

Introduction aux réseaux et à Internet.

Programmation réseau

email : kadionik@enseirb-matmeca.fr
web : kadionik.enseirb-matmeca.fr

Patrice KADIONIK
ENSEIRB-MATMECA

RE223 : Introduction aux réseaux et à Internet...



PARTIE 1

OBJECTIFS DU COURS

RE223 : Introduction aux réseaux et à Internet...



Objectifs

- Introduction aux réseaux de télécommunication, à Internet et à ses protocoles de communication associés.
- Introduction à la programmation réseau.
- Complément en informatique technique : maîtrise de la programmation (en langage C) des *sockets*. Perfectionnement du langage C.

Objectifs

- La maîtrise de l'utilisation des *sockets* liées à Internet impose naturellement d'aborder les thèmes suivants :
 - Introduction aux réseaux. Le modèle OSI.
 - Introduction à Internet et à Internet embarqué.
 - Introduction à l'Internet des objets.
 - Introduction à la norme Ethernet.
 - Présentation des protocoles Internet : IP, TCP, UDP...
 - Architecture client/serveur.
 - Programmation réseau avec l'API *sockets*.

API=*Application Programming Interface*

RE223 : Introduction aux réseaux et à Internet...



PARTIE 2

INTRODUCTION AUX RESEAUX

RE223 : Introduction aux réseaux et à Internet...



A quoi servent les réseaux ?

- Partage de ressources :
 - Programmes.
 - Equipements.
- Echange de données :
 - Fiabilité.
 - Systèmes distribués.

Topologie

- Nœud : ordinateur, commutateur, routeur...
- PAN : *Personal Area Network* :
 - Inférieur à 10-100 mètres : Bluetooth, réseaux de capteurs...
- LAN : *Local Area Network* :
 - De 100 mètres à quelques kilomètres : Ethernet...
- MAN : *Metropolitan Area Network* :
 - De 10 à 100 km : réseau fibre optique...
- WAN : *Wide Area Network* :
 - De 100 km à la terre entière : Internet, Réseau Téléphonique Commuté (RTC), ADSL...

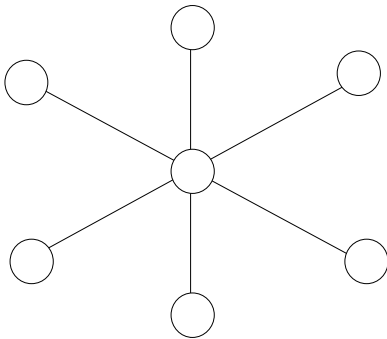
Topologie

- 2 grandes familles de topologie adaptées au transfert de l'information :
 - Transmission en mode point à point :
 - Téléphone filaire...
 - Diffusion :
 - Radio, télévision...

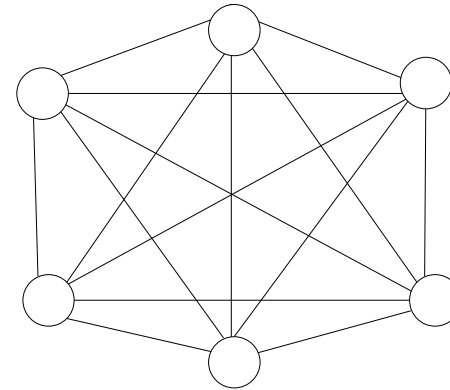
Topologie

- Réseaux point à point :

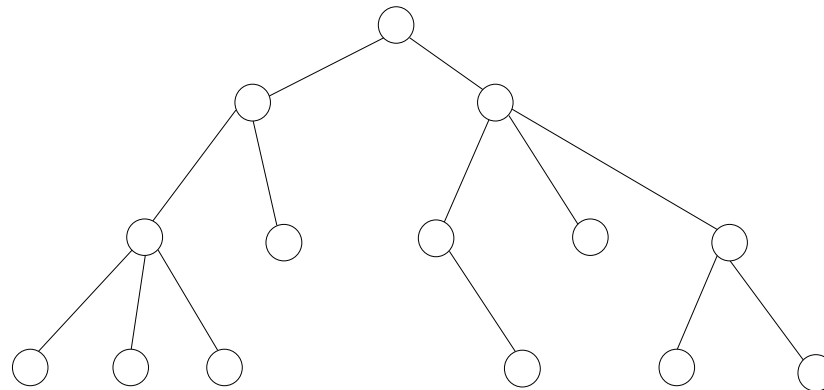
Etoile



Maillage régulier



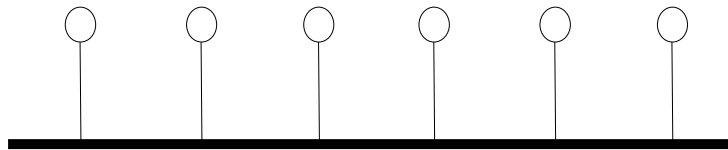
Arbre



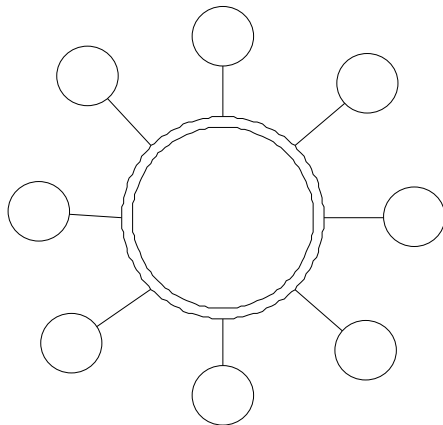
Topologie

- Réseaux multipoint :

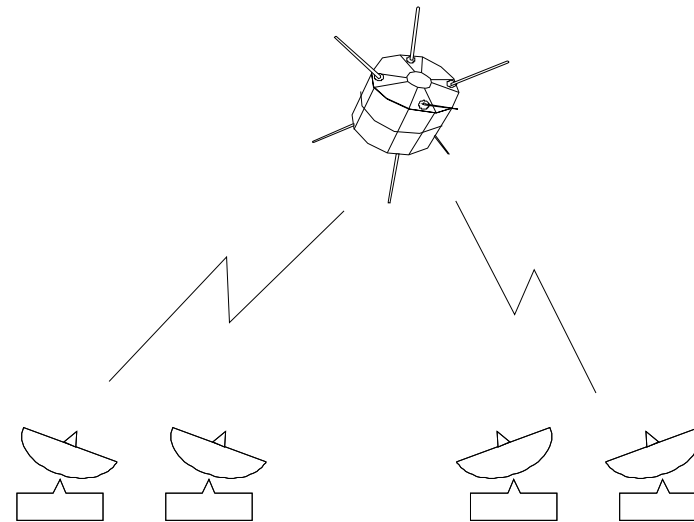
Bus



Anneau



Satellite



Quelques chiffres

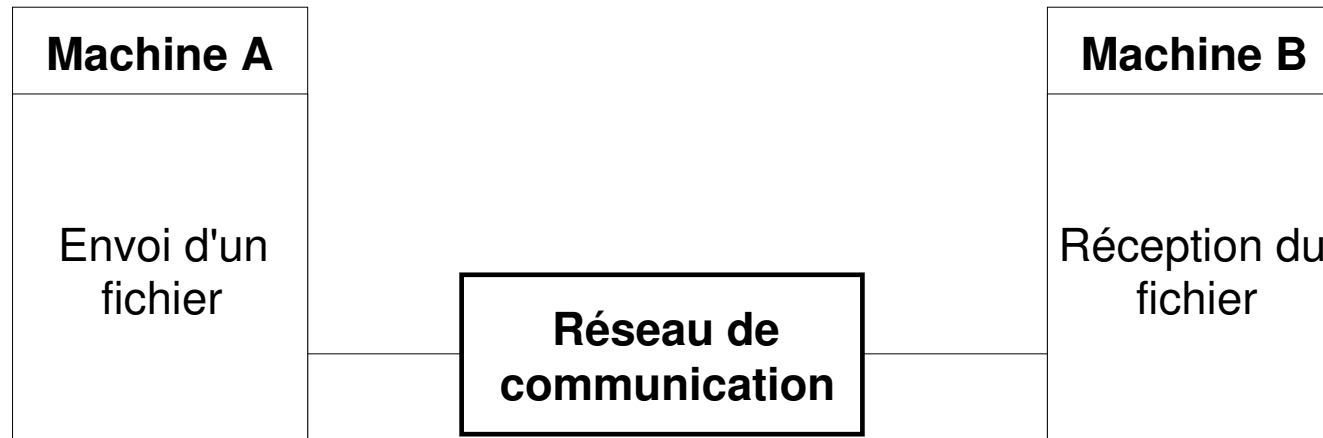
- Unités :
 - baud (Bd) : rapidité de modulation.
 - bit par seconde (b/s) : débit de source : kb/s, Mb/s, Gb/s, Tb/s.
- Attention !!! :
 - 1 octet = 8 bits.
 - 1 K = 1024 (2^{10}).
 - 1 k = 1000.
- Dans les télécoms, on utilise préférentiellement les puissances de 10.

Quelques chiffres

- Connexion série sur un PC : de 75 b/s à 115200 b/s.
- Connexion Internet par modem bas débit sur RTC : de 300 à 56 kb/s.
- Réseau local : 10 Mb/s à 1000 Mb/s.
- Epines dorsales des réseaux (*backbone*) : de quelques centaines de Mb/s à quelques centaines de Gb/s !

Le besoin de normalisation

- Pourquoi des protocoles et des architectures de réseaux ?



Le besoin de normalisation

- Besoin de définir des protocoles normalisés ou standardisés afin d'assurer l'interopérabilité. Seule, l'implémentation peut changer...
- Standards de fait ou définis par des organismes privés.
- Organismes les plus connus :
 - ISO : *International Organisation for Standardisation.*
 - ITU : *International Telecommunication Union.*
 - IEEE : *Institute of Electrical and Electronic Engineers.*
 - AFNOR.

PARTIE 3

LE MODELE OSI

Qu'est-ce que l'ISO ?

- L'organisation internationale de normalisation ISO est une fédération mondiale d'organismes nationaux de normalisation de quelque 140 pays à raison d'un organisme par pays.
- L'ISO est une organisation non gouvernementale créée en 1947. Elle a pour mission de favoriser le développement de la normalisation et des activités connexes dans le monde en vue de faciliter entre les nations les échanges de biens et de services et de développer la coopération dans les domaines intellectuel, scientifique, technique et économique.

<http://www.iso.org>



Le modèle OSI

- Dans les années 70 apparaît la nécessité de systèmes (matériel et logiciel) ouverts par opposition aux systèmes propriétaires (IBM, BULL, DEC...).
- Caractéristiques d'un système ouvert :
 - *Portabilité.*
 - *Interopérabilité.*
- Le but d'un modèle est de proposer aux constructeurs un schéma sur lequel ils pourront bâtir leurs solutions matérielles et logicielles. En s'appuyant sur un modèle normalisé, ils s'assurent d'un produit ouvert aux autres systèmes qui s'appuient sur la même norme.

Le modèle OSI

- 1977 : l'ISO démarre une réflexion sur une architecture de réseau en couches.
- 1984 : définition du modèle OSI :
 - *Open* : systèmes ouverts à la communication avec d'autres systèmes.
 - *System* : ensemble des moyens informatiques (matériel et logiciel) contribuant au traitement et au transfert de l'information.
 - *Interconnection*.
- Modèle d'architecture de réseau : le modèle OSI !

OSI=*Open System Interconnection*

Terminologie

- Le modèle OSI repose sur trois concepts importants :
 - Les couches.
 - Les protocoles.
 - Les interfaces.
- On distingue deux grands groupes de couches :
 - Les couches 1 à 3 sont les couches basses orientées transmission des données.
 - Les couches 4 à 7 sont les couches hautes orientées traitement des données.

Terminologie

- Chaque couche va rendre des services à la couche immédiatement supérieure et utiliser les services de la couche immédiatement inférieure. Les couches ne communiquent qu'avec les couches qui leur sont adjacentes.
- Les protocoles de communication sont les règles qui définissent le dialogue entre couches de même niveau de deux systèmes différents. Les règles et conventions utilisées lors du dialogue entre deux couches n sont appelées protocole de communication de couche n .
- Une interface est donc un ensemble de services proposés par les couches aux autres couches.

Architecture en couches

- Le concept de couches s'impose en informatique dès qu'il s'agit de subdiviser les tâches d'un système.
- Il est utilisé aussi dans les systèmes d'exploitation.
- On le retrouve dans la conception des réseaux.

Architecture en couches

- Chaque couche offre des services à celle qui lui est directement supérieure.
- Elle lui masque les détails de son implémentation.
- Elle utilise les services de la couche directement inférieure.

Couche inférieure = plus proche du matériel.

Couche supérieure = plus proche des applications.

Avantages des systèmes en couches

- Décomposition en modules relativement simples.
- Possibilité de modifier un module sans devoir adapter les autres.
- Abstraction de la complexité d'un module aux concepteurs des autres modules.
- Il est inutile de comprendre tous les détails pour pouvoir comprendre l'ensemble.
- Développement, corrections, modifications et évolutions facilités...

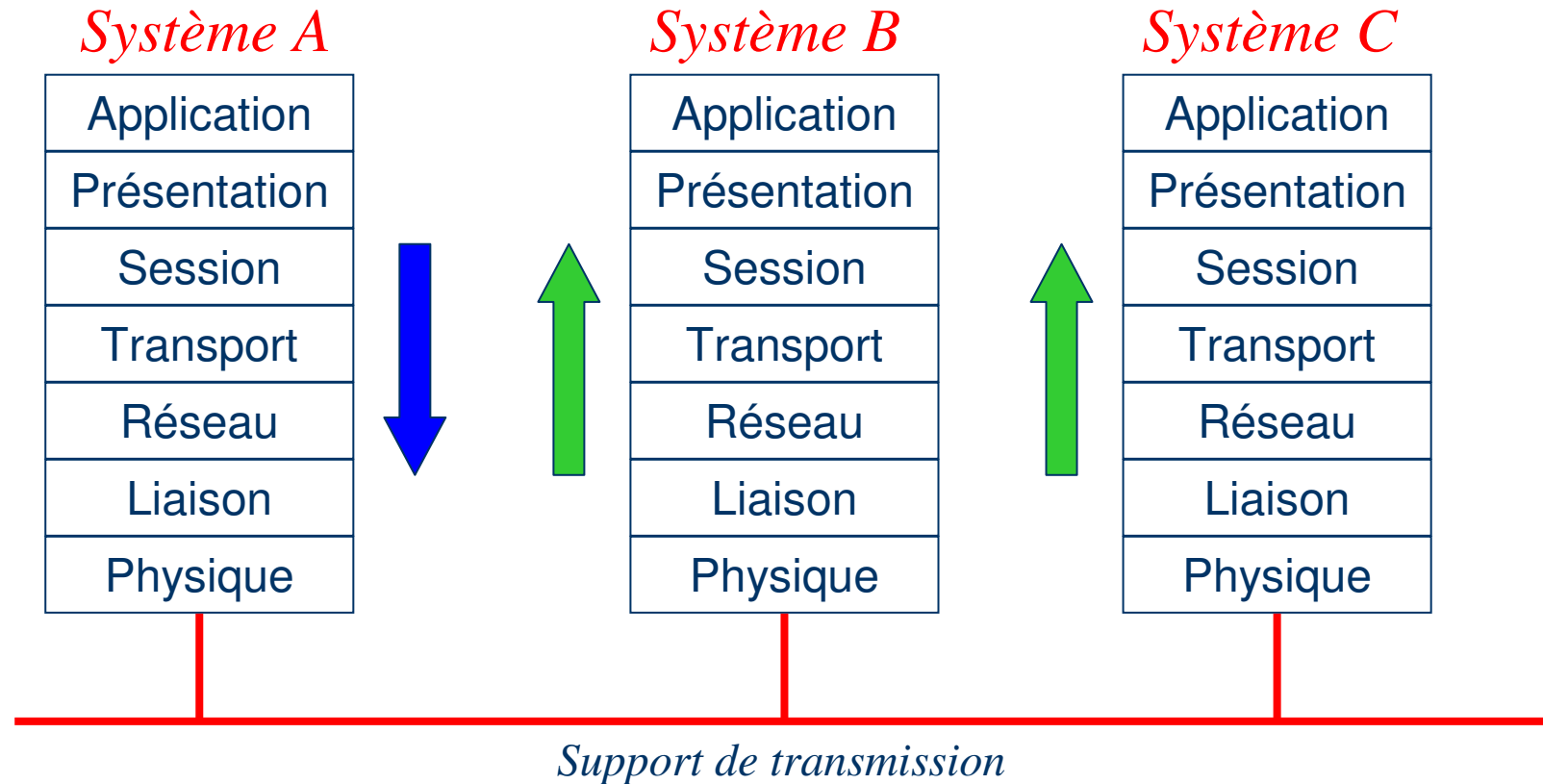
Protocole

- Ce sont les conventions entre entités pour échanger des données.
- Le protocole gère des informations de contrôle qui accompagnent les blocs de données.
- But :
 - Identification du début et de la fin de chaque élément d'un bloc.
 - Fonctions de commandes telles que l'initialisation, l'interrogation, l'identification des équipements.
 - Détection des erreurs de transmission.

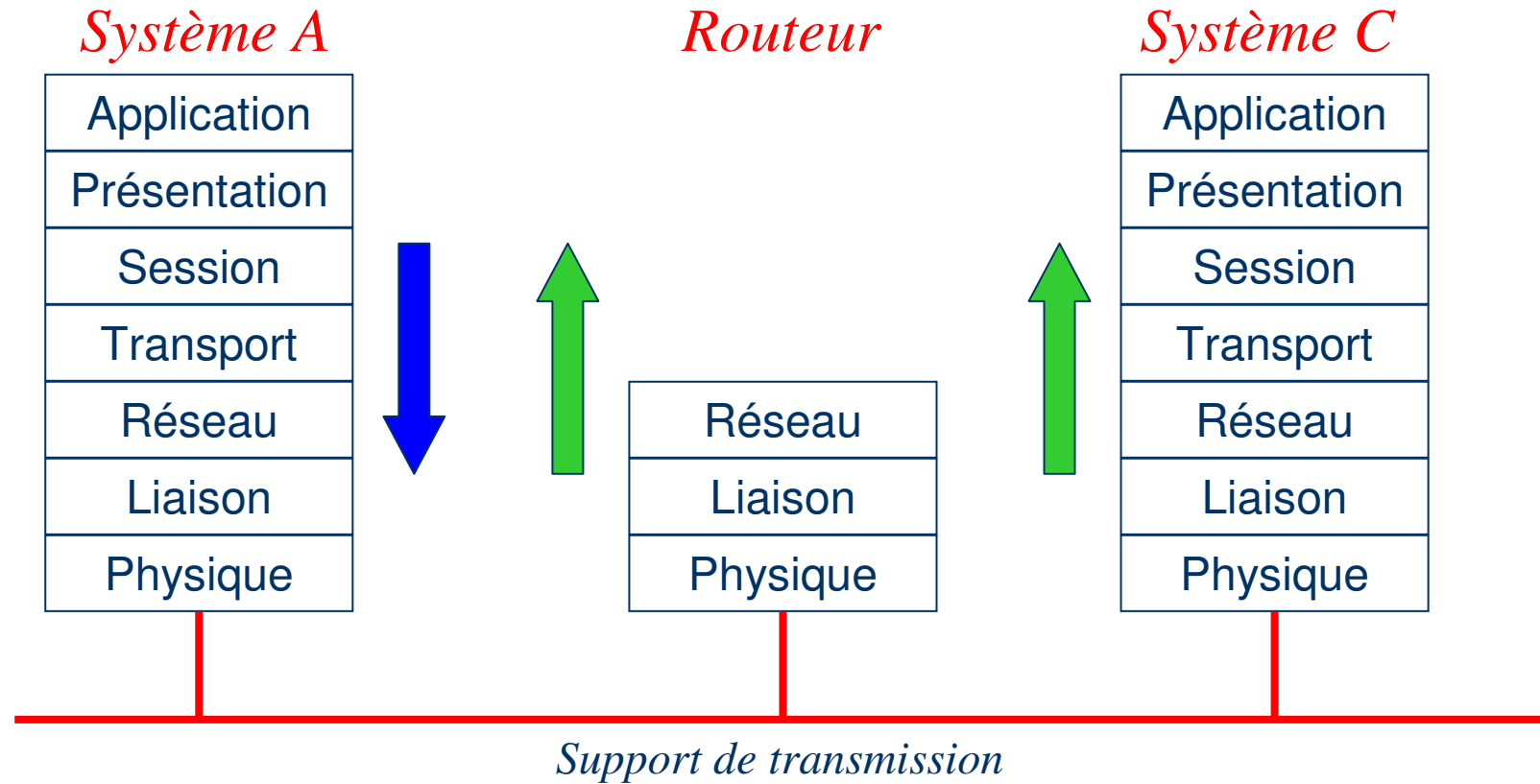
Les 7 couches du modèle OSI

- 7 Application
- 6 Présentation
- 5 Session
- 4 Transport
- 3 Réseau
- 2 Liaison
- 1 Physique

Les 7 couches du modèle OSI



Le modèle OSI. Routeur



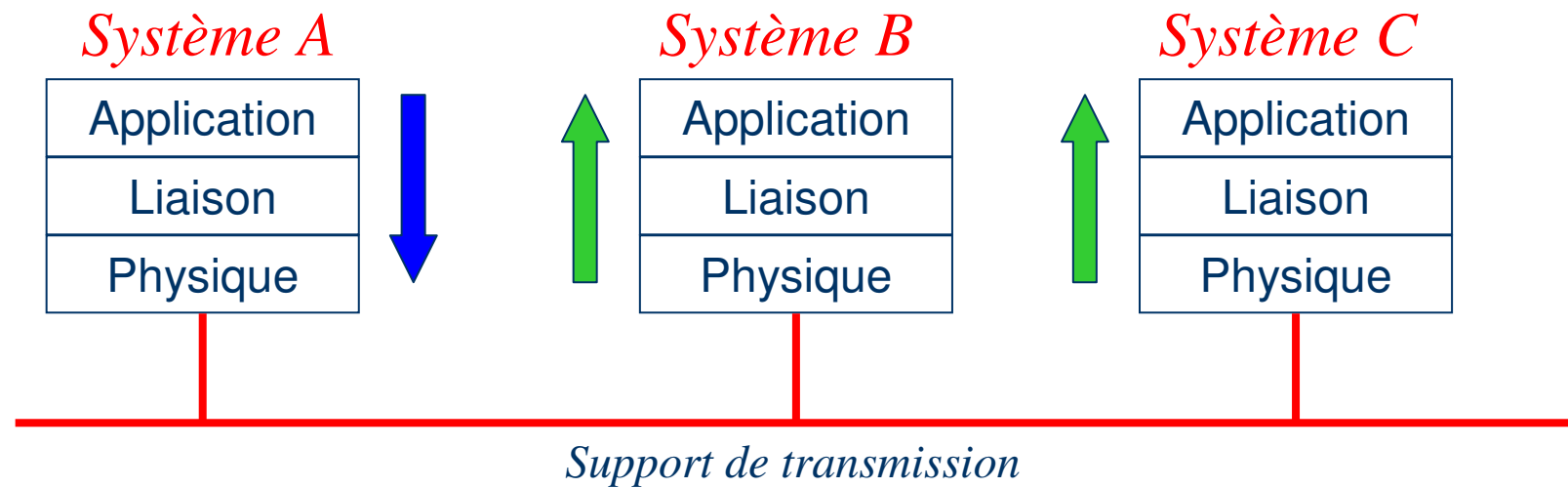
Le modèle OSI. Bus de terrain

- Les bus de terrain sont des réseaux utilisés pour les automatismes où le respect des contraintes temporelles est important (ex : bus CAN...).

- 7	Application	Spécifié par l'utilisateur
- 6	Présentation	(vide)
- 5	Session	(vide)
- 4	Transport	(vide)
- 3	Réseau	(vide)
- 2	Liaison	Protocole CAN...
- 1	Physique	Libre choix

CAN=*Control Area Network*

Le modèle OSI. Bus de terrain



Couche 1. Couche physique



- Elle décrit les caractéristiques électriques, logiques et physiques de la connexion de la station au réseau, c'est à dire tout ce qui concerne les câbles, les connecteurs et les cartes réseau.
- Elle définit les aspects physiques du raccordement : interfaces mécanique et électrique et protocole d'échange des éléments binaires : caractéristiques physiques et électriques du support de transmission (paire torsadée...), méthode de transmission (bande de base...), débits et types de transmission (synchrone / asynchrone).

Couche 1. Couche physique

- En résumé : tout ce qui constitue le support physique qui assure le transport des données.
- **L'unité de données est le bit.**

Couche 2. Couche liaison



- Son rôle est de définir des règles pour l'émission et la réception de données à travers la connexion physique de deux systèmes :
 - Transmettre les données sans erreurs.
 - Déterminer la méthode d'accès au support de transmission.
- La couche liaison utilise des protocoles d'accès au support qui peuvent être déterministes ou probabilistes.

Couche 2. Couche liaison

- En résumé : contrôle la transmission des données afin de les transmettre sans erreur.
- **L'unité de données est la trame.**

Couche 3. Couche réseau



- Elle gère l'acheminement des données à travers le réseau en assurant le routage des paquets de données entre les nœuds du réseau. Si un nœud est surchargé ou hors service, les données seront alors routées vers un autre nœud.
- Elle assure l'opération d'adressage et de routage. Elle assure également le contrôle des flux au niveau des nœuds (engorgement, perte de paquets...).

Couche 3. Couche réseau

- En résumé : responsable de l'acheminement des données de manière à ce qu'elles arrivent à la bonne adresse.
- **L'unité de données est le paquet.**

Couche 4. Couche transport



- Elle est responsable du contrôle du transport de bout en bout à travers le réseau. Elle assure les fonctions d'adressage, de découpage et de réassemblage des données.
- La qualité de service (QoS) est souvent utilisée pour décrire l'utilité de la couche transport.
- La couche transport de l'émetteur fragmente les messages de données en paquets et la couche transport du récepteur reconstitue les messages en défragmentant les paquets dans le bon ordre.
- Elle permet également de multiplexer plusieurs flux d'informations sur le même support.

QoS=*Quality of Service*

Couche 5. Couche session



- Première couche orientée traitement. Elle permet l'ouverture et la fermeture d'une session de travail entre deux systèmes distants.
- Elle a pour rôle la mise en place et le contrôle du dialogue entre les tâches distantes : connexion, gestion, déconnexion... Elle assure la synchronisation du dialogue entre les tâches distantes.

Couche 6. Couche présentation



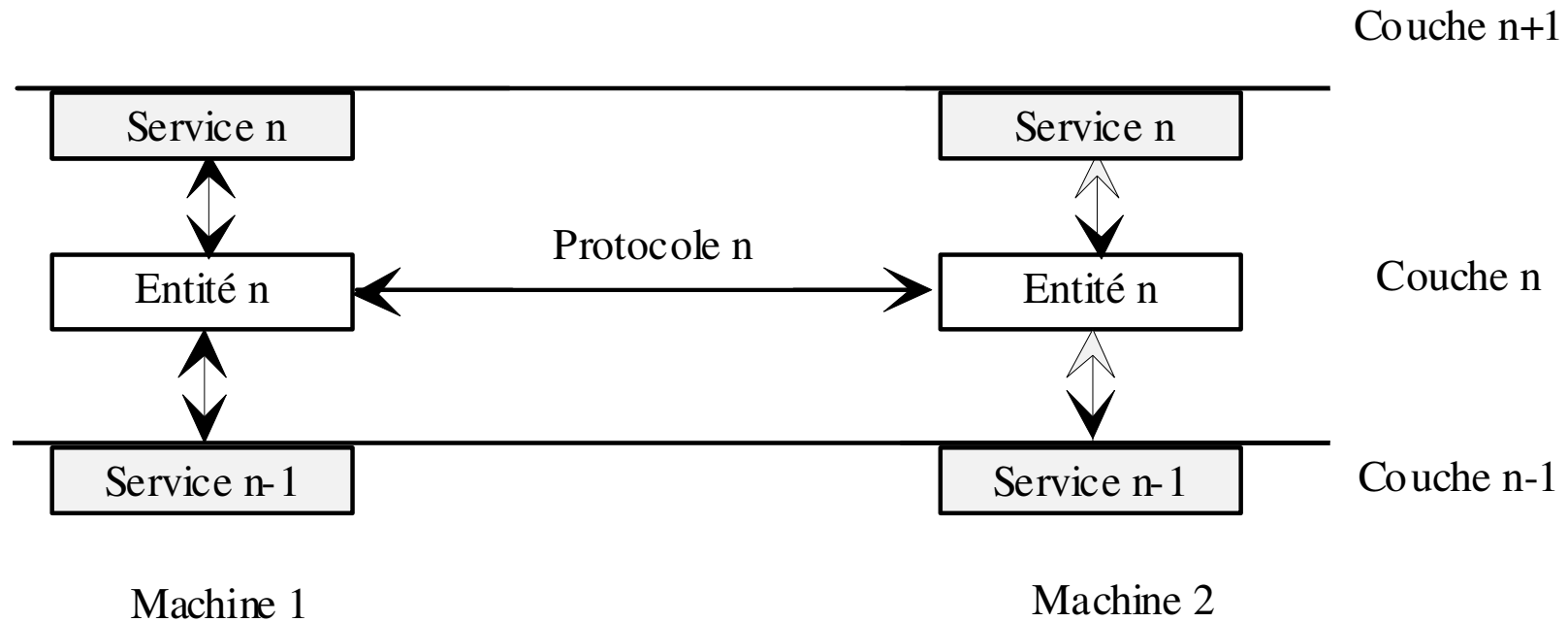
- La couche présentation assure trois fonctions principales :
 - Le formatage des données (présentation).
 - Le chiffrement des données.
 - La compression des données.
- Permet de formater les données dans un format compréhensible par les 2 systèmes distants.

Couche 7. Couche application



- La couche application crée une interface directe avec le reste du modèle OSI par le biais d'applications réseau normalisées ou non : navigateur Web, messagerie électronique, ftp, telnet...) ou une interface indirecte par le biais d'applications autonomes (traitement de texte, tableurs...).

Service, entité, protocole



Service, entité, protocole

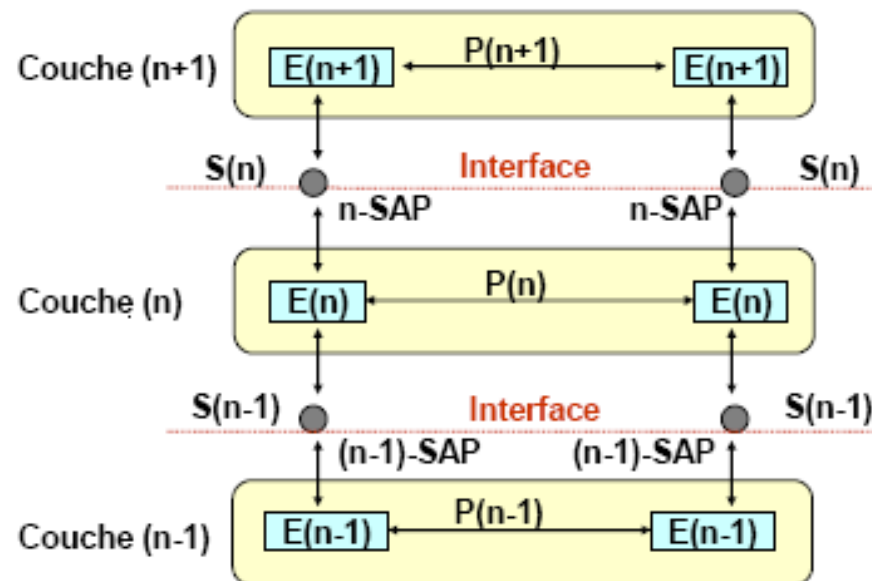
- Exemples de services :
 - Connexion.
 - Echange de données.
 - Déconnexion.

Service, entité, protocole

- Une demande de service se fait par le biais d'une primitive.
- 4 primitives de service sont définies pour permettre à un utilisateur du service de s'adresser à une entité ou à une entité de répondre à un utilisateur de service :
 - Emission : primitive de demande d'émission : REQUEST.
 - Emission : primitive de confirmation d'émission : CONFIRM.
 - Réception : primitive d'indication de réception : INDICATION.
 - Réception : primitive de confirmation de traitement des données reçues : RESPONSE.

Service, entité, protocole

- Une entité de la couche n qui désire utiliser un service de la couche $(n-1)$ par le biais d'une primitive doit préciser ce service.
- Elle le fait grâce à un identificateur unique appelé point d'accès au service n -SAP (*Service Access Point*).



