INGI1341 - Implémentation d'un protocole de transfert fiable

Ortegat Pierre (INSERT-NOMA-HERE) &
Dubray Alexandre (1178-14-00)

25 octobre 2016

I Le *sender*

I.1 L'architecture générale

Nous passerons les détails de l'établissement de la connexion pour nous concentrer sur l'essentiel du protocole et des choix d'implémentations du côté du *sender*. Supposons donc une connexion établie à un hôte distant. Faisons abstraction des contraintes (c.f code source) et supposons l'envoie d'une suite $S_1, S_2, \dots S_N$ de segment de données. Chaque segment envoyé sera stocké dans une queue [1].

Un des avantages d'une queue est que, une allocation dynamique de la mémoire permet une utilisation minimale de celle-ci lorsque le réseau est peu actif. En outre, nous gardons facilement cette queue ordonnée par ordre de *retransimission timer* [2]. Ainsi, lorsque nous parcourons notre queue pour retransmettre les paquets, nous savons que lorsque nous en rencontrons un qui n'a pas encore *time out*, alors ceux qui suivent non plus.

Lorsque nous recevons un acquittement, la première chose à faire est, si possible, retirer des paquets de la queue. Ensuite, il y a plusieurs possibilités, en fonction du champ *window* du segment. Soit k la valeur de ce champ.

— Si $k > 0$, alors on peut envoyé jusqu'à k (k compris) nouveau paquets.
— Si $k = 0$, alors le *receiver* n'a plus de place libre dans son buffer de réception et aucune nouvelles données ne peut être envoyée.. Entre alors en scène un système de temporisateur grâce à l'appel système alarm(). Le temps de ce temporisateur est fixé à quatre secondes (deux étant le RTT maximum). À chaque

1. Nous avons utilisé l'implémentation d'une *"tail queue"* définie dans la librairie *BSD libc* (TAILQ)

2. Remarquons que nous pourrions faire le raisonnement qui suit avec les numéros de séquences. L'opération optimisée ne serait donc plus la retransmission des paquets mais les acquittements

fois qu'un signal SIGALRM est intercepté, le dernier paquet envoyé et renvoyé. Ainsi, si les acquittements du *receiver* se perdent, on peut le "réveiller".

Lorsqu'il n'y a plus de données à lire sur l'entrée standard, le segment de fin de connexion est envoyé si et seulement si toutes les données ont été envoyées ET acquittée. Ainsi, même si il y a des problèmes pour fermer la connexion, toutes les données auront été envoyée.

I.2 Le *retransmission timeout*

Remarquons dans un premier temps, que le champ *timestamp* des segments envoyés sont tous nuls. En effet, les élément de la queue dans laquelle nous stockons les segments sont en fait une structure qui contient le segment et le temps (via une struct timeval) auquel le segment devra être retransmit.

Nous avons décider d'implémenter un *retransmission timer* (RT0) dynamique. L'avantage par rapport à un RTO fixe est que l'hôte réagit mieux aux variations du réseaux. Lorsqu'un acquittement est reçu pour un paquet, le (RT0) est diminuer.

Lorsqu'un segment doit être renvoyé, le RTO est augmenté. Une retransmission peut être due à deux facteurs

- Une congestion du réseau. Alors, il est important de d'augmenter le RTO pour éviter d'envoyer trop de segment sur le réseau et, ce faisant, le saturer encore plus.
- Une perte du segment. Celle-ci ne peut être devinée lorsque l'on augmente le RTO. Le fait d'augmenter le RTO alors que le réseau n'est pas congestionné n'est pas un problème vu que, lorsque le segment sera acquitté, le RTO sera diminué.

II Le *receiver*

III Les tests