Truncated Reliable Transport Protocol

1 Introduction

L'entreprise Intelligent Data for the Internet Of Things a pris contact avec les étudiants de l'EPL pour implémenter un protocole d'envoi de fichiers. La moitié des étudiants a reçu pour tâche d'implémenter un envoyeur de fichiers fiable et l'autre moitié un receveur fiable, compatible avec l'envoyeur. Notre groupe a implémenté un receveur.

2 Implémentation

2.1 Options supplémentaires de l'exécutable

Notre executable prend en option les champs suivants:

- ***-w*** : Cette option permet l'utilisation d'une taille de fenêtre variable. Cette fenêtre s'adapte au nombre d'erreurs de transmission. Plus les erreurs seront fréquentes, plus la taille de la fenêtre sera petite. Sans cette option, la fenêtre est par défault de taille 30.
- ***-l*** : Cette option permet d'afficher les *logs* du programme pendant l'execution.
- ***-t*** **time** : Cette option prend **time** en argument et détermine le temps maximum d'inaction ou d'attente d'un hôte avant de fermer le programme.

2.2 Selective repeat et fenêtre de connection

L'objectif de notre produit visant à envoyer des fichiers entiers, il semble indispensable d'implémenter un protocole qui assure la bonne réception de tous les paquets envoyés. Pour ce faire, nous implémentons un protocole suivant le principe de selective repeat. Lorsqu'un paquet est reçu, son destinataire renvoie un paquet d'acquittement portant le numéro de séquence du paquet suivant attendu. Si nous appliquons ce protocole paquet par paquet, nous limitons fortement la bande passante. C'est pourquoi l'envoyeur peut envoyer un nombre n de paquets à la suite dont le numéro de séquence est contenu dans la fenêtre de réception de taille souhaitée (équivalent à un intervalle d'entiers). Le récepteur acquittera uniquement les paquets dont le numéro de séquence est contenu dans la fenêtre de réception.

Pour notre récepteur, la fenêtre de réception commence par [0,1]. Seul le paquet avec le numéro de séquence 0 ou 1 sera donc accepté. Le premier paquet reçu doit selon l'énoncé contenir la taille de la fenêtre souhaitée. La fenêtre du récepteur est par conséquent adaptée en taille, et les deux bornes sont incrémentées: [1,n-1]. Un paquet avec le numéro de séquence 0 ou supérieur à n sera ignoré. Pratiquement, la fenêtre de réception est stockée sous forme de deux integer: la borne inférieure et supérieure de la fenêtre. Lorsqu'un paquet est reçu si c'est le "prochain" numéro de séquence, il sera délivré et un acquitement sera généré. S'il n'est pas le prochain, mais qu'il est contenu dans la fenêtre de réception, il est stocké dans une liste chaînée triée par numéro de séquence croissant.

La taille de la fenêtre peut être définie comme "variable" avec l'argument -w. Lorsque cette option est activée, une variable est incrémentée chaque fois qu'un paquet reçu est invalide (tronqué, corrompu ou jamais arrivé). Lorsque cette variable atteint un certain seuil, la taille de la fenêtre de réception est diminuée. L'opération inverse a lieu lors de la réception d'un paquet valide. Nos tests montrent qu'une taille de fenêtre variable ne diminue pas vraiment le temps de transfert. Nous avons donc préféré laisser la taille de fenêtre fixée au maximum.

2.3 Génération d'acquitements

L'envoyeur s'attend à recevoir un acquittement contenant le numéro de séquence du prochain paquet attendu. Lorsqu'un paquet est reçu et délivré, un paquet d'acquittement est généré. Son numéro de séquence correspont au numéro du paquet délivré +1, et son champ timestamp est la copie du dernier paquet délivré. Avant d'envoyer le paquet d'acquittement, le récepteur vérifie son buffer contenant les paquets stockés, si le paquet n+1 y est stocké, aucun acquittement n'est envoyé et le paquet n+1 est traîté et délivré. Cet appel est récursif. Lorsque le buffer a été vidé de tous les paquets en séquence, l'acquittement n+1 peut être envoyé.

