

APPLICATIONS CLIENT-SERVEUR, IOT

Réseaux TCP-IP, sockets
Internet des Objets

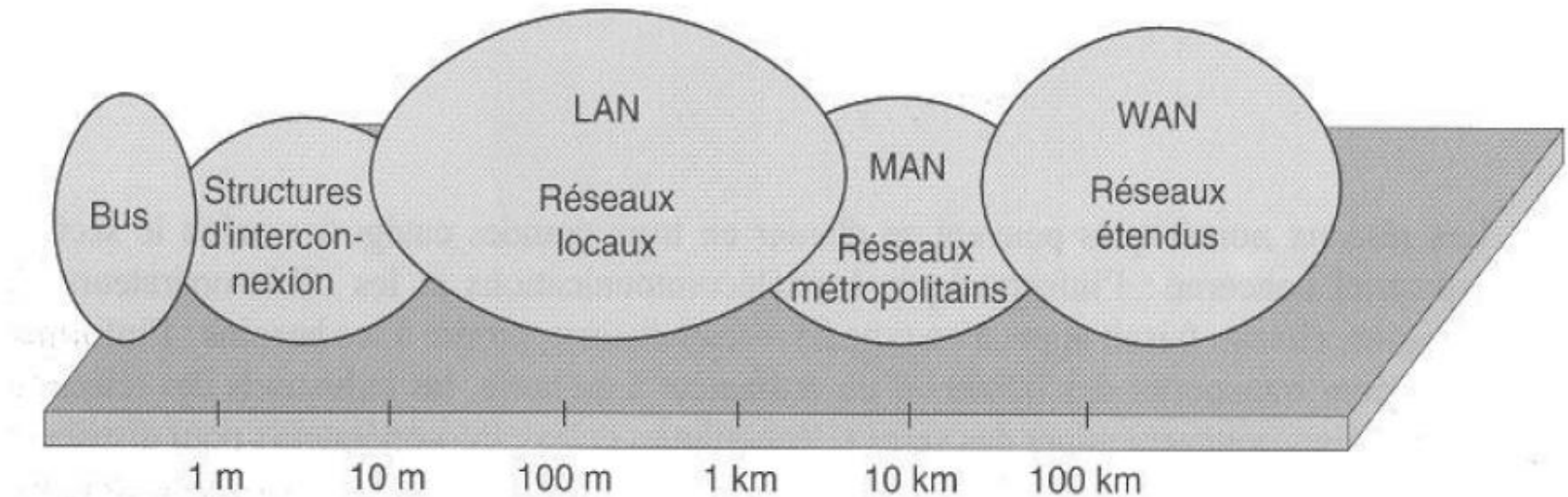
pfister@connecthive.com

Objectifs

- Compréhension des architectures client-serveur
- Développement d'applications réseau
- Technologie: TCP-IP (sockets)
- Langages: java, (python, c, php, vba, dotnet, javascript)
- Plateformes:
 - PC (windows, linux); Mac
 - Android
 - Arduino
 - Arm
 - Esp32

Rappels: qu'est ce qu'un réseau ?

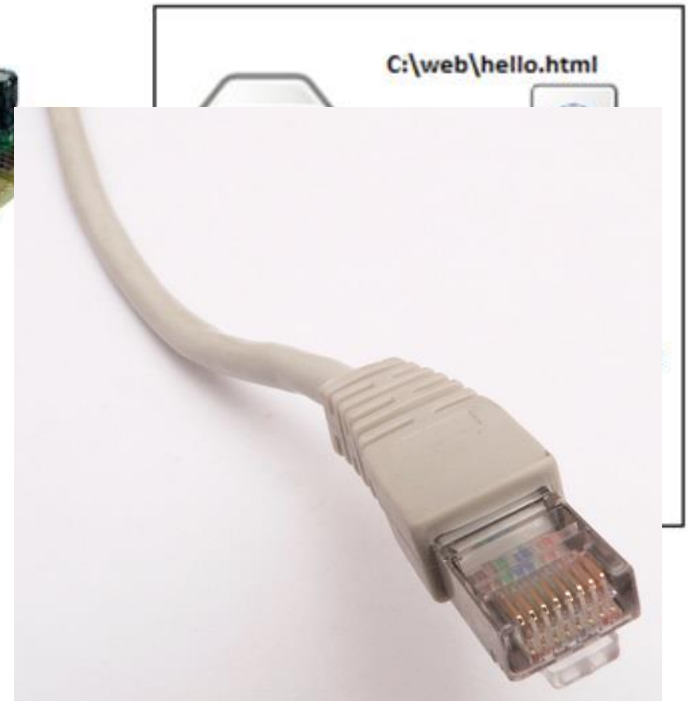
- C'est un ensemble d'ordinateurs (y compris les périphériques qui y sont connectés) reliés ensemble par des **canaux électroniques de communication**, qui leur permettent d'échanger des informations entre eux.
- Un **réseau** est caractérisé par sa **taille**, sa **topologie** et son **accès**.
On rencontre les structures suivantes: des **LAN** (réseaux d'entreprise), des **MAN** (réseaux métropolitains), des **PAN** (bluetooth)



Approche par l'exemple

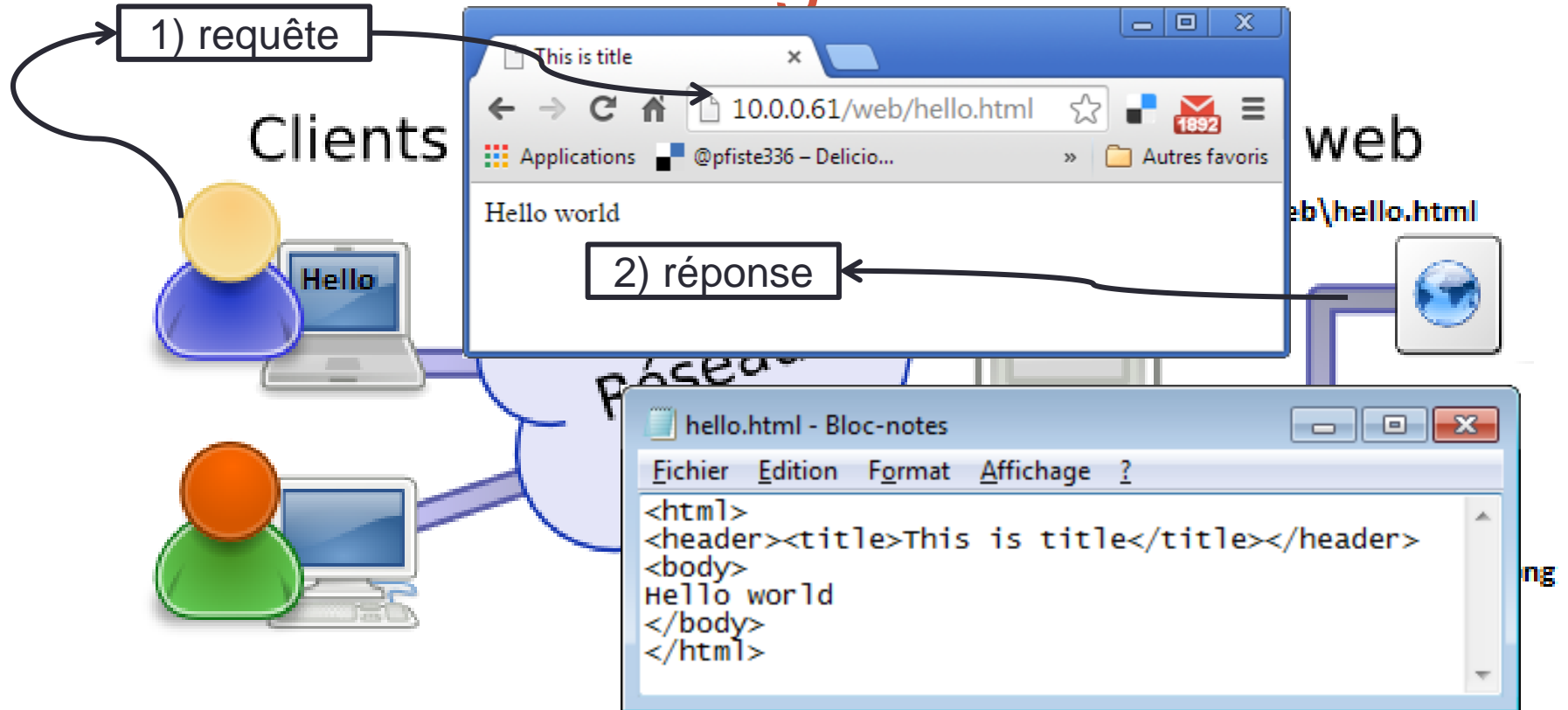


Clients



Dans tout réseau local, les données sont sérialisées (un seul "fil" pour les transporter)

