

# Les sockets java

N. Melab  
melab@lifl.fr

Sockets

DESS-ISIS

## Plan

- Le modèle
- Sockets java
- Sockets C-UNIX BSD 4.X

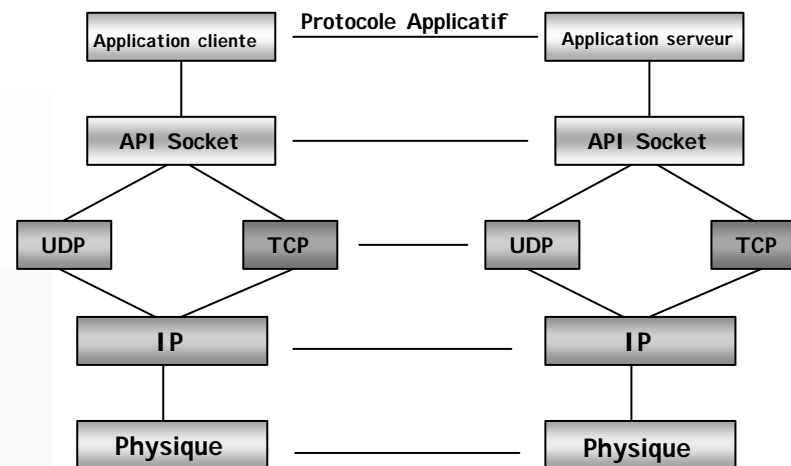
## Le modèle

- Sockets : interface (point de communication) client/serveur utilisée à l'origine dans le monde *UNIX* et *TCP/IP*
- Etendue aux PCs (*Winsock*) et *mainframes*
- Primitives pour le support de communications reposant sur les protocoles (TCP/IP, UDP/IP)
- Les applications client/serveur ne voient les couches de communication qu'à travers l'API socket (abstraction)

Sockets

DESS-ISISIS

## Sockets/OSI



## Rôle des sockets

- Connexion à une machine distante
- Envoi/Réception de données
- Fermeture d'une connexion
- Attachement à un port
- Acceptation d'une demande de connexion à un port local
- Attente de demandes de connexion

Sockets

DESS-ISIS DIS

## Notion de port

- Connexion réseau
  - ☒ Adresse internet de la machine
  - ☒ Numéro du port
- Pourquoi les ports ?
  - ☒ Sur une même machine, plusieurs services sont accessibles *simultanément* (web, email, etc.)
  - ☒ Points d'accès : ports logiques (65535)
  - ☒ Rien à avoir avec les ports physiques (série et parallèle)

## Désignation des ports

- Port : numéro allant de 1 à 65535
- Les ports 1 à 1023 sont réservés aux services courants : finger, ftp, http (80), SMTP (25), etc.
- Fichier d'assignation de ports : /etc/services

Sockets

DESS-ISIS

## Adresse internet

- Connexion réseau
  - ☒ Adresse internet de la machine
  - ☒ Numéro : 193.49.192.193
- Désignation par des noms symboliques
  - ☒ Association de noms symboliques aux adresses numériques
  - ☒ Domain Name System (ou *DNS*)
  - ☒ Exemple : lil.univ-littoral.fr : 193.49.192.193

# Sockets java

Sockets

DESS-ISIS

## Gestion des ports (adresses) en Java

### ■ Classe InetAddress

- ☒ Dans java.net

- ☒ Champs

  - ⇨ hostName (exemple : lil.univ-littoral.fr)

  - ⇨ address (32 bits, exemple : 193.49.192.193)

- ☒ Pas de constructeur

- ☒ 3 méthodes statiques

  - ⇨ public static InetAddress InetAddress .getByName(String hote)

  - ⇨ public static InetAddress[] InetAddress .getAllByName(String hote)

  - ⇨ public static InetAddress InetAddress .getLocalHost()

### ■ Classes Socket, SocketServer et SocketImpl

- ☒ getInetAddress()