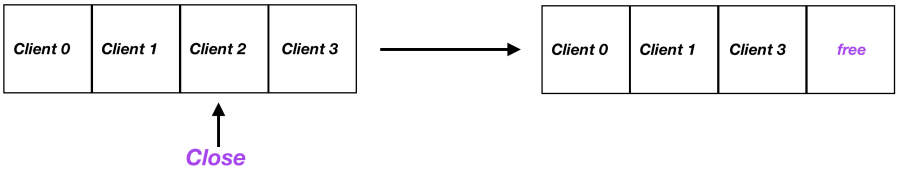

Figure 1: Fonction qui réarange les tableaux

2.4 Connections concurrentes

Nous définissons une connection concurrente comme l'envoi d'un paquet provenant d'une adresse IP différente. Les adresses IP des envoyeurs sont stockées dans une liste. Chaque adresse est donc associée à un indice. Toutes les variables (windowsize, lastack, la liste chaînée) de la structure implémentant le selective repeat sont stockées dans une liste. Ainsi, chaque envoyeur est associé au i ème élément dans chaque liste de variables. Lorsqu'un client a fini d'envoyer ses paquets et se voit fermer sa connexion, toutes les listes sont réduites au nombre de clients encore présents et une nouvelle place se libère à la fin de chaque tableau comme indiqué sur l'image ci dessous.

3 Evalutation

3.1 Section critique (lente) du code

Les parties lentes du code sont liées à l'appel bloquant select (recvfrom n'est pas bloquant dans ce cas car le select laissera passer une connection qui aura envoyé quelque chose). Le réarrangement des tableaux a une complexité en $O(n)$, avec un grand nombre de tableau, cette appel peut prendre un certains temps. L'ajout dans le buffer quant à lui a une complexité en $O(1)$ étant donné que sa taille est bornée à 32.

3.2 Fermeture des connections

Lorsqu'un client se déconnecte, l'entièreté des tableaux contenant les informations de ce client sont raccourcis comme indiqué sur le chemin ci-dessus [Figure 1]. Ainsi une place se libère pour un potentiel nouveau client (celui qui vient de se déconnecter également). Lorsqu'il n'y a plus de client, un 'Timeout' a été initialisé pour laisser des potentiels nouveaux clients venir mais pour fermer le programme après un certain temps d'inaction.

3.3 Phase de tests

La commande "make tests" exécute des tests sur l'encodage et le décodage des packets. Il teste aussi l'ajout de paquets dans le buffer d'une connections.

Tout au long de l'implémentation du programme, nous avons testé les fonctions une par une. D'abord, un des membre du groupe implémenté le selective repeat avec la structure de paquets pendant que l'autre s'occupait d'encoder et décoder des paquets. Toutes les fonctions ont été testées intensivement avant de tout mettre ensemble. La mise en commun des sous-parties du programme implique des malloc et des ouvertures de file descriptors. Il est donc impossible de tester à nouveau toutes les sous fonctions sans avoir à modifier le code explicitement pour les tests. Nous avons implémenté des tests sur l'encodage seul et l'ajout de paquets dans le buffer avec "make tests".

3.4 Evaluation des performances

Une performance intéressante à évaluer est la variation du temps d'envoi d'un paquet en fonction de la qualité de la connection. Le dispositif suivant a été mis en place:

Notre receiver recoit les paquets sur toutes les connections (./receiver :: port1), le sender de référence téléchargeable sur Moodle envoie un fichier sur l'adresse loopback (./sender ::1 port2), et le link-simulator simule des erreurs de connections. La démarche a été répétée avec différentes valeurs pour le cutoff rate, le loss rate et le error rate. Les résultats sont présentés en Figure 2.

