Eléments de correction du TD4 (couche liaison de données - niveau 2)

Au niveau de la couche liaison, la trame encapsule le paquet fourni par la couche réseau (niveau 3) et ajoute des en-têtes et en-queues spécifiques.

Quelques notes sur le programme avec Nachos :
* On travaille dans le répertoire reseau, sur le fichier reseau/liaison.cc (version 1, 2, ...).
* La fonction main est dans threads/main.cc, qui appelle le code de l'émetteur ou du récepteur dans nettest.cc.
* Plus précisément, l'émetteur (./nachos -m 1 -l 1) envoie 10 paquets avec la commande coucheLiaison->EnvoyerPaquet(...) et le récepteur (./nachos -m 0 -l 1) est en attente de réception des paquets avec la commande coucheLiaison->RecevoirPaquet(...).
* La couche liaison utilise trois threads dont le comportement est implanté dans le fichier reseau/liaison.cc : DemonReception, ProtocoleEmission et ProtocoleReception.
* Le rôle de démon réception est de recevoir la trame depuis la couche physique, d'ajouter la trame dans un tampon, d'analyser le header de la trame et d'envoyer un événement au thread ProtocoleEmission ou au thread ProtocoleReception selon la nature de la trame (info ou ack).
* Dans les versions 1 à 3, la taille du tampon en émission est de 5 et de 1 en réception. Remarque : Les tampons sont bloquants sur l'ajout d'une trame qd il n'y a plus de place. C'est pourquoi, on observe 6 tentatives d'envoi lors du lancement du programme émetteur, la 6ème étant en attente bloquant qu'il y a de la place pour ajouter la nouvelle trame dans le tampon d'emission. Côté réception, une taille du tampon fixée à 1 signifie simplement : dés que je reçoit, je traite.
* A compléter...

1) Version simple (liaison-v1.cc)

Principe : envoyer et attendre.

L'émetteur attend que le récepteur ait eu le temps de traiter la trame envoyée avant d'envoyer la suivante.

Dans cette première version du protocole, le récepteur de la trame envoie un acquitement (ack). Cet acquittement est attendu par l'émetteur avant d'envoyer une nouvelle trame. Après ce délai de temporisation, si l'acquittement n'est pas parvenu à l'émetteur, alors il décide de réémettre la trame.

Problème des doublons :

Quand on perd un ack, on envoie plusieurs fois la même trame.

Autres difficultés :

a) si la durée de temporisation est trop courte, alors on a pas le temps de recevoir les acks...
b) si la durée de temporisation est trop longue, perte d'efficacité...

2) Version sans doublon en réception (liaison-v2.cc)

Principe : Pour remédier au problème des doublons, numérotation sur 1 bit des trames émises (côté émetteur uniquement), i.e. trame 0, puis trame 1, puis trame 0, ...

Dans cette version du protocole, on numérote sur 1 bit la trame (information stockée dans le header de la trame). Cette information permet de vérifier à la réception si la trame n'a pas été déjà reçue lorsque le problème des doublons survient. Dans ce cas, il suffit de jeter la trame reçue en double.

Un problème survient lorsque l'ack arrive après le délai de temporisation : une nouvelle trame est réémise et deux acks vont potentiellement être reçu. Comme les acks ne sont pas numérotés, cela peut entraîner l'ack d'une trame qui se sera perdue !

3) Version du bit alternée (liaison-v3.cc)

Principe : On remédie au problème précédent en numérotant (toujours sur 1 bit) les trames et les acks.

Il n'y a plus de problème dans cette version du protocole (fiabilité). En revanche, il y a une faible utilisation de la bande passante.

4) Version fenêtre coulissante (liaison-v4.cc)

Principe : Protocole à fenêtre coulissante sans rejet sélectif.

On envoie un nombre prédéterminée de trames sans avoir reçu l'acquittement de la première. Le récepteur rejette toutes les trames qui ne correspondent pas au numéro attendu.

Cela suppose que l'on va numéroter les trames et les acks avec un entier et non plus un simple bit.

L'émetteur doit s'arrêter d'émettre lorsqu'il n'y a plus de place dans son tampon d'émission. Il doit alors attendre un acquittement avant de faire de la place dans son tampon, et de continuer son émission.

Gestion des acks : Lorsqu'un acquittement est reçu hors-séquence, ce dernier est ignoré, et une réémission de la trame correspondante est effectuée.

Problème : Lorsque l'application (côté récepteur) ne consomme pas suffisament vite les trames reçues, le récepteur est obligé de jeter les trames même si elles ont été correctement acheminées.

Remarques sur le code :

 * on utilise un tampon de taille a priori non limitée pour le démon (SyncList), une sorte de file à accès synchronisée

 * on utilisera dans le code la méthode TenterAjout() des tampons (Tampon) qui retourne -1 si le tampon est plein, au lieu de bloquer comme Ajouter()

5) Contrôle de flux (liaison-v5.cc)

A compléter...