# Gestion des sockets

## ■ Taxinomie

### ☒ Sockets TCP

⇔ Point à point : Socket, SocketServer, SocketImplp

### ☒ Sockets UDP

⇔ Point à point : DatagramSocket

⇔ Multi-point : MultiCastSocket

### ☒ Dans java.net

# Sockets TCP

## ■ Classe *Socket*

⇓ Connexion à une machine distante

⇓ Envoi/Réception de données

⇓ Fermeture d'une connexion

## ■ Classe *SocketServer*

⇓ Attachement à un port

⇓ Acceptation d'une demande de connexion à un port local

⇓ Attente de demandes de connexion

## Classe *Socket* (1)

### ■ Constructeurs

- ↳ public **Socket(String hote, int port)** throws  
UnknownHostException, IOException
- ↳ public **Socket(InetAddress hote, int port)** throws IOException
- ↳ public **Socket(String hote, int port, InetAddress interface, int portLocal)** throws IOException
- ↳ public **Socket(InetAddress hote, int port, InetAddress interface, int portLocal)** throws IOException
- ↳ protected **Socket()**
- ↳ protected **Socket(SocketImpl impl)**

Sockets

DESS-ISIS

## Classe *Socket* (2)

### ■ Information

- ↳ public InetAddress **getInetAddress()**
- ↳ public int **getPort()**
- ↳ public int **getLocalPort()**
- ↳ public InetAddress **getLocalAddress()**

### ■ Envoi/Réception de données

- ↳ public InputStream **getInputStream()** throws IOException
- ↳ public OutputStream **getOutputStream()** throws IOException

### ■ Fermeture

- ↳ public **synchronized void close()** throws IOException

## Classe *Socket* (3)

### ■ Options

#### ↳ TCP\_NODELAY

- ↔ Données expédiées aussitôt que possible quelque soit leur taille
- ↔ Méthodes **setTcpNoDelay (boolean valid)** et **setTcpNoDelay()**

#### ↳ SO\_LINGER

- ↔ Attente ou non avant de fermer une socket au cas où il reste des données à envoyer
- ↔ Méthodes **setSoLinger (boolean valid, int secondes)** et **getSoLinger()**

Sockets

DESS-ISIS

## Classe *Socket* (4)

#### ↳ SO\_TIMEOUT

- ↔ Déclenchement d'exception si le délai spécifié est dépassé lors de la réception de données
- ↔ Méthodes **setSoTimeout (int ms)** et **getSoTimeout()**

### ■ Exceptions

#### ↳ *SocketException*

- ↔ *BindException* : port déjà utilisé ou le programme n'a pas le droit de l'utiliser
- ↔ *ConnectException* : hôte occupé ou aucun processus n'écoute sur le port
- ↔ *NoRouteToHostException* : dépassement de temps

## Exercices

- Ecrire un programme qui :
  - ↳ se connecte au service **daytime** d'internet (port numéro 13)
  - ↳ affiche la date et l'heure
- Ecrire un programme qui implémente le service **echo** (port numéro 7) - Utiliser les options des sockets
- Implémenter le protocole **finger** (port numéro 79)

Sockets

DESS-ISIS

## Classe *ServerSocket* (1)

- Rôle : standardiste
- Gestion d'une connexion
  - ↳ Création d'un nouvel objet *ServerSocket* affecté à un port : constructeur *ServerSocket*
  - ↳ Attente de connexion : *accept()*
  - ↳ Echange d'informations : *getInputStream()* et *getOutputStream()*
  - ↳ Clôture de la connexion par le client ou le serveur : *close()*
  - ↳ Nouvelle attente

## Classe *ServerSocket* (2)

- Prise en compte des connexions
  - ↳ Un thread par connexion
  - ↳ Dans **ftp**, un processus par connexion
    - ⇔ Limite : 400 connexions
- Gestion des demandes de connexion
  - ↳ File d'attente gérée par le système
    - ⇔ Taille = 50 par défaut mais peut être modifiée

Sockets

DESS-ISIS

## Classe *ServerSocket* (3)

- Constructeurs
  - ↳ public ***ServerSocket***(int port) throws IOException, BindException
  - ↳ public ***ServerSocket***(int port, int taillefile) throws IOException, BindException
  - ↳ public ***ServerSocket***(int port, int taillefile, InetAddress adresseAttache) throws IOException
  - ↳ protected ***ServerSocket***()
- Prise en compte et clôture d'une connexion
  - ↳ public Socket ***accept***() throws IOException
  - ↳ public void ***close***() throws IOException

## Classe *ServerSocket* (4)

---

### ■ Information

- ↳ public InetAddress **getInetAddress()**
- ↳ public int **getLocalPort()**

### ■ Options

- ↳ SO\_TIMEOUT
  - ⇒ Doit être initialisé avant *accept*
  - ⇒ Méthodes **setSoTimeout (int ms)** et **getSoTimeout()**

