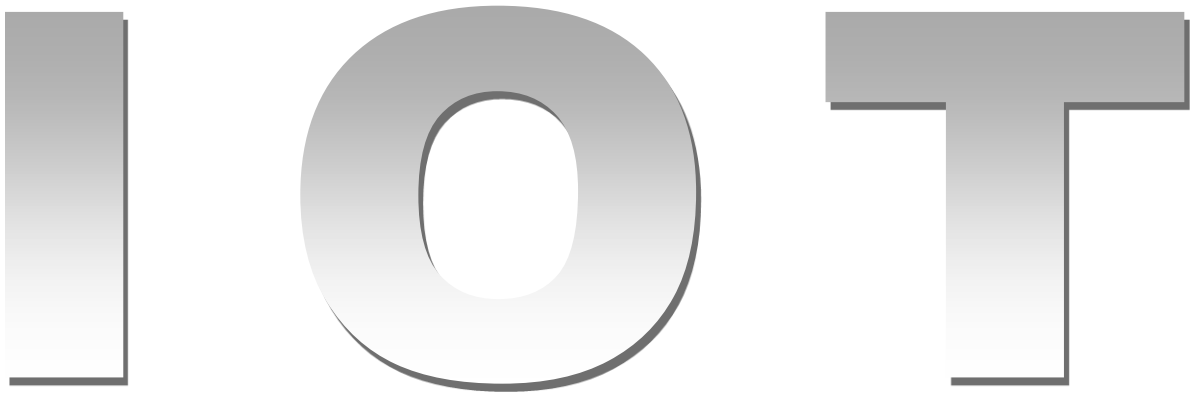


Internet Of Thing

Internet Des Objets (IdO)

Généralité (2)

Définitions

- L'internet des objets est « une infrastructure mondiale pour la société de l'information, qui permet de disposer de services évolués en interconnectant des objets (physiques ou virtuels) grâce aux technologies de l'information et de la communication interopérables existantes ou en évolution. » -- ITU via wikipedia

- The Internet of Things, also called The Internet of Objects, refers to a wireless network between objects, usually the network will be wireless and self-configuring, such as household appliances.
  -- autre wiki

- **But :** pouvoir être connecté n'importe où, n'importe quand, avec n'importe quoi

Généralité (3)

- **Résumé IOT:**

un ensemble d'objet physique : actionneurs ou capteurs (relais, capteur température, cardiaque, ..) reliés via I2C, SPI, digital I/O, etc…

qui veulent communiquer entre eux ou via internet

avec d'autres logiciels (serveur web, serveur applicatif,…)

via différents protocoles de communication.

(RFID, NFC, Wi-fi, GSM, LTE,….)

Problème :

Unicité pour l'identification des objets

Problématique

- Quelles informations récupérées ?

  capteurs ➔ data

  Que faire de ces données ?

  ⟹ Big DATA

- Sécurisation des données ?
- Autonomie ?
- Diversité, standardisation ??

  (chacun veut créer quelque chose ..)

Communication

- Permet de relier 2 objets entre eux (wireless)
- De nombreux protocoles existent
  - Propriétaires /standardisés
  - Secteurs d'activités
  - Longue distance, courte distance
  - Grande bande passante, très petite
  - Chiffrage possible ou non
  - Nombre de capteurs
  - Accès à internet ?
  - Autonomie…

Répondre à ces questions permets de choisir un protocole et ensuite d'acheter les objets adéquats.

Protocoles connus (1)

- Longue ou moyenne distance
  - Protocole orienté téléphonie : GSM, GPRS, UMTS, LTE
    (nécessite une carte SIM)
  - Protocole réseau non-filaire → Wi-fi

Très consommateur en énergie
→ pas recommandé en ioT
sauf si relié au courant électrique

