École Polytechnique de l'Université de Tours
64, Avenue Jean Portalis
37200 TOURS, FRANCE
Tél. +33 (0)2 47 36 14 14
www.polytech.univ-tours.fr

Département Informatique
3e année
2012 - 2013

Rapport Projet Système d'Exploitation

Partage de données images au sein d'un réseau mobile massivement utilisateurs

Encadrants
Mathieu DELALANDRE
mathieu.delalandre@univ-tours.fr

Université François-Rabelais, Tours

Étudiants
Anthony BRAIN
anthony.brain@etu.univ-tours.fr
Baptiste JAECKERT
baptiste.jaeckert@etu.univ-tours.fr

DI3 2012 - 2013

Version du 10 juin 2013

Table des matières

Table des figures

Liste des tableaux

Introduction

De plus en plus de smartphones voient le jour chaque années, et de plus en plus d'images, de vidéos et de données circulent à travers le monde. L'interconnexion entre tous les appareils est maintenant établie, facilitant ce transfert de données. Cependant cette incroyable augmentation du nombre d'utilisateurs et de données, nous amène à se poser des questions sur la manière d'optimiser et de garantir le transfert de ces données.

En effet, l'évolution des technologies de capture d'images ou de vidéo par exemple, permet d'acquérir des données de bien meilleure qualité, mais aussi par conséquent de plus grandes tailles. Ceci combiné à la croissance du nombre d'utilisateurs peut poser des problèmes de surcharge de réseau et d'encombrement de données.

Ce projet portera donc sur l'étude du partage de données au sein d'un réseau mobile massivement utilisateurs. Le but de ce projet sera donc d'établir un système permettant le transfert de données images, parmi un grand nombre de mobiles présent sur un même site, et d'y apporter des optimisations afin de garantir la fiabilité et l'efficacité des transferts.

Les utilisateurs pourront uploader des images sur un serveur, qui pourra à son tour retransmettre à tous les utilisateurs connectés, l'image reçu précédemment.

Dans ce rapport nous présenterons tout d'abord les différents protocoles de connexion utilisés. Nous présenterons dans une seconde partie le travail que nous avons effectué, pour finir dans une dernière partie par les parties du projet que nous n'avons pas pu aborder.

Présentation du problème

La problématique posée par ce projet est intéressante dans le cas où nous avons un très grand rassemblement de personnes. Le principe est que si une personne prend une photo qui serait très intéressante pour toutes les personnes présentes (exemple une photo prise au premier rang dans un concert), que celle-ci puisse la faire partager à toutes les personnes intéressées (les personnes au loin qui ne voient rien).

Cependant, pour que cela puisse être possible, il faut que les supports (ex : mobiles, tablettes,...) sur lesquels les personnes visionnent la photo, soient connectées entre eux. Pour cela, il est nécessaire d'avoir accès à un réseau local présent sur le site, où les personnes puissent toutes se connecter, via des bornes wifi par exemple. Sur ce réseau nous avons un serveur qui a un rôle très important, celui de rediffuser les images prises par les utilisateurs aux autres utilisateurs qui le souhaitent.

Voici un schéma expliquant le principe :

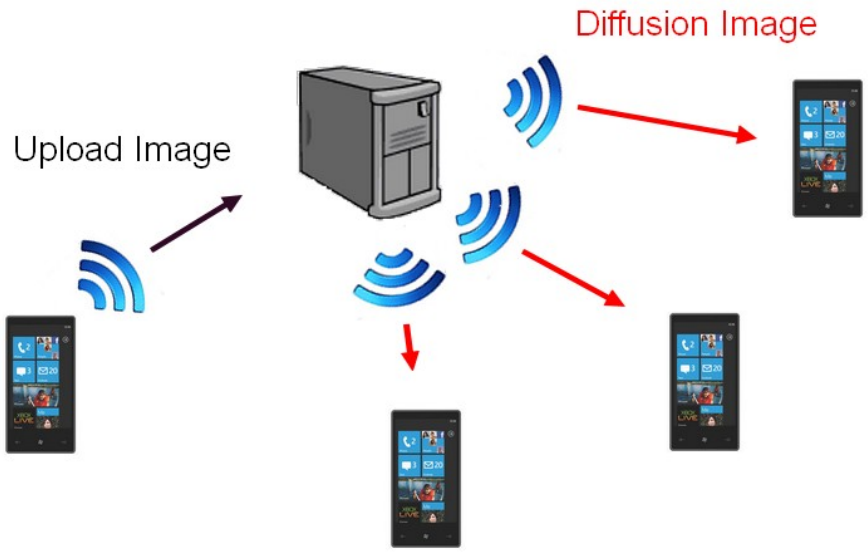

FIGURE 2.1 – Connexion entre le serveur et les mobiles

Voici donc ce que nous allons tenter d'établir. Il faut cependant faire attention car de nombreux problèmes comme énoncés en introduction vont se confronter a nous.

En effet il va falloir gérer l'envoi de plusieurs images en même temps tout en minimisant l'efficacité des transferts. Mais aussi choisir les bons protocoles de connexion entre les mobiles et le serveur afin de pouvoir transférer les images sans perte de données. L'élaboration d'application de visionnage et de téléchargement, tout en minimisant la consommation de la batterie est un problème qui doit être pris en compte également.