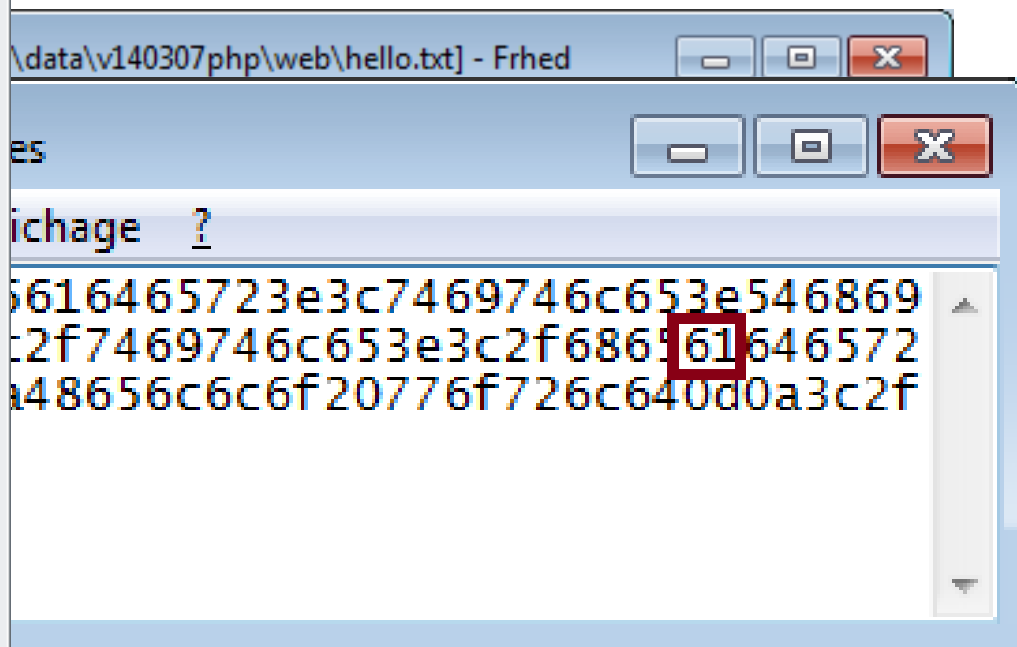
Documents échangés



Le document présent sur le serveur

Recherche des données

The image shows a screenshot of a Microsoft Excel spreadsheet. The top ribbon is visible with tabs for 'Fichier', 'Accueil', 'Insertion', 'Mise en forme', and 'Formules'. The 'Formules' tab is active, showing icons for Coller, Police, Alignement, Nombre, and Style. The spreadsheet has a grid with columns labeled A, B, and C, and rows numbered 68 to 88. Row 78 is highlighted in orange. The data in row 78 is 'a' in column A, '61' in column B, and '01100001' in column C. The status bar at the bottom shows 'Feuil1' and 'Feuil2' tabs, and a zoom level of 100%.



\\data\v140307php\web\hello.txt] - Frhed

es

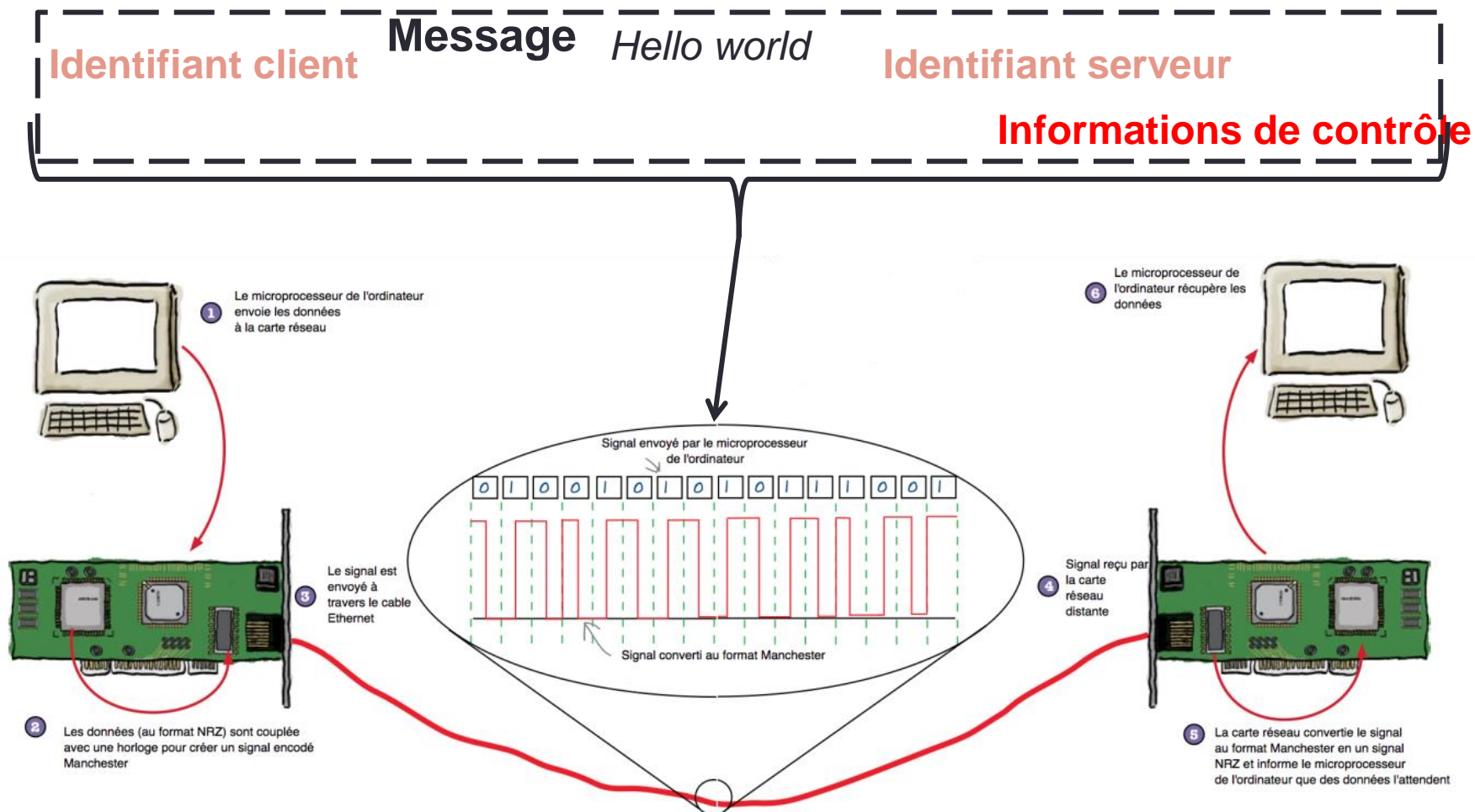
ichage ?

```

616465723e3c7469746c653e546869
c2f7469746c653e3c2f686961646572
a48656c6c6f20776f726c640d0a3c2f
  
