

TP Programmation de protocoles de communication
Basé sur un TP de M1- Master IST, Université Paris-Sud

Ce TP a pour objectif d'initier à la programmation d'un protocole de niveau liaison sous des hypothèses de plus en plus restrictives et se rapprochant de la réalité.

Créer le sous répertoire ASR2-Réseaux/DataLink dans votre répertoire de référence et copier dans ce répertoire le contenu du répertoire /net/Exemples/ASR2-Réseaux/DataLink.

1. Cadre

On considère la liaison de données entre deux machines A et B reliées entre elles par un support.



Figure 1: Transmission de données de A vers B.

Interface avec les niveaux physique et application

A cette structure matérielle, on associe une architecture partielle des niveaux : réseau, liaison et physique (cf. figure 2).

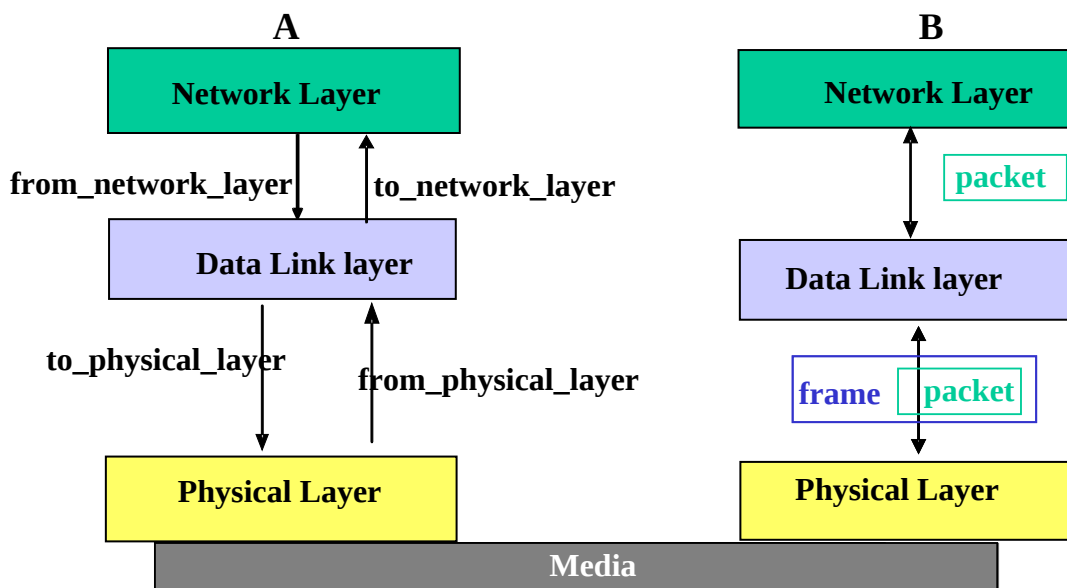


Figure 2 : Architecture partielle

Le niveau **liaison de données** décharge le niveau application des fonctions de transfert des données. Il récupère les données de la couche « Réseaux » sous forme de paquets, les envoie vers le destinataire (membre du réseau physique) de manière la plus fiable possible. Il assure aussi la réception des trames (frame, en anglais). Le niveau liaison de données est réalisé par le contrôleur de communication des machines A et B.

Le niveau **physique** assure la transmission physique des bits sur le support: codage bande de base, adaptation électrique, établissement du circuit, détection d'erreurs,... Le niveau physique comprend les interfaces ETTD/ETCD, les modems et support.

2. Structure de données et procédures communes

Les interfaces entre le niveau réseau et le niveau liaison de données d'une part, le niveau liaison de données et le niveau physique d'autre part, sont définies sous forme de procédures. Deux procédures permettent de prendre un

paquet à envoyer ou de donner un paquet reçu au niveau « réseau » : **from_network_layer** et **to_network_layer**. Le niveau « liaison de données » place alors le message à transmettre dans une structure d'enveloppe appelée trame (unité de données du protocole de niveau liaison). Pour transférer une trame, le niveau liaison utilise les services du niveau physique grâce à deux primitives du niveau physique: **from_physical_layer** et **to_physical_layer**.

Deux structures de données sont définies ici:

- *paquet (packet)* est l'unité de transfert du niveau application ; c'est une suite d'informations à transmettre de A vers B par la procédure de transmission ; la taille d'un paquet sera supposée constante de longueur maximum `MAX_PKT` octets.
- *trame (frame)* est l'unité de données du niveau liaison; elle est construite et gérée par la procédure de transmission. Elle est composée de plusieurs champs; certains champs spécifiques sont groupés dans une partie de l'enveloppe appelée "en-tête" ; ensuite viennent le champ d'information qui contient un paquet, suivi du caractère de contrôle d'erreur. La séquence de contrôle d'erreurs est calculée par le niveau physique.

