Maîtrise Informatique 2003 / 2004

Devoir de TéléInformatique et réseaux

« Simulation d'une liaison HDLC »

Julien Van Den Bossche / Benoît Moulin

Sommaire

1. Introduction, généralités

On souhaite faire communiquer des clients X25 par l'intermédiaire d'une liaison HDLC.
En fait, on envoi des paquets X25 à travers une liaison HDLC en les encapsulant dans des trames HDLC. On fait parcourir une trame HDLC d'une extrémité à une autre de la liaison.
La liaison comporte plusieurs voies logiques. Nous en avons choisi 4 : #A, #B, #C, #D.
Les clients X25 communiquent en établissant des circuits virtuels. Un circuit virtuel peut comporter plusieurs voies logiques en fonction de l'itinéraire que prennent les clients X25.
Dans notre devoir, étant donné que l'on ne s'occupe pas de clients X25 mais des paquets qui arrivent à l'une et l'autre extrémité de la liaison, on ne s'occupera pas vraiment des circuits virtuels (on les créent quand même, pour le niveau conceptuel) mais plutôt de la voie logique concernée pour la connexion au niveau de la liaison.
Le fichier source contenant les paquets X25 est supposé sans erreur.
Les bits des trames sont numérotés modulo 8.
Les sources se trouvent sur http://www.julienvdb.com/reseau

2. Spécifications, choix importants

2.1. PaquetX25 et Trame HDLC

Comme il est demandé dans le devoir, on doit mettre au début du paquet X25 l'extrémité qui envoie le paquet à travers la liaison. Notre paquet X25, intégrera donc dans son modèle le numéro d'extrémité.
Concernant la Trame HDLC, on l'a faite hériter d'un paquet X25 (utilisation d'un langage objet, JAVA). En effet, on doit encapsuler un paquet X25 dans une trame HDLC, et il est aisé, de représenter ce mécanisme par héritage de classe.
Même si cette modélisation objet, n'est pas totalement correcte car on transmet des octets et non des objets dans une liaison HDLC en réalité, on l'a adopter car elle est aisé à manipuler, à modifier, à réutiliser et facilement compréhensible par le monde extérieur.

Exemples :

Paquet X25 : 2-@6,@2,#A,AE
Trame HDLC : 2-0/0/0 @6,@2,#A,AE

2.2. Gestion des voies logiques

La prise d'une voie logique se fait de deux manières :
Soit par un appel *A*, ou bien par un appel entrant *AE*.

La libération d'une voie se fait uniquement si on a une demande de libération *DL*.

Si on a un appel et que la voie n'est pas libre, alors on renvoi un message d'erreur.
Si une voie est libre et que l'on a une demande de libération alors on ne change pas l'état de la voie et on renvoi un message d'erreur.

2.3. Gestion des tours de paroles

Pour gérer les tours de paroles entre les extrémités de la liaison on a procédé comme suit :
-Au départ, c'est l'extrémité qui a demandé une connexion (*SABM*) qui prend la parole en premier.
-Ensuite, si on se trouve dans le cas où les deux extrémités ont des paquets à transmettre :
    -Si au tour d'avant une extrémité a émis un *RNR* alors le tour de parole restera à cette extrémité pour bloquer l'envoi de l'autre côté jusqu'a récupérer un *RR*.
    -Sinon on tire au hasard une des deux extrémités et on traite un paquet :
        -Si le paquet contient *CA* ou *CE* et que la voie logique sur laquelle il communique est libre alors on ne traite pas ce paquet et on demande de traiter l'autre extrémité. En effet, cela signifie que l'on a une confirmation de connexion sans avoir eu d'appel. On va donc chercher d'abord l'appel.

-Si une extrémité n'a plus rien à transmettre alors le tour de parole est donné à l'autre extrémité

2.4. Acquittements explicites

Un acquittement explicite dans une trame HDLC est demandé quand on a un nombre de trames reçues sans acquittement égal à 7 (fenêtrage par défaut pour HDLC).
Dans notre simulation, nous envoyons la trame avec le bit *PF* à 1 et :
-Si l'autre extrémité, recevant la trame envoyé, n'a plus rien à transmettre alors elle acquitte les trames reçues en envoyant une trame HDLC avec *NS* valant *RR* et le bit *NR* acquittant les trames reçues.
-Sinon elle renvoie un paquet à envoyer en acquittant avec le bit *NR*.

