

TP Réseaux : Modèle Client/serveur

MIAGE nantes | Master 1 alternance

Rapport

Sebastien heuzé, Cyril Le Driant

2013

Table des matières

[1. Introduction 2](#_Toc379274213)

[2. Analyse 3](#_Toc379274214)

[2.1. Diagramme de cas d’utilisation 3](#_Toc379274215)

[2.2. Diagramme de classes. 4](#_Toc379274216)

[2.3. Diagrammes de séquence 5](#_Toc379274217)

[3. Conception 6](#_Toc379274218)

[3.1. Présentation du protocole 6](#_Toc379274219)

[3.2. Diagramme de déploiement 7](#_Toc379274220)

[4. Problèmes rencontrés 8](#_Toc379274221)

[5. Exemple d’utilisation 9](#_Toc379274222)

[6. Conclusion 12](#_Toc379274223)

# Introduction

Actuellement en master MIAGE première année, nous suivons un module de formation réseau dont l’objectif est de nous faire comprendre le fonctionnement d’un réseau en détail (différentes couches, protocoles…)

L'objectif de ce module est de comprendre :

* Le fonctionnement des principaux protocoles utilisés dans l’internet
* Les enjeux et les évolutions de l’adressage IP
* Le principe du routage aussi bien statique que dynamique
* Le principe de configuration des applications client/serveur (FTP, DHCP, TELNET)
* Utiliser les principaux services (DNS, SAMBA) et outils de l’internet (ping, etc…)
* Les notions de sécurité (pare-feu)

Afin de mettre en pratique les connaissances acquises durant le module, un projet d’application client/serveur clôture ce dernier.

Dans ce projet, les étudiants vont devoir mettre en application l’ensemble des connaissances acquises durant le module via une application C.

Le projet se propose de construire une application avec échange de données entre machins distantes en utilisant les sockets.

L’équipe de projet est donc composée par Sébastien Heuzé et Cyril Le Driant, tous deux alternants à Cap Gemini.

Le présent rapport a pour but de présenter l’application (analyse, détails du protocole, manuel)

Bonne lecture.

# Analyse

L’analyse du système a pour but de modéliser le domaine fonctionnel couvert par le futur logiciel. Le besoin est exprimé sous forme de diagrammes respectant des normes de modélisation afin d’être à la fois rigoureux et compréhensibles par des personnes n’ayant pas de compétences particulières dans les techniques de modélisation ni dans le domaine fonctionnel.

Dans cette partie vont être présentés différents diagrammes UML permettant de décrire certaines fonctionnalités du logiciel ainsi que son organisation logique.

## Diagramme de cas d’utilisation

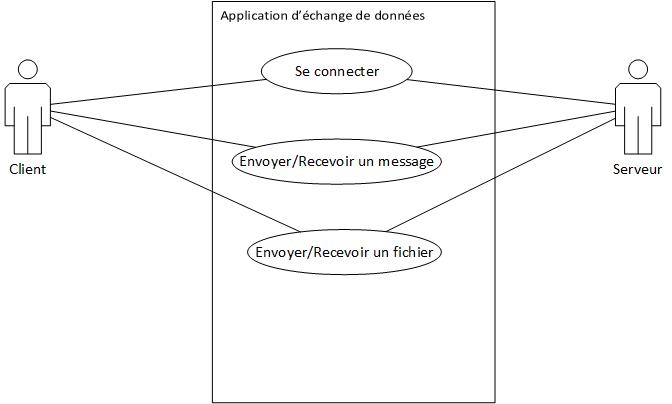


Figure - Diagramme de CU

Le diagramme de cas d’utilisation ci-dessus présente les différentes fonctionnalités que doit permettre le programme

Celles-ci comprennent :

* La connexion de l’utilisateur au serveur : ouverture d’un canal de communication entre le client et le serveur
* L’envoi et la réception de message : Possibilité d’envoyer un message aux différents clients connectés via le serveur et inversement
* L’envoi et la réception de fichiers : Possibilité d’uploader un fichier sur le serveur pour ainsi permettre à d’autres utilisateurs de le télécharger