Service, entité, protocole

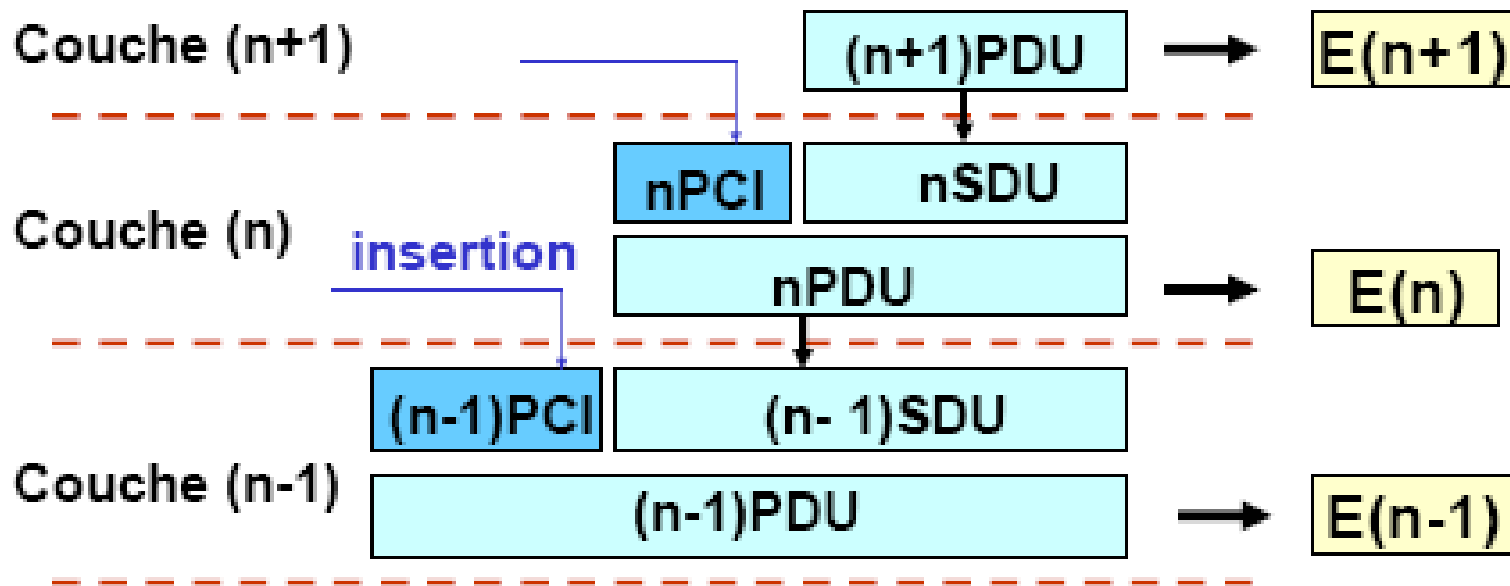
- Jargon du modèle OSI :
 - n-SDU (*Service Data Unit*) : données de service de la couche n.
 - n-PCI (*Protocol Control Information*) : information de la couche n.
 - n-PDU (*Protocol Data Unit*) : unité de données de la couche n.

$$\begin{aligned} (n+1)\text{-PDU} &= n\text{-SDU} \\ n\text{-PDU} &= n\text{-PCI} + n\text{-SDU} \\ n\text{-PDU} &= (n-1)\text{-SDU} \end{aligned}$$

Service, entité, protocole

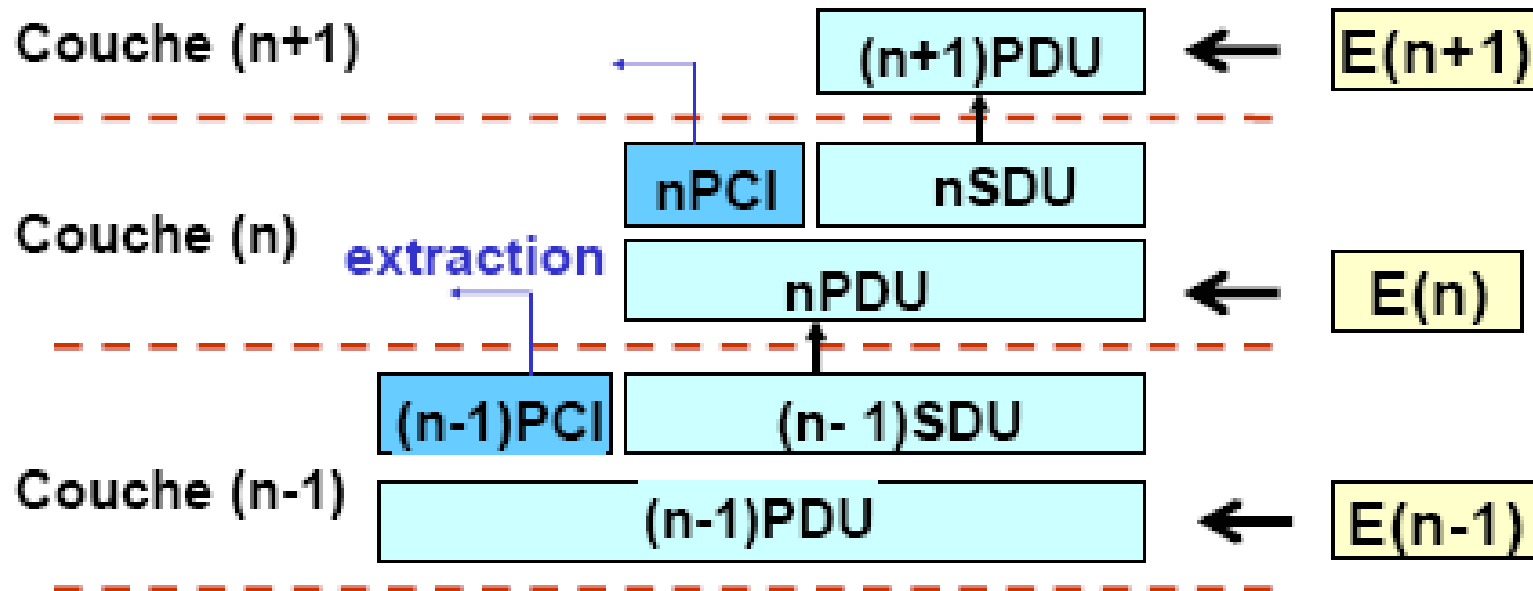
- Une des conséquence de cette structuration est :
 - L'encapsulation des données à l'émission.
 - La désencapsulation des données à la réception.
- Les données de contrôle d'un protocole ne doivent pas être trop volumineuses par rapport au données utiles : c'est le rendement du protocole !

Encapsulation des données



L'encapsulation des données

Désencapsulation des données



La décapsulation des données

Exemple de transmission de données

Transmission

message m couche 7

Réception

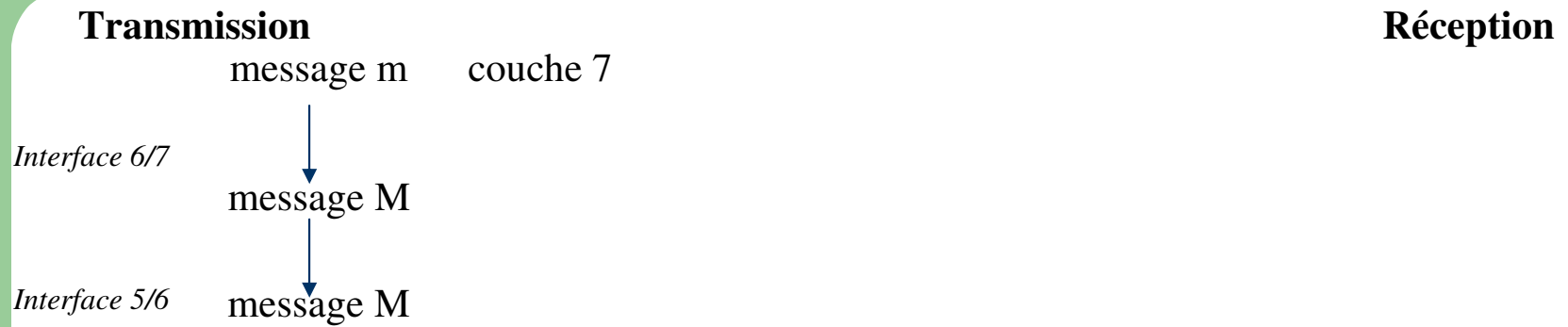
Interface 6/7

H : Header (entête)
T : Trailer (fin)

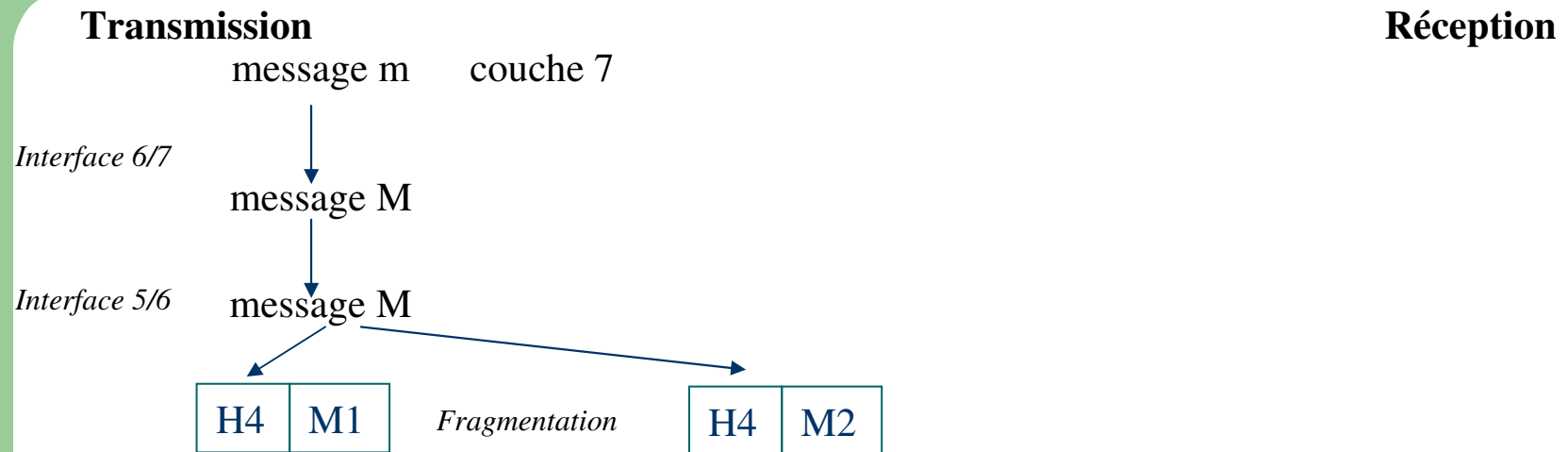
Exemple de transmission de données



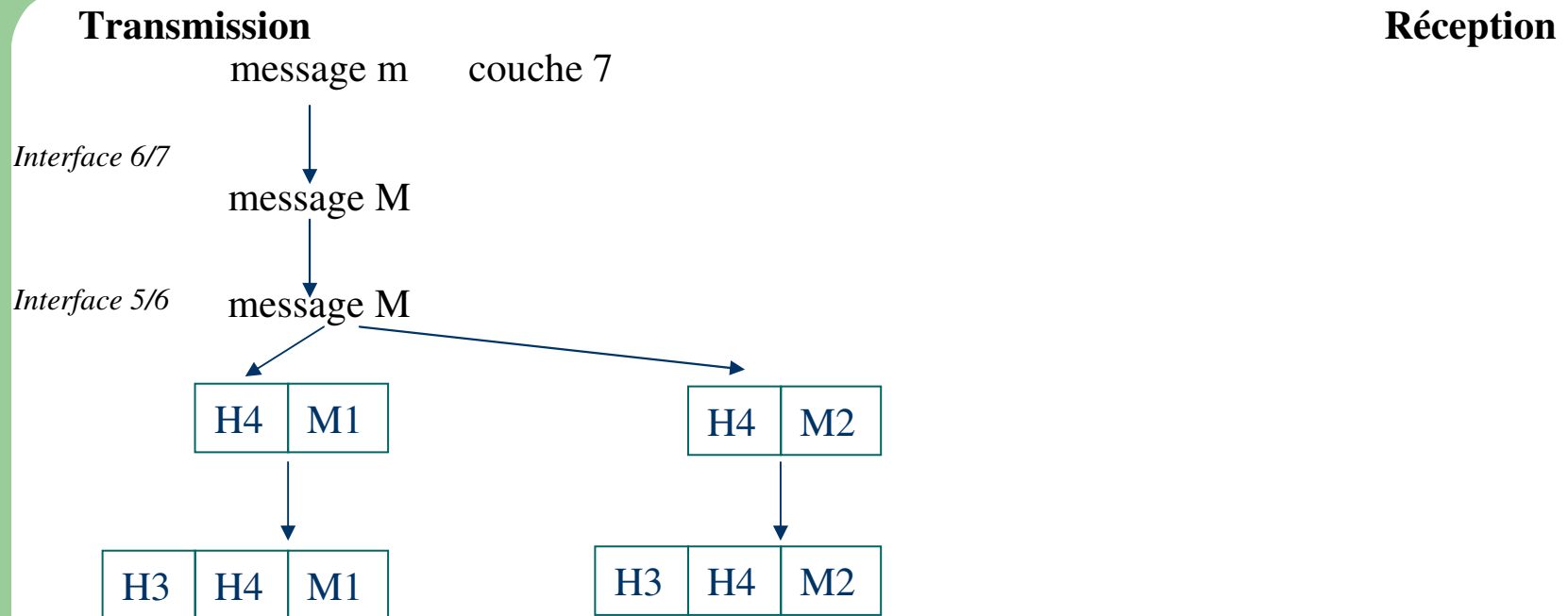
Exemple de transmission de données



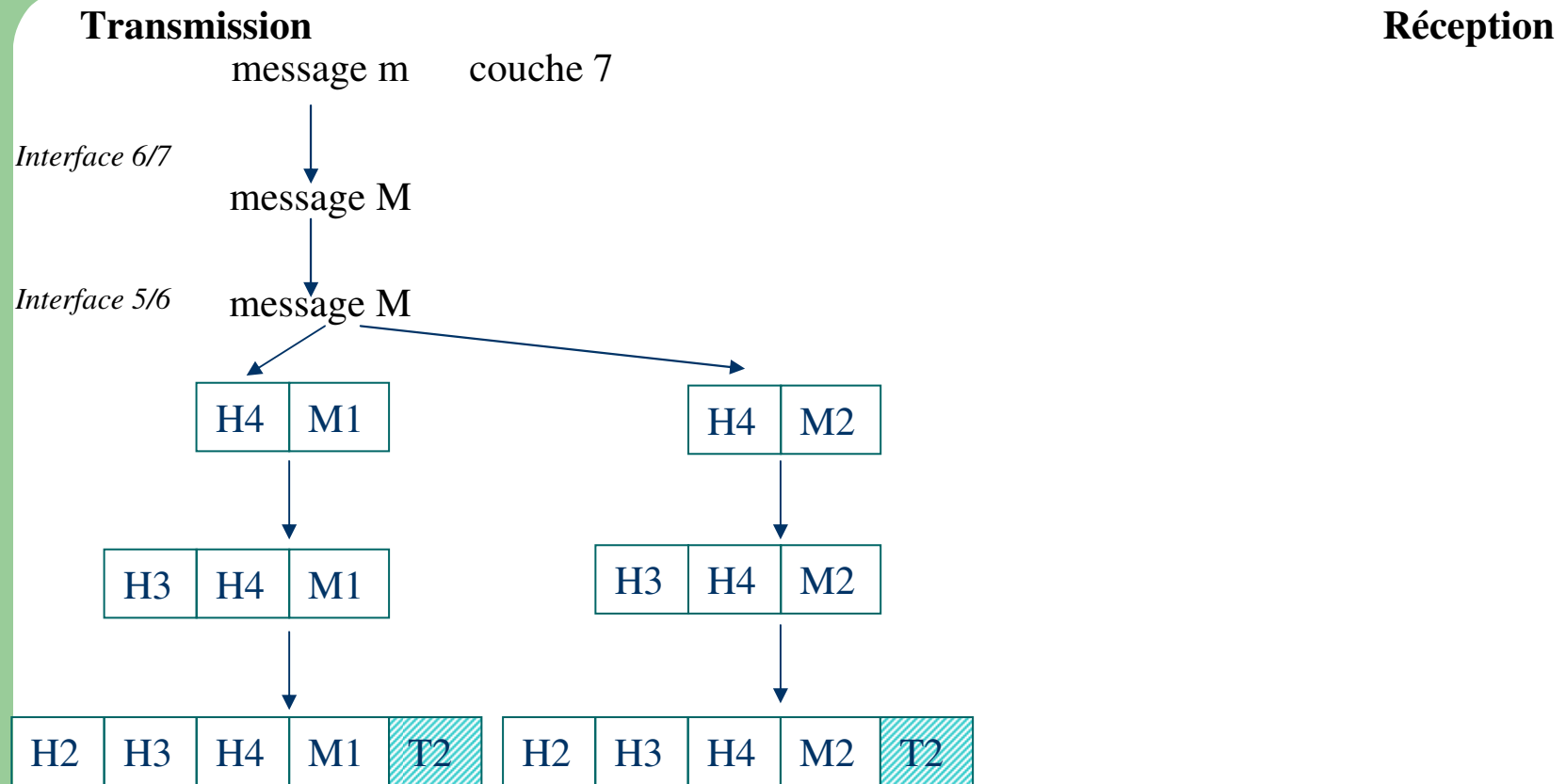
Exemple de transmission de données



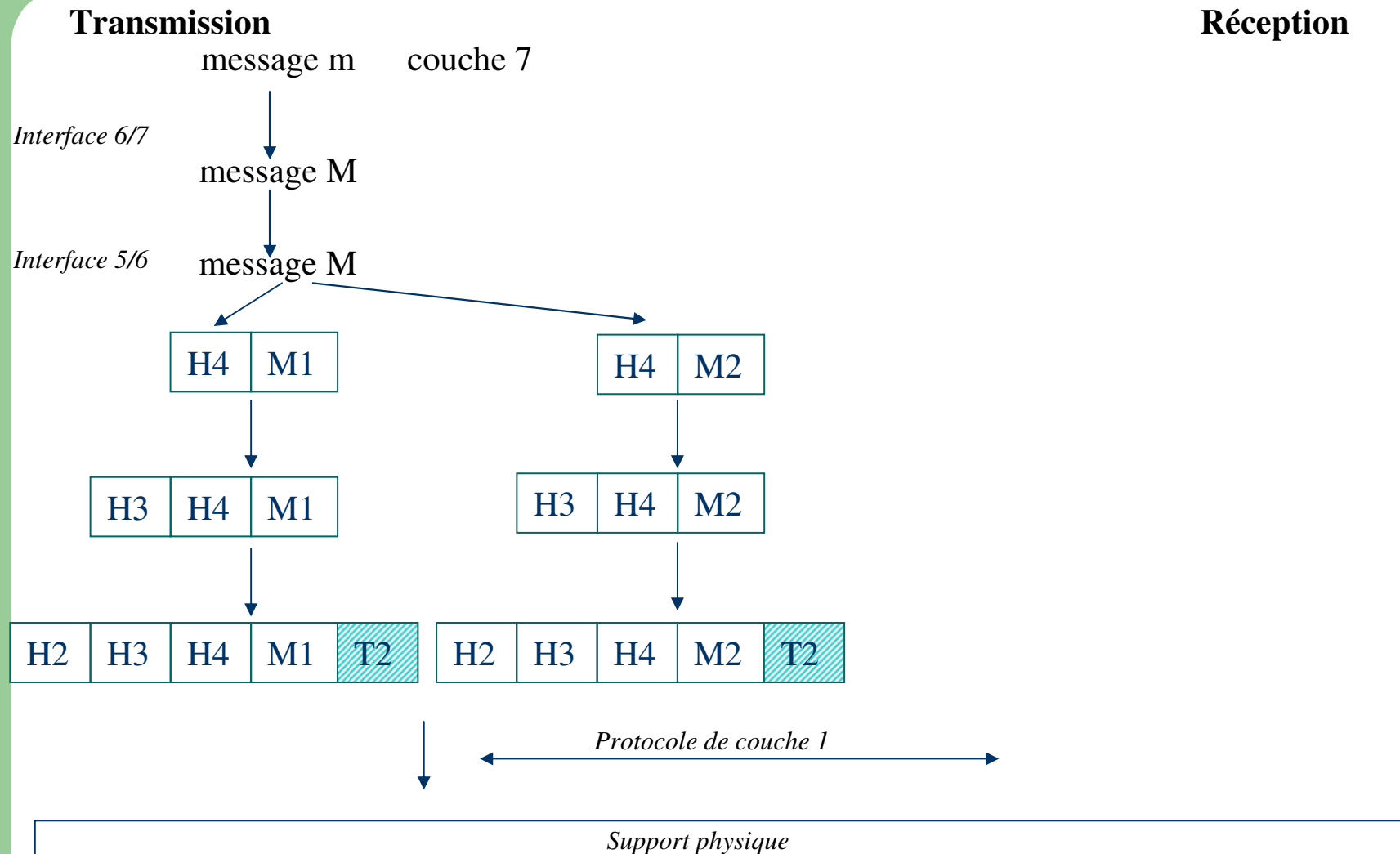
Exemple de transmission de données



Exemple de transmission de données



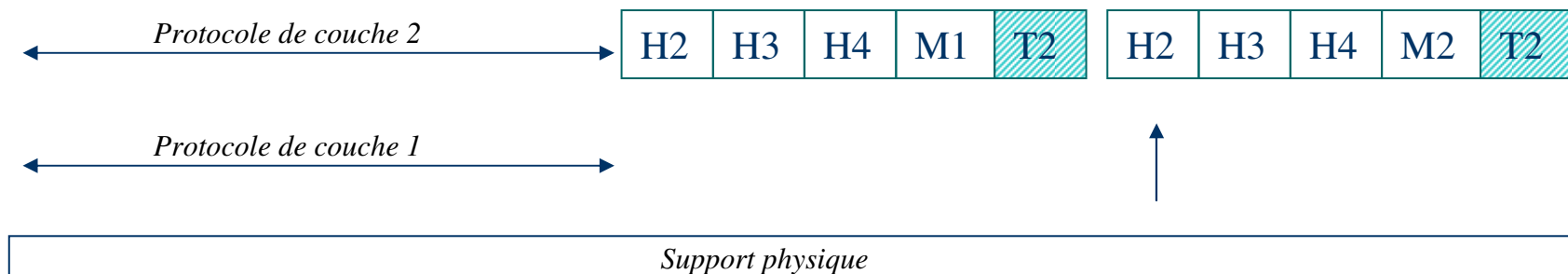
Exemple de transmission de données



Exemple de transmission de données

Transmission

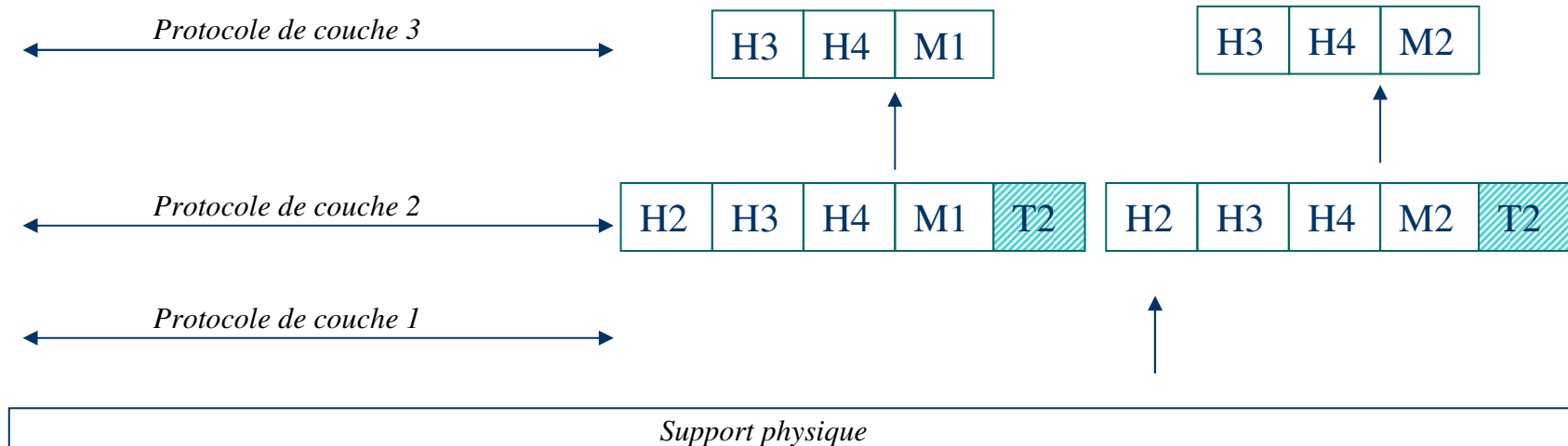
Réception



Exemple de transmission de données

Transmission

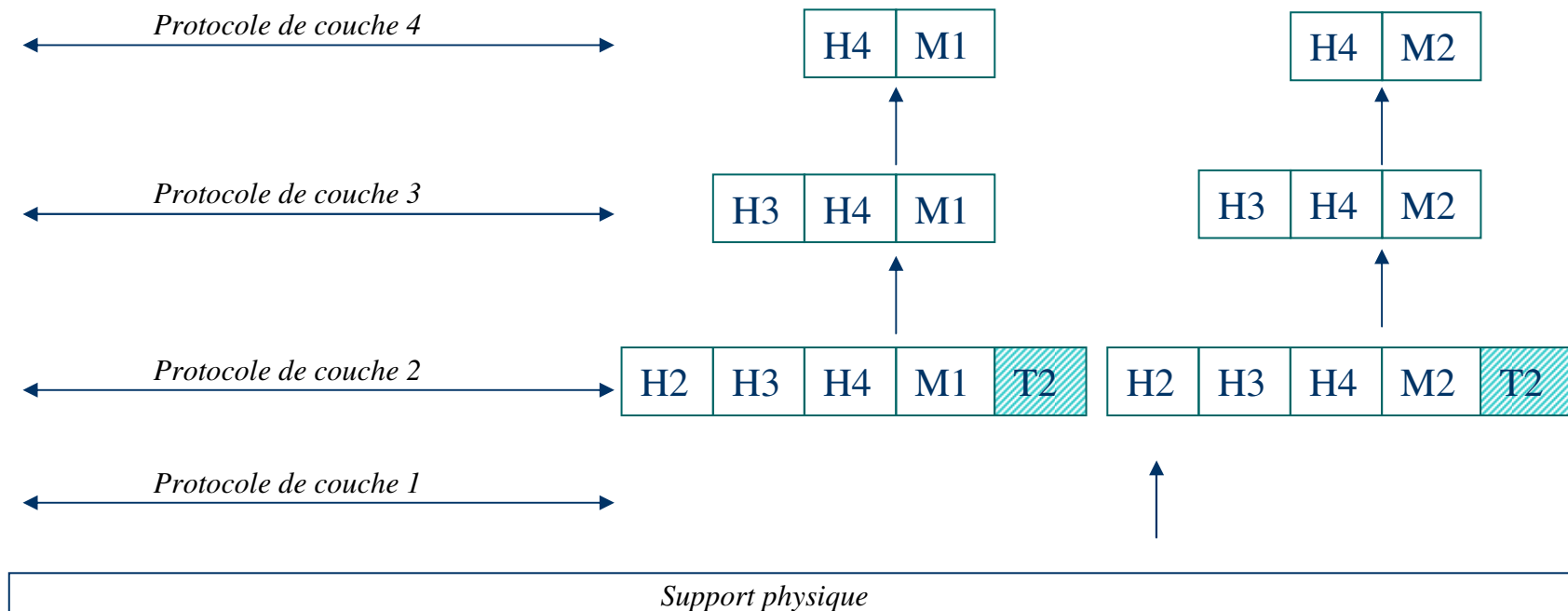
Réception



Exemple de transmission de données

Transmission

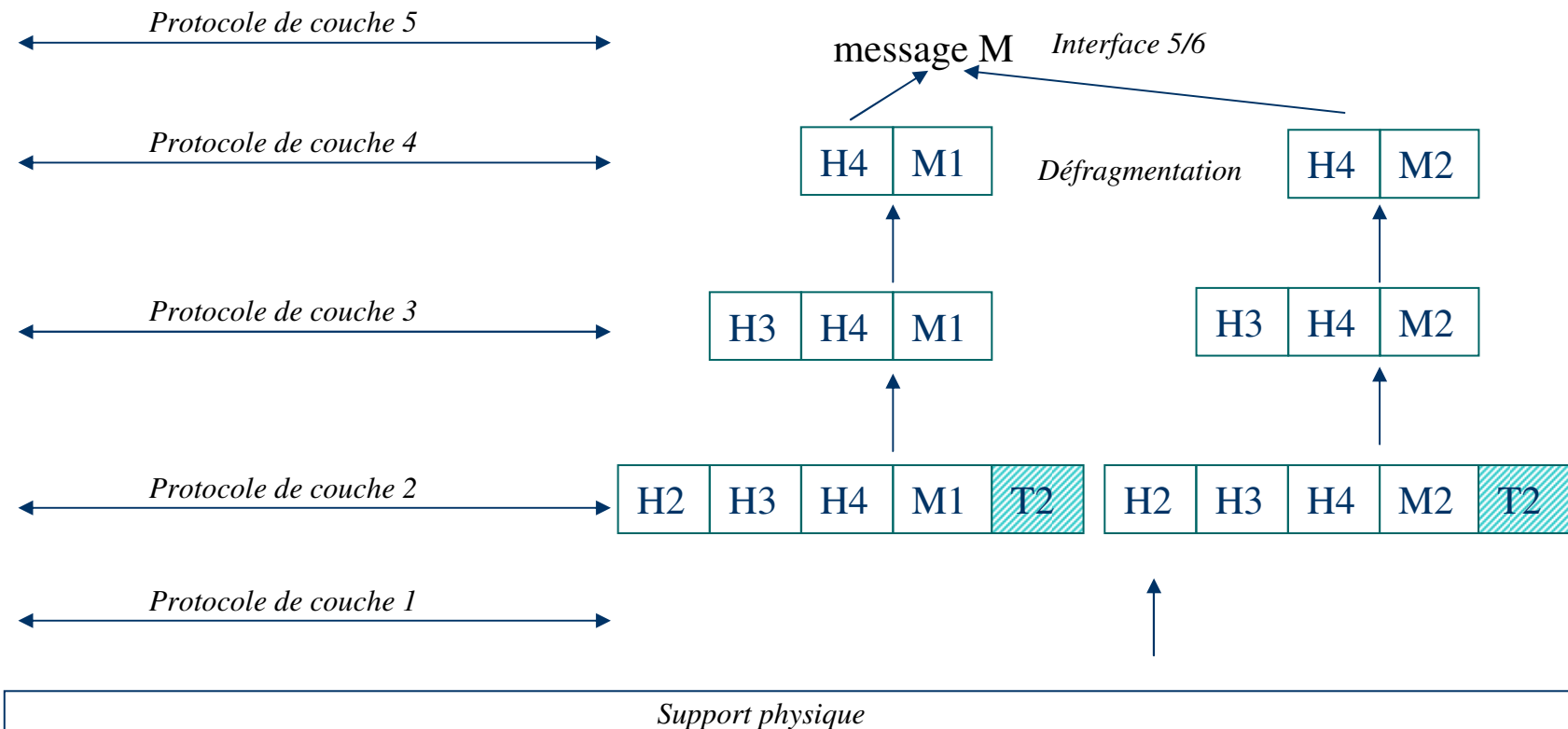
Réception



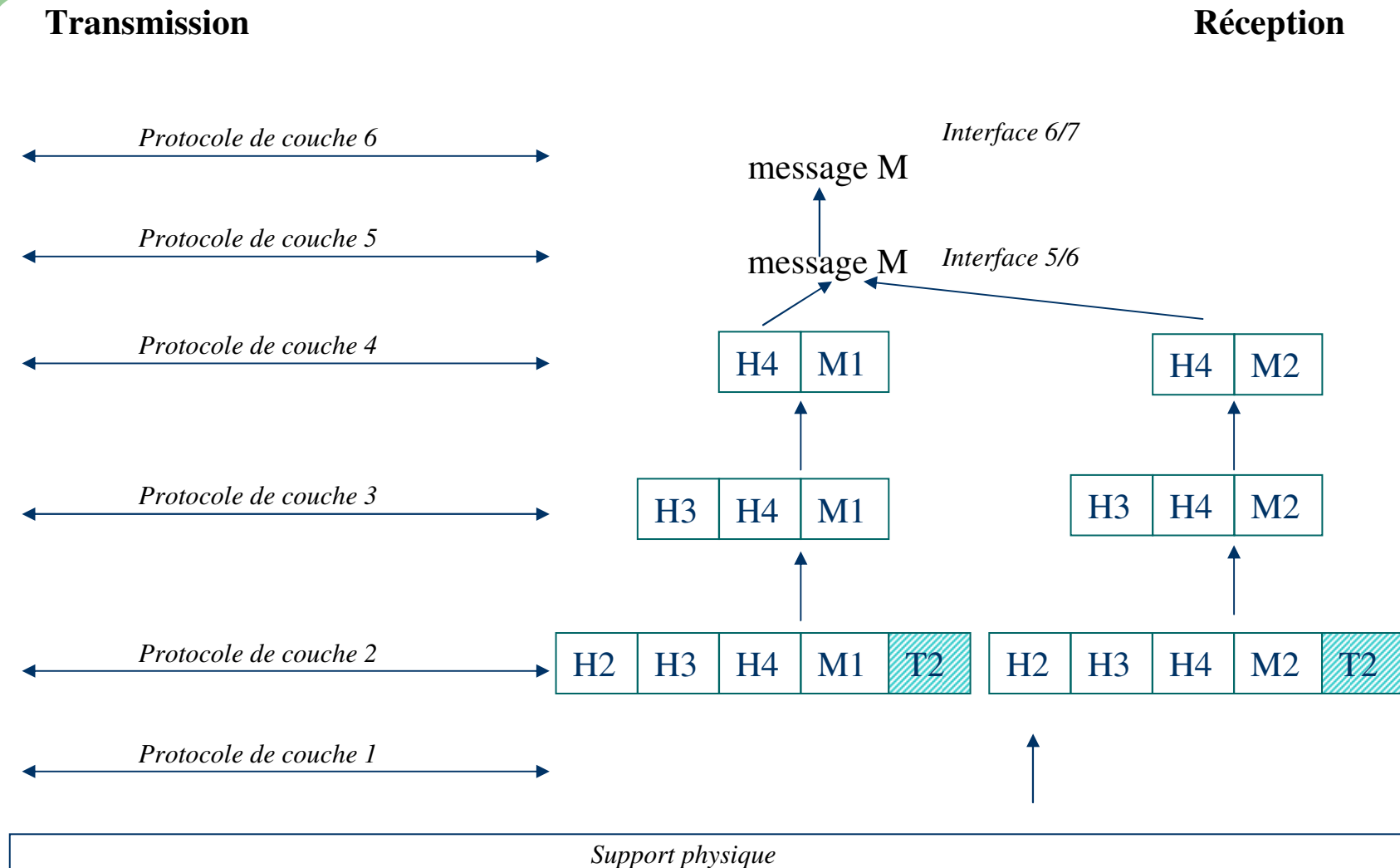
Exemple de transmission de données

Transmission

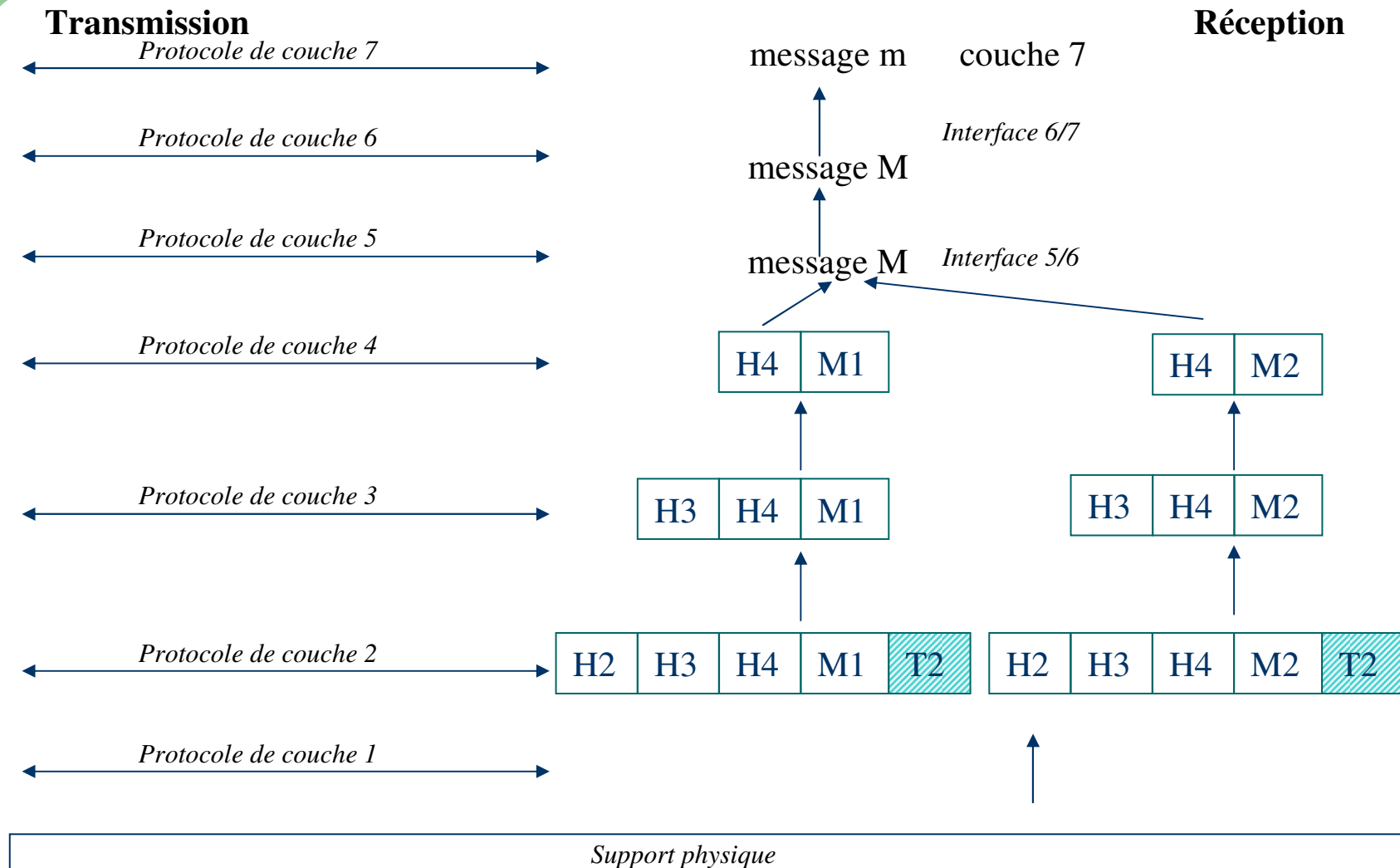
Réception



Exemple de transmission de données



Exemple de transmission de données



PARTIE 4

INTRODUCTION A INTERNET

Historique

- En 1969, le DoD (département américain de la défense) voulait relier ses centres critiques par un réseau non centralisé et adaptatif : ARPANET. Première connexion le 21 novembre 1969.
- 5 décembre 1969 : ARPANET = 4 machines.
- 1972 : présentation d'ARPANET au public.
- 1973 : mise au point des protocoles TCP/IP.
- 1980 : adoption de TCP/IP.

