# “IPortbook”
# Projet de Programmation Réseaux

**License Informatique - Université Paris Diderot**

**2017**

*Attention :* Prenez le temps de lire attentivement TOUT le document et ce dans ses moindres détails. Relisez plusieurs fois, même en groupe. Ne vous lancez pas immédiatement dans la réalisation à moins que vous ne soyez déjà à l'aise avec le développement réseau et la communication inter-applications. Faites donc des diagrammes, imaginez des scénarios de communication entre entités, essayez d'en analyser la portée, les problèmes, les informations dont les entités ont besoin au cours de leur vie, etc. Essayez d'abord de vous abstraire des problèmes techniques.

*Note :* Dans le document le signe ␣ représentera un simple caractère d'espacement (ASCII 32). De plus les messages circulant sont indiqués entre crochets, **les crochets ne faisant pas partie du message**. À la fin du document le format des parties constituant les messages est décrit de façon précise.

## Introduction

Le but de ce projet est de programmer un système servant à différentes entités clients d'établir des relations et de pouvoir communiquer. Le fonctionnement est inspiré des réseaux sociaux actuels.

Il y a trois types d'entités dans ce système :

1. Les serveurs
2. Les clients
3. Les promoteurs

Intuitivement les clients peuvent s'enregistrer sur le serveur, ils peuvent établir des liens d'amitiés et s'échanger des messages. Les serveur a la liste de tous ses clients ainsi que leur lien d'amitiés, il a pour rôle d'envoyer des notifications aux clients, transmettre les messages entre clients et diffuser les annonces des promoteurs.

Le fonctionnement est le suivant, le serveur gère pour chacun de ses clients une liste de flux, un flux peut être une demande d'amitiés, un message d'un autre client, une réponse à une demande d'amitiés, un message d'inondations, une publicité. À chaque nouveau flux créé, le serveur envoie une notification sous forme de message UDP au client concerné codé sur 4 octets et de la forme `[YXX]` où `Y` est un code pour le flux concerné et `XX` est le codage en *litte-endian* du nombre de flux non consultés. Le client connecté peut consulter ses flux grâce à la commande `CONSU`.

Le but de ce projet sera de programmer ces entités en respectant le protocole correspondant et en proposant des extensions.

## Description détaillée du protocole

### Les clients

Les caractéristiques d'un client sont les suivantes :

— Un identifiant (d'au plus 8 caractères alphanumérique)
— Un port d'écoute UDP pour recevoir les notifications (inférieur à 9999)
— Un mot de passe (un nombre compris entre 0 et 65 535)

Lorsqu'un client est nouveau sur un serveur, il s'enregistrera en donnant ses informations. Le port UDP sert à recevoir des notifications pour les nouveaux flux arrivés (nouveaux messages, nouvelle demande d'amitiés, annonce d'un promoteur).

## Les serveurs

Les caractéristiques d'un serveur sont les suivantes :

— Un port d'écoute TCP pour communiquer avec les clients (inférieur à 9999)
— Un port d'écoute TCP pour communiquer avec les promoteurs (inférieur à 9999)

Le port TCP du serveur sert pour communiquer avec les clients.

## Les promoteurs

Les caractéristiques d'un promoteur sont les suivantes :

— Une adresse IP de multi-diffusion
— Un port de multi-diffusion (inférieur à 9999)

Un promoteur communique avec le serveur pour lui dire d'envoyer de la publicité aux clients pour ses newsletters qu'il multi-diffuse.

## Interaction client-serveur

**S'inscrire.** Pour s'inscrire à un serveur, un client se connecte au serveur et lui envoie un message de la forme [`REGIS␣id␣port␣mdp+++`] où `id` est l'identité du client , `port` est son port UDP et `mdp` son mot de passe. Le serveur lui répond soit par un message [`WELCO+++`] si il accepte que ce client soit un nouveau client, soit par un message [`GOBYE+++`] si il refuse (à vous de voir pourquoi un serveur pourrait refuser un client). On limitera le nombre de clients sur un serveur à 100. Si il accepte le client, celui-ci peut continuer à communiquer dans le cas contraire, le serveur ferme la connexion. Lorsqu'un serveur accepte le client il stocke bien entendu son IP et ces caractéristiques pour pouvoir communiquer avec lui. Un client accepté peut ensuite faire les taches suivantes : obtenir la liste des clients inscrits, faire des demandes d'amitiés, envoyer des messages à ses amis, envoyer des messages d'inondation et consulter ses flux.

