSIX.1-2008).
- La fonction *inet_aton* n'est pas POSIX. La fonction *inet_ntoa* ne permet de traiter que IPv4, il faut lui préférer *inet_ntop*. La fonction *inet_addr* est mal conçue.

# Manipulation des adresses internet IPv4 : synthèse



# Plan

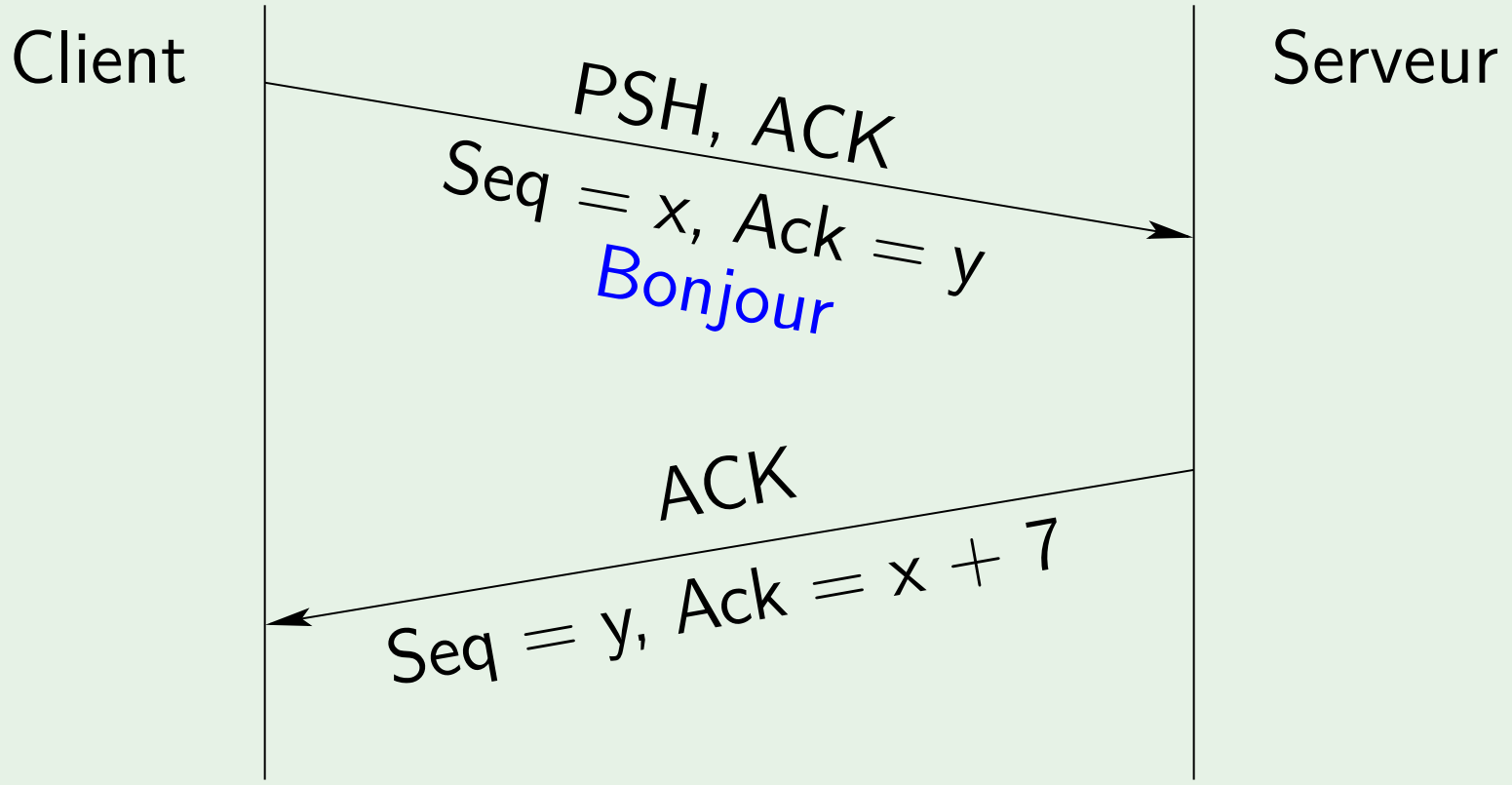
# Numéro de séquence

## Séquence

- La fiabilité du protocole est assurée (entre autre) par les numéros de séquence présents dans les en-têtes TCP.
- Au début de la connexion, un numéro de séquence est générée par chacune des parties.
- Chaque paquet est ensuite émis avec son numéro de séquence.
- En première approximation, on peut dire que le numéro de séquence augmente en fonction du nombre d'octets de données envoyé.
- Chaque message envoyé est acquitté par le récepteur à l'aide d'un message `ACK`.
- En cas de non réception de l'acquittement, un timer permet de décider si on peut considérer le message comme perdu et s'il faut le renvoyer.

# Envoi de données

## Exemple d'envoi de données



# Établissement de la connexion

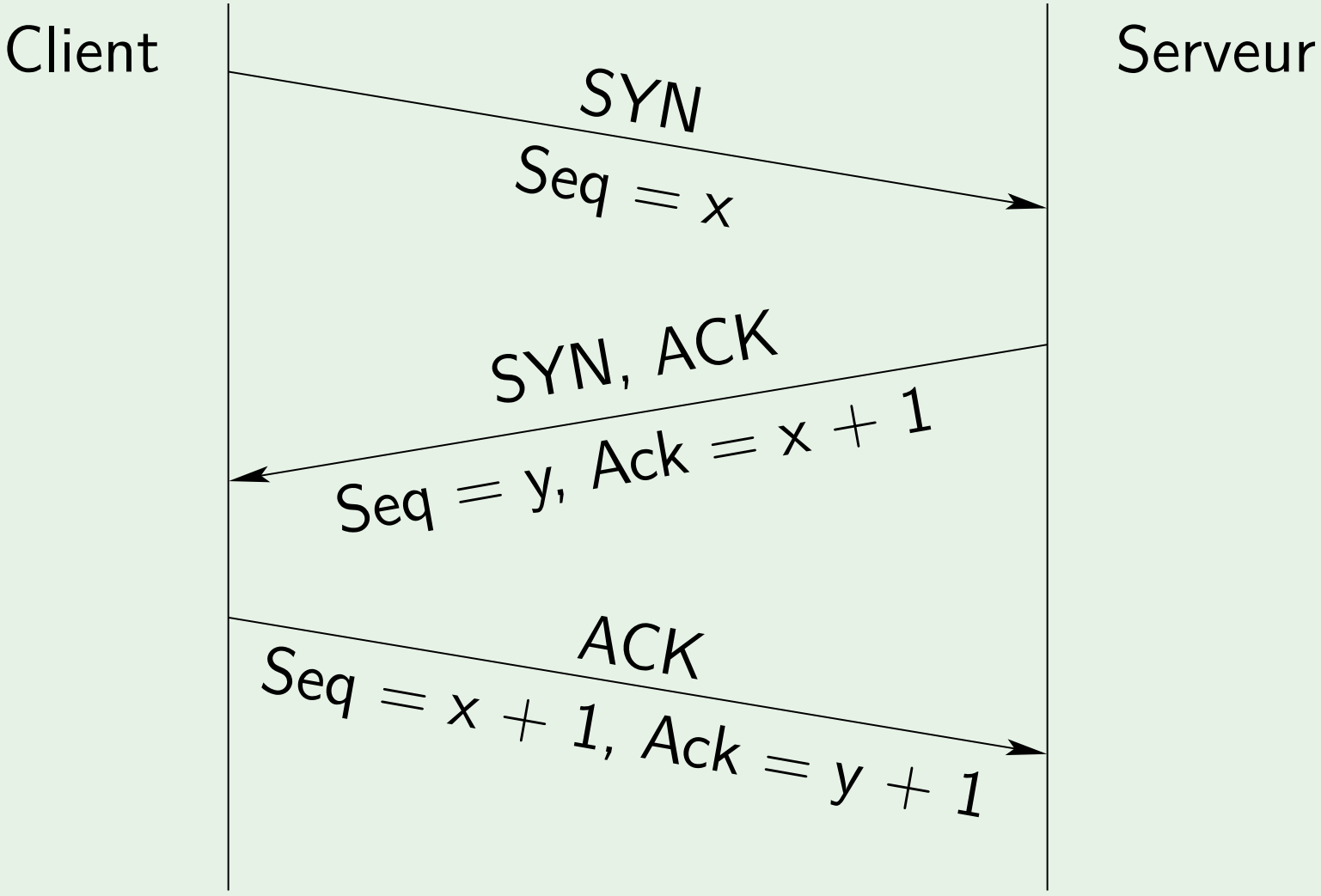
## Connexion

Pour établir la connexion, on utilise une procédure en trois étapes (*three ways handshake*).

- Le client émet un paquet avec le bit `SYN` à 1 et le bit `ACK` à 0.
- Si du côté du serveur il n'y a pas de processus à l'écoute du port, l'entité TCP émet un paquet avec le bit `RST` à 1.
- Sinon le démon côté serveur peut accepter la connexion en envoyant un paquet avec les bits `SYN` et `ACK` à 1.
- Enfin le client acquitte le dernier paquet.
- Chaque paquet `SYN` est considéré comme ayant une taille de 1 octet.

