Lille iut
Département Informatique

Programmation répartie

Programmation clients / serveurs UDP

Durée : 1 séance
M4102 -N4P2
2014–2015

Rappel : Toujours avoir un accès à l'API sous la main : Utilisez la version hébergée localement pour éviter les problèmes de liaison :
- Version locale : localhost (puis section Documentation Java ou API java, en bas de la page)
- Version web : java 7 documentation

Vous avez dans cette documentation toutes les informations nécessaires à la réalisation de vos programmes pour nos séances de TP.

Résumé

Le but de ce TP est la mise en œuvre d'une communication distante via un transport en mode non connecté et asynchrone. Dans ce mode de communication, le paquet de données est appelé **datagramme**. Chaque datagramme est transporté indépendamment des autres. C'est donc l'application qui a la charge de découper son flot de données pour l'adapter au transport en réseau et d'indiquer à chaque datagramme sa destination.

Principes généraux

Socket

Dans une pile de communication standard TCP/IP, un *socket* est l'interface de communication logicielle inter-application. Il permet l'échange bidirectionnel de données entre deux applications pouvant se trouver sur la même machine ou pas (à travers le réseau).

En java, nous pouvons considérer qu'un socket est caractérisé par deux éléments : une adresse IP et un numéro qui sera lié à l'application souhaitant communiquer en réseau. Ce numéro s'appelle le port (logiciel).

En java, l'objet java.net.InetSocketAddress permet d'associer au sein d'un seul objet ces deux caractéristiques. Attention, cela ne représente pas un socket. C'est simplement un identifiant permettant de simplifier les manipulations.

L'API Java reconnaît deux types de socket. L'un pour les communications orientées sans connexion (que nous allons voir dans ce TP) et l'autre avec connexion.

Communication orientée sans connexion

Le mode de communication dit *sans connexion* est le plus simple qui soit. Il n'y a pas de gestion des pertes de datagramme. La seule garantie apportée par ce mode de communication est que lorsque les données transférées sont reçues par l'application de destination, alors ces données peuvent être considérées comme valides (le datagramme a été vérifié à l'aide d'une somme de contrôle sur le pseudo en-tête UDP).

De même, lors de l'envoi de plusieurs datagrammes consécutifs, rien ne garantie que l'ordre d'arrivée sera identique à celui de départ.

Pour la programmation d'applications client/serveur en UDP, Java utilise deux classes du paquetage java.net :
- `java.net.DatagramPacket` : pour la conception des datagrammes (gestion des données, de l'émetteur et de la destination).
- `java.net.DatagramSocket` pour la gestion de la communication proprement dite (envoie et réception des datagrammes).
- `java.net.MulticastSocket` pour la gestion de la communication en multi-diffusion (adresse de classe D en IPv4 par exemple).

Une communication entre deux applications sera donc réalisée de la manière suivante :
- Créer un socket UDP assurant la réservation d'un port de communication et sa liaison à l'adresse IP locale concernée ;
- Configurer la connexion (**Attention**, il n'y a pas de connexion en UDP) ;
- Effectuer l'envoi et la réception des datagrammes via ce socket.
- Pour la **réception** d'un datagramme :
  Un espace mémoire permettant le stockage des données reçues. *Attention* : Comme pour le flot de données en TCP, Java ne connait pas le type des données transmises. Ces données seront donc à placer dans un *tableau d'octets* (forme de données la plus générique en Java).
- Pour l'**envoie** d'un datagramme :
  - Les données utilisateurs sous les même conditions que pour la réception (tableau d'octets) ;
  - L'identification du socket de la destination (IP et port).

Exercice 1 : Les interfaces de communication réseau

Q 1 Quelle est la volume maximal théorique exacte de données contenues dans un datagramme UDP (justifier) ?
Q 2 Écrivez un programme simple qui utilise une classe du paquetage java.net pour retrouver la liste des adresses IP associées à chacune des interfaces réseau active de votre machine, ainsi que la taille de la MTU [1] qui y est associée.

Exercice 2 : Démarrage en douceur

Q 1 Voici le code d'un serveur UDP. Recopiez ce code et testez le avec la commande netcat. Que fait ce service ?

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

public class ServeurUDP {
  private DatagramSocket dgSocket;

  ServeurUDP(int pSrv) throws IOException {
    dgSocket = new DatagramSocket(pSrv);
  }

  void go() throws IOException {
    DatagramPacket dgPacket = new DatagramPacket(new byte[0], 0);
    while( true ) {
      dgSocket.receive(dgPacket);
      System.out.println("Datagram␣received␣from␣" + dgPacket.getSocketAddress());
      dgPacket.setSocketAddress(dgPacket.getSocketAddress());
      String str = new java.util.Date().toString() + "\n";
      byte[] buf = str.getBytes();
      dgPacket.setData(buf, 0, buf.length);

