LINGI1341 - Computer networks : information transfer

Projet 1 - Rapport

Guillaume Reginster [10591500]
François Duchêne[14241500]

Année académique 2019 - 2020

1 Introduction

A la demande de l'entreprise "Intelligent Data for the Internet Of Things" il nous a été demandé d'implémenter un sender fiable. Ce sender implémente un protocole particulier, le TRTP ainsi que la stratégie du selective repeat et l'acceptation de paquets spéciaux indiquant une congestion dans le réseau. Nous allons vous décrire dans les prochaines pages l'architecture de notre programme ainsi que nos différentes stratégies utilisées pour finir sur les résultats des tests réalisés.

2 Architecture générale du programme

En addition avec le Makefile, notre programme est composé de deux dossiers principaux : src et tests. Le dossier "tests" contient plusieurs fichiers c permettant de tester différentes parties du programme. Le dossier src contient tous les fichiers permettant l'exécution du dit programme. Développons donc son contenu.

Tout d'abord, nous avons le fichier "packet_implem.c" associé à l'interface "packet_interface.h". Ces derniers nous permettent de créer des paquets mais également de les encoder pour les envoyer sur le réseau et de les décoder, ce qui nous sert lors de la réception de aks. Nous y trouverons également les fonctions "get" et "set" pour chacune des variables propres à la structure "pkt_t".

Ensuite, nous avons le fichier general.h qui contient simplement un listing des différentes erreurs possibles que nous pourront rencontrer lors de l'exécution du programme.

Par après, les fichiers "init_connexion" permettent la création des sockets ainsi que la récupération de l'adresse du destinataire. Ces fonctions seront appelées au commencement du programme.

Ainsi, le fichier nommé "sender.c" permet de lancer l'exécution du programme. Il lit et traite les arguments entrés en ligne de commande, appelle les fonctions de "init_connexion" puis lance la fonction principale du fichier "read_write_loop_final".

Quant à lui, "read_write_loop_final" se charge de toute l'exécution en stratégie sélective repeat. Il lit le fichier (ou le chat), envoie les paquets au destinataires, traite la réception des acks, ainsi que les réenvois de paquets et met fin à la connexion. Dans sa tâche, il appellera les trois derniers fichiers de notre dossier src qui contiennent quant à eux différentes fonctions utilitaires :

— Premièrement, dans un soucis de lisibilité, "pkt_builder" sert simplement de transition lors la création de paquet en appelant les fonctions set.

— Par après, les fichiers "window" permettent de mettre à jour les indices de la fenêtre active en fonction des acks reçus.

— Finalement, le dernier fichier du programme nommé "linked_list.c" est composé de différentes méthodes utiles à la gestion de queues. Ainsi, cette file permet de stocker les paquets lus et envoyés dont les acks

n'ont pas encore été reçu. Ceci dans le but de les réenvoyer lorsque leur timer, également stocké dans les noeuds de la queue, dépasse la limite fixée.

3 Sender

1. Gestion du timestamp :

   Dans notre programme, à chaque création de paquet, nous initialisons le timestamp à 0 mais nous n'utilisons jamais ce timestamp par la suite. En effet, nous ne lui avons pas encore trouvé d'utilité dans notre implémentation actuelle.

   Cependant, nous avons pensé à une piste d'amélioration pour la suite qui utiliserait ce champs : il s'agirait de stocker un ratio contenant le nombre total de paquets à envoyer (estimé par le sender) ainsi que le numéro du paquet envoyé. Ce faisant, il serait possible d'estimer l'état d'avancement de l'envoi de données.

2. Réception de nack :

   Lors de la réception d'un nack, nous vérifions tout d'abord que le numéro de seqnum est valide. Si tel est le cas, nous lançons un "sleep" d'une durée aléatoire. Le temps minimal de l'attente est de une seconde tandis que le temps maximal est de quatre secondes. Comme le délai de retransmission est de une seconde, les timers de retransmission des paquets auront expiré une fois le sleep terminé. Par conséquent, passé ce délai, le programme reprendra son exécution en retransmettant tous les paquets dont les acks n'ont pas encore été reçu.

3. Valeur de retransmission :

   Notre valeur du timer de retransmission est de une seconde. Nous avons choisi cette valeur de timer parce qu'une seconde nous paraissait suffisant sans être excessif.

4. Stratégie en cas de difficultés d'ouverture de connexion :

   — Si notre programme reçoit un paquet de type NACK, il réagit, comme dit précédemment, en attendant plusieurs secondes avant de réessayer d'envoyer des paquets.