## Exercice

---

### ■ Programmer le service *daytime* d'internet

- ↳ Ne pas utiliser le port numéro 13 sauf si vous avez les droits super-utilisateur
- ↳ Utiliser un numéro de port supérieur à 1024

## Protocoles TCP et UDP

### ■ TCP

- ☒ Garantie d'arrivée dans l'ordre de paquets de données
- ☒ Fiabilité de transmission
- ☒ Lenteur des transmissions (http par exemple)
- ☒ Vu comme un «service téléphonique»

### ■ UDP

- ☒ Arrivée dans le bon ordre non garantie
- ☒ Non fiabilité des transmissions
- ☒ Rapidité des transmissions
- ☒ Applications audio/vidéo
- ☒ Vu comme un «service postal»

## Sockets UDP point-à-point

- Pas de différence de classe entre le client et le serveur
- Deux classes Java dans *java.net*
  - ☒ *DatagramPacket*
  - ☒ *DatagramSocket*
- *DatagramPacket*
  - ☒ Assemblage des données en partance en *datagrammes*
  - ☒ Extraction des données des *datagrammes* reçus
- *DatagramSocket*
  - ☒ Envoi et réception des *datagrammes* UDP (objets

# Classe *DatagramPacket* (1)

## ■ Constructeurs

☒ *DatagramPacket* destiné à la réception de données

☞ `public DatagramPacket(byte tampon[], int longueur)`

☞ Réception des données dans *tampon*

☞ Longueur maximale : 65 507 octets

☞ Java remplit les champs de *DatagramPacket* : adresse IP de la machine distante et le numéro de port concerné

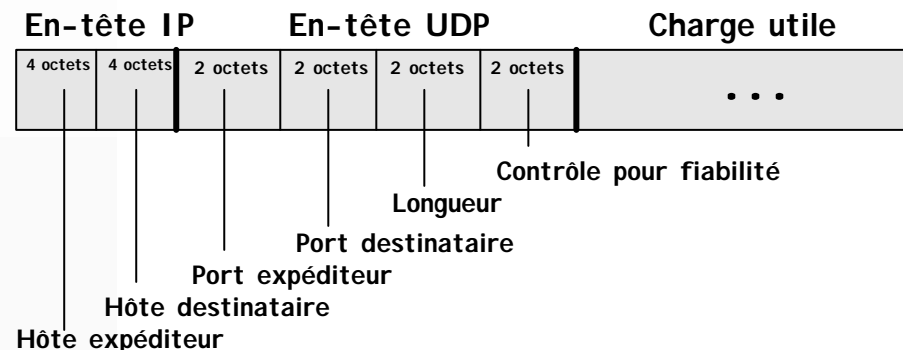
☒ *DatagramPacket* destiné à l'envoi de données

☞ `public DatagramPacket(byte tampon[], int longueur, InetAddress ia, int port)`

☞ Conseil : utiliser *getBytes()* de la classe *String* pour transformer des chaînes de caractères en tableaux d'octets

# Classe *DatagramPacket* (2)

## Structure d'un datagramme



## Classe *DatagramPacket* (3)

### ■ Information

✉ public int *getPort()*

- ☞ Numéro de port de provenance si datagramme reçu
- ☞ Numéro de port de destination si datagramme créé localement

✉ public InetAddress *getAddress()*

- ☞ Adresse du hôte distant si datagramme reçu
- ☞ Adresse du hôte local si datagramme créé localement

## Classe *DatagramPacket* (4)

### ■ Information (suite)

✉ public byte[] *getData()*

- ☞ Tableau d'octets du datagramme
- ☞ Caractères ASCII : String s=new **String**(dp.getData(), 0, 0, dp.getLength());
- ☞ Caractères non ASCII : ByteArrayInputStream b=new ByteArrayInputStream(dp.getData(), 0, 0, dp.getLength());

✉ public int *getLength()*

- ☞ Taille en octets du datagramme

## Classe *DatagramSocket* (1)

### ■ Constructeurs

- ☒ *DatagramSocket* utilisé par le client
  - ☞ public ***DatagramSocket()*** throws SocketException
  - ☞ Ports de destination et de partance spécifiés dans le datagramme (objet *DatagramPacket*)
- ☒ *DatagramSocket* utilisé par le serveur
  - ☞ public ***DatagramSocket(int port)*** throws SocketException
  - ☞ Remarque : les ports TCP et leurs équivalents UDP ne sont pas liés i.e. un même numéro de port peut être associé à la fois à une socket TCP et une socket UDP sans provoquer aucun problème de conflit
- ☒ *DatagramSocket* utilisé par le serveur multi-adresse
  - ☞ public ***DatagramSocket(int port, InetAddress ia)*** throws SocketException

