Rapport Reseaux 1:
Projet: Protocole go-back-n avec contrôle de congestion

Libert Robin 161501
Mattens Simon 160846
BA2 Info

11 mai 2018

1. Construction et exécution

Pour lancer l'application, il faut se positionner dans le répertoire src, compiler les fichiers à l'aide de la commande "java reso/examples/gobackn/*.java", exécuter le fichier Program à l'aide de la commande "java reso/examples/gobackn/Program arg1 arg2" ou arg1 représente le nombre de message à envoyer et arg2 la probabilité de perte des messages. Une série de log apparaîtra dans la console pour expliquer ce qu'il se passe dans l'application et les plots seront dans le fichier "Plots.txt" situé dans le répertoire src.

2 Approche utilisée dans l'implémentation

Notre programme simule une application émettrice, représentée par la classe AppSender, qui envoie un certain nombre de messages à une application réceptrice, représentée par la classe AppReceiver. Les messages envoyés par AppSender seront simplement l'entier 42. Le nombre de messages à envoyer par AppSender est fixé par l'utilisateur (premier paramètre de notre programme). Une fois que AppReceiver reçoit un message, elle renvoie un acquittement à AppSender.

Un message peut prendre 2 formes. La première est un message standard avec un numéro de séquence, un entier qui est le message à envoyer et un booléen qui est mis sur faux car le message n'est pas un acquittement. La seconde forme est un acquittement avec le booléen mis sur true et avec un numéro de séquence qui a la valeur du numéro de séquence du précédent message reçu.

L'utilisateur peut entrer un deuxième paramètre, qui est un nombre entre 0 et 100. Ce nombre est la probabilité de perdre un message et/ou un acquittement.

Le protocole GoBackN a été implémenté en deux grandes parties. La première partie est la classe ProtocolSenderSide, qui gère l'envoi d'un message, la réception d'un acquittement. Cette partie gère également le contrôle de congestion, réception de 3 acquittements dupliqués, et les time out, dépassement de délai pour la réception d'un acquittement plus grand ou égale au numéro de séquence du dernier message envoyé. La seconde partie est la classe ProtocolReceiverSide, qui gère la réception d'un message et renvoie l'acquittement approprié.

Nous stockons tous nos messages dans une ArrayList. Ensuite nous envoyons un message spécifique qui ne peut pas être perdu pour lancer le protocole (Actuellement, cette étape pourrait être enlevée) et initialiser les variables et le timer nécessaire au bon déroulement de celui-ci. Lors de l'initialisation, nous chargeons dans le protocole l'ArrayList de messages et temps que tous les messages n'ont pas été envoyés, le protocole continue de travailler.

Notre sliding window ne se remplit pas à proprement parler, c'est plutôt un repère qui va délimiter des bornes dans notre ArrayList. Ainsi notre fonction send() envoie uniquement les messages qui sont entre les bornes de la sliding windows. Ainsi, la taille de la fenêtre peut être augmentée et diminuée à tout moment.

Pour le reste de l'implémentation, nous avons codé et géré chaque événement comme indiqué dans le cours. Pour plus d'informations, nous avons documenté chaque méthode de notre code.

3 Difficultés rencontrées

— Lors de l'implémentation du protocole GoBackN, nous avons eu du mal a visualiser comment faire pour le système des acquittements et pour apprendre à utiliser le simulateur.
— Pour le contrôle de congestion, il a fallu du temps avant de bien comprendre TCP Reno ainsi que les différentes techniques demandées.
— Ce qui est difficile dans ce genre de projet est de penser aux différents cas possibles. Nous avons essayé d'en gérer le maximum mais nous ne pouvons pas garantir que tout a été traité.

— Le plus gros problème que nous avons rencontré a été de savoir comment déclencher l'événement "envoyer un message" correctement.

4 État de l'implémentation finale

Le protocole goBackN a été correctement implémente ainsi que les différentes techniques de TCP Reno. Les plots sont présents dans le fichier "Plots.txt", ce fichier représente l'évolution de la taille de la fenêtre de congestion au fil du temps. Dans le fichier "Plot.txt" (SS) signifie que l'on est dans le mode slowstart et (AI) que l'on est dans le mode additive increase. Pour l'additive increase, on constate dans les plots que une taille de fenêtre revient plusieurs fois d'affilé. Ceci est normal car une taille de fenêtre doit être une valeur entière, or dans l'additive increase, une taille de fenêtre augmente par des nombres plus petits que 1. Quand la taille de fenêtre augmente d'une valeur < 1, nous plaçons cette valeur dans un accumulateur et rafraîchissons réellement la taille de la fenêtre quand l'accumulateur à été augmenté de plus de 1.

En ce qui concerne les logs, nous affichons dans l'invite de commande les messages avec leur numéro de séquence. Les acquittements avec le numéro de séquence des messages correspondant. La valeur de RTO, qui initialement vaut 0. La valeur de ssthresh qui initialement vaut un nombre très grand. Nous affichons également la taille de la fenêtre à chaque ack reçu et le mode actuel (slowstart ou additive increase). Un message apparaît et est mis en évidence lors d'un timeout ou d'une congestion.