Bien évidemment il a été difficile pour nous de traiter tout ces aspects lors du déroulement de notre projet, et nous ne nous sommes attarder que sur l'élaboration d'une connexion entre le serveur et les appareils mobiles. Nous avons également, après bien étudier les problèmes restant, proposé différentes pistes afin de venir a bout de ce projet, tout en prenant en compte les différentes contraintes a ne pas

négliger.

Les différents protocoles disponibles

3.1 Les protocoles

Avant de commencer à programmer pour réaliser ce que nous avons énoncés précédemment, il nous faut tout d'abord choisir un protocole de connexion.

En effet nous ne pouvons pas nous permettre de communiquer n'importe comment entre le serveur et les mobiles. D'ailleurs il en est de même pour n'importe quelle communication, il faut établir des règles, c'est à dire utiliser une convention, ou plus simplement ce que l'on appelle un protocole de communication.

L'utilisation de protocoles est très important, car imaginons que deux machines communiquent entre elles. Lors de l'envoi du message par une machine A à une machine B, le message est transmis sous une certaine forme de codage, cependant arrivé à destination, si la machine B ne sait pas dans quel protocole le message a été codé, il ne saura jamais le lire correctement.

C'est pourquoi dans chaque communication, il existe des en-tête qui sont rajouter au message, afin de pouvoir identifier le type de protocole que l'on utilise.

Il existe de trop nombreux protocoles pour pouvoir tous les énoncer, cependant on peut en distinguer 2 types :

- Les protocoles dit connectés
- Les protocoles dit non-connectés

Les protocoles connectés nécessitent la mise en place d'une connexion au préalable, une fois la connexion établie le transfert de données peut commencer. Une fois qu'il est fini il suffit de fermer la connexion. Parmi ces protocoles il existe le protocole TCP qui est très utilisé.

Les protocoles dit non-connectés, eux ne nécessitent aucune connexion au préalable, le client envoie directement les données directement à l'endroit désiré. Le protocole UDP fait parti de ces protocoles

3.2 Avantages/Désavantages

Cependant chaque type de protocoles à ses avantages et désavantages. En effet les protocoles connectés seront beaucoup plus fiables, la connexion étant établie initialement, il y aura moins de risque que les données soient envoyées au mauvais endroit. De plus, si l'ensemble des données n'est pas bien transmis, le protocole connecté renvoie l'état de la réception, si elle n'est pas bonne il indique qu'il faut renvoyer le paquet de données.
De même, la vitesse de transfert sera beaucoup plus importante en cas d'image de grande qualité.

D'autre part, l'inconvénient de ce type de protocoles est que le temps que la connexion soit établie, cela augmente le temps de transfert total.

Les protocoles non-connectés eux, gagnent du temps en évitant la phase de connexion, cependant comme elle n'est pas établie il n'y a pas d'acquittement des données de la part du serveur.

Nous sommes donc sur un protocole plus rapide mais moins fiable. Il est donc conseillé de l'utiliser pour l'envoi de données peu importantes tant au niveau sécurité que volume.

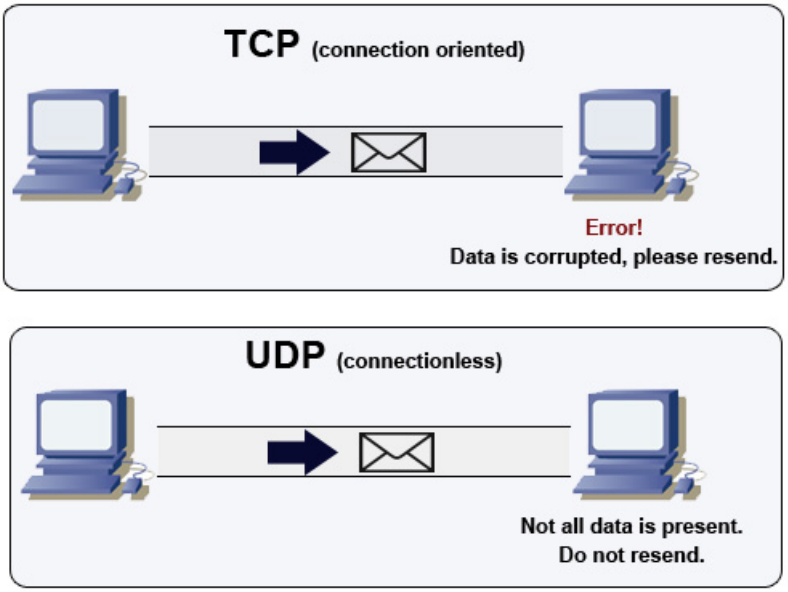

FIGURE 3.1 – Différences entre le protocoles connecté et non-connecté

3.3 Comment savoir où envoyer ?

Cependant, quelque soit le type de protocole employé on peut se poser une question, comment le mobile peut savoir où envoyer l'image qu'il souhaite transmettre (à quelles adresse Ip) ?

Pour cela il faut savoir que dans le cas de connexion et de transferts d'images, nous sommes dans une connexion client/serveur. Ici le client est le mobile, et il communique avec le serveur.

Le client a différentes manières de connaître l'adresse du serveur :

- Le client connaît déjà l'adresse du serveur et n'a plus qu'à lui envoyer son image.
- Le client envoie un message sur l'ensemble du réseau, indiquant qu'il souhaite envoyer une image au serveur, le serveur lui répond en lui fournissant son adresse.
- Le client communique avec un système qui a pour but de répertorier toutes les adresses du réseau.