Des procédures permettent la réalisation de fonctions couramment rencontrées dans les procédures de transmission: *wait_for_event*, *start_timer*, *stop_timer*, *incr*.

3. Hypothèses de travail

Votre répertoire de travail comporte un simulateur de l'architecture partielle présentée précédemment comportant une implémentation des structures de données et des procédures décrites ci-dessus.

Il prend en paramètre le protocole de communication à utiliser, ainsi que des indications sur le temps de simulation, les délais de retransmission, les erreurs simulées...

La ligne de commande doit impérativement respecter la structure suivante quel que soit le protocole :

```
protocol evenement timeout taux-perde taux-erreur trace
```

protocol est le nom du programme

evenement est durée de la simulation

timeout est le délai de retransmission

taux-perde est le taux en % de trames perdues

taux-erreur est le taux en % de trames reçues mais erronées (0= pas d'erreurs),

trace est 1 si mode trace sinon sa valeur est 0.

Scénario :

La couche réseau de la station A souhaite émettre des données vers la couche réseau de la station B. La station A possède donc des informations à transmettre jusqu'à la fin de la simulation. Le protocole n'a donc jamais à attendre la production de données.

Travail à réaliser :

Il s'agit de programmer, pour chaque protocole, deux procédures de communication qui réaliseront les fonctions du niveau liaison : une procédure d'émission de message et une procédure de réception de message. Le langage de programmation utilisé est le langage C.

Un compte rendu clair de ce travail, accompagné de votre code commenté, devra être rendu sous forme d'une archive comportant vos noms à l'adresse stephanie.moreaud@labri.fr.

Il vous est conseillé de lire la totalité du sujet avant de commencer.

Structures de données et procédures communes aux protocoles de communication (extrait de protocol.h »)

```
// constantes :
#define MAX_PKT 4 /* determines packet size in bytes */
#define MAXSEQ 2 /* 2 trames max */

// types :
typedef struct {unsigned char data[MAX_PKT];} packet; /* packet definition */
typedef enum {data, ack, nak} frame_kind; /* frame_kind definition */

typedef struct { /* frames are transported in this layer */
    frame_kind kind; /* what kind of a frame is it? */
    seq_nr seq; /* sequence number */
    seq_nr ack; /* acknowledgement number */
    packet info; /* the network layer packet */
} frame;

{ Procédures utilisées pour la programmation des différents protocoles }
type_evt wait_for_event()
    { attendre un évènement et retourner son type }
    { indispensable avant une réception physique qui n'est pas bloquante }
int incr( int k )
    { retourner l'incrément de k modulo MAXSEQ }

{ interface avec le niveau réseau }
void from_network_layer(packet *p);
    { lecture dans le tampon où la couche « réseau » écrit les paquets à émettre }
void to_network_layer(packet *p) { transmission d'un paquet reçu à la couche réseau }

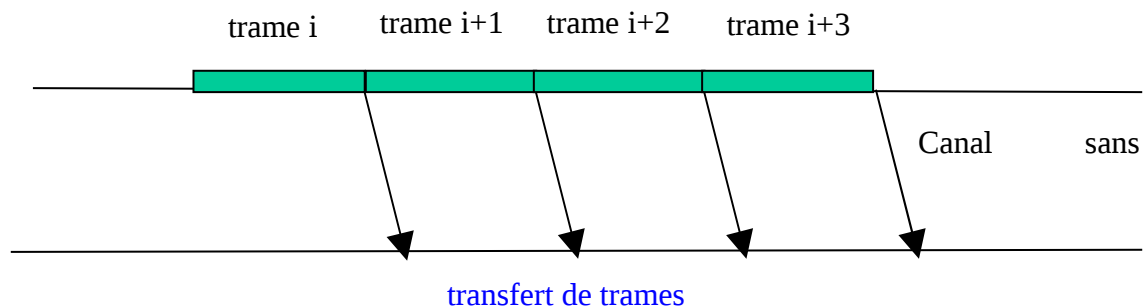
{ interface avec le niveau physique }
void to_physical_layer(frame *r){ émission de la trame stockée dans r }
void from_physical_layer(frame *r) { réception physique d'une trame stockée dans r }

{ gestion des temporisateurs }
void start_timer( int k )
    { armer le « timer » associé à la trame de numéro de séquence k }
void stop_timer(int k)
    { désarmer le « timer » associé à la trame de numéro de séquence k }
```

4. Protocole 1 (cas utopique)

Nous prendrons comme exemple initial un protocole très simple ; ses caractéristiques sont les suivantes :

- transmission de données uniquement de A vers B,
- A possède toujours des informations à transmettre,
- les temps de traitement sont négligeables,
- l'espace mémoire en réception est infini (pas de contrôle de flux),
- le canal est sans erreur.



