Projet 1 - Rapport

Antoine Gennart

October 22, 2018

1 Introduction

Dans le cadre du cours LINGI1341 : *Computer networks : information transfer*, il nous a été demandé de réaliser un protocole de transfer fiable basé sur des segments UDP.

Ce rapport décrit l'architecture globale du projet, et répond aux questions imposées dans le cadre du projet.

2 Architecture

- **sender.c** Implémente la fonction principale du sender.
- **receiver.c** Implémente la fonction principale du receiver.
- **pkt.c** Implémente toute la structure des packets, ainsi que toutes les fonctions pour manipuler ces packets.
- **network.c** Implémente la connection UDP entre deux machines.
- **utils.c** Contient un certain nombre de fonction utiles et qui n'avaient pas leur place dans les autres fichiers.
- **fifo.c** Implémente la mémoire FIFO (*first in first out*) qui va être utilisé par le receiver pour gérer les *acknoledgment*

3 Questions

3.1 Que mettez-vous dans le champs Timestamp, et quelle utilisation en faites-vous?

Le champs timestamp contient un nombre de microsecondes. Il est calculé comme suit :

$$\text{timestamp} = t_{epoch} \% (2^{32} - 1)[\mu s]$$

ou t_{epoch} correspond au temps écoulé depuis l'epoch en microsecondes et % représente l'opération modulo.

Le timestamp étant enregistré sur 32 bits, il est limité dans sa précision. Utiliser la valeur calculée ci dessus correspond à un cycle d'environ 1 heure entre chaque réinitialisation du timestamp.

3.2 Comment réagissez-vous à la réception de paquets PTYPE_NACK?

Lorsque le *sender* reçoit un packet de type PTYPE_NACK, il sait que le packet n'a pas été correctement transmit au *receiver*, il va donc directement renvoyer un packet sans prendre en compte la valeur du retransmission timeout.

3.3 Comment avez-vous choisi la valeur du retransmission timeout?

La valeur du *retransmission timeout* est calculé via l'algorithme de *Van Jacobson*. Comme décrit dans le cours (p. 159 du syllabus), il est initialisé de la manière suivante :

$$\begin{aligned} srtt &= rtt \\ rttvar &= \frac{rtt}{2} \\ rto &= srtt + 4 \cdot rttvar \end{aligned}$$

Lorsque d'autres mesures du rtt arrivent, nous pouvons mettre a jours les valeurs de rtt, srtt et rto comme suit:

$$\begin{aligned} rttvar &= (1-\beta) \cdot rttvar + \beta \cdot (srtt - rtt) \\ srtt &= (1-\alpha) \cdot srtt + \alpha \cdot rtt \\ rto &= srtt + 4 \cdot rttvar \end{aligned}$$

Dans le cadre du projet, j'utilise les valeurs conseillées dans le syllabus, a savoir $\alpha = \frac{1}{8}$ et $\beta = \frac{1}{4}$. Ces valeurs particulières permettent de remplacer des *floating point operations* par des *bits shifts*, ce qui est beaucoup moins couteux pour le processseur.

3.4 Quelle est la partie critique de votre implémentation, affectant la vitesse de transfert?

La partie la plus critique est l'attente d'un *acknowledgment* lorsque qu'un packet a été perdu. Cela nous fait perdre le temps d'un *retransmission timeout*. Sur un réseau local, on mesure un retransmission timeout inférieur à 1 milliseconde.

3.5 Quelles sont les performances de votre protocole?

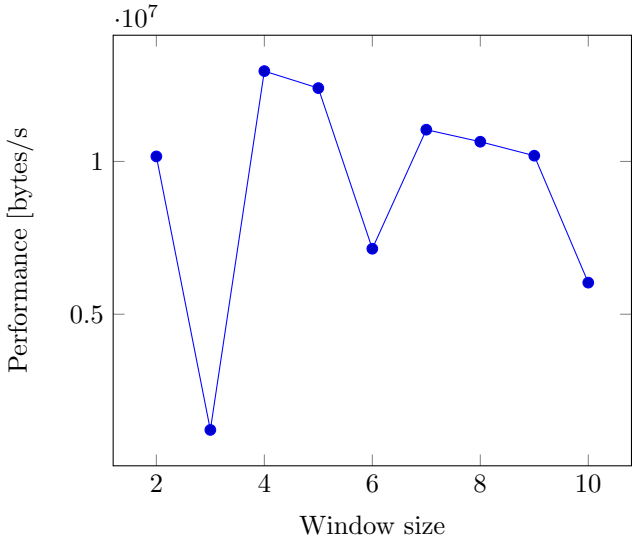

Figure 1: Mesure de la performance en fonction de la taille de la *sliding window*

3.6 Quelles sont les stratégies de test que vous avez utilisées?

La stratégie de test consiste a envoyer et recevoir un fichier, pour ensuite comparer le fichier d'entrée avec le fichier de sortie. Si ces deux fichiers sont différents, le test échoue.

Une seconde stratégie consiste à réaliser exactement la même chose que précédement, mais en utilisant un simulateur de lien prévu à cet effet.