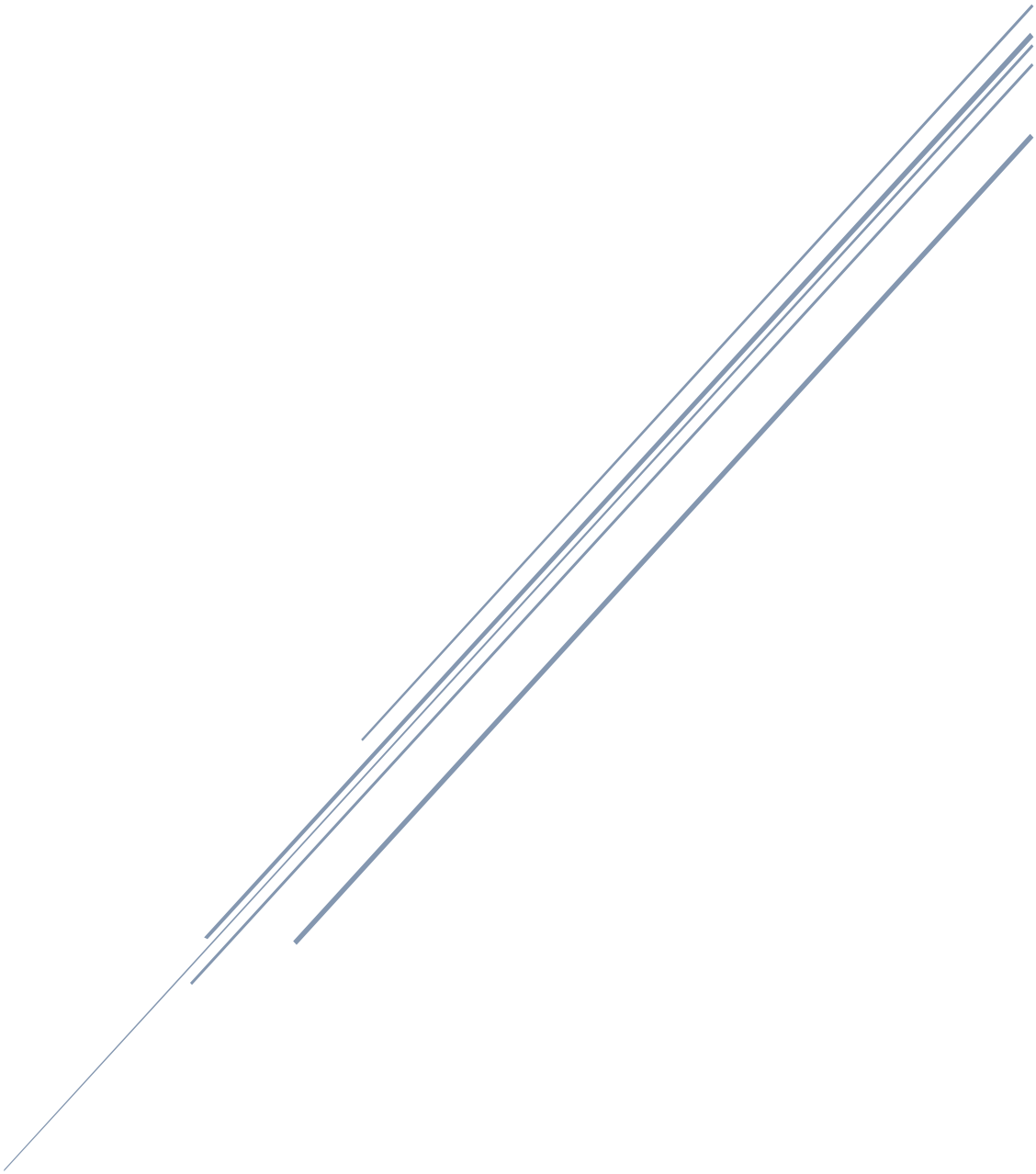
RAPPORT PROJET RESEAUX

Groupe 8



Université de Mons : 2021-2022

Hugue Likwema Nkany - Moustapha AIT SALAH

Table des matières

Introduction :

Lors de notre cours de Réseaux 1 nous avions eu l'occasion d'étudier différents protocoles de pipelining comme notamment le Selective Repeat.

Et il nous a été donc demandé dans le cadre des TP d'implémenter au sein du bq-simulator un protocole de type Selective Repeat avec un contrôle de la congestion de type Reno.

Lien github : https://github.com/Hugugus/pReseau

Construction et exécution :

Notre projet est composé de 7 classes :

- ACK.java cette classe permet la création d'un ACK contenant son numéro de séquence.

- App.java dans cette classe nous avons la méthode start qui initialise une connexion et lançant une simulation

- Demo.java dans cette class se trouve tous les liens

- SRProtocol.java dans cette classe se trouve le code du protocole

- Segment.java dans cette classe se trouve le code d'un Segment contenant le numéro de séquence et le contenu du segment.

Pour compiler notre projet il suffit de lancer Demo (le package doit bien être dans exemple)

Choix d'implémentation :

Il a fallu utiliser plusieurs plusieurs array et plusieurs contantes global :

Les arrayList permettent de stocker stocker les segments.

Les constantes permettent de gérer les tailles des fenêtres, le numéro de séquences attendu par le protocole, le RTO, …

Au début cela va découper le message (un String) à envoyer en plusieurs parties. Des segments seront créés avec un numéro de séquences défini et un timer sera lancé avec RTO comme temps lorsque le segment sera envoyé vers le récepteur. Lorsqu'un segment est reçu, s'il correspond au numéro de séquence attendu, le segment sera ajouté au buffer order sinon il sera ajouté au bufferNoOrder. Un ACK sera ensuite crée et envoyé vers l'émetteur et l'ACK sera utilisé ou non par l'émetteur en fonction d'un nombre aléatoire.

Dès qu'un timer sur un segment expire, il sera marqué comme à renvoyer avec repeat qui vaut true . Après chaque réception d'un ACK, on vérifie parmi tous les segments envoyés et tant que les segments marqués comme à renvoyer n'ont pas reçu d'ACK, on va renvoyer les segments marqués comme à renvoyer et attendre un temps de RTO.

On va avancer dans la fenêtre d'envoi à chaque fois que la taille de la fenêtre sera atteinte et cette taille de fenêtre sera modifié en fonction qu'on soit en additive increase ou slow start. Le slow start

fonctionne jusqu'à un certain seuil slowEnd puis on passera en addtive increase jusqu'à ce qu'un timer de segment expire.

Le RTO se mettra à jour après que la taille de la fenêtre sera atteinte (lorsque l'ACK du segment suivant sera reçu). Le nombre de segments sera compté pour savoir quand est-ce qu'on aura tout envoyé après quoi, on va mettre dans une ArrayList le contenu de bufferOrder et bufferNOrder puis trier cela en fonction de leur numéro de séquence. Après quoi on va afficher le message reçu.

Erreurs rencontrées :

Lorsque le 1er segment ou le 1er segment envoyé après que la taille de fenêtre ne reçoit pas d'ACK, le programme est bloqué. J'ai donc enlevé cette possibilité.

Conclusion :

Ce projet nous a été très enrichissant étant donné que nous avons pu mettre en pratique la vision théorique d'une notion vue en cours.