Rapport 'Réseau'

Bontemps Martin - 59451200, Pignolet Aurélien - 75281200

31 October 2014

1 Introduction

L'objectif de ce projet était de réaliser un transfert fiable d'information suivant un protocole "Selective Repeat". Nous avons donc créé deux entités, un "sender" et un "reciever", capables de s'envoyer de l'information malgré les corruptions et pertes de packets simulées. Notre programme est à même de s'envoyer des données de nombreux types, tels que txt, jpeg, mp3, avi, ...

2 Algorithme général

Le sender envoie les éléments contenus dans sa fenêtre en attendant de recevoir des "acknoledgements". Dès que le receiver reçoit un 'packet' (dont le CRC est bon), il envoie un acknoledgement correspondant au dernier packet non corrompu en séquence et stocke ce packet dans un buffer. Cette façon de faire permet de prendre en compte les packets hors séquence ainsi que les packets corrompus. Une fois que le sender reçoit un acknoledgement il n'envoie plus que les packets se trouvant à la suite de celui venant d'être acquitté. Dès que tous les packets d'une fenêtre ont été acquittés, le sender lit la suite du fichier et recrée un tableau, tandis que le receiver écrit le contenu de son buffer avant de le vider. On répète cette séquence jusqu'à la fin du fichier.

3 Limitation d'implémentation

Notre programme reste perfectible, certains problèmes étant encore présents

- Comme nous déterminons le nombre de packets en fonction de la taille du fichier à envoyer, il nous est impossible d'envoyer l'entrée standard car nous ne pouvons pas connaitre sa longueur.
- Si le sender doit envoyer un multiple de 256 packets, nous ne parvenons pas à donner des conditions d'arrêt dans nos boucles qui continueront à l'infini.

De plus, notre algorithme n'est pas exactement un selective repeat. En effet, certaines hypothèses ont été prises pour facilité l'implémentation du programme. Comme par exemple le fait qu'un sender doive attendre d'avoir recu le dernier acknoledgement d'une fenêtre pour mettre cette denière à jour.

4 Sender

Le sender lit un fichier "fileName", remplit les données dans un tableau de structure "packet" (ce tableau peut contenir au maximum 32 packets) et y calcule un CRC. Ces packets ont des numéros de séquence compris entre 0 et 255. On envoie les packets contenus dans le tableau sur le port "port". Lorsque le sender reçoit un acknoledgement pour un numéro de séquence ("seqnum"), il renvoie tous les packets ayant un numéro de séquence supérieur à seqnum compris dans le tableau. Si l'acknoledgement reçu correspond au dernier packet de la "window", le programme avance et recommence l'opération jusqu'avoir envoyé toutes les données. Notre programme fonctionne en "fast retransmitting", c'est-à-dire qu'il envoie tous les packets compris dans le tableau tant qu'il ne reçoit pas d'acknoledgement pour le mettre à jour. Remarque: le fichier est lu au fur et à mesure qu'il faut remplir les structures afin de ne pas surcharger la mémoire. Remarque bis: Les arguments "sber" et "splr" permettent de simuler une corruption ou une perte de packet.

5 Receiver

Le receiver lit le port "port". Si une structure est en attente, il vérifie son CRC et son numéro de séquence. Si les deux sont corrects, il met à jour son dernier bon packet reçu, le stocke dans un buffer et envoie un acknoledgement. Si le numéro de séquence est incorrect, il le stocke dans le buffer au bon endroit et envoie un acknoledgement de dernier packet bien reçu mais sans mettre à jour ce dernier. Par contre, si le CRC est incorrect, il enverra juste un acknoledgement du dernier packet bien reçu. Une fois que sa window est pleine il l'écrit et la vide jusqu'avoir reçu les dernières données.

6 Tests

6.1 1^{er} Test

Le premier test que nous avons effectué n'était pas convaincant. En effet, le CRC calculé par le receiver était tout le temps différent de celui envoyé par le sender. Cela provoquait une boucle infinie (dans les 2 combinaisons possible). En effet, le sender envoie des packets mais ne reçoit jamais d'acknoledgement, il continue donc d'envoyer sa window à l'infini. Comme nous calculons le CRC sur les 516 premiers bytes de la trame, nous en avons conclu que l'autre programme ne calculait les données que sur les datas. Nous avons décidé de ne pas implémenter

notre programme de cette façon afin de pouvoir remarquer quand le numéro de séquence est corrompu.

6.2 Autres tests

Nous avons également testé les autres exécutables sur le studentShare cependant il est apparu pas mal de problèmes:

- Certains exécutables font des "seq. fault" lorsqu'ils sont lancés avec une de nos entités.
- La façon dont la fenêtre/le numéro de séquence est calculé/utilisé peut provoquer une boucle infinie.
- De plus, nous calculons un CRC sur nos acknoledgement afin de gagner en sécurité, ce qui n'est pas le cas des groupes avec lesquels nous avons fait des tests.

Nous pouvons conclure que les tests effectués n'ont pas été très concluants. Cela est dû en grande partie aux choix d'implémentation différents des personnes avec lesquelles nous avons essayé d'interagir.

7 Conclusion

Nous avons créé un programme permettant d'envoyer de l'information de façon sécurisée entre deux entités en suivant l'algorithme "selective repeat". Cependant certaines restrictions sont présentes, à savoir, l'impossibilité de lire l'entrée standard et l'impossibilité de gérer un fichier dont la taille est comprise entre [(n-1*512*256;n*512*256] avec $n \in R_0^+$.

8 Sources

http://manpagesfr.free.fr/man/man3/getaddrinfo.3.html
http://beej.us/guide/bgnet/output/html/singlepage/bgnet.htmlsendtorecv
http://www.barrgroup.com/Embedded-Systems/How-To/CRC-Calculation-C-Code