## Diagramme de classes.

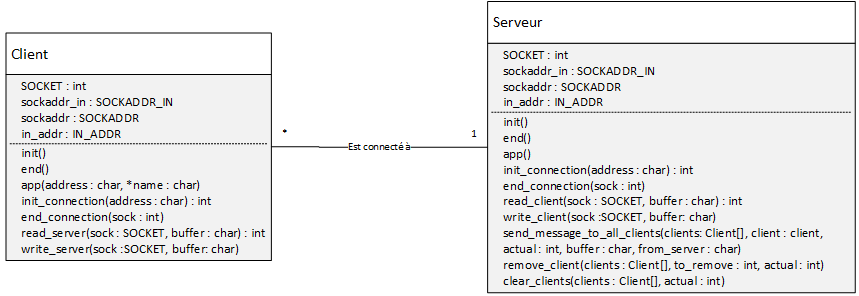


Figure - Diagramme de classes

Le programme est structuré comme suit :

* Une classe client et une classe serveur.

Chaque classe dispose de ses méthodes pour initialiser la connexion réseau et envoyer/recevoir des messages.  
   
Le transfert de fichier se réalisant via les messages, ce dernier n’a pas de fonction spécifique

## Diagrammes de séquence

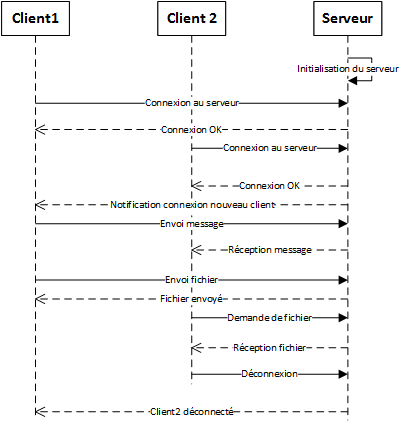


Figure - Diagramme de séquence

Ce diagramme présente un échange basique entre deux clients. Cela nous permet de voir les différents messages envoyés et reçus par les clients ou le serveur.

# Conception

La conception a pour but de répondre de manière technique aux besoins posés dans la phase d’analyse. Pour cela, une présentation du protocole ainsi qu’un diagramme de déploiement vont être présentés

## Présentation du protocole

Les sockets sont des flux de données, permettant à des machines locales ou distantes de communiquer entre elles via des protocoles.  
Les différents protocoles sont TCP qui est un protocole dit "connecté", et UDP qui est un protocole dit "non connecté".

#### Le protocole UDP

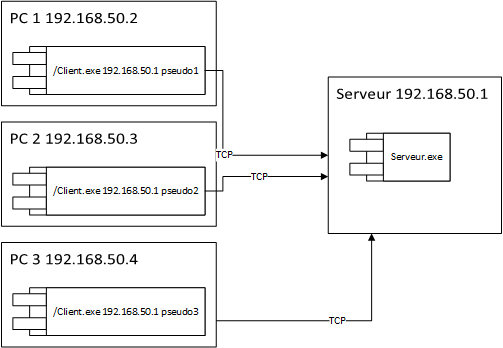
UDP est un protocole orienté "non connexion". Pour faire simple, lorsqu'une machine A envoie des paquets à destination d'une machine B, ce flux est unidirectionnel. En effet, la transmission des données se fait sans prévenir le destinataire (la machine B), et le destinataire reçoit les données sans effectuer d'accusé de réception vers l'émetteur (la machine A). Ceci est dû au fait que l'encapsulation des données envoyées par le protocole UDP ne permet pas de transmettre les informations concernant l'émetteur. De ce fait, le destinataire ne connait pas l'émetteur des données hormis son IP. 

#### Le protocole TCP

#### Contrairement à l'UDP, le TCP est orienté "connexion". Lorsqu'une machine A envoie des données vers une machine B, la machine B est prévenue de l'arrivée des données, et témoigne de la bonne réception de ces données par un accusé de réception. Ici, intervient le contrôle CRC des données. Celui-ci repose sur une équation mathématique, permettant de vérifier l'intégrité des données transmises. Ainsi, si les données reçues sont corrompues, le protocole TCP permet aux destinataires de demander à l'émetteur de renvoyer les données corrompues.