```

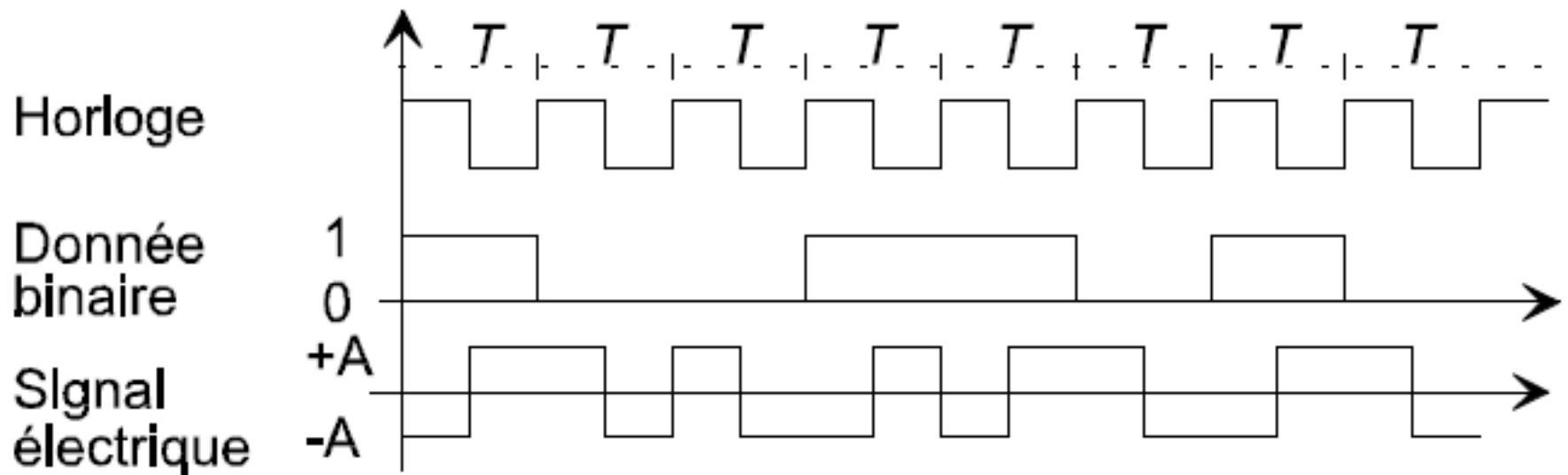
Interprétation hexadécimale

Transmission des données



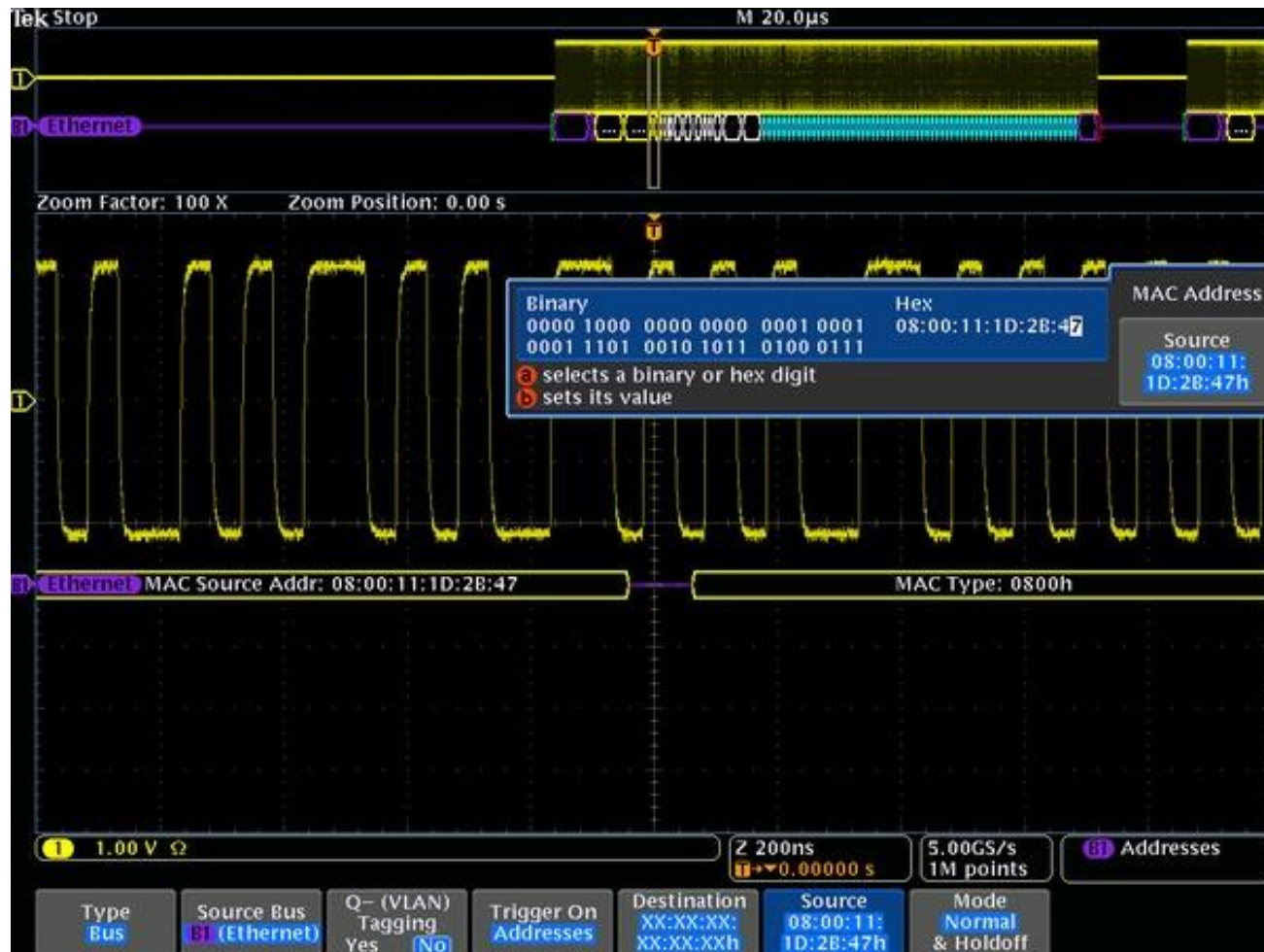
Cas d'un réseau local

Transmission synchrone avec le Codage Manchester



L'octet 10011010 sérialisé sur une ligne unique (XOR entre l'horloge et la donnée)

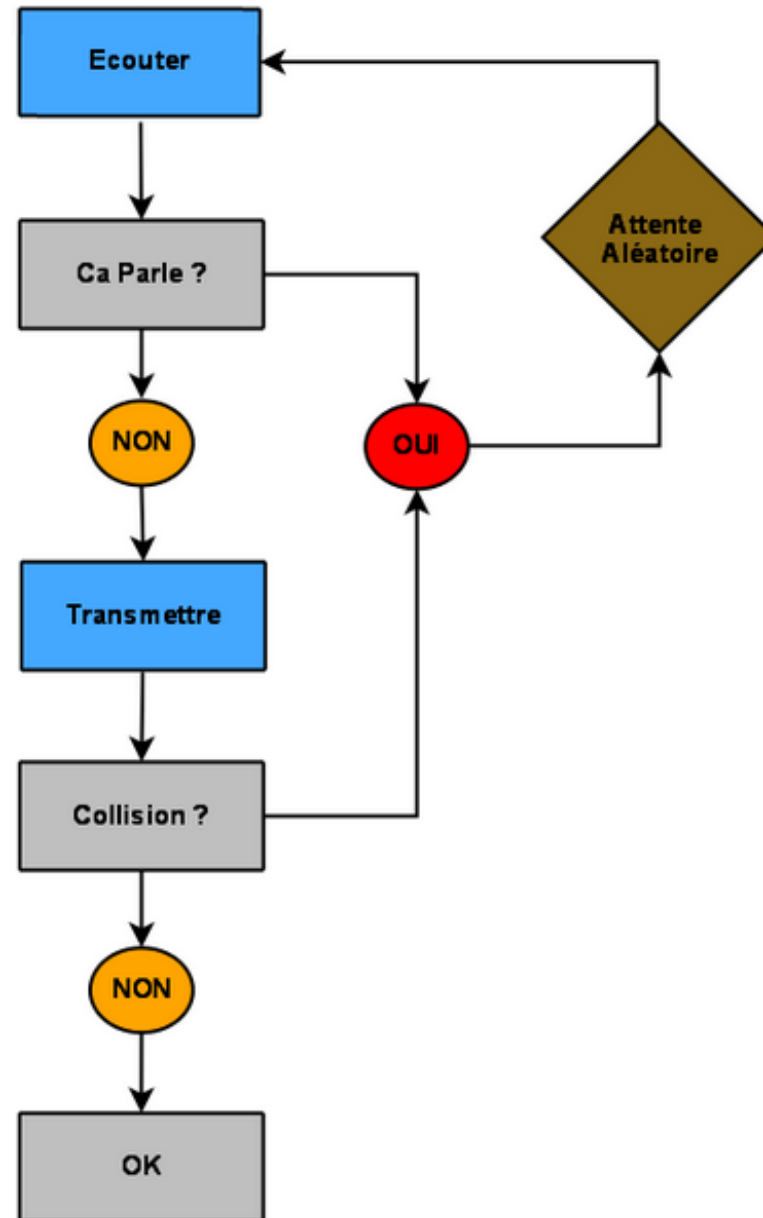
Les trames d'un réseau peuvent être analysées



