Université de Strasbourg — Algorithmes des Réseaux

Licence Math-Info – Semestre 5 — 2021/2022

El Chamaa Omar

Eizouki Issam

Compte Rendu Du Projet :

Notification de congestion

Sommaire

I. Introduction

a. Le Sujet

Pour ce Projet, Ils nous ai demandé d'implémenter les protocole de contrôle de flux et de gestion vus en cours, plus précisément, Stop N Wait et Go-back-N, en utilisant le User Datagram Protocol, ou UDP. Ce dernier permet la transmission de données très simplement entre deux entités, en effet, des procèdes comme le 3 Way handshake, acquittement, et manipulation de la fenêtre de congestion n'ont pas lieu. Ceci est problématique, car un programme qui utilise ce protocole ne prendra pas en compte les éventuels problèmes de fiabilité de réseaux et de congestion. Pour cela, l'objectif de ce projet serait d'implémenter ces procèdes en utilisant que le UDP, ou en autres mots, implémenter le TCP. Pour simuler la congestion et la perte de paquets, un programme ‘’*Perturbateur*’’ nous ai fournis.

b. Organisation du Travail

Une bonne compréhension des procédés est primordiale pour pouvoir les implémenter, nous avons alors commence par la décomposition de toute leurs

étapes. Les cours sur Moodle et td nous ont été très utiles, car on y trouvait des exemples très détaillés avec des schémas, mais aussi le pseudo-code des différents procédés qui nous a permis de comprendre leur logique.

Afin de mieux synchroniser notre avancement, nous avons créé un dépôt Git, et diviser les taches pour essayer d'être le plus efficace possible.

c. Utilisation

Pour générer les exécutables, utilisez le makefile. Make plot permet de générer une figure des résultats qui se trouvera dans le répertoire bin. Il faut exécuter serveur avant source pour le bon fonctionnement du programme, idéalement, ce bug aurait été corrigé.

II. Implémentation

a. Structure du packet

Les messages, ou packet, doivent suivre le format énoncé dans le sujet, et il est important de le respecter sinon le programme medium ne pourra pas reconnaitre les attributs du message.

```
typedef struct packet
{
    char id ;
    char type ;
    short seq  ;
    short acq  ;
    char ecn ;
    char fenetre ;
    char data[42] ;
}packet;
```

Cette implémentation respecte les contraintes de nombre de bit sur lequel chaque champ doit être encodé.

b. Etablissement de connexion

Le 3 way handshake permet au client et au serveur de synchroniser et établir leur connexion, et de s'assurer qu'aucun problème n'a eu lieu. Cette partie était assez rapide et rapide à implémenter.

c. Stop N Wait

Après avoir compris et implémenté l'établissement de connexion, ce procédé était évident à mettre en place. La logique est assez directe.

d. Go-Back-N

L'implémentation du go-back-n, quant à elle, était plus complexe.

Pour commencer cette fonctionnalité, nous nous sommes mis d'accord sur une liste doublement chainée,

```
typedef struct liste {
    struct packet* p;
    int NumSeq ;
    struct liste *suivant;
    struct liste *precedent;
}liste;
```

Cette dernière prend comme attributs sa position dans la liste/Num de séquence, un pointeur vers un packet, et deux autres pointeurs vers le suivant et le précèdent. Cette liste chainée sera donc notre fenêtre d'émission.

Sa taille étant un entier, comment faire lorsque la taille de cette fenêtre diminue de 10% par

exemple ? Cela veut dire que si on a notre taille fixée 2, soit 104 octets, il faudra dorénavant en envoyer 93. On divise alors 93 / 52, la partie entière de cette division, 1, signifie qu'on aura un message de taille '*normale*. Le reste de cette division, 41, signifie que pour le second message à envoyer, n'enverra que 31 octets de données. Si le reste est plus petit ou égal à 10 on supprime ce message car il consistera seulement de l'entête.

A la réception d'un acquittement, on test si ecn est active, si c'est le cas on divise la taille de la fenêtre par 0.1, soit 10% de réduction de la taille. On test aussi si On a 3 messages perdus à la suite à l'aide d'un compteur qu'on incrémente si l'acquittement reçu est égal au dernier reçu. Si 3 messages sont perdus on réinitialise la taille a 52. Si les messages sont bien reçus on incrémente la taille de 52.

