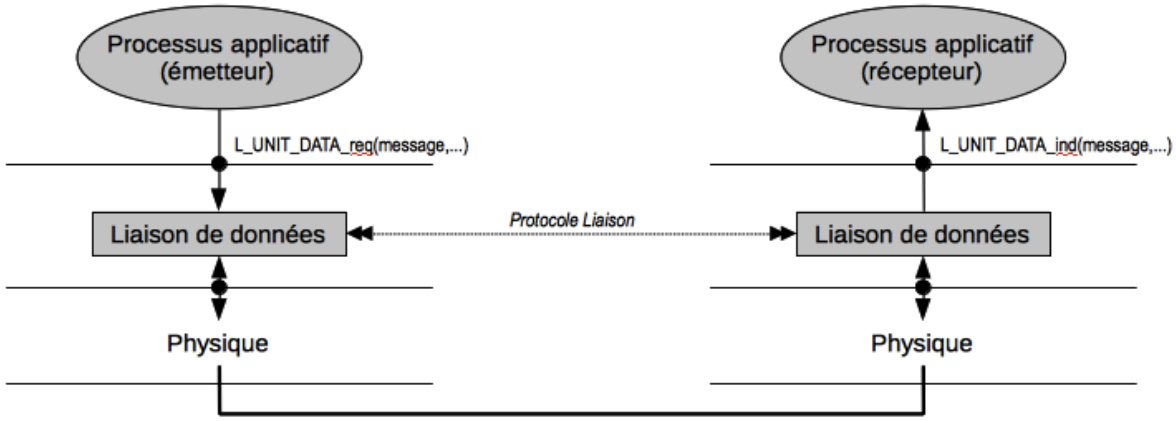
Projet réseaux : développement et test d'une couche liaison de données

1 Objectif général

On souhaite réaliser un programme qui implémente le fonctionnement de la couche liaison de données, ainsi qu'un programme de transfert de fichiers s'appuyant sur cette couche liaison de données. Le projet sera réalisé en langage C.
Exemple de mise en oeuvre en mode non connecté :



2 Éléments de spécification

2.1 Primitives pour accéder aux services de la couche liaison de données

La couche liaison pourra fonctionner soit en mode non connecté (phase unique de transfert de données), soit en mode connecté (trois phases successives : établissement de la connexion, transfert de données, terminaison de connexion).

Primitives pour le mode non connecté :

```
/* Primitive pour transférer une unité de données */
/* Equivalent dans le monde OSI : L_UNIT_DATA.req */
void emettre_sur_liaison(char* adr_src, char* adr_dest, char* msg, int lg_msg);

/* Primitive pour récupérer une unité de données */
/* Equivalent dans le monde OSI : L_UNIT_DATA.ind */
/* Valeur de retour : taille du message */
int recevoir_de_liaison(char* adr_src, char* adr_dest, char* msg);
```

Primitives pour le mode connecté :

```
/* Primitive de demande d'établissement d'une connexion */
/* Equivalent dans le monde OSI : L_CONNECT.req et L_CONNECT.conf */
/* Valeur de retour : réponse à la demande de connexion
   (1 acceptation, 0 refus) */
int etablir_connexion_liaison(char* adr_src, char* adr_dest);
```

```
/* Primitive pour notifier d'une demande de connexion */
/* Equivalent dans le monde OSI : L_CONNECT.ind */
void recevoir_demande_connexion_liaison(char* adr_src, char* adr_dest);

/* Primitive de réponse à la demande de connexion */
/* Equivalent dans le monde OSI : L_CONNECT.resp */
void repondre_demande_connexion_liaison(int reponse);

/* Primitive pour transférer une unité de données au sein d'une connexion */
/* Equivalent dans le monde OSI : L_TX_DATA.req */
void emettre_sur_liaison_connectee(char* msg, int lg_msg);

/* Primitive pour récupérer une unité de données au sein d'une connexion */
/* Equivalent dans le monde OSI : L_TX_DATA.ind */
/* Valeur de retour : taille du message */
int recevoir_de_liaison_connectee(char* msg);

/* Primitive pour mettre fin à une connexion */
/* Equivalent dans le monde OSI : L_DISCONNECT.req */
void terminer_connexion_liaison();

/* Primitive de notification d'une fermeture de connexion */
/* Equivalent dans le monde OSI : L_DISCONNECT.ind */
void recevoir_terminaison_connexion_liaison();
```

2.2 Structure des unités de données du protocole liaison

Les unités de données du protocole liaison (trames) sont structurées comme ci-dessous.
La taille maximale d'une trame sera de 100 octets.

Champ	Nb d'octets	Signification
`deb_trame`	1	Octet marquant le début de la PDU
`adr_dst`	6	Identification du destinataire de la trame
`adr_src`	6	Identification de l'émetteur de la trame
`ctrl`	1	Ce champ permet d'identifier le type de la PDU : 0 demande d'établissement de connexion 1 acceptation de connexion 2 refus d'établissement de connexion 3 demande de déconnexion 4 transfert d'une PDU de données 5 accusé de réception positif d'une PDU de données 6 accusé de réception négatif d'une PDU de données 7 autre
`num_seq`	1	Numéro de séquence de la PDU
`lg_info`	1	Longueur du champ `info`
`info`	n	Données (le message transmis)
`fcs`	1	Séquence de contrôle calculée en réalisant un « ou exclusif » bit-à-bit sur tous les octets contenus dans le champ `info`
`fin_trame`	1	Octet marquant la fin de la PDU

