Eléments de solution sur le TD :
Liaison de données

1 Gestion d'une liaison de données

Dans les procédures qui suivent on suppose que l'utilisateur de la couche liaison utilise la procédure d'émission chaque fois qu'il a envie d'émettre des données. En réception, il appelle la procédure de réception chaque fois qu'il souhaite lire les données provenant du réseau (Ce mode de fonctionnement s'appelle le mode par scrutation). On aurait pu supposer un autre mode de fonctionnement dans lequel l'appel de la procédure de réception n'est pas à l'initiative de l'utilisateur mais de la couche physique (Ce mode de fonctionnement s'appelle le mode par interruption).

1.1 Etape 1

Les fonctions à mettre en œuvre sont la synchronisation et le remplissage inter-trame, la délimitation, la transparence (nécessaire à cause de la délimitation). La structure la plus simple va consister à délimiter l'information utile par 2 séquences particulières appelées FLAG ou Fanion. On peut prendre la même séquence pour délimiter la fin et le début d'une trame. On pourra alors n'utiliser qu'un fanion qui pourra être à la fois la fin de la trame précédente et le début de la trame suivante. D'autre part, pour le cas où le niveau physique désire une occupation permanente du support, on fera du bourrage par émission continue de fanions. Par exemple on aura la transmission suivante :

.... FLAG Info1 FLAG FLAG FLAG Info2 FLAG Info3 FLAG ...

La procédure d'émission et de réception peuvent alors s'écrire :

```
E1(Suite de bits: buffer)
debut
        Envoyer(Flag);
        Envoyer(Insere_zero(buffer));
        Envoyer(Flag);
fin

R1(Suite de bits: buffer)
debut
        buffer = Supprime_zero(Lecture_jusqu_a_flag());
fin
```

A noter que les procdures employées ici ne suppose pas l'émission de traffic de synchronisation entre les émissions de trames.

1.2 Et les erreurs alors !

Il faut maintenant gérer d'une part la détection d'erreur et d'autre part la reprise sur erreur. La détection d'erreur va être réalisée en rajoutant une information redondante sur chaque trame transmise avec l' algorithme CRC (A et B doivent utiliser le même). Pour la reprise, l'émetteur devra réémettre une trame si celle ci n'est pas acquittée au bout d'un certain temps. Pour le récepteur, il n'acquittera pas les trames non reçues ou mal reçues. A partir du moment où l'on retransmet des trames, il y a risque de duplication d'une trame de données. Cela va arriver en particulier lors de la perte d'une trame d'acquittement. A aura envoyé 2 fois

les mêmes données qui sont reçus correctement par B, mais B ne doit les délivrer qu'une fois à son utilisateur. Il faut donc que B puisse identifier si il reçoit une nouvelle trame ou la réémission de la trame reçue précédemment. Le seul moyen simple est de numéroter les trames de données émises afin que B puisse décider quand il y a duplication.
La structure de la trame est maintenant :

... Type Compteur Données Redondance ...

Type et Redondance sont obligatoires, Compteur et Données ne sont présents que dans les trames de données. On considére de plus, un temporisateur (timer) t dont l'accés est partagé entre la procédure d'émission et la procédure de réception. Ce partage permet d'armer le temporisateur au moment de l'émission et de le désarmer lorsque l'acquittement correspondant est reçu. Par ailleurs on supposera le partage d'un booléen *acquitte* qui est validé lorsque en A un acquittement est reçu par la procédure R2. Les procédures démission et de reception deviennent alors :

```
Timer: t;
Booleen: acquitte;
Entier: compteur = 0;

E2(Suite de bits: buffer)
variables
        Suite de bits: type_env, code, trame;
debut
        acquitte = faux;
        tant que non(acquitte) faire
                type_env = "donnees";
                trame = type_env + compteur + buffer;
                code = CRC(trame);
                trame = trame + code;
                E1(trame);
                Armer(t);
                tant que non(Expire(t)) & non(acquitte) faire
                        Attendre();
                fin tant que
        fin tant que
        compteur = compteur + 1;
fin


Entier: compteur_att = 0;

R2(Suite de bits: buffer)
variables
        Entier: compteur_rec;
        Suite de bits: type_env, type_rec, code_rec, trame_rec;
        Suite de bits: code_env, trame_env, code_cal;
debut
        R1(trame_rec);
        code_rec = Extraire("code",trame_rec);
        trame_rec = Extraire("type_compteur_buffer",trame_rec);
        code_cal = CRC(trame_rec);
```

```
        si (code_rec==code_cal) alors
                type_rec = Extraire("type", trame_rec);
                si (type_rec == "donnees") alors
                        type_env = "ack";
                        trame_env = type_env;
                        code_env = CRC(trame_env);
                        trame_env = trame_env + code_env;
                        E1(trame_env);
                        compteur_rec = Extraire("compteur",trame_rec);
                        si (compteur_rec==compteur_att) alors
                                buffer = Extraire("buffer", trame_rec);
                                compteur_att = compteur_att + 1;
                        fin si
                fin si
                si (type_rec == "ack") alors
                        Desarmer(t);
                        acquitte = vrai;
                fin si
        fin si
fin
```