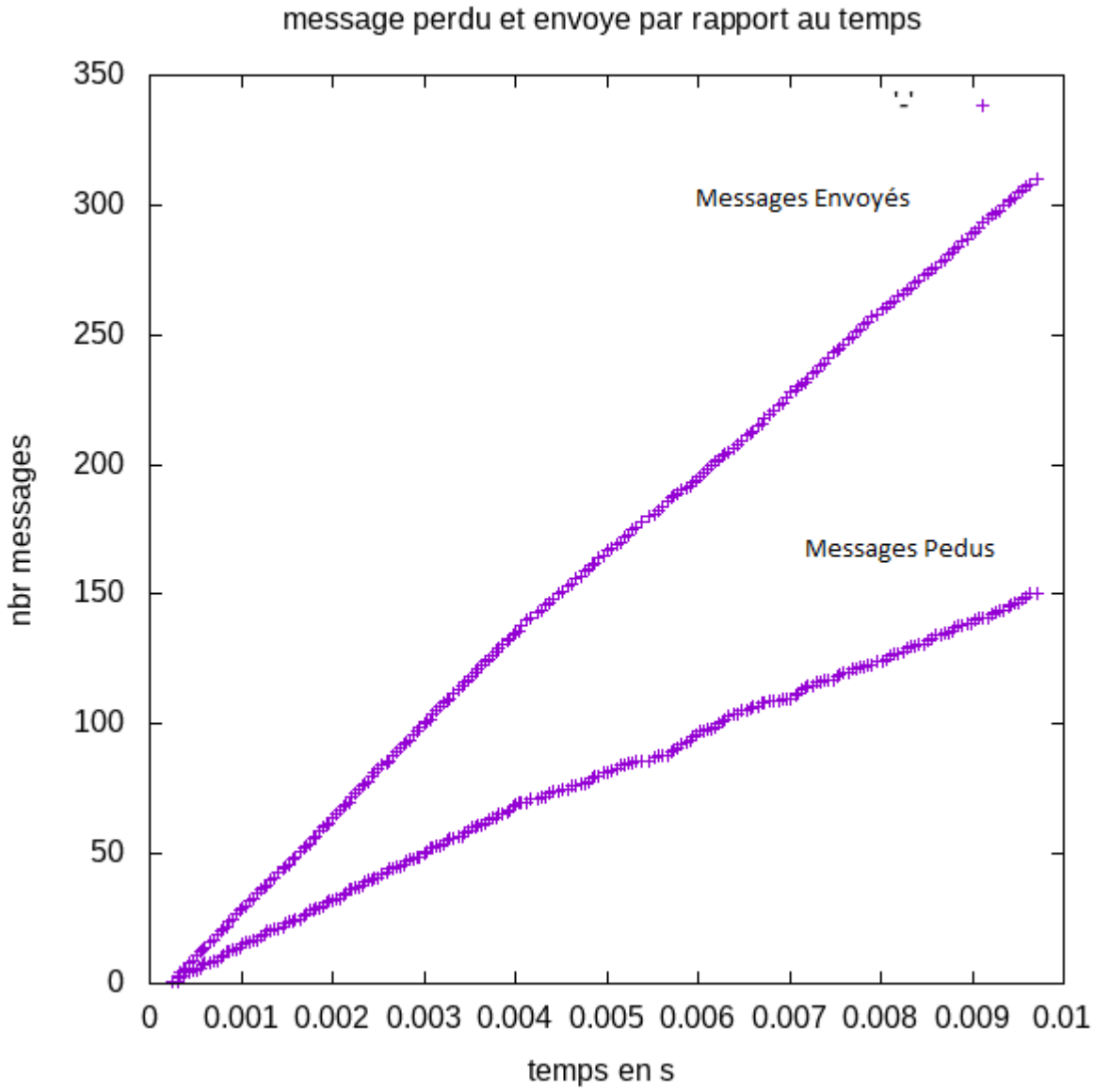
e. Fermeture

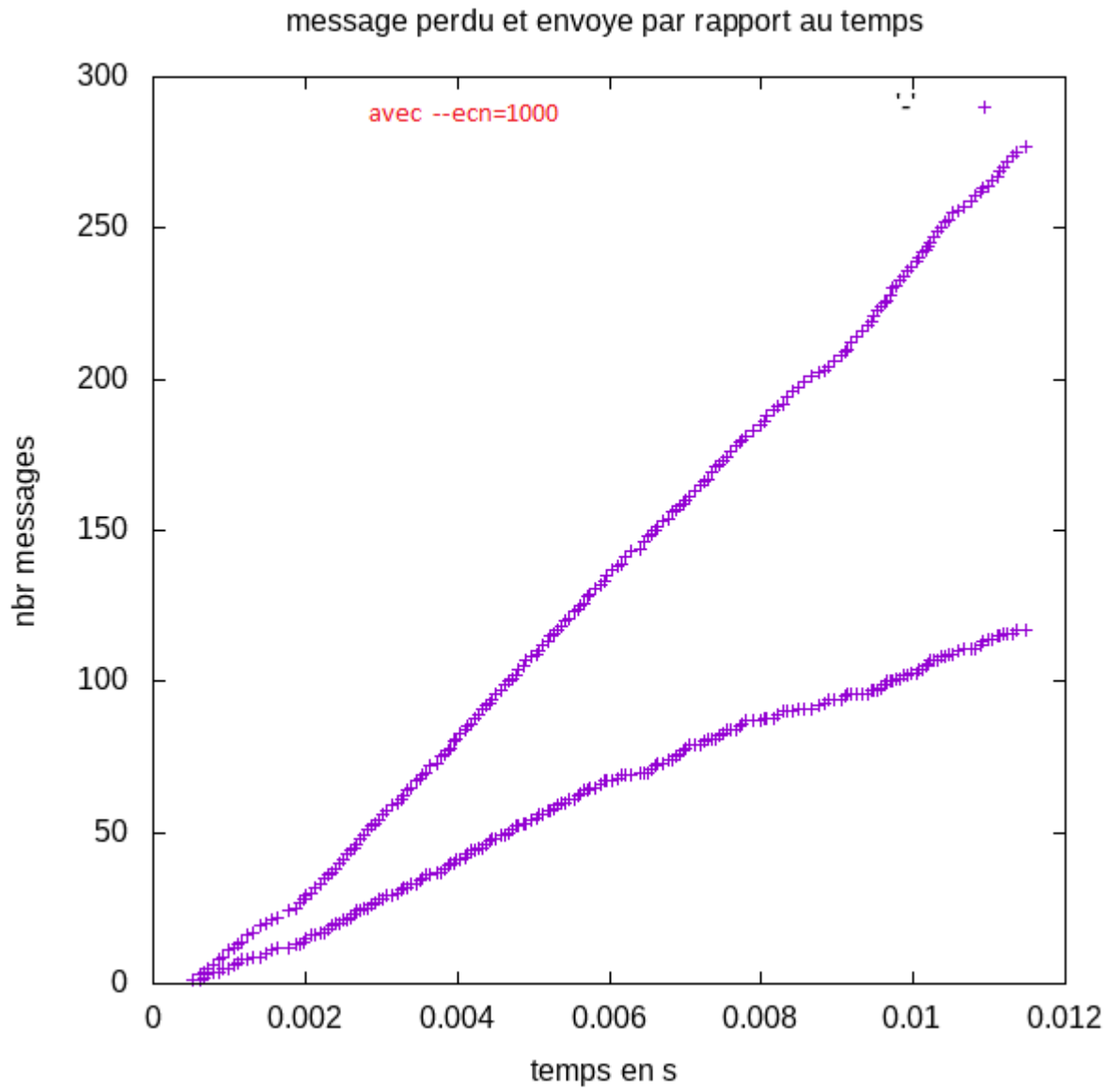
Pour une fermeture correcte, le procédé mis en place est le 4 way handshake. Il s'assure que le client et le serveur sachent qu'ils doivent fermer la connexion. Durant cette étape on vérifie bien que le socket est fermée et que toute mémoire allouée et libérée.

III. Résultats

Pour mieux visualiser les résultats obtenus, nous utiliserons gnuplot pour dessiner nos figures. Pour Stop N Wait, on dessine le nombre de messages envoyés et perdus par rapport au

temps, avec et sans ECN active.





En comparant ces deux figures, on constate que quand ECN est non active, le nombre de messages perdus et envoyé est plus important, cependant le temps d'envoi est moins important. Ce résultat est attendu car quand la simulation de congestion est active les messages prennent plus de temps à parvenir à leur destination.

Pour le Go-Back-N, les données sont indisponibles car nous n'avons simplement pas réussis à l'implanter. On peut quand même supposer que le temps d'envoie des données serait moins important.

IV. Conclusion

Pour conclure, ce projet était très intéressant car il montre bien les problèmes gérés par les protocoles TCP, mais aussi son optimisation de l'envoie de paquets pour, avec Go-Back-N. On trouve cela dommage de ne pas avoir été capable d'implémenter ce dernier, car comparer ses données avec celles de Stop N Wait nous aurait permis de voir l'évolution du débit et les différences des deux.