Synthèse BE Réseau

Abderrahman Elouali, Nathan Billard

I. Introduction

Le but de ce bureau d'étude était de programmer un protocole pour permettre une meilleure transmission de vidéo que TCP dans un réseau avec beaucoup de pertes.

II. Notre avancée dans le projet

Nous sommes arrivés à la fin de la version 4.2. Nous avons commencé par écrire des macros et des fonctions pour aider à la visualisation et au débogage.

III. Choix remarquables

1. Gestion de la fiabilité partielle

   Nous avons choisi de permettre un pourcentage d'acceptation de perte entier entre 0 et 100. Nous avons un buffer de chars de taille 100, dont on initialise toutes les valeurs à 1. Quand le client/source émet un pdu, mais qu'il ne reçoit pas d'ack, alors il va mettre une des valeurs dans ce buffer à 0. Puis il appelle une fonction qui permet ou non d'accepter cette perte, suivant le seuil négocié durant l'établissement de la connexion.

2. Représentation d'un socket

   Nous avons pris l'approche la plus simple que nous avons pu imaginer, c'est-à-dire de n'avoir qu'un seul socket. Notre application ne peut donc pas être lancée plusieurs fois dans le même mode sur une même machine sans dysfonctionnement.

3. Gestion de la synchronicité côté puits/serveur

   Pour gérer l'établissement de la connexion côté serveur/puits nous avons besoin d'utiliser un mutex condition que nous bloquons dans la fonction accept quand on attend l'ack du client/de la source. Nous broadcastons

ensuite depuis le thread qui fait régulièrement appel à la fonction process_recieved_pdu.
C'est d'ailleur nous qui lançons ce thread dans la fonction mic_tcp_socket si l'application est lancée en mode serveur.

Conclusion

Nous avons fait les tests avec la version texte et ces tests semblaient très concluants, cependant nous n'avons pas eu le temps de faire les tests avec la vidéo.