Mode de transmission d'Ethernet

- L'information électrique est directement appliquée sur la ligne
- Les débits obtenus peuvent être très élevés, mais les distances ne peuvent être importantes à cause des phénomènes d'atténuation.
- Ethernet est réservé aux réseaux locaux

Détection des collisions



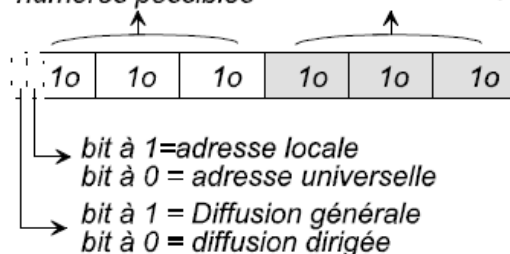
Format des trames Ethernet

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14 ... 1513	1514	1515	1516	1517
Adresse MAC destination						Adresse MAC source						Type de protocole	Données	FCS/CRC				

Format logique d'une trame Ethernet

ID Constructeur= OUI
3 octets - 2 bits
22 bits soit 2^{22} numéros possibles

Numéro de carte = 3 octets
soit 2^{24} numéros possibles



0x0800	IPv4
0x86DD	IPv6
0x0806	ARP
0x8035	RARP
0x0600	XNS
0x809B	AppleTalk
0x88CD	SERCOS III

Format d'une adresse MAC

Type de protocole

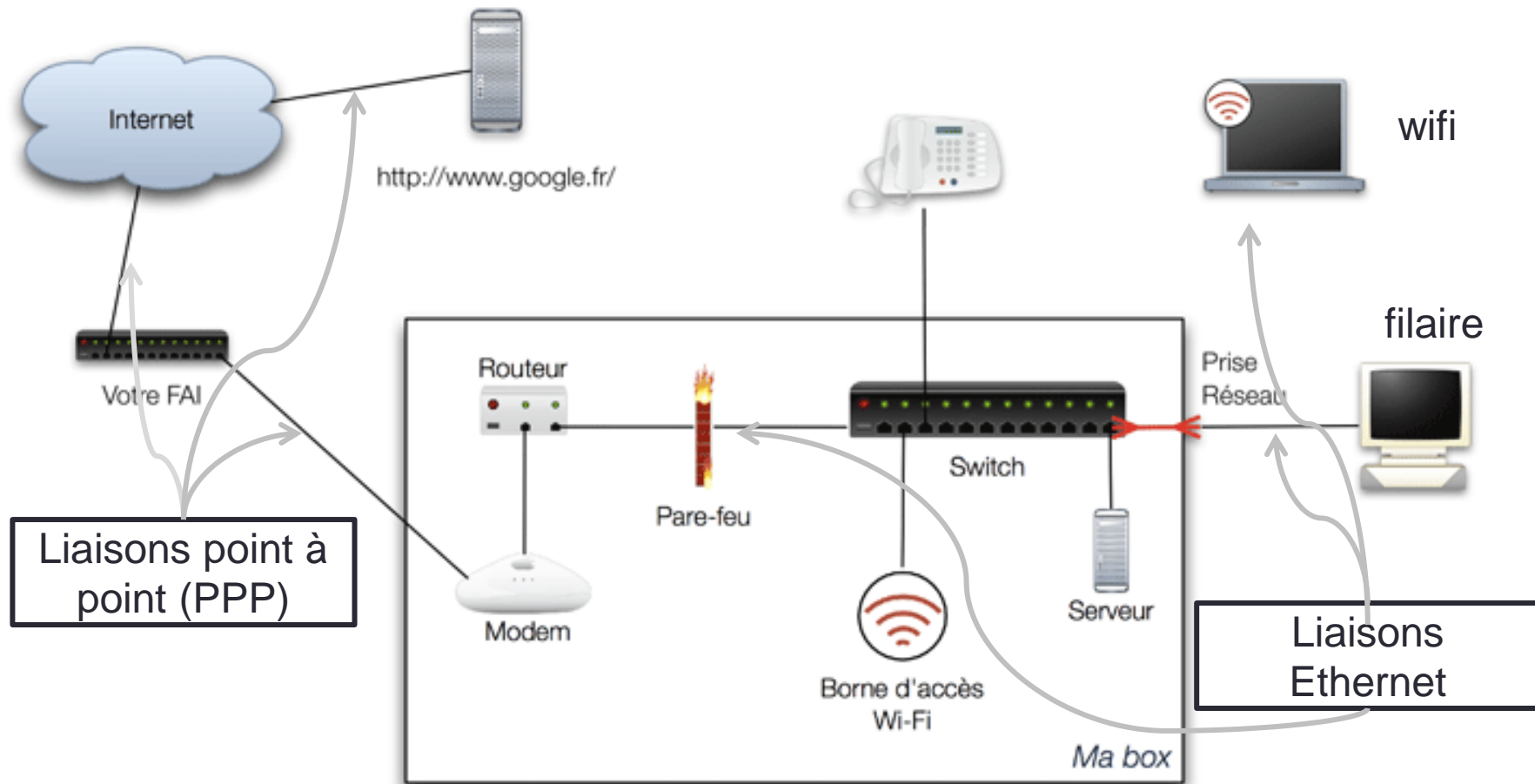
CRC: contrôle de redondance cyclique ou CRC (Cyclic Redundancy Check), c'est une fonction logicielle permettant de détecter les erreurs de transmission ou de transfert par ajout, combinaison et comparaison de données redondantes, obtenues grâce à une procédure de hachage.

http://fr.wikipedia.org/wiki/Contr%C3%B4le_de_redondance_cyclique

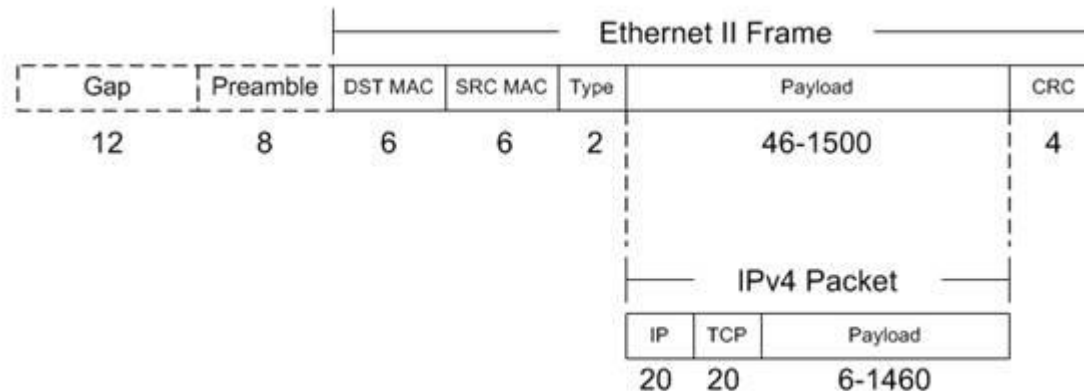
Transmission large bande

- Lorsque les distances augmentent, on utilise un signal sinusoïdal modulé par la valeur binaire à transporter.
- Les **modems** sont les dispositifs qui effectuent la transformation.
 - Le multiplexage fréquentiel permet d'obtenir plusieurs canaux.
 - Les liaisons modem sont des liaisons point à point.
- Les **routeurs** sont des dispositifs qui connectent modems et réseaux locaux.