4 Tests d'interopérabilité

Les test d'interoperabilité ont été effectués avec trois groupes: Gauthier de Moffrats, Nicolas du Roy, Antoine Caytan (et leurs binomes respectifs)

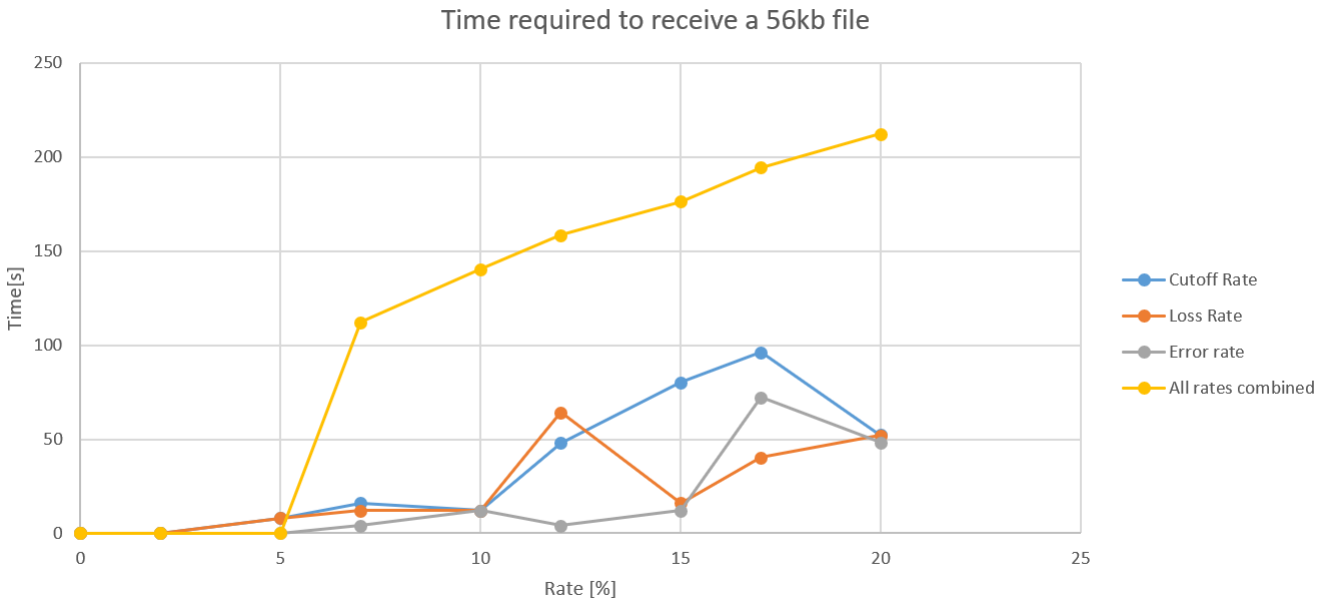

Figure 2: Temps d'envoi en fonction des erreurs réseau.

Les changements suivants ont été effectués au code:

- La fenêtre de connection n'était pas incrémentée lorsqu'un paquet était retiré ajouté du buffer. La fenêtre était incrémentée lors de l'envoi d'un ack, toutefois, en implémentant des ack cumulatifs avec le buffer, la fenêtre n'était incrémentée que de 1 lorsque le buffer de réception était vidé intégralement. Cela avait pour résultat que le sender et receiver n'avaient pas la même fenêtre d'envoi.
- La condition de fin de programme a été modifiée: un simple timeout et aucune condition sur le nombre de clients. La raison de ce changement est que parfois, un client envoyait ses n derniers paquets alors qu'ils sont déjà dans le buffer. Le receiver terminait la connection via le dernier paquet du buffer, puis recevait les paquets du sender, lançant une nouvelle connection avec les derniers paquets. Le timeout était de 30 secondes, mais le groupe de Gauthier nous a fait remarquer qu'avec un haut taux de packet loss, ce timeout n'était pas suffisant.
- Une optimisation a été faite en permettant a la taille de la fenêtre d'être variable. Cette option peut être activée avec l'argument -w