GEISSERT Cédric

VILLARD Arthur

Novembre 2021

PROJET RESEAU

Index

1. Implémentation

Le champ ID Flux identifie le flux du paquet (la source pouvant envoyer plusieurs flux à la fois). Le champ type, sur un octet, prend les valeurs 16 (ACK), 4 (RST), 2 (FIN), 1 (SYN). Un paquet peut posséder plusieurs types à la fois. Par exemple, un paquet acquittant des données peut aussi signaler l'ouverture de la communication avec le drapeau SYN. Il suffit de tester la valeur des bits correspondants aux drapeaux pour déterminer les différentes données pertinentes dans un message. Le champ Fen. Emission indique, sur un octet, la taille courante de la fenêtre d'émission. La version du médium utilisé est la dernière.

1.1 Source

1.1.1 Connexion

L'établissement de la connexion entre les deux hôtes source et destination s'effectue avec le *three-way handshake* qui s'assure que les données envoyées soient bien reçues par le destinataire en négociant l'ouverture de la communication.

Comme son nom l'indique, il se déroule en trois étapes :

1) Premièrement envoyer un paquet *SYN* (Synchronized) au serveur. Ce paquet sert a initialiser la connexion. Dans ce paquet, on va indiquer un nombre aléatoire que l'on va nommer A.

2) Le serveur, après avoir reçu le paquet de l'ordinateur, va lui renvoyer un paquet *SYN-ACK* (Synchronize, Acknowledge). Le paquet *ACK* signifie qu'il a bien reçu le paquet que l'ordinateur lui a envoyé. Dans le paquet qu'il va envoyer, il va indiquer le numéro du paquet *SYN* précédent qu'il va incrémenté de 1 (soit A+1). Il va aussi indiquer un nombre aléatoire B dans le paquet *SYN-ACK*.

3) L'ordinateur, une fois qu'il aura reçu le paquet du serveur, va envoyer un paquet *ACK* au serveur. Cela signifie qu'il a bien reçu le paquet. Encore une fois, dans le paquet qu'il va envoyer, il va indiquer le nombre reçu précédemment et l'incrémenter, dans notre cas B+1 pour le paquet *ACK*.

Dans notre programme *source.c*, nous envoyons un packet de type *SYN* avec un numéro de séquence généré aléatoirement avec la fonction *rand()* de la librairie *stdlib.h*. On retourne ce numéro afin de le réutiliser pour la fermeture de connexion.

Dans le programme *destination.c* le switch gérant le type des messages reçus contient un cas SYN et un cas ACK chacun gérant les deux types de messages que la fonction *3wayhandshake* de *source.c* peut envoyer.

1.1.2 Envoi de messages

1.1.2.1 Stop and wait

Le protocole **Stop and wait** dans *source* doit recevoir un acquittement du message de *destination* pour pouvoir envoyer un message. Plus précisément, l'émetteur (*source)* envoie une seule trame de données à la fois. Après avoir émis une trame, l'émetteur (*source)* n'envoie pas de données supplémentaires tant qu'il n'a pas reçu d'acquittement (*ACK*) de la part du destinataire (*destination).* Ce dernier n'envoie un *ACK* qu'après avoir reçu une trame correcte. Si l'émetteur ne reçoit pas d'*ACK* avant l'expiration d'un délai prédéfini (notre fonction *timeout*), il réémet la trame précédemment envoyée.

De plus, le protocole du **bit alterné** y est également implémenté, le principe est le suivant : la première trame envoyée par A aura pour drapeau 0, dès cette trame reçue par B, ce dernier va envoyer un accusé de réception avec le drapeau 1 (ce 1 signifie "la prochaine trame que A va m'envoyer devra avoir son drapeau à 1"). Dès que A reçoit l'accusé de réception avec le drapeau à 1, il envoie la 2e trame avec un drapeau à 1 et attend un accusé de réception avec le drapeau à 2. On recommence ce processus jusqu'à ce que le nombre *nb_msg* de message soient envoyés. Schéma en annexe.

1.1.2.2 Go-back-n

Le protocole Go-back-n permet dans *source* permet d'envoyer un nombre de messages équivalent à la taille de la fenêtre de congestion avant de recevoir un acquittement. Chaque acquittement libère de la place dans la fenêtre d'émission et permet l'envoi d'un nouveau message. Plus exactement à chaque trame reçue, le destinataire (*destination)* garde en mémoire le numéro de la prochaine trame qu'il s'attend à recevoir. En effet, il envoie ce numéro avec l'*ACK* qu'il émet. De plus, il éliminera toute trame reçue qui ne possède pas le numéro attendu, soit parce qu'il s'agit d'une copie de la trame précédente ou parce que la trame attendue s'est perdue. Une fois que l'émetteur (*source)* a envoyé toutes les trames de sa fenêtre, il va s'intéresser aux *ACK* reçus et, éventuellement, remarquer que tous les paquets émis depuis la première perte sont toujours attendus par le destinataire (*destination).* Il reviendra alors en arrière et émettra une nouvelle fenêtre de trames à partir de la première perte. La taille de la fenêtre d'émission ne doit pas être supérieure au numéro de trame maximum possible pour que le mécanisme de retransmission fonctionne dans tous les cas. Pour respecter cela, l'initialisation de la taille de la fenêtre de congestion est effectuée à 52 octets pour 1 message, ainsi qu'un accroissement de la taille de la fenêtre de congestion de 52 octets à chaque réception d'un acquittement en séquence ainsi que d'une division de la fenêtre de congestion par deux lorsqu'une perte est détectée par l'expiration du timer (notre fonction *timeout*). Finalement, s'il y a détection d'une perte par un retour de la fenêtre de congestion à 52 octets lors de la réception de 3 acquis dupliqués alors une réduction de la fenêtre de congestion sera effectuée, elle sera de 10% lors de la réception d'un message avec le champ ECN supérieur à zéro.