Dans notre cas, et pour notre exemple du concert, on considérera que le client connait l'adresse du serveur.

3.4 Choix du protocole utilisé

Maintenant que nous connaissons les différents types de protocoles existants, nous allons devoir en choisir un pour avancer dans notre projet.

La logique voudrait que nous choisissions le protocole TCP, il est beaucoup plus fiable et comme les images représentes des données assez importantes, il est préférable de l'utiliser.

Cependant nous allons choisir le protocole UDP pour différentes raisons.

Tout d'abord parce que nous allons commencer à envoyer simplement des chaînes de caractères au début, et ensuite car nous voulons aborder la gestion des erreurs, qui elle est plus simple à établir en UDP du fait que nous n'ayons aucune connexion établie.

Début de la mise en place de la connexion UDP

4.1 Les problèmes de Socket rencontrés sous Linux

Une fois que nous avons décidés le protocole à utiliser, nous avons commencé à programmer un code en langage C, afin d'établir la connexion entre un client et un serveur.

Nous avons décidés de coder en C et de créer un programme sur ordinateur, et non sur mobile, tout du moins au début. Cette décision a été prise car aucun de nous deux, n'avait dans nos cursus précédents, appris les langages utilisés sur mobile.

Nous avons donc commencé à programmer sous Linux. Cependant nous sommes rester bloqués assez rapidement. Pour établir une connexion entre un client et un serveur, il faut dans n'importe quel type de protocoles, utiliser des Socket.

Les Socket sont ce qui permet d'ouvrir des ports et de faire passer les données par eux. Sans Socket il n'est même pas envisageable d'établir une connexion UDP.

Or, lorsque nous avons commencé à vouloir utiliser les Socket, nous nous sommes rendus compte que sous Linux il n'existe pas de librairie déjà existante, et qu'il fallait déclarer nous même la structure des Socket que nous voulions utilisée. Nous avons perdu énormément de temps lors de cette phase, qui malgré tous nos efforts n'a pas abouti.

Nous n'avons finalement pas réussi a faire marcher le Socket malgré nos tentatives pour déclarer sa structure.

Après plusieurs recherches nous avons pris connaissance d'une API (Application Programming Interface), fonctionnant sous Windows et se nommant Winsock. Nous nous sommes donc redirigés vers une programmation sous Windows.

4.2 Winsock

4.2.1 Principes

Depuis des années, les applications se sont tournées vers une communication réseaux afin d'interagir entre elles. La Socket Windows, ou plus couramment appelé Winsock, est une API (Application Programming Interface) procurant des fonctions d'accès aux protocoles réseaux. Winsock représente donc une interface API permettant l'utilisation du protocole TCP/IP ou UDP sur une interface Windows.

Cette API a connu plusieurs versions, Winsock 1.1 à été publié le 20 janvier 1993 afin de créer un standard universel pour les applications TCP/IP. Les auteurs des Windows Socket 1.1 ont limité les possibilités à l'utilisation d'un seul protocole TCP/IP, contrairement à la version 2.0 qui propose en plus, la suite des protocoles ATM, IPX/SPX, DECnet et le sans fil.

Le passage à Winsock 2.0 s'est fait en gardant 100% de compatibilité permettant aux applications déjà développées de continuer à fonctionner.

Il faut savoir que lorsque l'API Winsock est utilisée, celle-ci ne fonctionne que si le protocole utilisé par le client est le même que le serveur (ce qui reprend ce que l'on a expliqué dernièrement). Les socket se situent entre la couche transport et les couches applicatives du modèle OSI.

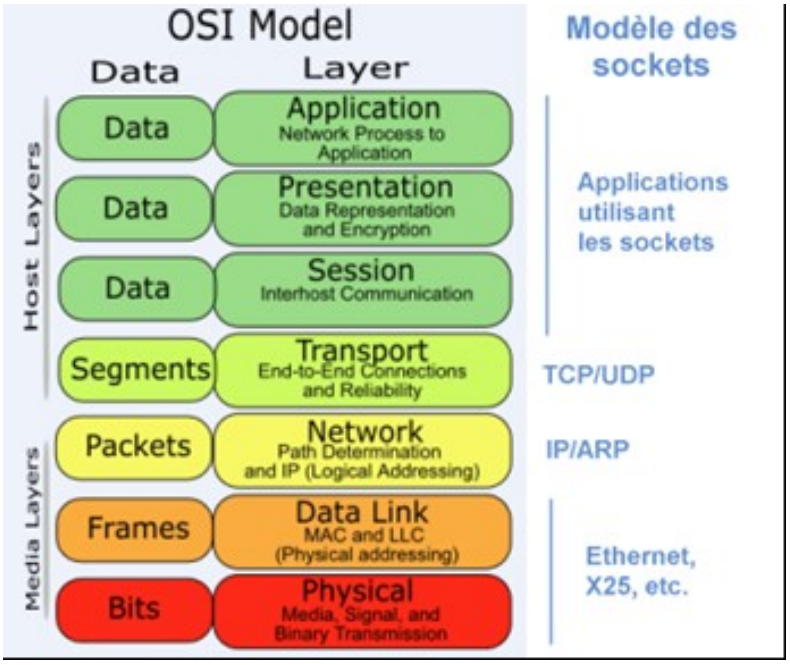

FIGURE 4.1 – Utilisation des sockets dans le modèles OSI

Grâce à Winsock nous pouvons donc utiliser beaucoup plus facilement les socket, les fonctions que l'API nous permet d'appeler créer directement le socket et il ne nous reste plus qu'à manipuler la structure du socket pour l'utiliser.