**Se connecter.** Lorsqu'un client accepté par un serveur s'est déconnecté et veut se reconnecter, il se connecte en TCP et il envoie le message [`CONNE␣id␣mdp+++`] avec son identité `id` et son mot de passe `mdp`. Si le serveur le reconnaît, il l'accepte en lui envoyant un message [`HELLO+++`] ou le refuse en lui envoyant un message [`GOBYE+++`] et en fermant la connexion.

**Les demandes d'amitiés.** Un client peut demander à être ami avec un autre client. Pour cela, il envoie le message suivant : [`FRIE?␣id+++`] où `id` est l'identité du client avec lequel il souhaite être ami. Le serveur lui répond alors par le message [`FRIE>+++`] pour dire qu'il a transmis sous forme de flux sa demande d'amitiés ou bien [`FRIE<+++`] si il ne connaît pas le client `id`. Dans le premier cas, le serveur envoie une notification sur le port UDP du client `id` sous la forme d'un message [`0XX`] et il ajoute au flux de `id` la demande d'amitiés. Si plus tard `id` accepte la demande d'amitiés, le serveur enverra un message de la forme [`1XX`] au client ayant fait la demande et de la forme [`2XX`] si `id` refuse et dans les deux cas il ajoutera la réponse aux flux du client demandeur.

**Les envois de messages.** Un client peut envoyer un message à un ami. Pour ce faire il envoie [MESS?␣id ␣num-mess+++] où id correspond à l'identité de l'ami avec qui on souhaite communiquer et num-mess est le nombre de messages partiels envoyé qui correspondront au message global, il envoie ensuite num-mess messages [MENUM␣num␣mess+++] numéroté de 0 à num-mess-1 où mess correspond à une partie du message global et est une chaîne de caractères d'au plus 200 caractères. Ainsi pour envoyer un message de 1100 caractères, le client devra en réalité envoyé 6 messages MENUM. Quand le serveur a reçu le dernier message, il répond en envoyant [MESS>+++] pour signaler qu'il a transmis les messages à id ou [MESS<+++] pour signaler qu'il n'a pas transmis ce message, une raison pouvant être que id n'est pas ami avec le client émetteur du message (Dans le premier cas, le message est donc ajouté à la liste des flux à consulter par id). Lorsqu'un client reçoit un tel message, le serveur lui envoie sur son port UDP une notification de la forme [3XX] . Ainsi le client pourra consulter ce message lorsqu'ils consultera ses notifications. (Le serveur stocke donc sous forme de flux pour chaque client les messages qui lui sont envoyés jusqu'à ce que celui-ci demande à les lire).

**Inondation.** Un client peut envoyer un message à tous ses amis et aux amis de ses amis, et aux amis des amis de ses amis etc. Pour cela le client envoie un message [FLOO?␣mess+++] où mess est le message envoyé d'au plus 200 caractères. Notons que dans ce cas les messages d'inondation ne sont jamais longs. Le serveur répond alors [FLOO>+++] pour dire qu'il a transmis le message. Pour les clients concernés, le serveur leur envoie sur leur port UDP une notification de la forme [4XX] et il ajoute aux flux de tous ces clients le message d'inondation.

**La liste des clients.** Un client peut demander à voir la liste des clients d'un serveur en envoyant une requête de la forme [LIST?+++] à laquelle le serveur répond en envoyant tout d'abord un message [RLIST␣ num-item+++] où num-item est le nombre de clients dans la liste. Il envoie ensuite num-item messages de la forme [LINUM␣id+++] avec l'id de chacun des clients.