## Classe *DatagramSocket* (2)

### ■ Emission/Réception de datagrammes

- ☒ public void ***send(DatagramPacket dp)*** throws IOException
- ☒ public void ***receive(DatagramPacket dp)*** throws IOException

### ■ Information

- ☒ public int ***getLocalPort()***
- ☒ Numéro de port (anonyme et assigné par le système) sur lequel la socket courante écoute

### ■ Fermeture

- ☒ public void ***close()***

### ■ Option : SO\_TIMEOUT

- ☒ public synchronized void ***setSoTimeout(int timeout)*** throws SocketException

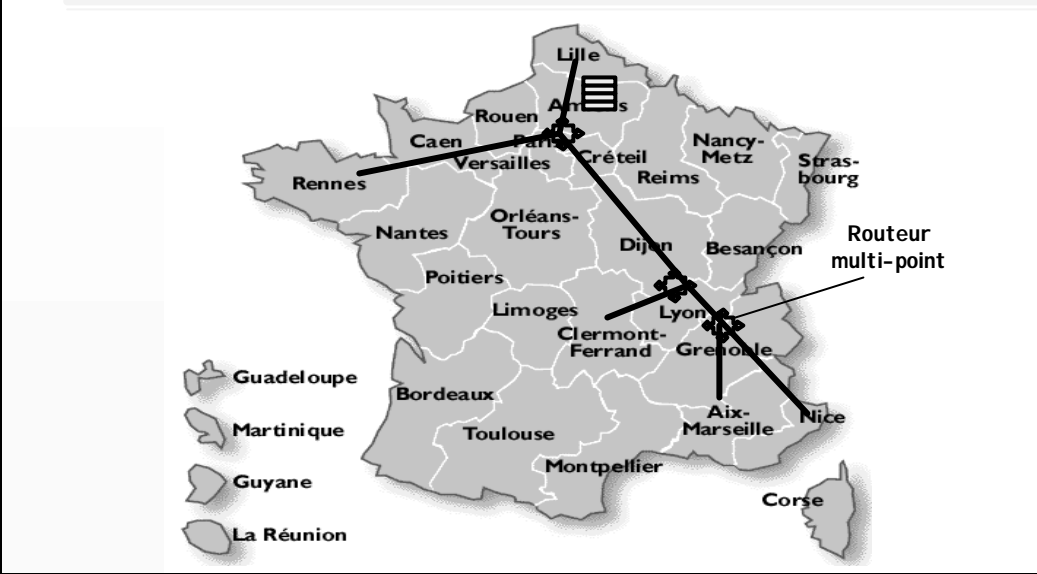
# Exercices

- Ecrire un client UDP générique
  - ⇔ Une classe qui permet au programmeur d'utiliser les sockets UDP sans avoir à manipuler les datagrammes (pas de DatagramPacket)
- Ecrire un client UDP *daytime*
- Ecrire une application **echo** client/serveur UDP multi-threadée

