Université de Mons
Faculté des sciences
Département d'Informatique

Selective Repeat Protocol

Rapport de projet

Professeur :
Bruno Quoitin
Assistants :
Jérémy Dubrulle

Auteurs :
Thomas Bernard
Augustin Houba

Année académique 2021-2022

Table des matières

1 Build et exécution du code

1.1 Building

Pour pouvoir build et run le projet il faut effectueur les deux commandes au sein du package suivant *bqsim.src* :

1. **Build :** javac reso/examples/selectiveRepeat/Demo.java
2. **Running :** java reso/examples/selectiveRepeat/Demo

1.2 Lancement de l'application

Pour l'éxécution de l'application et les choix des paramètres liés à la simualtion, nous avons décidé d'utiliser des entrées clavier plutôt que de demander à l'utilisateur d'ajouter un argument à la commande passée à la console pour l'exécution.

C'est pourquoi lors du lancement l'utilisateur est invité à choisir le nombre de paquets qu'il souhaite envoyer, la longueur du lien en km, le bitrate en bit/s et pour finir le taux de chance lié à la perte d'un paquet et/ou ACK.

Concernant ce taux de probabilité il varie entre 0 et 1. Au cours du testing nous avons remarqué que le format d'entrée variait en fonction du système d'exploitation. En effet, sous linux, le double doit être représenté comme suit (x.x) et sous macOS comme suit (0,0).

Une fois les paramètres entrés l'application démarre et on peut suivre l'évolution de la simulation via la console.

2 Choix d'implémentation

2.1 AppSender et Appreceiver

Ces deux classes représentent la couche application du Sender et du Receiver dans le protocole. Leur implémentation est basée sur l'implémentation du protocole d'exemple *PingPong*.

Néanmoins, dans le cas du Sender, lors de l'instanciation nous avons rajouté un paramètre **totalPacketNumber** afin de pouvoir plus tard générer la liste de paquets à envoyer via le protocole.

Nous avons également rajouté un paramètre commun aux deux applications qui est le **lossProb** et qui est la probabilité de perdre un paquet ou un ack c'est pourquoi le paramètre est présent des deux côtés.

2.2 SRPacket

La classe SRPacket incarne la gestion des messages du protocole, c'est au sein de cette classe que sont instanciés les paquets envoyés via notre protocole. Cette classe est composée de deux constructeurs au lieu d'un car en effet, nous avons jugé qu'il était plus simple de gérer les paquets si nous faisions une distinction entre les ACK et les paquets de données.

Nous avons également décidé que pour des facilités d'implémentation les données contenues dans un paquet seraient un unique entier ce qui est comme stipulé dans la consigne une donnée "bête"

Constructeur pour un paquet de données Le constructeur d'un packet de données prend deux paramètres : le numéro de séquence du paquet **seqNumber** qui permet d'identifier le paquet et un entier qui représente la donnée contenue dans le paquet **data**.

Constructeur pour un ACK Le constrcuteur d'un ACK se compose de 3 paramètres : le numéro de séquence de l'ACK, afin de pouvoir identifier le paquet pour lequel il a été envoyé. Le numéro de séquence actuellement attendu par la fenêtre de réception **recvBase**. La raison pour laquelle ce numéro de séquence se retrouve au sein du paquet sera énoncée ci-après dans les choix d'implémentation liés au SRProtocol. Et enfin, un boolean permettant de savoir s'il s'agit bien d'un ACK ou non **isAnAck**. Cet argument n'était pas nécessaire mais a été ajouté afin de faire la distinction entre les deux constructeurs.

Méthodes propres à la classe Il y a 3 méthodes qui sont propres aux paquets du protocole de Selective Repeat et qui sont les suivantes :

1. **isAnAck() :** La méthode retourne simplement un boolean qui définit s'il s'agit d'un ACK

2. **isAcknowledged() :** La méthode retourne un boolean pour savoir si un ACK a bien été reçu pour le paquet en question.
3. **setAsAcknowledged() :** La méthode permet de définir un paquet comme ayant reçu son ACK.

2.3 SRProtocol

La classe SRProtocol incarne la couche transport du Selective Repeat, il s'agit de la classe principale du projet, c'est elle qui gère l'envoi et la réception des paquets ainsi que ce qui en découle. Elle contient également une classe interne **SRTimer** qui représente une implémentation d'un timer dédié au Selective Repeat sur base de la classe **AbstractTimer**

SRTimer Dans le cadre de l'implémentation des timers nous avons décidé d'appliquer un timer à chaque paquet en lui passant le même numéro de séquence. Ainsi au sein de notre protocole nous avons un tableau de SRTimer dont chaque SRTimer à l'indice i est associé au paquet dont le numéro de séquence est également i.