Nous verrons par la suite dans les prochaines parties comment nous l'utilisons.

4.2.2 Adaptable sur mobile ?

Dans l'optique ou nous voudrions transposé notre code sur mobiles, il était intéressant de se renseigner si l'API était utilisable dans un autre langage de programmation.

Winsock étant fonctionnel sous Windows nous nous sommes donc naturellement tournés vers le Windows phones (téléphone conçut par Microsoft et possédant son propre système d'exploitation).

Malheureusement nous nous sommes rendus compte que comme le langage de programmation utilisé pour le Windows phone était le C#, et que ce langage était orienté-objet, nous ne pouvions donc pas l'utiliser.

La transposition du code sur Windows phone fera l'objet d'une partie dans la suite du rapport.

Mise en place de la connexion sur PC

Nous avons donc commencé à programmer, afin de mettre en place la connexion UDP. Au niveau du déroulement du code, l'utilisation des socket doit se faire dans un ordre précis. Voici un schéma synthétisant les différentes étapes qui ont lieu lors de l'échange de données, et qui prend en compte l'utilisation des socket :

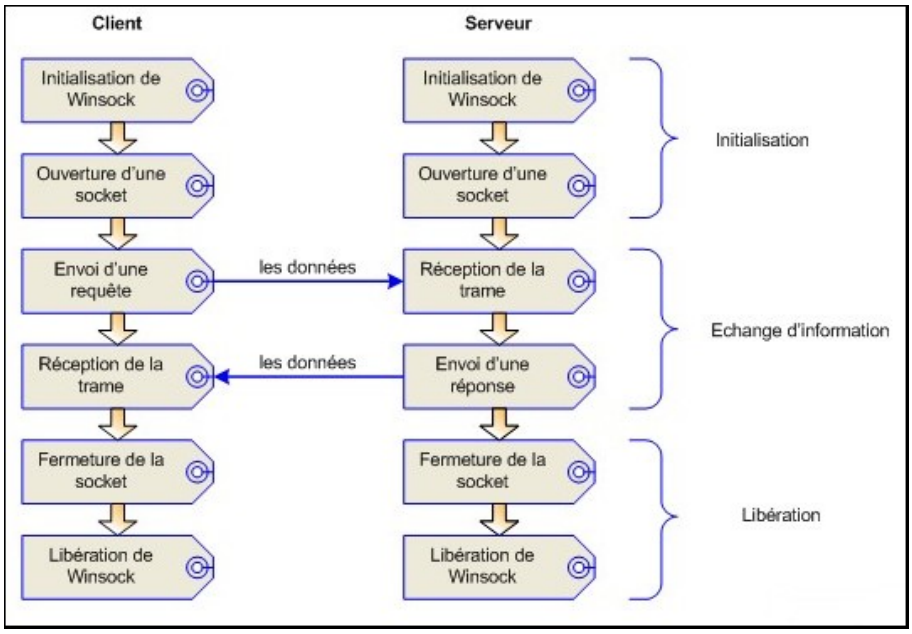

FIGURE 5.1 – Déroulement des étapes de la connexion UDP utilisant les sockets

Nous allons détailler les étapes du coté client et du coté serveur, en reprenant le schéma, et en détaillant les principales fonctions indispensables à la connexion UDP.

5.1 Côté client

Du côté du client nous commençons tout d'abord à démarrer winsock, afin de pouvoir utiliser les fonctions que contient la librairie winsock2.h. Pour démarrer winsock il suffit d'utiliser la fonction suivante :

WSAStartup(MAKEWORD(2,2),& initialisation_win32).

Une fois winsock démarré, nous pouvons créer un nouveau socket avec la fonction :

socket(AF_INET,SOCK_DGRAM,0).

Le 1er paramètre est la famille du socket, et SOCK_DGRAM permet d'envoyer un paquet, directement à la destination souhaité, sans faire d'accept() ou de connect() qui sont des fonctions nécessaires lors d'une connexion TCP. SOCK_DGRAM est donc propre à la connexion UDP.

Bien entendu le socket possède une structure. Les informations sur celle-ci se trouve dans SOCKADDR_IN. Cette structure contient des informations comme la famille du socket, l'adresse du serveur, ou bien le port sur lequel on souhaite écouter ou émettre.

Nous déclarons une structure SOCKADDR_IN propre au client, dans laquelle nous mettons les informations du client. Par la suite nous lions cette structure au socket créer, grâce à la fonction :

bind(le_socket_a_lié , structure_a_lié , taille_structure).

Une fois tous ces éléments initialisés nous allons pouvoir passer à l'envoi et la réception des données.

Il existe deux fonctions importantes pour l'UDP, la fonction sendto qui envoie les données, et la fonction recvfrom qui les reçoit.

Pour pouvoir utiliser la fonction sendto il faut d'abord avoir remplit une structure contenant les informations du destinataire, dont notamment l'adresse de destination pour savoir où envoyer. A la suite de ça, nous pouvons utiliser la fonction en prenant en paramètre la structure et les données à envoyer.