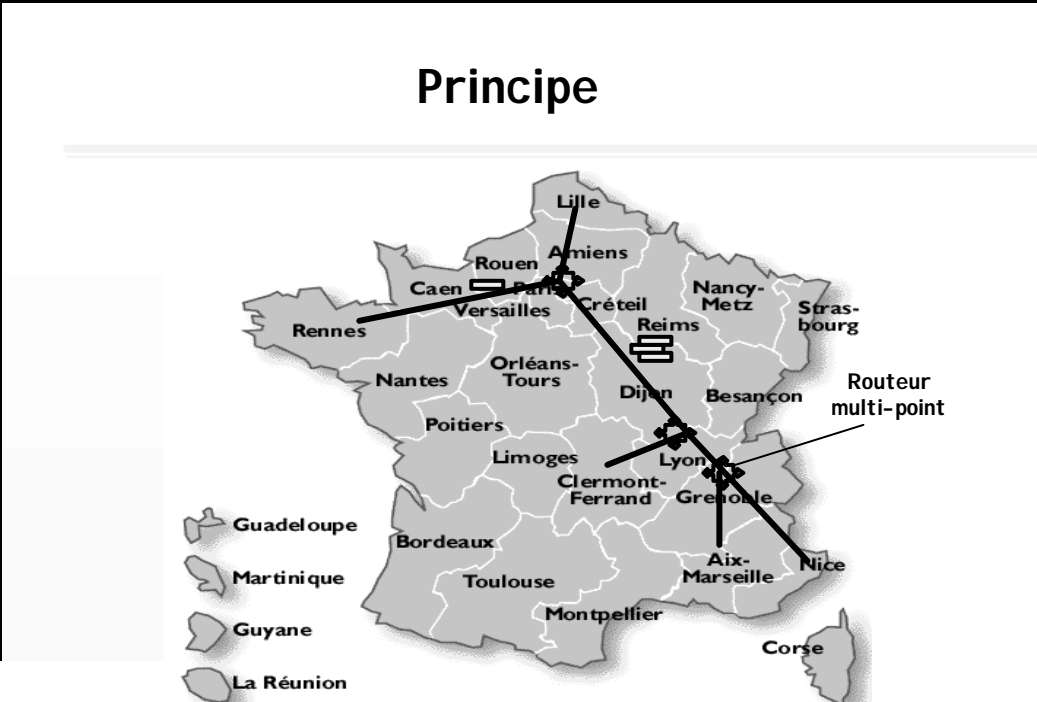
# Sockets multi-point

- Communication multi-point : on s'adresse à un groupe qu'on peut intégrer et quitter à sa guise
  - ☒ Exemples : visio-conférence, radio, télévision, etc.
- Multi-point = compromis
  - ☒ Multi-diffusion
  - ☒ Point-à-point
- Caractéristiques
  - ☒ Efficace car données dupliquées au moment opportun
  - ☒ Basée sur l'utilisation de routeurs multi-point
- Applications
  - ☒ Jeux réseau multi-utilisateurs, SGF distribués, calcul parallèle & distribué, etc.

