LINGI1341 - Projet TRTP

Constantin de Schaetzen Jean Gillain

22 octobre 2018

1 Introduction

Ce document est un rapport détaillant certains aspects de notre projet pour le cours de réseaux informatiques. Il tend à préciser certains choix de conceptions et montre en quelques lignes le déroulement du programme. Pour toutes informations concernant l'utilisation du programme et son lancement en ligne de commande, rendez-vous dans le fichier Readme.

2 Structure générale

Vous trouverez dans le dossier /src deux fichiers avec l'extension '.c'. Il s'agit de sender.c et receiver.c qui sont les deux fonctions à appeler pour utiliser le chat. Vous trouverez dans ce même répertoire un autre dossier, /tools, contenant les différentes fonctions relatives aux sockets, aux paquets et aux listes chaînées que nous avons implémenté. Enfin, un dernier dossier /tests contient différentes fonctions qui permettent de tester certaines fonctionnalités de notre programme.

2.1 Déroulement du programme

2.1.1 Sender.c

Le fichier sender.c gère l'envoie de données vers le receveur. Nous utilisons le sliding window protocol et le selective repeat. La fenêtre est représentée par une liste simplement chaînée et sa taille maximale est fixée à 32. Pour transmettre les datagrammes, nous utilisons une socket. Le sender procède en trois étapes. Dans un premier temps, il cherche à lire l'information et procède de la manière suivante :

Si il y a de la place dans la liste :

- récupération d'un morceau d'au plus 512 bytes (le payload)
- création d'un paquet et encodage du header. Le type du paquet est initialisé à P_TYPE_DATA et l'attribut *length* est fixé à la taille du morceau récupéré. On fixe un numéro de séquence dans l'intervalle [0,255] - le numéro de séquence est une variable de notre programme et est incrémentée à chaque attribution (on utilise modulo pour rester dans la bonne gamme).
- encodage du paquet (header + payload) dans une chaîne de caractère et écriture sur la socket.

Lecture sur la socket (si il y a quelque chose d'écrit) :

- Décodage de la chaîne de caractère afin de constituer un paquet.
- Evaluation du type du paquet (ACK ou NACK)

  ACK : Si c'est un ack, cela signifie que le feedback est positif. Deux cas de figures peuvent toutefois se produire. Imaginons que nous ayons reçu un ack avec comme numéro de séquence 3. Cela signifie que le receveur attend le paquet contenant le numéro de séquence 3. Mais a-t-il

  NACK : A la réception d'un NACK, le sender va chercher le paquet en question dans la liste qu'il entretient. Lorsqu'il le trouve, il met l'attribut timestamp à jour et renvoie le paquet. Dans le même temps, il supprime chaque paquet contenant un numéro de sequence inférieur à celui du paquet non-acquis, ceux-ci ont bien été récupéré par le receiver.

Routine time out Enfin, la troisième et dernière étape du sender est de lancer une routine afin d'évaluer quels paquets ont expiré. Cette routine remplie sa mission en comparant la valeur de l'attribut timestamp avec "l'heure" actuelle. Si la différence est supérieure à la valeur du retransmission timeout alors le paquet doit être ré-envoyé (avec une mise à jour de son timestamp). La routine boucle sur la liste tant que le paquet rencontré a expiré. Puisque les paquets sont ajoutés chronologiquement, les paquets sont triés du plus ancien au plus récent.

2.1.2 receiver.c

La fonction receiver est assez simple. Le receiver entretient également une liste de paquet ainsi qu'une variable contenant le paquet attendu.

Elle cherche à lire de l'information sur une socket. Dès qu'un client est trouvé, le receveur est connecté à son sender. Ensuite, on procède comme suit :

- Décodage de la chaîne de caractère afin de créer un paquet.
- Si l'attribut tr du paquet est à 1, cela signifie que le paquet est tronqué. On ne peut pas envoyé un nack pour ce paquet. En effet, puisqu'il est tronqué, les données qu'il contient ne sont pas fiables et par conséquent le nack serait une potentielle source d'erreur.
- Si le paquet n'est pas tronqué, le receiver regarde si le numéro de séquence du paquet est celui attendu.

Si tel est le cas :

- Le receiver récupère le payload et le dirige sur la sortie standard puis, si l'option -f est mentionnée, redirige le flux vers le fichier spécifié.
- envoi d'un ack contenant le timestamp et le numéro de séquence de la donnée reçue.
- incrémentation de la variable indiquant le paquet attendu
- parcours du buffer pour identifier si le paquet attendu si trouve. Si tel est le cas, les trois opérations ci-dessus sont d'application et la place du paquet est libérée.

Si ce n'est pas le cas :

- Le paquet est stocker en queue du buffer.
- Un Nack contenant le nuémro de séquence de la donnée attendue et le timestamp du paquet reçu.

2.2 Choix de programmation

Quelle valeur avons-nous choisi pour le retransmission timeout ? Puisque l'émission d'un paquet met entre 0 et 2000 millisecondes nous avons décidés de mettre le timer a 4500 millisecondes afin de laisser le temps au receveur de renvoyer un ACK si le transfert s'est correctement déroulé. Une décision par calcul aurait toutefois été plus pertinente mais nous avons décidé de nous fier aux spécifications du projet.

Que mettre dans le champs Timestamp, et quelle utilisation en faire ? Dans le champ timestamp, nous mettons l'heure à laquelle le paquet a été émis. Ce choix a deux intérêts. Premièrement, il nous permet de gérer notre timer car c'est grâce à cet attribut que nous pourrons savoir si le paquet a expiré. Deuxièmement, il nous permet de gérer la réception d'ACK et de NACK. Grâce à lui, nous disposons d'un deuxième moyen de vérification qui nous permet en outre de déduire quel packet a généré l'envoie de l'ACK ou du NACK et de gérer notre liste en conséquence.

Pourquoi une liste et pas simplement un buffer

2.3 Tests

A ce stade, nous n'avons pas pu implémenter autant de tests que nous l'aurions voulu. Nous avons pu tester le fonctionnement des fonctions permettant d'implémenter la liste et les noeuds qui la composent mais les fonctions sender et receiver n'ont pu être testées que via les scrits mis à notre disposition par les assistants et par appel successif de sender et receiver. Toutefois, notre programme semble fonctionné correctement.

2.4 Performance

2.5 Conclusion

En conclusion, nous pensons remettre une implémentation correcte pour répondre à la problématique. Malheureusement, nous n'avons pas pu tester énormément notre programme et il y a certainement de nombreux cas limite qui pourrait être mieux géré. C'est l'objectif que nous avons d'ici la deuxième soumission. Aucun changement majeur ne sera fait puisque ce n'est pas le but mais nous aimerions rendre notre programme plus robuste.