**Consultation.** Un client peut demander à consulter la liste de ses flux. Pour cela il envoie un message de la forme [CONSU+++] qui lui permet de consulter (et donc de retirer de la liste un des flux). Le serveur lui répond alors de la façon suivante selon les cas :

- Si il s'agit d'un flux correspondant au message d'un client, le serveur envoie [SSEM>␣id␣num-mess+++] où id est l'identité du client ayant envoyé le message et num-mess est le nombre de messages partiels envoyés qui correspondront au message global, il envoie ensuite num-mess messages [MUNEM␣num ␣mess+++] numérotés de 0 à num-mess-1 où mess correspond à une partie du message global.
- Si il s'agit d'un flux correspondant à un message d'inondation, le serveur envoie [OOLF>␣id␣mess+++] où id est l'identité du client ayant envoyé le message et mess est une chaîne de caractères d'au plus 200 caractères contenant le message.
- Si il s'agit d'une demande d'amitiés, le serveur envoie [EIRF>␣id+++] pour signaler qu'il y a une demande d'amitié venant du client id. Le client répond alors [OKIRF+++] si il accepte la demande ou [NOKRF+++] si il refuse. Le serveur répond alors par [ACKRF+++] et il envoie une notification UDP de la forme [1XX] au client ayant fait la demande d'amitiés si la demande est acceptée ou [2XX] si la demande est refusée et il ajoute un flux contenant la réponse à la liste de flux du client id.
- Si il s'agit d'une acceptation ou d'un refus de demande d'amitiés, le serveur envoie le message [FRIEN␣id+++] pour signaler que id accepte une demande ou si il s'agit d'un refus, le serveur envoie le message [NOFRI␣id+++].
- Si il s'agit d'une publicité d'un promoteur, le serveur envoie le message [LBUP>␣ip-diff␣port ␣mess+++] où ip-diff code l'IP de multi-diffusion du promoteur et port son port de multi-diffusion et mess est le message de publicité d'au plus 200 caractères.
- Si il n'y a plus de flux à consulter, le serveur envoie [NOCON+++].

Ainsi à chaque fois que le client veut consulter un de ces flux, il envoie le message [CONSU+++]. Les flux transmis sont ensuite effacés.

**Se Déconnecter.** Lorsqu'un client souhaite se déconnecté, il envoie un message [IQUIT+++] auquel le serveur répond par [GOBYE+++] et il ferme la connexion.

*Exemple de scénario* Mettons qu'il y a deux clients bob12345 et alice123 (que nous appelerons Alice et Bob). Alice est déjà enregistré sur un serveur et a consulté tous ses flots. Bob se connecte une première fois au serveur et je s'enregistre en envoyant un message [REGIS␣bob12345␣07878␣1010+++ ] (où 1010 correspond à deux octets contenant chacun l'entier 10), le serveur accepte en envoyant [WELCO+++]. Ensuite Bob reste connecté. Alice peut être connecté ou non. Bob fait une demade d'amitiés en envoyant au serveur le message [FRIE?␣alice123+++]. Le serveur envoie alors sur le port UDP d'Alice un message [010] (le premier 0 pour dire qu'elle a reçu une demande d'amitiés, les deux octets 1 et 0 pour dire qu'elle a un flux à consulter). Elle peut alors se connecter au serveur et envoyer le message [CONSU+++] pour consulter ses flux, vu qu'elle n'a qu'un seul flux en attente qui est la demande d'amitiés, le serveur lui envoie [EIRF>␣bob12345+++], disons qu'elle accepte cette demande d'amitiés, elle répondra par [OKIRF+++], le serveur acquitte en envoyant [ACKRF+++] et il envoie à Bob sur son port UDP un message [110] le premier octet 1 signifiant que quelqu'un a accepté une de ses demandes d'amitiés et les octets 1 et 0 signifie qu'il a un flux à consulter. Il peut alors (si il est encore connecté) envoyer [CONSU+++] au serveur qui enverra le message [FRIEN␣alice123] signifiant qu'Alice a accepté sa demande d'amitiés.