Pour programmer ce protocole (peu réaliste), il a été écrit deux procédures **sender1** et **receiver1** (sans argument) dans le fichier `~/ASR2-Reseaux/DataLink/p1.c`.

Questions :

1. Représenter l'encapsulation des paquets dans une trame
2. Comprendre les deux procédures qui réalisent le protocole 1 en utilisant les structures de données et les procédures définies auparavant.
3. Compiler le protocole 1 (lire le fichier README) qui explique comment compiler.

Remarque : l'exécution de protocole 1 ne se termine jamais (bug du simulateur).

Un des résultats possibles de la simulation est :

```
./protocoll 100 40 0 0 0
```

```
Simulating Protocol 1
```

```
Events: 100 Parameters: 40 0 0
```

```
Tick 1. Proc 1 got protocol error. Packet delivered out of order.  
Expected payload 0 but got payload 300000
```

```
Process 0:
```

```
Total data frames sent:      305032  
Data frames lost:           0  
Data frames not lost:       305032  
Frames retransmitted:       0  
Good ack frames rec'd:     0  
Bad ack frames rec'd:      0  
  
Good data frames rec'd:     0  
Bad data frames rec'd:     0  
Payloads accepted:         0  
Total ack frames sent:     0  
Ack frames lost:           0  
Ack frames not lost:       0  
Timeouts:                  0  
Ack timeouts:              0
```

5. Protocole 2 (rajout d'un contrôle de flux)

Revenons à une situation plus réaliste, et considérons que le récepteur ne peut pas traiter les données entrantes aussi vite qu'elles sont émises. On relâche l'hypothèse selon laquelle l'hôte récepteur dispose d'une quantité d'espace mémoire infinie. Par contre, on suppose toujours que le canal est sans erreur.

L'émetteur doit transmettre à une vitesse inférieure à la capacité d'absorption du récepteur afin d'éviter l'engorgement de ce dernier : si le récepteur a besoin d'un délai t pour exécuter le code de `from_physical_layer` à `to_network_layer`, l'émetteur doit émettre à une vitesse moyenne inférieure à une trame par temps t .

Questions :

1. Donner une solution simple au problème de contrôle de flux permettant d'asservir l'émetteur au récepteur.
2. Ecrire en C les deux procédures **sender2** et **receiver2** qui réalisent le protocole 2 dans le fichier `~/ASR2-Reseaux/DataLink/p2.c`.
3. Exécuter votre protocole, vérifier qu'il n'y a pas de blocage si les taux de perte et d'erreur de trame sont nuls.

6. Protocole 3 (canal avec erreurs)

Considérons maintenant le cas d'un canal de transmission produisant des erreurs. Les trames peuvent subir des erreurs de transmission ou être perdues. On suppose que lorsqu'une trame subit une erreur de transmission, cette erreur est détectée par le récepteur, par calcul d'un CRC.

Questions :

1. Peut-on perdre des trames de données ? Quelles en sont les conséquences si on utilise le protocole 2 ?

Le mécanisme d'accusé de réception (en anglais : `acknowledgement`) positif est l'envoi par le receveur d'un accusé de réception lorsqu'il reçoit une trame.

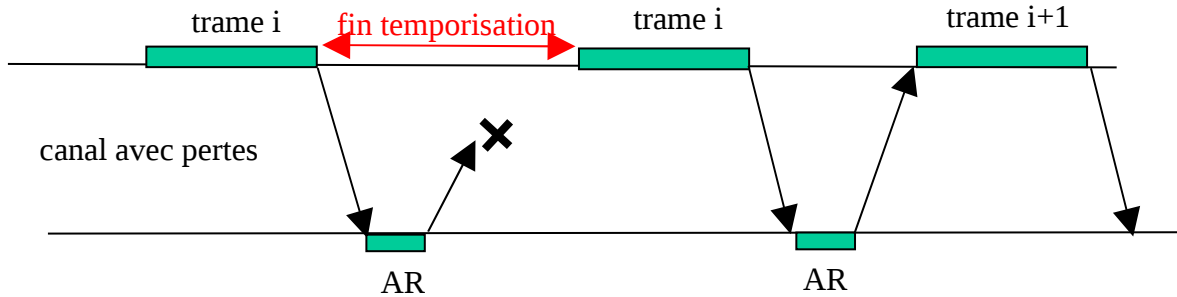
2. Un mécanisme d'accusé de réception vous semble-t-il suffisant pour garantir le bon fonctionnement du protocole en cas de perte de trame de données ?
3. Que faut-il ajouter au protocole ?
4. Peut-on perdre des accusés de réception ?
5. A-t-on besoin d'un mécanisme supplémentaire pour suffisant pour garantir le bon fonctionnement du protocole en cas de perte des accusés de réception ?
6. Que faire des trames erronées (données ou accusé de réception) ? Comment peut s'effectuer la reprise sur erreur ?