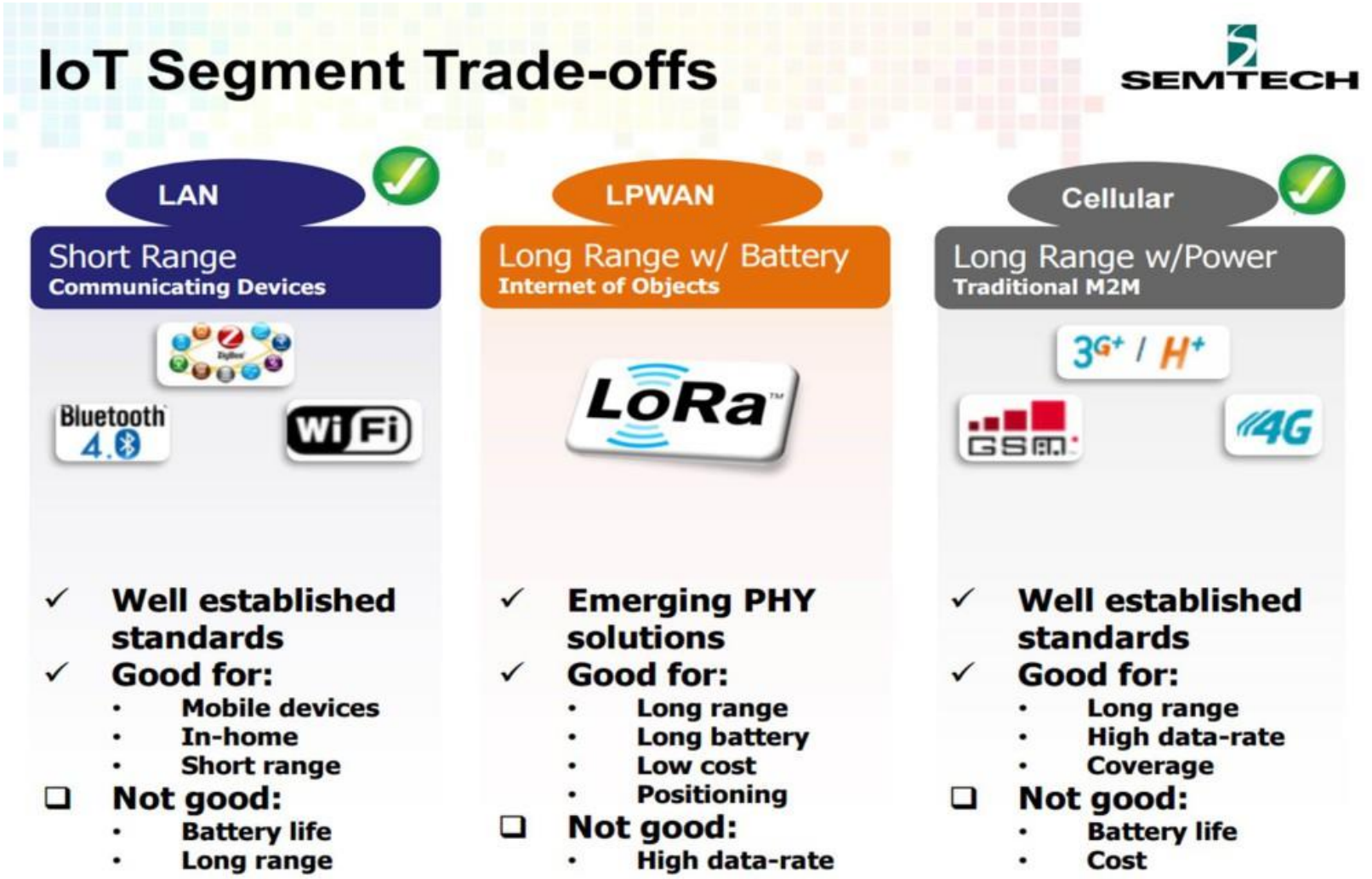
Protocoles connus (2)

- Mais évolution de ces protocoles….
  - Apparition du wifi low energy
    - Théoriquement 10000 fois moins consommateur que Wifi
      - Mais même utilisation
      - Compatible Wifi
  - NB-IOT (Narrow Band IOT) → LTE-M
    - Évolution de la 4G vers une norme 5G NR (New Radio)
    - Utilisation de la bande passante opérateur
    - Utilisation de la couverture opérateur
    - Fait partie des protocoles LPWAN (Low Power Wide Area Network)
    - Faible débit, faible temps de réponse, consommation puissance très faible
    - Utilisation carte SIM

Protocoles (1)

- Autres protocoles longue distance
  - Caractéristique : - très économe en énergie
    - distance > 50 km sur le 433Mhz et **868Mhz** (bande passante libre)
    - très peu de débit , sécurisé
    - chaque capteur a un numéro unique
  - Echange entre un capteur et récepteur (point à point)
  - Sigfox : protocole propriétaire (Ultra Narrow Band)
    - Max 140 messages / jour, 12 octets par message max
    - 4 catégories : 0, meilleur qualité, 3 erreurs de transmission possible
  - LoRaWAN (Low Power Wide Area Network)
    - Standardisé, consortium d'opérateur (Bouygue, Orange,..)
    - Débit de 0,3kps à 50 kps
    - Message de 50 octets

Protocoles (1 bis)