# Connexion

## Exemple d'établissement de connexion



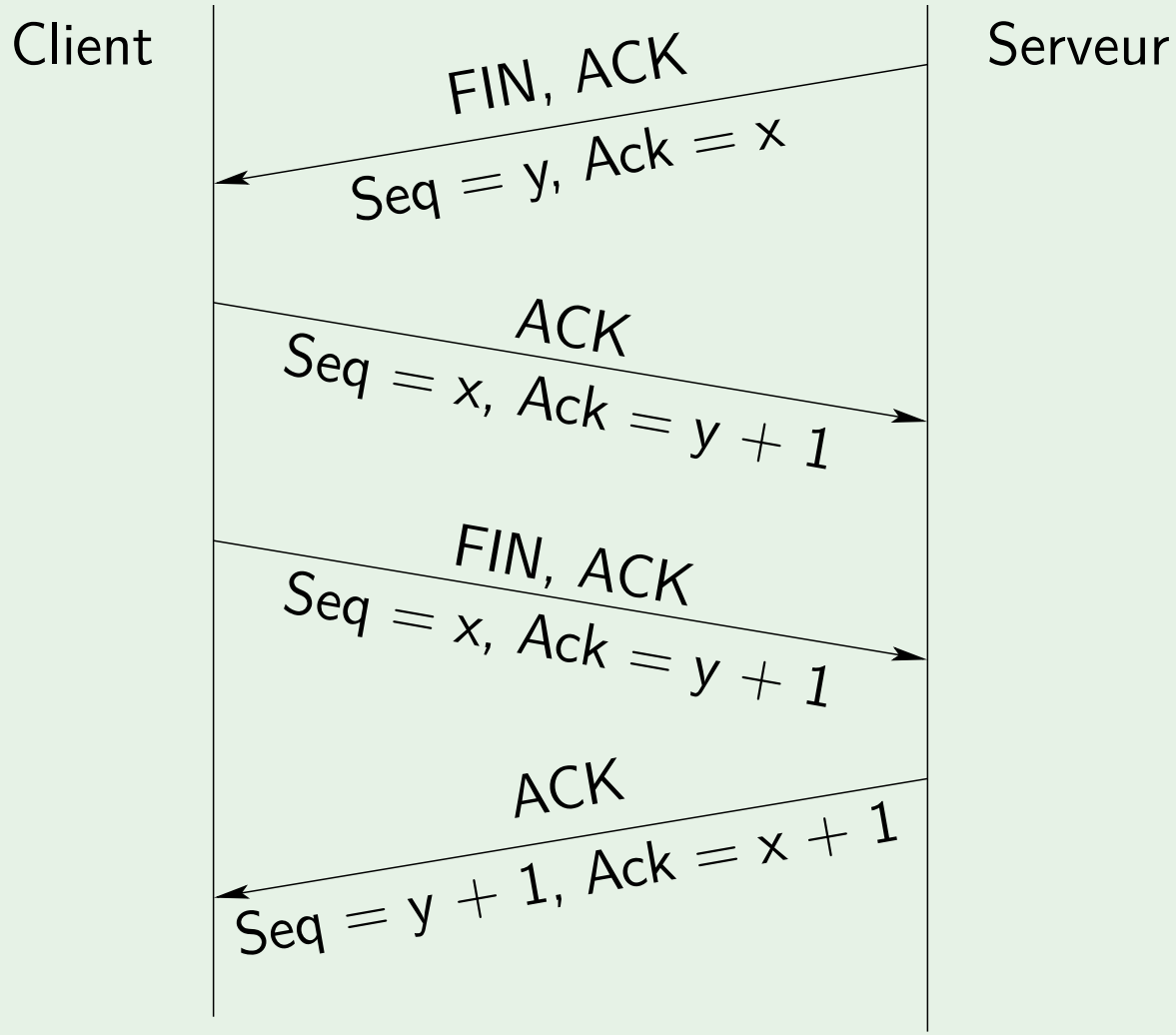
# Fermeture de la connexion

## Libération d'une connexion

- Quatre étapes sont généralement nécessaires pour fermer une connexion.
- Ceci est du au fait qu'une connexion est bidirectionnelle, elle peut donc n'être fermée que partiellement.
- Chaque partie doit indiquer qu'elle n'aura plus besoin d'envoyer de données à l'aide d'un paquet `FIN` qui sera acquitté par le correspondant.

# Libération d'une connexion

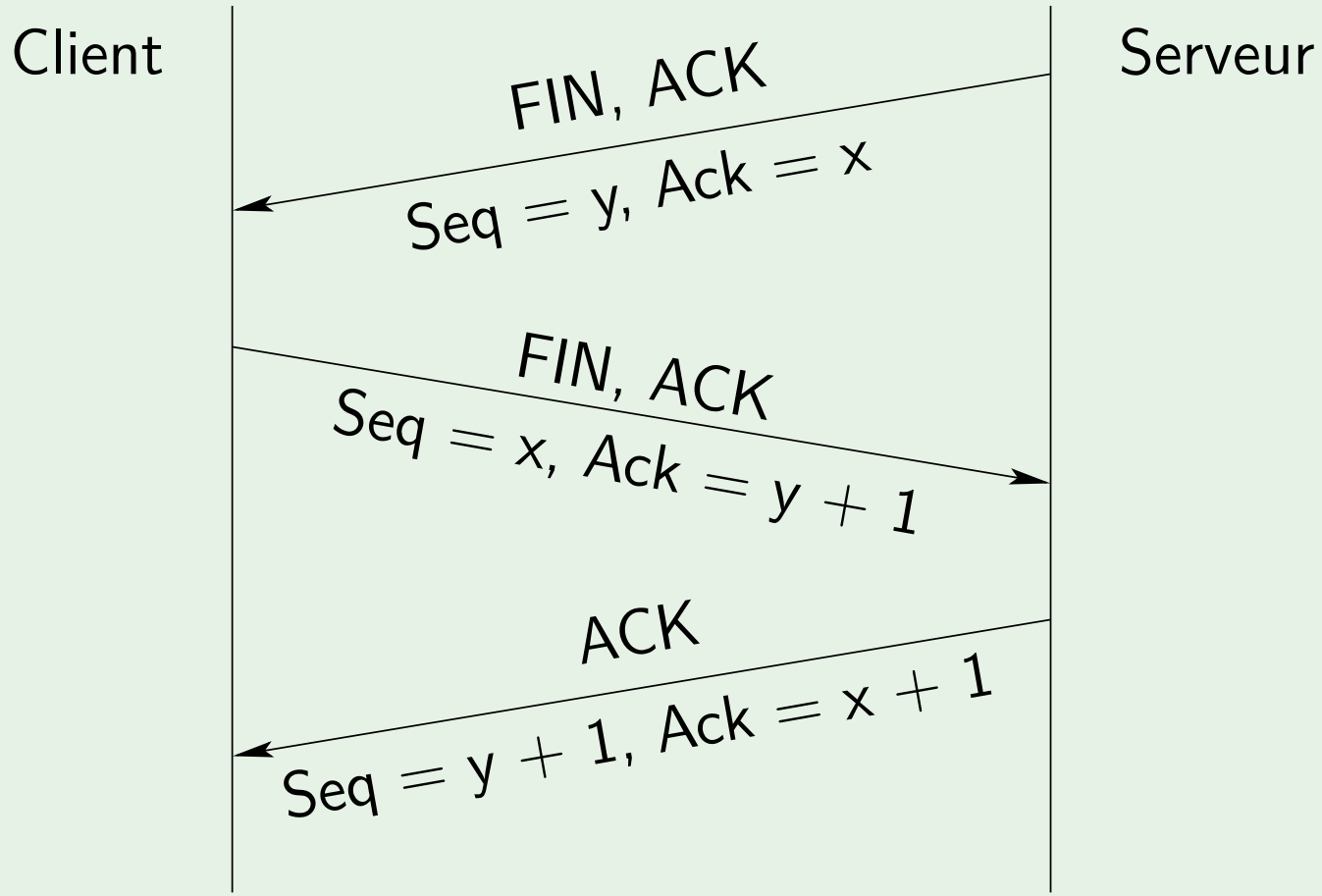
## Exemple de libération de connexion



# Libération d'une connexion

## Exemple de libération de connexion en trois étapes

Lorsque les deux parties ferment la connexion simultanément, les quatre étapes peuvent être « condensées » en trois :



# Plan

# Java et les sockets

## API Java

- L'API Java propose une implémentation des sockets dans le package java.net.
- La manipulation des sockets en Java est similaire à celle avec les sockets UNIX.

# Java et les sockets

## Un exemple de client

```
Socket socket = null;
PrintWriter out = null;
try {
  socket = new Socket("localhost", 44400);
  out = new PrintWriter(socket.getOutputStream());
} catch (UnknownHostException e) {
  System.err.println("Couldn't find host");
  System.exit(1);
} catch (IOException e) {
  System.err.println("Couldn't get I/O");
  System.exit(1);
}
```

# Java et les sockets

## Un exemple de client

```
out.print("bonjour ");
out.print("tout le monde");
out.close();

socket.close(); // could throw IOException
```

# Java et les sockets

## Un exemple de serveur

```
ServerSocket listeningSocket = null;
try {
  listeningSocket = new ServerSocket(44400);
} catch (IOException e) {
  System.err.println("Couldn't listen");
  System.exit(1);
}
while (true) {
    Socket serviceSocket = null;
    try {
      serviceSocket = listeningSocket.accept();
    } catch (IOException e) {
      System.err.println("I/O error while accept");
      System.exit(1);
    }
    new ProcessConnection(serviceSocket).start();
}
```

# Java et les sockets

## Un exemple de serveur

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

public class ProcessConnection extends Thread {
  private Socket socket = null;

  public ProcessConnection(Socket serviceSocket) {
    socket = serviceSocket;
  }
```

# Java et les sockets

## Un exemple de serveur

```
  public void run() {
    try {
      BufferedReader in =
          new BufferedReader(new InputStreamReader(
              socket.getInputStream()));

      String line;
      while ((line = in.readLine()) != null) {
        System.out.println(line);
      }
      in.close();
      socket.close();
    } catch (IOException e) {
      e.printStackTrace();
    }
  }
}
```

# Java et les sockets

## InetAddress

Java propose également des classes pour gérer les adresses IP : *Inet4Address* et *Inet6Address* qui héritent toutes deux de *InetAddress* (*cf* javadoc).

## InetAddress

On peut, par exemple, obtenir une adresse IP grâce à la méthode statique de *InetAddress* :

```
public static InetAddress getByName(String host)
                                    throws UnknownHostException
```

# Java et les sockets

## Un exemple de serveur ECHO en UDP

```
public static void main(String[] args)
        throws IOException {
    DatagramSocket socket = new DatagramSocket(4445);
    while (true) {
        byte[] buf = new byte[256];
        DatagramPacket packet =
            new DatagramPacket(buf, buf.length);
        socket.receive(packet);

        InetAddress address = packet.getAddress();
        int port = packet.getPort();
        packet = new DatagramPacket(buf, buf.length,
                                    address, port);
        socket.send(packet);
    }
}
```

# Java et les sockets

## Le client UDP

```
public static void main(String[] args)
        throws IOException {
    if (args.length != 1) {
        System.out.println(
            "Usage: java ClientSocketUDP message");
        return;
    }
    DatagramSocket socket = new DatagramSocket();
    InetAddress address =
        InetAddress.getByName("localhost");
```

# Java et les sockets

## Le client UDP

```
    byte[] buf = args[0].getBytes();
    DatagramPacket packet =
        new DatagramPacket(buf, buf.length,
                           address, 4445);
    socket.send(packet);

    byte[] resp = new byte[256];
    packet = new DatagramPacket(resp, resp.length);
    socket.receive(packet);