Pour recevoir des données du serveur, comme par exemple un message d'acquittement, nous utilisons la fonction recvfrom, qui s'utilise de la même manière que sendto.

Si la communication est terminée, il suffit de fermer le socket avec la fonction closesocket.

Et pour finir on éteint winsock avec WSACleanup.

Toute ces étapes avec ces fonctions qui sont utilisées correspondent au côté client, passons maintenant du coté serveurs.

5.2 Côté serveur

En regardant avec attention le schéma du déroulement des étapes affichées précédemment, on peut constater qu'il n'y a pas beaucoup de différence avec le déroulement du côté client.

En effet la seule différence va se faire dans l'ordre de réception et d'envoi des données. Comme nous sommes du coté serveur, c'est lui qui écoute en premier, par conséquent la fonction recvfrom sera avant la fonction sendto. Le serveur reçoit les données puis envoie une réponse pour confirmer la réception.

Comme à la base le serveur n'a aucune donnée sur le client, il récupère parmi les informations reçues, l'adresse d'émission du client afin de pouvoir lui répondre.

Cependant, dans une communication comme celle-ci, il faut bien faire attention à ce que le client qui émet sur un port soit le même que celui sur lequel écoute le serveur.

Ces informations seront connues de différentes manières comme nous l'avons expliquer dans la partie 3.3.

Les différentes étapes détaillées dans ces deux parties fonctionnent très bien dans un cas ou il n'y a aucun problème lors du transfert de données. Cependant si les données sont mal transmises la connexion s'arrête et les données sont perdues. Il faut donc introduire une gestion d'erreurs.

5.3 Gestion des erreurs

Un des problèmes majeurs avec le protocole UDP est qu'il est orienté "non connexion". Il n'y a pas de gestion d'erreurs et dans le cas où un message n'aurait pas réussi à s'envoyer ou qu'il se serait perdu en cours de transmission, il n'y a aucun moyen de le savoir.

Il va donc falloir mettre en place un mécanisme pour prévoir et pouvoir faire face à ces différentes erreurs possible. Pour cela il existe différentes solutions possibles, mais nous ne présenterons ici que les deux solutions que nous avons retenu pour notre projet.

La première solution a mettre en place est la notion de RRA , pour Request Reply Acknowledge reply. Pour schématiser le principe est le suivant :

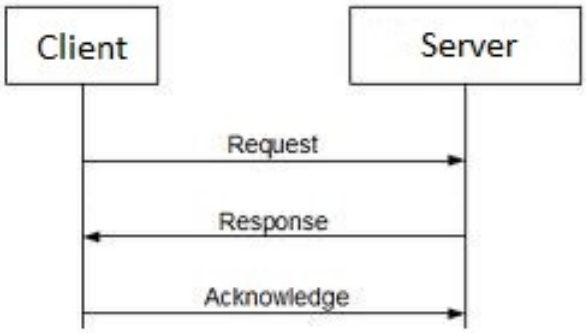

FIGURE 5.2 – Mécanisme RRA

- Request : le client envoie un message (contenant ses données) au serveur
- Reply : lorsque le serveur reçoit ces données il envoie un message de réponse au client
- Acknowledge reply : le client renvoie une réponse au serveur pour dire qu'il a bien reçu sa réponse

De cette façon le client est sûr que le serveur a bien reçu ses données, puis le serveur est sûr que le client est bien reçu sa réponse.

Mais cependant le problème n'est pas entièrement résolu, car si ce mécanisme nous permet de détecter une erreur lors de la transmission, maintenant il faut savoir comment réagir.

C'est donc le deuxième mécanisme à mettre en place, qui est le Timeouts, qu'on pourrait résumer tout simplement par la création d'un Timer qui attend la réponse du message envoyé, et qui au bout d'un temps donné sans recevoir de réponse exécute une action spécifique.

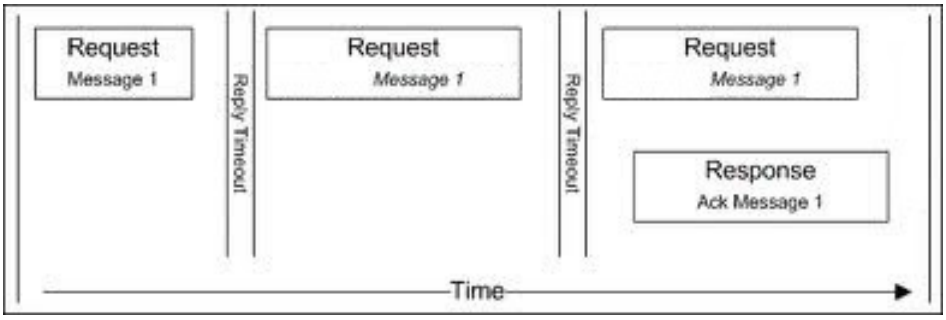

FIGURE 5.3 – Principe du Timeouts

Une des solutions les plus basiques concernant le choix de l'action à effectuer est tout simplement de notifier le client ou le serveur que le message n'a pas été envoyé correctement puis de quitter l'application.

Cependant ce n'est pas adapté à notre application, car dans le cas ou c'est la réponse du serveur au client qui est perdu pendant la transmission, le Timeouts du client se déclenchera et indiquera que le message n'a pas été envoyé alors que ce n'est pas le cas.