      dgSocket.send(dgPacket);
    }
  }

  public static void main(String[] args) throws IOException {
    new ServeurUDP(9876).go();
  }
}
```

Q 2 Réalisez un programme ClientUDP.java *"se connectant"* à ce serveur et affichant en console les informations reçues. L'adresse et le numéro de port du serveur pourront être entrée en ligne de commande.

Exercice 3 : Echo multiplicateur UDP

Q 1 Il s'agit de la réalisation d'un client et d'un serveur simples avec des communications en mode Datagramme. Le client envoie une chaîne de caractères au serveur. Le premier caractère de cette chaîne sera un chiffre et les caractères suivants formeront une phrase simple (sans ponctuation) et courte.

Le serveur affiche localement les informations sur le datagramme reçu (adresse :port émetteur, données), et renvoie à l'émetteur une chaîne de caractères dont tous les mots sont écrit n fois, n étant le chiffre se trouvant en premier dans la chaîne reçue. Si la chaîne reçue ne possède pas un chiffre en premier caractère, alors un message d'erreur est envoyé.

Exemple :

Console *client* sur frene05	Console *Serveur*
	> java ServeurEcho_n 5555 Serveur Echo_n prêt.
> java Echo_n frene02 5555 "2Bonjour le monde"	Reçu de frene05.iut-info.univ-lille.fr51754 Multiplicateur : 2, Phrase : "Bonjour le monde".
Résultat : "Bonjour Bonjour le le monde monde"	
> java Echo_n frene02 5555 "Salut coco"	Reçu de frene05.iut-info.univ-lille.fr51754 Multiplicateur : aucun, Phrase : "Salut coco".
Erreur : "Salut coco" n'est pas correcte.	

Remarque : Vérifiez bien que le chiffre zéro est un élément absorbant : Le résultat de "0Bonsoir tous le monde" doit être une chaîne vide.

Q 2 La chaîne envoyée aura maintenant le format suivant : un nombre (et non plus un chiffre) puis une phrase simple séparée par ":" (exemple : "56:blablabla"). Transformez votre serveur afin d'obtenir le même résultat que précédemment avec le nouveau format de chaîne de caractères.

Q 3 Que se passe-t-il si votre message est "4000 :Bonjour le monde" ? Proposez une solution.

1. Maximum Transfert Unit : volume maximal de données transportable dans une trame

Exercice 4 : Encore un chat !

Mais celui-là sera en UDP et en multi-diffusion.

Si on souhaite envoyer un même message à un grand nombre de personnes ne se trouvant pas obligatoirement sur le même réseau, il est préférable de diffuser ce message en utilisant un protocole de diffusion multi-points qui prendra en charge la transmission vers les destinataires. Ce type de protocole est associé à un routage multi-destinataire, plus efficace que le simple routage point à point et l'engorgement des réseaux s'en trouve ainsi réduit.

En pratique, on utilise les adresse IP de classe D comme une adresse d'abonnement à une liste de diffusion. Ainsi, toutes machines ayant un accès réseau pourra envoyer un message à destination de tous les abonnés à cette adresse de classe D. De plus, toutes machines ayant effectuée un abonnement sur une adresse IP de classe D pourra recevoir les messages envoyés vers cette adresse. Seul le serveur multi-cast (le routeur en général) connaît donc la liste des abonnés à une adresse de classe D.

Il existe plusieurs types d'adresse IP de classe D. Certaines ne peuvent pas traverser les routeurs (à la manière des adrsse privées en classe A, B et C). Un routeur connaissant des protocoles de diffusion multi-destinataires (multicast), s'il est normalement configuré ne transmet pas les adresses entre 224.0.0.0 à 224.0.0.255 inclus. Ainsi, vous pouvez à loisir utiliser ces adresses pour vos expérimentations, c'est ce que vous ferez dans ce TP.

Mise en oeuvre en Java

En java, c'est la classe `java.net.MulticastSocket` qui étend les possibilités de la classe `DatagramSocket` au multicast en ajoutant les méthodes d'abonnements. La gestion des messages se fera donc par le protocole UDP et la classe `DatagramSocket` dont elle reprend toutes les méthodes.

Les principales opérations supplémentaires sont donc :

- Adhérer à un groupe
  `joinGroup(InetAddress) throws IOException`
- Quitter un groupe
  `leaveGroup(InetAddress) throws SocketException`

Remarque : Afin de limiter la zone atteinte par un message multidestinataire, il est bon de régler le TTL de ces messages à une valeur pas trop élevée. Ce dernier représente le nombre de routeurs que le message peut traverser avant d'être abandonné faute d'avoir atteint sa destination (0 pour l'émetteur, 1 pour le (sous-)réseaux local, 2 pour le suivant ...).

Q 1 Rechercher sur le site de l'IANA, la liste des adresses multicasts réservée à l'entreprise Walt Disney.

.....

Q 2 En utilisant un socket multicast, écrivez deux classes :

- l'une permettant de lire les messages d'un groupe multicast donné en argument (couple adresse IP / port) et de les afficher sur la console

- l'autre permettant à l'utilisateur de rentrer des petites lignes de texte et qui les diffuse sur le groupe multicast donné en argument (couple adresse IP / port).

Exemples :

Fenêtre affichage	Fenêtre saisie
> java ReceptionMsg 224.0.0.24 9876	> java DiffuseMsg 224.0.0.24 9876
	Bonjour, je suis l'afficheur
`Phrase :` *Message en une ligne*	frene05 dit : Message en une ligne
`Phrase :` *FIN*	

Testez vos deux programmes ensembles sur le même groupe multicast que vous choisirez parmi les adresses non assignées (c.f. IANA).