    String received =
        new String(packet.getData(),
                   0, packet.getLength());
    System.out.println("Echo: " + received);

    socket.close();
}
```

# Windows et les sockets

## L'interface Winsock

- Sous Windows, les applications ont accès au réseau à travers l'interface Winsock
- Cette interface reprend le paradigme des sockets Berkeley.
- On retrouve - entre autres - les appels systèmes déjà présentés dans ce cours comme *socket*, *bind*, *listen*, *accept*, *connect*, *sendto*, *getaddrinfo*, ...

# Plan

# Objectifs de la couche réseau

## Rôle de la couche réseau

- La couche réseau se charge d'acheminer des paquets d'une source vers une destination.
- Elle prend en charge le paquet de bout en bout contrairement à la couche inférieure qui se contente d'acheminer une trame sur une ligne entre deux interfaces réseau.

# Objectifs de la couche réseau

## Routage

La principale tâche de la couche réseau est donc le routage des paquets. Cela implique :

- une bonne connaissance de la topologie du sous-réseau de communication ;
- des algorithmes qui essaient de ne pas surcharger des lignes de communication alors que d'autres sont inactives ;
- la capacité à passer d'un type de réseau à un autre.

# Services

## Objectifs des services fournis

- Les services doivent être indépendants des technologies de routeur mises en places ;
- La couche transport doit être indépendante du nombre et du type des routeurs ainsi que de la topologie du sous-réseau ;
- Les adresses réseau doivent rester uniformes sur des LAN comme sur des WAN.

# Plan

# Retour sur les adresses IP

## Adresses

- Une adresse IP identifie chaque interface avec le réseau.
- Une machine (routeur, ordinateur, modem ADSL, imprimante réseau, *etc*) peut avoir plusieurs interfaces.
- Une interface peut avoir plusieurs adresses IP.

# Retour sur les adresses IP

## Sous-réseau

IP étant utilisé dans le cadre d'une interconnexion de réseaux, les adresses IP contiennent en fait :

- une partie permettant de distinguer le sous-réseau (nécessaire pour un routage efficace) ;
- une partie permettant de distinguer l'interface dans le sous-réseau.

# Retour sur les adresses IP

## Masque de sous-réseau

Afin de déterminer où s'arrête la partie contenant le sous-réseau et où commence le numéro de l'interface

- on utilisait autrefois (jusqu'au milieu des années 90) le préfixe de l'adresse qui déterminait le masque à utiliser (on parlait de classes d'adresses A, B ou C) ;
- on utilise désormais un masque de sous-réseau qui est un mot de 32 bits où les bits à 1 indiquent les bits de l'adresse qui sont utilisés pour le sous-réseau.

# Retour sur les adresses IP

## Masque de sous-réseau

À partir d'une adresse IP et d'un masque, on peut donc déterminer :

- l'adresse de sous-réseau en appliquant un ET binaire entre l'adresse et le masque de sous-réseau ;
- l'adresse de l'hôte à l'intérieur du sous-réseau en appliquant un ET binaire entre l'adresse et le complément à 1 du masque de sous-réseau.

# Retour sur les adresses IP

## Exemple

Pour l'adresse `10.195.163.94` et le masque `255.255.240.0`.

- L'adresse de sous-réseau est :

$$\begin{array}{lcccccccc} & 00001010 & & 11000011 & & 10100011 & & 01011110 \\ \wedge & 11111111 & & 11111111 & & 11110000 & & 00000000 \\ = & 00001010 & & 11000011 & & 10100000 & & 00000000 \\ = & 10 & . & 195 & . & 160 & . & 0 \end{array}$$

- L'adresse de l'interface au sein du sous-réseau est :

$$\begin{array}{lcccccccc} & 00001010 & & 11000011 & & 10100011 & & 01011110 \\ \wedge & 00000000 & & 00000000 & & 00001111 & & 11111111 \\ = & 00000000 & & 00000000 & & 00000011 & & 01011110 \\ = & 0 & . & 0 & . & 3 & . & 94 \end{array}$$

# Retour sur les adresses IP

## Notation CIDR

On peut également utiliser la notation CIDR (*Classless Inter-Domain Routing*) pour indiquer le masque de sous-réseau. L'adresse IP est suivie d'une barre oblique et d'un numéro qui indique le nombre de bits à 1 dans le masque de sous-réseau.

## Exemple

L'adresse `10.195.163.94` avec le masque `255.255.240.0` peut ainsi être notée :

10.195.163.94/20

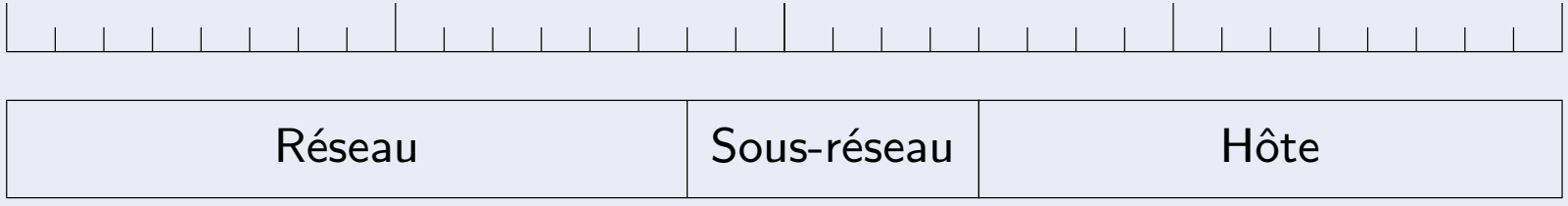
# Retour sur les adresses IP

## Sous-réseaux

- Le masque de sous-réseau peut également permettre de créer des sous-réseaux à l'intérieur d'un réseau : on parle alors de sous-réseaux (*subnet*).
- Le réseau est partitionné en plusieurs entités à usage interne mais il se comporte comme un seul réseau vis-à-vis de l'extérieur.

## Adresses de sous-réseaux

Une adresse d'une machine à l'intérieur d'un réseau utilisant le *subnetting* sera alors :



# Adresses IPv4 spécifiques

## Adresses particulières

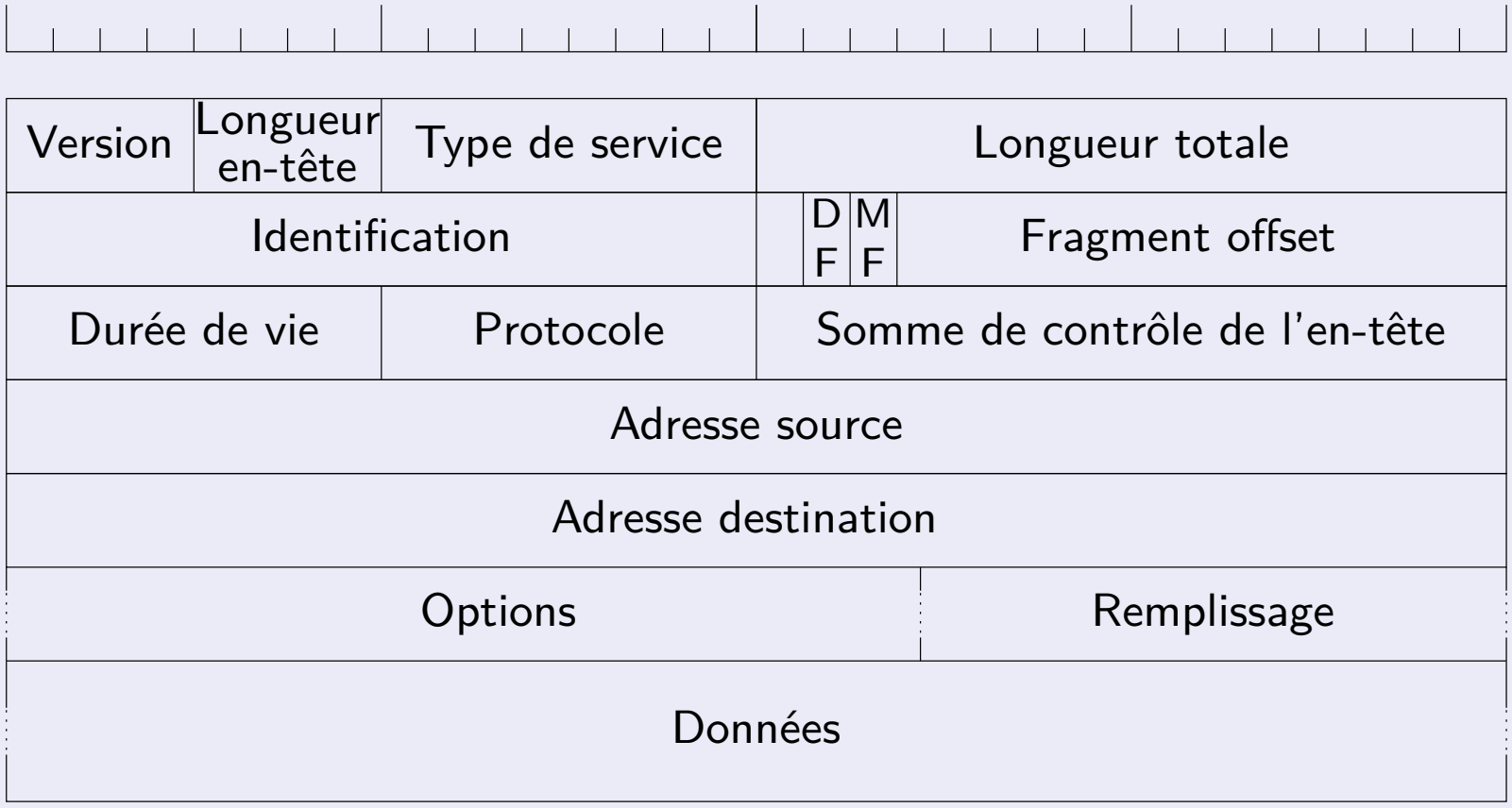
Certaines adresses IPv4 ont une signification particulière. En voici quelques unes :

| Adresses | Signification |
|---|---|
| 0.0.0.0/255 | désigne l'interface |
| 127.0.0.0/8 | adresse de bouclage (loopback) |
| | |
| 10.0.0.0/8 | adresses privées (NAT) |
| 172.16.0.0/12 | adresses privées (NAT) |
| 192.168.0.0/16 | adresses privées (NAT) |
| 255.255.255.255/32 | diffusion limitée |

De plus la première adresse d'un réseau spécifie le réseau lui-même, la dernière est une adresse de diffusion.

# Protocole IP

## Segment IPv4



# Protocole IP

**Exemple**

Voici un exemple de capture d'un paquet IP lors de l'envoi d'un *GET* via http : Voir l'exemple.

# Adresses IPv6

## IPv6

Nous ne détaillerons pas IPv6, sachez seulement que :

- Les adresses IPv6 utilisent 128 bits.
- Les adresses sont notées sous la forme de 8 blocs de 16 bits représentés par des nombres hexadécimaux et séparés par des « : ».
- Il existe différents types d'adresses : adresses réservées, adresses unicast, adresses locales lien, adresses anycast ou encore adresses multicast.
- En IPv6 le masque de sous-réseau des adresses unicast est fixé à 64.

# Adresses IPv6

## Exemples d'adresses IPv6

- `2a00:1450:4007:801:0:0:0:1010` est l'adresse *ipv6.google.com*
- `::1/128` indique la boucle locale
- *fe80*`::221:70`*ff*`:`*febd*`:355`*f*`/64` est l'adresse locale de ma machine
- `::`*ffff*`:10.195.163.94` est une adresse IPv4 "mappée".
- Les adresses locales (uniques) appartiennent à la plage *fc00*`::/7`.
- ...

# Plan

# Protocoles de contrôle

## Autres protocoles de la couche réseau

Internet dispose de plusieurs protocoles de contrôle exécutés dans la couche réseau. Nous présenterons :

- ICMP
- ARP
- RARP, BOOTP et DHCP

# Protocole ICMP

## ICMP

Lorsqu'un événement inattendu est détecté par un routeur, il est signalé à l'aide du protocole ICMP (*Internet Control Message Protocol*).
Ce protocole est également utilisé à des fins de test.

## Paquet

Bien qu'étant au même niveau que le protocole IP, un paquet ICMP est néanmoins encapsulé dans un datagramme IP

# Protocole ICMP

## Messages ICMP

18 types de messages ICMP ont été définis. Par exemple :

| Type | Codes | Message |
|---|---|---|
| 0 | 0 | Écho à un message de type 8 |
| 3 | 0 - 15 | Destinataire inaccessible |
| | 4 | Fragmentation nécessaire mais bit DF à 1 |
| 8 | 0 | Demande d'écho (ping) |
| 11 | | Temps dépassé (par exemple, TTL à 0) |
| 12 | | En-tête erroné |

## Exemple

Voici un exemple de capture réseau lors d'un « *ping* » sur *www.google.com* : Voir l'exemple.