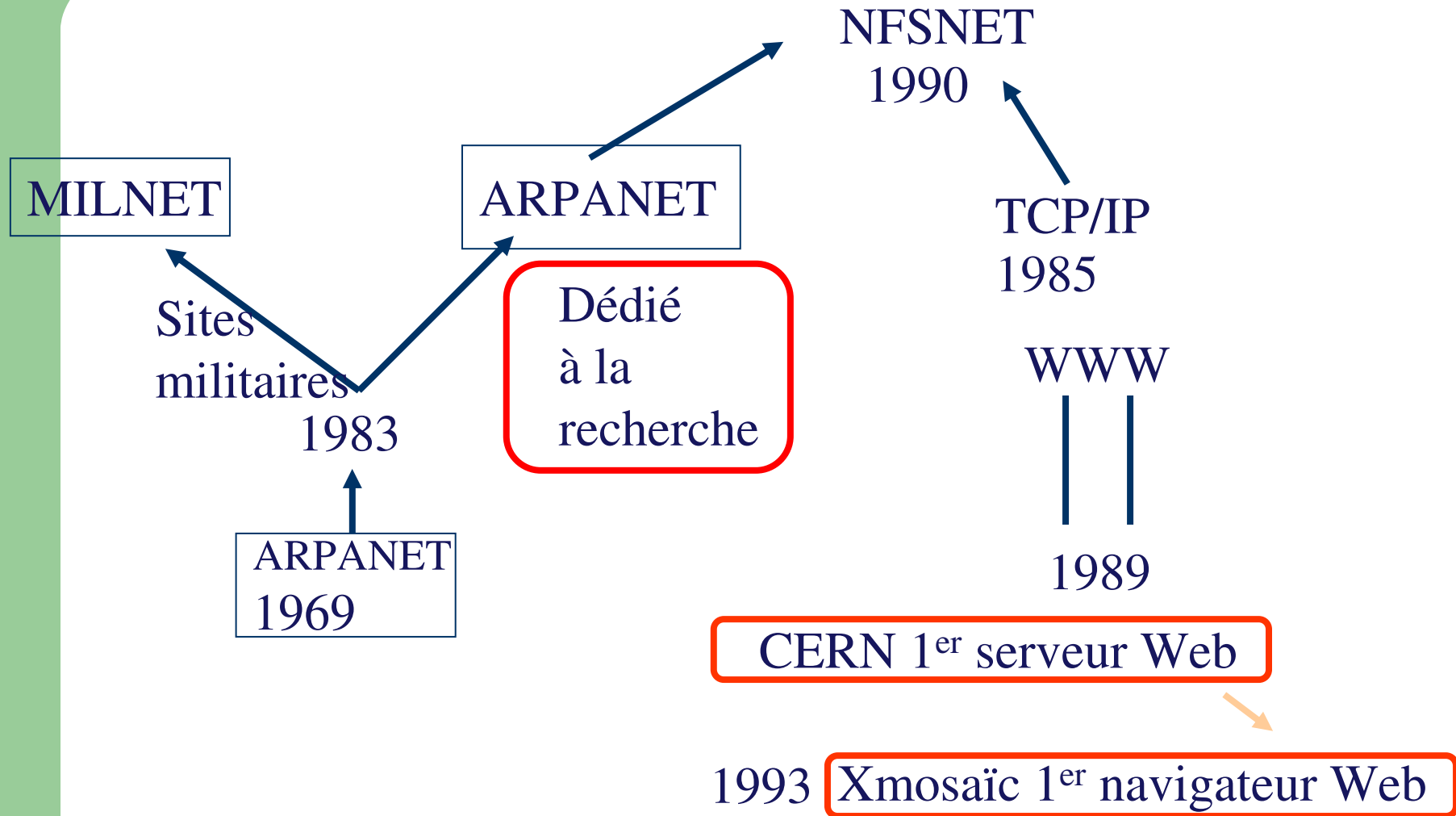
Historique

- 1983 : séparation d'ARPANET en deux : (MILNET (réseau militaire) et ARPANET (réseau de recherche)).
- 1985-86 : création du réseau NSFNET par les grandes administrations américaines...
- 1990 : fusion de NSFNET + ARPANET = Internet.
- 1992 : le CERN (*Centre Européen de Recherche Nucléaire*) propose le projet *World Wide Web* qui fournit un aspect conviviale à Internet. Il n'est pas nécessaire d'avoir des compétences en informatique pour utiliser Internet.

Historique

- 1992 : naissance du réseau de la recherche française RENATER.
- 1995 : le grand public français découvre Internet.
- En 2006, Internet a 37 ans. Il compte alors des millions de machines (dont plus de 6,5 millions de serveurs qui hébergent plus de 1 milliard de pages Web) et 939 millions d'internautes (14,6 % de taux de pénétration dans le monde).
- Aujourd'hui : Internet a envahi notre quotidien !

Historique



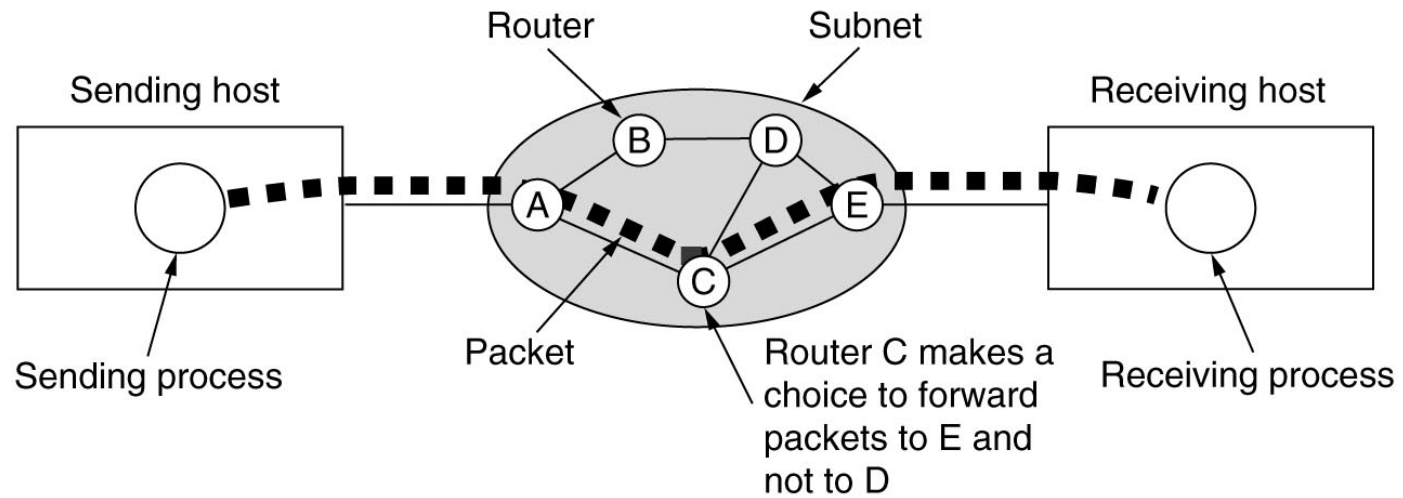
Les époques

- ARPANET : 1958-1986.
- Internet, le réseau de la recherche : 1981-1992.
- Internet grand public : 1992-2002.
- Internet aujourd'hui. Internet des objets IoT !

ARPANET

- ARPANET a comme objectifs de concevoir un réseau extensible :
 - Avec les équipements disponibles.
 - Ne requérant pas de centre de contrôle central.
 - Décentralisé et redondant.
 - Où l'information circule sous forme de paquets (*packet switching*).
 - Où ces paquets sont acheminés par stockage et retransmission (*store & forward*).

ARPANET



Les premiers protocoles

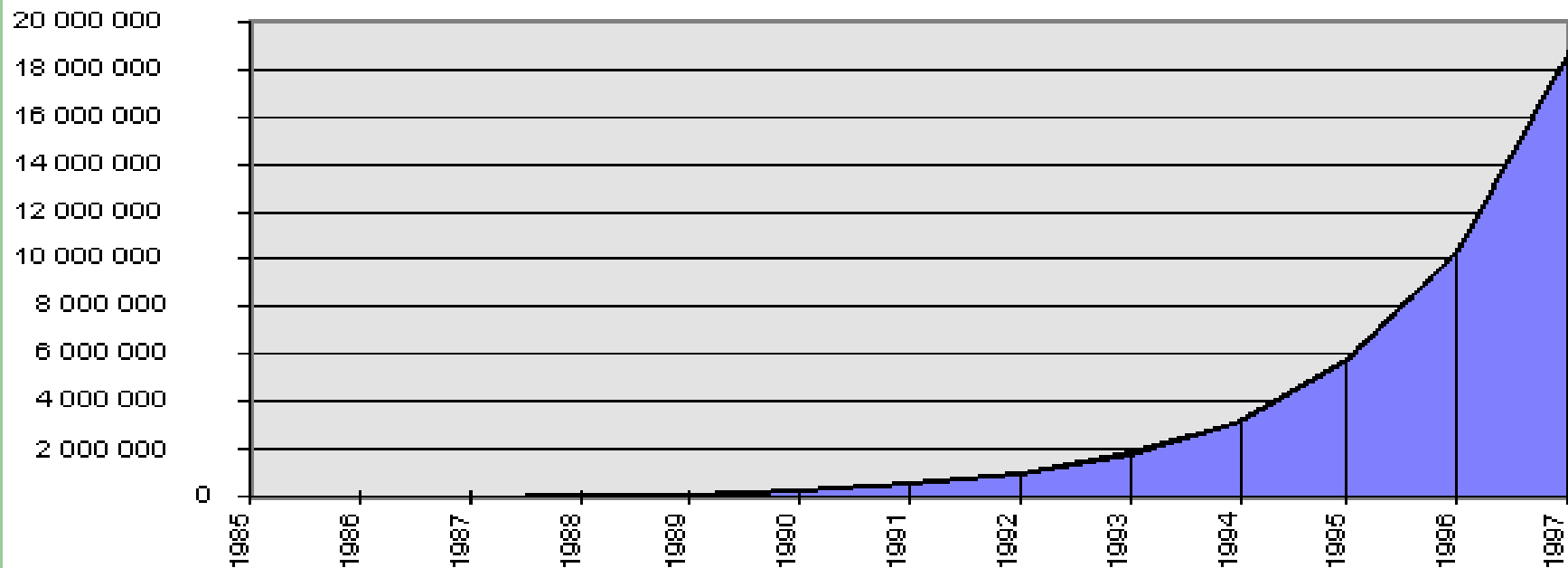
- Début des RFC (Crocker, avril 1969) :
 - *Request For Comments* : présentation ouverte et accessible d'un problème technique, d'une solution ou d'une norme technique.
 - Le nom est resté et les RFC rassemblent toujours les décisions techniques prises au cours du développement d'Internet (<http://www.faqs.org/rfcs>).
- Les premiers protocoles (1969) :
 - *File Transfer Protocol* (ftp).
 - *Telnet*.
- Le courrier électronique (Tomlinson, 1972).

D'ARPANET à Internet

- ARPANET est réservé aux universités et aux centres de recherche impliqués dans des projets militaires.
- Au milieu des années 1980, la *National Science Foundation* prend la suite d'ARPANET pour la gestion du réseau non militaire. Pour cela, elle crée le réseau NFSNET (1986).
- Vers la fin des années 1980, ARPANET est fermé. Dorénavant, c'est Internet (et ses protocoles) !

Internet devient grand public (1992-2002)

Nombre d'ordinateurs reliés à l'Internet



RE223 : Introduction aux réseaux et à Internet...



Début de la diffusion d'information

- Apparition du protocole Gopher (1991).
- Apparition du Web (1992) :
 - Xmosaic (1993).
 - Netscape (1994).
 - IE (1995).
- Les premiers sites Web...



Début du commerce électronique

- Amazon, eBay (1995).
- Google (1997).
- PayPal (1998).

- Autres développements d'importance :
 - *Peer to Peer* (P2P).
 - Napster (1999).

- Netscape propose le protocole SSL (*Secure Socket Layer*) (1994) pour des échanges sécurisés par chiffrement pour le paiement électronique.

Internet aujourd'hui

- Consultation de pages Web et recherche d'information.
- Transfert de fichiers (ftp).
- Messagerie électronique (*email*).
- Forums de discussion en différé (*news*) et en direct (*chat*).
- visioconférence, visioenseignement, téléphonie et voix sur IP (VoIP)...
- Accès à distance (à des données sur des serveurs distants), Gopher (accès et navigation dans des BD mondiales à l'aide de catalogues)...
- Accès à des informations multimédia et jeux en réseau (télévision, radio, films à la demande, échange de fichiers numériques...).
- Contrôle à distance, télémédecine...
- Internet des objets !

PARTIE 5

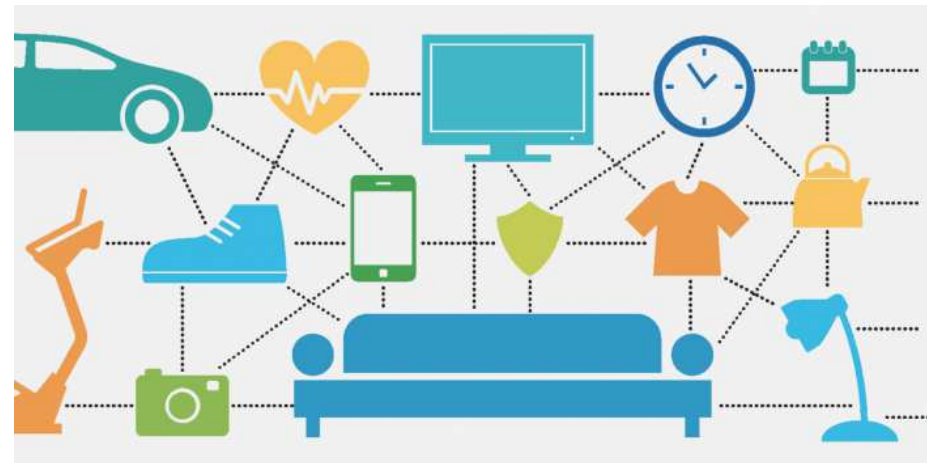
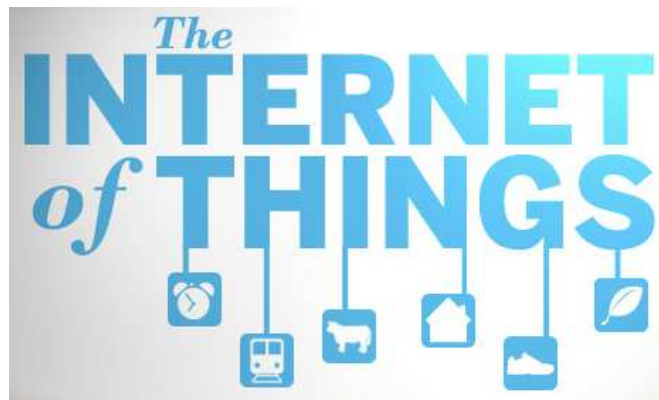
D'INTERNET EMBARQUE A L'INTERNET DES OBJETS

Objets connectés. Internet des objets

- Connecter un système électronique embarqué à Internet que l'on appellera objet suppose qu'il embarque Internet.
- Chaque objet connecté embarque donc une pile TCP/UDP/IP et possède aussi une adresse IP (publique ou privée).
- Internet embarqué permet ainsi de connecter à Internet tout type d'objets :
 - Chaudière, réfrigérateur, caméra.
 - ...
- Ces objets connectés à Internet forment l'Internet des objets !

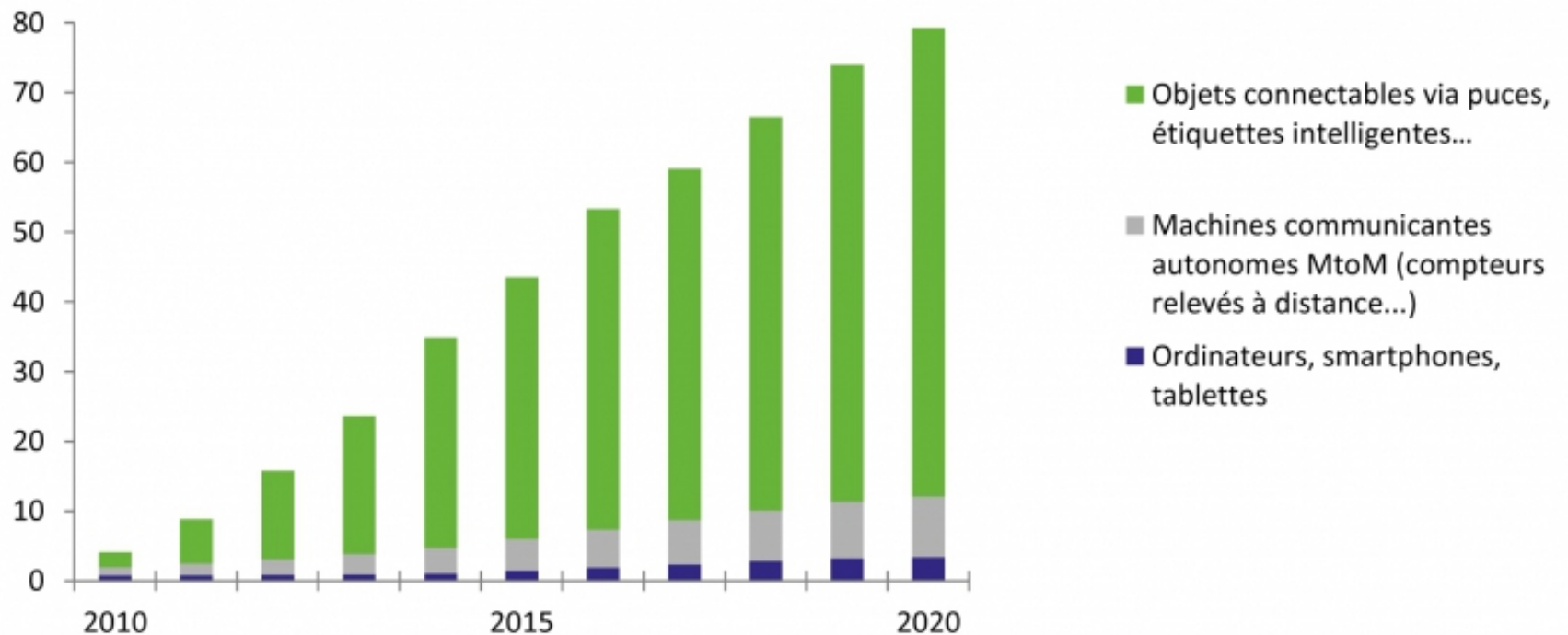
Objets connectés. Internet des objets

- L'Internet des objets (connectés) peut se définir comme :
« *des objets ayant des identités et des personnalités virtuelles, opérant dans des espaces intelligents et utilisant des interfaces intelligentes pour se connecter et communiquer au sein de contextes d'usages variés.* »



Objets connectés. Internet des objets

- L'Internet des objets connaît une croissance exponentielle et c'est un domaine d'activité pourvoyeur d'emplois :



Objets connectés. Internet des objets

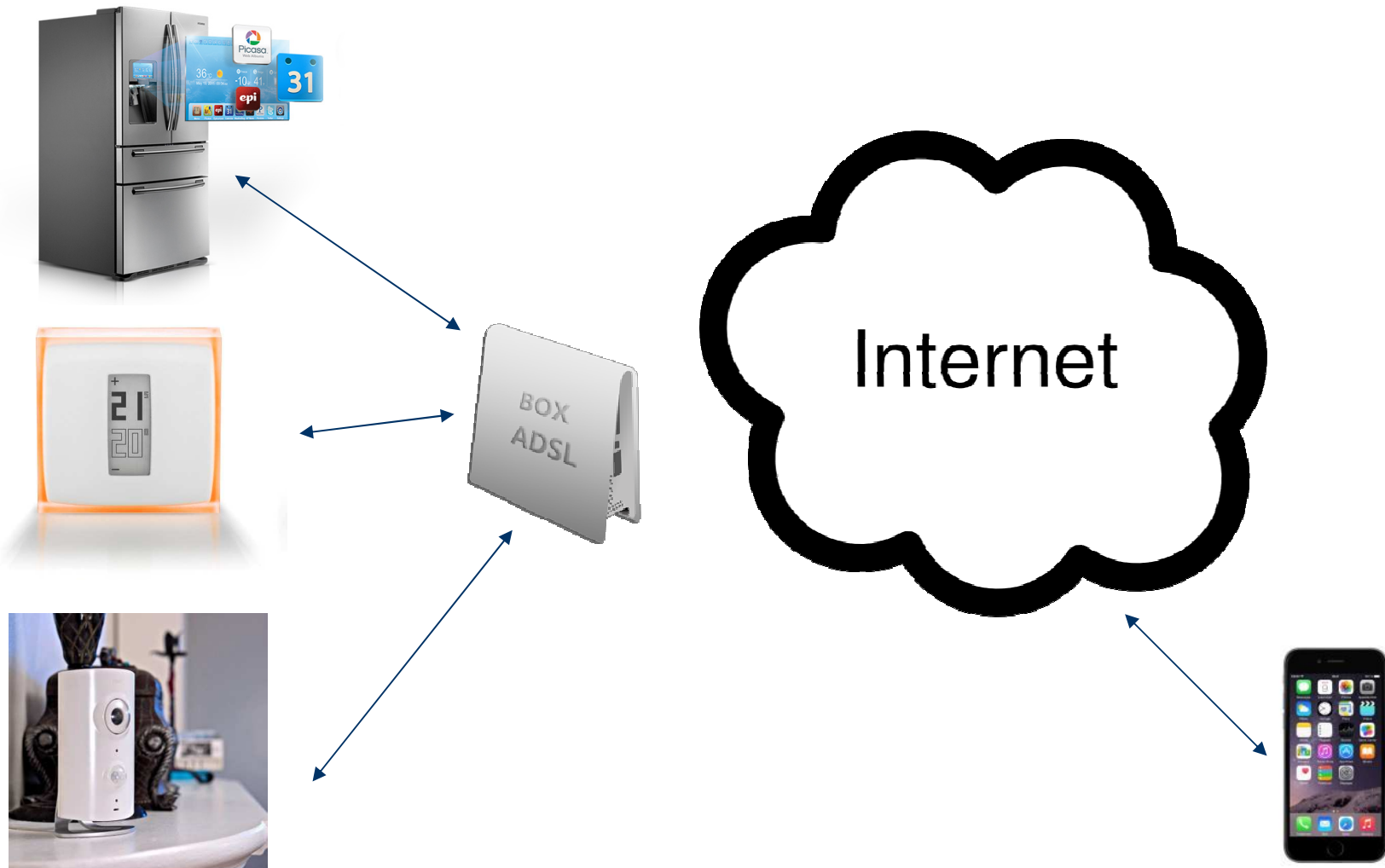
- L'Internet des objets, c'est :
 - 80 milliards d'objets connectés en 2020 (200 milliards en 2030)* :
 - 85 % d'objets grand public.
 - 11 % de terminaux mobiles (*smartphones*).
 - 4 % de M2M (*Machine to Machine*).
 - 11,5 milliards d'euros de CA sur 10 ans.
 - Domaines où ils se développeront :
 - Santé.
 - Sécurité.
 - Maison connectée.
 - Mobilité.

* *La puce à l'oreille No 44. CNFM. 06/20*

Objets connectés. Internet des objets

- Il y a deux champs d'application importants dans l'Internet des objets :
 - Domotique : contrôle (à distance) des équipements électroniques dans une maison.
 - M2M : contrôle et dialogue entre machines sans intervention humaine.
- Il y a deux éléments importants dans l'Internet des objets :
 - La *box* ADSL : connexion permanente à Internet.
 - Le *smartphone* : application *user centric* de contrôle à distance.

Objets connectés. Internet des objets



RE223 : Introduction aux réseaux et à Internet...

Objets connectés. Internet des objets

- En conclusion, vouloir développer un objet connecté demande d'y intégrer une connectivité Internet et de développer des applications Internet client/serveur.
- Ce module s'inscrit donc pleinement dans ce contexte pour :
 - Comprendre les protocoles essentiels d'Internet.
 - Développer des applications Internet client/serveur avec l'API *sockets*.
- l'API *sockets* sera vu avec le langage C sous *NIX mais existe sous Java, Python... C'est donc une technologie universelle !

API=Application Programming Interface

PARTIE 6

LES NORMES ETHERNET IEEE 802.3

LES BASES : NORME 10BASE5

Historique

- Origine : système ALOHA exploité à Hawaï en 1970 basé sur la méthode d'accès CSMA (*Carrier Sense Multiple Access*).
- Ethernet V1 (1976) : conçu et mis en œuvre par Xerox :
 - Débit : 3 Mb/s (*Experimental Ethernet*).
 - Médium : coaxial de 1000 m.
 - Nombre maximum de stations : 100.
- Ethernet V2 (1980) DIX (Digital - Intel - Xerox) :
 - Débit : 10 Mb/s.
 - Base de travail à la norme IEEE 802.3.

Historique de la norme IEEE 802.3

- 1985 : 10 Base 5 (802.3)
- 1988 : 10 Base 2 (802.3b)
- 1990 : 10 Base T (802.3i)
- 1993 : 10 Base F (802.3j)
- 1995 : 100 Base X (802.3u)
100 Base VG (802.12)
- 1998 : 1000 Base LX, SX et CX (802.3z)
- 1999 : 1000 Base T (802.3ab)
- ...

Introduction

- Cette partie est basée sur la norme de 1985 qui conditionne les évolutions d'Ethernet.
- Norme de 1985 (10Base5) : câble coaxial épais.
- Ethernet = réseau local.
- Ethernet est basé sur la méthode d'accès CSMA/CD :
 - CSMA : *Carrier Sense Multiple Access* (accès multiple avec écoute de la porteuse).
 - CD : *Collision Detection* (détection de collision).

Introduction

- Ce qui est pris en charge :
 - Simple, faible coût.
 - Peu de fonctions optionnelles.
 - Pas de priorité.
 - On ne peut pas faire taire son voisin.
 - Débit : 10 Mb/s.

Introduction

- Ce qui n'est pas pris en charge :
 - Full duplex.
 - Contrôle d'erreur.
 - Sécurité et confidentialité.
 - Vitesse variable (auto-négociation).
 - Priorité.
 - Protection contre un utilisateur malveillant.
 - Déterminisme (capacité de borner en temps les transmissions de données).

Principes

- Support de transmission :
 - Segment = bus = câble coaxial.
 - Bus passif.
 - Pas de boucle, pas de sens de circulation.
 - Diffusion de l'information, écoute sélective.
 - Transmission en bande de base.
- Équipement raccordé sur ce câble par un *transceiver* :
$$\text{transmitter} + \text{receiver} = \text{transceiver}$$
- Un équipement Ethernet a une adresse unique au monde (adresse Ethernet ou adresse MAC (*Media Access Control*)).

Principes

- Sur le câble circulent des trames :
 - Suites d'éléments binaires.
 - À un instant donné, une seule trame circule sur le câble.
 - Pas de multiplexage en fréquence.
 - Pas de *full duplex*.
- La trame émise par un équipement est reçue par tous les *transceivers* du segment Ethernet.
- La trame contient l'adresse de l'émetteur et du destinataire.

Principes

- Un coupleur est à l'écoute de la totalité des trames qui circulent sur le câble :
 - Si une trame lui est destinée :
 - Adresse destinataire = sa propre adresse physique.
 - Il la prend, la traite et la délivre à la couche supérieure.
 - Sinon, le coupleur ne fait rien.

Principes

- Une station qui veut émettre :
 - Regarde si le câble est libre.
 - Si oui, elle envoie sa trame.
 - Si non, elle attend que le câble soit libre.
- Si 2 stations émettent ensemble, il y a collision :
 - Les 2 trames sont inexploitable.
 - Les 2 stations détectent la collision, elles ré-émettront leur trame ultérieurement.

Principes

- Ethernet est donc un réseau :
 - Probabiliste.
 - Sans chef.
 - Egalitaire.

Format d'une trame IEEE 802.3



- Débit d'émission/réception : 10 Mb/s :
 - 10 bits par μ s.
- Longueur des trames (avec préambule et SFD) :
 - 26 octets réservés au protocole.
 - Longueur minimale : **72 octets (64+8)**.
 - Longueur maximale : **1526 octets (1518+8)**.

