Le contrôle de flux et la récupération des erreurs

Introduction

- Protocoles apparaissant dans la couche liaison de données

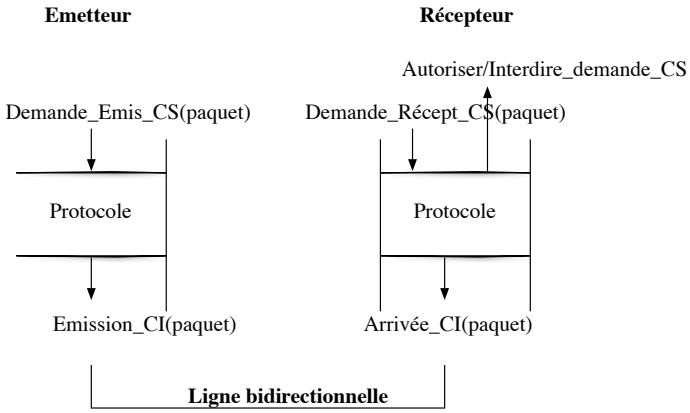
  ou dans la couche transport
- Différences suivant les cas dues au
  - Temps de transfert:
    - » Variable et important au niveau de la couche transport (temps de traversée des routeurs dépend de la charge du réseau)
    - » Fixe au niveau de la couche liaison (dépend seulement du temps de propagation et temps d'émission)
  - Désordonnancement possible des paquets au niveau transport
- On se place ici dans le cas de la couche liaison de données

Le contrôle de flux

- Problème : asservir le flot de production des données de l'émetteur à la capacité de les consommer du récepteur
- Cause : récepteur moins performant ou surchargé
- Le fait de mettre un tampon (buffer) en réception en attendant que les données soient traitées en réception ne résout pas le problème. Le tampon possède une taille bornée.
- On va estimer l'efficacité des protocoles proposés en terme de débit utile (ou effectif) au niveau de la couche supérieure par rapport au débit réel supposé dépendant seulement du temps d'émission

Environnement

- **On donne les primitives à l'interface avec la couche supérieure (CS) et inférieure (CI) du protocole que l'on veut définir**



- **Programmation événementiel**
  - Instruction particulière
  - Attendre un événement extérieur possible parmi une liste d'événements potentiels
    - » Suivant événement
      événement 1: instructions
      événement 2: instructions
      ...
      événement n: instructions

  Elle est exécutée à l'arrivée d'au moins un des événements
  Sinon l'instruction est "bloquante"

- **Le protocole utopique:**
  - On émet tout de suite sans aucun contrôle
  - Problème si le récepteur n'arrive pas à traiter les paquets à une vitesse suffisante : perte de paquets

Algorithmes

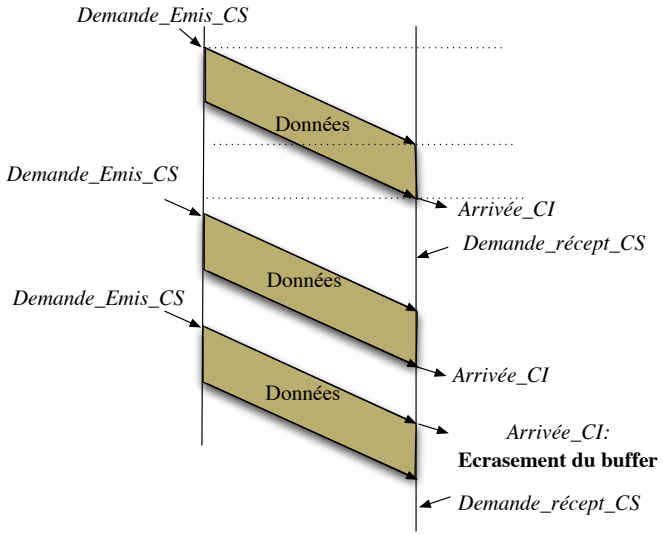
- ***Emetteur***

tant que vrai
  Suivant événement
    Si Demande_emis_CS (paquet);
      Emission_CI (paquet);

- ***Récepteur***

Interdire_demande_CS;
tant que vrai
  Suivant événement
    Si Demande_recept_CS (paquet);
      paquet= Buffer;
      Interdire_demande_CS;
    Si Arrivée_CI (paquet)
      Buffer=paquet;
      Autoriser_demande_CS;