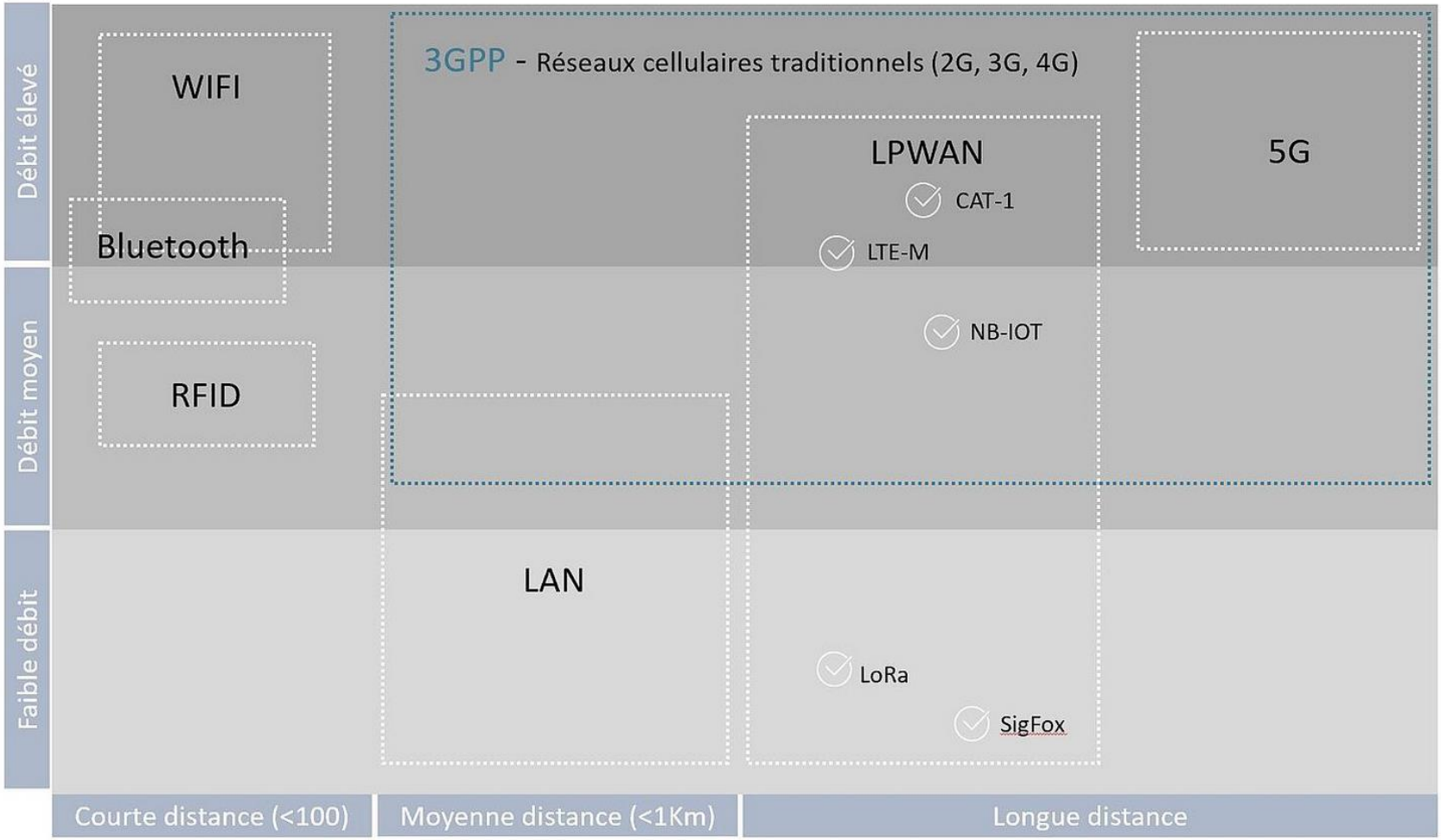


Plus d'info : http://www.oezratty.net/wordpress/2015/reseaux-m2m-1/

Protocole (2)

- Protocoles courte distance (domotique)
  - Sans connexion vers internet
    - Bluetooth, Z-wave, 433 Mhz, NFC, enocean, ANT+, BACnet…
    - Protocole propriétaire …
    - Utilisation d'une passerelle pour aller sur internet
  - Avec connexion vers internet
    - ZigBee, 6LowPan, Thread…
- Protocoles de haut niveau
    - HTTP, XMPP, MQTT, DDS,…,

Protocole (3)


Débit élevé
Débit moyen
Faible débit
WIFI
Bluetooth
RFID
3GPP - Réseaux cellulaires traditionnels (2G, 3G, 4G)
LPWAN
CAT-1
LTE-M
NB-IOT
5G
LAN
LoRa
SigFox
Courte distance (<100)
Moyenne distance (<1Km)
Longue distance

Enocean

- Société, filiale de Siemens
- Avantage : sans fil, sans énergie

  ⟹ utilisation énergie photovoltaïque, piezo-électrique

- Norme Smart Home ISO/IEC 14543.3.1x
  - Utilisation 868 MHz
  - Max 30 m intérieur, 300 m extérieur
  - Très court message (1ms) à 125 kb/s
  - Généralement , monodirectionnel,
    (bidirectionnel si batterie ou courant)
  - économie d'énergie car endormi sauf au moment d'émettre
  - Possibilité de cryptage en AES128
- Nécessite une passerelle pour communiquer

Bluetooth (1)

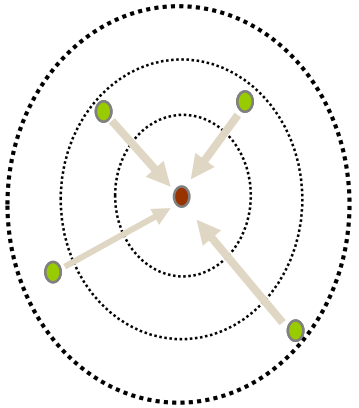
- Historique
  - Début 1994, via Ericsson
  - Evolution des normes : 802.15.1 et 802.15.2
    - Norme : bluetooth v4.2 → débit 100 Mb/s, sur 100 m
    - Apparition du Bluetooth Low Energy (ble ou smart)
    - Actuellement : bluetooth 5
  - Bande passante à 2,4 Ghz
  - Technologie FHSS
    (saut de fréquences, 79 canaux, toutes les 625 µs)
  - Identifiant sur 48 bits (Bluetooth Device Address)

Bluetooth (2)

- Piconet
  - Utilisation d'un maître et de 7 esclaves max
  - Topologie en étoile
  - Communication entre maitre-esclave, et pas entre esclave-esclave
  - Le maitre interroge l'esclave qui répond (réponse max sur 3 slots→ 3* 625 µs)
- Fonctionnement
  - Périphérique en mode passif → écoute
  - Communication synchrone ou asynchrone
- Connexion
  - Phase inquisition (qui se trouve à porter du maître ?)
  - Phase de paging (choix des esclaves) → appareillage (code PIN)

Bluetooth (3)

- Connexion
  - Phase inquisition (inquiry)
    - Maître envoie une demande de présence
    - Chaque esclave répond avec son adresse



