Création d'application mobile

Rapport du projet de Réseau et système avancé

Auteurs :
M. Henry Lagarde
M. Khaled Hajjar

Encadrants :
M. Rémi Badonnel
Mme. Isabelle Chrisment

Table des matières

Table des figures

Introduction

Le Trivial File Transfer Protocol, abrégé en TFTP, est un protocole très simple de type client/serveur, qui permet de gérer le transfert de fichiers au sein de réseaux composés d'ordinateurs. Une première spécification de ce protocole a été publiée au mois de juin 1981 dans la RFC 783. La version actuelle date de 1992 de la RFC 1350 sous une mouture améliorée. Le protocole TFTP repose sur un autre protocole très simple, le protocole de transport UDP (User Datagram Protocol), qui permet d'effectuer un transfert de données sans connexion fixe entre les partenaires de communication. L'implémentation de TFTP sur la base d'autres protocoles est à la fois possible et envisageable.
Ce protocole, qui repose sur l'envoi de paquets de données, s'inscrit dans la famille des protocoles TCP/IP, et a été conçu de telle sorte qu'il reste le moins encombrant et le plus léger possible à l'implémentation.
Dans notre travail nous avons donc implémenté un serveur ainsi qu'un client TFTP en accord avec la version actuelle (RFC 1350), ce serveur peux fonctionner en IPv4 et IPv6.

Chapitre 1

Partie client

Le client peux effectuer deux types de requêtes vers le serveur :

— RRQ (Read Request) : Permettant de lire un fichier présent sur le serveur (création d'un fichier .client contenant l'ensemble des données) ;

— WRQ (Write Request) : Permettant d'écrire un fichier présent sur la machine client sur le serveur (création d'un fichier .server contenant le fichier transmis).

1.1 Structure d'une requête

Une requête côté client d'après la version RFC 1350 suit la structure suivante (visible en Figure 1.1 :

— Type : codé sous 2 octets (OPCODE), permettant de déterminer le type de requête à effecture (RRQ = 01, WRQ = 02)

— Nom du fichier : codé sous n octets

— un octet nul séparant les données

— Mode de transmission : codé sous 6 octets (ici octet)

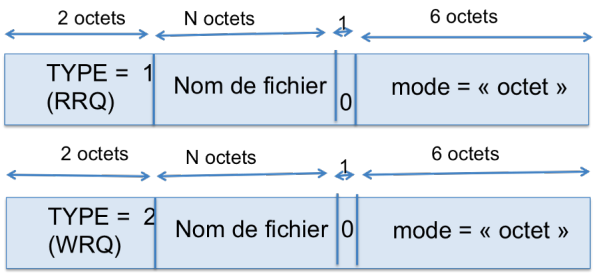

FIGURE 1.1 – Structure requête

Les messages sont par la suite découpés en blocs de 512 octets maximum sauf le der-

nier bloc qui contient entre 0 et 511 octets. La structure diffère légèrement de la requête précédente (voir Figure 1.2) :

— Type : codé sous 2 octets (OPCODE), ici Type = 03 indiquant qu'il s'agit de la donnée qui est transmise ;
— Numéro de Block : codé sous 2 octets, ce numéro permet de reconstruire le fichier découpé et de vérifier la bonne réception du block grâce à l'aquittement (ACK) qui utilisera ce numéro ;
— Données : codées sur un maximum de 512 octets.

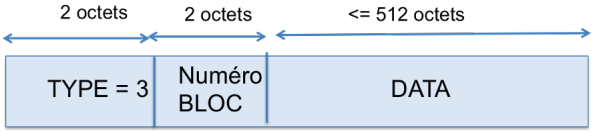

FIGURE 1.2 – Structure bloc de donnée

1.2 Simulation pertes d'aquittement

Afin de simuler la perte d'aquittement (ACK), un système aléatoire a été mis en place. Nous générons un nombre aléatoire et regardons sa parité.

— Nombre pair : On saute l'envoie de l'ACK simulant ainsi la perte de celui-ci ;
— Nombre impair : On valide la réception de l'ACK.

Si un ACK n'est pas reçu dans le temps imparti (5 secondes) le serveur produit un TIMEOUT, nous tentons donc de renvoyer l'ACK (5 tentatives). Si après ces tentatives aucun ACK n'a atteint le client, nous coupons la connection entre le client et le serveur.

Chapitre 2

Partie serveur

2.1 Particularités du serveur

Le serveur, en cas de demande de lecture (RRQ) ne renvoie au client que les données demandées. Nous avons, dans notre algorithme, tout d'abord testé que le fichier existe bien puis envoyé les données par paquets de 512 octets sauf le dernier paquet ayant une taille comprise entre 0 et 511 octets.
Lors d'une demande de l'écriture (WRQ), le serveur renvoie cette fois ci uniquement les aquittements de bonne reception des packets. Le premier aquittement correspond à la reception de la demande faite par le client. De la même manière que pour la demande précédente, nous testons l'existence du document mis à part que cette fois-ci nous retournons une erreur dans le cas de l'existence de document. Nous vérifions également que les accès soit adéquats à la modification de ce même document.

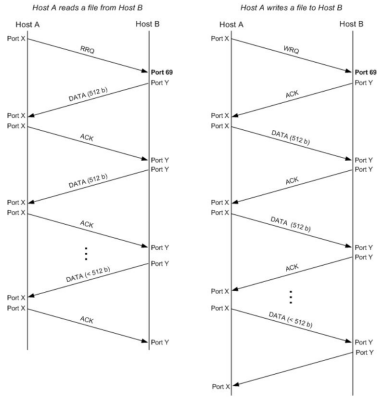

FIGURE 2.1 – Réponses du serveur au client pour les demandes RRQ et WRQ

2.2 Messages envoyés par le serveur au client

Le serveur peut envoyer différents types de messages au client en gardant la structure décrite dans la partie 2.

— Des acquittements indiquant la bonne réception de la requête du client ;
— Des requêtes contenant les données demandées par le client ;
— Des messages d'erreurs (Absence d'un fichier, fichier déjà existant).

Ces messages sont par la suite traité par le client en fonction de l'OPCODE présent en tête de la requête.

2.3 Simulation de pertes d'acquittement

La simulation de pertes d'acquittement a été implémenté de la même manière que du côté client, permettant ainsi de simuler des pertes dans les deux sens.

2.4 Erreurs que peut retourner le serveur

Notre serveur teste un bon nombre de variables et retourne une erreur dans le cas ou il existe un soucis au niveau du serveur ou bien des demandes du client comme nous l'avons vu précédemment.
En effet, dans un premier temps, nous testons le lien entre la socket et l'adresse IP du serveur qui renvoie une faute en cas de mauvaise connexion.
De plus, notre algorithme contôle également si les paquets envoyés par le serveur, pour les demandes de lecture, sont bien envoyées.
Notre code permet, de plus, d'examiner la taille des paquets afin que ceux-ci n'ai pas une taille différente de celles acceptées.
Enfin notre code détecte les correspondances ou bien les différences entre la demande et la structure du paquet.

Conclusion

Ce sujet nous a ainsi donc permis d'implémenter un système de transfert de fichier, complétant ainsi les TP effectués, ainsi que nos expériences dans le domaine du réseau. Ce projet fut un bon moyen d'acquérir de l'expérience dans la transmission de donnée.