La solution qu'on a retenu est donc de renvoyer le message à chaque fois jusqu'à obtenir une réponse du serveur.

Avec cette solution, il reste à prévoir un dernier problème possible, c'est le cas où le serveur aurait quand même répondu à la requête du client, mais un peu trop tard, ce qui fait que le client renverrait une nouvelle fois sa requête alors que le serveur y avait bien répondu. Dans ce cas le serveur devra répondre 2 fois, même si les requêtes sont identiques.

Là ou cela pose problème pour nous, c'est que nous envoyons des images qui peuvent être lourdes en données, donc qui devront s'envoyer en plusieurs requêtes, avec une reconstitution au niveau du serveur de l'image en rajoutant chaque segment de données l'un derrière l'autre.

Donc si on reprend l'exemple du problème cité ci-dessus, le serveur recevrait les 2 requêtes identiques et répondrait aux deux, puis il les additionnerait à la suite pour reconstituer l'image, ce qui fait qu'on aura une séquence en trop et que l'image restituée ne sera pas la bonne.

Pour résoudre ce problème il faudra donc mettre en place, au niveau du serveur, un historique des réponses aux requêtes du client. Et comme le client devra envoyer plusieurs segments de données à la suite pour une image, on pourra alors considérer une requête comme étant l'Acknowledge reply de la requête précédente et ainsi de suite.

De cette façon dans notre historique on aura juste à sauvegarder la réponse du serveur à la dernière requête, ce qui permet un gain considérable en terme de mémoire surtout pour une application sur téléphone.

Pour savoir si le serveur reçoit une requête en double, il aura juste à comparer l'identifiant de la nouvelle requête à l'identifiant de la dernière requête a laquelle il a répondu.

Ainsi notre protocole UDP se rapprochera du mécanisme d'un protocole UDP, tout en gardant sa rapidité de transfert des données.

Transposition sur mobile Windows Phone en C#

Nous avons donc réussi à créer un programme qui transmet des données entre client et un serveur en UDP, tout en gérant les erreurs.

Cependant notre programme utilise les socket à travers winsock, et est programmé afin de fonctionner lors d'une communication entre deux ordinateurs.

Si l'on reprend la problématique du projet, le but est de l'établir sur mobiles. Nous nous sommes donc posés la question de savoir s'il était possible de transposer le code dans un autre langage, fonctionnant sur mobile, et utilisant également winsock.

Comme winsock est propre a Windows il était évident que sous langage Android et Objective-C, winsock ne fonctionnerait pas. Nous nous sommes donc tournés vers le Windows Phone et son système d'exploitation créé par Microsoft. Son langage de programmation est le C#.

FIGURE 6.1 – Windows Phone 7

6.1 La classe Socket

Le C# est un langage dit orienté-objet, par conséquent il utilise ce que l'on appelle des classes. Nous espérions que la librairie winsock était prise en compte, mais ce ne fut pas le cas. Cependant nous avons découvert qu'il existait une classe qui remplaçait winsock, et qui est la classe socket.

La classe Socket fournit une grande variété de méthodes et de propriétés pour les communications réseau. Elle permet d'effectuer des transferts de données synchrones et asynchrones à l'aide de nombreux protocoles de communication les plus courant.

Lors d'un transfert synchrone, le client et le serveur peuvent communiquer en même temps et les données peuvent se croiser. Tandis que dans un transfert asynchrone le client parle d'abord, puis le serveur répond, et ainsi de suite. Chaque interlocuteur attend que l'autre ait fini de parler.

Comme nous sommes dans un mode asynchrone avec le protocole UDP, les fonctions de la classe socket à utilisé pour envoyer et recevoir des données seront : BeginSendTo et EndSendTo pour envoyer des datagrammes, et BeginReceiveFrom et EndReceiveFrom pour recevoir des datagrammes.

Lorsque l'on a terminé d'envoyer et de recevoir des données, il faut utiliser la méthode Shutdown pour désactiver le socket . Après avoir appelé la méthode Shutdown, il faudrait utiliser Close pour libérer toutes les ressources associées à Socket.

Par manque de temps et surtout parce que nous n'avons aucune base en orienté-objet et en C#, nous n'avons pas pu aller plus loin pour proposer un code pouvant peut être fonctionner sur le mobile Windows phone. Nous n'aurions pas pu dans tous les cas le tester, car l'école ne possède pas de Windows phone.

L'envoi d'images au sein du réseau, concept P2P

Le Peer to Peer (P2P) est un système permettant à plusieurs ordinateurs de communiquer via un réseau, de partager simplement des objets, des fichiers le plus souvent, mais également des flux multimédia continus (streaming), un service (comme la téléphonie avec Skype), etc.. sur Internet.

Le P2P a permis une décentralisation des systèmes : il permet à tous les ordinateurs de jouer directement le rôle de client et serveur.

En P2P, les systèmes de partage de fichiers permettent de rendre les objets d'autant plus disponibles qu'ils sont populaires, et donc répliqués chez un grand nombre de personnes.

Cela permet alors de diminuer la charge, en nombre de requêtes, imposée aux appareils des personnes partageant les fichiers populaires, ce qui facilite l'augmentation du nombre d'appareils et donc de fichiers dans le réseau.