3. Scénarios possibles, déroulement

On crée une liaison avec ses voies logiques libres et deux extrémités. Chaque extrémité à des adresses X25 qui leur arrive dessus. L'extrémité a les adresses de 0 à 4 et l'extrémité 2 a les adresses de 5 à 9.
Ensuite on charge un fichier texte contenant les données des paquets X25. Pour chaque ligne du fichier on transforme ces lignes en objets PaquetX25. Selon l'extrémité du paquet on va insérer le paquet soit dans le vecteur de paquets à envoyer pour l'extrémité 1 ou dans celui pour l'extrémité 2.

-Pour chaque paquet à envoyer on regarde la validité des adresses, des voies demandées.
Attention : Dans le cas d'une erreur, le fichier continue à se traiter et les informations affichées à l'écran sont fausses.

-Le premier paquet traiter est le paquet de l'extrémité qui a demandé la connexion

-Dans tous les cas si une extrémité a émis 6 trames (et qui lui reste des paquets à envoyer) sans recevoir d'acquittements alors on envoi la $7^{ième}$ avec le bit *PF* à 1.
    -Si le destinataire n'a plus de paquet à envoyer alors il répondra par la trame *RR/0/n*
    -Sinon par le paquet qu'il encapsule, acquittant ainsi les trames en attentes.

-Dans tous les cas si on envoi une trame *RNR* on force le tour de parole sur l'extrémité qui veut l'envoyer jusqu'à obtenir le *RR*.

-Dans tous les cas si une extrémité n'a plus de paquet à envoyer, le tour de parole sera donné systématiquement à l'autre extrémité.

-Le cas d'arrêt est celui où les extrémités n'ont plus rien à envoyer.

-Dans les autres cas le tour de parole se fait aléatoirement (sauf *CE* sans A avant, cf. plus bas).

-Si c'est un appel (*A ou AE*)

On regarde la disponibilité de la voie logique demandée :
-Si elle est libre on la prend puis on encapsule le paquet et on envoi le message à l'autre extrémité.
-Si elle n'est pas libre on envoi un message d'erreur.

-Si c'est une confirmation de circuit établit (*CA, CE*) alors on regarde si la voie demandée est prise.

-Si elle ne l'est pas alors cela signifie qu'on a pas eu d'appel avant et on va forcer le tour de parole en traitant l'autre extrémité d'abord (où se trouve la demande d'appel) Le paquet sera traité, uniquement quand on aura eu l'appel.

-Si elle l'est on encapsule le paquet dans une trame HDLC et on l'envoi à l'autre extrémité.

-Si c'est une demande de libération alors on vérifie que la voie logique est bien prise
-Si c'est le cas on la libère et on encapsule le paquet dans une trame que l'on envoie.
-Si elle est libre on envoi un message d'erreur.

-Si c'est un envoi de données on encapsule le paquet et on l'envoi

4. Modélisation de la simulation en JAVA

JAVA étant un langage fortement orienté objet nous avons défini plusieurs objets permettant de modéliser notre simulation.

Pour pouvoir mieux visionner notre modèle (descriptions des classes, méthodes…) vous pouvez utiliser la commande javadoc qui permet de générer l'API de notre programme :
javadoc *.java
Des fichiers HTML sont alors créés. Le fichier « index.html » est le fichier d'index.
Ces fichiers sont aussi disponibles sur http://www.julienvdb.com/reseau/doc

Classes définies :

- Une classe Liaison
- Une classe Extremite
- Une classe AdresseX25

- Une classe Voie
- Une classe Cv
- Une classe PaquetX25
- Une classe TrameHDLC
- Une classe Simulation

Classe Liaison

Cette classe représente donc la liaison HDLC. Une liaison est composée de deux extrémités. La liaison comportera un certain nombre de voies logiques. Ces voies sont stockées dans un vecteur. Une voie sera représentée par un objet Voie.

