LINGI 1341 - Computer network : information transfer
Implémentation d'un protocole de transfert de données sans perte

Crochet Christophe
Raquet Damien

November 2017

1 Introduction

Pour ce travail, il nous a été demandé d'implémenter un protocole de transfert de données. La stratégie que nous devions utiliser était celle du selective-repeat. Le programme à rendre devait créer deux exécutables, un receiver et un sender l'un pour recevoir des données, l'autre pour les envoyer. Ces deux exécutables devait être interopérable.

2 Architecture

Notre programme se divise en plusieurs fichiers. Les cinq fichiers wait_for_client.c, create_socket.c, read_write_loop.c, real_address.c et packet_implement.c sont les fichiers qui remplissent les taches inginious "envoyer et recevoir des données" et "segment de données". Nous les avons légèrement adaptés pour le bon fonctionnement de notre code.

Les deux fichiers sender.c et receiver.c sont les fichiers qui contiennent le reste du code. Le début de chaque exécutable est semblable. Nous récupérons d'abord les arguments passé au programmes, puis nous récupérons une adresse valide a l'aide des arguments. Une fois que nous avons une adresse valide, nous créons une socket. La si c'est le sender qui appelle create_socket, la socket est juste connectée au port, tandis que si cette fonction est appellée par receiver, la socket est juste liée. Ensuite, nous vérifions si les données a envoyées sont à récupérées depuis un fichier, ou depuis la sortie standart.

Une fois que les données et structures de base sont initialisées, les fonctions en selective repeat sont appellées. Ces deux fonctions, sender_SR et receiver_SR

sont les deux fonctions principales du protocole. La fonction sender, s'occupe de diviser en pacquets les données à envoyer sur le socket. Ces paquets sont ensuite lu par le receiver qui extrait les données du fichier ou de la sortie standart du paquets, avant de renvoyer un paquets d'acquittement, permettant au sender de savoir que x paquets sont bien arrivés. Les paquets perdus sont gérés avec une intervalle de temps, si un paquets est depuis trop longtemps sur la socket, il est considéré comme perdu, et le receiver envoie alors un paquets spécifique pour que le sender renvoie le paquet perdu. Une fois tous les paquets envoyés, le sender envoie un paquet vide signifiant la fin du transfert. Une fois que le receiver a recu le paquet de déconnexion, celui-ci envoie un paquet de déconnexion pour le sender. Le sender ne se déconnecte qu'a la réception de ce paquet, ou lorsque la fonction renvoie -1, signe que le receiver s'est arrêté.

note sur le Timestamp. Nous n'utilisons pas ce champ dans notre implémentation, car nous n'en avons pas vu l'utilité.

3 Résultats obtenus

En pratique, nous sommes arrivés à faire fonctionner le programme avec des fichiers de tailles variables. Cependant, nous n'avons pas testé si notre implémentation fonctionnait dans tous les cas critique décrit plus bas.

De plus, ayant quelques soucis d'organisation, nous n'avions pas terminé le programme a temps pour pouvoir faire des tests d'interopérabilité. nous ne savons donc pas comment réagirait notre programme avec d'autre exécutables. Cependant, il est probable que la section ou nous gérons les paquets perdus avec le time

4 Tests

Etant donné que nous n'avions pas terminé dans les temps, nous n'avons également pas pu créer des test efficace et utiles. Nous avons cependant pensé à plusieurs cas critiques qui pourraient faire planter le programme.

Premier test : un test avec un cas idéal, c'est à dire, aucune perte, ni corruptions de données.
Second test: un test avec des paquets qui sont régulièrement perdus. Permet de vérifier si le sender gère correctement les paquets du receiver et réagit en conséquence.
Troisième test : un test avec des données corrompues. Cela se rapproche assez du test 2, entendu que le sender doit réagir de facon très similaire : il réceptionne le message d'erreur du receiver et renvoie le paquet corrompu.
Quatrième test : un quatrième test serai de vérifier avec un délai assez long, afin de vérifier que le receiver n'interpète pas un paquet qui mettrai du temps à

arriver comme étant perdu. ce qui pourrait mener à une duplication de paquets. Cinquième test : il pourrait également être intérressant de tester plusieurs types d'erreurs en même temps, afin de s'assurer que le sender et le receiver n'envoie pas plusieurs fois un même paquet, ou qu'il n'y ait des souci de compatibilité entre les gestions d'erreurs.