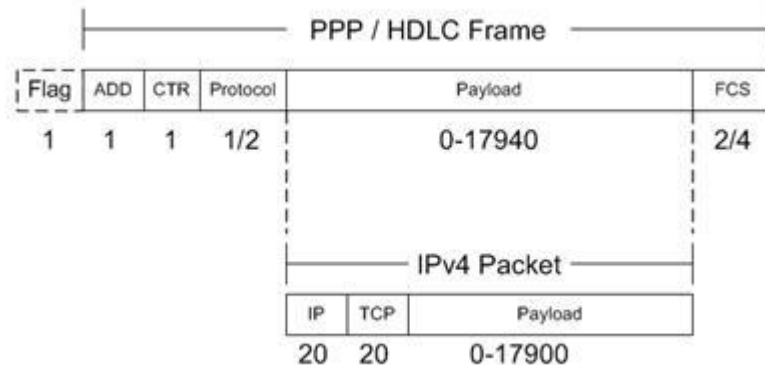
Configuration réseau domestique



Réseau TCP/IP : local vs étendu

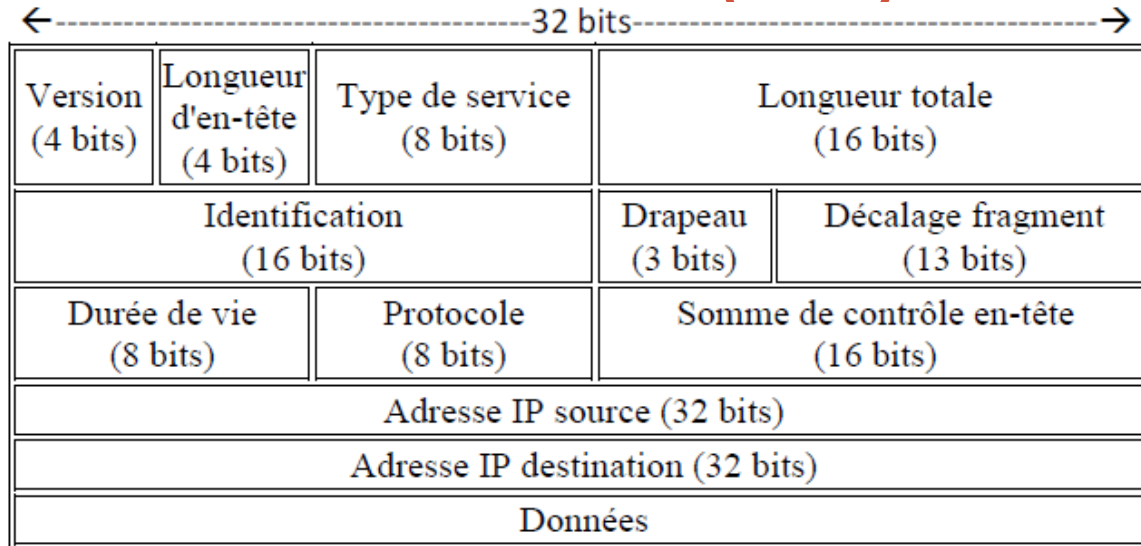


Trame ip dans une frame Ethernet (réseau local)



Trame ip dans une frame PPP (réseau étendu)

Réseau TCP/IP (V4)

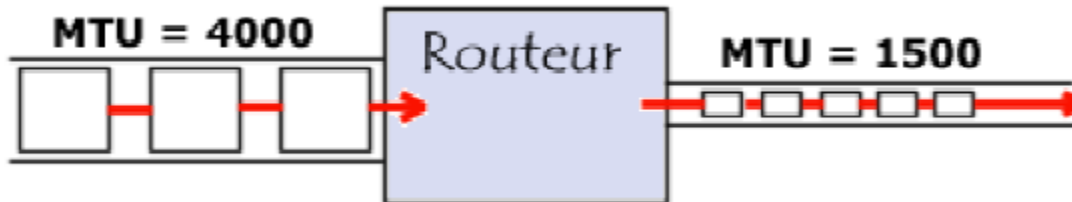


Datagramme IPV4

Le routage utilise les champs Adresse IP pour acheminer un datagramme IP à travers un réseau en empruntant le chemin le plus court. Ce rôle est assuré par des machines appelées routeurs reliant au moins deux réseaux.

Les champs Identification, drapeaux (flags) et déplacement de fragment permettent de gérer la fragmentation des datagrammes: la taille maximale d'un datagramme, 65536 octets, ne peut jamais être atteinte car les réseaux locaux ne le permettent pas.

Fragmentation des datagrammes



MTU = Maximum Transfer Unit

Le routeur fragmente les datagrammes en accord avec le MTU du réseau. Les fragments sont envoyés indépendamment les uns des autres et n'arriveront pas forcément dans le bon ordre.

Les champs Identification, Drapeaux et Déplacement permettent au destinataire le réassemblage correct des fragments.

Une trame UDP

Port Source (16 bits)	Port Destination (16 bits)
Longueur (16 bits)	Somme de contrôle (16 bits)
Données (longueur variable)	

User Datagram Protocol: appartient à la couche transport de la pile TCP/IP. Permet la transmission de paquets de manière simplifiée et rapide, lorsque l'intégrité des données n'est pas une priorité (voix, image, jeux en réseau).

- Chaque entité est définie par une adresse IP et un port.
- Travaille en mode non connecté. Pas de contrôle de flux ni de séquençement.
- Non fiable, cependant l'intégrité de chaque datagramme est garantie.

Une trame TCP

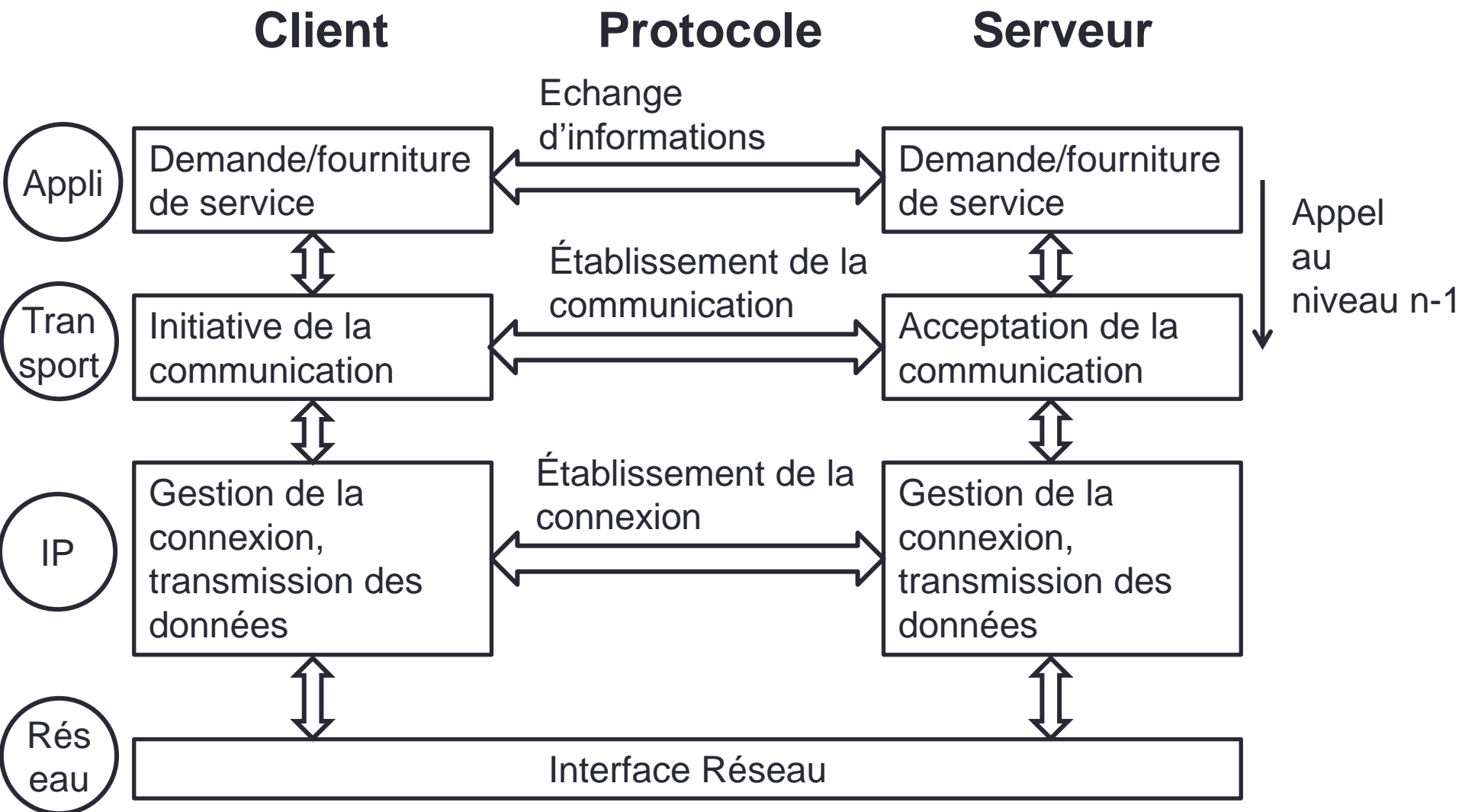
En bits

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Port Source																Port destination															
Numéro de séquence																															
Numéro d'acquittement																															
Taille de l'en-tête		réservé		ECN	URG	ACK	PSH	RST	SYN	FIN	Fenêtre																				
Somme de contrôle																Pointeur de données urgentes															
Options																						Remplissage									
Données																															