Le Protocole Comme précisé ci-dessus les timers sont gérés sous forme de tableau de la longueur de la liste des paquets.

Toutefois, pour ce qui est de la gestion du *buffer* dans le cas de la réception de paquets marqués comme *out-of-order* nous avons créé un tableau de taille égale à 50 de manière arbitraire afin d'avoir la possibilité d'avoir une taille maximale de paquet out of order décente. Le buffer est donc vidé à chaque fois que les paquets ont été remis dans l'ordre. Les indices auxquels sont placés les paquets out-of-order sont calculés à l'aide du numéro de séquence du paquet initialement attendu avec l'opération suivante : *rcvdPacket.seqNumber - recvBase* en effet si nous attendons le paquet 5 et que nous recevons le 6 nous avons *recvBase = 5 et seqNumber = 6* donc le paquet sera placé à l'indice 1 du tableau. Lors de la réception du paquet 5 le rcvBase est incrémenté et le rcvdPacket.seqNumber vaut à présent 5. Ainsi, on va dans une boucle envoyer tous les paquets du buffer à l'indice *recvBase - seqNumber* tant que celui ci n'est pas *Null* en augmentant à chaque passage le recvBase.

Double constructeur Nous avons pris la décision d'utiliser deux constructeur afin d'avoir un constructeur par interface *Receiver et Sender*.

1. **Sender :** S'occupe de l'initialisation du tableau des Timers, ainsi que de la liste de paquets ou encore du tableaux pour le triple ack.
2. **Receiver :** S'occupe de l'initialisation du buffer.

2.4 Congestion Control

Pour la gestion de la fenêtre de congestion nous nous sommes simplement basé sur les formules de congestion control disponibles dans le chapitre 3 du cours de réseaux.

Pour la technique du slow start nous avons initailisé une valeur arbitraire *slowStartThresh = 20* qui définit la taille maximale de la fenêtre avant de passer à l'additive increase. Le paramètre est changé à chaque timeout comme suit :

$$\text{slowStartThresh} = \frac{\text{oldsize}}{2}$$

Où oldsize est l'ancienne taille de la fenêtre.

Une fois ce seuil atteint l'augmentation de la taille de la fenêtre est régi pas la formule :

$$\text{windowSize+} = \frac{1}{\text{windowSize}}$$

.

2.5 Plot de l'évolution de la windowSize

A chaque fois que nous modifions la taille de la window, nous ajoutons la modification dans une *String* qui contient la taille actuelle ainsi que le temps du *Scheduler* ce changement a eu lieu. A la fin de la simulation nous enregistrons les résultats obetnus dans un fichier au format .csv qui se trouve dans le package **bqsim**. Afin de porter les résulats en un graphique nous utilisons le site suivant : https ://www.csvplot.com/

3 Problèmes rencontrés au cours de l'implémentation

Le problème majeur rencontré au cours de l'implémentation est le problème responsable de la présence du numéro de séquence du paquet attendu par la fenêtre de réception dans le constructeur d'un SRPacket de type ACK.

En effet, à cause du double constructeur dans le SRProtocol nous nous retrouvons avec 2 instances possédant des attributs différents. Le recvBase n'était en effet mis à jour que du côté du *Receiver* ce qui posait problème lors de la réception des ACK car nous vérifions à chaque réception d'un ACK si le recvBase est égal au nombre de paquets total afin d'écrire le message de fin et de sauvegarder les données de la window dans un fichier. Or la réception d'ACK se fait dans l'instance du *Sender* c'est donc la raison pour laquelle lors de la création de l'ACK la valeur actuelle de recvBase est passée en paramètre afin de mettre celle du *Sender* à jour.

Un moyen de solutionner ce problème serait d'écrire directement les modifications dans le fichier au fur et à mesure qu'elles ont lieu. Cela aurait impliqué une trop grande révision du code et aurait rajouté une lenteur au cours de la simulation plutôt qu'elle ait lieu une fois la simulation terminée. C'est pourquoi nous avons décidé de laisser le code en l'état.

4 Améliorations

En l'état actuel du projet, la majorit du protocol est fonctionnelle. Bien que l'on ressente quelques fois certaines faiblesses qui sont les suivantes :

En termes d'améliorations nous pourrions faire une refonte du code afin qu'il n'y ait plus qu'un seul constructeur au sein du SRProtocol et retravailler la sauvegarde des données de la window dans un fichier pour ne plus devoir passerc le recvBase en paramètre d'un ACK.

On pourrait aussi améliorer la gestion de la taille du buffer ainsi que la valeur du slowStartThresh. Ces valeurs pourraient être basées sur le nombre de paquets total afin de ne plus être définies arbitrairement ce qui permettrait dans le cas du slowStartThresh d'observer un graphique plus fidèle à la réalité quant à l'évolution de la taille de la window.