# Protocole ARP

## Adresses couche liaison de données

Les adresses IP ne peuvent pas être directement utilisées car les équipements de la couche liaison de données ne les comprennent pas. Par exemple, les cartes réseau Ethernet utilisent les adresses MAC (*Media Access Control*) pour émettre et recevoir des trames.

## ARP

Le protocole ARP (*Address Resolution Protocol*) permet à une machine de découvrir quelle est l'adresse MAC associée à une adresse IP appartenant au même sous réseau.

## NDP

En IPv6, c'est le protocole NDP (*Neighbor Discovery Protocol*) qui reprend les services d'ARP.

# Protocole ARP

## Fonctionnement

Lorsqu'une machine souhaite émettre un paquet à destination d'une machine du même sous-réseau et qu'elle connait son adresse IP (via DNS) :

- elle envoie sur le segment Ethernet (ou token ring) un paquet broadcast demandant à qui appartient l'adresse IP ;
- La machine concernée reconnait son adresse IP et peut alors répondre en envoyant son adresse matérielle.

## Fonctionnement

Un exemple de capture ici.

# Découverte d'adresse IP à partir d'adresses MAC

## Le problème

Lorsqu'une machine démarre elle peut avoir besoin de demander qu'elle est l'adresse IP qui lui est affectée. C'est le cas, par exemple, pour :

- un terminal X (boot à partir d'un disque réseau) ;
- un ordinateur nomade qui se connecte sur un nouveau réseau.

# Protocole RARP

## Protocole RARP

Le protocole RARP (*Reverse Address Resolution Protocol*) permet à une machine de connaître son adresse IP en fonction de son adresse matérielle.

## Fonctionnement

Le principe de RARP est le suivant :

- la machine souhaitant connaître son adresse IP émet une requête *broadcast* du type : « mon adresse matérielle est xxxxxx, quelle est mon adresse IP ? »
- le serveur RARP du sous-réseau voit la requête et répond en renvoyant l'adresse IP correspondante.

# Protocole BOOTP

## Problème

Le protocole RARP utilise une adresse de diffusion pour l'envoi des requêtes. Il faut donc un serveur RARP pour chaque sous-réseaux.

## Solution

Le protocole BOOTP (*bootstrap*) a été développé pour pallier à cet inconvénient.

## Fonctionnements

BOOTP utilise des paquets UDP qui, eux, sont transmis par les routeurs. Il permet également de transmettre des informations additionnelles pour les machines sans disque.

# Protocole DHCP

## Problème

BOOTP nécessite des tables de correspondances (adresse IP, adresse matérielle) configurées manuellement. On ne peut donc pas ajouter de façon dynamique un nouveau matériel.

## Solution

Le protocole DHCP (*Dynamic Host Configuration Protocol*) est une extension de BOOTP qui autorise à la fois une configuration manuelle et une assignation automatique des adresses IP.

# Protocole DHCP

## Principe

DHCP utilise un serveur spécifique qui se charge d'attribuer les adresses IP. Ce serveur ne se situe pas forcément sur le même sous-réseau que l'hôte demandeur aussi un agent de relais DHCP est nécessaire sur chaque sous-réseau.

## Fonctionnement

Une machine désirant connaître son adresse IP

- envoie une requête par diffusion *broadcast* ;
- l'agent de relais DHCP voit la requête et envoie une demande unicast au serveur DHCP ;
- le serveur DHCP peut alors transmettre l'adresse IP.

# Protocole DHCP

## Bail

- Les adresses allouées dynamiquement peuvent avoir une durée du vie limitée dans le temps : il s'agit du bail DHCP (*leasing*).
- Cela évite la pénurie d'adresses si des hôtes quittent le réseau sans restituer leur adresse IP.
- Un hôte dont son adresse arrive à expiration doit en demander une nouvelle sous peine de ne plus pouvoir se servir de son adresse IP.

## Exemple

Voici des exemples de capture d'une transaction DHCP entre ma machine et le serveur DHCP du département : ici et ici.

# Plan

# Routage

## Deux types de routage

Internet est constitué d'un très grand nombre de systèmes autonomes. Chacun d'eux peut utiliser son propre protocole de routage. Les objectifs de routage diffèrent selon qu'on doit acheminer un paquet dans un système autonome ou qu'on doit traverser plusieurs systèmes. Aussi on distingue deux types de protocoles de routage :

- Les protocoles de routage interne s'occupent d'acheminer le plus efficacement possible les paquets de la source à la destination.
- Les protocoles de routage externe tiennent compte de considération politiques, économiques, de sécurité et autre.

# Routage internet

## Normes internet pour le routage

Internet propose plusieurs protocoles de routage, dont :

- routage interne : RIPv2 et OSPF ;
- routage externe : BGP.

# Protocole RIP

## RIP

RIP (*Routing Information Protocol*) est un protocole à vecteur de distance basé sur l'algorithme de Bellman-Ford.

## Principe

- Avec l'algorithme de Bellman-Ford, un routeur maintient sa propre table de routage.
- Il s'agit d'un vecteur lui indiquant quel est la meilleure distance vers chaque destination ainsi que la ligne à utiliser.
- Le routeur met régulièrement à jour sa table avec les informations reçues de ses voisins.

# Protocole RIP

## RIPv2

RIPv1 ne supporte pas les masques de sous-réseau. La RFC 2453, développée en 1994, propose donc un RIPv2 gérant cette technologie. RIPv2 apporte également quelques nouveautés comme l'authentification et l'envoi des messages sur une adresse multicast au lieu de l'adresse de broadcast.

## Limitations de RIP

- Pour limiter les boucles de routage, le nombre de sauts est limité à 15.
- RIP ne prend en compte que la distance entre deux machines en terme de saut mais il ne considère pas la qualité de la ligne.
- RIP réagit rapidement aux bonnes nouvelles mais lentement aux mauvaises (les tables peuvent prendre du temps à se mettre à jour en cas de panne d'un routeur).

# Protocole OSPF

## OSPF

- Le protocole OSPF (*Open Shortest Path First*) est le protocole de routage interne dominant sur internet. Il a été normalisé en 1990 dans la RFC 2328.
- Il s'agit d'un protocole de routage hiérarchique par état de lien.
- OSPF utilise l'algorithme de Dijkstra pour déterminer la route la plus courte vers chacune des destinations connues.
- OSPFv2 (RFC 2328) est spécifique à IPv4. La RFC 5340 décrit une version d'OSPF pour IPv6.

# Protocole OSPF

## Exigences fonctionnelles d'OSPF

Le groupe chargé de concevoir le protocole OSPF à défini une suite d'exigences auxquelles il devait se conformer.

1. L'algorithme est ouvert (le « O » d'OSPF).
2. Il supporte une variété de métrique de distance : distance physique, délai de transmission, ...
3. Il est dynamique : il s'adapte automatiquement et rapidement aux changements de topologies.
4. Il réalise le routage en fonction du type de service (champ *Type de service* de l'en-tête IP). Cette exigence a été supprimée par défaut d'utilisation.

# Protocole OSPF

## Exigences fonctionnelles d'OSPF (suite)

5. Il réalise un équilibrage de charge en répartissant le trafic entre plusieurs liaisons.
6. Il supporte les systèmes hiérarchiques.
7. Il supporte un minimum de sécurité.
8. Il supporte les routeurs connectés à l'internet via un tunnel.

# Protocole OSPF

## Principe

- Lorsqu'un routeur s'initialise il diffuse des messages *HELLO* en multicast sur les autres LAN à destination du groupe incluant tous les routeurs.
- Grâce aux réponses qu'il reçoit, chaque routeur découvre qui sont ses voisins. Les routeurs appartenant à un même LAN sont tous des routeurs voisins.