### Interaction avec les promoteurs

Un promoteur multi-diffuse sur son adresse et port de multi-diffusion des nouvelles qui sont des messages de la forme [PROM␣prom-mess] où prom-mess est une chaîne de caractères d'exactement 300 caractères (le message est complété par des # si il est plus court que 300 caractères). Pour faire de la publicité, un promoteur peut demander au serveur d'envoyer des messages de publicité à ces clients. Pour cela, il se connecte au port TCP dédié aux promoteurs et il envoie le message [PUBL?␣ip-diff␣port␣mess+++] où ip-diff code son IP de multi-diffusion et port son port de multi-diffusion et mess est le message de publicité d'au plus 200 caractères. Le serveur répond alors par [PUBL>+++] pour dire qu'il a transmis la publicité à ses clients à qui il envoie une notification sur leur port UDP de la forme [5XX] pour leur signaler qu'ils ont reçu un message de publicité qu'il stocke dans leur liste de flux.

## Spécification de la forme des messages

Afin que vos projets puissent communiquer entre eux, il est important que tous les messages respectent une forme bien précise. Nous décrivons dans cette partie de façon plus détaillée le format de chacun des messages.

### Spécification des messages TCP

Comme vous avez pu le remarquer, les cinq premiers octets d'un message serviront à encoder dans une chaîne de caractères correspondant au type de message. Les types des messages étant : REGIS, WELCO, GOBYE, CONNE, HELLO, FRIE?, FRIE>, FRIE<, MESS?, MENUM, MESS>, MESS<, FLOO>, FLOO?, LIST?, RLIST, LINUM,CONSU, SSEM>, MUNEM, OOLF>, EIRF>, OKIRF, NOKRF, ACKRF, FRIEN, NOFRI, LBUP>, NOCON, PUBL? et PUBL>.

Chaque message TCP finit de plus par trois octets contenant le codage en chaîne de caractères du signe arithmétique pour l'addition (ASCII 43)

Voilà maintenant les caractéristiques pour chacun des champs contenus dans les messages :

— id est codé sur 8 octets et contient la chaîne de caractères correspondant à l'identifiant d'un client (l'identifiant ne peut pas contenir trois + consécutifs).

— `port` est codé sur 4 octets et contient la chaîne de caractères correspondant au numéro de port (complété avec des 0 au début si nécessaire), par exemple pour le port 52, le codage sera la chaîne de caractères 0052.
— `mdp` est codé sur deux octets et il correspond au codage en `little-endian` du mot de passe.
— `num-mess` est codé sur 4 octets et contient la chaîne de caractères correspondant au nombre de messages (complété avec des 0 au début si nécessaire), par exemple 345 messages, le codage sera la chaîne de caractères 0345.
— `mess` est une chaîne de caractères ne contenant pas trois + consécutifs et dont la taille est d'au plus 200 caractères.
— `num-item` est codé sur 3 octets et contient la chaîne de caractères correspondant au nombre de clients dans la liste (complété avec des 0 au début si nécessaire), par exemple pour 8 clients , le codage sera la chaîne de caractères 008.
— `ip-diff` est codé sur 15 octets et contient la chaîne de caractères correspondant à l'adresse IP de multi-diffusion complété par des # (si sa taille est plus petite que 15). Par exemple, pour l'adresse de multi-diffusion 225.10.12.4, le chaîne de caractère correspondante sera `225.10.12.4####`.

### Spécification des messages UDP

Les messages UDP correspondant aux notifications des serveurs aux clients font trois octets et ont toujours la forme suivante `[YXX]` où `Y` est un octet contenant le code de la notification et `XX` est le codage sur deux octets en *little endian* du nombre de flux non consultés par le client.

Les messages UDP multi-diffusés par les promoteurs auront toujours la forme suivante `[PROM␣prom-mess]` où `prom-mess` est une chaîne de caractères d'exactement 300 caractères (le message est complété par des # si il est plus court que 300 caractères).

## Réalisation

La réalisation se fera nécessairement en C ou en Java, et possiblement dans les deux (ce qui serait un très bon exercice et un point pris en compte lors de l'évaluation). Un groupe devra programmer au moins un client, un serveur et un promoteur faisant tout le protocole.