  - Phase paging
    - Maître choisit ses esclaves
    - Affectation d'une adresse sur 3 bits
    - Communication via l'adresse, retour par ACK
  - Découverte des services (SDP)
  - Création d'un canal de communication via le protocole L2CAP, et RFCOMM (port série)

    (sécurisation via un code PIN 0000 ou 1234)

Z-Wave (1)

- Développé par Zen-SYS et racheté par Sigma designs
- Toutes les puces sont fabriquées par Sigma designs

  ➔ aucun problème d'interopérabilité

  mais module de communication assez cher

- **But :** protocole fiable, économe en énergie et orienté vers la domotique

- Caractéristiques
  - 30 m intérieur, 300 m extérieur
  - Débit de 9,6 kb/s à 100 kb/s
  - Utilisation des ondes 868 Mhz
  - Mode non connecté

Z-Wave (2)

- Différents nœuds
  - Device final (esclave) : reçoit des ordres et envoie de temps en temps des données

    ➔ 95% du temps endormi pour économiser la batterie

  - Nœuds avec alimentation → option routage
    - Contrôleurs:
      - Contrôleur primaire : carte des nœuds Z-wave, ajout d'un nœud, suppression, etc.. (232 nœuds max)
      - Contrôleur secondaire : recopie la carte des nœuds

      Seuls les contrôleurs peuvent faire des requêtes

    - Esclaves : permet de faire passer un message d'un de ses voisins vers un autres de ses voisins ou répond à une requête

Z-Wave (3)

- Cas particulier
  - Alarme
    - Inondation du réseau par le device
    - Récupérer par le contrôleur primaire
- Durée de vie
  - Entre 12 et 20 mois….
- Chiffrement AES 128 bits

802.15.4 (1)

- ZigBee, 6lowPAN
  - Couche accès réseau : 802.15.4
    - Bande passante 2,4 Ghz, ou 868Mhz
    - Technologie DSSS (concurrent du Wifi)

      Wifi 13 canaux

      -- canal 1 : 2,404 à 2,420 canal 6 : 2,429 à 2, 445 canal 13: 2,456 à 2,470 sans chevauchement seulement pour 3

      802.15.4

      -- canal 11 : 2,405 canal 14: 2,420 canal 15 : 2,425 canal 26 : 2,480

      utilisation canal 15, 20, 25 et 26 car wifi utilise 1, 6 ,13

    - 2 méthodes accès au médium
      - Accès coordonné → élection d'un chef
      - Accès asynchrone : CSMA/CA (le plus utilisé), avec acquittement obligatoire

802.15.4 (2)

Association

- Le coordinateur envoie régulièrement une *trame beacon* pour indiquer le PAN ID
- Un nœud fait une requête d'association en indiquant le PAN ID
- Si le coordinateur est ok, envoie d'une adresse courte (16 bits) au nœud pour association

Trame

Préambule	Start Frame delimiter	longueur	Data
32 à 40 bits	11100101	7 bits + 1 b à 0	max 127 octets

Au niveau data, rajout entête niveau 2 non sécurisé, donc environs 102 octets de données….

⟹ Insuffisant pour de l'IP ou IPv6

ZigBee (1)

- Au dessus du 802.15.4
- PAN id entre 0X0000 et 0X03FF
- 250 kb/s
- De 50 à 200 m
- Économie d'énergie

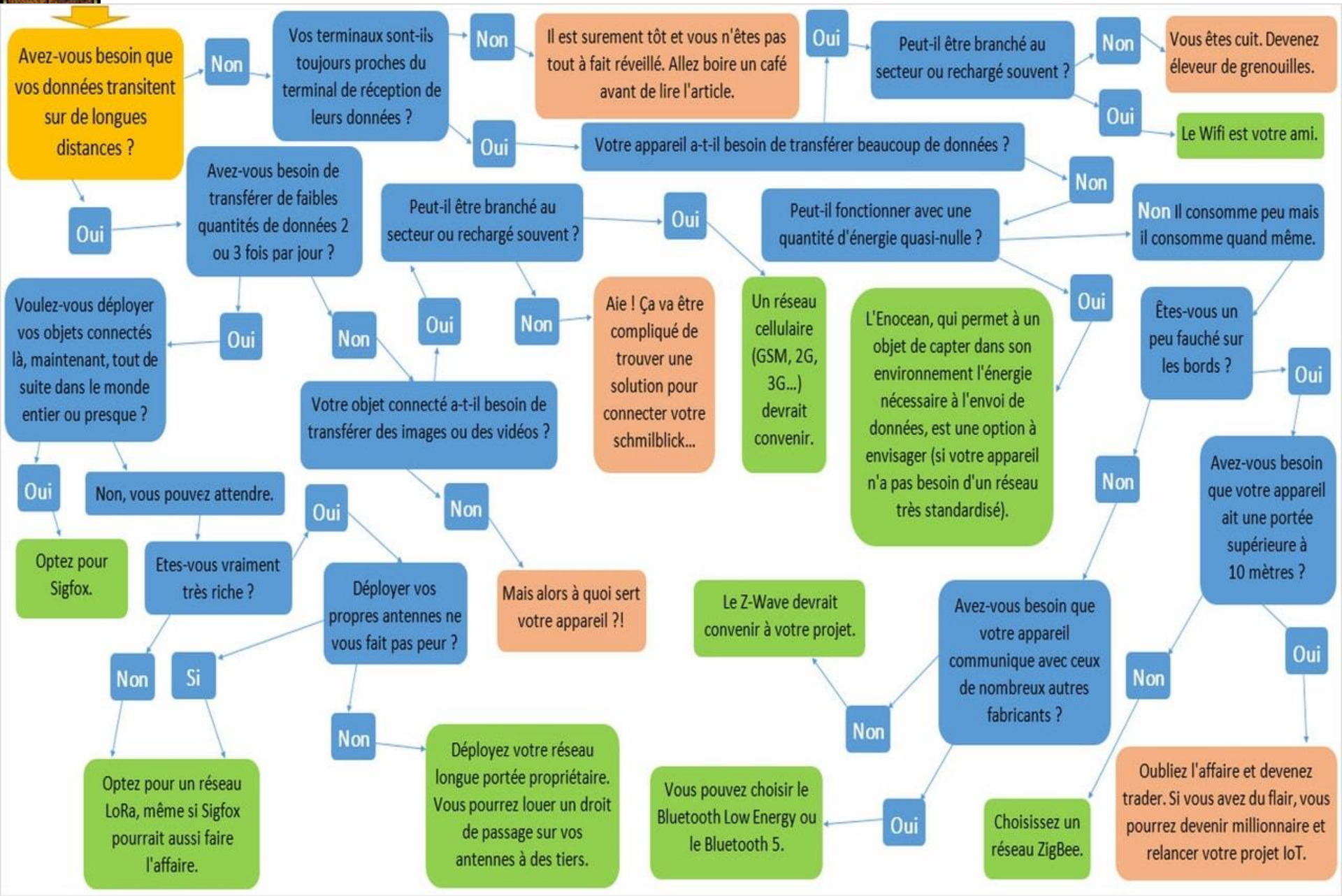
- Couche APS (Application Support Layer)
  - Assure la correspondance @supérieur @16 bits
  - Gère la fragmentation des données
  - Gère la gestion des acquittements
  - Gère la table de voisinage

