Projet de réseaux informatiques (LINGI-1341)

Benoît Michel(2397 16 00) et Edouard Chatzopoulos(3740 16 00)

1 Introduction

Dans le cadre du cours LINGI-1341, il nous a été demandé de réaliser deux programmes d'envoi et de réception de paquets UDP sur base d'une architecture IPV6 afin d'assister l'entreprise Croustic-croc pour l'installation de son nouveau data-center. Pour cela, nous avons entrepris l'implémentation d'un sender et d'un receiver fonctionnant avec une stratégie de selective repeat.

2 Implémentation

2.1 Timestamp

Le champ timestamp permet de connaître le temps de vie du paquet sur le réseau. On y stocke le temps actuel lorsqu'on crée le paquet. Ensuite, lorsque le receiver reçoit le paquet, il compare le temps actuel avec ce qui est contenu dans le timestamp et il sait ainsi depuis combien de temps ce paquet est dans le réseau. Ce système permet de pouvoir choisir des informations actuelles et de pouvoir comparer deux paquets avec le même numéro de séquence pour savoir lequel est le plus récent et donc le plus fiable.
Le timestamp permet également de fixer le temps pour le retransmission timer. En prenant le timestamp multiplié par deux, on connaît, à une certaine précision près, le RTT (round-trip-time) et on peut fixer, selon un certain facteur, le retransmission timer le plus approprié. Une telle pratique permet d'optimiser la vitesse de transmission sur le réseau : on passe moins de temps à attendre un ack perdu qui n'arrivera jamais et on ne renvoit pas trop tôt des frames dont on a pas encore eu le temps de recevoir la confirmation de bonne réception.

2.2 Gestion des PTYPE_NACK

En cas de réception d'un paquet de type PTYPE_NACK, on sait que le paquet PTYPE_DATA correspondant n'a pas été reçu ou a été tronqué. Dans ce cas, il faut renvoyer le paquet de données concerné (identifié par le numéro de séquence contenu dans le paquet PTYPE_NACK). On ne va pas renvoyer tous les paquets dans la fenêtre d'envoi à partir du numéro de séquence reçu car on implémente ici un selective repeat, il est donc possible que les paquets de données suivants soient déjà reçus. Les renvoyer encombrerait le réseau pour des informations redondantes. Si les paquets de données suivants n'étaient pas reçus, on les renverra lorsque le récepteur enverra un paquet de type PTYPE_NACK pour ceux-ci ou lorsque le retransmission timer arrivera à la fin du timeout.

2.3 Retransmission timeout

Le retransmission timer doit être choisi de manière à ce qu'il soit un peu plus long que le temps d'un RTT (round-trip time). S'il est trop court on prend le risque d'envoyer trop de paquets en cas de congestion, et ainsi d'empirer la situation. S'il est trop long par contre, on rallonge le délai en cas de perte de paquet et cela affecte négativement les performances du programme.

2.4 Vitesse de transfert

La vitesse de transfert de notre implémentation va dépendre de plusieurs paramètres. En fonction du retransmission timeout, l'émetteur et le récepteur passeront plus ou moins de temps à attendre l'autre. L'idéal serait que le retransmission timeout soit adapté à chaque échange et varie mais cela semble ardu à implémenter. Un autre paramètre qui va affecter la vitesse de transfert, c'est le taux de paquets corrompus et perdus sur le réseau. En fonction de ce taux, le temps pour échanger un paquet de données pourra passer du simple au double voire pire...

2.5 Performances

N'ayant pas encore terminé le projet total, nous n'avons pas encore pu évaluer les performances de notre algorithme.

2.6 Stratégie de test

N'ayant pas réussi à mener l'implémentation à son terme, nous n'avons pas encore eu l'occasion de tester notre projet dans son entièreté. Seules certaines sous-fonctions importantes ont été testées en local. De plus, la plupart de nos fonctions disponibles sur Inginious pour nous aider à débuter le projet ont passé les tests mis à disposition.

3 Interopérabilité

Les tests d'interopérabilité effectués n'ont mené qu'à des échecs étant donné que notre programme n'est pas encore fonctionnel à 100%. Nous faisons face à des pertes de données, à des corruptions des bits voire une mauvaise réception. Nous avons été pris de court : les tests d'interopérabilité ont été disponibles alors que notre programme n'était pas encore entièrement pensé et codé. Nous n'avons donc su commencer les tests que trop tard pour rectifier le tir et adapter le code en fonction des résultats.

4 Conclusion

Malgré nos effort le programme compile enfin mais ne semble pas fonctionner correctement. Il nous reste donc encore du travail afin de pouvoir mettre en application le programme de transfert de paquets avec le protocole IPv6. Cependant nous avons acquis de nombreuses nouvelles connaissances que nous allons continuer d'approfondir au cours du quadrimestre. Ce travail nous a fourni une solide base pratique sur le fonctionnement et la communication des réseaux informatiques.