ÉCOLE POLYTECHNIQUE DE LOUVAIN

LINFO1341 - Computer networks : information transfer

# Rapport Projet TRTP : Truncated Reliable Transport Protocol


Marques Mathias 1318-17-00
Grognard Simon 3781-17-00


29 Mars 2021

# 1 Introduction

Un consortium d'entreprise nous a demandé d'implémenter un protocole de transfert fiable basé sur des paquets UDP, cette méthode est appelée Truncated Reliable Transport Protocol (TRTP). Ce Protocole utilise la stratégie du selective repeat. Ce projet permet également la réception de paquet tronqué c'est à dire des paquets dont le payload a été retiré par le réseaux car se dernier est surchargé. Ce rapport a pour but d'expliquer notre code et de simplifier la lecture de celui-ci en pointant les diverses solutions que nous avons mis en place afin d'accomplir la tâche qui nous a été demandé.

# 2 Fenêtre de réception

Pour manipuler la fenêtre de réception, notre receiver va modifier une variable tout au long du programme soit en l'incrémentant soit en la décrémentant. Lorsque le receiver reçoit un paquet, il va regarder si celui-ci est reçu en séquence, il aura alors deux choix devant lui.
**1. Paquet hors séquence :** Si le numéro de séquence appartient à la fenêtre de réception, alors il va ajouter ce paquet dans une queue (dataReceive) créer au préalable et on décremente la variable window. Cette variable permet de savoir la taille de la queue ainsi que de transmettre les places vides dans celle-ci au sender.
**2. Paquet en séquence :** Dans ce cas présent, le receiver va automatiquement écrire le contenu du payload du paquet sur la sortie standard (qui peut être redirigé vers un fichier). Ensuite, il va parcourir la queue et regarder si celle-ci contient des numéros de séquences attendus pour ne pas corrompre le fichier. Lorsqu'il imprimera ces paquets sur la sortie standard, il décrémentera la queue de ce paquet et augmentera la window.
Notre sender va récupérer la valeur de la window du receiver en recevant l'acquittement du premier paquet (seqnum=0). A partir de ce moment, il va toujours faire attention à ce que la quantité de paquets envoyés en une fois soit inférieur à la taille window du receiver. Pour libérer de la place dans la queue des paquets déjà envoyé, le sender va attendre la réception d'acquittement, comme ces derniers son cumulatif on sait que l'on peut supprimer tous les paquets qui précède le numéro de séquence de l'acquittement reçu.

# 3 Génération des acquittements

Il y a 3 situations différentes dans lesquels on envoie un ack.
1. Lorsque l'on reçoit un paquet en séquence, on va regarder si nous avons déjà reçu les paquets suivants. Nous allons envoyé un acquittement avec le prochain numéro de séquence attendu par le receiver.
2. Deuxième cas de figure nous recevons un paquet hors séquence qui doit alors être rangé dans la queue s'il ne s'y trouve pas encore (même si le paquet reçu est déjà dans la queue nous renvoyons tout de même un acquittement). L'acquittement envoyé dans ce cas de figure contiendra encore une fois le numéro de séquence attendu par le receiver et non le numéro de séquence du paquet reçu.
3. Le troisième cas de figure est relatif à la condition de fermeture de notre programme, c'est pourquoi il est explicité dans le point suivant de ce rapport.

# 4 Fermeture de la connexion

Il y a deux scénario pour le fermeture de notre receiver le premier est qu'il n'aie pas reçu de nouvelles du sender depuis 30 secondes. Si c'est le cas on sort de la boucle qui permet de réceptionner les paquets et on close la socket. Deuxième cas de figure notre receiver à reçu un paquet dont le champs length est mis à zéro. Si ce paquet n'a pas le numéro de séquence que nous attendons nous ignorons ce paquet sinon nous envoyons l'acquittement au sender et on ferme la socket. Pour notre sender la stratégie est un peu différente nous envoyons nos donné jusqu'à la fin du fichier (avec notre dernier paquet qui a un champs length à 0). Une fois que le dataSent (queue qui contient la liste des paquets déjà envoyé qui attendent de recevoir leur acquittement) ne contient plus que ce paquet nous allons compter le nombre de fois ou nous renvoyons ce paquet si nous l'envoyons 10 fois on va considérer que le ack c'est perdu ou a été corrompu par le réseaux et on va fermer le programme sinon il se ferme à la réception de l'acquittement de ce dernier paquet.