- OSPF fonctionne par échange d'information entre routeurs adjacents. Pour chaque LAN, un routeur est élu désigné : il est adjacent à tous les autres routeurs.

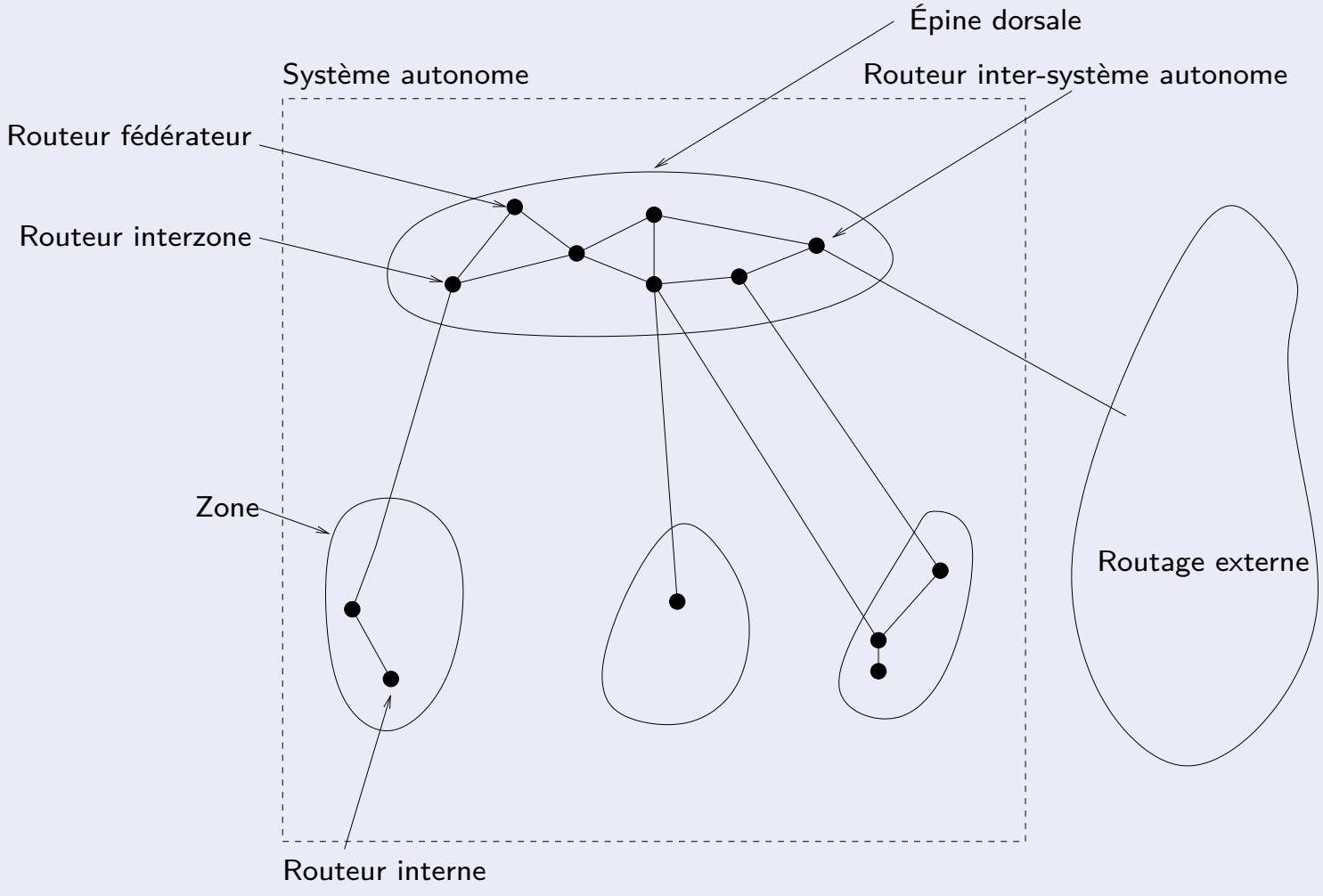
# Protocole OSPF

## Principe

- Chaque routeur émet régulièrement des messages *Mise à jour état de lien* vers ses routeurs adjacents. Ses messages sont acquittés par les autres routeurs.
- Les messages *Description de la base de données* permet à un routeur de comparer sa base de données avec celle d'un autre routeur.
- Un routeur peut demander à un routeur adjacent de lui communiquer des informations par le biais d'un message *Demande état de lien*.
- Chaque routeur construit un graphe pour sa zone et calcule les chemins les plus courts.

# Protocole OSPF

## Catégories de routeurs dans OSPF



# Protocole BGP

## BGP

- Le protocole BGP (*Border Gateway Protocol*) est un protocole de routage externe (routage entre système autonomes).
- BGP convoie des informations de routage pour IPv4 mais des extensions permettent le routage d'autres protocoles comme IPv6.

## Principes

Pour BGP le monde est constitué de systèmes autonomes et de liaisons les interconnectant.

# Protocole BGP

## Principes

BGP regroupe les réseaux en trois catégories.

- Les réseaux sans issues (*stub*) qui ont une seule connexion avec le graphe BGP et qui ne peuvent servir à acheminer du trafic.
- Les réseaux multiconnectés qui peuvent être utilisés pour transporter du trafic.
- Les réseaux de transit (comme les réseaux fédérateurs) qui transportent les paquets tiers généralement moyennant finance.

# Protocole BGP

## Principes

- Les routeurs BGP communiquent entre eux à travers des connexions TCP.
- BGP est dans son principe général un protocole à vecteur de distance.
- Au lieu de maintenir seulement le coût, chaque routeur consigne le chemin complet vers chaque destination.
- L'administrateur du système autonome peut également définir un ensemble de règles de sélection pour les routes.

# Plan

# Diffusion multicast

## Diffusion multicast

- Le multicast est l'émission d'un paquet d'un émetteur vers un ensemble de récepteurs.
- On parle également de diffusion multipoint ou de diffusion multigroupe.

## Diffusion multicast sur internet

- IP supporte la diffusion multicast à travers les adresses 224.0.0.0/4 (`224.0.0.0` à `239.255.255.255`, correspond à l'ancienne classe d'adresse D).
- Deux types d'adresses de groupes sont disponibles : les adresses permanentes et les adresses temporaires.
- Pour IPv6, les adresses de multicast sont `ff00::/8`

# Diffusion multicast

## Adresses de diffusion réservées

L'IANA réserve certaines adresses, par exemple :

- 224.0.0.0/24 : adresses multicast locale uniquement.
- 224.0.1.0/24 : adresses multicast pour internet (utilisé par NTP par exemple)
- 232.0.0.0/8 : SSM (*Source-Specific Multicast*, protocole de gestion de groupes à travers des « channels » : on indique la source des paquets qu'on veut recevoir)
- 233.0.0.0/8 : GLOP (RFC 2770, utilisé par les fournisseurs d'accès)
- 234.0.0.0/8 : *Unicast-Prefix-Based IPv4 Multicast Addresses* (RFC 6034, multicast à l'intérieur de réseaux disposant d'espace d'adressage)
- 239.0.0.0/8 : Adresses multicast de site (gestion du multicast au niveau d'une organisation)

# Diffusion multicast

## Exemple d'adresses de diffusion multicast permanentes

Parmi les adresses de diffusion multicast permanentes on trouve, par exemple :

- `224.0.0.1` : tous les systèmes d'un LAN
- `224.0.0.2` : tous les routeurs d'un LAN
- `224.0.0.5` : tous les routeurs OSPF d'un LAN
- `224.0.0.6` : tous les routeurs OSPF désignés d'un LAN
- `224.0.0.9` : tous les routeurs RIPv2 désignés d'un LAN

# Diffusion multicast

## Groupes temporaires

- Les groupes temporaires doivent être créés avant de pouvoir être utilisés
- Un processus peut demander à son hôte de rejoindre un groupe spécifique
- Chaque hôte effectue un suivi des groupes dont ses processus sont membres
- La diffusion est assurée par des routeurs multicast spécifiques
- Les routeurs multicast communiquent avec les hôtes à l'aide du protocole IGMP (*Internet Group Management Protocol*)

# Plan

# Objectifs de la couche liaison de données

## Rôle de la couche liaison de données

La couche liaison de données doit :

- Traiter les erreurs de transmission.
- Réguler le flux de données pour éviter que des destinataires lents soient submergés par des émetteurs rapides.

## Principe de fonctionnement

- La couche liaison de données prend les paquets émis par la couche réseau et les encapsule dans des trames de transmission.
- Chaque trame contient un champ en-tête (*header*), un champ de données et un champ de queue (*trailer*).

# Services

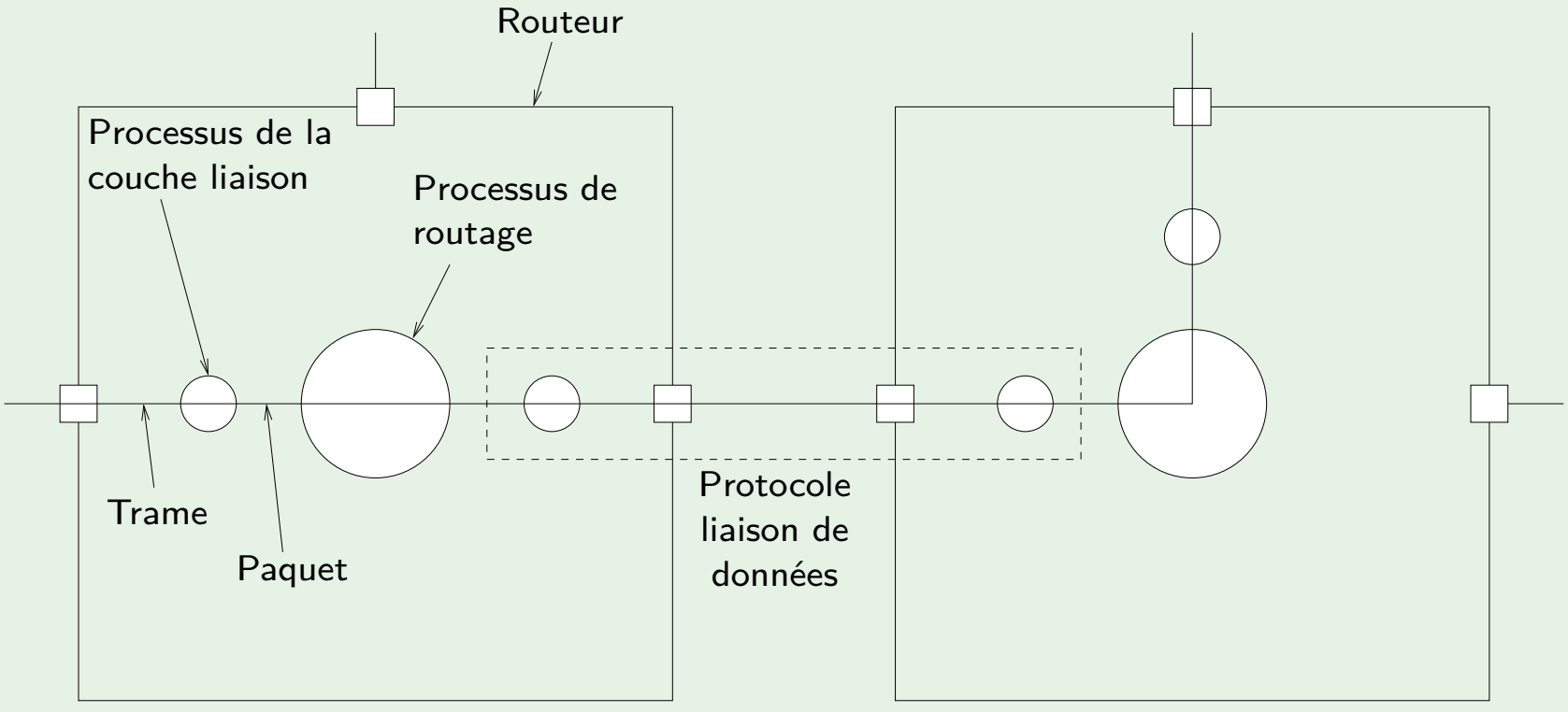
## Services

La couche liaison de données peut offrir trois types de services :

- Service sans connexion, sans accusé de réception.
- Service sans connexion, avec accusé de réception.
- Service avec connexion, avec accusé de réception.

# Protocole liaison de données

## Un exemple de fonctionnement



# Internet et la couche liaison de données

## Internet

Internet est essentiellement constitué :

- De réseaux locaux
- De liaisons point-à-point

## Couche et sous-couche

- La gestion des réseaux locaux est affectée à une sous-couche de la couche liaison : la couche de contrôle d'accès au canal.
- Les liaisons point-à-point sont directement gérées par la couche liaison de données.

# Plan

# Internet et la couche liaison de données

## Protocoles point-à-point

Utilisation des protocoles point-à-point :

- Liaisons louées point-à-point entre les réseaux locaux d'entreprises et des routeurs distants
- Liaisons point-à-point entre le modem de l'utilisateur et son FAI

# Protocole point-à-point de l'internet

## Protocole PPP

Le protocole PPP (*Point-to-Point Protocol*) est définit dans la RFC 1661. PPP repose sur trois composants :

1. Une encapsulation des datagrammes permettant de détecter sans ambiguïté le début et la fin d'une trame. Le format de la trame permet également la détection des erreurs.
2. Un protocole de contrôle de liaison LCP (*Link Control Protocol*) qui active une ligne, la teste, négocie les options et la désactive proprement lorsqu'on n'en a plus besoin.
3. Des protocoles de contrôle de la couche réseau NCP (*Network Control Protocol*) qui négocient les options de la couche réseau.

# Protocole PPP

## Trame PPP

| Octets | 1 | 1 | 1 | 1 ou 2 | Variable | 2 ou 4 | 1 |
|---|---|---|---|---|---|---|---|
| | Fanion 01111110 | Adresse 11111111 | Contrôle 00000011 | Protocole | Charge utile | Total de contrôle | Fanion 01111110 |

# Protocole point-à-point de l'internet

## Exemple de fonctionnement

Prenons l'exemple d'un utilisateur qui appelle son fournisseur d'accès pour établir une liaison avec l'internet :

- L'ordinateur de l'utilisateur commence par appeler par téléphone, via un modem, le routeur du fournisseur d'accès.
- Après établissement de la liaison physique le PC envoie des paquets LCP dans le champ de données de trames PPP.
- Les paramètres étant définis, le PC envoie des paquets NCP pour configurer la couche réseau (comme, par exemple, l'attribution d'une adresse IP).
- À la fin de la connexion, on utilise NCP pour libérer la connexion réseau.
- Le PC demande enfin au modem de raccrocher ce qui libère la connexion physique.