1.2 Destination

Le destinataire doit recalculer la somme de contrôle de chaque segment reçu, et si elle correspond à la somme de contrôle reçue, on considère que le segment a été reçu intact et sans erreur. Ainsi, le numéro d'acquittement représente le numéro de séquence du destinataire. Ainsi, pour s'assurer de la fiabilité, le destinataire (*destination)* doit acquitter les segments reçus en indiquant qu'il a reçu toutes les données jusqu'à un certain numéro de séquence.

1.3 Déconnexion

La phase de déconnexion utilise une sorte de *handshaking* en quatre étapes. C'est à dire, chaque extrémité de la connexion effectue sa terminaison de manière indépendante. Ainsi, la fermeture de connexion bidirectionnelle a une paire de segments FIN et ACK pour chaque extrémité.

Dans le fichier *destination.c*, le switch de traitement de type contient un cas FIN qui appelle, après avoir testé le contenu du packet, la fonction *disconnection* qui gère la fermeture de connexion propre pour le programme.

La source de son côté contient également une fonction *disconnection* qui gère les erreurs de packet afin de permettre une fermeture de connexion propre pour le programme.

1.4 Flux

Par manque de temps, nous n'avons pu implémenter cette fonctionnalité. Mais nous vous faisons part de la procédure à suivre.

La source doit pouvoir envoyer plusieurs flux en simultané à la destination. Cela permet d'évaluer l'efficacité des différents mécanismes de manière à se rapprocher de la vie réelle. Ce qui veut dire que le modèle des applications transmet des flux de données sur la connexion réseau. Ainsi, il faut découper le flux d'octets en segments dont la taille dépend de la MTU du réseau sous-jacent (maximum transmission unit).

Chaque partenaire dans une connexion devrait disposer d'un tampon de réception dont la taille n'est pas illimitée. Afin d'éviter qu'un hôte ne surcharge l'autre, il aurait fallu prévoir plusieurs mécanismes de contrôle de flux. Ainsi, chaque segment pour contenir la taille disponible dans le tampon de réception de l'hôte qui l'a envoyé. En réponse, l'hôte distant pourrait limiter la taille de la fenêtre d'envoi afin de ne pas le surcharger.

Ainsi, une étude du partage de la bande passante entre les différents flux aurait été envisageable.

2. Impact du mécanisme ECN

La notification explicite de congestion ECN est une extension qui permet de signaler la congestion du réseau avant que la perte de paquets ne se produise. ECN est prise en charge à la fois par les couches réseau et transport. Ainsi, les protocoles implémentés ci-dessus augmentent le débit tant qu'aucun paquet n'est perdu et ce ce n'est que lorsqu'un paquet est perdu que le débit est modéré. Ainsi, avec ECN, un routeur pourrait explicitement signaler un début de congestion avant de commencer à perdre des paquets. ECN sera négociée pour chaque connexion. Ainsi, lorsque la notification explicite de congestion est mise en place, le routeur marque les paquets qui le traversent lorsqu'il détecte une congestion. Mais, ces paquets marqués sont reçus par le destinataire (*destination)*, alors que c'est l'émetteur (*source)* qui devrait être averti de la congestion. Pour cette raison, le récepteur (*destination)* doit transmettre cette information en retour à l'émetteur (*source)* des paquets, et un aller-retour complet est nécessaire avant que l'émetteur ne soit averti du problème de congestion.

Pour effectuer ce mécanisme rigoureusement, ECN doit utiliser deux bits (les moins significatifs de l'en-tête IPv4) :

1) 00 : transport incapable de gérer l'ECN — Non-ECT
2) 10 : transport capable de gérer l'ECN — ECT (0)
3) 01 : transport capable de gérer l'ECN — ECT (1)
4) 11 : congestion subie — CE

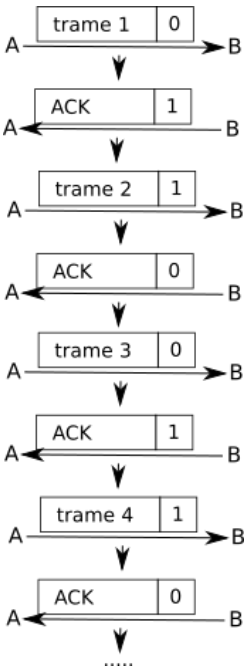
Quand les deux extrémités de la transmission prennent en charge ECN, ils marquent leurs paquets avec ECT (0) / (1). Si le paquet traverse le routeur qui la prend en charge et qui est congestionné, il peut changer cette valeur en Congestion Experienced (CE).

3. Conclusion

L'implémentation était complexe, nous avons produit une application fonctionnelle, ce qui nous a permit d'apprendre d'avantage dans ce domaine.

ANNEXE

Schéma du protocole du bit alterné :



Mécanisme ECN :