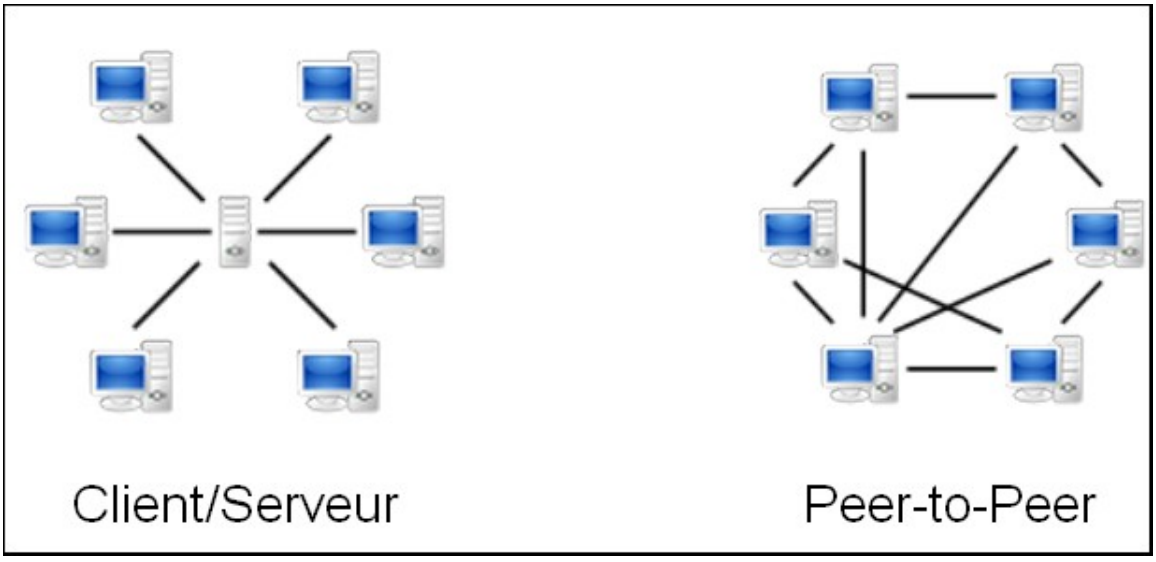

FIGURE 7.1 – Comparaison client/serveur et P2P

Nous allons maintenant présenter P2PBluetooth, un logiciel de partage de fichier sur téléphone portable, créé par 3 chercheurs italiens.

L'idée était de créer un logiciel P2P capable d'échanger des données (photo, vidéo, son) automatiquement (sans que l'utilisateur n'ait d'action spécifique à faire) et à tout moment dès que 2 téléphones sont suffisamment proches pour échanger leurs données via bluetooth,wifi ou autre.

Un des gros problèmes était le faible stockage des batteries de téléphone portable, il fallait donc pouvoir mettre en place un système qui ne serait pas trop gourmand en énergie. Face à cette constatation, la meilleure solution à prendre était d'opter pour une communication bluetooth, pouvant atteindre un débit 1Mb/s et jusqu'à 10 mètres de distance, et surtout beaucoup moins consommatrice en énergie qu'une communication wifi.

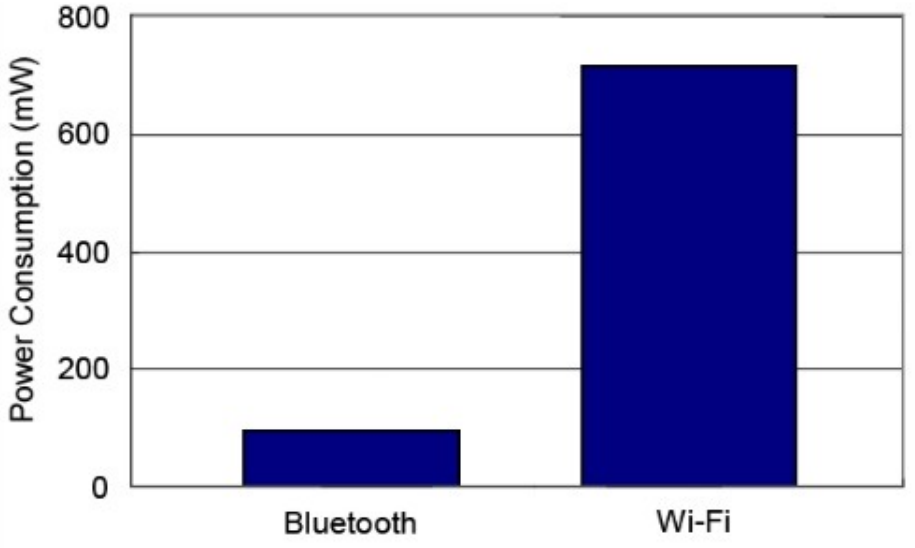

FIGURE 7.2 – Comparaison de la consommation de la batterie

Les logiciels P2P créés pour les ordinateurs ne conviennent pas ici car il y a plus de contraintes pour un logiciel sur téléphone portable (processeur moins performant, moins de mémoire disponible, moins de bande passante), il faut donc créer un nouveau logiciel, le P2PBluetooth.

Ce logiciel doit pouvoir localiser le fichier voulu par le client dès qu'un serveur (un autre client partageant ses fichiers) ayant ce fichier se trouve à proximité, pour ensuite pouvoir le télécharger. Il devra également autoriser, mettre à jour, prioriser et ordonnancer l'accès aux ressources internes, donc aux fichiers partagés par le client.