# Protocole PPP

## Exemple de fonctionnement

- De nos jours, avec l'ADSL, le protocole PPP est encapsulé dans des trames ethernet : on parle de PPPoE (*PPP over Ethernet*). Il existe également un PPPoA (*PPP over ATM*).
- La connexion avec votre fournisseur d'accès n'est plus réalisée par votre PC mais par votre *box* ADSL.
- On utilise cependant toujours PPP afin de bénéficier de ses services d'authentification et de négociation des options de la couche réseau.

# Plan

# Les réseaux à diffusion

## Les LAN

- Les LAN fondent souvent leur système de communication sur le principe de canal à accès multiple.
- Il s'agit alors de réseaux à diffusion.
- Les protocoles chargés de la gestion des accès au canal partagé appartiennent à une sous-couche de la couche liaison de données : la sous-couche d'accès au canal ou sous-couche MAC (*Medium Access Control*).
- La sous-couche MAC forme la partie inférieure de la couche liaison de données.

# Les réseaux à diffusion

## Les protocoles de sous-couche MAC

- L'IEEE a normalisé un certain nombre de réseaux locaux et métropolitains sous la désignation globale IEEE 802.
- Les LAN filaires ont été normalisés avec la norme IEEE 802.3.
- Les LAN sans fils ont été normalisés avec la norme IEEE 802.11.

# Plan

# Les réseaux à diffusion

## Ethernet

- La norme 802.3 est dénommée Ethernet Standard.
- Le terme Ethernet se réfère au câble (l'éther) utilisé pour les transmissions.

## Ethernet

Les câbles utilisés pour un réseaux Ethernet peuvent être :

- Coaxial épais (obsolète, pas plus de 500 m)
- Coaxial fin (pas plus de 185 m)
- Paire torsadée (machines reliées à l'aide d'un hub ou d'un commutateur, pas plus de 100 m à 200 m)
- Fibre optique (pas plus de 2 km)

# Réseaux ethernet

## Réseaux de grandes tailles

- Plusieurs segments peuvent être reliés par des répéteurs.
- Il ne peut y avoir plus de quatre répéteurs entre deux machines (*transceivers*)
- Deux *transceivers* ne peuvent être éloignés de plus de 2,5 km

# Codage

## Codage Manchester

Afin de pouvoir facilement détecter le début, la fin et le milieu d'un bit, Ethernet utilise le codage de Manchester :

- Le signal de haute tension est à $+0,85$ volt, celui de faible tension est à $-0,85$ volt
- Chaque période représentative d'un bit est divisée en deux intervalles
- Pour coder un bit à 1, on envoie une tension haute dans le premier intervalle et une tension faible dans le suivant
- On fait l'inverse pour un bit à 0

# Les réseaux à diffusion

## Transmission

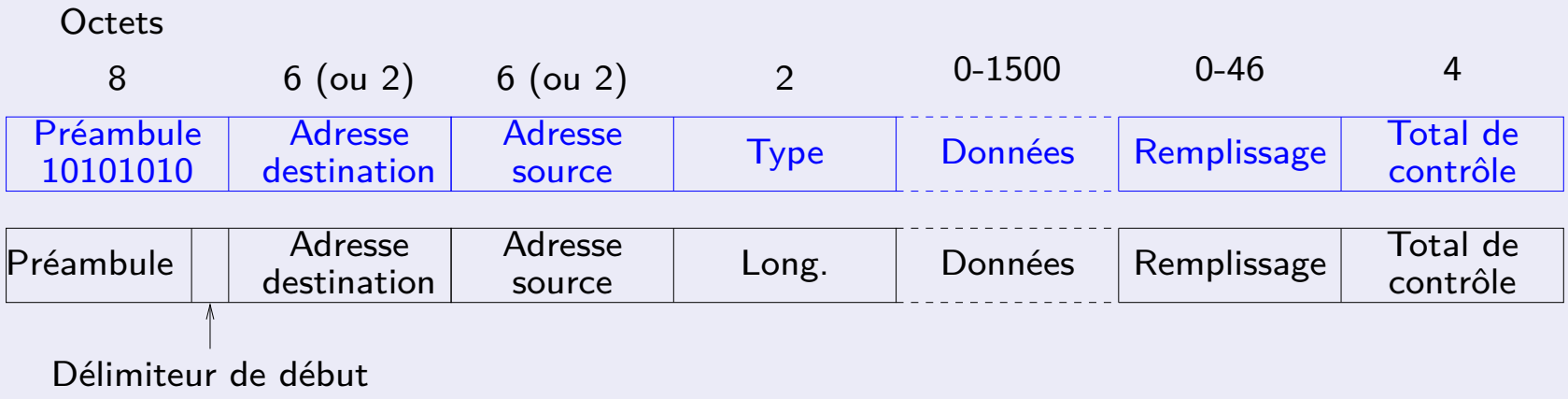
La transmission sur un réseau ethernet est régie par le protocole CSMA/CD (*Carrier Sense Multiple Access with Collision Detection*) :

- Les machines écoutent le support de transmission pour savoir s'il est occupé avant de transmettre
- Une machine qui émet des données écoute en même temps la porteuse afin de détecter des collisions
- Si une collision est détectée il faut continuer à transmettre le temps que toutes les machines détectent la collision
- En cas de collision, il faut attendre un temps aléatoire avant de réessayer

# Les réseaux à diffusion

## Trame Ethernet

Il existe deux formats de trames Ethernet : le format DIX (*DEC-Intel-Xerox*) et le format de la norme IEEE 802.3.



# Trame Ethernet

## Préambule

- Le préambule permet à l'horloge du récepteur de se caler avec celle de l'émetteur. Pour les trames DIX, le codage Manchester produit un signal rectangulaire de 10 $\mathrm{MHz}$ pendant 6,4 $\mu\mathrm{s}$.
- Les parties doivent ensuite rester synchronisées et utiliser le codage Manchester pour repérer les délimitations de bits.
- Dans la norme 802.3, le préambule est ramené à 7 octets afin d'insérer un octet délimiteur de début de trame pour être compatible avec les normes 802.4 (token bus) et 802.5 (token ring).

# Trame Ethernet

## Adresses

- Les adresses peuvent être sur 2 ou 6 octets.
- La norme à 10 $\mathrm{Mbit/s}$ n'emploie que les adresses sur 6 octets : on les appellent adresses MAC.
- 1 bit indique si l'adresse est une adresse unicast ou une adresse multicast/broadcast.
- 1 bit indique si l'adresse est universelle ou locale.
- C'est l'IEEE qui est chargée d'attribuer les adresses globales.
- 22 bits indiquent l'adresse du constructeur (à 0 pour une adresse locale).
- 24 bits pour l'adresse unique.

# Trame Ethernet

## Type

- Le champ type indique le protocole encapsulé par la trame ethernet.
- Dans la norme 802.3 le champ type est remplacé par un champ longueur. On ajoute alors un petit en-tête dans le champ des données afin de spécifier le protocole utilisé.
- En 1997, voyant que le champ de longueur était mal accueillit, l'IEEE annonça que les deux formats étaient valides. Heureusement, tous les champs de type utilisés avant 1997 avaient une valeur supérieure à 1 500.

# Trame Ethernet

## Données

- La longueur maximale du champ des données est de 1500 octets.
- Il s'agit d'une limite arbitraire qui semblait raisonnable en 1978.
- Une trame doit avoir une longueur minimale de 64 octets sans compter les 8 octets du préambule.
- Le champ remplissage peut être utilisé pour compléter une trame trop courte.
- La longueur minimale permet de distinguer les trames valides des trames parasites.
- La longueur minimale permet également d'éviter qu'une station termine sa transmission avant que le début de la trame n'arrive à destination.

# Détection de collisions

## Collisions

- Une collision peut-être détectée par une station si elle mesure une puissance supérieure à ce qu'elle émet.
- Lorsqu'une collision est détectée, la station émet un signal de brouillage de 48 bits afin d'avertir les autres stations.
- Après une collision, les deux stations émettrices attendent un temps aléatoire avant de ré-émettre.

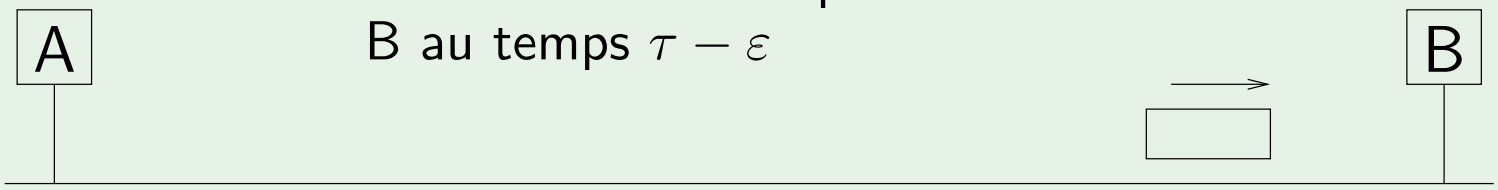
# Exemple

## Exemple de collision

# Exemple

## Exemple de collision

La trame est sur le point d'atteindre B au temps $\tau - \varepsilon$

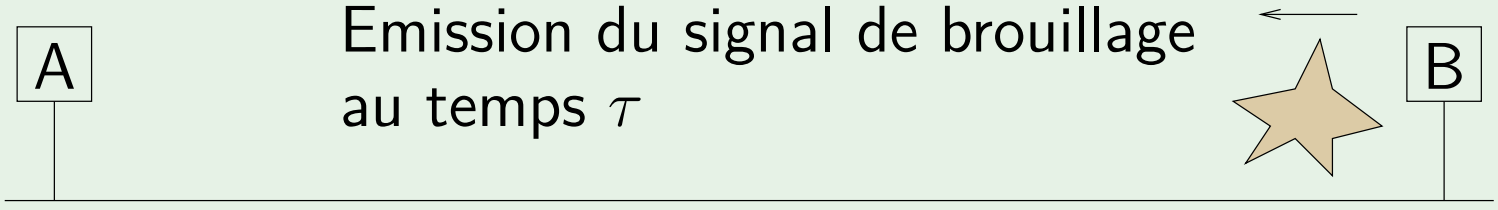


# Exemple

## Exemple de collision

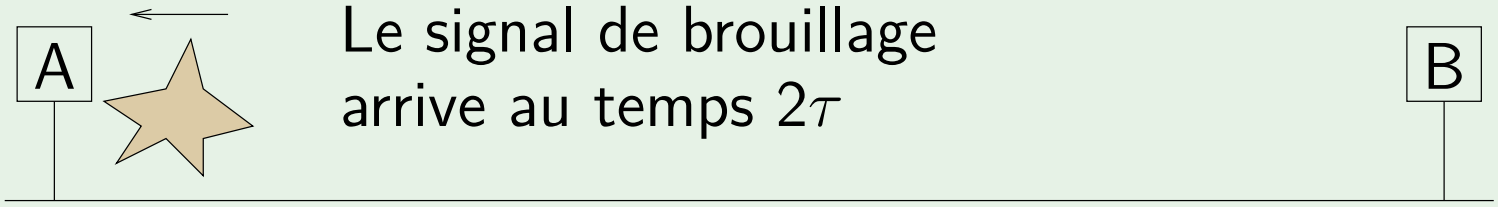
# Exemple

## Exemple de collision



# Exemple

## Exemple de collision




Le signal de brouillage arrive au temps $2\tau$

# Détection de collisions

## Collisions

- Si la trame est trop courte, une station peut avoir déjà terminé sa transmission avant que le signal de brouillage ne lui parvienne.
- La station conclut donc à tort que tout s'est bien passé et que le signal de brouillage concerne une autre transmission.
- Pour éviter cette situation, le temps de transport d'une trame doit être supérieur à $2\tau$.
- Sur un LAN à 10 $\mathrm{Mbit/s}$ d'une longueur maximale de 2500 $\mathrm{m}$ et composé de quatre répéteurs (norme 802.3) : $2\tau \simeq 50$ $\mu\mathrm{s}$.
- À un débit de 10 $\mathrm{Mbit/s}$, le temps prit par un bit est de $0,1$ $\mu\mathrm{s}$, il faut donc des trames de 500 bits. Avec une marge de sécurité, cela donne 64 octets.

# Algorithme stochastique d'attente exponentielle

## Attente aléatoire

- Après une collision, le temps est divisé en slots d'une durée $2\tau$.
- Chaque station attend alors aléatoirement entre 0 et 1 slot avant de ré-émettre.
- S'il se produit encore une collision, les stations attendent alors entre 0 et 3 slots.
- S'il y a toujours des collisions, après $i$ collisions, le temps à attendre est tiré entre 0 et $2^i - 1$