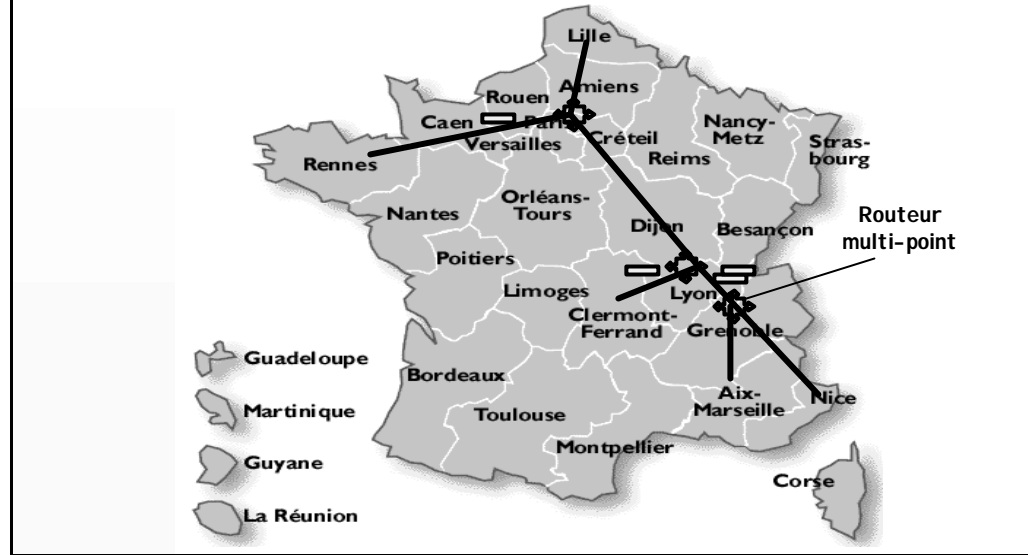
# Principe



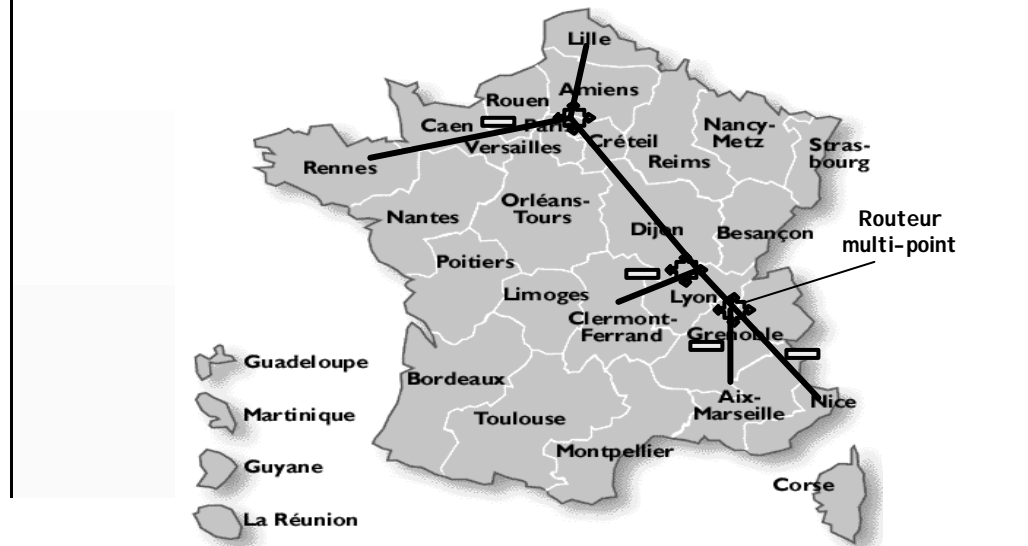
# Principe



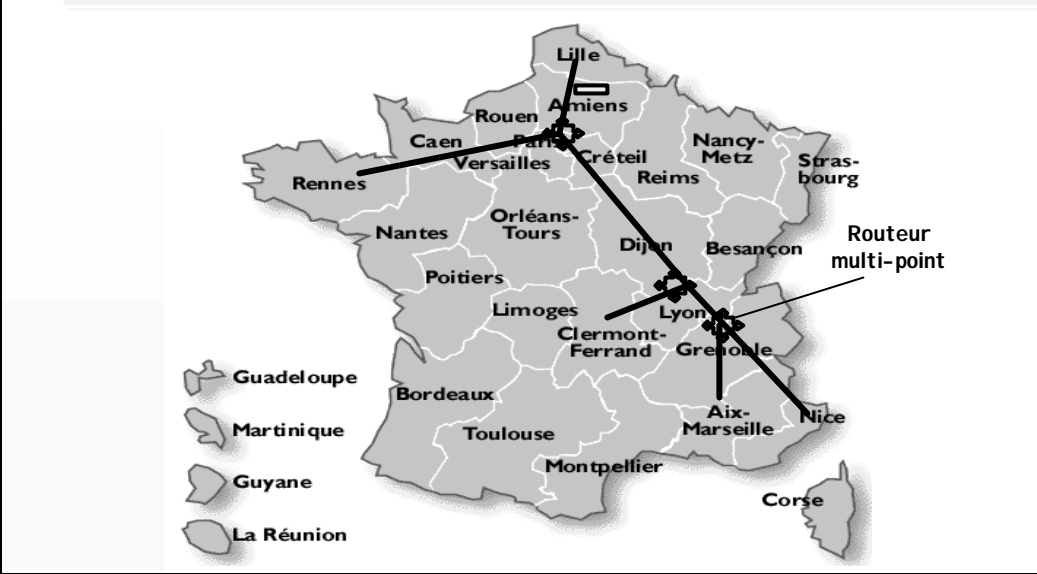
# Principe



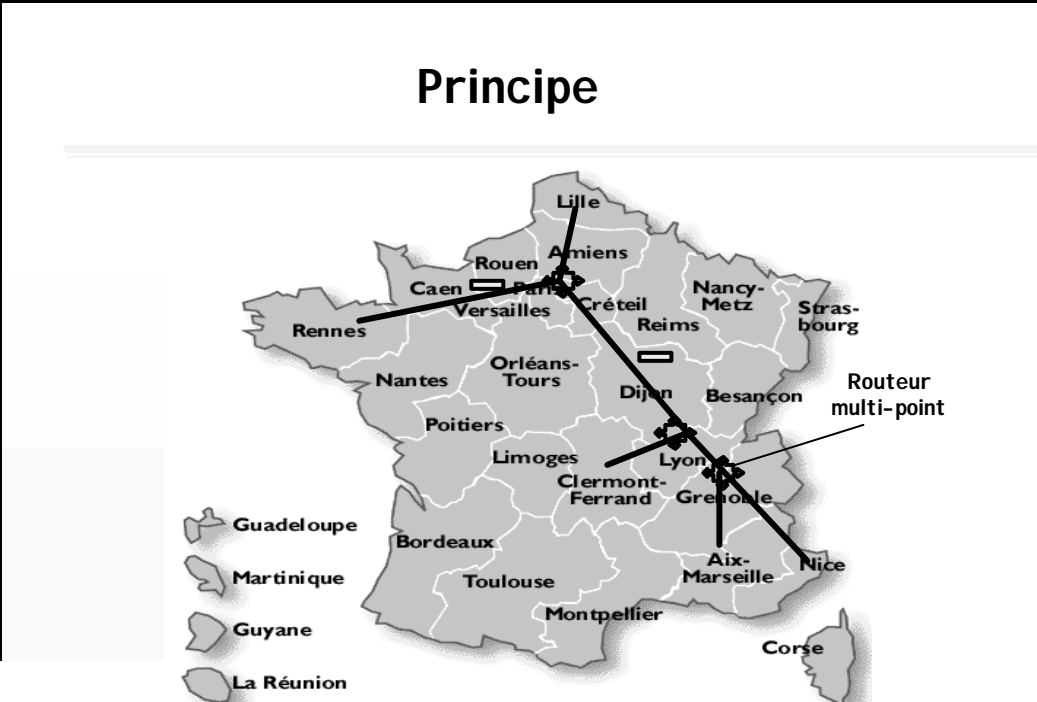
# Principe



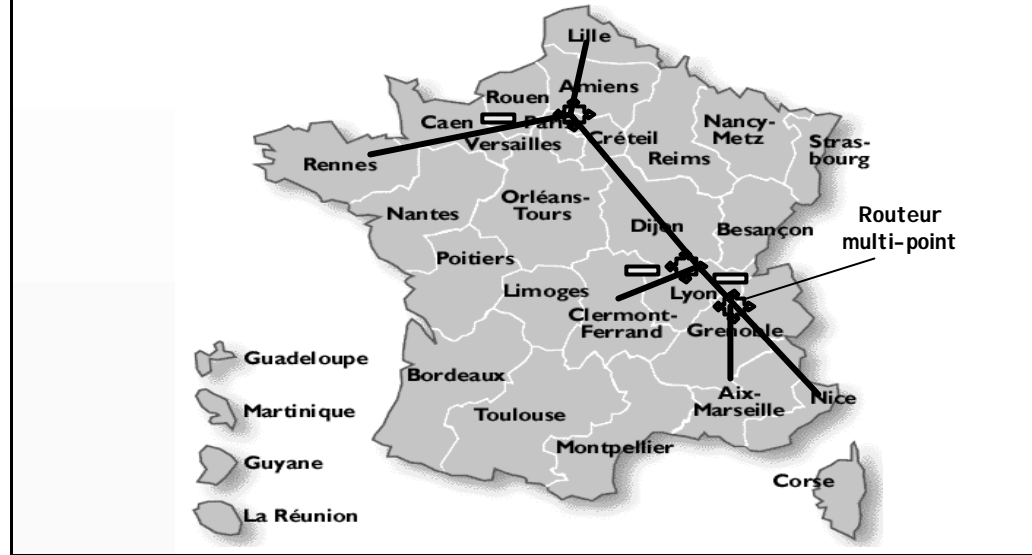
# Principe



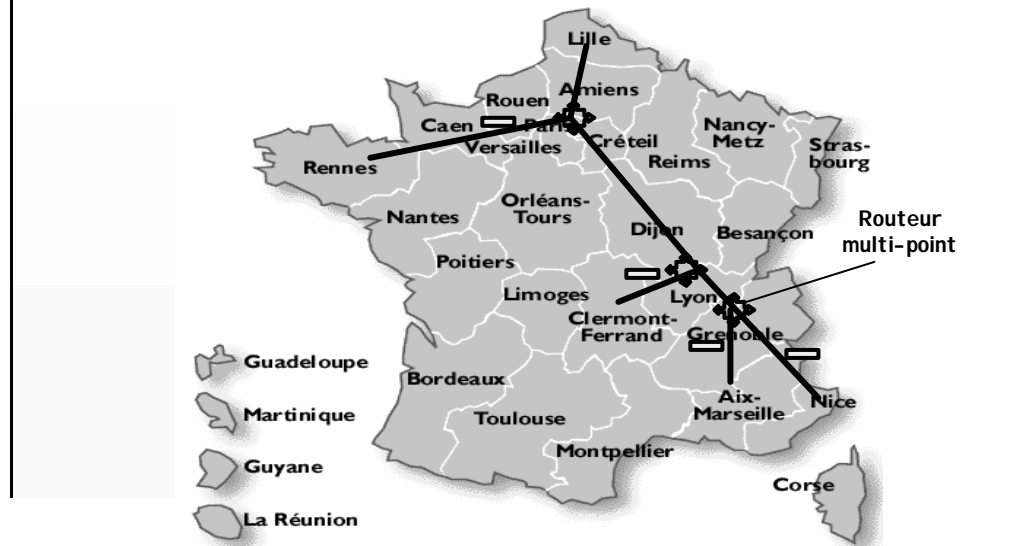
# Principe



# Principe



# Principe



# Adresses et groupes multi-point (1)

- Adresse multi-point
  - ↳ Référence un groupe d'hôtes (groupe multi-point)
  - ↳ Rangée dans la classe D (4 premiers bits : 1110)
- Groupe multi-point
  - ↳ Plusieurs hôtes Internet partageant la même adresse multi-point
  - ↳ Libre adhésion ou départ au/du groupe
- *Internet Assigned Number Authority (IANA)* est responsable de l'affectation des adresses multi-point
  - ↳ Il en existe 80
  - ↳ Exemple : AUDIO NEWS.MCAST.NET --> 224.0.1.7 - Actualité audio transmise en multi-point

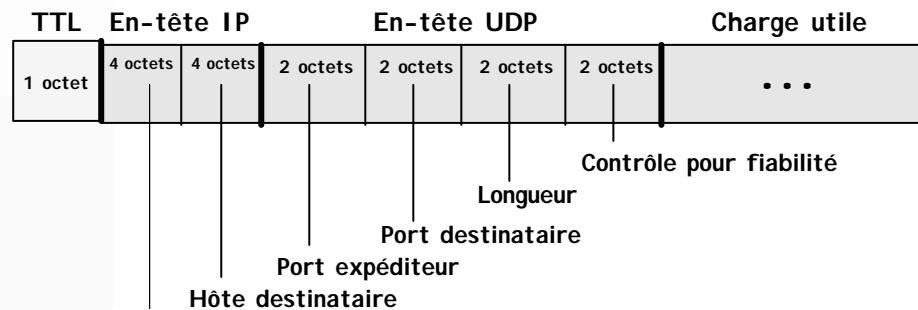
Sockets