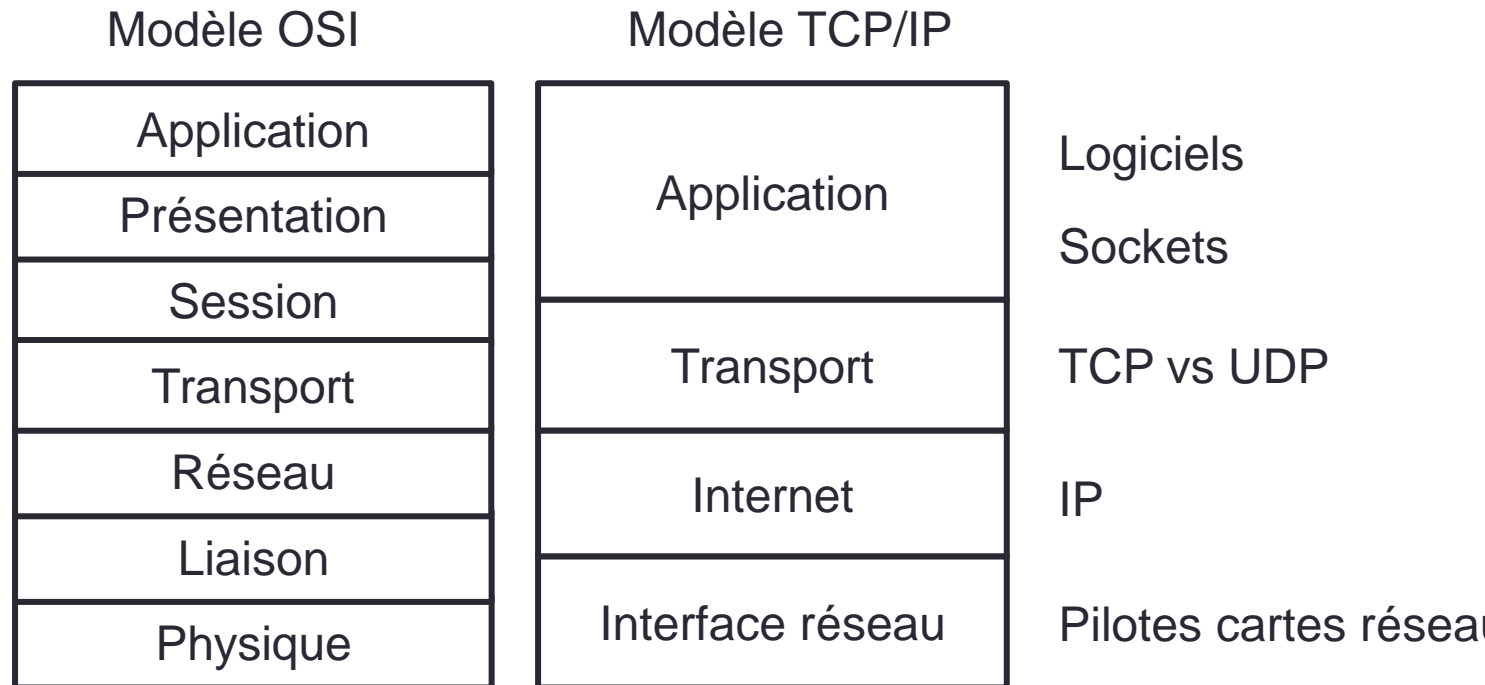
Transmission Control Protocol: appartient à la couche transport de la pile TCP/IP. Assure une communication sûre (accusés de réception), indépendamment des couches inférieures.

- Les routeurs (couche IP) ayant pour seul rôle l'acheminement des datagrammes sans gérer leur ordre, c'est la couche TCP qui réalise ce contrôle.
- Etablissement d'une connexion **client-serveur**: la machine qui **crée** la communication est le client, la machine qui **accepte** la communication est le serveur. Les machines dans un tel environnement communiquent en mode connecté, c'est-à-dire que la communication, après connexion, se fait **symétriquement** dans les deux sens, avec **maintien** d'une session.
- Les données sont augmentées d'un en-tête qui permet leur synchronisation.

Structuration en couches



Modèle TCP/IP vs Modèle OSI



OSI = Open Systems Interconnection, créé en 1978 par l'ISO, vocation normative, pas d'implémentation réellement opérationnelle.

TCP/IP: détails

Telnet	FTP	HTTP	DNS	RIP	SNMP
TCP (Transport Control Protocol)			UDP (User Datagram Protocol)		
IP (Internet Protocol)					
PPP	Ethernet		Token Ring	Frame Relay	ATM

Quelques services TCP/IP

- Telnet - port 23 : prise de contrôle à distance
- FTP = File Transfert Protocol - port 21 : transfert de fichiers
- SMTP = Simple Mail Transfert Protocol - port 25
envoi de messages électroniques
- POP = Post Office Protocol - port 110
lecture boîte aux lettres électroniques
- NNTP = Network Net Transport Protocol - port 119
Forums de discussions (newgroup)
- HTTP = Hyper Text Transfert Protocol - port 80
affichage de pages WEB

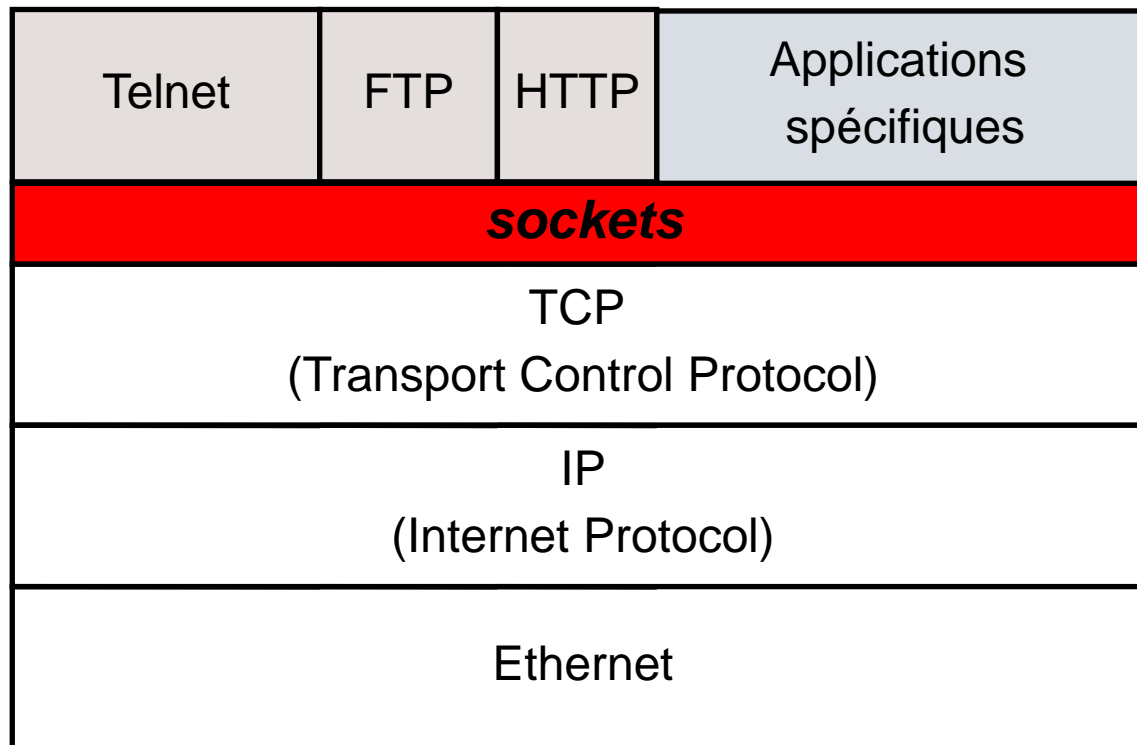
Conception des applications

- Deux approches sont possibles pour la conception d'applications en mode client-serveur
 - Orientation protocole applicatif: on définit un protocole (mots-clés, séquences) spécifique portant la sémantique applicative. (arch. Unix: SMTP, FTP, etc..)
 - Orientation application: on conçoit une application classique, et on la sépare en modules qui communiquent avec des appels de procédures à distance. (RPC, Corba, RMI)

Les protocoles applicatifs

- Le client et le serveur applicatif implémenteront ces protocoles (mots-clés, séquences, algorithmes) pour générer et analyser les données échangées, en conformité avec la sémantique applicative.
- Ces protocoles s'empilent par dessus les protocoles internes de TCP/IP.