Format d'une trame IEEE 802.3

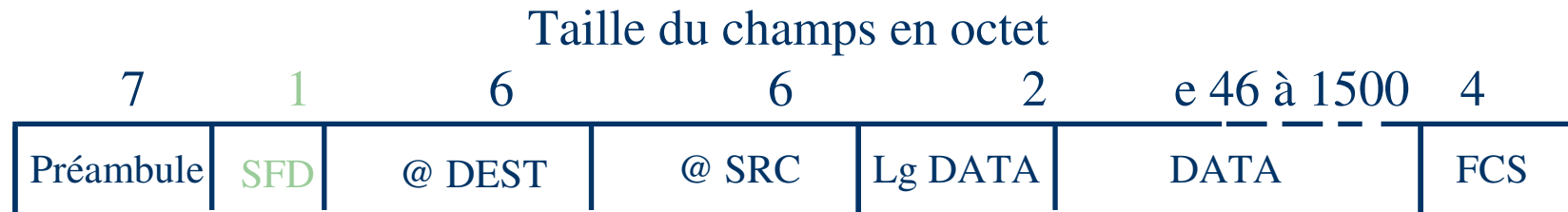
- Espace intertrame minimal de $9.6 \mu\text{s}$:
 - Espace intertrame = 96 bits time soit 12 octets.
 - Utilisation du réseau dans un délai relativement faible.
 - Une machine ne peut pas émettre toutes ses trames en même temps : seulement les unes à la suite des autres.
- Cet espace intertrame permet :
 - Aux circuits électroniques de récupérer l'état de repos du médium.
 - Aux autres machines de reprendre la main à ce moment là.

Trame IEEE 802.3 : préambule



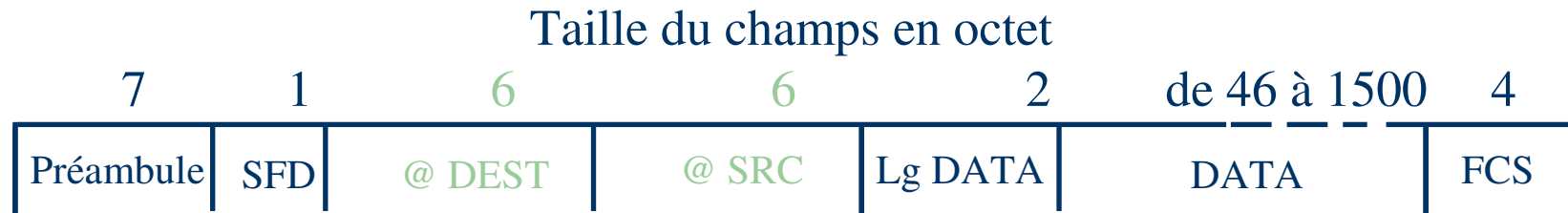
- Taille : 7 octets identiques (10101010). Simple suite continue de bit à 0 et de bit à 1.
- Assez long pour servir à la synchronisation des PLL.
- Pas de fin de trame (pas d'échappement).

Trame IEEE 802.3 : SFD



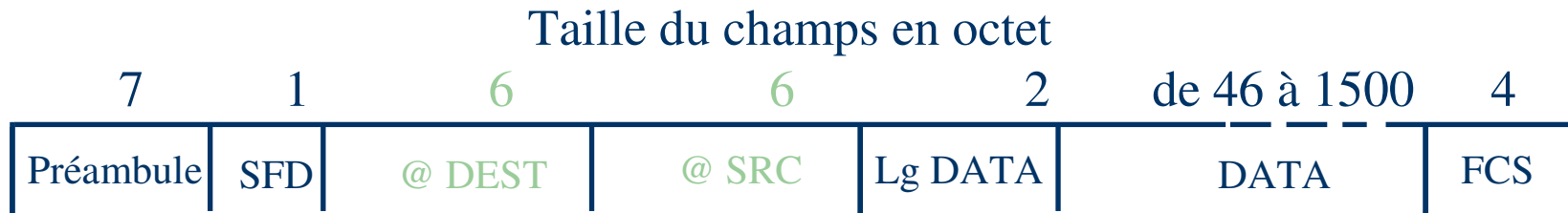
- SFD : *Start Frame Delimitator*.
- Marque le début de la trame.
- Taille : 1 octet.
- SFD = 10101011

Trame IEEE 802.3 : adresses



- Détails dans la RFC 1700.
- Adresse IEEE 802.3 ou Ethernet : 48 bits (6 octets).
 - Syntaxe : 08:00:20:05:B3:A7 ou 08:00:20:05:B3:A7
 - ❖ Cisco 00:00:0C:XX:XX:XX Sun 08:00:20:XX:XX:XX
 - ❖ HP 08:00:09:XX:XX:XX

Trame IEEE 802.3 : adresses



- L'adresse destinataire peut donc représenter :
 - L'adresse physique d'une machine locale.
 - Toutes les machines du réseau local (*broadcast*).
- L'adresse source représente seulement :
 - L'adresse physique de la station émettrice.

Trame IEEE 802.3 : longueur



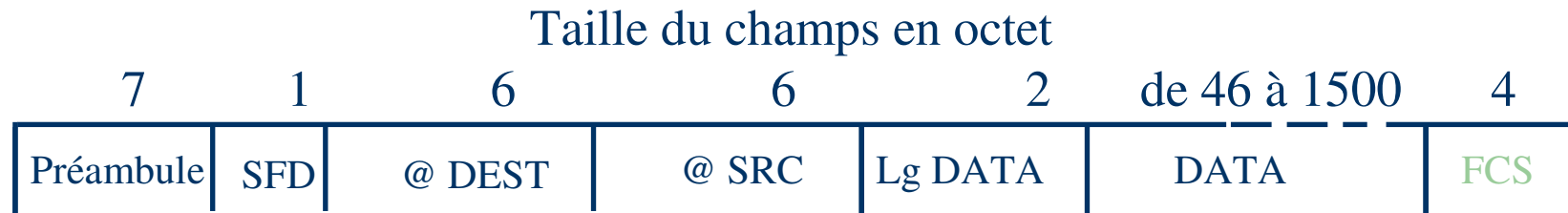
- Taille : 2 octets (valeur ≤ 1500).
- Donne le nombre d'octets utilisé par les données dans 1 trame.
- *Padding* : Ajout d'octets sans signification pour envoyer moins de 46 octets de données.
- Longueur minimale de la trame : **72 octets (64+8)**.

Trame Ethernet : type données



- Type de données transportées : 2 octets.

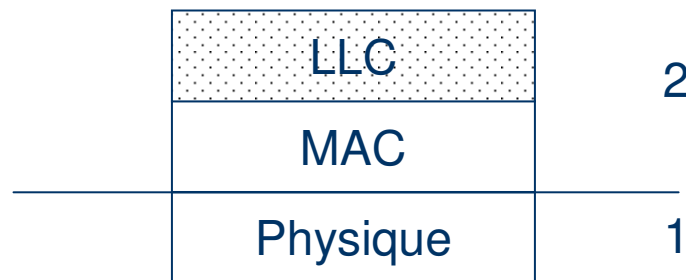
Trame IEEE 802.3 : FCS



- FCS : *Frame Check Sequence*.
- Contrôle à la réception de la trame par calcul :
 - Code de Redondance Cyclique (CRC) calculé sur les champs destination, source, longueur et données.
- Taille de 4 octets.

Sous couche MAC : transmission

- La sous couche MAC :
 - Ajoute préambule, SFD, padding si nécessaire.
 - Assemble les champs : @source, @destinataire, taille, données et padding.
 - Calcule le FCS et l'ajoute à la trame.
 - Transmet la trame à la couche physique :
 - Si "écoute porteuse" faux depuis $9.6 \mu\text{s}$ au moins, la transmission s'effectue.



MAC : *Media Access Control*
LLC : *Logical Link Control*

Sous couche MAC : réception

- La sous couche MAC :
 - Reçoit les bits qui circulent sur le câble.
 - Les limites des trames sont indiquées par le signal "écoute porteuse" .
 - Ote le préambule et le SFD.
 - Analyse l'adresse du destinataire dans la trame.
- Si l'adresse de destination de la trame est différente de l'adresse de la station alors poubelle.

Sous couche MAC : réception

- Si l'adresse destination est la station :
 - Elle découpe la suite de bits reçus en octet, puis en champs.
 - Transmet à la sous couche LLC les champs :
 - ❖ @destination, @source, taille et données .
 - Calcule le FCS et indique une erreur à la couche LLC si :
 - ❖ FCS incorrect.
 - ❖ **Trame trop grande : > 1526 octets.**
 - ❖ **Trame trop petite : < 72 octets.**
 - ❖ Longueur de la trame n'est pas un nombre entier d'octets (erreur d'alignement).

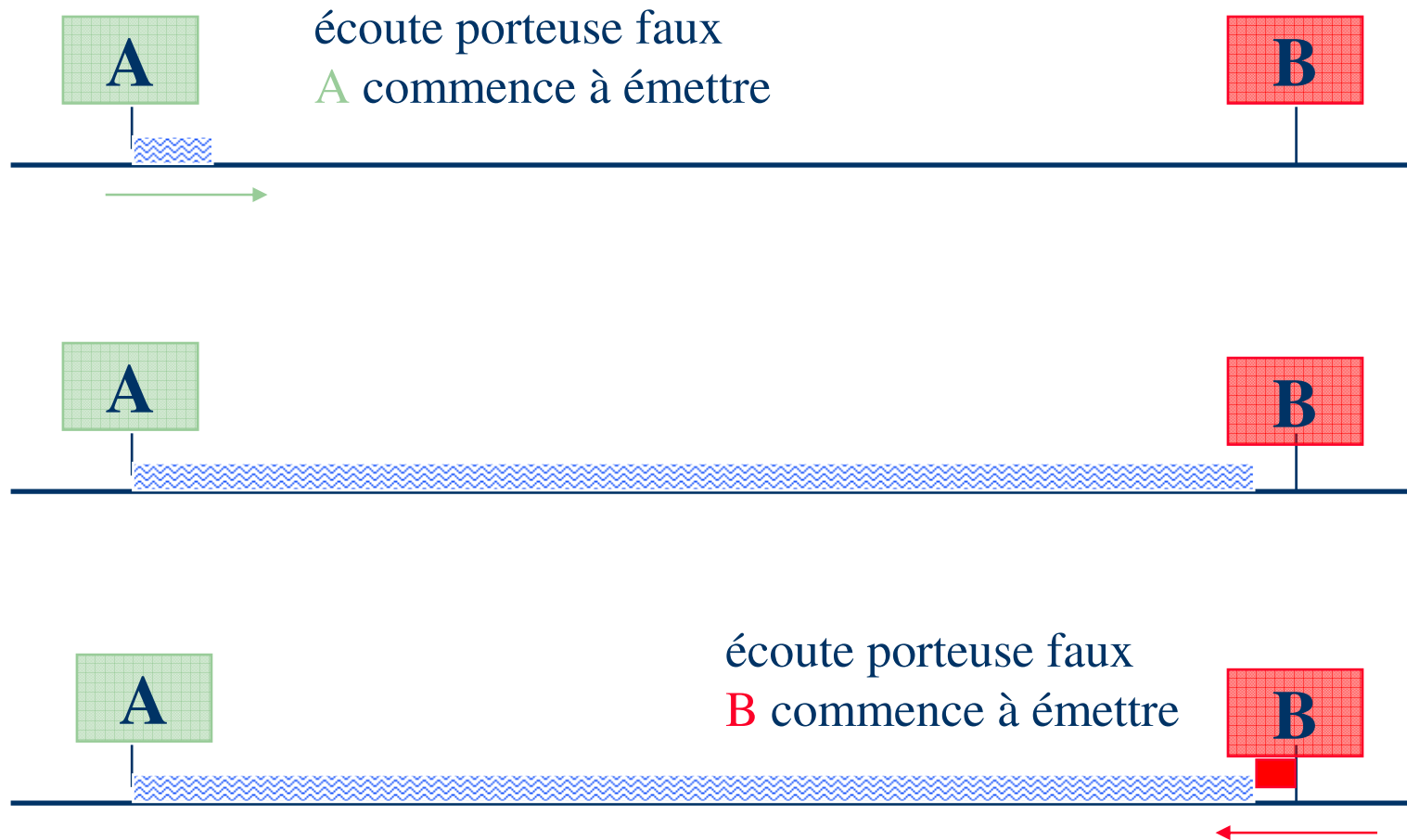
Couche physique

- Fonctions de la couche physique :
 - Permet de recevoir et d'émettre des suites d'éléments binaires.
 - Détecte la transmission par une autre station :
 - Pendant que la station n'émet pas : signal "écoute porteuse".
 - Pendant que la station émet : signal "détection de collision".

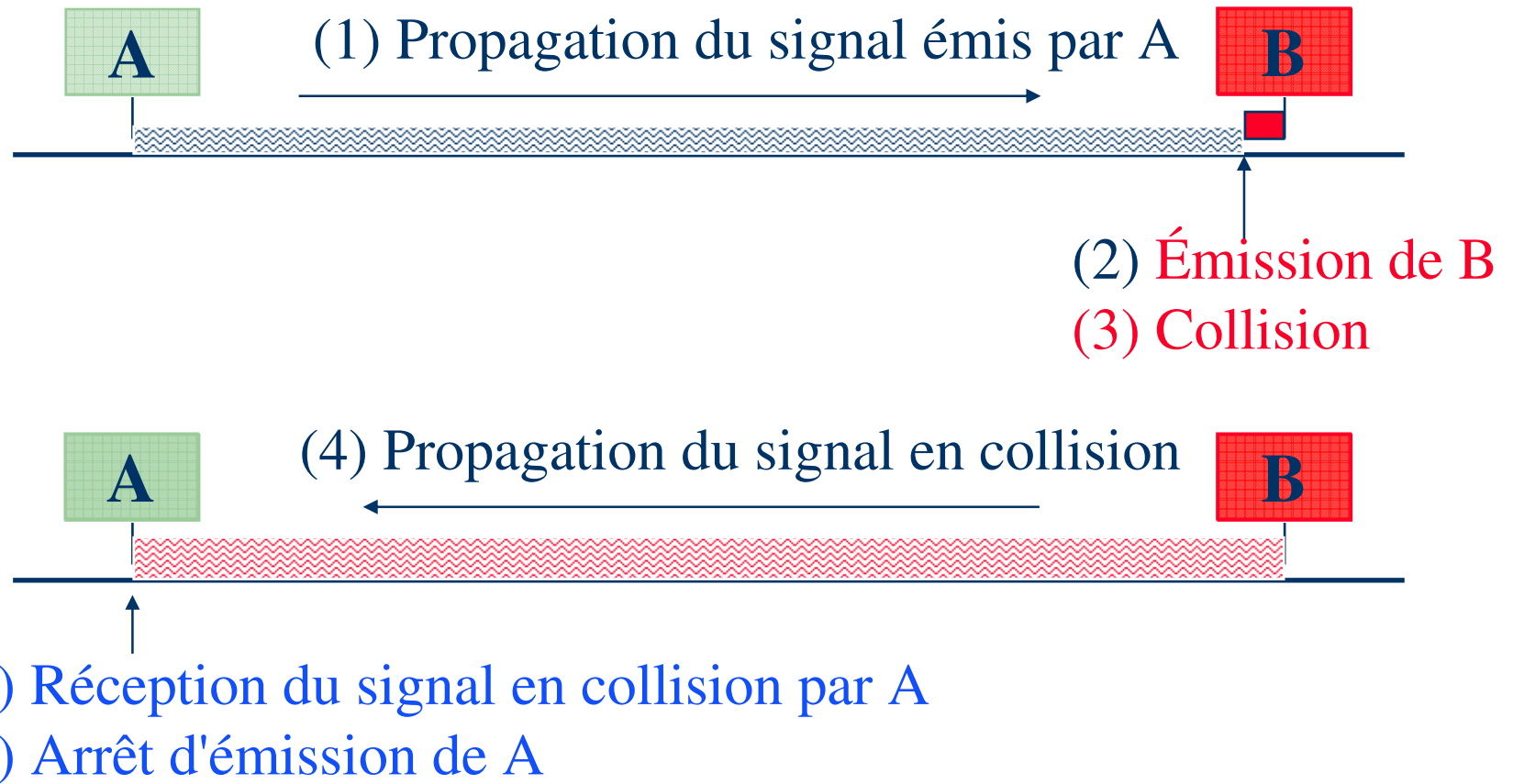
Collisions : problématique

- Une station regarde si le câble est libre avant d'émettre (signal "écoute porteuse").
- Mais le délai de propagation d'une trame sur le réseau n'est pas nul : une station peut émettre alors qu'une autre a déjà commencé à émettre.
- Quand ces 2 trames émises presque simultanément se « croisent », il y a collision.

Collision : exemple



Collision : exemple



Temps maximum écoulé = Aller + Retour $\approx 50 \mu\text{s} \approx$ temps d'émission de 63 octets

Collision : solutions

- Temps aller/retour = RTD (*Round Trip Delay*) fixé à $50 \mu\text{s} \approx 63$ octets.
- On fixe un *Slot Time* = $51.2 \mu\text{s}$ (64 octets) : une collision ne peut se produire que durant le *Slot Time*.
 - La station émettrice ne peut pas se déconnecter avant la fin du *Slot Time* pour avoir la certitude que la transmission se soit passée sans collision.
- Pour respecter la valeur RTD, on impose des limitations :
 - Longueur entre stations les plus distantes.
 - Nombre de segments.

Collision : détection

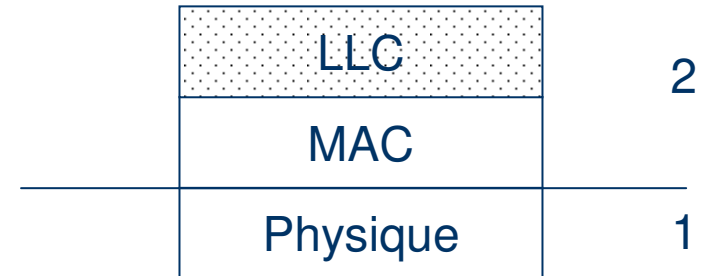
- Émetteur :
 - Émet au minimum après $9.6 \mu\text{s}$ d'intertrame.
 - Écoute le signal "détection de collision" pendant le *Slot Time* de $51.2 \mu\text{s}$ (64 octets) à partir du début d'émission.
 - S'arrête d'émettre quand il détecte une collision.
 - Longueur maximale trame erronée: 64 octets (**longueur minimale trame correcte : 72 octets**).
- Récepteur :
 - **Si réception d'une trame < 72 octets alors collision.**
- **Toute trame reçue de longueur < 72 octets est rejetée.**

Collision : réémission

- La station attend = $R \times 51,2 \mu s = R \times \textit{Slot Time}$, R entier.
- Elle émet à nouveau, 15 réémissions maximum.
- Si la 15ème réémission échoue, la station est déconnectée.

Différences entre 802.3 et Ethernet

- Ethernet :
 - Pas de sous couche LLC.
- Champ Type des trames Ethernet :
 - 2 octets.
 - Champs Type définis (protocole de niveau 3 utilisé) :
 - ❖ 0x0800 : IP.
 - ❖ 0x0806 : ARP.



ARP=Address Resolution Protocol

Conclusion

- Ethernet/IEEE 802.3 est une technologie éprouvée.
- C'est le protocole de réseau local de loin le plus répandu.
- Les problèmes qui restent sont connus :
 - Sécurité.
 - Confidentialité.
 - Priorité.

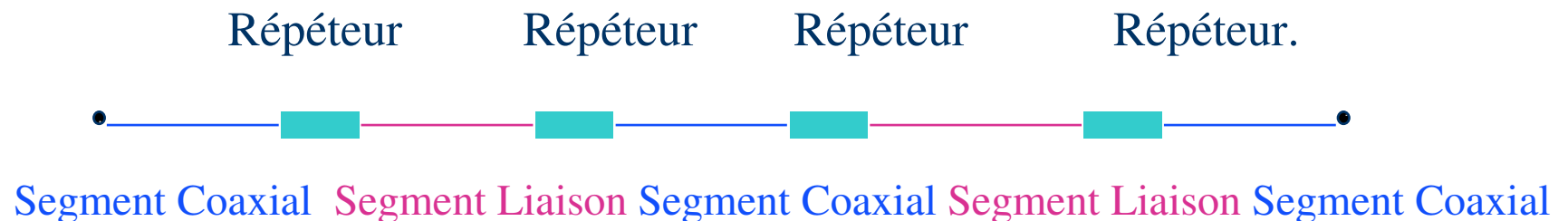
ETHERNET 10 Mb/s

10Base5 : câble coaxial épais

- 10 comme 10 Mb/s, Base = Baseband, 5 = 500 m.
- Coaxial 50 Ω . \varnothing 10 mm. Gros Ethernet (*thick*).
- Longueur maximale : 500 mètres.
- Topologie bus avec *transceivers*.

10Base5 : câble coaxial épais

- Plus long chemin :
 - 3 segments avec machines.
 - 2 segments de liaison sans machine.
 - 4 répéteurs.
 - 2.5 km si réseau tout en coaxial.



10Base5 : câble coaxial épais

- Segment coaxial :
 - Atténuation maximale de 17 dB/km à 10 MHz.
 - Signal :
 - Asynchrone en bande de base.
 - Code Manchester.
 - 2 niveaux : +2 V et -2 V.
 - Taux d'erreur bits TEB $< 10^{-8}$.
 - Longueur ≤ 500 m.
 - 100 *transceivers* maximum par segment.
 - Terminaison de 50 Ω .

10Base5 : câble coaxial épais

- Câble *transceiver* :
 - Appelé AUI (*Attachment Unit Interface*).
 - Relie le *transceiver* et le coupleur Ethernet de la station

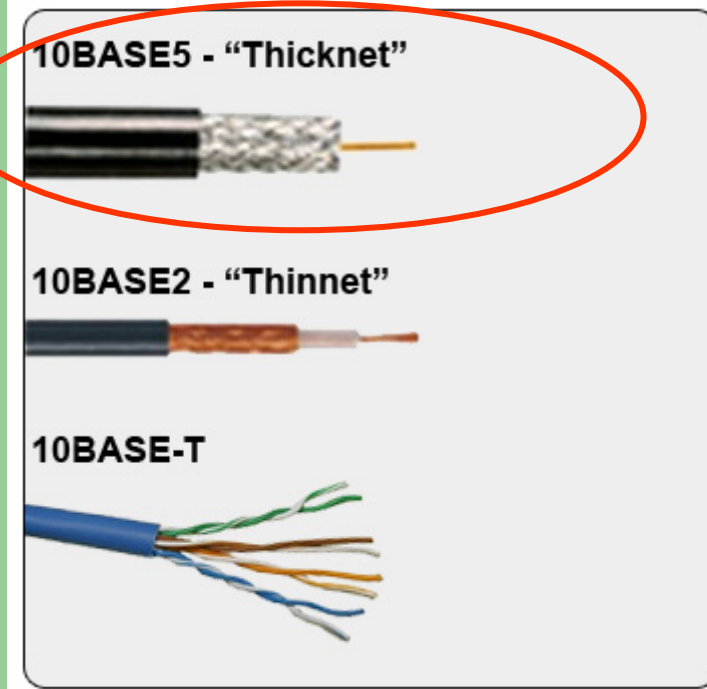
10Base5 : câble coaxial épais

- *Transceiver* :
 - Appelé MAU (*Medium Attachment Unit*).
 - Prise vampire :
 - Boîtier avec 2 pointes qui percent le câble.
 - 1 qui va jusqu'à l'âme (véhicule les données).
 - 1 qui va jusqu'à la protection (référence).
 - Avantages : pose sans couper le câble et sans interrompre le trafic.
 - Inconvénients : précautions pour ne pas couper l'âme.

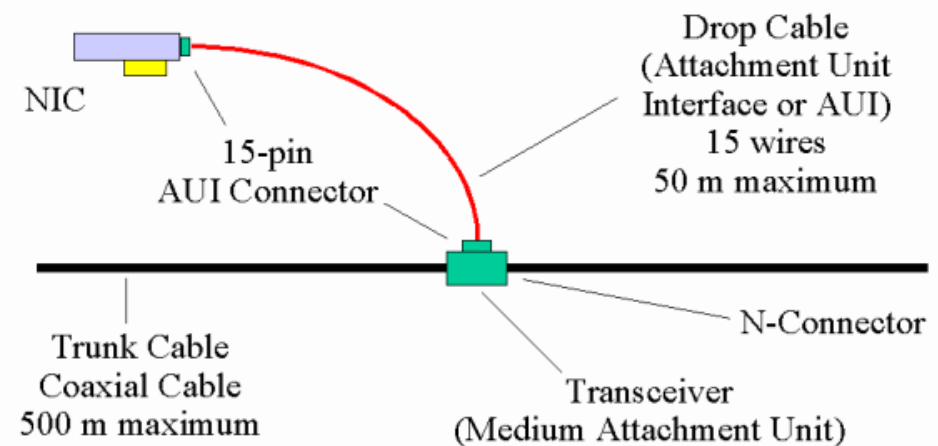
10Base5 : câble coaxial épais

- Avantages :
 - Très bien normalisé, depuis longtemps.
 - Pas de perturbations quand on ajoute une station.
- Inconvénients :
 - Coût, difficilement maniable, longueur des ajouts...
 - *Transceiver* vampire non récupérable quand on enlève une station.

10Base5 : câble coaxial épais



Ethernet 10Base5



10Base5 : câble coaxial épais



RE223 : Introduction aux réseaux et à Internet...

10Base2 : câble coaxial fin

- Coaxial 50 Ω . \varnothing 4.6 mm. Ethernet fin (*thin*).
- Longueur maximale : 185 m (10Base2 \approx 200 m).
- Topologie en bus avec 30 transceivers au maximum espacés de 0.5 m minimum.
- Connecteur BNC.

10Base2 : câble coaxial fin

- Avantages :
 - Moins cher.
 - Plus maniable que 10Base5.
- Inconvénients :
 - Si on enlève un *transceiver* (volontairement ou involontairement), on arrête tout le réseau.

10Base2 : câble coaxial fin

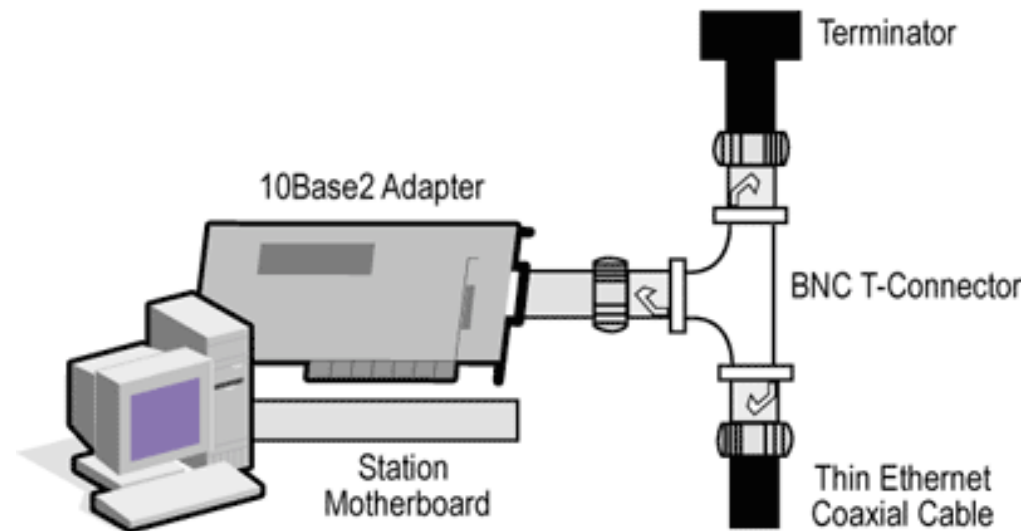
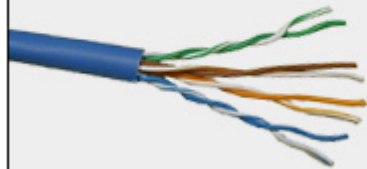
10BASE5 - "Thicknet"



10BASE2 - "Thinnet"



10BASE-T



10Base2 : câble coaxial fin



RE223 : Introduction aux réseaux et à Internet...

10BaseT : paires torsadées

- 10BaseT (*Twisted*).
- Câbles 2 paires torsadées de 100 Ω non blindés (UTP).
- Topologie physique en étoile (liaison point à point).
- Longueur maximale station/répéteur : 100 m.
- Connecteur RJ45 2 paires (émission/réception).

10BaseT : paires torsadées

- Avantages :
 - Câble universel utilisé pour de plus hauts débits Ethernet (100BaseT, 1000BaseT).
 - Insensible aux erreurs de manipulation des utilisateurs.
- Inconvénients :
 - Limitation en distance.

10BaseT : paires torsadées

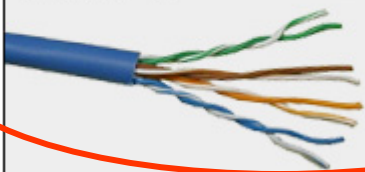
10BASE5 - "Thicknet"



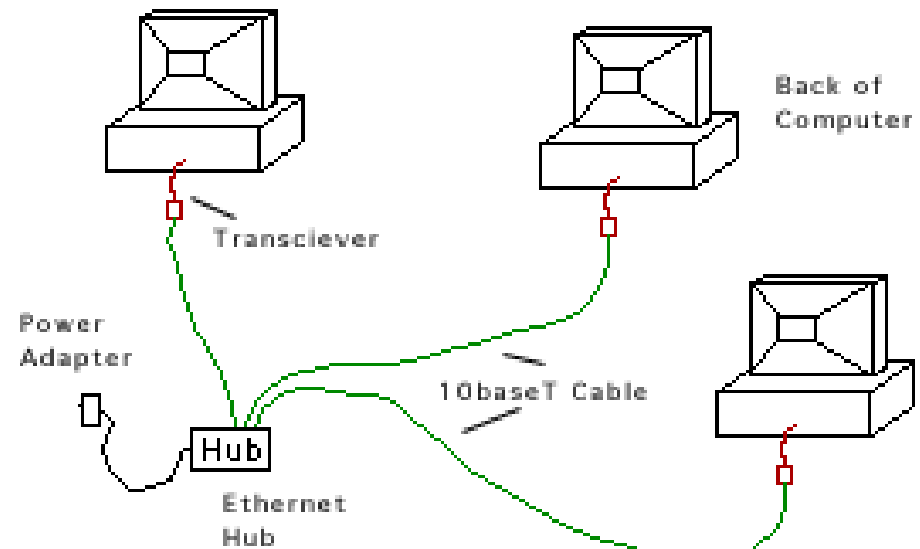
10BASE2 - "Thinnet"



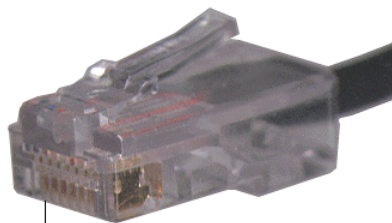
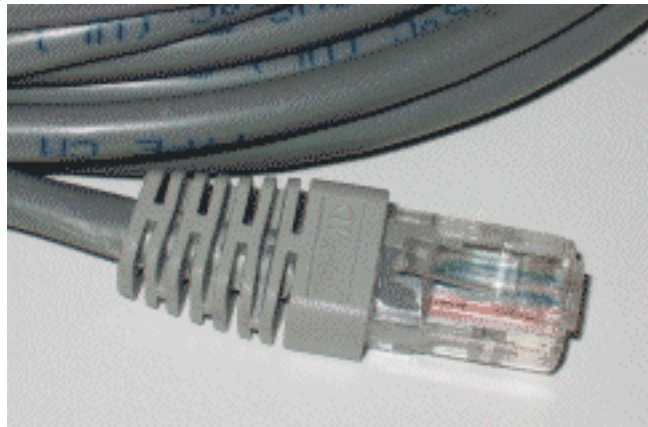
10BASE-T



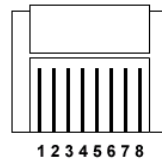
Simple Diagram of a 10baseT Network:



10BaseT : paires torsadées



Pin 1



RE223 : Introduction aux réseaux et à Internet...

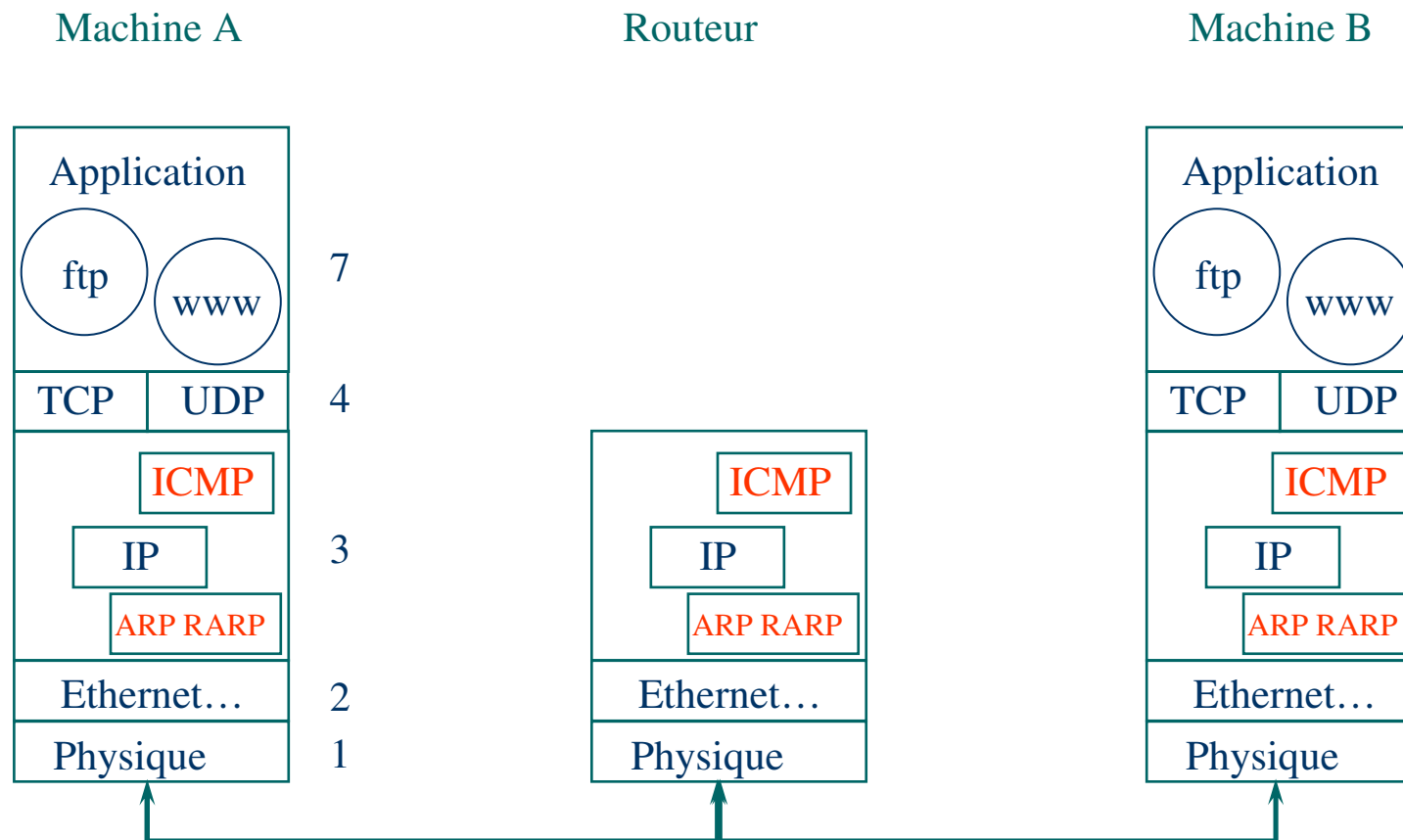
10Base? : bilan

- Le coaxial gros et fin sont obsolètes mais sont à la base de la théorie de la technologie Ethernet.
- Le présent est à la paire torsadée.