DESS-ISIDIS

# Adresses et groupes multi-point (2)

↳ Liste exhaustive dans :  
<http://www.iana.org/assignments/multicast-addresses>

- Il existe des routeurs multi-point (*mrouteurs*)
- Datagramme multi-point



## *Time-To-Live (TTL)*

---

- Définition

- ↳ Portée d'un datagramme i.e. nombre de routeurs maximum par lesquels un datagramme transite avant sa suppression
- ↳ Réseau local : TTL=1, Région : TTL=16, Monde entier : TTL=127

- Utilisation

- ☒ TTL décrémente à chaque passage dans un routeur
- ☒ TTL=0 : datagramme supprimé

## Mise en œuvre des sockets multi-point

---

- Classe *java.net.MulticastSocket*

- ↳ Dérivée de la classe *java.net.DatagramSocket*

- Constructeurs

- ↳ **public MulticastSocket()** throws SocketException
  - ⇒ Port assigné par le système
  - ⇒ `getLocalPort()` : renvoie le numéro du port
- ↳ **public MulticastSocket(int port)** throws SocketException

## Communication avec un groupe multi-point (1)

---

### ■ Opérations possibles

- ↳ Adhésion à un groupe
- ↳ Echanger (Emettre/Recevoir) des données avec les membres d'un groupe
- ↳ Quitter un groupe

### ■ Méthodes

- ↳ public void **joinGroup(InetAddress adrmultipt)** throws SocketException
- ↳ public void **leaveGroup(InetAddress adrmultipt)** throws SocketException

## Communication avec un groupe multi-point (2)

---

- ↳ Public synchronized void **send(DatagramPacket dp, byte ttl)** throws IOException, SocketException
- ↳ public void **setInterface(InetAddress interface)** throws SocketException
- ↳ public **InetAddress getInterface()** throws SocketException

## Exercices

---

- Programmer une taupe multi-point
  - ↳ Adhésion à un groupe multi-point
  - ↳ Boucle sur réception de données envoyées au groupe
- Ecrire un expéditeur de données multi-point
  - ↳ Adhésion à un groupe multi-point
  - ↳ Envoi de données aux membres du groupe
  - ↳ Sortie du groupe