Il est impératif de respecter scrupuleusement la spécification fournie dans le sujet ainsi que les formats de message. Toute violation sera jugée très défavorablement !

Il est vivement recommandé de proposer des extensions du projet, par exemple en proposant des nouvelles applications comme des échanges de fichiers, des hackers de mot de passe, des clients qui ont des amis sur plusieurs serveurs, des mesures statistiques sur les messages échangés, etc. Sur ces points **à vous de jouer** ! Ces applications devront être décrites de façon précise au moment du rendu du projet.

Nous mettrons en place un forum sur Moodle pour que vous puissiez communiquer entre vous et avec nous, par exemple pour faire part d'imprécisions dans le sujet, ou pour signaler que vous avez une entité qui tourne quelque part et ainsi donner l'opportunité à vos collègues de se connecter à elle. Ou encore pour préciser vos extensions. **Toute demande sur le projet devra passer par le forum.** Prenez garde à ne pas vous fier à la rumeur, la seule source d'information fiable sera le forum sur Moodle ! Au moindre doute, n'hésitez pas à y poster votre question !

La communication verbale entre groupes est non seulement autorisée mais encouragée, cependant il est **strictement interdit** d'échanger du code ; ceci serait considéré comme plagiat et par conséquent jugé sévèrement. Les discussions doivent donc seulement porter sur le fonctionnement du protocole et son interprétation, ou les formats des messages ; il vaut donc mieux éviter de donner trop d'indications sur la façon de coder les fonctionnalités.

La **notation finale** prendra en compte le fait que des groupes auront réussi à faire communiquer leurs applications entre elles. Il faudra donc penser à en faire la démonstration. D'une certaine manière, ceux

qui collaborent dans les limites indiquées ci-dessus seront récompensés ; pour ceux qui décident de faire quelque chose dans leur coin et qui ressemblerait vaguement à ce qui est décrit, ce sera l'inverse, la notation sera revue à la baisse. Un **programme qui s'exécute n'est pas suffisant** ! Vous **devez** écrire un programme qui se comporte comme indiqué ; et s'il ne communique pas avec le programme écrit par d'autres, c'est que vous n'avez pas compris ce qu'est la programmation réseau.

Pour la soutenance, il sera **nécessaire de prévoir un mode de fonctionnement verbeux** dans lequel suffisamment d'informations seront disponibles à l'écran pour comprendre ce qui se passe (affichage des messages circulant, etc).

Vos programmes devront nécessairement pouvoir être exécutés sur les machines des salles de TP de l'UFR. Toute solution ne respectant pas ce critère sera jugée invalide.

Il n'est pas demandé de fournir une interface graphique ; il est même recommandé de s'abstenir d'en faire une (quoique vous en pensiez). Cela ne vous apportera rien, vous fera perdre du temps et n'impressionnera en rien les examinateurs ; et surtout **ce n'est absolument pas dans le programme de cet enseignement** par conséquent, nous insistons : **pas d'interface graphique**.

Votre projet devra bien entendu être robuste (c'est à dire sans erreur) et devra être capable de gérer des messages erronés sans planter.

La réalisation du projet se fera par groupe **d'au moins deux** étudiants et **d'au plus trois** (cette règle ne souffrira aucune exception, un peu d'effort de la part de chacun pour ne pas prendre que des amis proches dans son groupe devrait aider). Et bien entendu, chacun dans un groupe devra travailler et il n'est pas exclu que des étudiants d'un même groupe n'aient pas la même note au final.

La composition des groupes devra être envoyée par mail à `sangnier@irif.fr` avant **le Vendredi 24 Mars 2017 23h59**. Toute personne n'ayant pas soumis de groupe avant cette date prend le risque de ne pas avoir de note au projet.

Le projet devra être rendu sur Moodle la veille de la soutenance, dont la date vous sera fournie ultérieurement. Des informations sur la soutenance (ordre de passage et instructions) seront aussi fournies plus tard sur Moodle.