PARTIE 7

LES PROTOCOLES D'INTERNET

Protocoles Internet et modèle OSI

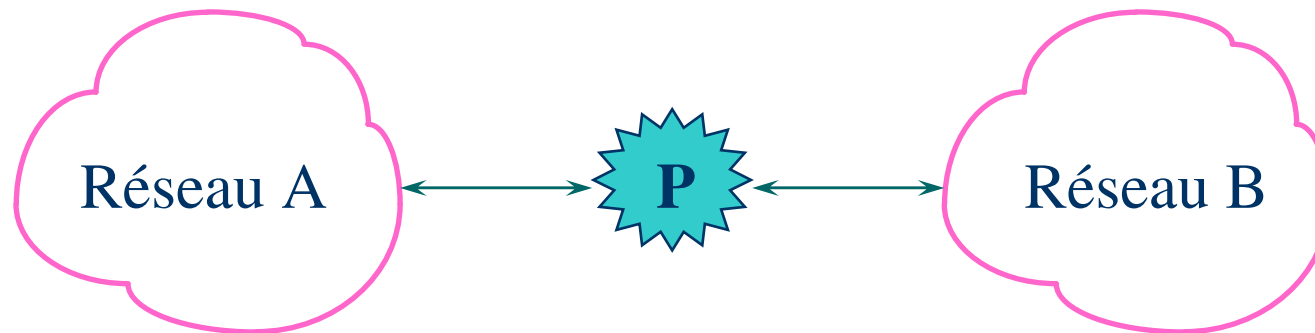


Protocoles Internet et modèle OSI

- Chaque machine aux extrémités implémente la pile complète de protocoles TCP/UDP/IP.
- Les protocoles ICMP, ARP et RARP sont des protocoles de gestion du bon fonctionnement du réseau Internet.
- Un routeur n'implémente qu'IP et est transparent aux données échangées par l'utilisateur.
- Les données sont donc traitées aux extrémités à l'émission et à la réception.

Interconnexion

- L'interconnexion permet de faire transiter des informations depuis un réseau vers un autre réseau par des nœuds spécialisés appelés passerelles (*gateway*) (qui peut être aussi routeur).
- Les passerelles possèdent une connexion sur chacun des réseaux.

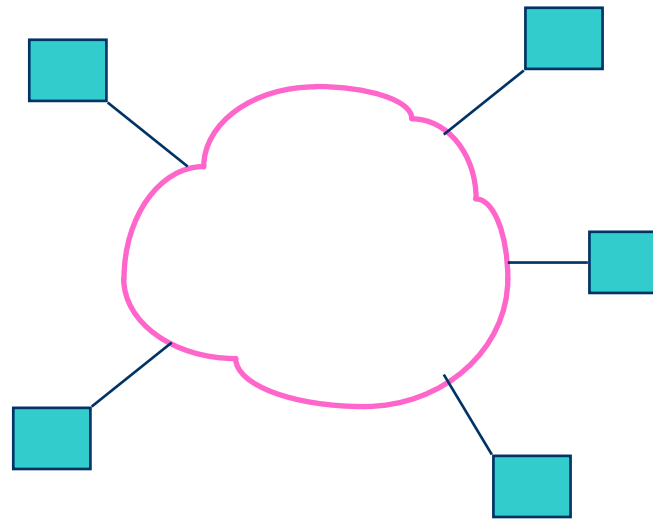


Interconnexion

- P1 transfère sur le réseau B les données circulant sur le réseau A et destinées aux réseaux B et C.
- P1 doit avoir connaissance de la topologie du réseau à savoir que C est accessible depuis le réseau B.
- Le routage n'est pas effectué sur la base de la machine destinataire mais sur la base du réseau destinataire.

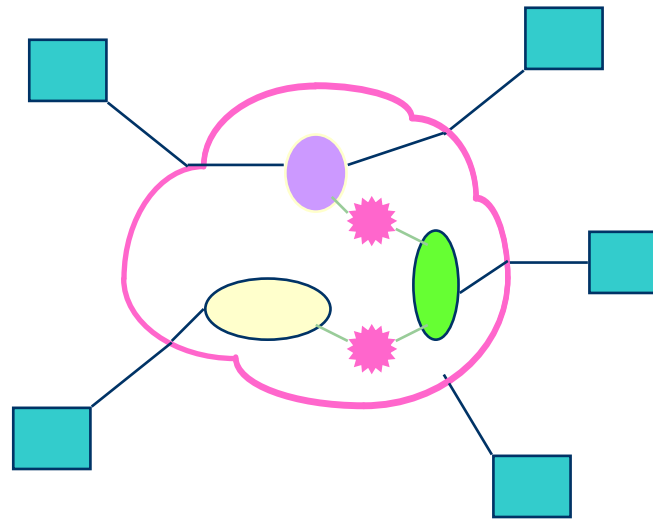


Interconnexion



Vue de l'utilisateur du réseau Internet

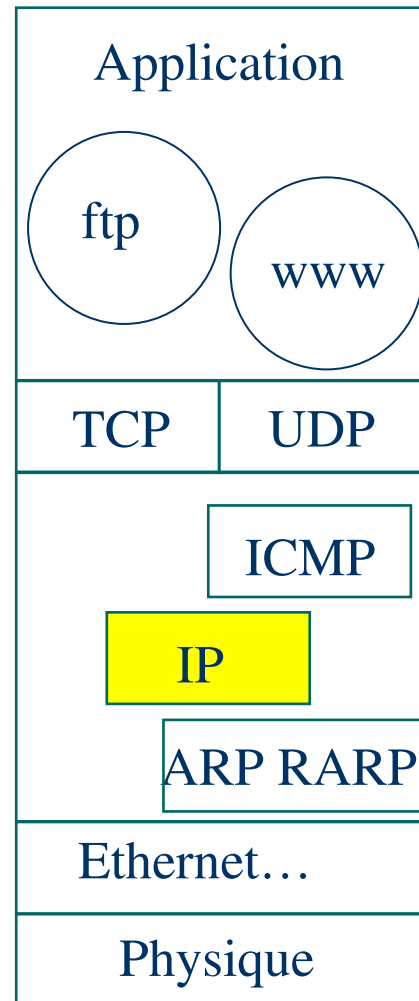
Interconnexion



Vue réelle du réseau Internet

Services d'Internet

- Interopérabilité au niveau des applications.
- Les utilisateurs invoquent les applications Internet sans avoir besoin de connaître les technologies d'Internet ni son architecture.
- Quelques services populaires :
 - Courrier électronique : smtp.
 - Transfert de fichiers : tftp, ftp.
 - Web : www.
 - Accès à des machines distantes : telnet, ssh.
 - ...



Protocole IP

- IP pour *Internet Protocol* (RFC 791).
- IP correspond au niveau 3 du modèle OSI.
- IP est au dessus de tout. C'est un protocole de convergence. Il fonctionne sur tout :
 - Ethernet : RFC 894.
 - ATM (*Asynchronous Transfer Protocol*) : RFC 1483, RFC 1577.
 - Liaison série RS232 : SLIP (RFC 1055), PPP (RFC 1353).
 - ...
- IP est un protocole sans connexion et non fiable vis-à-vis des données transportées.

Adressage IP

- L'adressage IP fournit un service de communication universel permettant à toute machine de communiquer avec toute autre machine.
- Une machine doit être accessible par d'autres machines.
- Une machine doit pouvoir être identifiée par :
 - Un nom symbolique (www.enseirb-matmeca.fr).
 - Une adresse qui doit être un identificateur universel de la machine.
 - Une route précisant comment la machine peut être atteinte.

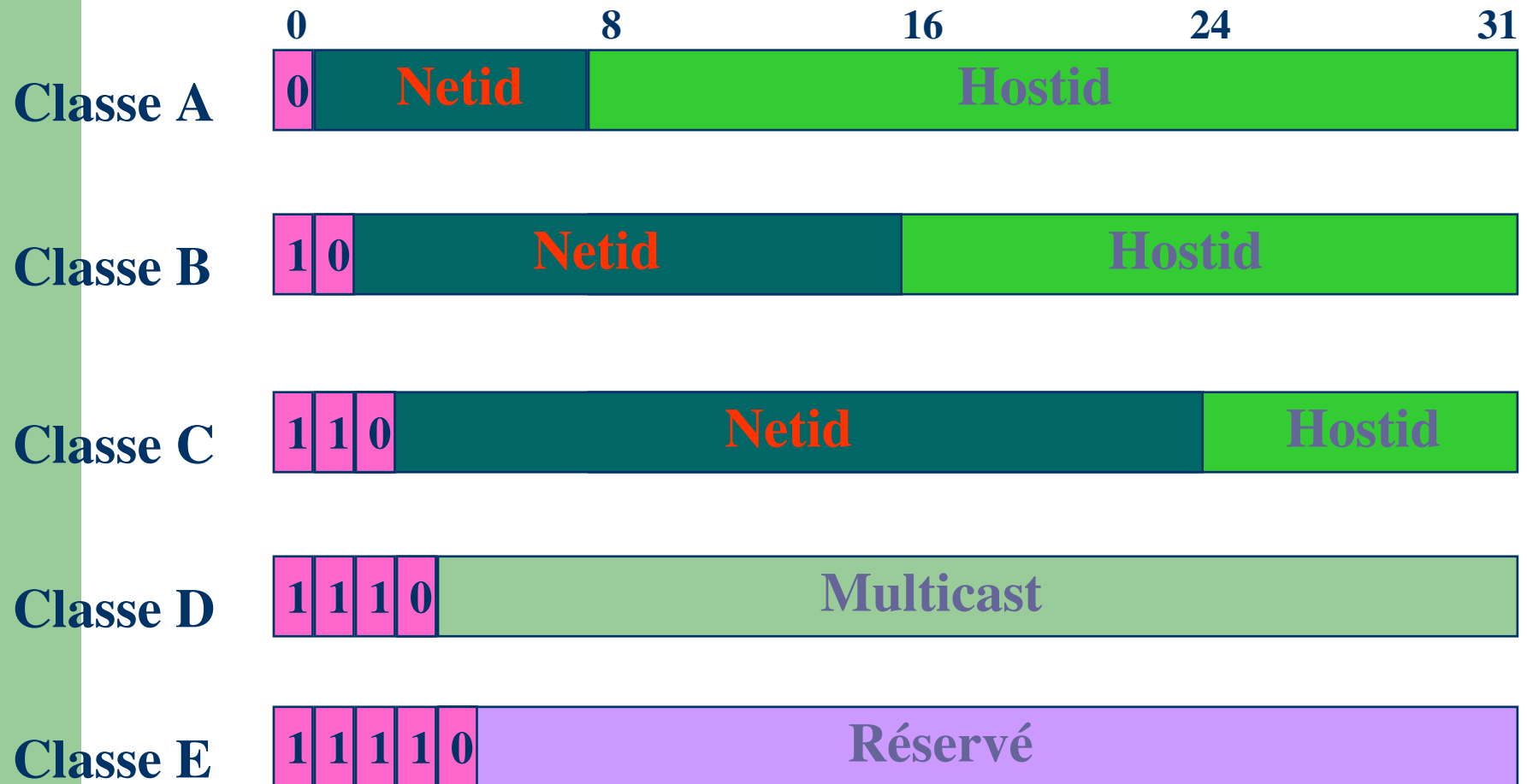
Adressage IP

- La solution est un adressage binaire compact assurant un routage efficace (32 bits).
- Adressage à plat par opposition à un adressage hiérarchisé comme avec le Réseau Téléphonique Commuté (RTC).
- Utilisation de noms symboliques pour identifier de façon humaine les machines (*Domain Name Service*).

Adressage IP

- Une adresse IP :
 - 4 octets (32 bits) ou *IP address*.
 - Constituée d'une paire (*netid* N, *hostid* H) où *netid* identifie un réseau et *hostid* identifie une machine de ce réseau.
 - Cette paire est structurée de manière à définir cinq classes d'adresses.
 - Notation en décimal pointé A.B.C.D.
- Elle doit être unique au monde (comme un numéro de téléphone RTC) :
 - Configurée par logiciel.
 - Associée à l'interface réseau.

Adressage IP



Adressage IP

- Classification :
 - Classe A : N.H.H.H
 - Classe B : N.N.H.H
 - Classe C : N.N.N.H
 - Classe D : multicast. Visioconférence.
 - Classe E : réservé. C'est une plage d'adresses perdue !

Adressage IP : classe A

- 7 bits pour le numéro de réseau *netid* N :
 - 1.0.0.0 à 126.0.0.0
- 24 bits pour le *hostid* H de la machine :
 - Environ 256^3 machines possibles sur un réseau de classe A.
- En France, il n’y a pas de réseaux de classe A :
 - 16.0.0.0 : MIT.



Adressage IP : classe B

- 16 bits pour le numéro de réseau *netid* N :
 - 128.1.0.0 à 191.255.0.0
- 16 bits pour le *hostid* H de la machine :
 - Environ 256^2 machines possibles sur un réseau de classe B.
- Quasiment épuisé en France !



Adressage IP : classe C

- 24 bits pour le numéro de réseau *netid* N :
 - 192.0.1.0 à 223.255.255.0
- 8 bits pour le *hostid* H de la machine :
 - 254 machines possibles sur un réseau de classe C.



Adressage IP : classe D

- Adresse de *multicast* pour une liaison point à multipoint : un émetteur et plusieurs récepteurs :
 - 224 à 231.
- Il n'y a pas de structuration car utilisé de façon ponctuelle sans organisation gérant la structuration.



Adressage IP : classe E

- Réservé pour un usage futur :
 - 239 à 254.

Adressage IP

- Adresses particulières :
 - Soi-même : 127.0.0.1 : *loopback* ou *localhost*.
 - Tous les bits à 0 pour le *hostid* : le réseau lui-même :
 - 130.190.0.0 désigne le réseau 130.190
 - Tous les bits à 1 pour le *hostid* : toutes les machines du réseau : diffusion ou *broadcast* :
 - 130.190.255.255
 - Une machine ne connaît pas son adresse IP (pas de disque dur). On utilisera RARP :
 - 0.0.0.0

Sous-réseau IP

- Découpage d'un réseau en entités plus petites :
 - Sous-réseau ou *subnet*.
 - Permet une meilleure structuration d'un site.
 - C'est décidé par l'administrateur réseau.
 - L'adresse de sous-réseau est prélevée sur la partie *hostid*.
 - La longueur est donnée en bits par l'administrateur.
 - Tous les équipements doivent intégrer la notion de sous-réseau.
 - L'interconnexion de sous-réseaux se fait par un routeur.



Sous-réseau IP

- Exemple de découpage en 3 sous-réseaux avec numérotation par le 3^{ème} octet :



- 190.30.0.0 est l'adresse réseau du site seule connue de l'extérieur.

Sous-réseau IP

- Le découpage en sous-réseaux est inconnu de l'extérieur.
- L'identification d'un sous-réseau passe par un *subnet mask* :
 - Bits *netid* à 1.
 - Bits *subnetid* à 1.
 - Bits *hostid* effectif à 0.
 - Exemple : réseau de classe B 130.190.0.0 :
 - *Subnet mask* par défaut : 255.255.0.0
 - Si un octet (octet 3) pour *subnetid*, le *subnet mask* est 255.255.255.0

Sous-réseau IP

- Exemple de l'ENSEIRB-MATMECA :
 - Adresse réseau de l'université de Bordeaux, sciences : 147.210.0.0 : réseau de classe B. 65536-2 machines publiques au plus.
 - 147.210.0.0 à 147.210.255.255 (147.210.0.0 est le réseau UB1, 147.210.255.255 est le *broadcast*).
- L'ENSEIRB-MATMECA a une plage d'adresse contigüe à partir de 147.210.18.0 pour (512-2) machines publiques au plus :
 - 147.210.18.0 à 147.210.19.255 (147.210.18.0 est le sous-réseau ENSEIRB- MATMECA d'UB1, 147.210.19.255 est le *broadcast* ENSEIRB-MATMECA).

Sous-réseau IP

- Pour (512-2) machines publiques au plus, le *hostid* effectif est sur 9 bits. Donc pour le *subnet mask*, on a :
 - 16 bits *netid* à 1.
 - 7 bits *subnetid* à 1.
 - 9 bits *hostid* effectif à 0.
 - 11111111.11111111.11111110.00000000 soit un *subnet mask* de 255.255.254.0 pour l'ENSEIRB-MATMECA.

Fonctions d'IP

- IP permet le transport de bout en bout de datagrammes IP.
- Chaque datagramme IP contient :
 - L'adresse IP de l'expéditeur.
 - L'adresse IP du destinataire.
 - Chaque contrôleur réseau possède son adresse IP :
 - Commande : `% ifconfig`
- Il faut donc connaître l'adresse IP d'une machine pour communiquer avec elle.

Fonctions d'IP

- IP est un mode non connecté. Chaque datagramme IP est traité indépendamment des autres.
- IP est un mode non fiable sans garantie de bonne réception des datagrammes IP. C'est un mode au mieux (*best effort*).
- IP assure le routage des datagrammes IP : savoir où envoyer chaque datagramme IP reçu. Les algorithmes de routage sont donc importants dans un réseau IP (non abordé en détail ici)...

Fonctions d'IP

- Le routage est le processus permettant à un datagramme IP d'être acheminé vers le destinataire lorsque celui-ci n'est pas sur le même réseau physique que l'émetteur.
- Le chemin parcouru est le résultat du processus de routage qui effectue les choix nécessaires afin d'acheminer le datagramme.
- Les routeurs forment une structure coopérative de telle manière qu'un datagramme IP transite de routeur en routeur jusqu'à ce que l'un d'entre eux le délivre à son destinataire.

Fonctions d'IP

- Les tables de routage IP donnent seulement les adresses réseau et non pas les adresses de machines.
- Une table de routage contient des couples (R, P) où R est l'adresse IP d'un Réseau destination et où P est l'adresse IP de la Passerelle correspondant au prochain saut dans le cheminement vers le réseau destinataire.
- La passerelle ne connaît pas le chemin complet pour atteindre la destination.

Fonctions d'IP

- IP gère la fragmentation.
- Un réseau avec un medium particulier qui véhicule des datagrammes IP possède un MTU (*Maximum Transfer Unit*). Le MTU pour Ethernet est de 1500 octets par exemple.
- C'est la machine destinataire qui réassemble les fragments générés dans le réseau.

Fonctions d'IP



Entête datagramme IP (20 octets)

EF1 et EF2 ont le bit More (M) positionné.
Le déplacement (dépl) est relatif au datagramme initial.



Entête Fragment : M=0; dépl=1200

Entête Fragment : M=1; dépl=600

Entête Fragment : M=1; dépl=0



Fonctions d'IP

- IP ne gère pas :
 - Le multiplexage de transferts de données.
 - La vérification du séquençement des datagrammes IP reçus.
 - La détection de pertes de datagrammes IP.
 - La retransmission en cas d'erreurs.
 - Le contrôle de flux (asservissement de l'extrémité la plus rapide à l'extrémité adjacente la plus lente).

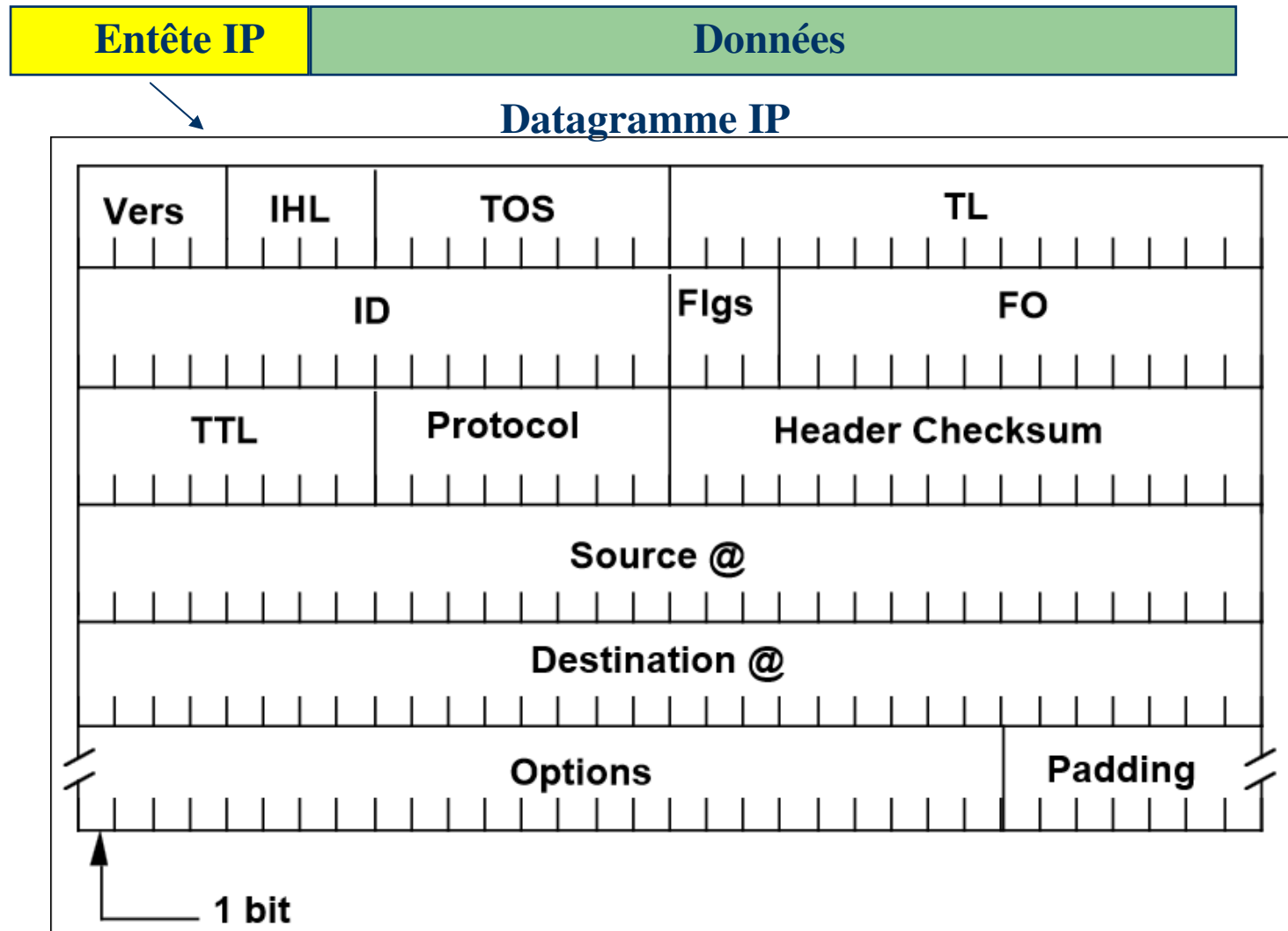
Datagramme IP

- Un datagramme possède :
 - Une entête de 20 octets.
 - Des données.



Datagramme IP

Datagramme IP



Datagramme IP

- Vers : version du protocole. Actuellement 4 pour IPv4.
- IHL : *Internet Header Length* :
 - Taille de l'entête IP en multiple de 4 octets. Ici 5.
 - Une entête d'un datagramme IP commence donc par la valeur 0x45 !
- TOS : *Type Of Service* :
 - Définition d'une qualité de service QoS (*Quality of Service*).
 - Utilisé pour la Voix sur IP (VoIP).
 - QoS comme : *low delay, high throughput...*

Datagramme IP

- *TL : Total Length :*
 - Longueur totale du datagramme IP en octets.
 - Au plus 64 Ko.
 - Limité par le MTU.
- *TTL : Time To Live :*
 - Durée de vie d'un datagramme IP en nombre de sauts.
 - Décrémenté de 1 à chaque routeur traversé.
 - Si TTL=0 alors le datagramme IP est détruit.
 - Evite les datagrammes IP fantômes dans des boucles de routage..

Datagramme IP

- *Protocol* :
 - Identifie le type des données transportées, c'est-à-dire les données du niveau supérieur.
 - 6 : TCP.
 - 17 : UDP.
 - 1 : ICMP.
- Démultiplexage en réception du datagramme IP et traitement par l'entité supérieure (TCP, UDP, ICMP...).

Datagramme IP

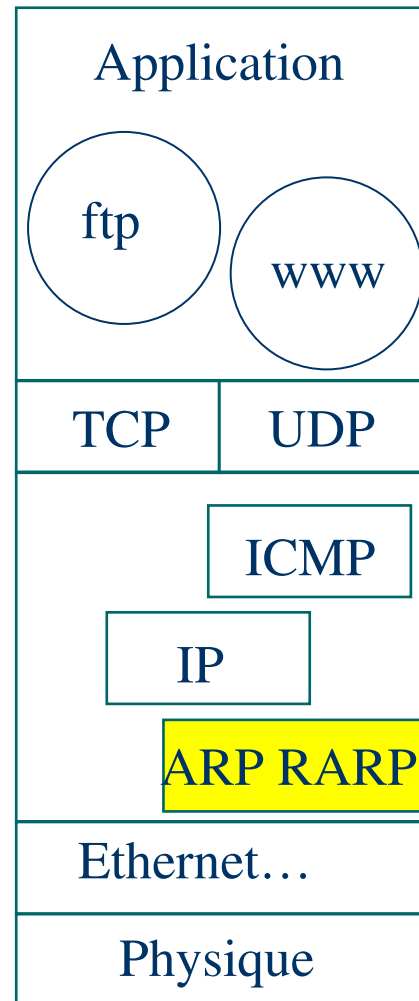
- Champs liés à la fragmentation IP :
 - ID : identification du datagramme IP :
 - Numérotation faite par l'émetteur pour identifier les fragments d'un même datagramme.
 - *Flags* :
 - 001 : il y a encore des fragments.
 - 000 : dernier fragment du datagramme IP.
 - ...
 - FO : *Fragment Offset* :
 - Position du fragment dans le datagramme IP fragmenté en unité de 8 octets.
 - Premier fragment = 0.
 - Le destinataire doit récupérer tous les fragments pour reconstruire le datagramme. Si un fragment manque, tout le datagramme IP est jeté.

Datagramme IP

- @ source : adresse IP de l'expéditeur.
- @ destination : adresse IP du destinataire.
- Ce sont les adresses IP des 2 machines distantes qui échangent des données.

Datagramme IP

- Options :
 - Variable en taille.
 - Pour des extensions futures.
 - En pratique, pas mis en œuvre.
- *Padding* :
 - Complète le champ Options pour avoir un multiple de 32 bits.
- *Header Checksum* :
 - Protection contre les erreurs de transmission. Calculé sur l'entête uniquement. Pourquoi change-t-il de valeur à chaque routeur traversé ?

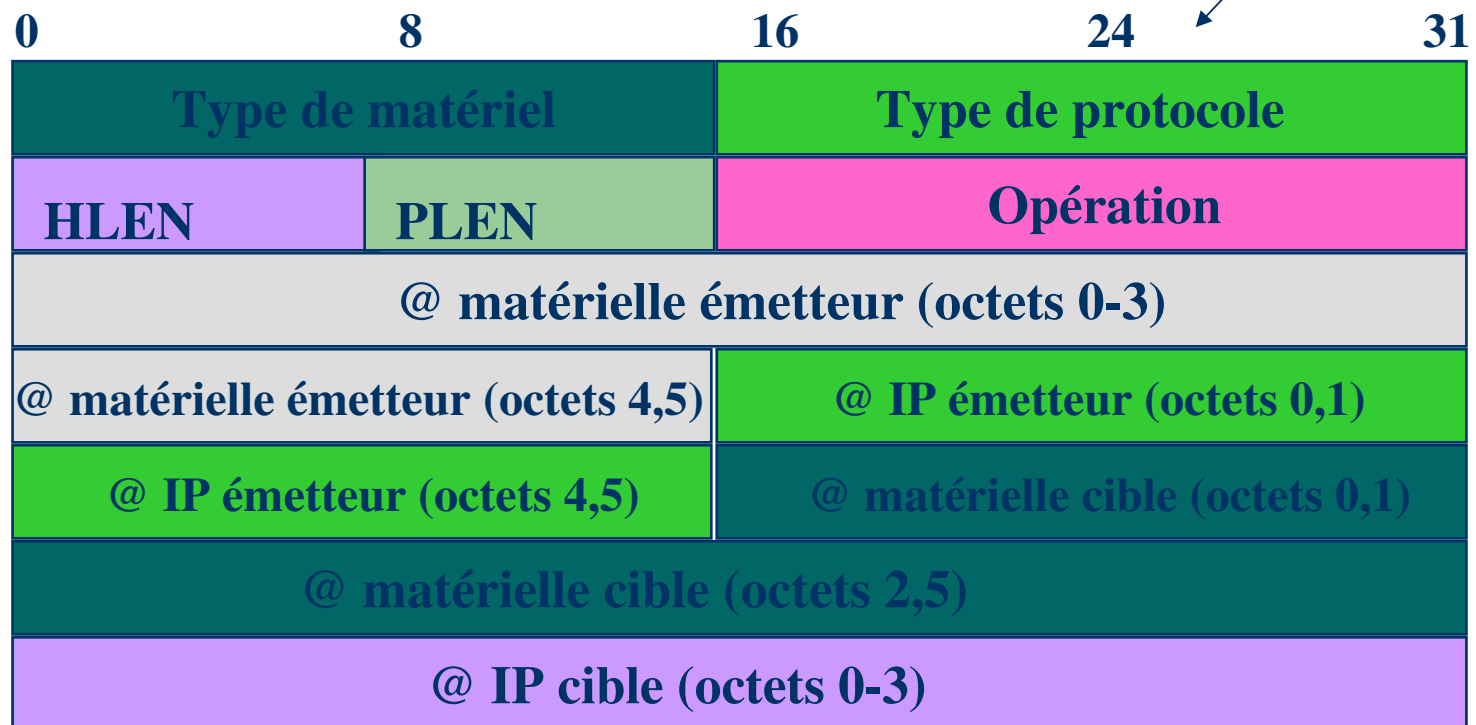


ARP

- ARP pour *Address Resolution Protocol* (RFC 826).



Opération ARP dans une trame Ethernet



ARP

- Type de matériel = 1 pour Ethernet
- Type de protocole = 0x0800 pour IP
- HLEN = 6 octets pour Ethernet
- PLEN = 4 octets pour IP
- Opération = :
 - 1 (requête ARP).
 - 2 (réponse ARP).
 - 3 (requête RARP).
 - 4 (réponse RARP).

ARP

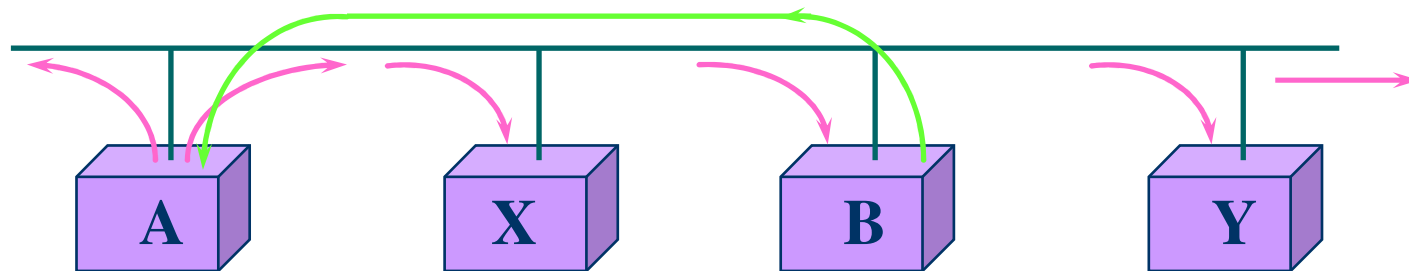
- ARP permet de trouver l'adresse matérielle (adresse physique) d'une machine d'un réseau local en fonction de son adresse IP.
- L'adresse IP est indépendante de l'adresse matérielle.
- Le couple (adresse physique, adresse IP) est stocké dans une table (cache) que l'on peut consulter via la commande :
 - % arp -a

ARP

- Une machine A veut envoyer un datagramme IP à une machine B sur un réseau Ethernet.
- Elle connaît son adresse IP mais pas son adresse physique ou adresse Ethernet ici.

