**BOUTCHICHI Ilias (76931200)** **27/10/2015**
**COLLIN Anne-Sophie (36151200)**

**LINGI1341**

**Implémentation d'un protocole de transport sans pertes**

Lors de son exécution, le sender ainsi que le receiver rentrent tous les deux dans une boucle infinie qui ne se terminera que lorsque le sender aura envoyé tout son fichier (ou envoyé un caractère EOF s'il n'y a pas de fichier dans l'appel à sender) et que le receiver aura tout écrit dans un fichier de sortie (ou sur la sortie standard si aucun fichier n'est spécifié).

# 1 Architecture du sender

Une fois que le sender entre dans sa boucle infinie, il va effectuer les tâches suivantes selon l'ordre de priorité qu'elles représentent :

- Lire ce qui se trouve sur la socket et stoker le dernier acquis reçu (si celui-ci est valide)
- Envoyer les paquets qui ont engendré un NACK
- Retransmettre les paquets qui n'ont pas été reçus correctement
- Avancer dans la lecture du fichier et envoyer de nouveaux paquets tant que le receiver peut les sotcker

## 1.1 Traitement de la congestion

Nous avons considéré que de la congestion apparaissait sur le réseau à partir du moment où le sender recevait deux paquets de type NACK consécutifs. Lorsque cela survient, le nombre de paquets que le sender peut envoyer est diviser par deux.
Lors que la congestion se dissipe, le nombre de paquets que le sender est susceptible est incrémenté d'une unité jusqu'à ce que cette valeur atteigne la taille de fenêtre transmise par le receiver.
Il s'agit donc d'un contrôle de congestion de type " Additive increase / multiplicative decrease".

## 1.2 Choix de la valeur de retransmission du timeout

Afin d'optimiser le temps à partir duquel un paquet doit être renvoyé sur réseau si son acquis n'est pas encore arrivé, nous avons décidé

de changer la valeur du rtt en fonction du temps réel qu'il a fallu entre l'envoi d'un paquet et la réception de son acquis.
La valeur initiale du rtt vaut le double de la latence maximale du réseau qui est de 2 secondes. Au fur et à mesure des exécutions, on change la valeur du rtt de sorte à ce que cette nouvelle valeur soit égale à 60% de la valeur initiale + 40% de la dernière valeur mesurée. Nous tendons ainsi vers la valeur moyenne du temps entre l'envoi d'un paquet et la réception de son acquis.
Toutefois, si nous gardions cette valeur, nous serions susceptibles de renvoyer la moitié des paquets. Pour que ce ne soit pas le cas, nous multiplions la valeur précédemment obtenue par 1.5 afin de garantir que le nombre de paquets renvoyés inutilement soit minimal.

## 2 Architecture du receiver

L'architecture de notre programme receiver correspond à ce qui était attendu. Ce programme peut prendre un nom de fichier en paramètre, auquel cas il enregistrera le payload des paquets recus dans le fichier spécifié, dont il prendra soin de créer s'il est inexistant, sinon le remplacera. Le programme est ensuite subdivisé en trois parties.
Deux d'entre elles sont notamment gérer grâce à select(). La première partie consiste à lire la socket pour récupérer un paquet, le décode, et l'écrit dans la destination spécifié au préalable (fichier/stdin) si le checksum est correct.
La seconde vide le buffer si un paquet reçu permet de libérer ceux éventuellement stockés.
Enfin la troisième permet de renvoyer des acquis ou des non-acquis (en cas de congestion).

## 3 Partie critique de l'implémentation

Le pont faible de notre implémentation est qu'on ne renvoie les paquets dont le timer a expiré qu'après la lecture de tous les acquis précédents. Cela n'interrompt donc pas l'exécution du programme en cours.
Un deuxième point à mentionner est que nous avons du ajouter un temps de repos (sleep) entre la lecture de deux acquis présents sur la socket. Si nous ne le faisions pas, le sender lisait plus vite que ce les paquets n'arrivait et renvoyaient des paquets dont l'acquis était en cours d'envoi. Cela ralentit donc l'exécution du sender.

## 4 Tests

Pour la partie tests, nous n'avons eu le temps que de créer un code bash qui permet de lancer l'exécution simultanée d'un sender et d'un receiver. Nous aurions voulu avoir le temps de tester les différentes valeurs de retour des différentes fonctions implémentées mais ce n'est malheureusement pas le cas.

## 5 Résultats des tests d'interopérabilité