Exemple

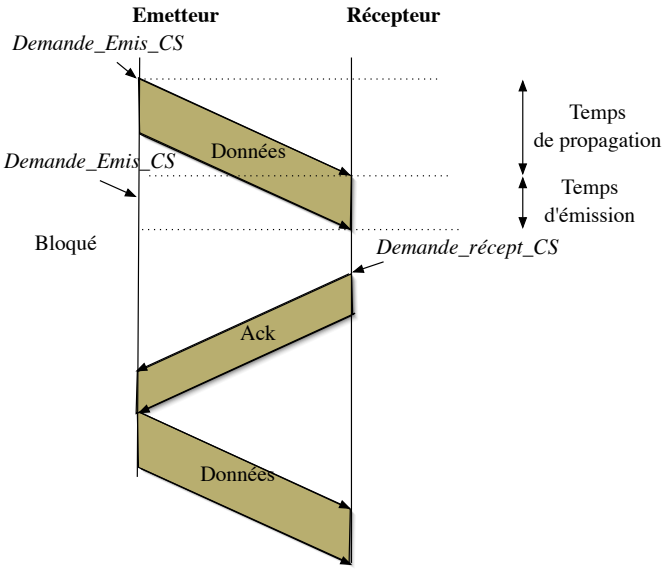


Le protocole "Envoyer et attendre"

- **Coté émetteur:**
  - Emission d'un paquet et attente d'un acquittement (paquet spécial) avant d'envoyer un autre paquet de donnée

- **Coté récepteur:**
  - Réception et Mémorisation d'un paquet de donnée
  - Emission d'un acquittement au moment où on passe le paquet à la couche supérieure (libération du buffer de réception)

Exemple



Débit applicatif du protocole "Envoyer et attendre"

- Notons
  - m: le temps de propagation entre les deux entités
  - n: le temps d'émission d'un paquet, on suppose ce temps constant pour tous les paquets (donnée et acquittement)
- **Débit : 1 paquet / 2 (m+n) s (au maximum)**
- **Très peu efficace si m est grand**
- **Exemple:**
  - Liaison satellite m= 250 ms,
  - Paquets de 10 kbits à 10 mégabits/s donc n= 1ms
  - On envoie un paquet toutes les 502 ms
  - Soit un débit de l'ordre de 20 kbits/s

Primitives supplémentaires

- **Pour l'émetteur:**
  - » **Autoriser/Interdire_demande_CS**
    - » **pour bloquer les demandes d'émission de la couche supérieure**
  - » **Arrivée_CI (paquet)**
    - » **pour les ACK**
- **Pour le récepteur**
  - » **Emission_CI (paquet)**
    - » **pour les émissions des ACK**

Algorithmes "Envoyer et attendre"

- **Emetteur**

```
Autoriser_demande_CS;
tant que vrai

   Suivant événement
        Si Demande_emis_CS (paquet);
                Emission_CI (paquet);
                Interdire_demande_CS;

        Si Arrivée_CI (paquet)
                si paquet = ACK
                        Autoriser_demande_CS;
```

Algorithmes “Envoyer et attendre”

- **Récepteur**

Interdire_demande_CS;

tant que vrai
  Suivant événement
    Si Demande_recept_CS (paquet);
      paquet= Buffer;
      Interdire_demande_CS;
      Emission_CI (ACK);

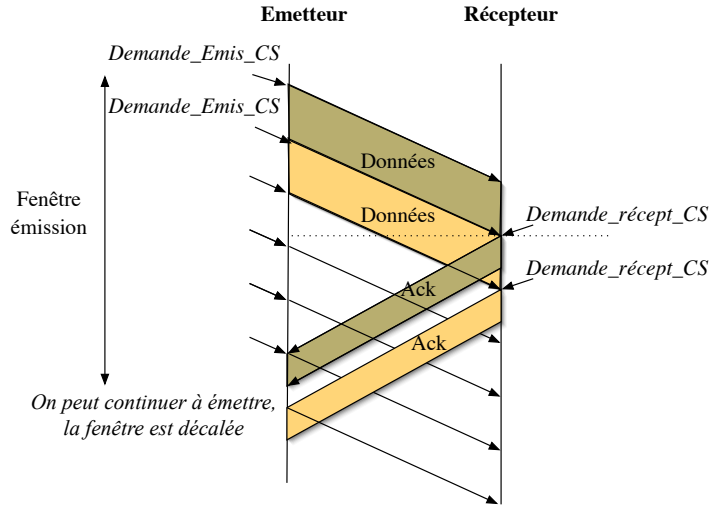
    Si Arrivée_CI (paquet)
      Buffer=paquet;
      Autoriser_demande_CS;