ARP

- A envoie une trame Ethernet de *broadcast* embarquant une requête ARP qui demande l'adresse Ethernet de B.
 - Adresse destinataire FF:FF:FF:FF:FF:FF en indiquant l'adresse IP de B.
 - Toutes les machines reçoivent la requête.
 - Seule B répond par une réponse ARP avec son adresse Ethernet.

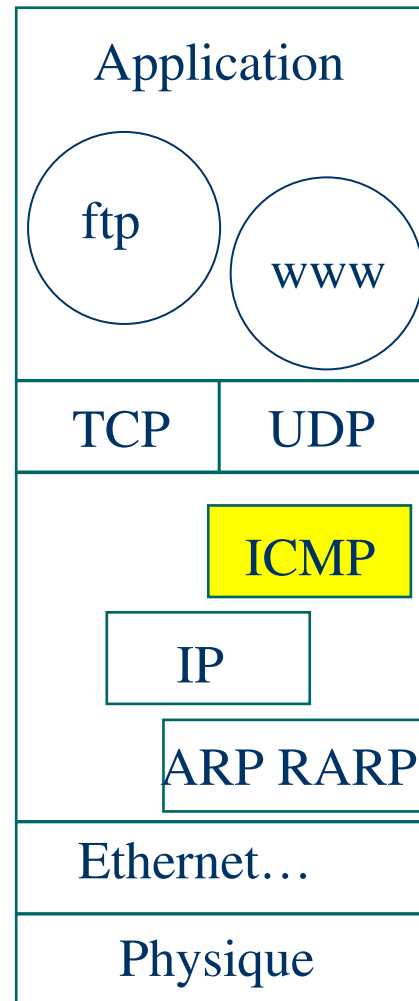


ARP

- Pour communiquer avec une machine, on a donc besoin jusqu'à présent :
 - @ IP source, @ IP destination.
 - @ physique source, @ physique destination.
 - Soit 2 couples de valeurs...
 - Est-ce suffisant ?

RARP

- RARP pour *Reverse Address Resolution Protocol* (RFC 903).
- Il s'agit de récupérer l'adresse IP en fonction de l'adresse physique.
- RARP est utilisé au démarrage de certains équipements :
 - Station sans disque dur.
 - Système embarqué.
- Le format d'une requête RARP est le même que celui d'ARP.

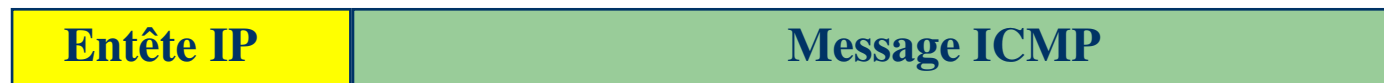


ICMP

- ICMP pour *Internet Control Message Protocol* (RFC 792).
- ICMP est un protocole de gestion du réseau IP.
- Il est implanté dans tous les équipements, routeurs inclus.
- Un message ICMP est envoyé soit par un équipement destinataire ou soit par un routeur quand :
 - Il s'aperçoit d'un problème dans un datagramme IP.
 - Il y a un problème de réseau (changement d'une table de routage).

ICMP

- Un message ICMP ne peut pas engendrer un autre message ICMP. Il ne demande aucune réponse.
- Un message ICMP est contenu dans un datagramme IP (champ *Protocol* égal à 1 dans l'entête IP).
- Un message ICMP possède 3 champs communs : TYPE, CODE, CHECKSUM

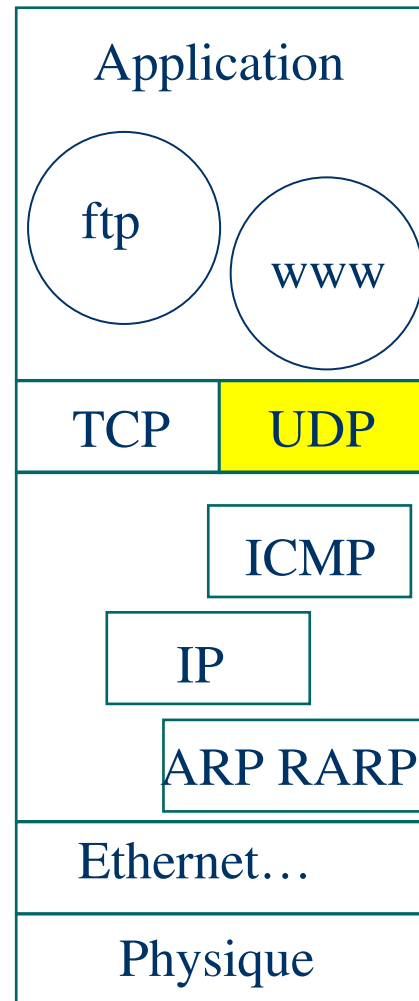


Message ICMP dans un datagramme IP

ICMP

- TYPE : 1 octet. Type de message.
- CODE : 1 octet. Informations complémentaires.
- CHECKSUM : 2 octets. Contrôle.

<u>TYPE</u>	<u>Message ICMP</u>	<u>TYPE</u>	<u>Message ICMP</u>
0	Echo Reply	13	Timestamp Request
3	Destination Unreachable	14	Timestamp Reply
4	Source Quench	15	Information Request (obsolete)
5	Redirect (change a route)	16	Information Reply (obsolète)
8	Echo Request	17	Address Mask Request
11	Time Exceeded (TTL)	18	Address Mask Reply
12	Parameter Problem with a Datagram		



Couche transport

- Il existe 2 protocoles de transport pour l'échange de données entre 2 machines :
 - TCP : *Transport Control Protocol*
 - Protocole en mode connecté et fiable.
 - UDP : *User Datagram Protocol*
 - Protocole en mode non connecté et non fiable.

Couche transport

- Dans les 2 cas, une application informatique est identifiée par un identificateur unique appelé port (entier sur 16 bits).
- Un numéro de port identifie l'application locale et un numéro de port identifie l'application distante.
- On appelle *socket* le couple de valeurs (@ IP, numéro de port) :
 - (147.210.18.10, 23) = application serveur telnet sur la machine distante 147.210.18.10.
- La combinaison de 2 *sockets* identifie complètement l'échange de données UDP ou TCP entre 2 applications distantes.

Couche transport

- Combinaison de 2 *sockets* :
 - (147.210.18.10, 32420) vers (147.210.19.62, 80).
 - Echange de données vers le serveur Web de l'ENSEIRB-MATMECA.
- Il existe des numéros de port réservés (*well-known port*) :

– 7	ECHO	Echo
– 13	DAYTIME	Daytime
– 21	FTP	File transfert protocol control
– 22	SSH	Secure Shell
– 23	TELNET	Terminal à distance
– 69	TFTP	Trivial File transfert protocol
– 80	WWW	World Wide Web
– ...		

Couche transport

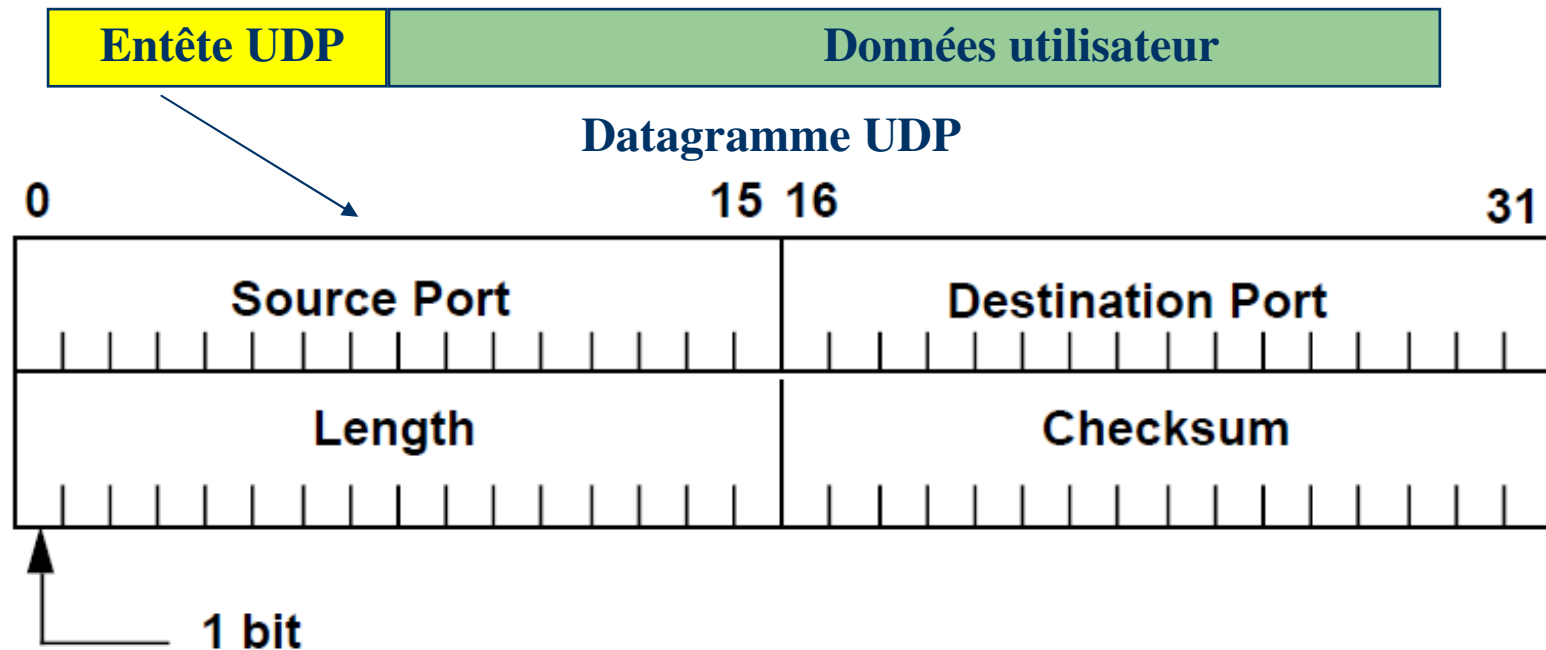
- Le mode d'échange de données est du type client/serveur :
 - Le client se voit attribuer un numéro de port non affecté élevé (32000...).
 - Le serveur a un numéro de port *well-known*.
- Les numéros de port réservés sont définis dans le fichier `/etc/services`.
- Sous Windows, la configuration réseau IP est « empruntée » à celle d'UNIX :
`C:\Windows\System32\drivers\etc\services`

Couche transport

- Pour communiquer avec une machine, on a donc besoin :
 - @ physique source, @ **physique destination**.
 - @ IP source, @ **IP destination**.
 - Port source, **port destination**.
 - Soit 3 couples de valeurs...
- L'@ physique destination est récupérée automatiquement par la machine source avec le protocole ARP.
- Sur ces 6 valeurs, seules les 2 valeurs destination seront à renseigner, les 3 valeurs source étant renseignées par l'API de programmation réseau...

UDP

- UDP pour *User Datagram Protocol* (RFC 768).
- UDP est un mode de transport sans connexion, non fiable et sans garantie qui utilise IP pour le transport.



UDP

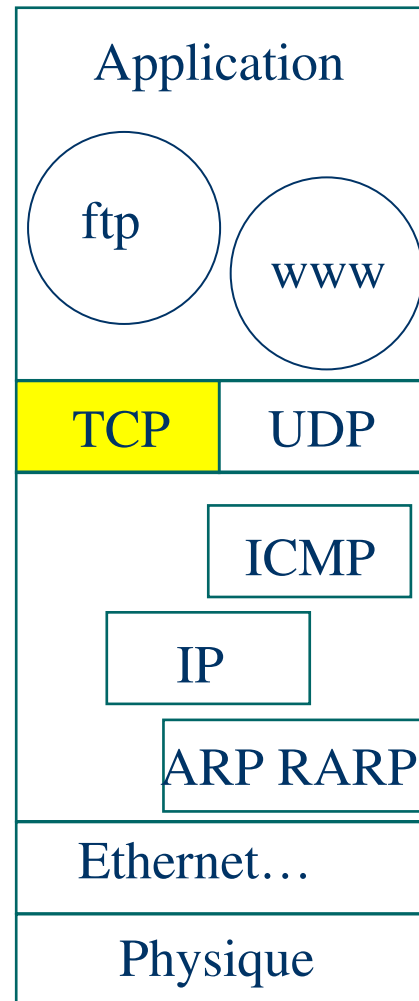
- *Source Port* (16 bits) :
 - Port source.
- *Destination Port* (16 bits) :
 - Port destination.
- *Length* (16 bits) :
 - Longueur totale du datagramme UDP en octets.
 - Au plus 64 Ko.
- *Checksum* (16 bits) :
 - Optionnel. 0.

UDP

- L'entête UDP fait 8 octets.
- En réception, le port destination permet de savoir quelle application réveiller pour traiter les données reçues.
- Un datagramme UDP est inclus dans le champ données d'un datagramme IP (encapsulation).

UDP

- UDP est un protocole de transport non fiable.
- Il n'assure pas :
 - Un mode connecté.
 - La retransmission des données s'il y a des erreurs ou des pertes.
 - Le séquençement des données.
 - Le contrôle de flux.
- TCP corrige ces défauts au détriment d'un protocole plus lourd et moins réactif.
- Pour transmettre des données rapidement sur un réseau local, UDP est préférable à TCP.



TCP

- TCP pour *Transmission Control Protocol* (RFC 793).
- TCP est un mode de transport avec connexion, fiable et avec garantie qui utilise IP pour le transport.
- Comme UDP, TCP n'est pas utilisé dans un routeur IP mais dans les machines distantes.

TCP

- TCP permet un transport des données :
 - De bout en bout entre applications distantes.
 - En mode connecté : phase de connexion, de transfert de données, de libération.
 - Sans erreurs de transmission : contrôle et retransmission si nécessaire.
 - Sans perte de données : numérotation des octets émis.
 - Ordonné.
 - Système d’acquittement des octets bien reçus.
 - Contrôle de flux (par fenêtre glissante).
 - *Full duplex*.

TCP

- TCP traite les données utilisateur du niveau supérieur comme un flux d'octets non structuré.
- TCP découpe ce flux d'octets en segments TCP :
 - Taille maximale d'un segment TCP : 64 Ko.
 - La taille réelle dépend du médium utilisé (MTU).

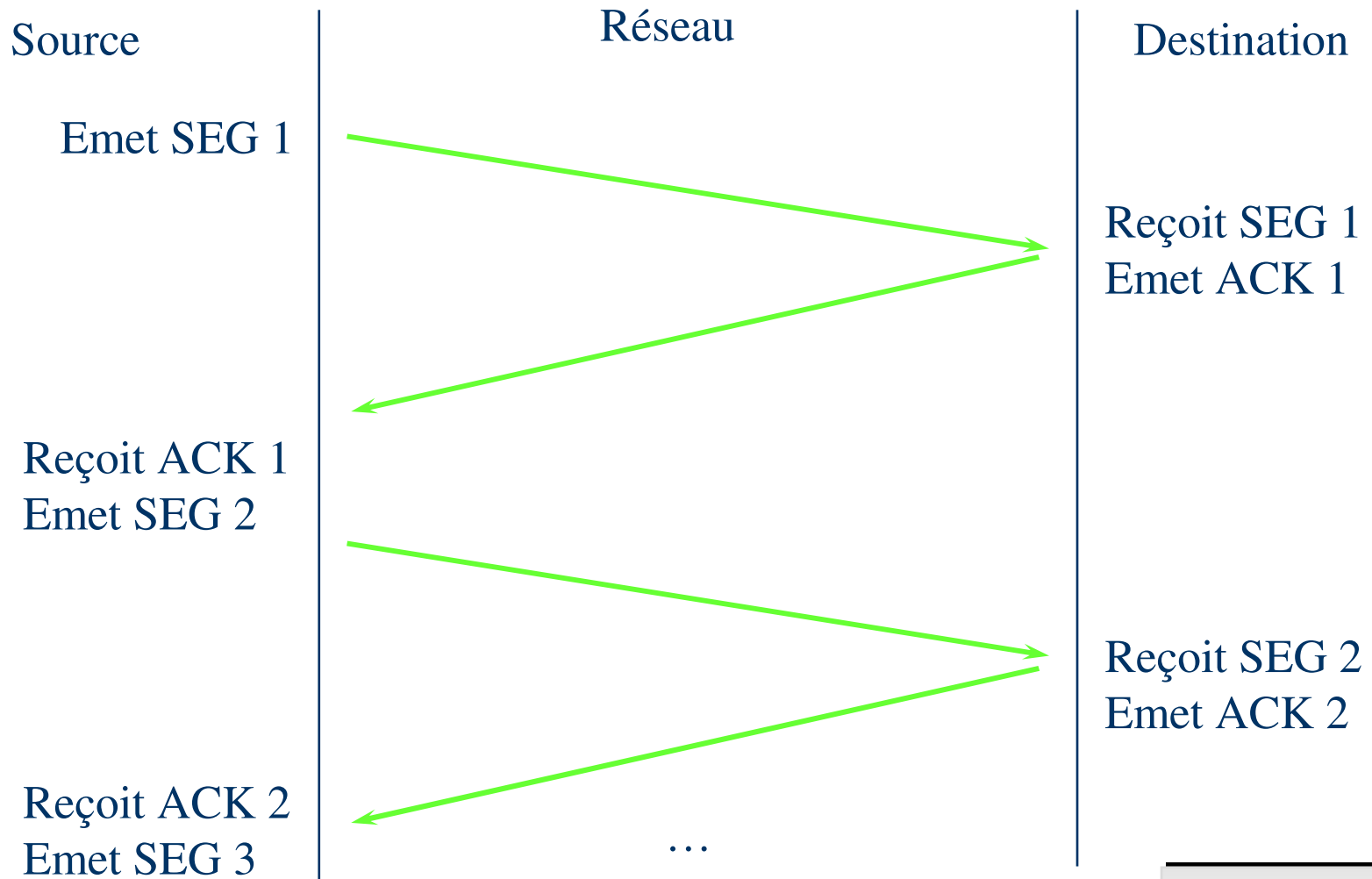
TCP

- Un segment TCP est contenu dans le champs données d'un datagramme IP (encapsulation).
- Des segments TCP sont définis pour :
 - Etablir la connexion.
 - Transférer des données.
 - Fermer la connexion.
 - Gérer la connexion (contrôle de flux...).

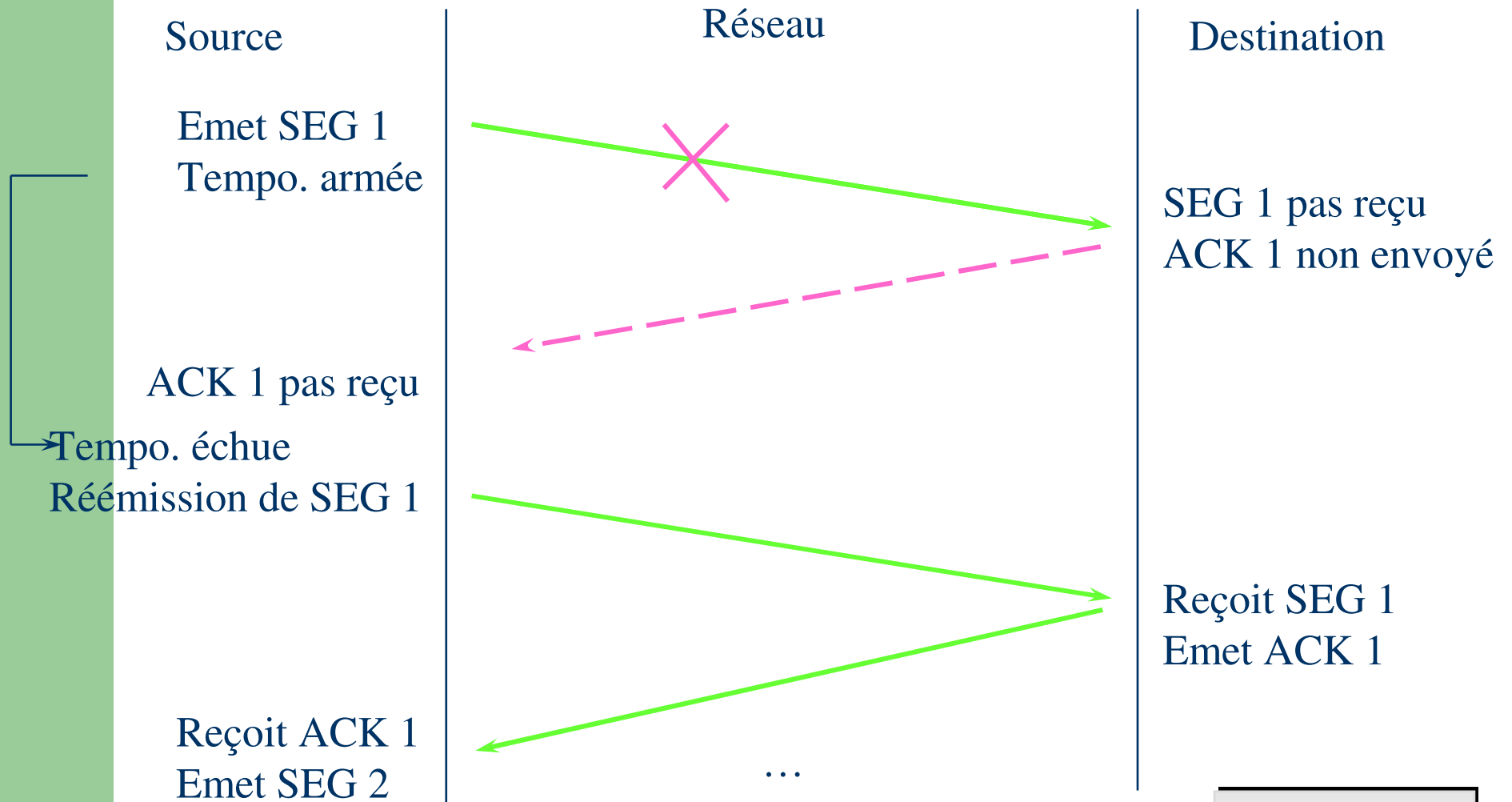
Efficacité d'un transfert

- Mécanisme de transfert *Send and Wait*. On transmet un segment puis on attend son acquittement avant de transmettre le suivant.

Efficacité d'un transfert



Efficacité d'un transfert



Efficacité d'un transfert

- Si l'on n'a pas d'acquiescement à l'arrivée à échéance du *timer*, on retransmet et on attend de nouveau.
- Le réseau est très mal exploité car il n'est utilisé que pendant la transmission.
- On introduit la notion de fenêtre d'anticipation ou fenêtre glissante pour plus d'efficacité (*sliding window*).

Efficacité d'un transfert

- Mécanisme de la fenêtre glissante :

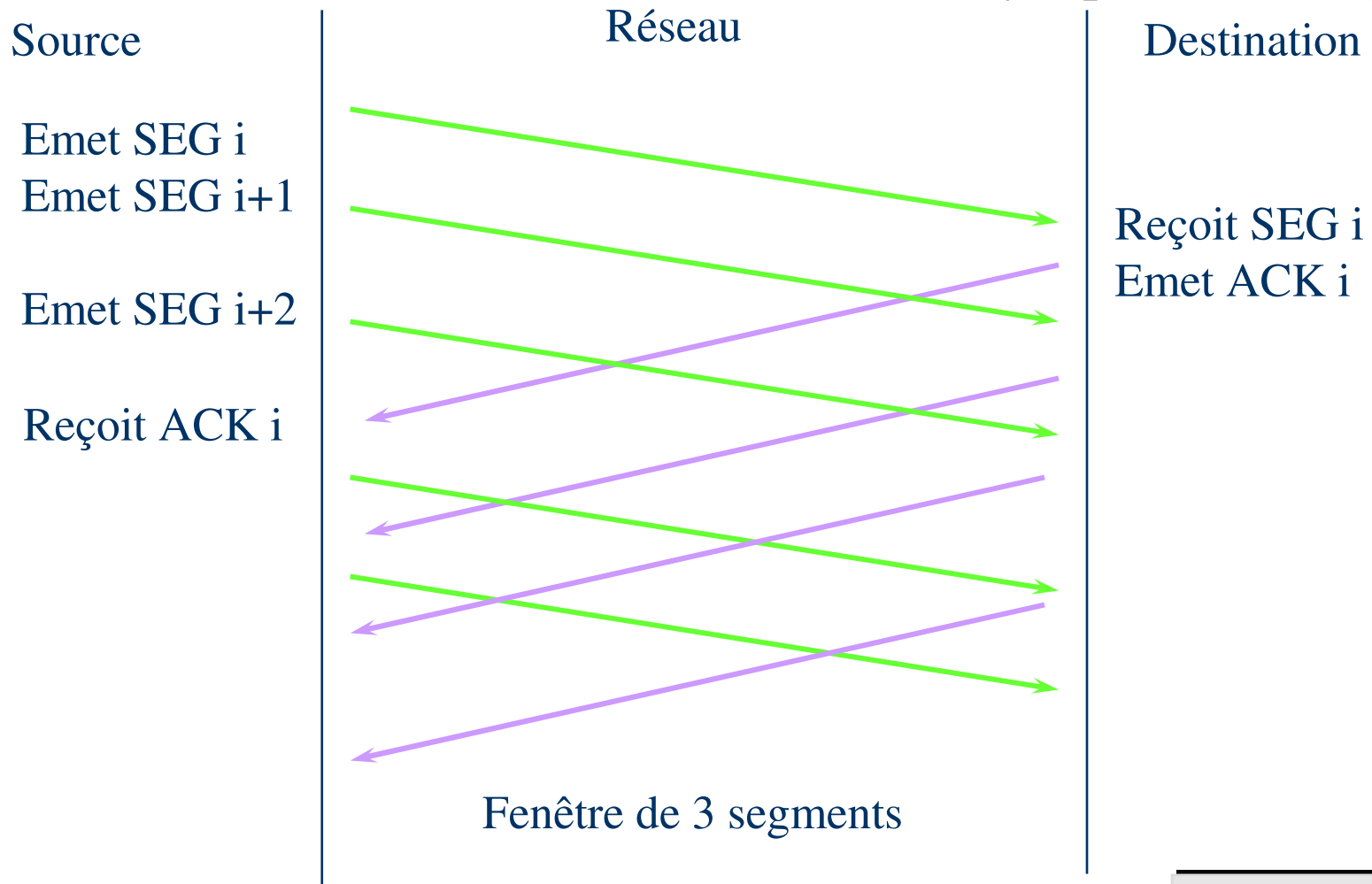


- L'émetteur peut envoyer au plus 3 segments avant de recevoir un acquittement (fenêtre de 3 segments). Avec l'arrivée de l'acquittement du segment 1, la fenêtre glisse de 1 (segment) vers la droite :



Efficacité d'un transfert

- Si la taille de la fenêtre est suffisante, il n'y a pas de blocage...



TCP

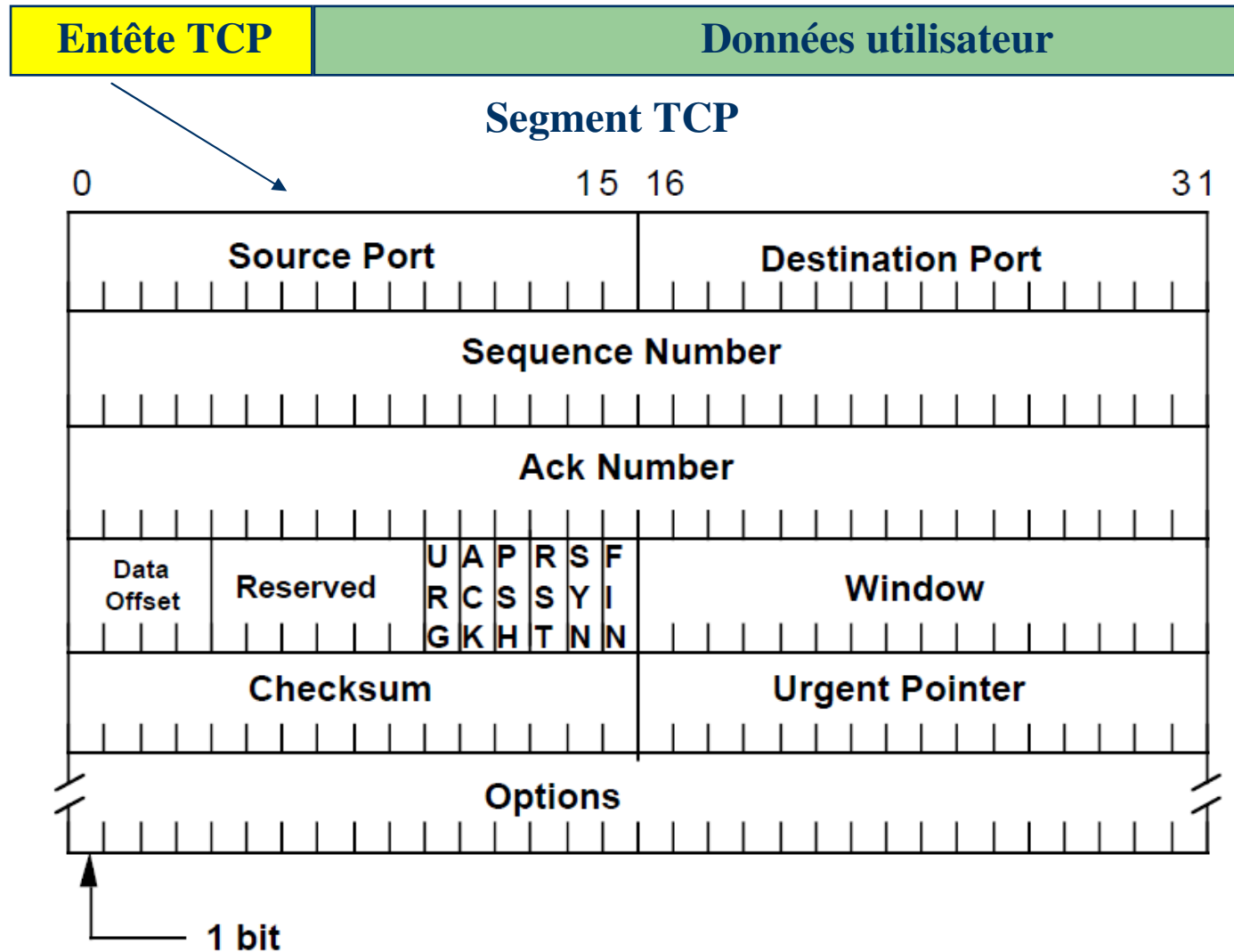
- TCP utilise le système de fenêtre glissante non pas sur les segments mais sur les **octets** :
 - Numérotation des octets.
 - Numérotation séquentielle.
 - L'émetteur gère 3 pointeurs sur les octets.
 - Ce mécanisme s'applique aux 2 extrémités de la connexion.



TCP

- La taille de la fenêtre glissante n'est pas fixe mais varie en cours de la connexion.
- Dans le segment TCP d'acquiescement, le destinataire précise le nombre d'octets qu'il est prêt à accepter. La fenêtre de l'émetteur est modifiée alors.
- On commence avec une petite valeur pour la taille de la fenêtre puis on augmente (*slow start*) si tout va bien.
- Pour réduire le débit écoulé, il suffit de réduire la taille de la fenêtre ce qui contribue au contrôle du flux échangé !

TCP



TCP

- La taille de l'entête TCP fait 20 octets. Combinée à IP, cela fait $20+20=40$ octets, ce qui est beaucoup !
- Il existe des segments TCP pour :
 - Etablir la connexion.
 - Transférer les données utilisateur.
 - Envoyer des acquittements de données correctement reçues.
 - Ajuster la taille des fenêtres.
 - Fermer la connexion.

TCP

- *Source Port* (16 bits) :
 - Port source.
- *Destination Port* (16 bits) :
 - Port destination.
- *Sequence Number* (32 bits) :
 - Valeur en octets.
 - Numérote le numéro du dernier octet émis.
 - Assure le bon séquençement.

TCP

- *Acknowledge Number* (32 bits) :
 - Valeur en octets.
 - Numérote le numéro du dernier octet reçu + 1. Donne donc le numéro du prochain octet attendu.
 - Acquiescement tous les octets reçus précédemment.
 - Assure le bon séquençement.
- *Data offset* (4 bits) :
 - Taille de l'entête TCP en multiple de 32 bits. 5 ici.

TCP

- *URG* (1 bit) :
 - Urgent. Tag d'un segment TCP urgent (CTRL C dans ssh).
- *ACK* (1 bit) :
 - Tenir compte de *l'Acknowledge Number*.
- *PSH* (1 bit) :
 - Délivrer immédiatement les données. L'émetteur ne prévoit pas d'envoyer d'autres données dans l'immédiat.
- *RST* (1 bit) :
 - Reset. Reprendre une connexion au début.
- *SYN* (1 bit) :
 - Etablissement d'une connexion TCP.
- *FIN* (1 bit) :
 - Fin d'une connexion TCP.

TCP

- *Window* (16 bits) :
 - Taille de la fenêtre en octets.
 - Nombre d'octets maximum pouvant être émis sans en attendre l'acquittement.
 - Assure le contrôle de flux.
- *Checksum* (16 bits) :
 - Somme de contrôle sur l'entête et les données du segment TCP.

TCP

- *Urgent Pointer* (16 bits) :
 - Indique le numéro du dernier octet urgent quand URG=1.
- *Options* (32 bits) :
 - Champ optionnel. Pas utilisé.