Attributs de la classe :
Une liaison comporte deux extrémités, un ensemble de VL, deux vecteurs de paquets X25 à distribuer aux extrémités. La liaison comporte 4 VL.

-Extremite extremite1 ;
-Extremite extremite2 ;
-Vector vl ; // vecteur de Voie (class Voie)
-Vector pourExtremite1 ; //un vecteur contenant les paquetsX25 à transmettre à l'extrémité 1.
-Vector pourExtremite2 ; // un vecteur contenant les paquetsX25 à transmettre à l'extrémité 2.

Constructeur :
public Liaison() //initialise les VL, crée les extrémités.

Méthodes :
-public Extremite getExtremite (int witch) //permet de renvoyer une extrémité selon son numéro
-public Vector getPourExtremite(int n) //renvoi les paquets X25 à envoyer pour une extrémité
-public void addPourExtremite(int n, PaquetX25 p) //ajoute un paquet X25 dans le vecteur de l'extrémité n
-public Vector getVectorVL() //renvoi les voies logiques

Classe Extremite

Attributs
-int numero; //le numéro de l'extrémité
-AdresseX25[] adressesX25 ; //les clients X25 qui arrivent sur l'extrémité
-int numeroTrameRecu ; //le nombre de trames reçues, et numéro de la trame attendue modulo 8
-int numeroTrameEnvoye ; //le nombre de trame envoyée modulo 8
-int nbTrameSansAcq ; //le nombre de trames envoyées sans acquittements

Constructeur
-public Extremite(int numero, AdresseX25[] adresses)

Méthodes
- public TrameHDLC encapsule(PaquetX25 paquet, boolean flagAcq) //permet d'encapsuler un paquet X25 dans une trame HDLC. Le flag permet de savoir si on met le bit PF à 1 ou 0

- public void envoiEncapsulation(Extremite destinataire, TrameHDLC t) //permet d'envoyer une trame HDLC à travers la liaison à l'autre extrémité.
On incrémente le compteur des trames envoyées pour l'extrémité émettrice, et on augmente les compteurs des trames reçues pour le destinataire. On remet à zéro le compteur des trames reçues sans acquittement chez le destinataire.

-Des méthodes d'accès aux attributs : public int getNumero() ; public AdresseX25[] getArrivants() ; public int getNumeroTrameRecu() ; public int getNumeroTrameEnvoye() ; public void setNumeroTrameRecu() ; public void setNumeroTrameEnvoye() ; public int getNbTrameSansAcq() ; public void setNbTrameSansAcq(int n)

Classe AdresseX25

Cette classe représente les adressesX25 qui sont contenues dans un les paquets X25.
Une AdresseX25 possède un nom.

Classe PaquetX25

Comme on l'a vu plus haut nous avons décidé de prendre en compte l'extrémité dans le paquet X25.
Un paquet X25 peut avoir plusieurs formes :
-Soit être un paquet pour l'établissement de circuit virtuel (AE, A, CE, CA), dans ce cas le paquet contient une extrémité, une adresse X25 de l'émetteur, une du récepteur, une voie logique et une donnée.
-Soit un paquet pour l'envoi de données ou bien pour une demande de libération. Dans ce cas on n'utilise plus les adresses X25.

On a donc deux constructeurs :
-public PaquetX25(Extremite extr, AdresseX25 adr1, AdresseX25 adr2, Voie vl, String d)

-public PaquetX25(Extremite extr, Voie vl, String d)

Méthodes :
Les méthodes sont des méthodes d'accès aux attributs.

Classe Voie

Elle permet de définir une voie logique. Cette dernière possède un nom et un état (libre ou non). Au départ toutes les voies logiques sont libres.

Constructeur
public Voie(String nom)

Méthodes
Les méthodes sont des méthodes d'accès aux variables.

Classe Cv

Elle permet de définir un circuit virtuel. Un CV possède une voie logique et deux adresses X25. Au départ, il n'y a pas de CV créé.

Constructeur
public Cv(Voie v, AdresseX25 adr1, AdresseX25 adr2)

Méthodes
Les méthodes sont des méthodes d'accès aux variables.