3 Bibliothèque « physique.h »

Vous disposez d'une bibliothèque qui simule le comportement d'une couche physique. Pour utiliser cette bibliothèque, vous devez commencer par appeler une des deux fonctions d'initialisation suivantes :

```
/* Initialisation simple pour tests en local
 * Paramètres : proba_perte  taux de pertes (entre 0 et 1)
 *              proba_erreur taux d'erreurs (entre 0 et 1)
 *              reception    vaut 1 pour récepteur (sinon émetteur) */
void initialisation_simple(float proba_perte, float proba_erreur, int reception);

/* Initialisation pour tests en distribué (émetteur et récepteur sur machines distinctes)
 * Paramètres : proba_perte  taux de pertes (entre 0 et 1)
 *              proba_erreur taux d'erreurs (entre 0 et 1)
 *              autres       paramètres réseau, cf. votre enseignant de TP */
void initialisation(float proba_perte, float proba_erreur, unsigned short port_local,
                    char* destination, unsigned short port_destination);
```

Primitives de service de la couche physique pour émission et réception sur le canal :

```
/* Réception d'une trame arrivée (cf. fonction Attendre)
 * Paramètres : trame   pointeur vers votre structure de données trame
 *              taille  doit être sizeof(trame_t) où trame_t est votre type trame */
void de_canal(void *trame, int taille);

/* Emission d'une trame (paramètres identiques à DeCanal) */
void vers_canal(void *trame, int taille);
```

Vous disposez enfin de fonctions annexes pour vous aider dans l'implémentation de votre couche liaison :

```
/* Attend un évènement
 * Retourne  0 si une trame reçue est disponible
 *           un numéro de timer si un timeout a été généré */
int attendre();

/* Démarre le timer numéro n (0 < n < 100) qui s'arrête
 * après ms millisecondes (ms doit être un multiple de 100) */
void depart_compteur(int n, int ms);

/* Arrête le timer numéro n */
void arreter_compteur(int n);
```

4 Travail à réaliser

On vous demande de développer une couche liaison de données répondant à différentes spécifications (sections 2 et 4.x), ainsi qu'une application de test de votre couche liaison. Cette application consistera en un transfert de fichier. L'émetteur lit un fichier qu'il remet par blocs à la couche liaison; les blocs seront inférieurs ou égaux au champ `info` de la trame (c'est à dire la MTU – *Maximum Transmission Unit*). Le récepteur écrit dans un fichier les données remises par la couche liaison.

La couche liaison devra répondre aux spécifications ci-dessous : sections 4.2, 4.3, 4.4 et 4.5 (pour chacun de ces cas, vous développerez le corps des primitives de service dans des fichiers sources différents, voire des dossiers différents). Pour chaque cas vous testerez l'application de transfert de fichier. Afin de tester vos protocoles, vous ferez varier le taux de perte et d'erreurs au niveau de la couche physique (côté émetteur et récepteur) : observez et analysez les résultats; comparez-les par rapport au comportement attendu.

4.1 Structure de données et primitives de service

Dans un fichier `couche_liaison.h`, placez la structure de données de la trame.
Dans un fichier `service_liaison.h`, placez la bibliothèque des primitives de service de la couche liaison.

4.2 Liaison sans contrôle de flux ni reprise sur erreurs

Dans le fichier `service_liaison0.c`, développez le corps des primitives de service en supposant un protocole liaison en mode non connecté, sans contrôle de flux ni contrôle/reprise sur erreurs.

4.3 Liaison avec contrôle de flux « Stop and Wait » sans reprise sur erreurs

Dans le fichier `service_liaison1.c`, développer le corps des primitives de service en supposant un protocole liaison en mode non connecté avec contrôle de flux de type « Stop and Wait », sans contrôle/reprise d'erreurs.

4.4 Liaison avec contrôle de flux et reprise sur erreurs « Stop and Wait PAR »

On souhaite maintenant développer un protocole liaison avec contrôle de flux et reprise sur erreurs de type « Stop and Wait PAR » (*Positive Acknowledgement and Retransmission*). La détection d'erreur se basera sur une somme de contrôle (champ `FCS`) réalisée par un « ou exclusif » bit-à-bit sur tous les octets contenus dans le champ `info` de la trame.

Ce protocole sera mis en oeuvre selon deux modes :

a) Mode non connecté : placez le corps des primitives dans le fichier `service_liaison2.c`
b) Mode connecté : placez le corps des primitives dans le fichier `service_liaison3.c`

4.5 Liaison avec fenêtre d'anticipation et reprise sur erreurs « Go Back N »

On considère ici un protocole de type fenêtre d'anticipation (la taille de la fenêtre d'émission sera de 7) en mode non-connecté et utilisant un mécanisme de reprise sur erreurs de type « Go Back N ».
Notez que pour ce cas seulement, il n'est pas souhaitable de développer avec les primitives liaison de données. On travaillera directement avec un programme émetteur et un programme récepteur, eux-mêmes invoquant les primitives d'émission ou de réception sur le canal.

5 Validation

Le projet est à réaliser individuellement.

Vous disposez de 5 séances de TP. La validation se fera lors de la 6ème séance (démonstration à l'enseignant du transfert de fichier via les différents protocoles de liaison que vous avez implémentés).

Le jour de la validation, vous devrez envoyer votre code source commenté par courrier électronique à votre enseignant de TP (archive des fichiers au format .zip ou .tar.gz).