# 5 Timestamp et Valeur de retransmission

Pour la valeur timestamp nous n'avons pas réellement utilisé ce champs à des fin utile nous l'avons initialiser à 42 et sa valeur ne bouge pas tout au long de l'exécution du programme mais dans ce rapport nous avons donner dans la section sur les piste d'amélioration de notre programme un idée de l'utilisation du champs timestamp qui pourrait être utile mais que nous n'avons pas eu le temps de mettre en place.
Pour ce qui est du temps de la valeur de retransmission des paquets nous l'avons définis comme 3 fois le delay

que nous utilisons fréquemment lors de nos tests 300ms ce qui nous donne un temps de retransmission de 900ms que nous avons arrondi à 1000ms par soucis d'esthétisme.

# 6 Gérer les PTYPE_NACK

Lorsque notre receiver reçoit un paquet tronqué nous pouvons voir qu'il est tronqué car le réseau a changé le bit tr pour le mettre à 1 nous générons alors un nack qui a pour numéro de séquence le même numéro de séquence que le paquet tronqué reçu. Le sender reçoit le nack et récupère le numéro de séquence à l'intérieur. Ensuite, il va parcourir la queue dataSent (queue qui contient la liste des paquets déjà envoyé et qui attendent de recevoir leur acquittement) pour retrouver le paquet avec le numéro de séquence correspondant et ainsi pouvoir le renvoyer. Et pour finir on réinitialise le timer du paquets afin qu'il ne le renvoie pas en boucle si il n'a pas reçu de ack avant la fin du timer de retransmission.

# 7 Notre receiver est injoignable

Si nous lançons le sender sans receiver pour l'écouter, il est programmé pour s'arrêter lorsqu'il aura envoyé 10 fois le premier paquet sans jamais avoir reçu de signe de vie du receiver. S'il reçoit un ack ou un nack de la part du receiver il y a un receiver et donc il envoie le fichier en entier. Cela permet d'éviter que le sender n'envoie un paquet en boucle sans jamais s'arrêter. Nous avons fait plusieurs test avec cette fonctionnalité et elle n'a jamais empêché le sender de trouvé le receiver si il y en a un.

# 8 Amélioration du programme effectuée

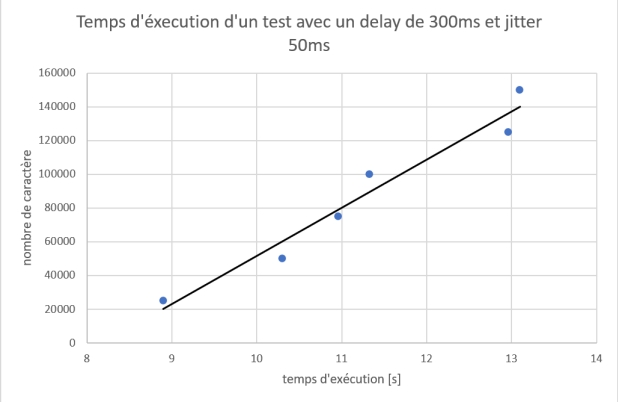
Nous avons rajouter 2 fonctionnalités à notre programme qui n'étais pas demandé dans l'énoncer de base du projet. Ces dernières bien que mineur se sont révélé très utile lors de l'élaboration du projet.
La première fonctionnalité nous a été très utile pour déboguer notre code il s'agit d'un flag que nous avons rajouter au programme le flag "-l" il permet d'afficher les "ERROR" qui servent à déboguer notre code et vérifier si il fonctionne correctement si les acquittement ont les bons numéros de séquence si le programme se termine correctement... Nous conseillons donc vivement l'utilisation de ce flag pour vérifier si il n'y a pas d'erreur lors de la transmission des données.
Le deuxième flag nous a été utile lors de la phase de test pour vérifier si nous pouvions utiliser n'importe quel window dans le receiver (compris entre 0 et 31). Nous avons donc rajouter un nouveau flag à notre receiver le "-w" qui permet de rajouter la taille de la window désiré en indiquant "-w 20" dans les options d'exécutions. Attention la window de base est initialisé à 31 et elle ne peut que supporter une fenêtre entre 0 et 31.