- Le nombre de slots maximal est plafonné à 1023.
- Après 16 collisions, le contrôleur abandonne et annonce l'erreur à la station.

# Détection d'erreurs

## Total de contrôle

- Le total de contrôle est une valeur de 32 bits issue du hachage des données.
- Si certains bits sont erronés alors la somme de contrôle ne correspondra probablement pas.
- L'algorithme utilisé est un algorithme de contrôle de redondance cyclique (*CRC : Cyclic Redundancy Check*).

# Ethernet commuté

## Commutateurs Ethernet

- Afin de répondre à l'augmentation de trafic sur un réseau Ethernet, on peut utiliser l'Ethernet commuté.
- On place alors des commutateurs Ethernet sur le réseau.
- Un commutateur (ou *switch*) dispose d'emplacements pour recevoir généralement de 4 à 32 cartes d'entrée/sortie.
- Chaque carte possède de un à huit connecteurs (ou ports) sur lequel on peut connecter une station.

# Ethernet commuté

## Transmission

- Lorsqu'une station veut émettre, elle envoie une trame Ethernet standard au commutateur.
- La carte d'E/S qui reçoit la trame vérifie si elle se destine à l'une des stations qui lui sont connectées.
- Si oui elle est recopiée dans la mémoire tampon, si non elle est transmise *via* un bus interne à la carte à laquelle la station destinataire est reliée.

# Ethernet commuté

## Domaine de collision

Il existe deux stratégies de gestion des cartes d'E/S :

- Chaque carte peut former un LAN local avec son propre domaine de collision.
- Chaque port peut disposer de sa propre mémoire tampon et former ainsi un domaine de collision distinct. Si les transmissions se font en *full duplex* alors il ne peut plus y avoir de collisions.

# Duplex

## Définition

- Un canal de communication **simplex** est un canal qui transporte l'information dans un seul sens ;
- Un canal de communication **half duplex** est un canal qui transporte l'information dans les deux sens mais pas simultanément ;
- Un canal de communication **full duplex** est un canal qui transporte l'information dans les deux sens.

# Fast Ethernet et Gigabit Ethernet

## LAN plus rapides

Il existe de nombreux ajouts ajouts à la norme 802.3 permettant de gérer des réseaux plus rapides.

- Fast Ethernet a été publié en 1995 sous le nom 802.3u.
- Gigabit Ethernet a été publié en 1998 sous le nom 802.3z.
- 10 Gigabit Ethernet a été publié en 2002 sous les noms 802.3ae, 802.3an, ....
- 25 Gigabit Ethernet a été publié en 2017 sous le nom 802.3cc.
- ...

# Fast Ethernet et Gigabit Ethernet

## Caractéristiques

- 802.3u et 802.3z utilisent uniquement des câbles à paires torsadés ou de la fibre optique.
- Ils reposent sur l'utilisation des hubs (transmission half duplex) et des commutateurs (transmission full duplex).
- Fast Ethernet autorise un débit à 100 $\mathrm{Mbits/s}$ et Gigabit Ethernet un débit de 1 $\mathrm{Gbits/s}$.

## Encore plus rapide

Il existe une norme pour l'Ethernet à 10 $\mathrm{Gbits/s}$ : 802.3ae. Cette norme repose sur l'utilisation de switchs et de transmission full duplex : il n'y a plus de gestion des collisions.

# Plan

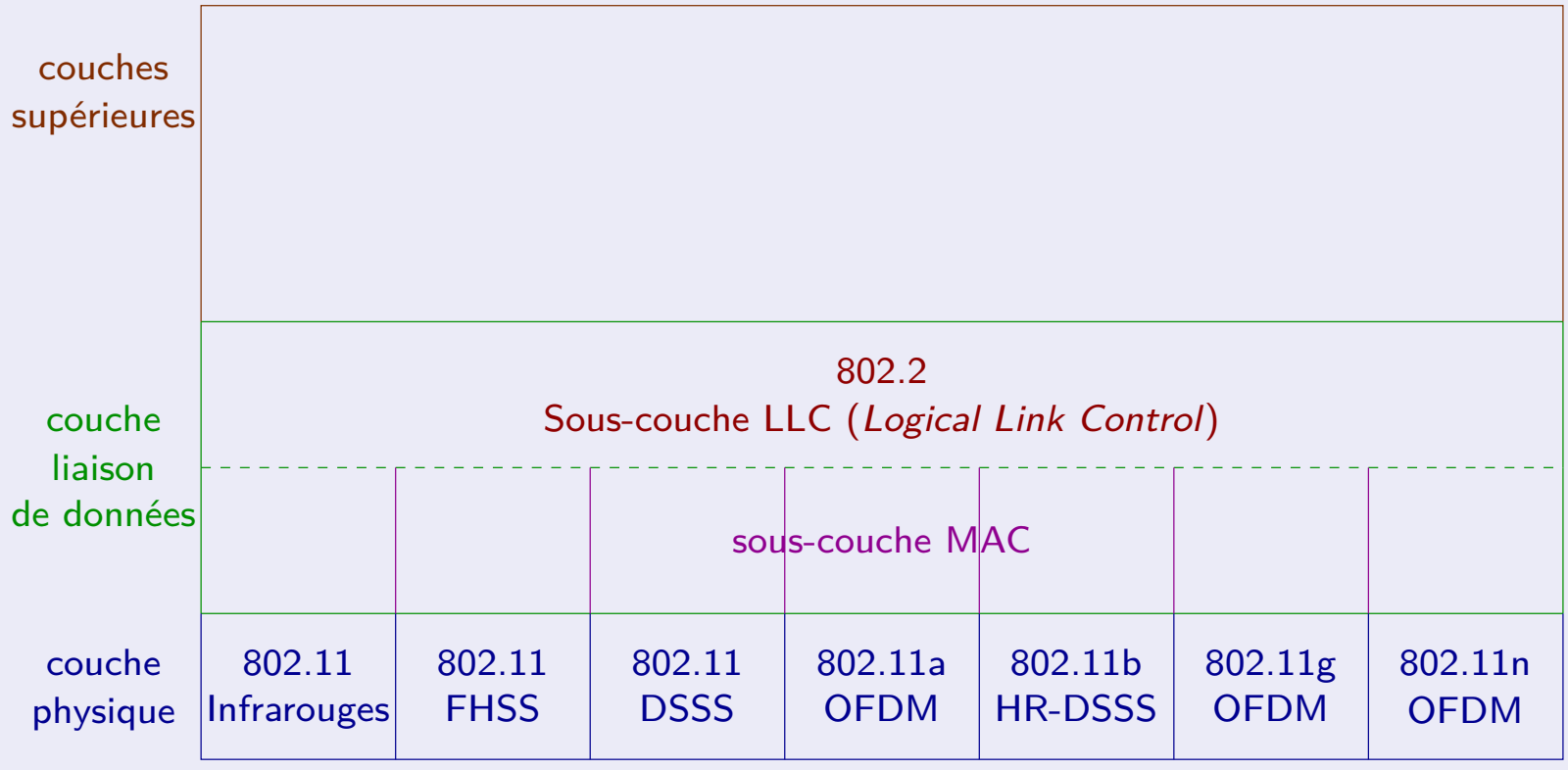
# LAN sans fil

## Protocoles 802.11

- La norme 802.11 définit plusieurs protocoles pour la couche physique des réseaux sans fil.
- Chaque couche physique a une sous-couche MAC qui détermine la façon dont le canal est alloué (*i.e.* quelle machine est autorisée à émettre).
- Au dessus de la sous-couche MAC, se trouve la sous-couche LLC assurant un contrôle de flux et d'erreur.

# Extrait de la pile de protocoles 802.11

## Pile de protocoles



# Accès à la porteuse

## Gestion des collisions

- Lorsqu'un récepteur reçoit le signal de deux émetteurs simultanément, le signal est altéré et inutilisable.
- On ne peut utiliser le protocole CSMA/CD d'ethernet car l'important n'est pas de détecter si on peut émettre mais plutôt de savoir si le récepteur peut recevoir.

# Sous-couche MAC et réseaux sans fil

## Problème de la station cachée

Supposons deux stations A et B souhaitant communiquer avec une station C. C est dans la zone de portée de A et B mais A est hors de portée de B.

- A écoute la porteuse
- A émet une trame vers C
- B écoute la porteuse
- Comme B ne détecte pas l'émission de A, B émet une trame vers C
- Il se produit une collision sans que B ne la détecte

# Sous-couche MAC et réseaux sans fil

## Problème de la station exposée

B souhaite envoyer une trame vers C. Une machine A est dans la zone de portée de B mais hors de portée de C.

- A émet une trame vers une station tierce
- B écoute la porteuse
- Comme B détecte l'émission de A, B pense qu'elle ne peut émettre vers C
- C étant hors de portée de A, elle pourrait très bien recevoir la trame de B

# Sous-couche MAC et réseaux sans fil

## Modes de fonctionnement

La norme 802.11 accepte deux modes de fonctionnement :

- Le mode DCF (*Distributed Coordinated Function*) qui ne fait appel à aucune entité de contrôle (comme Ethernet).
- Le mode PCF (*Point Coordination Function*) qui utilise la station de base pour contrôler l'activité de la cellule.

Toutes les implémentations doivent accepter le mode DCF mais, par contre, le mode PCF est facultatif.

# Sous-couche MAC et réseaux sans fil

## Mode DCF

Le mode DCF utilise le protocole CSMA/CA (*Carrier Sense Multiple Access with Collision Avoidance*).

- La station émettrice écoute le canal et transmet si aucune activité n'est perçue durant un certain laps de temps.
- Si la porteuse est libre, la station émet une trame RTS (*Ready To Send*) contenant des informations sur ce qu'il y a à transmettre.
- Le récepteur envoie une trame CTS (*Clear To Send*) prévenant qu'une station va émettre.

## Mode DCF

- La station émettrice ne peut plus écouter le canal lorsqu'elle émet, elle envoie donc la trame entière.
- À la fin de la transmission, le récepteur envoie un ACK.
- Lorsqu'une station (non émettrice) reçoit une trame RTS ou une trame CTS, elle mémorise l'occupation du canal à l'aide d'un signal (virtuel) d'allocation de réseau (*NAV*, Network Allocation Vector).
- Si une collision se produit, les stations utilisent l'algorithme stochastique d'attente avant de ré-émettre.

# Sous-couche MAC et réseaux sans fil

## Erreurs de transmission

- La probabilité de réussite de transmission d'une trame décroit fortement lorsque la taille de la trame grandit (elle est de la forme $(1-p)^n$ pour une trame de taille $n$ et une probabilité de recevoir un bit erroné $p$).
- Le protocole 802.11 autorise les trames à être fragmentée en portions plus petites, chacune avec son total de contrôle. Chaque fragment est alors numéroté individuellement et acquitté à l'aide d'un protocole du type « arrêt et attente » (*stop-and-wait*).

## Modes PCF

- Le mode PCF met en œuvre une station de base qui invite les stations à émettre (*polling*) en leur demandant si elles ont des trames à émettre. L'ordre d'émission étant régulé par la station de base, il ne se produit pas de collisions.
- La station de base émet régulièrement une une trame de signalisation (*beacon frame*). Elle se charge aussi de rappeler aux stations de s'inscrire sur la liste des stations invitées à émettre.

# Réseaux sans fil

## Services des LAN sans fil

Les LAN sans fil doivent fournir 5 services de distribution