Protocole à fenêtre d’anticipation

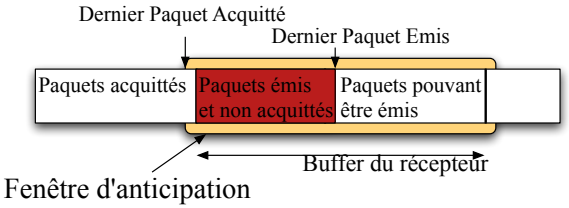
- Pour améliorer le débit effectif du protocole précédent, l'émetteur va se permettre d'envoyer un certain nombre de paquet avant de se bloquer en attendant les acquittements
- Un acquittement est toujours envoyé pour chaque message reçu
- Fenêtre d’émission: nombres de trames pouvant être émises sans acquittements
- Il faut que le récepteur puisse stocker les paquets de la fenêtre dans un buffer, en attendant que la couche supérieure les récupère

Exemple

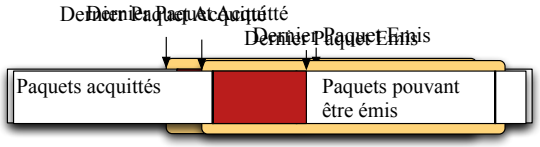
Cas où le récepteur suit le rythme, l’émetteur émet sans aucun blocage



Fenêtre d’anticipation côté émetteur

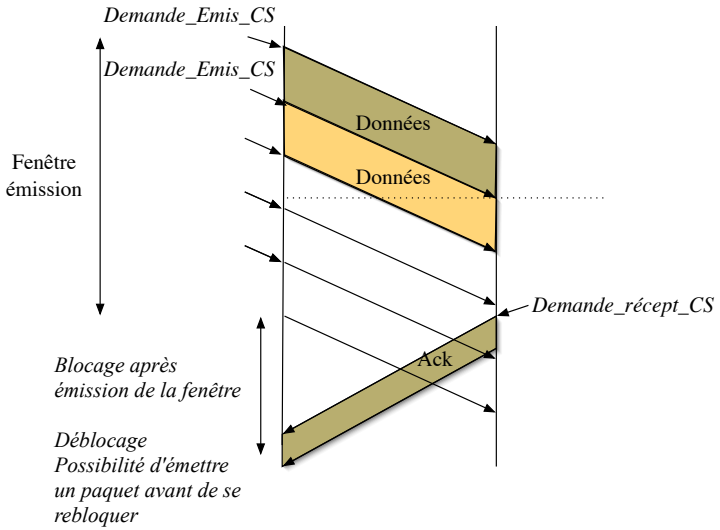


- A la réception d’un acquittement la fenêtre «glisse»:



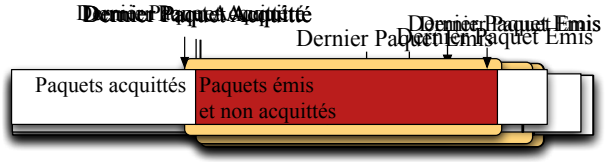
Exemple

Cas où le récepteur ne suit pas le rythme, l'émetteur émet la fenêtre puis se bloque



Fenêtre d'anticipation côté émetteur

- Cas de blocage: émission de tous les paquets de la fenêtre sans réception d'acquittement



Algorithmes à fenêtre d'anticipation

- **Emetteur**

```
Autoriser_demande_CS;
NbEmis=0;
tant que vrai

   Suivant évènement
        Si Demande_emis_CS (paquet);
                Emission_CI (paquet);
                NbEmis ++;
                Si NbEMis= TAILLE_FENETRE
                        Interdire_demande_CS;
        Si Arrivée_CI (paquet)
                si paquet = ACK
                        Autoriser_demande_CS;
                        NbEmis - -;
```


Algorithmes à fenêtre d'anticipation

- **Récepteur**

```
Interdire_demande_CS;
NoRecept=0;     /* Numéro du prochain buffer de réception */
NoTrans=0;      /* Numéro du prochain buffer à transmettre à la CS */
tant que vrai
   Suivant événement
        Si Demande_recept_CS (paquet);
                paquet= Buffer[NoTrans]; (NoTrans ++ ) mod TAILLE_FEN;
                si NoTrans=NoRecept
                        Interdire_demande_CS;
                Emission_CI (ACK);

        Si Arrivée_CI (paquet)
                 Buffer[NoRecept]=paquet; (NoRecept ++ ) mod TAILLE_FEN;
                Autoriser_demande_CS;
```