Chaque machine dispose de procédures E2 et R2 (similaires aux procédures E1 et R1) permettant de gérer la réception et l'émission des acquittements. Par ailleurs le symbole + représente la concaténation de chaines.

1.3 Contrôle de flux

Il peut y avoir saturation au niveau du récepteur. Pour la procédure R1 cela correspond à un rythme d'appel de R1 trop lent par rapport au rythme d'arrivée des trames. Les trames sont alors perdues ce que l'on désirerais éviter. Il faut donc que B puisse limiter le débit d'émission de A de telle sorte que celles-ci ne soient émises que lorsque la procédure de réception est appelée en B. On a principalement 2 stratégies, soit B est par défaut prêt à recevoir et il enverra une trame de contrôle particulière en cas de saturation (RNR en HDLC), soit B est potentiellement toujours saturable et alors A n'émettra une nouvelle trame qu'après autorisation de B. On se place dans ce deuxième cas. Chaque trame envoyée donne lieu à un acquittement et une trame d'information ne peut être envoyée qu'après réception de l'acquittement de la trame précédente. L'acquittement indique la bonne réception et le traitement d'une trame. Pour pouvoir gerer plusieurs types de trame, on peut maintenant structurer la trame sous la forme :

... Type Info ...

Où Type est codé 1 bit pour indiquer le type *Donnees* ou *Invitation* (à émettre) (Info est présent si Type=Données). On a alors :

```
Booleen: acquitte = vrai;

E3(Suite de bits: buffer)
variables
        Suite de bits: trame, type_env;
debut
        tant que non(acquitte) faire
                Attendre();
```

```
        fin tant que
        acquitte = faux;
        type_env = "donnees";
        trame = type_env + buffer;
        E1(trame);
fin
```

La variable *acquitte* est mise à jour par la procédure R3 de réception des acquittements en A. Sa valeur est initialisée à *vrai* de telle sorte que A puisse émettre un première trame. Par la suite une émission ne sera possible que si la trame précédente a été acquittée.

```
R3(Suite de bits: buffer)
variables
        Suite de bits: trame, type_rec, type_env;
debut
        R1(trame);
        type_rec = Extraire("type", trame);
        si (type_rec == "donnees") alors
                buffer = Extraire("buffer", trame);
                type_env = "invitation";
                trame = type_env;
                E1(trame);
        fin si
        si (type_rec == "invitation") alors
                acquitte = vrai;
        fin si
fin
```

1.4 Conclusion

A noter que l'on peut facilement combiner les différents services en combinant les différentes procédures. Par exemple si on utilise les procédures $R2$ et $E2$ à la place de $R1$ et $E1$ dans $R3$ et $E3$, l'appel à ces procédures permettra de rendre les services de détection et de reprise sur perte sans avoir à modifier le contenu des autres procédures. On voit ici l'interêt de séparer les différentes procédures et d'utiliser une interface similaire pour leur appel. C'est l'idée de base du modèle OSI. Cependant cette flexibilité a un coût: la duplication du champs *type* dans la trame dans notre exemple.