Université de Mons
Faculté des sciences
Département d'Informatique

Projet Reseau 1

Selective Repeat et Congestion

Professeur :
Bruno Quoitin
Jérémy Dubrulle

Auteur :
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 maquants ou incorrectes, les paramètres par défault seront appliqués, càd respectivement 10, 0.2, 0.0.

2 Approche utilisée pour l'implémentation

2.1 go-back-n

- Les paquets à envoyer sont stockés dans une *ArrayDeque*. La fenêtre est délimitée par des variables *SendBase* et *N* représentant respectivement le début et la taille de la fenêtre.
- Lorsque la couche application envoit les paquets vers go-back-n, le protocole place tous les paquets du Sender dans l'*ArrayDeque* et envoit uniquement les paquets compris entre *SendBase* et *SendBase+N*.
- *exp_seq_num* est initialisé à -1. En effet, lorsque le premier paquet de numero de séquence 0 est envoyé, s'il est perdu, le ACK envoyé doit avoir le numéro de séquence du paquet reçu précédemment.
- Quand un ACK est reçu, si c'est le ACK d'un paquet déja envoyé et pour lequel on a pas encore reçu de ACK, la fenêtre avance jusqu'au paquet suivant. Sinon, le ACKs est ignoré.
- Comme de la place s'est libérée dans la fenêtre, la méthode *sendNext* est appelée. Cette méthode envoie des paquets jusqu'au moment où la fenêtre est pleine.
- Quand un timeout à lieu, *nextSeqNum* est réinitialisé à la valeur de *sendBase* et la méthode *sendNext* est appelée pour renvoyer tous les paquets de la fenêtre.
- Le rto est recalculé à chaque réception d'un ACK. Pour ce faire, à chaque envoi de paquet, le temps auquel est envoyé celui-ci est maintenu en mémoire. Lors de la réception du ACK correspondant, le rtt vaut la différence entre le temps où le ACK est reçu et le temps auquel a été envoyé le paquet. Le calcul du rto se fait par les formules vues en cours.
- A chaque nouveau rto, le timer en cours est arrêté et un nouveau timer est instancié et démarré avec le nouveau rto.

2.2 Congestion Control

- A chaque évènement, go-back-n appelle la méthode de la classe *CongestionControl* correspondante (*isALoss* pour un timeout et *receiveAck* pour un ACK reçu). Ces méthodes recalculent la nouvelle taille de fenêtre qu'elles retournent à go-back-n qui modifie la taille de sa fenêtre.
- Dans *CongestionControl*, la taille de la fenêtre (représentée par la variable *size*) est un double car les formules de calcul de taille de fenêtre peuvent retourner des nombres flotants et le fait d'utiliser un type double permet d'accumuler les parties décimales qui ne seraient pas prises en compte si nous avions utilisé un entier.
- Quand un ACK est reçu, si le numéro de séquence (appelons le rNew) est plus grand que le numéro de séquence (appelons le rOld) du ACK précédent, la taille de la fenêtre est incrémentée de rNew-rOld. Nous sommes alors en slow start si la taille de la fenêtre ne dépasse pas le seuil, *ssthresh*. Si la taille de la fenêtre dépasse ce seuil, nous sommes en congestion avoidance et la taille de la fenêtre vaut size=size+(1*(1/size)).

— Lorsque 3 ACKs avec le même numéro de séquence ont été détéctés, le seuil et la taille de la fenêtre vallent size/2.
— Lors d'un *timeout*, le seuil vaut la moitié de la taille de la fenêtre et la taille de la fenêtre vaut 1.

3 Les difficultés rencontrées

3.1 difficultés liées à l'utilisation du simulateur

— Scheduler : Dans l'*applicationSender*, on envoyait les paquets au protocole Go-Back-N à travers une boucle *while(true)*. Cependant, lors de la simulation, seul le dernier paquet envoyé à Go-Back-N était envoyé à l'*applicationReceiver*.
— Initialisation de la connexion : Nous avons rajouté un premier message d'initialisation à go-back-n : un premier message spécial est envoyé vers le destinataire et tant qu'une réponse n'est pas reçue, l'envoi de paquets ne commence pas.
Ce rajout est dû à des problèmes engendrés par l'implémentation du protocol ARP dans le simulateur.

3.2 Difficultés liées à l'implémentation de go-back-n et de Congestion Control

— Timeout : Pendant la simulation, lorsque l'on reçoit trois ACK avec un même numéro de séquence, tout de suite après, un timeout se produit. Pour éviter ce problème, nous démarrons un nouveau timer.
— Par la suite, nous nous sommes rendu compte qu'il était plus efficace de renvoyer tous les paquets de la fenêtre dès la détection de trois ACKs dupliqués.

4 L'état de l'implémentation finale

Avec l'analyse des résultats, on peut remarquer que l'envoi des paquets commence bien en slow start, comme le *ssthresh* diminue fortement, une grande partie de l'envoi de paquets se passe en congestion avoidance.
Nous nous sommes rendu compte qu'il était possible d'améliorer notre protocole, notamment lorsque l'on détecte plusieurs fois 3 ACKs dupliqués et lorsque le *ssthresh* diminue trop rapidement

5 Note d'utilisation

— Etant donné que pour chaque utilisation la taille de la fenetre 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 fenètre à ce moment là. Pour créer le plot de la dernière exécution, il faut renter la commande suivante via gnuplot :

plot "data.txt" with linespoints

— Les logs du déroulement de l'envoi des paquets sont affichés dans la console.