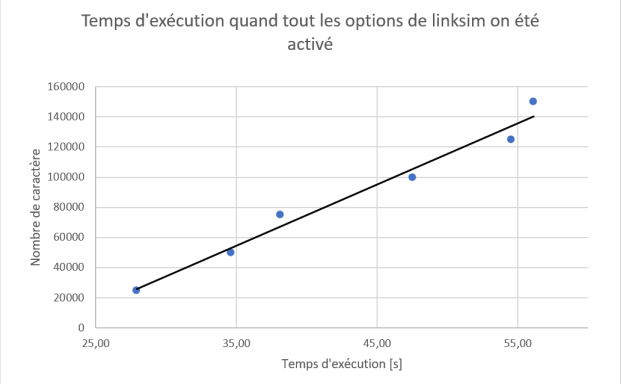
# 9 Piste d'amélioration

Lors de ce projet nous avons commis quelques erreurs. Si nous avions eu plus de temps cela aurait permis d'améliorer légèrement les performance de notre programme en rectifiant ces erreurs ou en ajoutant des fonctionnalité à notre implémentation.
Une des principales erreurs est la création de variable qui après réflexion n'était pas forcément très utile. Un exemple concrets de cette erreur est pour la structure de notre queue qui pointe vers une autre structure elem nous aurions pu nous contenter de la structure elem pas besoin donc de cette structure queue mais par manque de temps nous n'avons pas pu modifier cette structure.
Une autre piste d'amélioration qui nous est venu en tête lors de se projet et que nous n'avons pas eu le temps de mettre en place. C'est la possibilité pour notre receiver de se connecter avec plusieurs sender et de mettre tous les fichiers qu'ils envoient dans des buffers différents afin de les imprimer séparément une fois que le programme se termine.
Une autre piste d'amélioration est d'utiliser le timestamp pour limiter le renvoie de paquet. En donnant comme valeur au timestamp le numéro de séquence ainsi quand le receiver envoie le ack pour un paquet qu'il a mis dans sa queue il peut renvoyer le numéro de séquence de se dernier et ainsi éviter le renvoie de se dernier ce qui permet d'éviter d'encombrer le réseaux avec des paquets qui ont déjà été reçu. Une autre solution aurait été de mettre un grand String pour augmenter l'efficacité du CRC1.
Si le programme crash pour une raison inconnu nous nous retrouverions avec beaucoup de fuite de mémoire que nous n'avons pas forcément gérer. C'est pourquoi une des amélioration possible à notre projet serait de gérer les fuites de mémoire en cas de crash du programme.

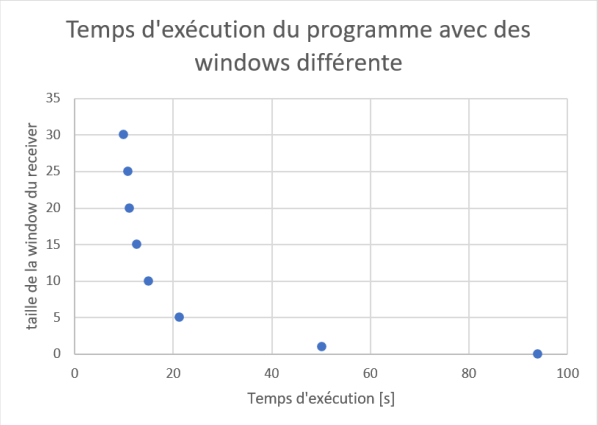
# 10 Performance de notre implémentation

Pour évaluer les performance de notre programme nous avons effectué divers tests. Un des points que nous trouvions intéressant d'évaluer est le temps nécessaire pour envoyer des fichiers textes de taille différentes. Nous avons effectué 2 tests de ce type : un avec un link_sim qui ne contient que du délai et un jitter (graphe de gauche) et un autre avec toutes les erreurs, perte, cut, délai et jitter (graphe de droite) le link_sim pouvait effectué. (./link_sim -p 64342 -P 64341 -R -l 20 -c 20 -e 20 -d 300 -j 50)




Un autre test intéressant est de voir l'évolution du temps si la window diminue c'est ce que montre le schéma suivant pour un paquet de 75000 caractères et un link_sim ne contenant que du delay et un jitter.




# 11 Stratégie de test et interopérabilité

Nous avons effectué plusieurs stratégie de tests pour nos programmes. D'abord, nous avions fait des tests séparés pour tester notre queue.c et notre packet_implemn.c. Ces tests sont disponibles dans le Makefile grâce à make suitetests.
Ensuite lorsque nous écrivions nos programmes receiver et sender, nous testions ceux-ci grâce à trois terminal et le link_sim. Nous avons fait de cette manière par facilité et parce que nous étions déjà habitué à cette manière de tester. Enfin, nous avons testé notre programme avec le fichier test.sh donné en template. Nous avons réalisé plusieurs test intéressant qui sont disponible grâce à make tests. Les erreurs et/ou informations de l'exécution du programme se trouve dans un dossier créer 'logFile' qui se supprime avec make clean.

Pour les séances d'interopérabilité :
**Group 101 :** Lors du test avec ce groupe, nous avons eu du mal à comprendre au début pourquoi lorsque nous lancions nos tests cela fonctionnait tout le temps alors que lorsque l'autre groupe lançait leur test cela ne fonctionnait pas dans tous les cas. C'était dù au fait que nos tests avec link_sim ne contenait pas l'option '-R' qui est pourtant importante. Nous avons donc améliorer notre projet grâce à cette séance.
Pour vérifier si tout se passait encore bien nous avons lancé nos tests, revérifié avec ce premier groupe.