Projet - Réseaux 1
Implémentation d'un protocole de type Go-Back-N avec contrôle de la congestion

UMONS
Faculté des Sciences
BAC 2 Science-Informatique.

Delplanque Julien
Charlier Maximilien

Année académique
2013-2014

i Introduction

Le but de ce projet est d'implémenter une application offrant un service de pipelining de type go-back-n couplé avec un contrôle de la congestion similaire à TCP reno.

ii Fonctionnalités

- **Envoi-Réception** :
  Au lancement de l'application un timer est lancé, sont but est d'essayer d'envoyer 10 paquets tous les 100ms, afin de simuler une application qui tante d'envoyer des données.
  Ce timer va donc saturer la couche physique (une fois saturer des `BufferFullException` seront générer).
  On aura donc le Go-Back-N utilisé au maximum de ses possibilités.
  Concernant la réception, vous aurez la possibilité via la GUI de simuler des latences et des pertes de paquets en temps réel.
- **Bo-Back-N** :
  - **Réception d'un ACK** :
    On reçois un ACK possédant un numéro de séquence (qu'on va appeler y), signifiant que tout les messages ayant un numéro inférieur ou égal à y on bien était reçu.
    Si y est plus grand que le numéro de début de fenêtre d'envois ($sendBase$), on avance celle si jusqu'à y + 1 et on réinitialise le timer de **r**etransmition de **t**ime **o**ut (si l'on s'apperçois que c'était le dernier paquet à recevoir, on arrête le timer RTO).
    Maintenant si $y < sendBase$:
    Il s'agit d'un ack dupliqué. Ici on ne sinteresse qu'a seux qui valent sendBase-1 (seux inferieur à se nombre montrent qu'il y a de la congestion dans le sens receiver > sender (et on s'en moque).)
    Si l'on reçois 3 fois $y = sendBase - 1$, on détecte une perte de paquet et l'on déclenche un ACK DUP.
  - **Réaction à un ACK dupliqué** :
    On définit la valeur du seuil comme étant la taille de la fenêtre d'envois sur deux. Ensuite, on definit la taille de la fenêtre d'envois par la valeur du seuil (on passe donc en **fast recovery**).
    Afin de débloquer la situation on fait un **fast retransmit**.
  - **Réaction à un timeout** :
    On définit la valeur du seuil comme étant la taille de la fenêtre d'envois sur deux. Ensuite, on definit la taille de la fenêtre d'envois par defaut de celle-si(1) (on passe donc en **slow start**).
    Afin de débloquer la situation le premier paquet de la fenêtre est rénvoyé.
  - **Réaction à un Time Out** :
    On définit la valeur du seuil comme étant la taille de la fenêtre d'envois sur deux. Ensuite, on definit la taille de la fenêtre d'envois par defaut de celle-si(1) (on passe donc en **slow start**).
    Afin de débloquer la situation on fait un **fast retransmit**.
- **Contrôle de la congestion** :
  - **Slow-start** :
    Consiste à une incrémentation multiplicative de la taille de la fenêtre d'émission(*2 toute les 100ms). Cette agrandissement s'arrête à partir d'un sertain seuil (calculé dynami-

quement), pour ensuite passer en Fast-Recovery.
  - **Fast-Recovery** :
    Consiste à une incrémentation constante de la taille de la fenêtre d'émission (+1 toute les 100ms).
  - **Fast-Retransmit** :
    Suite à la perte d'un paquet, cette évènement et déclenché.
    Il consiste a renvoyer tout les paquet présent dans la fenêtre d'émission.
- **Interface graphique** :

  - **Graphique temps-réel** :
    Le graphique affiche en temps réel la taille de la fenêtre en fonction du temps (ligne rouge) et le "threshold" (ligne bleu). Cela permet de visualiser le comportement du protocole et de vérifier son bon fonctionnement.
  - **Affichage des données importantes** :
    Sous le graphique, des données importantes sont affichées. Elle aussi sont rafraichies en temps réel.
  - **Perte de paquet sur demande** :
    Le bouton "Lose a packet" permet d'ordonner à la classe GoBackNReceiverProtocol de "perdre" le prochain paquet qui sera reçu.
  - **Choix de la probabilité de perte de paquet** :
    TODO si assez de temps
  - **Choix du délais ajouté avant l'envoies d'ACK** :
    TODO si assez de temps

Remarque : Les timeout et les pertes de paquets n'étant pas générées par le simulateur, ceux-ci sont pris en charge par la classe GoBackNProtocolReceiver.
En effet, celle-ci simule les pertes de paquets de façon aléatoire et crée un délais de temps aléatoire avant l'envoies d'ACK.

iii Utilisation

- **$ ant build** : Construit l'application dans le répertoire bin/.
- **$ ant run** : Construit l'application dans le répertoire bin/ et lance l'application.
- **$ ant clear** : Supprime le répetoire bin/.
- **$ ant doc** : Génère la documentation au format HTML dans le répertoire doc/.
- **$ ant jar** : Génère le jar de l'application. Attention : le répertoire lib/ contenant les jar des librairies utilisées doivent impérativement se trouver dans le même répertoire que le jar. Sinon au lancement, le jar indiquera une erreur.

Remarque : Afin d'utiliser efficacement les log de l'application, il est conseillé d'utiliser la commande grep couplée à la commande de lancement du programme. Par exemple : ant run | grep "mot clé". TODO mettre a liste des mots clés utiles

iv Répartition des tâches

Maximilien Charlier s'est occupé de la partie "envoi de paquets" de l'application et Julien Delplanque s'est occupé de la partie "réception de pacquets" et de l'interface graphique.

v Remarques

Voici les outils avec lesquels le proxy a été développé :

Nom	`OS`	Version `java`	Éditeur de texte	Version `GIT`
Julien Delplanque	`Archlinux` 3.13.8	1.7.0_55	`Sublime text 2`	1.7.2.5
Maximilien Charlien	`Ubuntu` 13.4	1.7.0_55	`Sublime text 3`	1.7.2.5

vi Sources

- *JFreeChart (GNU LESSER GENERAL PUBLIC LICENSE)*