6lowPAN (1)

- Protocole conçu pour IPv6
- *IPv6 Low power Wireless Personal Area Networks*
  - Réussir à intégrer les @IPv6 dans peu d'octets…
    utilisation compression

    102 octets – AES128 (21) –h_IP(40) –h_TCP(20) =21 octets…..

- Caractéristiques
  - Routage IP
  - Fragmentation et réassemblage
  - Compression en-tête
  - Auto-configuration des @IP



http://www.journaldunet.com/ebusiness/telecoms-fai/1181267-les-reseaux-iot/

Programmation réseau

- Généralités
- Les sockets en IPv4
  - En langage C, en Java

Les couches de l'OSI

7	Application *Application*	échanges de données d'application (selon l'application)
6	Présentation *Presentation*	mise en forme des données pour la transmission
5	Session *Session*	synchronisation de processus
4	Transport *Transport*	transfert de blocs d'octets entre processus
3	Réseau *Network*	transfert de blocs d'octets entre systèmes (pas forcément raccordés au même médium)
LLC	Contrôle de lien logique *Logical Link Control*	transfert fiable de blocs d'octets entre systèmes raccordés au même médium
MAC	Contrôle d'accès au médium *Medium Access Control*	transfert de bits entre systèmes raccordés au même médium, avec contrôle du droit d'émission
1	Physique *Physical*	transfert de bits entre systèmes raccordés au même médium

Les couches de TCP/IP

Application	échanges de données d'application (selon l'application) mise en forme de données échangées synchronisation de processus distants
Transport	transfert de blocs d'octets entre processus
Internet	transfert de blocs d'octets entre systèmes distants (pas forcément raccordés au même médium)

Accès Réseau	transfert fiable de blocs d'octets entre systèmes raccordés au même médium

Notion client /serveur (1)

- Architecture client/serveur
  - Un client : envoie des requêtes
  - Un serveur : attend des requêtes et renvoie des réponses
  - Une machine peut être à la fois cliente d'une application et serveur d'une autre application …

    mais

    aussi de la même application.

- Nom du serveur :
  - FQDN (Fully Qualified Domain Name)
  - Nom de la machine, unique
  - Adresse IP

Notion client /serveur (2)

- Exemple
  - Serveur : Web (démon httpd, apache, iis, nodejs, etc…)
    - Démarrer au lancement de la machine
    - En attente de connexions ou de requêtes
  - Client : navigateur web
    - *Fait une requête en utilisant une syntaxe précise*
      (décrit dans le protocole http)

- Réception d'une requête
  - Vérification de la syntaxe
    - Ok, la demande est traitée
    - Pas ok, soit abandon, soit envoi d'un message d'erreur