TCP

- Les accusés de (bonne) réception (des données précédemment reçues) :
 - Peuvent être envoyés en même temps que des données : technique du *piggy backing*.
 - Ne sont pas obligatoires à chaque segment TCP reçu.
- TCP s'adapte sans paramétrage à tous les débits, à tous les temps de réponse donc à tous les réseaux (taille de fenêtre dynamique, *timers* dynamiques...).

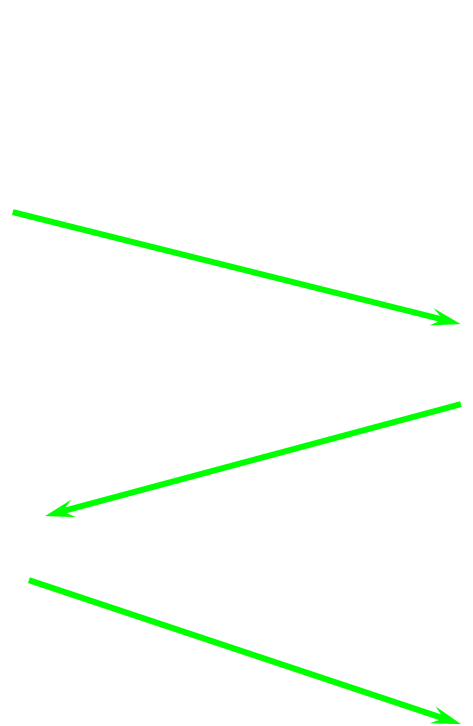
TCP

- Etablissement d'une connexion TCP :

TCP source

SYN, SN=x

ACK, AN=y+1



TCP destination

SYN, SN=y
ACK, AN=x+1

TCP

- Libération d'une connexion TCP :

TCP source

FIN, SN=x

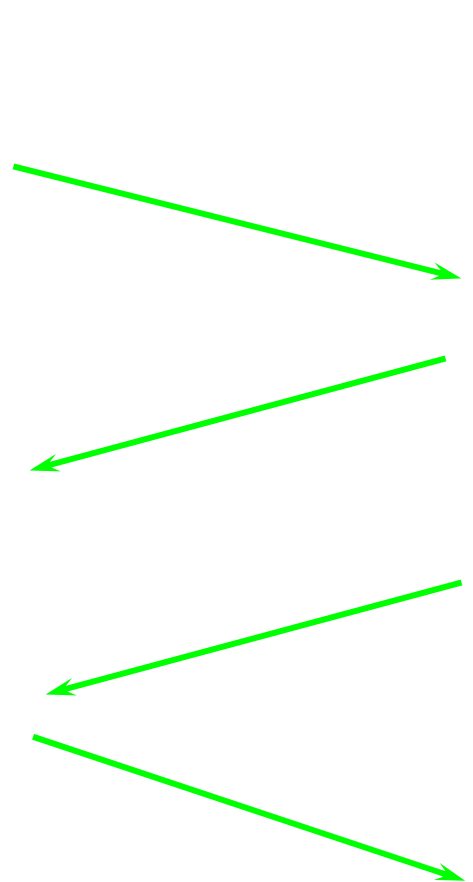
TCP destination

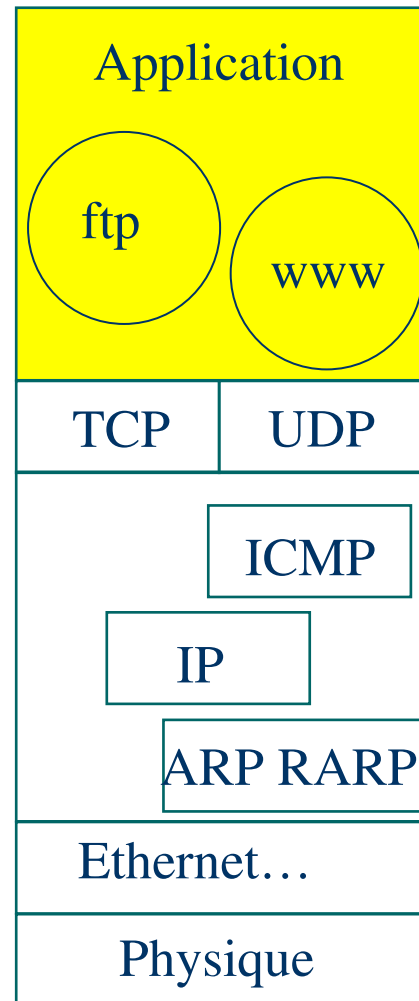
ACK, AN=x+1

Fin application

FIN, SN=y

ACK, AN=y+1





Configuration réseau sous UNIX

- Il s'agit de voir les principaux fichiers de configuration d'un réseau IP sous UNIX.
- Il s'agit aussi de voir les principales commandes de configuration réseau IP sous UNIX.

Configuration réseau sous UNIX

- Ces fichiers et ces commandes existe sous Windows. Il suffit d'ouvrir une « invite de commandes » !!! :

```
Microsoft Windows [version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. Tous droits réservés.

C:\Users\Patrice>netstat -nr
=====
Liste d'Interfaces
11...00 1b fc be 9f 03 .....Marvell Yukon 88E8056 PCI-E Gigabit Ethernet Contr
oller
17...08 00 27 00 2c ea .....VirtualBox Host-Only Ethernet Adapter
1 .....Software Loopback Interface 1
25...00 00 00 00 00 00 00 e0 Carte Microsoft ISATAP
12...00 00 00 00 00 00 00 e0 Teredo Tunneling Pseudo-Interface
16...00 00 00 00 00 00 00 e0 Carte Microsoft ISATAP #2
=====

IPv4 Table de routage
=====
Itinéraires actifs :
Destination réseau      Masque réseau      Adr. passerelle    Adr. interface  Métrique
0.0.0.0                0.0.0.0            192.168.0.254      192.168.0.1     266
127.0.0.0              255.0.0.0          On-link            127.0.0.1       306
127.0.0.1              255.255.255.255    On-link            127.0.0.1       306
127.255.255.255        255.255.255.255    On-link            127.0.0.1       306
192.168.0.0            255.255.255.0      On-link            192.168.0.1     266
192.168.0.1            255.255.255.255    On-link            192.168.0.1     266
```

Configuration réseau sous UNIX

- Fichier `/etc/hosts` :
 - Donne l'association adresse IP – nom symbolique d'une machine.
 - Exemple : `127.0.0.1 localhost`
- Fichier `/etc/networks` :
 - Donne l'association adresse du réseau – nom symbolique du réseau.
 - Exemple : `147.210.18.0 enseirb.fr`

Configuration réseau sous UNIX

- Fichier `/etc/services` :
 - Donne l'association numéro de port – nom symbolique du service.
 - Exemple : `ftp 21/tcp`
- Fichier `/etc/inetd.conf` ou `/etc/xinetd.conf` :
 - Configuration du serveur *inetd* de services. Serveur de serveurs.
 - Permet de valider ou pas un service Internet sur la machine (ftp, telnet...).

Configuration réseau sous UNIX

- Commande `ifconfig` :
 - Configuration de l'interface réseau.
 - Exemple : `% ifconfig -a`
- Commande `netstat` :
 - Affichage de statistiques réseau : table de routage, échanges UDP, TCP.
 - Exemple : `% netstat -na`
 - Exemple : `% netstat -nr`

Configuration réseau sous UNIX

- Commande `route` :
 - Affichage de la table de routage.
 - Exemple : `% route`
- Commande `arp` :
 - Affichage du cache ARP.
 - Exemple : `% arp -a`

Configuration réseau sous UNIX

- Commande `ping` :
 - Permet de voir si une machine cible est « vivante ».
 - Exemple : `% ping www.enseirb.fr`
- Commande `traceroute` :
 - Trace le chemin emprunté par les données dans le réseau IP à destination d'une machine cible.
 - Exemple : `% traceroute www.enseirb.fr`

Les protocoles d'Internet : bilan

- Les protocoles TCP/IP et à moindre mesure UDP/IP sont les protocoles les plus utilisés actuellement.
- C'est d'autant plus vrai avec l'Internet des objets et les objets connectés où chaque objet intègre une pile TCP/IP pour communiquer.
- Si l'on intègre dans l'objet un système d'exploitation comme Linux « au hasard », la pile TCP/IP est là de facto. Mais c'est une autre histoire...

Les protocoles d'Internet : bilan

- Avantages de la suite de protocoles TCP/UDP/IP :
 - Gratuit.
 - Indépendance vis-à-vis des constructeurs de matériels.
 - Disponible sur tout type de matériel : systèmes embarqués, objets connectés...
 - Facile à mettre en œuvre.
 - Intégré dans les systèmes d'exploitation modernes.
 - Epruvé depuis longtemps.
 - Bien documenté.
 - Simple mais efficace.

Les protocoles d'Internet : bilan

- Inconvénients de la suite de protocoles TCP/UDP/IP :
 - Pas une norme internationale. Différentes souches implantées ce qui posait des problèmes d'interopérabilité et de sécurité.
 - Classe B épuisée. IPv6, la solution ! L'adresse IPv6 fait 128 bits soit 340282366920938463463374607431768211456 (340 sextillions) d'adresses IPv6 soit 667126144800000000 (667 millions de milliards) adresses IPv6 par mm² de surface de la terre !
 - Sécurité pas prise en compte au niveau IP mais au niveau de l'application. Aujourd'hui, on intègre la sécurité au niveau réseau. IPsec, la solution !

PARTIE 8

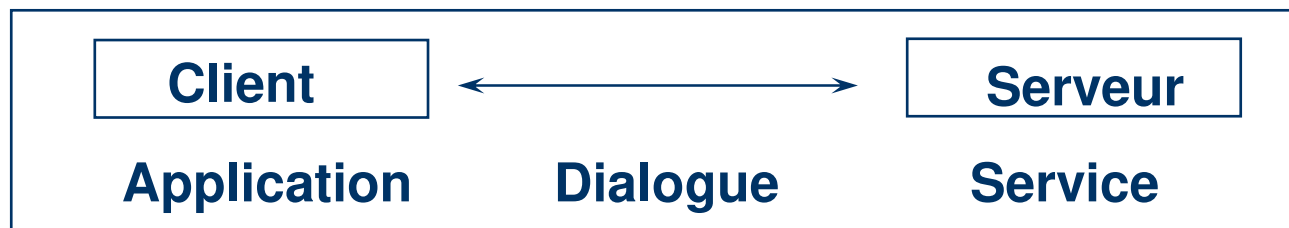
L'ARCHITECTURE CLIENT/SERVEUR

Introduction

- L'architecture client/serveur est l'aboutissement d'un ensemble d'évolutions technologiques survenues dans les années 1980 :
 - Augmentation des capacités mémoire.
 - Performances des processeurs et des réseaux.
 - Evolutions des logiciels : interfaces graphiques, multimédia, des interfaces de communications.

Le modèle

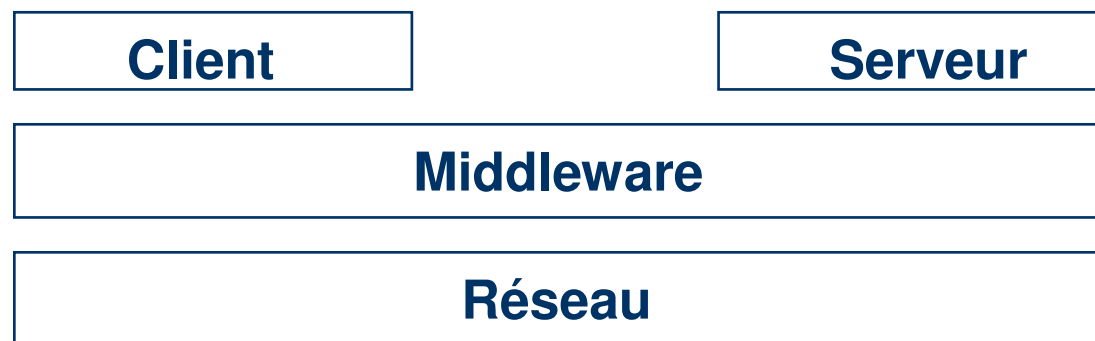
- Repose sur une communication d'égal à égal entre les applications distantes.
- Communication réalisée par dialogue entre processus deux à deux.
- Un processus est le client, l'autre le serveur.
- Les processus ne sont pas identiques mais forment plutôt un système coopératif.
- Le résultat de cette coopération se traduit par un échange de données, le client réceptionne les résultats finaux délivrés par le serveur :
 - Le client initie l'échange.
 - Le serveur est à l'écoute d'une requête cliente éventuelle.



Le service rendu = traitement effectué par le serveur

Le middleware

- L'objectif essentiel du *middleware* ou intergiciel est d'offrir aux applications une interface unifiée permettant l'accès à l'ensemble des services disponibles sur le réseau : l'API (*Application Programming Interface*).
- API du *middleware* = ciment entre les protocoles réseau et les applications.



Conception du client

- Une application cliente est moins complexe que son homologue serveur :
 - La plupart des applications clientes ne gèrent pas d'interactions avec plusieurs serveurs.
 - La plupart des applications clientes sont traitées comme un processus conventionnel. Un serveur peut nécessiter des accès privilégiés au *middleware*.

Conception du serveur

- Entre indéfiniment dans un processus d'attente de requêtes client. Lorsqu'une requête arrive, le serveur déclenche les processus associés à cette requête puis émet la ou les réponses vers le client.
- Problème : gérer plusieurs clients simultanément !
- Les types de serveurs :
 - serveurs itératifs : ne gèrent qu'un seul client à la fois.
 - serveurs parallèles : fonctionnent en mode concurrent.

Les sockets

- Les *sockets* : interface client/serveur (API) utilisée à l'origine dans le monde UNIX et TCP/IP.
- Etendues aujourd'hui du micro (*winsock*) au *mainframe*.
- L'API *sockets* est le standard de fait pour la programmation réseau.
- Dans l'environnement UNIX, les *sockets* sont traitées de la même manière que les fichiers : on a donc les appels système d'ouverture (qui permet d'avoir un descripteur de référence), de lecture, d'écriture, de contrôle et de fermeture.

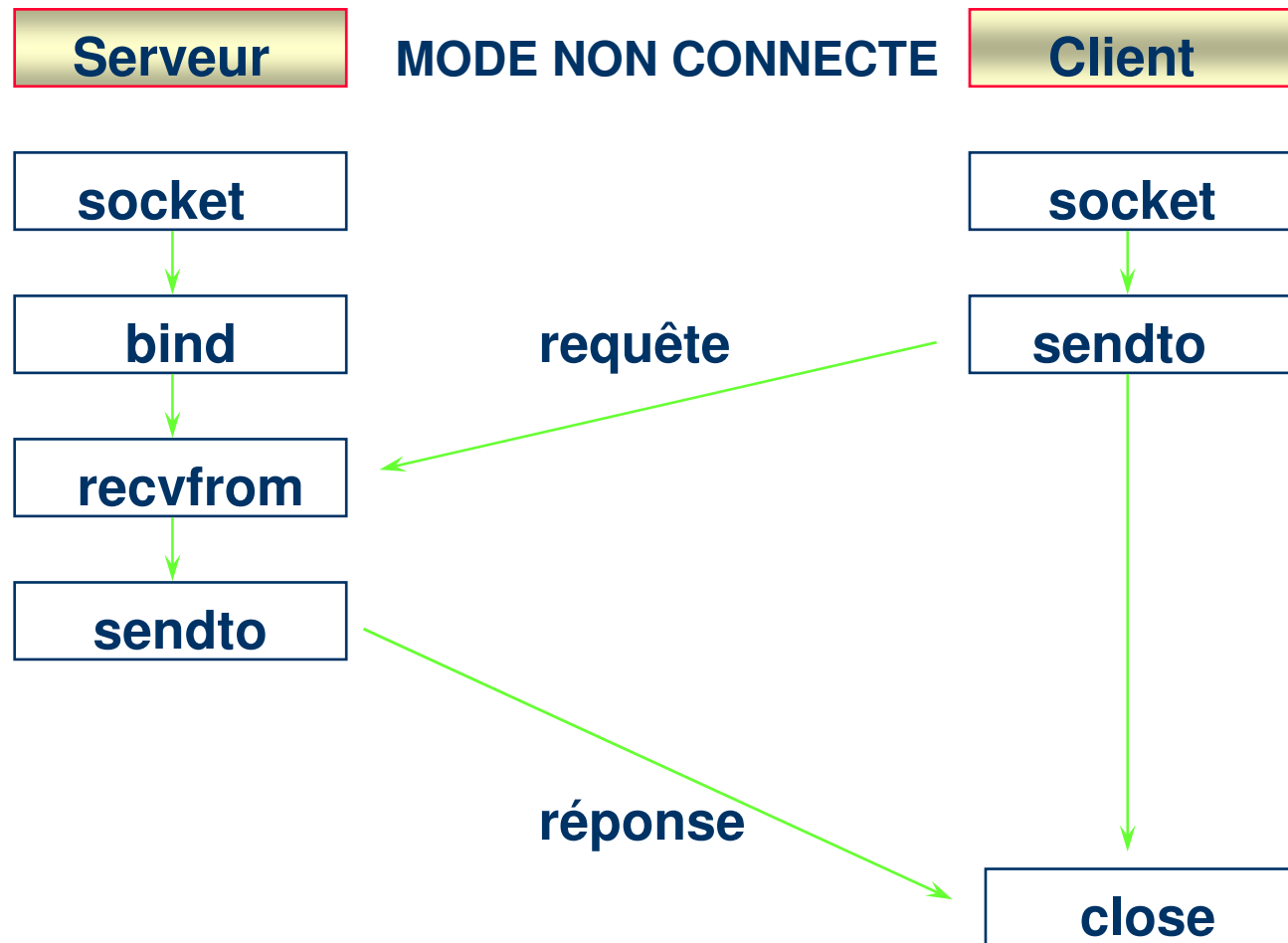
Les sockets

- Il existe d'autres API de programmation réseau :
 - *Streams*.
 - TLI (*Transport Level Interface*).
 - RPC (*Remote Procedure Call*).
 - XDR (*eXternal Data Representation*).
 - Propriétaires.
- Les applications client/serveur ne voient les couches de communication qu'à travers l'API *sockets* (abstraction).

Les sockets

- Elles permettent d'établir un lien de communication en mode connecté ou non connecté sur un réseau.
- Elle structurent une application :
 - Soit en mode client.
 - Soit en mode serveur.
- Elles permettent d'échanger des données entre ces applications.

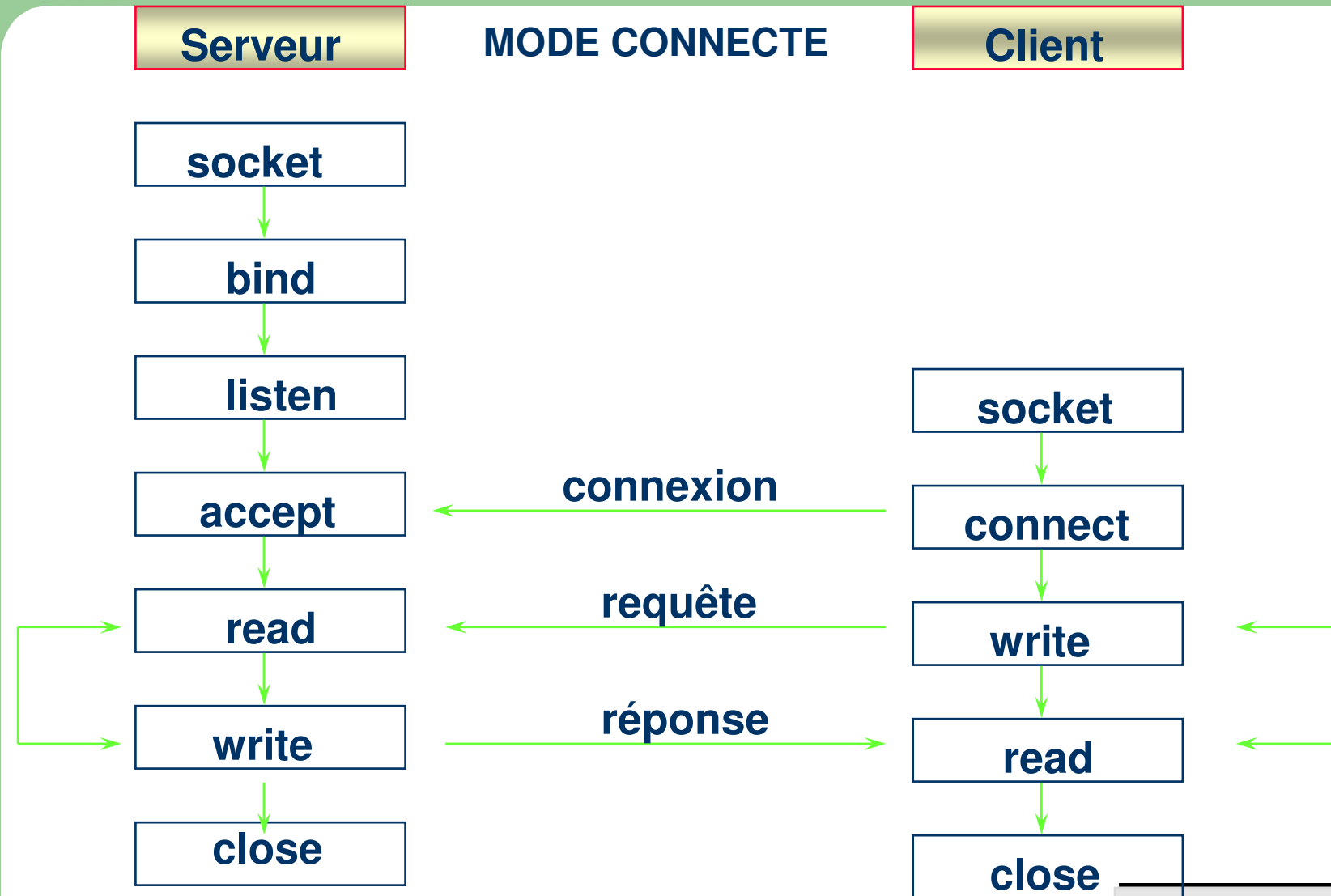
Les sockets : mode non connecté



Les sockets : mode non connecté

- Le client n'établit pas de connexion avec le serveur mais émet un datagramme UDP (*sendto*) vers le serveur.
- Le serveur n'accepte pas de connexion mais attend un datagramme UDP d'un client (*recvfrom*) qui transmet le datagramme à l'application.

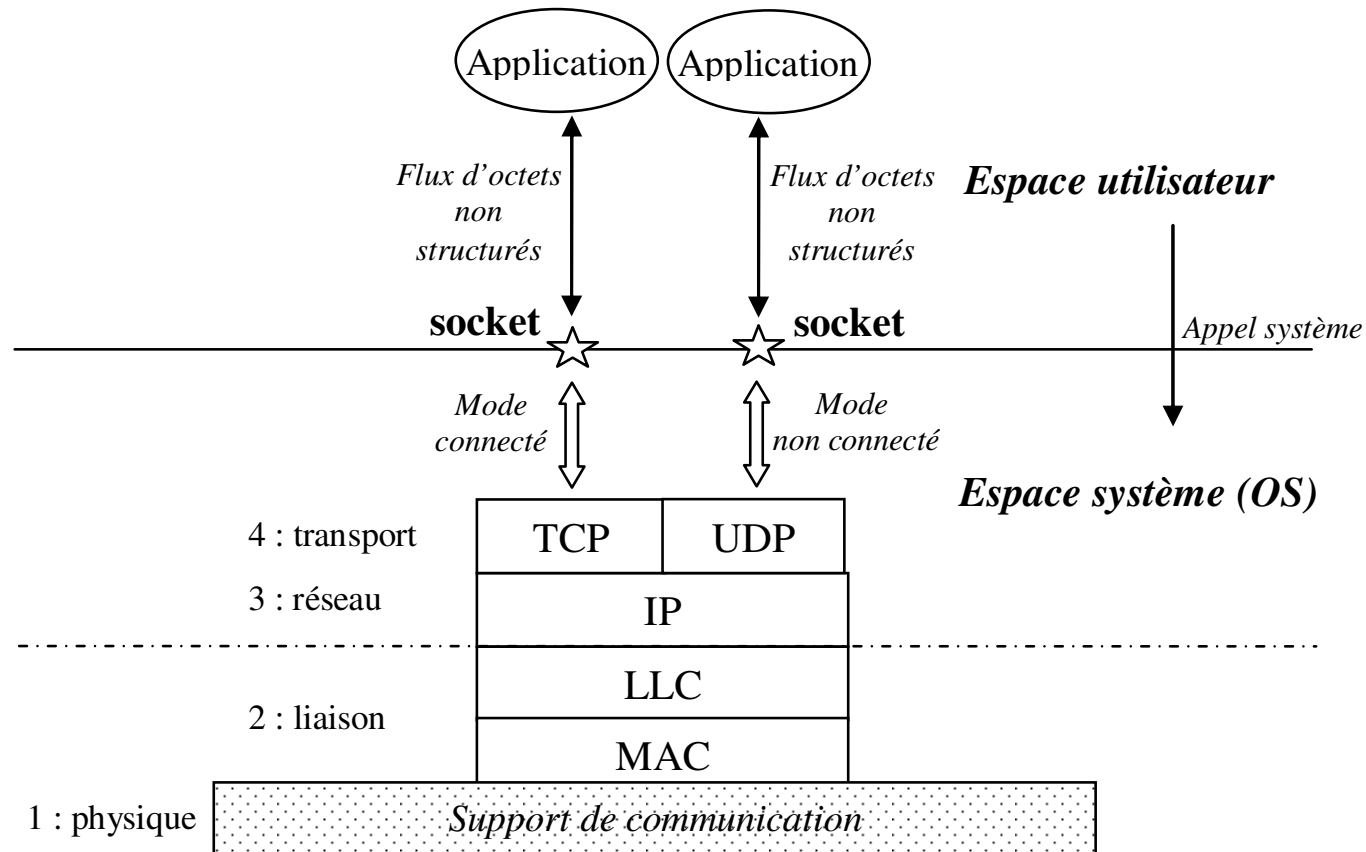
Les sockets : mode connecté



PARTIE 9

L'API SOCKETS

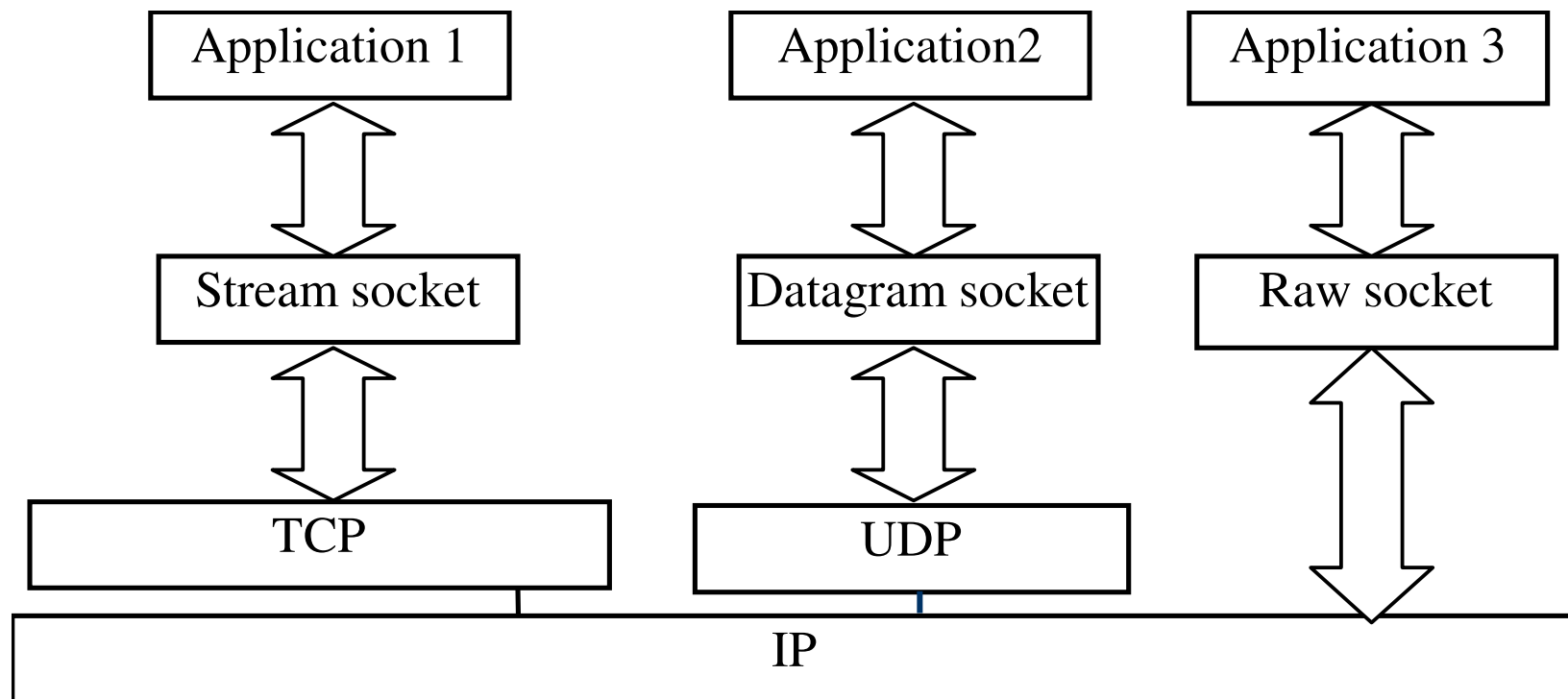
Protocoles Internet : rappel



Attributs des sockets

- Un nom :
 - Descripteur de fichier.
- Un type :
 - `SOCK_STREAM` : mode connecté fiable avec TCP/IP.
 - `SOCK_DGRAM` : mode non connecté non fiable avec UDP/IP.
 - `SOCK_RAW` : mode brut caractère pour accès direct aux couches inférieures. Seul, le superutilisateur peut l'utiliser.
- Associé à un processus.
- Une adresse : adresse IP + n° port.

Types de sockets et protocoles



Mode de dialogue et appels système

- Liste des principaux appels système de l'API *socket* :

Client UDP	Serveur UDP	Client TCP	Serveur TCP
<code>socket ()</code>	<code>socket ()</code>	<code>socket ()</code>	<code>socket ()</code>
	<code>bind ()</code>		<code>bind ()</code>
			<code>listen ()</code>
		<code>connect ()</code>	<code>accept ()</code>
<code>recvfrom ()</code> <code>sendto ()</code>	<code>recvfrom ()</code> <code>sendto ()</code>	<code>read ()</code> <code>write ()</code>	<code>read ()</code> <code>write ()</code>
<code>close ()</code>	<code>close ()</code>	<code>close ()</code>	<code>close ()</code>

Mode de dialogue et appels système

- Un processus (le serveur) est en attente (à l'écoute) sur un numéro de port :
=> Tous les processus (les clients) connaissant l'adresse IP du serveur et le numéro de port peuvent lui envoyer des messages (requêtes).
- Le serveur traite les messages entrants puis renvoie une réponse au client en utilisant le numéro de port source contenu dans l'en-tête du message.
- En mode connecté, il y a établissement préalable d'une connexion (service fiable).

*NIX et les descripteurs de fichier

- Sous *NIX (Unix, Linux...), tout est fichier.
- Tout est représenté par un fichier dans le système de fichiers :
 - Fichier ASCII : fichier normal.
 - Fichier binaire : fichier normal.
 - Répertoire : fichier spécial.
 - Périphérique matériel sous `/dev` : fichier spécial.
 - Tube nommé (*fifo*) : fichier spécial.

*NIX et les descripteurs de fichier

- Tout revient alors à manipuler les 5 appels système universels bas niveau :
 - Ouverture : `open()`.
 - Fermeture : `close()`.
 - Lecture : `read()`.
 - Ecriture : `write()`.
 - Configuration : `ioctl()`.
- L'ouverture d'un fichier renvoie un descripteur de fichier qui identifie le flux d'octets non structurés (*stream*) échangé.

*NIX et les descripteurs de fichier

- Un descripteur de fichier est unique pour chaque processus qui s'exécute. Il identifie de façon dynamique un flux échangé.
- Il est du type : `int fd;`
- Il existe aussi le descripteur de fichier pour des entrées/sorties formatées du type : `FILE *fd;`
- Un descripteur de fichier est une valeur positive ou nulle quand tout va bien.

*NIX et les descripteurs de fichier

- Quand un processus est créé, il possède par défaut 3 descripteurs de fichier :
 - 0 : entrée standard = clavier.
 - 1 : sortie standard = écran.
 - 2 : erreur standard = écran par défaut.
- L'appel système `open ()` renvoie un descripteur de fichier qui sera utilisé comme premier argument des 4 autres appels système.
- Il existe d'autres appels système qui créent des descripteurs de fichier...

*NIX et les descripteurs de fichier

- La « règle d'or » s'applique lors de la création d'un descripteur de fichier : l'appel système (`open()`) renvoie la plus petite valeur positive ou nulle libre.