Le plus efficace en P2P est d'avoir un serveur de stockage et un serveur index permettant de localiser les fichiers voulus, cependant dans notre cas cette solution est impossible dû au faible espace mémoire des téléphones et à la faible distance de fonctionnement du logiciel.

On en déduit que le principe du P2PBluetooth sera que chaque mobile envoie sa liste de fichiers partagés, et ceux qu'il recherche, en Bluetooth, et qu'il faudra donc dès qu'un client proche est détecté parcourir tous ses fichiers partagés pour voir s'il y a un des fichiers recherchés.

On a également un mécanisme de gestion d'erreur dans ce logiciel qui fonctionne sur le principe d'un temps limite pour recevoir, si ce temps est dépassé alors le logiciel met fin au transfert et remet le fichier dans la liste des fichiers voulus.

Conclusion

Lors de ce semestre, nous avons donc posé les bases d'un vaste projet, qui pourrait permettre de développer une application très intéressante, et qui pourrait très bien être utilisée dans la vie courante.

La possibilité de pouvoir prendre et partager des images en temps réel, et d'uploader ces images sur un serveur, nous semble être quelque chose d'utile, et passionnant à utiliser.

Bien évidemment il n'a pas été facile de comprendre et d'identifier les problèmes que ce projet nous a amenés, cependant en effectuant de nombreuses recherches et en utilisant la documentation fourni, nous avons pris la mesure de la tache qui nous attendait tout au long du développement de ce projet.

Pour ce qui est du travail réalisé, de nombreuses tâches étaient à effectuer afin de pouvoir prétendre à la création d'une application fonctionnelle.

En effet ce type de projet faisant appel à de nombreux domaines tels que le traitement de l'image, les langages de programmation, mais aussi les réseaux de communications, nous ne nous sommes consacrés qu'à une seule partie de ce vaste projet.

Nous avons principalement axé notre travail sur l'établissement d'une connexion entre deux interlocuteurs. Le type de protocole fut un choix assez difficile à prendre, du fait des nombreux domaines d'applications dans lesquels ils s'utilisent. Nous nous sommes finalement tournés vers un protocole UDP.

Ce choix de connexion nous a permis par la suite, d'apporter des améliorations sur la gestions des erreurs dans une connexion. Ce point du projet fut plus intéressant que ce que nous le pensions, et nous remercions d'ailleurs Mr. Delalandre pour la qualité des livres et des articles fourni pour traiter cette partie du projet.

Seulement une petite partie du projet à été effectuée, cependant il reste encore de nombreuses tâches à effectuées, comme par exemple l'installation d'une connexion TCP, qui serait plus efficace que l'UDP pour le transfert d'images ; l'étude de l'envoi d'une image de grande tailles, mais aussi le développement d'une application bluetooth, afin d'être plus économe sur la batterie.

Comme on le constate ce projet possède de nombreux domaines à exploiter, et mérite d'être reconduit l'année prochaine, afin d'y apporter des améliorations et de faire avancer un peu plus la création d'une application fonctionnelle.

Annexe : Utilisation de notre programme pour transmettre une image

Bien que nous n'ayons pas eu le temps d'aborder l'adaptation de notre programme pour pouvoir transmettre des données images au sein du réseau, nous tenons à laisser derrière nous une liste ordonnée des tâches à effectuer :

Il faudra d'abord établir une connexion avec un protocole UDP, que nous avons réalisée, pour :
- Envoyer les données pertinentes d'une image ayant été prise(Identifiant, Horodatage, Position GPS, Entropie, etc)
- Le serveur UDP devra recevoir et stocker ces données, puis les traiter pour déterminer les photos les plus intéressantes
- Le serveur enverra ensuite une requête UDP aux clients possédant les photos qu'il a sélectionnées

Puis une fois que le client à reçu la demande du serveur, il transmet son image au serveur mais cette fois grâce à une connexion avec un protocole TCP afin d'avoir un transfert entièrement sûr.

Les étapes à effectuer seront :
- Calculer le nombre de paquets TCP à envoyer suivant la taille de l'image
- Le client envoie son image au serveur avec le nombre de paquets TCP nécessaire
- Le serveur redistribue l'image à tous les clients présents sur le réseau

Partage de données images au sein d'un réseau mobile massivement utilisateurs

Département Informatique
3e année
2012 - 2013

Rapport Projet Système d'Exploitation

Résumé : Ce projet a pour but la création d'une communication, à l'aide d'un protocole UDP, entre 2 personnes. Elles pourront ainsi s'échanger des données images, tout en garantissant une gestion d'erreurs et une optimisation sur mobile

Mots clefs : Protocole, communication, UDP, mobile

Abstract: This project aims to create a communication using UDP protocol between two people. They will be able to exchange image data, while ensuring error management and optimization on mobile.

Keywords: Protocol, communication, UDP, mobile

Encadrants
Mathieu DELALANDRE
mathieu.delalandre@univ-tours.fr

Université François-Rabelais, Tours

Étudiants
Anthony BRAIN
anthony.brain@etu.univ-tours.fr
Baptiste JAECKERT
baptiste.jaeckert@etu.univ-tours.fr

DI3 2012 - 2013