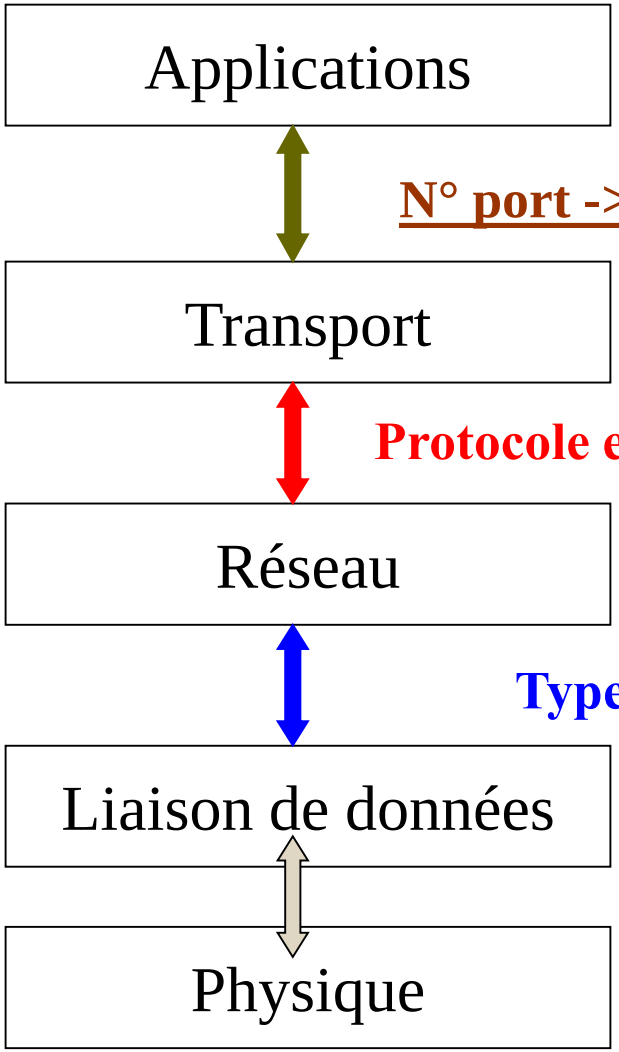
Généralités sur les sockets(1)

- Mécanisme d'interface de programmation
  - permet aux processus d'échanger des données
  - n'implique par forcément une communication par le réseau (ex socket unix)

- Une connexion est entièrement définie sur chaque machine par :
  - le type de protocole (TCP, UDP,…)
  - l'adresse IP
  - **le numéro de port associé au processus**

    → (statiquement pour le serveur, dynamiquement pour le client)

Généralités (2)



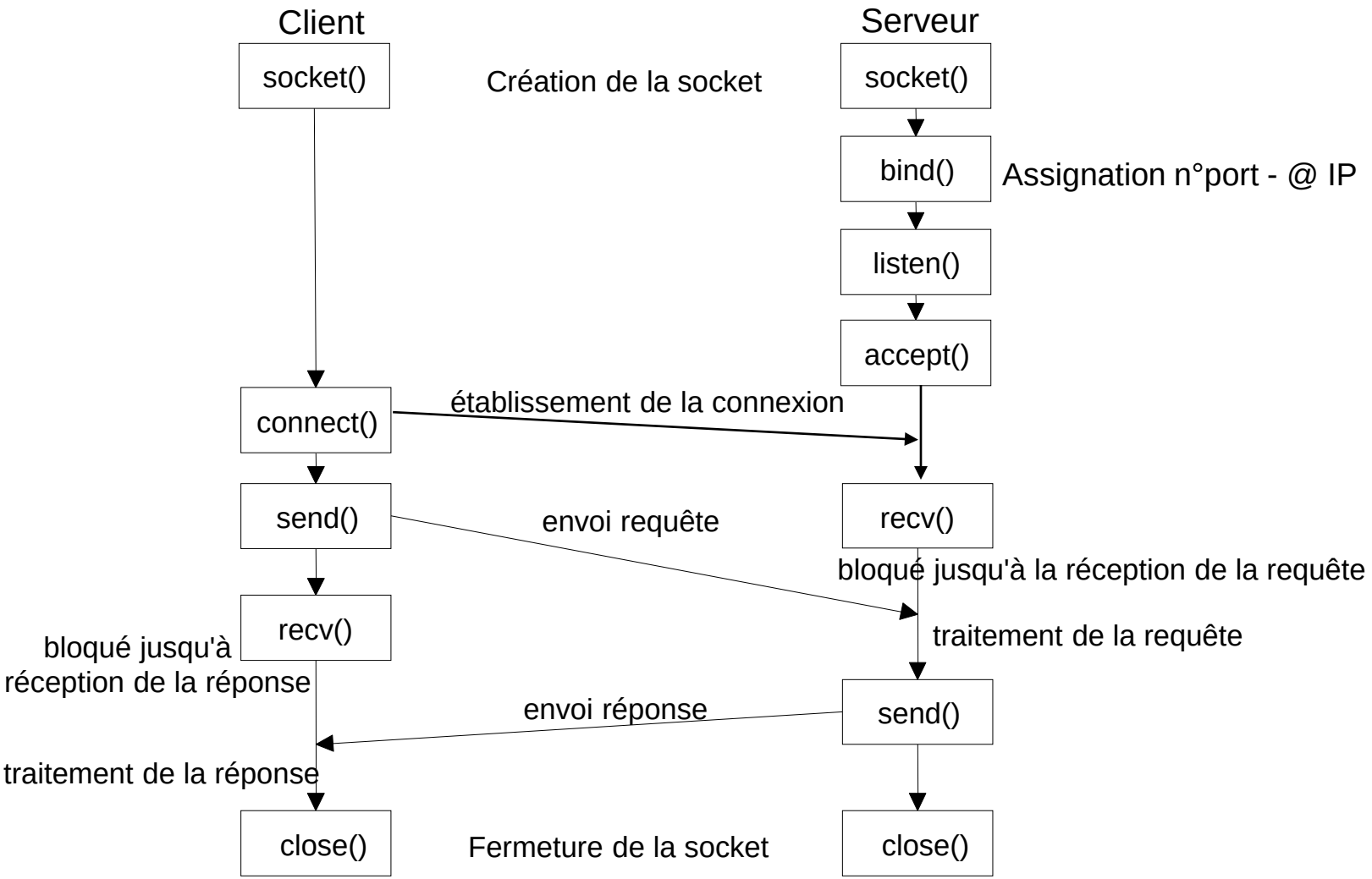
Mode connecté/ non connecté

- Mode connecté (TCP)
  - Problèmes de communication gérés automatiquement
  - Gestion de la *connexion coûteuse en message*
  - Primitives simples d'émission et de réception
  - Pas de délimitation des messages dans le tampon