TCP/IP: les sockets



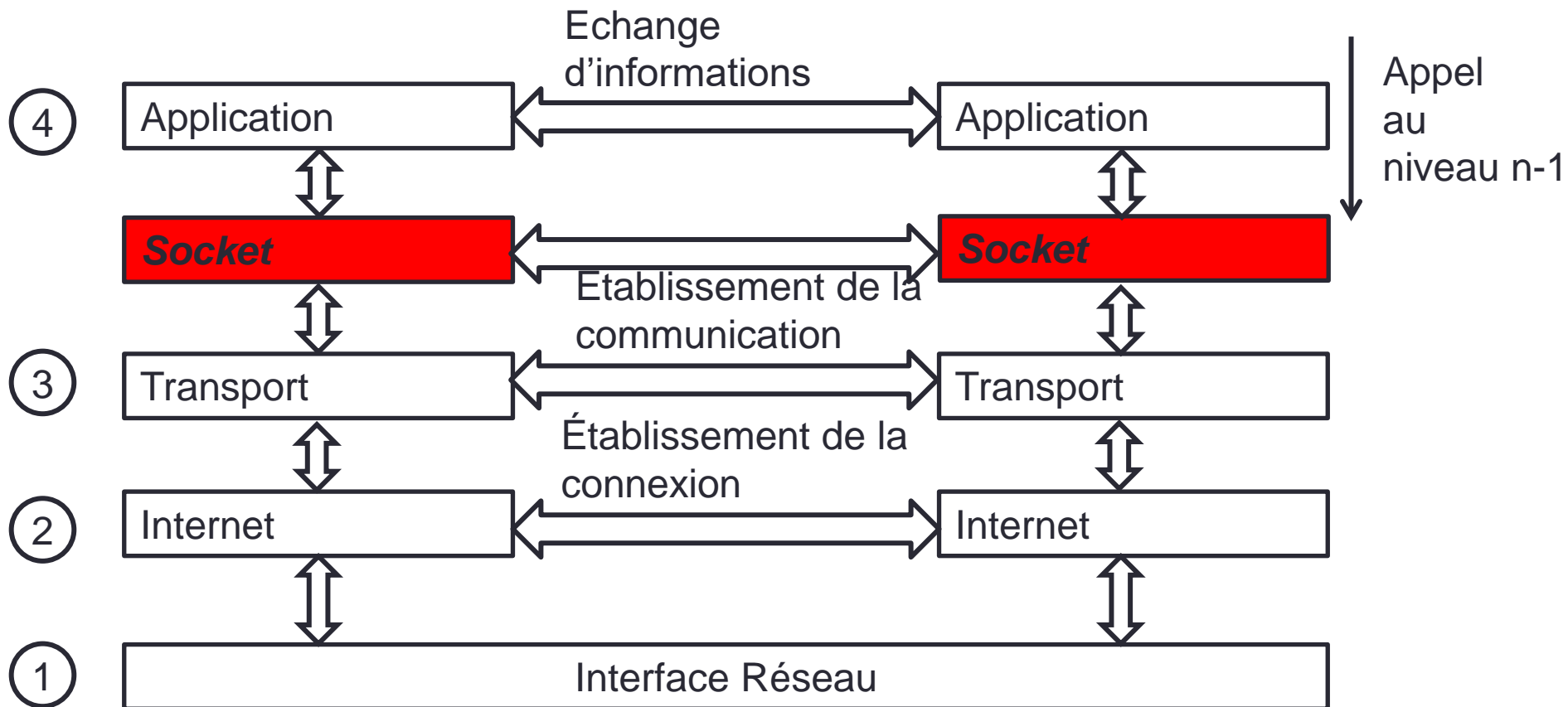
socket = bibliothèque logicielle pour programmer des applications réseau TCP/IP. Disponible dans tous les langages et tous les systèmes d'exploitation.

Structuration en couches

Client

Protocole

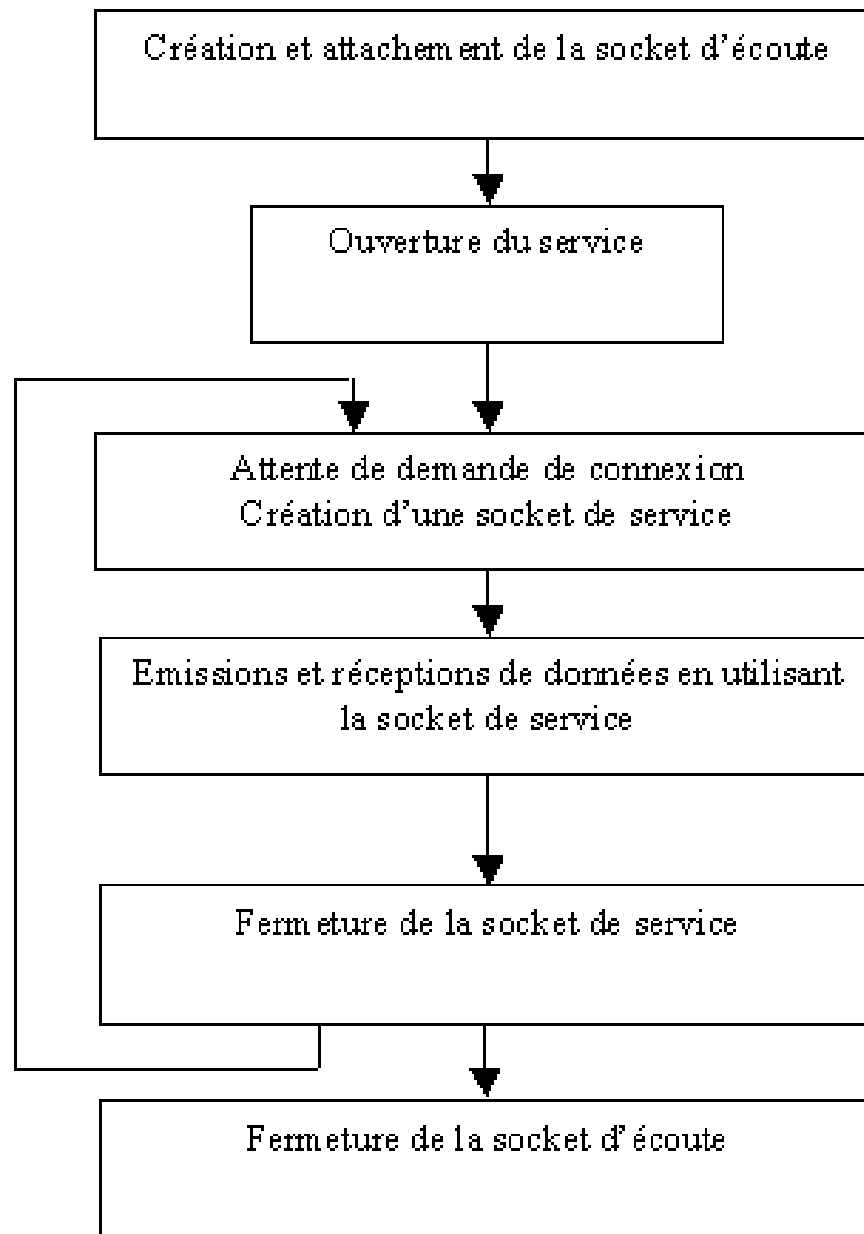
Serveur



Socket

- Permet les connexions à d'autres machines
- Envoie des données
- Reçoit des données
- Ferme la connexion
- S'attache à un port
- Écoute pour l'arrivée des données
- Accepte la connexion provenant d'une autre machine sur le port attaché

Fonctionnement du serveur



Deux types de socket:

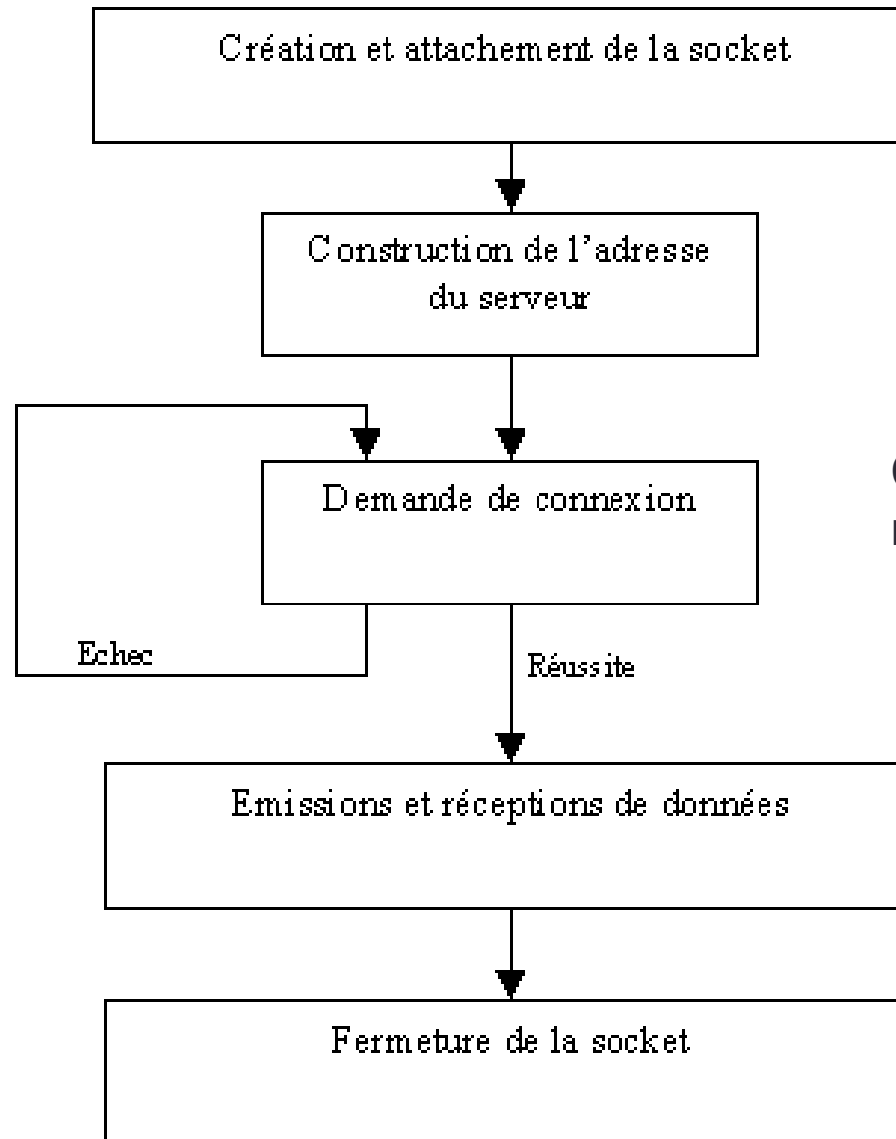
- Socket d'écoute
- Socket de service

- Une seule socket d'écoute
- Autant de sockets de service que de clients connectés

Cycle de vie d'un serveur

1. Un nouveau `ServerSocket` est créé utilisant un port spécifique.
2. Le `ServerSocket` écoute sur ce port pour d'éventuel demande de connexion.
3. Avec la commande `Accept`, le serveur peut engager la connexion.
4. le `Socket` de service utilise les méthodes `getInputStream()` et `getOutputStream()` pour la communication avec le client.
5. Le serveur et le client interagissent en fonction d'un protocole commun jusqu'à ce que la communication soit coupée.
6. Le serveur retourne à l'étape 2 et attend pour une autre demande de connexion.

Fonctionnement du client



Une seule socket

Cas de la connexion non persistante.

Des constructeurs

- **public ServerSocket(int port) throws IOException**

Creates a server socket, bound to the specified port. A port number of 0 means that the port number is automatically allocated, typically from an ephemeral port range. This port number can then be retrieved by calling getLocalPort.

- **public Socket(String host,int port) throws UnknownHostException, IOException**

Ce constructeur crée un socket TCP et tente de se connecter sur le port indiqué de l'hôte visé. Le premier paramètre de ce constructeur représente le nom de la machine serveur. Si l'hôte est inconnu ou que le serveur de noms de domaine est inopérant, le constructeur générera une UnknownHostException. Les autres causes d'échec, qui déclenchent l'envoi d'une IOException sont multiples : machine cible refusant la connexion sur le port précisé ou sur tous les ports, problème lié à la connexion Internet, erreur de routage des paquets...

Voici un exemple d'utilisation de ce constructeur :

```
Socket leSocket = new Socket("smtp.free.fr", 25);
```


Quelques méthodes de la classe Socket

- **public Socket accept() throws IOException**

(Class ServerSocket) Listens for a connection to be made to this socket and accepts it. The method blocks until a connection is made.

- **public InputStream getInputStream() throws IOException**

Cette méthode renvoie un flux d'entrées brutes grâce auquel un programme peut lire des informations à partir d'un socket.

```
DataInputStream fluxEnEntree = new  
DataInputStream(leSocket.getInputStream());
```

- **public OutputStream getOutputStream() throws IOException**

Cette méthode renvoie un flux de sortie brutes grâce auquel un programme peut écrire des informations sur un socket.

```
DataOutputStream fluxEnSortie = new  
DataOutputStream(leSocket.getOutputStream());
```

Un serveur simpliste

```

11 public class OneshotServer {
12     static int port = 23;
13
14     public static void main(String[] args) throws IOException {
15         System.out.println("Je suis le serveur, j'écoute sur le port " + port);
16         ServerSocket socketEcoule = new ServerSocket(port);
17         System.out.println("En attente d'une connexion");
18         Socket socketService = socketEcoule.accept();
19         System.out.println("Une connexion est acceptée ("
20             + socketService.getRemoteSocketAddress() + ")");
21         BufferedReader entree = new BufferedReader(new InputStreamReader(
22             socketService.getInputStream()));
23         PrintStream sortie = new PrintStream(socketService.getOutputStream());
24         String requeteClient = entree.readLine();
25         System.out.println("le client demande: " + requeteClient);
26         String reponse = "commande inconnue";
27         if (requeteClient.endsWith("Quelle heure est-il ?")
28             || requeteClient.endsWith("date")) {
29             Date d = new Date();
30             reponse = d.toString();
31         }
32         System.out.println("réponse=" + reponse);
33         sortie.println("Bonjour, ici le serveur, vous avez demandé: "
34             + requeteClient + " ,voici la réponse [" + reponse + "]");
35         socketService.close();
36         socketEcoule.close();
37         System.out.println("Terminé !");
38     }
39 }

```

Un client simpliste

```

public class CircleClient {

    private static final int PORT = 8051;
    private static final String hostname = "127.0.0.1"; // "146.19.4.106" adapter au serveur
    private static boolean endSession;

    public static void main(String[] args) {
        PrintWriter out = null;
        BufferedReader networkIn = null;
        System.out.println("(démarrage du client) veuillez patienter...");
        System.out.println("(vers le serveur) " + hostname + ":" + PORT);
        Socket theSocket = null;
        try {
            theSocket = new Socket();
            theSocket.connect(new InetSocketAddress(hostname, PORT), 200);
            int localPort = theSocket.getLocalPort();
            System.out.println("Client démarré sur le port" + ":" + localPort);
            networkIn = new BufferedReader(new InputStreamReader(theSocket.getInputStream()));
            BufferedReader userIn = new BufferedReader(new InputStreamReader(System.in));
            out = new PrintWriter(theSocket.getOutputStream());
            System.out.println("Connecté au serveur");
            while (!endSession) {
                System.out.println("entrez une requête au clavier "
                    + "(le serveur comprend \n [date];\n"
                    + " [circle x y radius];\n tapez [quit] pour stopper ma session sur le serveur\n"+
                    "tapez [fin] pour me stopper)");
                String theLine = userIn.readLine();
                if (theLine.equals("quit")) {
                    System.out.println("le serveur va terminer ma session");
                    endSession = true;
                }
                if (theLine.equals("fin"))
                    break;
                out.println(theLine);
                out.flush();
                System.out.println(networkIn.readLine());
            }
        } catch (IOException e) {
            System.err.println(e);
            System.out.println("plus de connexion");
        } finally {
            try {
                if (networkIn != null)
                    networkIn.close();
                if (out != null)
                    out.close();
                if (theSocket != null)
                    theSocket.close();
            } catch (IOException ex) {
            }
        }
    }
}

```

Ressources

- Réseau
 - <http://www.hsc.fr/ressources/articles/protocoles/tcp/index.html>
 - <http://www.gipsa-lab.grenoble-inp.fr/~christian.bulfone/MIASS/PDF/>
- Sockets
 - <https://docs.oracle.com/javase/tutorial/networking/TOC.html>
 - Java Network Programming 4ed – 2013, Elliotte Rusty Harold (O'REILLY éditeur)