Université de Mons
Faculté des sciences
Département d'Informatique

Projet Reseau 1

Selective Repeat et Congestion

Professeurs :
Bruno Quoitin
Jérémy Dubrulle

Auteurs :
Laurent Bossart
Guillaume Proot

Année académique 2018-2019

Table des matières

1 Construction et exécution

Pour compiler notre programme, il faut se placer dans le dossier contenant le package reso et entrer la commande suivante : *'javac reso/examples/selectiverepeat/*.java -Xlint'*. Pour exécuter notre programme, il suffit d'entrer la commande suivante : *'java reso.examples.selectiverepeat.Demo'* en passant en paramètres : un entier qui est le nombre de paquets que l'on veut envoyer, deux floats qui sont respectivement le taux de perte de paquets et de ACKs (flotants compris entre 0 et 1. Où 1 représente une perte de 100 %). Si un ou plusieurs paramètres sont manquants ou incorrectes, les paramètres par défault seront appliqués, ils sont respectivement à 10, 0.2, 0.0.

2 Approche utilisée pour l'implémentation

2.1 Selective Repeat

- Les paquets à envoyer sont placés dans une ArrayList. La fenêtre se trouve entre deux variables (*sendBase* et N) qui représentent le début et la fin de la fenêtre.
- Les paquets envoyés par l'applications sont ceux se trouvants dans la fenêtre.
- Quand un paquet est reçu par le receveur, il est d'abord placé en cache. Ensuite, s'il s'agit du premier élément de la fenêtre, la fenêtre avance de 1, sinon on attend que le premier élément soit reçu.
- Quand un ACK est reçu par l'envoyeur, il est lui aussi placé en cache. Dans le même principe que pour le receveur, s'il s'agit du premier élément de la fenêtre, la fenêtre avance de une position.
- Lorsque l'envoyeur envoie un paquet, une place se fait dans la fenêtre. La méthode *sendNext* est alors appelée jusqu'à ce que la fenêtre soit remplie.
- Le *rto* est calculé à chaque réception d'ACK et à chaque fois qu'un timeur arrive à expiration. À chaque envoi de paquet, le moment auquel est envoyé ce paquet est gardé en mémoire. Lors de la réception de l'ACK correspondant, le rtt vaut la différence entre le moment où l'ACK est reçu et le moment auquel a été envoyé le paquet. Le calcul du rto se fait par les formules vues en cours.

2.2 Congestion Control

- TODO

3 Les difficultés rencontrées

3.1 Difficultés liées à l'utilisation du simulateur

- La prise en main d'une application développée par une tierce personne est, de façon surprenant, très chronophage. Nous avons passer un bon moment à relire le code du simulateur pour en comprendre du mieux possible le fonctionnement.
- TODO Je ne sais pas quoi mettre ensuite.

3.2 Difficultés liées à l'implémentation de Selective Repeat et de Congestion Control

- La plus grosse difficultée rencontrée est la gestion des *Timers*. Nous n'avons pas réussis à complètement implémenter ceux-ci. Nous n'avons pas trouver de solution pour stoper le bon Timer au bon moment. Nous avons essayer d'utiliser une ArrayList de Timers pour, lorsqu'un ACK(x) est reçu, arrêter le Timer(x) mais celà ne nous a pas donné de résultats probrants.

— TODO Compléter aussi.

4 État de l'implémentation finale

Le protocol fonctionne correctement si nous n'activons pas la perte de paquets et d'ACKs. Malheureusement notre mauvaise gestion des timers ne nous permets pas de gérer de manière totalement fonctionnelle le renvoi de paquets. Le protocol est améliorable en trouvant une meilleure solution pour la gestion des renvois de paquets.

5 Note d'utilisation

— Étant donné que pour chaque utilisation la taille de la fenêtre ne sera pas constante, lors de l'exécution de notre programme nous stockons dans un fichier *data.txt* les données de la taille de la fenetre de la dernière exécution. La colonne de gauche représente le temps du *scheduler* et la colonne de droite la taille de la fenêtre à ce moment là. Pour créer le plot de la dernière exécution, il faut enter la commande suivante via gnuplot :

'plot "data.txt" with linespoints'

— Nous avons choisi d'afficher les logs du déroulement de paquets directement dans la console.