- Exemple 1 :

```
int fd;
. . .
if((fd = open("toto.txt", O_RDWR)) == -1) {
    printf("Erreur d'ouverture\n");
    exit(-1);
}
. . .
```

fd vaut 3.

stdin	0
stdout	1
stderr	2
	3

fd

*NIX et les descripteurs de fichier

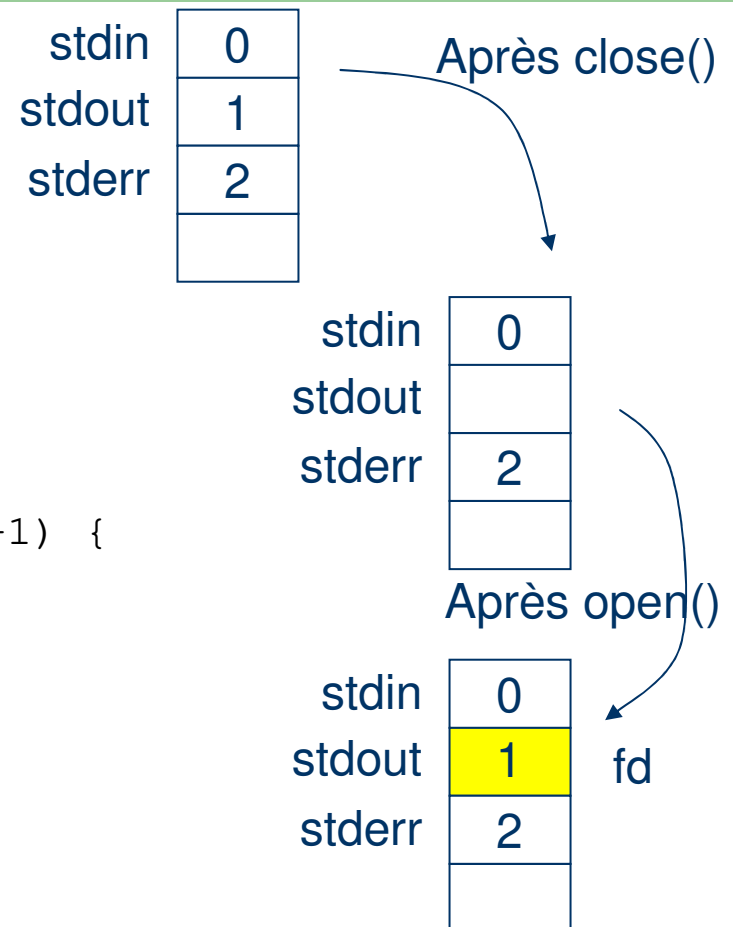
- Exemple 2 :

```
int fd;

...
close(1);

if((fd = open("toto.txt", O_RDWR)) == -1) {
    printf("Erreur d'ouverture\n");
    exit(-1);
}

...
printf("coucou");
```



fd vaut 1. Que se passe-t-il lorsque que j'exécute un printf () ?

Structures de données

```
#include <sys/types.h>           // Bibliothèques requises
#include <sys/socket.h>

struct sockaddr {
    unsigned short sa_family;     // Famille de protocole pour cette adresse
    char sa_data[14];            // 14 octets d'adresse
}

struct sockaddr_in {             // _in pour Internet
    short sin_family;           // Famille de protocole pour cette adresse
    u_short sin_port;           // Numéro de port
    struct in_addr sin_addr;     // Adresse IP
    char sin_zero[8];           // Non utilisé
}

struct in_addr {
    u_long s_addr;              // Soit 4 octets
};
```

Création et fermeture d'une socket

- **int socket(int af, int type, int protocole)**
 - Création de la structure de données permettant la communication.
 - af = famille de protocole (Internet, IPC, ...).
 - AF_INET (ou PF_INET) : domaine Internet.
 - AF_UNIX : domaine UNIX. Communications locales IPC.
 - type = SOCK_STREAM, SOCK_DGRAM, SOCK_RAW.
 - protocole : 0 pour protocole par défaut.
 - socket () retourne :
 - Un descripteur de *socket* suivant le respect de la « règle d'or ».
 - -1 si erreur (voir errno).
- **close(int socket)**
 - Ferme la connexion (SOCK_STREAM) et supprime la structure de données associée au descripteur de *socket*.

Fonctions utilitaires

- **struct hostent * gethostbyname(char *name)**

- Traduit un nom de domaine en adresse IP sur 32 bits.

```
struct hostent *h;  
h=gethostbyname("www.enseirb-matmeca.fr");  
printf("adresse IP: %s\n",  
       inet_ntoa(*(struct in_addr *)h->h_addr));
```

- **getservbyname(...)**

- Traduit en numéro de port le nom d'un service.

- **char * inet_ntoa(int in_addr)**

- Convertit une adresse IP 32 bits sous format ASCII.

```
char *adl_ascii;  
adl_ascii=inet_ntoa(sa.sin_addr),  
printf("adresse: %s\n",adl_ascii);
```

Fonctions utilitaires

- Conversion *Network Byte Order* vs *Host Byte Order* pour régler le problème de représentation des types simples *big endian* ou *little endian* :
 - **htons ()** : *Host to Network Short*.
 - **htonl ()** : *Host to Network Long*.
 - **ntohs ()** : *Network to Host to Short*.
 - **ntohl ()** : *Network to Host to Long*.

Client et serveur UDP

Client UDP	Serveur UDP
socket ()	socket ()
	bind ()
recvfrom () sendto ()	sendto () recvfrom ()
close ()	close ()

Lecture/écriture en mode UDP

- **int sendto(...)**
 - Côté client et serveur UDP.
 - Permet l'envoi d'un datagramme UDP.
 - Contient les coordonnées (structure `sockaddr`) du destinataire.
 - Renvoie le nombre d'octets réellement émis.
- **int recvfrom(...)**
 - Côté client et serveur UDP.
 - Réception bloquante d'un datagramme UDP.
 - Contient les coordonnées (structure `sockaddr`) de l'expéditeur.
 - Renvoie le nombre d'octets réellement reçus.

Lecture/écriture en mode UDP

```
int sendto(  
    int sd,                // Descripteur de socket  
    void *buffer,         // Buffer à envoyer  
    int longueur,        // Taille du buffer  
    int option,          // 0  
    struct sockaddr *coord, // Coordonnées destinataire  
    int longueur_coord   // Longueur structure coordonnées  
);  
  
int recvfrom(  
    int sd,                // Descripteur de socket  
    void *buffer,         // Buffer de réception  
    int longueur,        // Taille zone réservée  
    int option,          // 0  
    struct sockaddr *coord, // Coordonnées expéditeur  
    int *longueur_coord   // Longueur structure coordonnées  
);
```

- Les coordonnées du destinataire ou de l'expéditeur sont en fait une structure de type `sockaddr`.

Déclaration comme serveur UDP

- `int bind(int socket, struct sockaddr * coord_serveur, int longueur_coord)`
 - Côté serveur UDP.
 - Pour un serveur, associe un numéro de port et l'adresse locale à la socket, retourne -1 si erreur.
 - socket = descripteur de *socket*.
 - adresse-locale = structure qui contient adresse IP + n° de port.
 - adresse-locale : `struct sockaddr *`
 - `sockaddr_in` si `AF_INET` (adresse IP).
 - longueur-adresse : `sizeof(struct sock_addr)`.
 - Si `sin_addr.s_addr = INADDR_ANY` : utilisation de l'adresse IP de la machine qui héberge le serveur.

Client et serveur TCP

Client TCP	Serveur TCP
socket ()	socket ()
	bind ()
	listen ()
connect ()	accept ()
read () write ()	read () write ()
close ()	close ()

Connexion en mode TCP côté client

- `connect (int socket, struct sockaddr * coord_serveur, int longueur_coord)`
 - Côté client TCP.
 - Pour établir une connexion TCP avec le processus serveur.
 - Les coordonnées du serveur sont spécifiées.
 - Appel bloquant jusqu'à ce que la connexion TCP soit établie avec le serveur.

Lecture/écriture en mode TCP

- **write(int newsock, void * buffer, int longueur)**
 - Côté client et serveur TCP.
 - Envoie des données sur la connexion TCP établie.
 - Plus besoin des adresses émetteur/destinataire.
 - Renvoie le nombre d'octets réellement émis.
- **read(int newsock, void * buffer, int longueur)**
 - Côté client et serveur TCP.
 - Reçoit des données sur la connexion TCP établie.
 - Plus besoin des adresses émetteur/destinataire.
 - Renvoie le nombre d'octets réellement reçus.
- L'usage de ces 2 appels système est le même que lorsqu'ils sont utilisés sur fichier. L'appel `read()` est bloquant (comme le `write()` si saturation de la pile réseau).

Déclaration comme serveur TCP

- `int bind(int socket, struct sockaddr * coord_serveur, int longueur_coord)`
 - Côté serveur TCP.
 - Pour un serveur, associe un numéro de port et l'adresse locale à la socket, retourne -1 si erreur.
 - `socket` = descripteur de *socket*.
 - `adresse-locale` = structure qui contient adresse IP + n° de port.
 - `adresse-locale` : `struct sockaddr *` :
 - `sockaddr_in` si `AF_INET` (adresse IP).
 - `longueur-adresse` : `sizeof(struct sock_addr)`.
 - Si `sin_addr.s_addr= INADDR_ANY` : utilisation de l'adresse IP de la machine qui héberge le serveur.

Création d'une file d'attente

- `listen(int socket, int taille_file)`
 - Côté serveur TCP.
 - Création d'une file d'attente pour les demandes de connexion.
 - `lgr-file` indique le nombre maximum de demandes de connexion autorisées dans la file (5, 10 ou 20).
 - File d'attente exploitée par l'appel `accept ()`.

Acceptation d'une connexion entrante TCP

- `newsock = accept (int socket, struct sockaddr * coord_client, int * longueur_coord)`
 - Côté serveur TCP.
 - Prise en compte d'une demande de connexion entrante sur une *socket*.
 - Appel bloquant.
 - `newsock` : **nouveau** descripteur de *socket* de la connexion établie sur laquelle s'effectuera l'échange de données.
 - `adresse` : coordonnées du client TCP.
 - Le processus peut traiter lui-même la nouvelle connexion, puis revenir à `accept ()` ou bien se répliquer (`fork ()`) pour la traiter, le processus père se remettant alors à l'écoute d'une nouvelle connexion entrante.

Exemple 1 : myftp0.c

```
#include <stdio.h>
#include <stdlib.h>
#include <strings.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>

#define FTPPORT 21                                /* Numero du port du serveur ftp */

int main(argc,argv)
int argc ;
char *argv[] ;

{
int sd;
struct sockaddr_in sa;                          /* Structure Internet sockaddr_in */
struct hostent *hptr ;                          /* Infos sur le serveur */
```

Exemple 1 : myftp0.c

```
char *serveur ;           /* Nom du serveur distant */
char msg[] = "USER TOTO\n"; /* Buffer */
char buf[256];           /* Buffer */
int n;

/* verification du nombre d'arguments de la ligne de commande */
if (argc != 2) {
    printf("myftp. Erreur d'arguments\n");
    printf("Syntaxe : %% myftp nom_serveur_ftp\n");
    exit(1);
}

/* Recuperation nom du serveur */
serveur = argv[1];
```

Exemple 1 : myftp0.c

```
/* Recuperation des infos sur le serveur dans /etc/hosts ou par DNS */
if((hptr = gethostbyname(serveur)) == NULL) {
    printf("Probleme de recuperation d'infos sur le serveur\n");
    exit(1);
}

/* Initialisation la structure sockaddr sa avec les infos formattees : */
/* bcopy(void *source, void *destination, size_t taille);          */
bcopy((char *)hptr->h_addr, (char*)&sa.sin_addr, hptr->h_length);

/* Famille d'adresse : AF_INET = PF_INET */
sa.sin_family = AF_INET;

/* Initialisation du numero du port */
sa.sin_port = htons(FTPPORT);
```

Exemple 1 : myftp0.c

```
/* Creation de la socket TCP */
if((sd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
    printf("Probleme lors de la creation de socket\n");
    exit(1);
}

/* Etablissement de la connexion avec le serveur ftp */
if((connect(sd, (struct sockaddr *)&sa, sizeof(sa))) < 0 ) {
    printf("Probleme de connexion avec le serveur\n");
    exit(1);
}

/* Lecture de la banniere d'accueil du serveur ftp */
n = read(sd, buf, sizeof(buf));

/* Ecriture sur stdout */
write(1, buf, n);

/* Envoi de la commande ftp vers serveur ftp */
write(sd, msg, sizeof(msg));
```


Exemple 1 : myftp0.c

```
/* Lecture de la reponse du serveur ftp */  
n = read(sd, buf, sizeof(buf));  
write(1, buf, n);  
  
/* Fermeture de la socket */  
close(sd);  
  
printf("Fin de connexion. Bye...\n");  
exit(0);  
}
```

Exemple 2 : pingserveurTCP0.c

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>

main(argc, argv)
int argc ;
char *argv[] ;

{
int sd;
int port; /* Numero de port du serveur */
struct sockaddr_in sa; /* Structure Internet sockaddr_in */

int newsd; /* Id de la socket entrante */
struct sockaddr_in newsa; /* sockaddr_in de la connection entrante */
int newsalength;
```

Exemple 2 : pingserveurTCP0.c

```
int i;

/* verification du nombre d'arguments de la ligne de commande */
if (argc != 2) {
    printf("pingserveurTCP. Erreur d'arguments\n");
    printf("Syntaxe : %% pingserveurTCP numero_port\n");
    exit(1);
}

/* Recuperation numero port passe en argument */
port = atoi(argv[1]);
```

Exemple 2 : pingserveurTCP0.c

```
/* Initialisation la structure sockaddr sa */

/* Famille d'adresse : AF_INET = PF_INET */
sa.sin_family = AF_INET;

/* Initialisation du numero du port */
sa.sin_port = htons(port);
sa.sin_addr.s_addr = INADDR_ANY;

/* Creation de la socket TCP */
if((sd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
    printf("Probleme lors de la creation de socket\n");
    exit(1);
}

/* Bind sur la socket */
if(bind(sd, (struct sockaddr *)&sa, sizeof(sa)) == -1) {
    printf("Probleme avec le bind\n");
    exit(1);
}
```

Exemple 2 : pingserveurTCP0.c

```
/* Initialisation de la queue d'ecoute des requetes (5 max) */
listen(sd, 5);

printf("Serveur pingTCP en ecoute...\n");

i = 0;
while(1) {

/* newlength contient la taille de la structure sa attendue */
newlength = sizeof(newsa) ;

if((newsd = accept(sd, (struct sockaddr *)&newsa, &newlength)) < 0 ) {
    printf("Erreur sur accept\n");
    exit(1);
}

/* Compteur nombre connexions */
i++;
```

Exemple 2 : pingserveurTCP0.c

```
/* nom du client */
    printf("Connection No %d sur le port %d...\n", i, ntohs(newsa.sin_port));

/* Tempo pour pouvoir faire un netstat... */
    sleep(10);

    close(newsd);
}

/* Fermeture du serveur. Never reached */
close(sd);
printf("Fin du serveur. Bye...\n");
exit(0);
}
```

Exemple complet. Client TCP



- Service : vérification si l'utilisateur *quidam* est bien identifié sur le serveur (il l'est s'il possède une entrée dans le fichier `/etc/passwd` avec son nom de *login*).
- Le service KEY est défini dans le fichier `/etc/services` par l'entrée :
key 5002/tcp
- `% ./client nom_serveur quidam`

Source de l'exemple : Internetworking with TCP/IP (Douglas Comer)

Exemple complet. Client TCP

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#define BUFSIZE 256

main (argc,argv)
int argc;
char **argv;
{
int s;                /* Descripteur de socket */
int len;

struct sockaddr_in sa; /* Structure adresse Internet */
struct hostent *hp;    /* Structure service de nom */
struct servent *sp;    /* Structure service Internet */
char *myname;          /* Pointeur sur le nom du programme */
char buf[BUFSIZE];
char *host,*user;      /* Pointeurs sur le serveur et l'utilisateur */
```


Exemple complet. Client TCP

```
myname = argv[0];

if (argc != 3) {
    printf("Usage : %s serveur user\n", myname);
    exit(1);
}

user = argv[2];
host = argv[1];

/* Voir si le serveur existe, structure hp remplie */
if ((hp = gethostbyname(host)) == NULL) {
    printf("%s : %s serveur inconnu\n", myname, host);
    exit(1);
}

/* Copie de l'adresse du serveur et du type d'adresse dans sa */
bcopy ((char *)hp->h_addr, (char *)&sa.sin_addr, hp->h_length);

sa.sin_family = AF_INET;
```

Exemple complet. Client TCP

```
/* voir si le service est autorisé sur le client, rempli la structure sp */
if ((sp = getservbyname("key", "tcp")) == NULL) {
    printf("%s : Pas de service QUI sur ce système \n", myname);
    exit(1);
}
sa.sin_port = htons(sp->s_port);

/* Création de la socket client */
if ((s = socket (AF_INET, SOCK_STREAM, 0)) < 0 ) {
    perror("socket");
    exit(1);
}
```

Exemple complet. Client TCP

```
/* Connexion au serveur, infos dans la structure adresse Internet sa */
if (connect(s, &sa, sizeof(sa)) < 0 ) {
    perror("connect");
    exit(1);
}

/* Envoi de la requête */
write(s, user, strlen(user)+1);

/* Lecture de la réponse */
read(s, buf, BUFSIZE);

/* Affichage de la réponse */
printf("Réponse : %s\n", buf);
close (s);

/* Fermeture de la connexion */
exit(0);
}
```

Exemple complet. Serveur TCP

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <pwd.h>
#define BUFSIZE 256

/* Taille de la file d'attente initialisée par listen */
#define BACKLOG 1
#define MAXHOSTNAME 64

/* Nombre de requêtes traitées par le programme */
#define MAXREQ 3

main (argc,argv)
int argc;
char **argv;
void quiest();
int s, t;                                /* Descripteur de socket */
```

Exemple complet. Serveur TCP

```
int i, count_req;
struct sockaddr_in sa;    /* Structure adresse Internet */
struct sockaddr_in isa;
struct hostent *hp;      /* Structure service de nom */
struct servent *sp;      /* Structure service Internet */
char localhost[MAXHOSTNAME];

/* Voir si service KEY existe */
if ((sp = getservbyname("key", "tcp")) == NULL) {
    printf("Pas de service QUI sur ce système \n");
    exit(1);
}

/* Récupération du numéro de port serveur dans la structure sa */
sa.sin_port = htons(sp->s_port);

/* Récupération nom machine hôte, ici localhost */
gethostname (localhost, MAXHOSTNAME);
```

Exemple complet. Serveur TCP

```
/* Remplit la structure adresse sa */
hp = gethostbyname(localhost);
bcopy(hp->h_addr, (char*)&sa.sin_addr, hp->h_length);
sa.sin_family = AF_INET;

/* Allocation d'un descripteur de socket */
if ((s = socket(AF_INET, SOCK_STREAM, 0)) < 0 ) {
    perror("Serveur : problème création socket");
    exit(1);
}

/* Lien socket avec adresse IP et port */
if (bind(s, &sa, sizeof(sa)) < 0 ) {
    printf("Serveur : problème bind\n");
    exit(1);
}

listen(s, BACKLOG);
```

Exemple complet. Serveur TCP

```
for (count_req = 0; count_req <=MAXREQ ; count_req++) {
/* Attend une requête sur la primitive accept */
    i = sizeof(isa);
    t = accept (s, &isa, &i );
    printf ("Requête %d\n", count_req);

/* Traitement de la requête */
    quiest(t);
    close(t);
}

/* Fermeture de la socket réservée au traitement de cette requête */
close(s);
printf("fin du service pour %s\n",localhost);
exit(0);
}
```

Exemple complet. Serveur TCP

```
void quiest(sock) /* Traitement de la requête */
int sock;
{
    struct passwd *p;
    char buf[BUFSIZE];
    int i;

    /* Réception de la demande */
    if (( i = read(sock, buf, BUFSIZE)) <= 0)
        return;

    /* Recherche dans le fichier /etc/passwd */
    if (( p = getpwnam (buf)) == NULL)
        strcpy (buf, "Utilisateur inconnu sur le serveur\n");
    /* Envoi de la réponse */
    else
        sprintf (buf, "%s : %s \n", p->pw_name, p->pw_gecos);
        write(sock, buf, strlen (buf));
}
```


PARTIE 10

CONCLUSION

Conclusion

- Introduction aux réseaux.
- Présentation du modèle OSI.
- Présentation de la technologie Ethernet.
- Présentation d'Internet et de ses principaux protocoles, d'Internet embarqué et de l'Internet des objets.
- Présentation de l'API *sockets* et d'exemples complets de clients et de serveurs.

SYNTHESE DU COURS

Synthèse modèle OSI

- Le modèle OSI est la conceptualisation de ce que doit être un réseau de transmission de données.
- Le modèle OSI comprend les couches, les protocoles et les interfaces.
- Les couches 1 à 3 sont les couches basses orientées transmission des données.
- Les couches 4 à 7 sont les couches hautes orientées traitement des données.
- Il y a 7 couches : 7 Application, 6 Présentation, 5 Session, 4 Transport, 3 Réseau (niveau paquet), 2 Liaison (niveau trame), 1 Physique (niveau bit).
- On encapsule les données à l'émission, on désencapsule les données en réception.

Synthèse IP

- IP correspond au niveau 3 réseau du modèle OSI.
- IP est au dessus de tout. C'est un protocole de convergence.
- IP est un protocole sans connexion et non fiable vis-à-vis des données transportées.
- IP est implanté dans tous les équipements Internet : routeur et machines terminales.
- IP assure l'adressage, le routage et la fragmentation des données.
- On parle de datagramme IP.
- Une adresse IP est binaire et fait 32 bits. On utilise le protocole DNS pour passer d'une adresse IP symbolique à une adresse IP binaire.

Synthèse IP

- On utilise aussi la notation décimale pointée : 127.0.0.1 correspond au *loopback* i.e. soi même.
- Pour une gestion locale d'une plage d'adresses IP contigües, on introduit la notion de sous-réseau IP.
- La fragmentation est liée au MTU de la trame qui transporte le datagramme IP.
- L'entête d'un datagramme IP fait 20 octets.

Synthèse ARP et RARP

- ARP permet de trouver l'adresse matérielle (adresse physique) d'une machine en fonction de son adresse IP.
- RARP permet de récupérer l'adresse IP en fonction de l'adresse matérielle d'une machine.
- Une opération (R)ARP est contenue dans le champ données de la trame utilisée comme support de transmission (par exemple trame Ethernet).

Synthèse ICMP

- ICMP est un protocole de gestion du réseau IP.
- ICMP est implanté dans tous les équipements, routeurs inclus.
- Un message ICMP est envoyé soit par un équipement destinataire ou soit par un routeur quand il y a un problème dans un datagramme IP ou quand il y a un problème de réseau.
- Un message ICMP ne peut pas engendrer un autre message ICMP. Il ne demande aucune réponse.
- Un message ICMP est contenu dans le champ données d'un datagramme IP.

Synthèse UDP

- UDP correspond au niveau 4 transport du modèle OSI.
- UDP est un protocole de transport en mode non connecté et non fiable.
- On parle de datagramme UDP.
- UDP n'est pas utilisé dans un routeur IP mais dans les machines distantes.
- UDP introduit la notion de port (entier de 16 bits) qui identifie le processus d'une machine qui va traiter les données émises ou reçues.
- Une *socket* correspond au couple de valeurs (@ IP, numéro de port).

Synthèse UDP

- La combinaison de 2 *sockets* identifie complètement l'échange de données UDP entre 2 applications distantes.
- L'entête UDP fait 8 octets (UDP/IP = 28 octets !)..
- Un datagramme UDP est contenu dans le champ données d'un datagramme IP.

Synthèse TCP

- TCP correspond au niveau 4 transport du modèle OSI.
- TCP est un protocole de transport en mode connecté et fiable.
- On parle de segment TCP.
- Le checksum contenu dans l'entête TCP protège l'entête mais aussi les données du segment TCP.
- TCP n'est pas utilisé dans un routeur IP mais dans les machines distantes.
- On a avec TCP une phase de connexion, une phase de transfert de données et une phase de libération.
- On numérote les octets émis : numéro de séquence à l'émission.
- On numérote les octets bien reçus : numéro d'acquittement à la réception (en fait le numéro du prochain octet attendu).

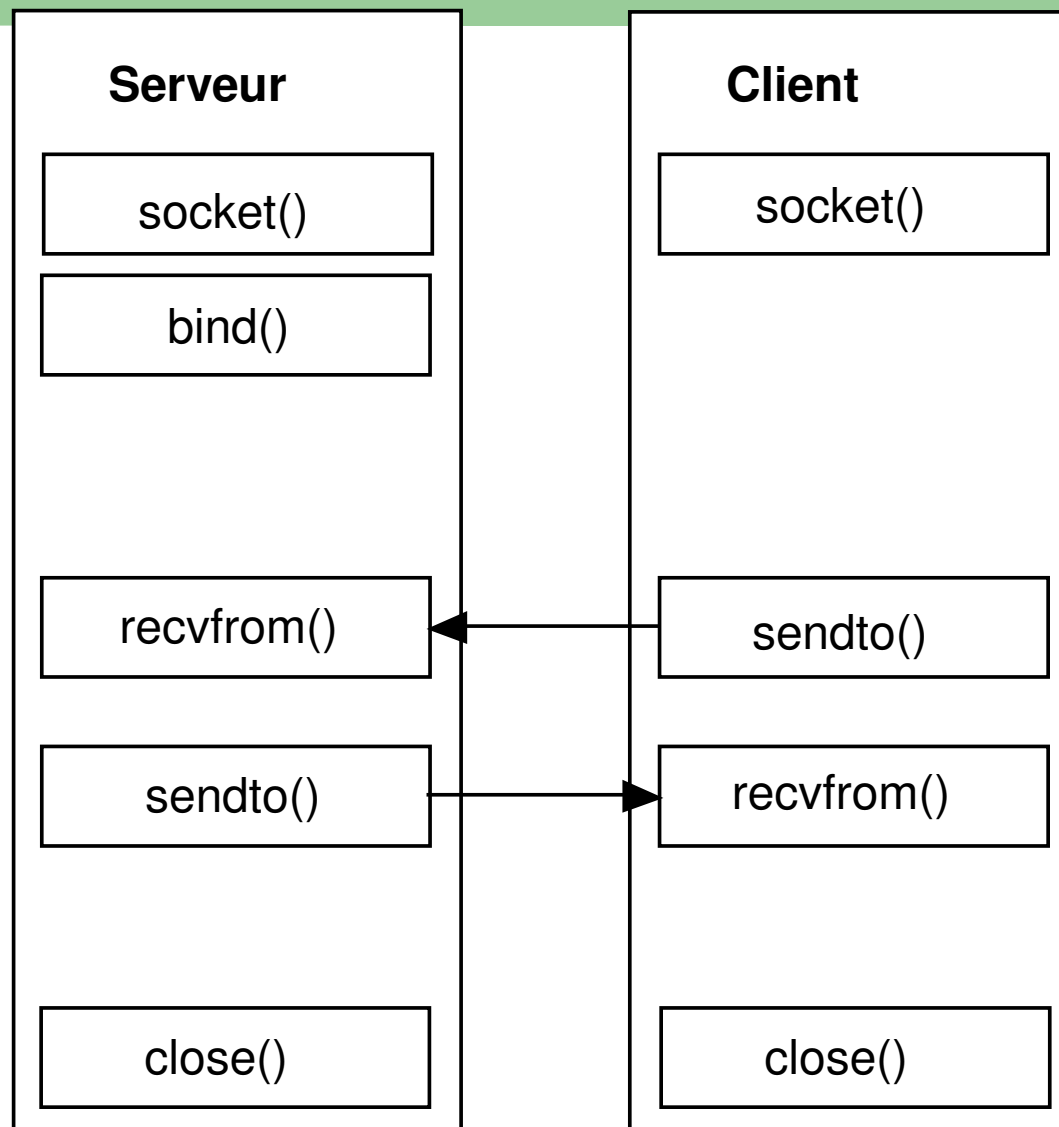
Synthèse TCP

- TCP introduit un contrôle de flux par fenêtre glissante.
- TCP introduit la notion de port (entier de 16 bits) qui identifie le processus d'une machine qui va traiter les données émises ou reçues.
- L'entête TCP fait 20 octets (TCP/IP = 40 octets !).
- Un segment TCP est contenu dans le champ données d'un datagramme IP.

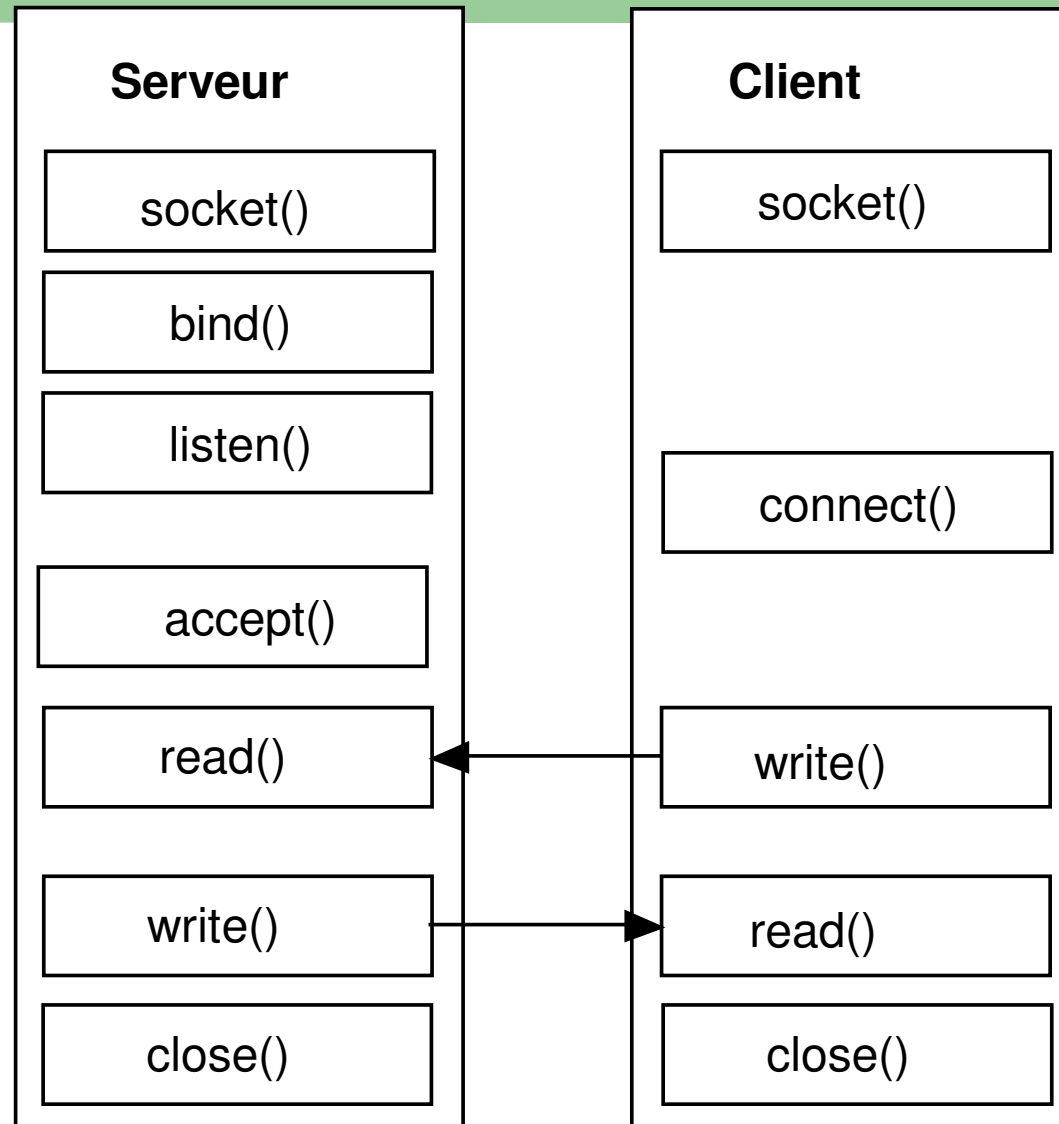
Synthèse communication

- Pour communiquer avec une machine, on a donc besoin :
 - @ physique source, @ physique destination.
 - @ IP source, @ **IP destination**.
 - Port source, **port destination**.
- L'@ physique destination est récupérée automatiquement par la machine source avec le protocole ARP.
- Seules les 2 valeurs destination (@ IP destination, port destination) seront à renseigner quand on utilisera l'API *sockets* de programmation réseau...

Synthèse client et serveur UDP



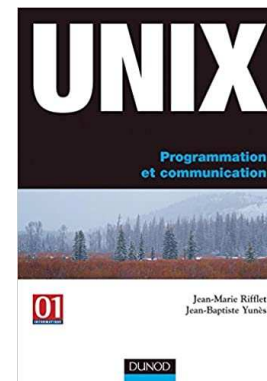
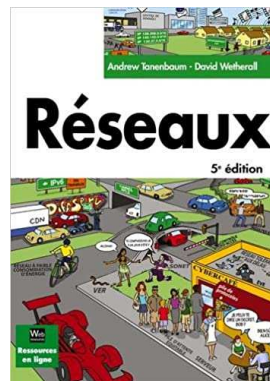
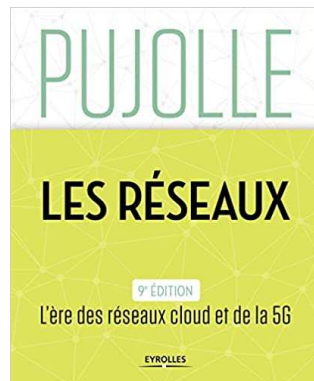
Synthèse client et serveur TCP



BIBLIOGRAPHIE

Bibliographie

- Ce cours est bâti autour des documents de formation réseaux du CNRS (www.urec.fr) de Jean Paul Gautier et Bernard Tuy.
- Les réseaux. G. Pujolle. Editions Eyrolles.
- Réseaux. A. Tanenbaum. Editions Pearson.
- UNIX. Programmation et communication. J.M. Rifflet. Editions Mc Graw Hill.



RE223 : Introduction aux réseaux et à Internet...

Bibliographie

- L'informatique répartie sous UNIX. M. Gabassi et B. Dupouy. Editions Eyrolles.
- Programmation système en C sous Linux : Signaux, processus, threads, IPC et sockets. C. Blaess. Editions Eyrolles