Classe TrameHDLC

Cette classe permet de représenter une trame HDLC. Etant donné que cette dernière encapsule un paquet X25 nous avons décidé de faire hérité cette classe de la classe PaquetX25. Comme on a plusieurs type de paquets on aura plusieurs types de TrameHDLC.

Constructeurs :
-public TrameHDLC(Extremite extr, String ns, String pf, String nr,AdresseX25 adr1, AdresseX25 adr2, Voie vl, String d)

-public TrameHDLC(Extremite extr, String ns, String pf, String nr, Voie vl, String d)

-public TrameHDLC(Extremite extr, String ns, String pf, String nr)

Le dernier constructeur permet de créer une trame sans encapsuler de paquet X25.
Exemple : réponse à une demande d'acquittement quand on a plus de paquet à transmettre : 1-RR/0/3

Méthodes :
Des méthodes de types « get » et « set » d'accès aux attributs.

Classe Simulation

C'est cette classe qui va utiliser tous les objets décrits plus haut pour démarrer la simulation. Dans cette classe on ne décrira pas les attributs et méthodes qui nous servent à faire notre interface graphique.

Attributs
Liaison liaisonHDLC; //la liaison HDLC qui va être simulée
Vector vl; //un vecteur qui permettra de stocker provisoirement les voies logiques
Vector donneesX25 = new Vector(); //stocke chaque info d'une ligne du fichier d'entrée dans un vecteur
int numeroDepart = 1; //le numéro de l'extrémité qui communique en premier : 1 par défaut
int nbSim; //un compteur qui permet de connaître le nombre de simulation effectuées
Vector vecteurCV = new Vector(); //les circuits virtuels créés

Constructeur
public Simulation(), crée un objet Simulation

Méthodes

-public Vector litDonnees(String donnees) permet de lire une ligne, formaté selon la norme demandé (cf poly R.Lozach) et de mettre chaque composant de cette ligne(vl, adresse, extremité…) dans un vecteur.

-public void transformeEnX25(Vector d) permet de transformer un vecteur de données (chaque élément du vecteur est une chaîne de caractères) en PaquetX25.

-public void traiteFichier(String url) permet de transformer cahque ligne du fichier source en PaquetX25 et les stocker dans le bon vecteur (à destination de l'extrémité 1 ou 2).

-public void ecrit(String l) permet de créer un fichier et d'écrire une ligne dedans. Le fichier texte sera nommé de la façon suivante : « resultats »+nième simulation effectuée+ « .txt »
Le fichier est stocké dans le répertoire courant de l'application.

-public void traiteVecteurX25(Vector v, int indice, boolean flagAcq) permet de traiter un paquetX25 lorsqu'il s'apprête à être envoyé par une extrémité. C'est dans cette méthode que l'on vérifie les ressources, la validité des données avant l'encapsulation et l'envoi.
flagAcq permet de savoir si on met le bit PF à zéro ou un. indice est l'indice du paquet que l'on prend de le vecteur (soit dans le vecteur de paquet que doit envoyer l'extrémité 1 ou bien celui de l'extrémité 2).

-public void boucle() permet de gérer les tours de paroles entre extrémité et de traiter tous les paquets à envoyer. Elle fait à la méthode ci-dessus une fois les vérifications faites.

-public String litFichier(String url) permet de lire un fichier et de mettre son contenu avec le même formatage dans une zone de texte

-public void creeLiaison(Liaison l) permet de créer une nouvelle liaison à chaque simulation