1. Association : permet aux stations mobiles de se connecter aux stations de base
2. Dissociation : permet aux stations de rompre une association
3. Réassociation : permet à une station mobile de changer de station de base
4. Distribution : spécifie comment les trames à destination d'une station de base sont routées
5. Intégration : gère la traduction du format 802.11 au format requis par le réseau destination

# Réseaux sans fil

## Services des LAN sans fil

Les LAN sans fil doivent fournir 4 services aux stations (intra-cellulaire)

1. Authentification : permet aux stations de s'authentifier les unes auprès des autres
2. Annulation d'authentification : permet d'annuler l'authentification d'une station quittant le réseau
3. Confidentialité : permet le chiffrement et le déchiffrement des données transmises
4. Livraison de données : permet la transmission et la réception des données

# Trame 802.11

## Trame 802.11

| Octets | 2 | 2 | 6 | 6 | 6 | 2 | 6 | 0-2312 | 4 |
|---|---|---|---|---|---|---|---|---|---|
| | Contrôle de trame | Durée | Adresse 1 | Adresse 2 | Adresse 3 | Séq. | Adresse 4 | Données | Total de contrôle |

# Trame 802.11

## Trame 802.11

- Le contrôle de trame indique les propriétés et la fonction de la trame.
- Le champ durée contient des informations relatives à la durée de la trame, à l'évitement de collision ou encore à l'association.
- Une trame peut contenir jusqu'à quatre adresses : celle du destinataire, celle de la source de la trame, la prochaine station devant recevoir la trame et la quatrième indique la station qui a émis la trame.
- Le champ de contrôle de séquence contient un identifiant de fragmentation ainsi qu'un identifiant de séquence.

# Plan

# ATM

## Principes

- ATM (*Asynchronous Transfer Mode*) est un protocole couvrant les couches 1 à 3 du modèle OSI.
- Il permet de multiplexer différents flots de données sur un même lien en espaçant les données dans le temps.
- Il est très utilisé dans les réseaux de type DSL.

# ATM

## Les cellules

- Les données à transmettre sont « découpées »afin d'être émises dans des cellules de taille fixe.
- Chaque cellule contient 53 octets : 5 octets d'en-tête et 48 de charge utile.
- Les cellules sont insérées dans un flux synchrone de cellules (chaque station ATM émet des données en continu).

# ATM

## Les circuits virtuels

- ATM utilise le concept de circuit virtuel : toutes les cellules d'un même message passeront par la même route.
- Les circuits virtuels peuvent être permanents (établis une fois pour toute) ou commutés (établis à la connexion).
- Chaque cellule contient un couple VPI/VCI (*Virtual Path Identifier/Virtual Channel Identifier*) qui peut varier lors de la traversée des routeurs ATM.
- Chaque VPI peut-être constitué de plusieurs VCI, ce qui permet par exemple, d'associer un VCI pour la vidéo, un autre pour la téléphonie, *etc*.

# ATM

## Les adresses ATM

- Les adresses ATM ne sont utilisées que lors de la connexion, ensuite seuls les VPI/VCI sont utilisés.
- Une adresse ATM contient 20 octets et peut être de trois formats différents que nous ne détaillerons pas ici.

# ATM

## Protocoles

ATM utilise deux protocoles différents selon le type de communication :

- Le NNI (*Network to Network Interface*) permet la communication entre deux nœuds du réseau (par exemple entre deux switchs).
- L'UNI (*User to Network Interface*) permet les échanges entre les nœuds et les extrémités (par exemple entre la station d'un utilisateur et le réseau d'un fournisseur d'accès).

# ATM

## Les types de services

ATM dispose d'une couche AAL (*ATM adaptation layer*) permettant son utilisation avec d'autres protocoles de transfert de données :

- Type 1 : débits constants, synchrones et orientés connexion.
- Type 2 : débits variables dépendant du temps, synchrones et orientés connexion (comme pour la voix).
- Type 3/4 : débits variables, asynchrones et orientés connexion ou non.
- Type 5 : similaire au type 3/4 mais en supposant que les données sont envoyées de manière séquentielles. La fin d'un paquet est alors déterminée par un bit présent dans l'en-tête des cellules.
- AAL 5 permet, entre autre, l'IP over ATM et l'Ethernet Over ATM.

# Plan

# Adresse de liaison d'une socket

## Obtenir l'adresse à laquelle est liée une socket

La fonction *getsockname* permet de récupérer l'adresse à laquelle est liée une socket :

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

- *addrlen* doit contenir en entrée la taille du tampon pointé par *addr*
- En sortie *addrlen* contient la taille de l'adresse stockée dans **addr*
- L'adresse est tronquée si le tampon pointé par *addr* est trop petit : *addrlen* aura alors en sortie une valeur plus grande que sa valeur d'entrée.

# Adresse de liaison d'une socket

**Exemple**

Voici un exemple d'utilisation de *getsockname* : voir le source.

# Fixer un délai d'attente maximum lors d'une communication

## Options de socket et délai d'attente maximal

Afin de fixer un *timeout* pour les opérations sur les sockets, on peut utiliser la fonction `setsockopt` afin de fixer un délai d'attente maximal d'émission ou de réception :

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

# `setsockopt` et *timeout*

## Options de cocket et délai d'attente maximal

- Pour un *timeout* concernant les opérations d'écriture, on fixera *optname* à *SO_SNDTIMEO*.
- Pour les opérations de lecture, nous fixerons *optname* à *SO_RCVTIMEO*.
- *level* doit être fixé à *SOL_SOCKET*.
- *optval* sera un pointeur sur une structure `struct timeval`.
- En cas de délai dépassé, les fonction retournent `-1` et *errno* vaudra *EAGAIN* ou *EWOULDBLOCK* : il faut tester les deux car POSIX ne spécifie pas que les deux macros ont la même valeur ni laquelle des deux est retournée.

## Exemple

Voici un exemple de socket avec *timeout* : voir le source.

# Attente passive d'opérations sur des descripteurs

## L'appel système *select*

Lorsqu'on a besoin d'un serveur dont le traitement des requêtes client n'est pas gourmand en ressources et que l'isolation des traitements des requêtes n'est pas une nécessité, il peut être intéressant de traiter les demandes de connexions ainsi que les requêtes client à l'intérieur d'une même boucle de traitement. Pour cela, nous pouvons utiliser, par exemple, l'appel système *select* :

```
int select(int nfds,
           fd_set *readfds, fd_set *writefds,
           fd_set *exceptfds,
           struct timeval *timeout);
```

# Attente passive d'opérations sur des descripteurs

## L'appel système *select*

*select* permet de placer le processus en attente que l'une des opérations attendues sur les descripteurs spécifiés puissent être réalisée.

- *nfds* contient la valeur du plus grand des descripteurs à surveiller **plus 1** ;
- *readfds*, *writefds* et *exceptfds* correspondent aux trois ensembles de descripteurs à surveiller : le premier pour les opérations de lecture, le second pour les opérations d'écriture et le dernier pour la survenue de conditions exceptionnelles (comme des données urgentes en attente de lecture).
- Les macros *FD_ZERO*, *FD_SET* et *FD_CLR* permettent de manipuler les ensembles de descripteurs.
- *timeout* permet de fixer un délai maximal d'attente.

# Attente passive d'opérations sur des descripteurs

**Exemple**

Voici un exemple d'utilisation de *`select`* : voir le source.