On s'oriente vers un mécanisme de reprise par accusés de réception positifs par le récepteur et une temporisation de la

rémission des trames de données par l'émetteur.

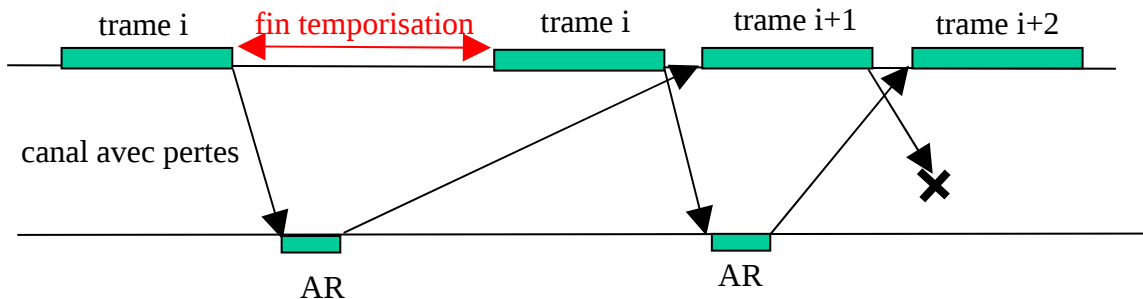
- Écrire les procédures `sender3` et `receiver3` qui réalise le protocole mettant en œuvre ces mécanismes dans le fichier `p3.c`
On utilisera une séquence maximum de 1 (numérotation des trames modulo 2), ainsi que les procédures de gestion des temporisateurs décrites page 3.

- La duplication de trames de données reçues par le receveur pose-t-elle un problème ? Expliquez.



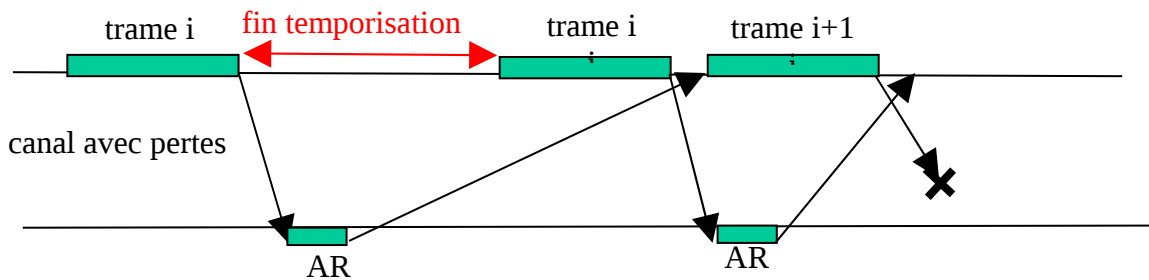
transfert de trames

- La numérotation des trames de données modulo 2 est-elle suffisante ? Expliquez.



transfert de trames

- La numérotation des accusés de réception assure le bon fonctionnement du protocole ? Complétez le schéma suivant.



transfert de trames

- Modifier le protocole en conséquence.

Le Round Trip Time du réseau physique est le temps nécessaire pour que la trame de données arrive au receveur, qu'il envoie l'accusé de réception, et que cette trame arrive à l'émetteur.

- Que se passe-t-il si le délai de retransmission des trames de données est deux fois plus court que le Round Trip Time du réseau physique ? Faire un schéma.
- Que se passe-t-il si le délai de retransmission des trames de données est deux fois plus long que le Round Trip Time du réseau physique ?
- Expliquer les trois types d'événements : `frame_arrival`, `cksum_err` et `timeout`.
- Lister les types de trames que votre protocole peut envoyer parmi la liste suivante : `data`, `ack` et `nak`
- Exécuter et vérifier l'exécution de votre protocole en observant les traces générées.
- Que manque-t-il à votre protocole pour simuler entièrement la couche liaison de données ?