Nous allons utiliser ici le protocole TCP

## Diagramme de déploiement



Ce diagramme représente l’application dans son contexte.   
Ci-dessus nous pouvons voir trois machines connectées simultanément à un même serveur qui se charge de relayer l’information. Ces machines sont situées sur le même réseau mais cela n’est pas obligatoire tant que le pare-feu le permet. (Filtrage des ports)

# http://b.vimeocdn.com/ps/448/158/4481583_300.jpgProblèmes rencontrés

Lors de la création de cette application nous avons rencontré deux types de problèmes :

* **Absence d’objet String** : en C, des tableaux de char sont utilisés pour stocker des suites de caractères, cela complique la gestion de ces dernières (Ne pas oublier le caractère de fin). De plus l’utilisation des pointeurs pour gérer ce tableau nécessite une attention particulière. En effet lorsque l’on traite cette chaine de caractère, il faut prendre en compte que l’on peut perdre la chaine principale à tout moment (la méthode split utilise la chaine de caractères originale pour ses traitements, il faut donc la cloner avant si l’on veut la sauvegarder en mémoire)
* **Compatibilité Windows** : Bien que l’on ait mis en place des ajustements pour assurer la compatibilité du programme sous Windows, il nous a été impossible de la garder. En effet la gestion d’entrée clavier est particulière, il est impossible dans la console de pouvoir écrire dans cette dernière pendant que le programme est en écoute sur d’autres entrées (sockets).

# Exemple d’utilisation

Pour cet exemple d’utilisation, le dossier racine correspond à la racine du dossier de rendu.

**Etape 1 : Démarrer le serveur**

Rendez-vous dans le dossier Serveur et lancez le serveur par cette commande (terminal séparé) :

**$> ./server**

Le serveur se lance.  
  
**Que se passe-t-il ?**

Lors de son lancement le serveur initialise la connexion sur le port 5000, initialise la liste de clients et reste à l’écoute.

**Etape 2 : Lancer client 1**

Rendez-vous dans le dossier Client 1 et lancez le client par cette commande (terminal séparé) :

**$> ./client localhost pseudoclient1**

Le client se lance.  
  
**Que se passe-t-il ?**

Lors de son lancement le client crée une connexion socket sur le port 5000 du serveur locahost, il notifie donc le serveur de sa connexion qui se charge d’ajouter le client à sa liste de client.

**Etape 3 : Lancer client 2**

Rendez-vous dans le dossier Client 1 et lancez le client par cette commande (terminal séparé):

**$> ./client localhost pseudoclient2**

Le client se lance.  
  
**Que se passe-t-il ?**   
Lors de son lancement le client crée une connexion socket sur le port 5000 du serveur locahost, il notifie donc le serveur de sa connexion qui se charge d’ajouter le client à sa liste de client.

**Etape 4 : Envoyer message depuis Client 1**

Prenez le terminal qui a servi à lancer le client 1 et tapez un message, puis terminez en appuyant sur la touche entrée.  
  
Le message est envoyé

**Que se passe-t-il ?**

Le client 1 envoie un message au serveur, le serveur reçoit ce message et se charge de renvoyer ce message aux autres clients, les autres clients reçoivent donc ce message précédé par le pseudo du client qui envoi le message.

**Etape 5 : Envoyer un fichier depuis Client 1**

Prenez le terminal qui a servi à lancer le client 1 puis taper cette commande (le fichier fichiertext.txt doit exister) :

**/sendfile fichiertest.txt**

Le fichier est envoyé

**Que se passe-t-il ?**

Le client 1 récupère le contenu du fichiertest.txt et le concatène à la suite du message, le contenu du fichier est donc envoyé avec le message au serveur qui le récupère et qui crée un fichier portant le même nom avec le contenu envoyé dans son répertoire. Le client est alors notifié du succès de l’échange.

**Etape 6 : Recevoir un fichier depuis Client 2**

Prenez le terminal qui a servi à lancer le client2 puis taper cette commande (le fichier fichiertext.txt n’existe pas dans le dossier Client 2) :

**/getfile fichiertest.txt**

Le fichier est reçu par le client 2

**Que se passe-t-il ?**

Le client 2 envoi une commande au serveur (/Getfile fichiertest.Txt) ce dernier reconnait la commande et va donc lire le contenu du fichier avant d’y retourner son contenu via le même canal (le client 2 étant resté à l’écoute après l’envoi de la commande). Le client 2 reçoit le contenu fichier et le sauvegarde en local dans son dossier.

# http://d3n8a8pro7vhmx.cloudfront.net/3dna/pages/4749/suggestion_pages/2639/sixhundred.jpg?1381260832Conclusion