-public void fenetre() permet d'avoir une interface graphique pour la simulation

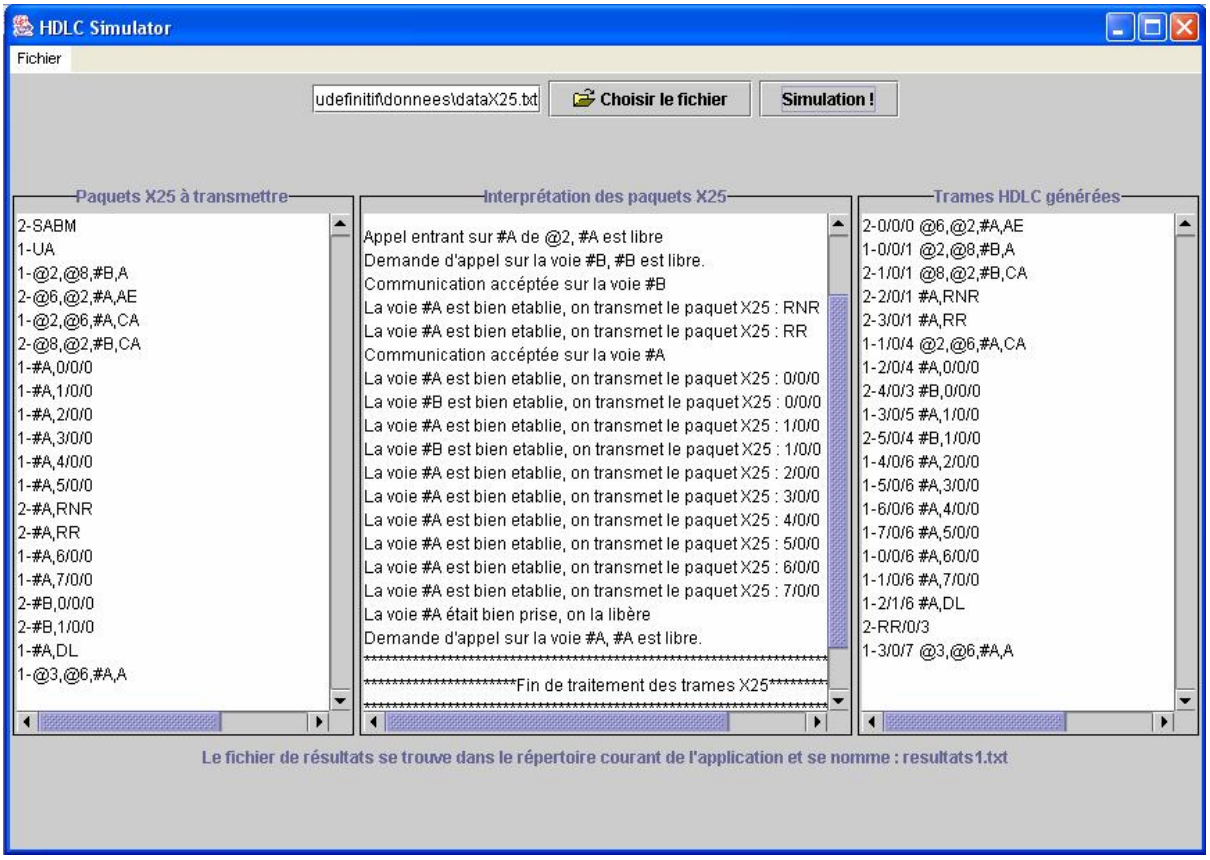
5. Présentation du programme

5.1. Présentation

Le programme fonctionne avec une interface graphique. On peut choisir l'emplacement du fichier texte, contenant les données X25 à charger par l'intermédiaire d'un bouton « parcourir ». Ensuite, le fichier apparaît dans la zone de texte de gauche.
Un bouton permet ensuite de lancer la simulation. A ce moment là, dans la zone de texte du milieu, on a une description des messages envoyés. Dans la zone de texte de gauche, nous avons les trames HDLC correspondants à l'encapsulation des paquets X25.
Un fichier texte se nommant « resultats »+nième simulation+ « .txt » se trouve dans le répertoire courant de l'application.

Le programme se trouve à l'adresse suivante : http://www.julienvdb.com/reseau
Les fichiers sources sont les fichiers .java. Les fichiers .class sont les fichiers compilés.
Pour compiler les fichiers sources, on peut utiliser la commande *javac *.java*
Pour exécuter le programme on le lance avec la commande *java Simulation*

5.2. Copie d'écran



6. Conclusion

Ce devoir, même s'il ne simule pas vraiment ce qu'est un réseau avec des liaisons HDLC nous ont bien permis de comprendre le fonctionnement des transmissions entre clients X25 d'un réseau. Les mécanismes d'encapsulation, d'acquittement, de réservation de ressources qui nous étaient un peu étrangers nous semblent beaucoup plus clairs à présent.