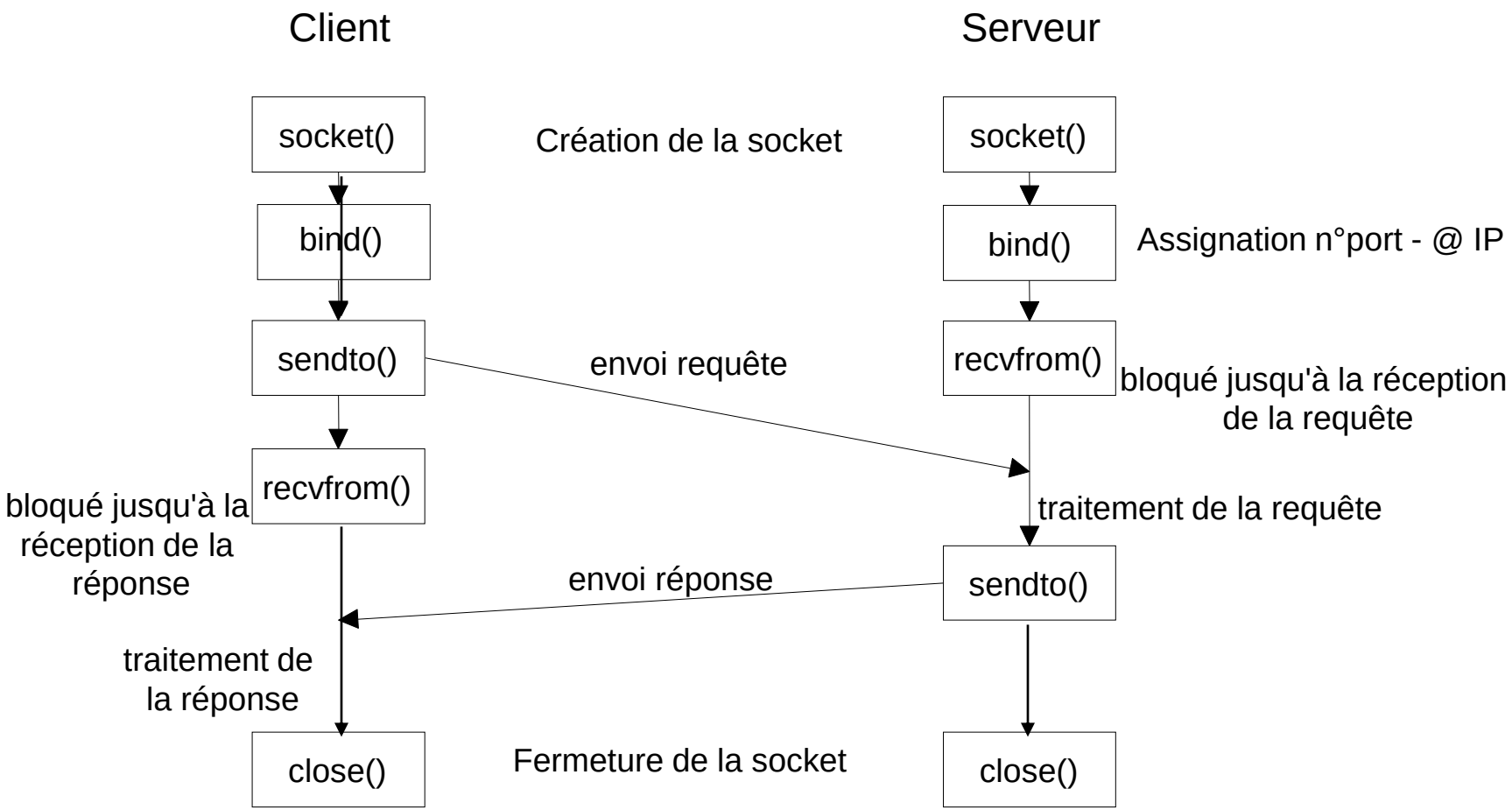
- Mode non connecté (UDP)
  - Consomme moins de ressources
  - Permet la diffusion
  - Aucune gestion des erreurs,

➡ c'est la couche applicative qui doit gérer ce problème

Mode Connecté



Mode non connecté



Généralités (3)

- Une connexion
  - @IP source, @IP destination, port source, port destination

- Une socket est un fichier virtuel sous Unix avec les opérations d'ouverture, fermeture, écriture, lecture …

  (concept légèrement différents sous windows)

- Ces opérations sont des appels systèmes

- Il existe différents types de sockets:
  - Stream socket (connection oriented) ➔ SOCK_STREAM

    mode connecté = TCP
  - Datagram sockets (connectionless) ➔ SOCK_DGRAM

    mode non connecté = UDP
  - Raw sockets ➔ SOCK_RAW

    accès direct au réseau (IP, ICMP)

Programmation en C (1)

- Définition d'une socket
  - **int s = socket (domaine, type, protocole)**
    - *Domaine*
      - AF_UNIX : communication interne à la machine
        structure sockaddr_un dans <sys/un.h>
      - ***AF_INET : communication sur internet en utilisant IP***
    - *Type*
      - ***SOCK_STREAM : mode connecté (TCP)***
      - ***SOCK_DGRAM : mode non connecté (UDP)***
      - SOCK_RAW : utilisation directe niveau 3 (IP, ICMP,…)
    - *Protocole*
      - Actuellement non utilisé, valeur = 0.