   — Si notre programme ne reçoit aucune réponse, il agira comme expliqué plus loin lors de la stratégie de fin de connexion : il renverra sa fenêtre d'envoi trois fois entre chaque timeout. Passé ce délai, il fermera la connexion à défaut de réponse du receiver.

3.1 Gestion de la mémoire

Nous n'allouons pas énormément de mémoire pour le programme. Pour donner une estimation, pour l'envoi d'une image de $100ko$, le programme utilise $200ko$ de mémoire au **total**. Au cours de l'exécution, ce nombre est bien plus petit, ce qui fait que le programme est assez léger à utiliser. Lors de l'un des tests où nous avons envoyé un film de $4Go$, seuls $200Mo$ étaient alloués en même temps, ce nombre restant relativement constant au cours de l'envoi.

Nous avons eu quelques problèmes avec valgrind nous indiquant que nous perdions 0 bytes sur 32 blocks. Nous savons quelle partie du code est la cause de cette "memory leak", cependant, nous ne comprenons pas pourquoi cela agit ainsi, d'autant plus qu'il n'y a apparement aucun byte perdu. Nous espérons dès lors que cela ne compte pas comme une memory leak.

4 Questions particulières

1. Partie critique de notre implémentation :

   La partie critique se trouve lors de la gestion de la retransmission de paquets. En effet, notre sender attend d'avoir envoyé toute sa fenêtre et même si il reçoit des acks lui disant qu'il ne retransmet pas le bon paquet, il attendra tout de même que le timer de retransmission du paquet le plus ancien envoyé expire. A ce moment là, il retransmettra l'entièreté de la fenêtre.

   Une façon plus intelligente de procéder consisterait à renvoyer un paquet non retransmis dès réception d'un ack non attendu. Cependant, comme l'implémentation d'un tel protocole de retransmission était prohibé dans les consignes du projet, nous ne l'avons pas implémenter. Nous pensons toutefois que cela améliorerait grandement les performances de notre sender.

2. Fermeture de connexion :

   Plusieurs cas peuvent provoquer une fermeture de connexion :

   — Tout d'abord, si nous renvoyons trois fois les mêmes paquets sans qu'aucune modification n'ait lieu, le programme s'arrêtera.

   — Ensuite, dans le cas où la lecture se fait sur le chat, nous pouvons mettre fin à la connexion en envoyer le signal EOF (End-Of-File) en tapant "ctrl+d".

   — Enfin, dans le cas où la lecture se fait dans un fichier, la connexion prendra fin lorsque ce dernier aura été entièrement lu et que nous nous serons assurés que tous les paquets envoyés aient été correctement reçus.

3. Performances :

   Mémoire Nous pouvons calculer une estimation de la mémoire totale utilisée : la partie fixe vaut environ 1560 bytes, pour une fenêtre constante de 30 paquets à envoyer, cela nous fait 30 fois 530 bytes soit

15900 bytes en faisant l'hypothèse que tous les paquets soient complets. Au final, nous utilisons environ 17500 bytes en permanence.

4. Stratégies de tests :

   Nous avons deux catégories de tests, les tests unitaires et les tests black-box. Les premiers testent certaines parties de notre implémentation et vérifient que le comportement des fonctions est bien celui attendu. La deuxième catégorie emploie directement l'exécutable de notre sender ainsi que plusieurs implémentations du receiver ; celui de référence d'une part et les receiver d'autres groupes que nous avons reçu lors de la séance d'interropérabilité d'autre part. L'utilisation de linksim afin de simuler des liens dans des conditions imparfaites était évidemment primordial.

5 Séance d'interropabilité

Nous avons effectué des tests d'interropérabilité avec 3 autres groupes, les groupes 98, 32 et 42. avec lesquels nous nous sommes échangés nos sender/receiver respectifs.

Nous n'avons pas trouvé de défaillances majeures dans notre implémentation lors de ces séances, mais néanmoins plusieurs défauts. Entre autre une memory leak non traitée, des printf qui imprimaient sur stdout au lieu de stderr ainsi qu'un problème dans la gestion du timeout de POLL.

La plus grande défaillance que nous aillons détecté lors de ces séances est lors de fort taux de corruptions de paquets, notre sender a tendance a ne pas retransmettre et à fermer la connexion plus tôt que prévu.

6 Conclusion

En conclusion, nous pensons que notre programme remplit le cahier des charges demandé bien que nous voyons quelques pistes d'améliorations futures comme développé précédemment. Nous déplorons l'étrange présence d'une memory leak de quelques blocs de 0 bytes, mais mis à part ce problème, au vu des tests effectués, tout semble fonctionner correctement.