Programmation en C (2)

- Attachement d'une socket
  - **int error = bind (int sock, struct sockaddr *p, int lg)**
    - *error :* entier qui contient le compte-rendu de l'instruction
      - 0 : opération correctement déroulée
      - -1 : une erreur est survenue (en général, problème dans p)
    - *sock :* descripteur de la socket
    - *p :* pointeur vers la zone contenant l'adresse de la station
    - *lg :* longueur de la zone p

Fonction définit avec sockaddr, utilisation réelle de sockaddr_in (AF_INET) ou sockaddr_un (AF_UNIX)

Programmation en C (3)

- Adressage (inclure fichier <sys/socket.h> et <netinet/in.h>)
  - **struct sockaddr_in {**
    short sin_family; ← domaine
    u_short sin_port; ← n° port
    struct in_addr sin_addr; } ← @IP système

  - *sin_port* = numéro de port
    - soit laissé libre
    - soit affecté : htons (n°port) (conversion short -> réseau)

  - *Struct in_addr sin_addr*
    - Si serveur sin_addr.s_addr = INADDR_ANY
    - Si client

```
struct hostent *hp;
hp = gethostbyname(" …. ");
bcopy(hp->h_addr, &x.sin_addr, hp->h_length);
```

Programmation en C (4)

Primitives du serveur

- **int error = listen (int sock, int taille)**
  - *error :* entier qui contient le compte-rendu de l'instruction
    - 0 : opération correctement déroulée , -1 erreur
  - *taille* : nombre de requêtes maximum autorisées

→ Permet de créer une file d'attente pour recevoir les demandes de connexion qui n'ont pas encore été prises en compte

- **int scom = accept (int sock, struct sockaddr *p, int *lg)**
  - *struct sockaddr :* stockage de l'adresse appelant
  - *lg* : longueur de l'adresse appelant
  - *scom* : nouvelle socket permettant la communication

→ **Primitive bloquante**

Programmation en C (5)

- Ouverture d'une connexion, côté client
  - **int error = connect (int sock, struct sockaddr *p, int lg)**
    - *error :* 0 : opération correctement déroulée , -1 erreur
    - *struct sockaddr* : adresse et port de la machine distante (serveur)
    - *lg* : taille de l'adresse de la machine distante

- Fermeture d'une connexion
  - **int close (int sock)**

    → le plus utilisé
  - **int shutdown (int sock, int how)**
    - *how* : 0 : réception désactivé,
      1 : émission désactivé,
      2 : socket désactivé

Programmation en C (6)

- Emission de données en mode connecté
  - **int send (int sock, char *buf, int lg, int option)**
    - *option :* 0 rien, MSG_OOB pour les messages urgents
  - **int write (int sock, char *buf, int lg)**
    - utilisable seulement en mode connecté, permet d'écrire dans un descripteur de fichier
- Emission de données en mode non connecté
  - **int sendto (int sock, char *buf, int lg, 0, struct sockaddr *p, int lg_addr)**
    - *p :* contient l'adresse du destinataire
    - *lg_addr :* longueur de l'adresse destinataire

Programmation en C (7)

- Réception de données en mode connecté
  - **int recv (int sock, char *buf, int lg, int option)**
    - *option :* 0 rien, MSG_OOB pour les messages urgents, MSG_PEEK lecture des données sans les retirer de la file d'attente
  - **int read (int sock, char *buf, int lg)**
    - utilisable seulement en mode connecté, renvoie le nombre d'octets réellement lus.
- Reception de données en mode non connecté
  - **int recvfrom (int sock, char *buf, int lg, 0, struct sockaddr *p, int lg_addr)**
    - *p :* contient l'adresse de la source
    - *lg_addr :* longueur de l'adresse source

Programmation en java (1)

- Création d'une socket en mode connecté et connexion
  - **Côté client**
    - *Socket sock = new socket (nom_serveur, n°port)*
      - permet la connexion direct en TCP
      - plus de recherche de nom
  - **Côté serveur**
    - *ServerSocket sock = new ServerSocket (n°port)*
      Socket scom = sock.accept()
      - la communication s'établit avec la socket scom
      - Plus besoin d'attachement, c'est automatique

Programmation en java (2)

- Lecture/ écriture sur des sockets en mode connecté

  plusieurs possibilités

  - **cas possible pour une socket soc**
    - *lecture*

      Reader reader = new InputStreamReader(soc.getInputStream())
      BufferedReader texte = new BufferedReader(reader);
      line = texte.readLine();

    - *écriture*

      Printstream sortie = new PrintStream(sock.getOutputStream());
      sortie.println(line);

Programmation en java (3)

- Création d'une socket en mode non connecté
  - **Côté client**
    - *DatagramSocket sock = new DatagramSocket ()*
  - **Côté serveur**
    - *DatagramSocket sock = new DatagramSocket (n°port)*
  - **Emission/réception**
    - *DatagramPacket msg = new DatagramPacket(buffer, taille, serveur, n°port)*
      sock.sent(msg);
    - *DatagramPacket recu = new DatagramPacket(buffer, taille);*
      sock.receive(recu);
      recu.getAddress() -> adresse de l'expéditeur
      recu.getPort() -> N° port de l'expéditeur
      recu.getData() -> les données.