Efficacité du protocole à fenêtre d'anticipation

- Si on veut émettre en continu (cas où le récepteur suit le rythme) il faut que l'on est fini de recevoir l'ACQ du premier paquet de la fenêtre quand on a fini d'émettre le dernier paquet de la fenêtre
- Taille_fenetre * n = Temps d'émission des paquets de la fenêtre
- m+n = temps d'arrivée du premier paquet = temps de retour de l'ACK correspondant
- donc il faut que Taille-fenêtre * n ≥ 2* (m+n) pour obtenir le débit du protocole utopique
- Pour que le protocole fonctionne correctement (sans perte), il faut que le buffer de réception soit supérieur ou égal à la taille de la fenêtre d'émission
- Ça se complique si le délai de transfert est variable (cas de la couche transport): la taille de la fenêtre optimale est difficile à fixer

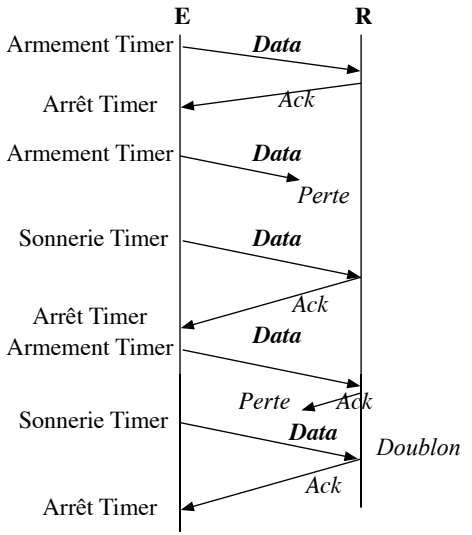
Efficacité du protocole à fenêtre d'anticipation

- Exemple de la liaison satellite:
  - Liaison satellite m= 250 ms,
  - Paquets de 10 kbits à 10 mégabits/s donc n= 1ms
  - On envoie un paquet toutes les 502 ms
- Quelle est la taille minimale du buffer de réception pour obtenir un débit maximal ?
- Temps d'aller retour : 502 ms

La récupération des erreurs

- **Problème:**
  - » provoquer la re-émission des trames perdues ou pour lesquelles on a détecté une erreur à l'arrivée
  - » On ne s'occupe pas ici du contrôle de flux mais on peut mixer facilement les deux problèmes qui sont résolus par des techniques semblables (voir plus loin)
- **Principe**
  - **Emetteur :**
    - Emission d'un paquet de donnée
    - Attente d'un paquet d'acquittement de la part du récepteur ("j'ai bien reçu le paquet")
  - **Récepteur:**
    - A la réception d'un paquet de donnée, émission d'un paquet d'acquittement

Exemple

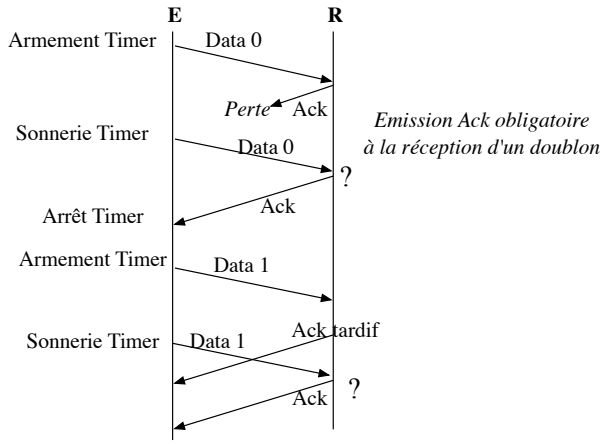


Protocole du bit alterné

- **Activation d'un timer après émission des données, si ce timer sonne avant la réception de l'acquittement , on re-émet**
- **Deux cas à problème:**
  - Un acquittement se perd, le paquet est re-émis
  - Le timer sonne trop tôt, le paquet est re-émis
- **Dans les deux cas le récepteur reçoit deux fois le même paquet (doublon)**
  - Il faut donc qu'il soit capable de différencier les paquets
  - On numérote les paquets (0 ou 1) de données

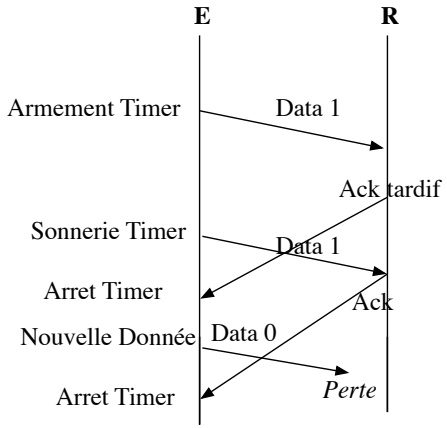
A la réception d'un doublon ?

- **L'ACK précédent s'est perdu ou est arrivé après expiration du timer: ré-émission de l'ACK obligatoire**



Numérotation des acquittements ?

- **Obligatoire !**



Algorithme du bit alterné

- **Emetteur**

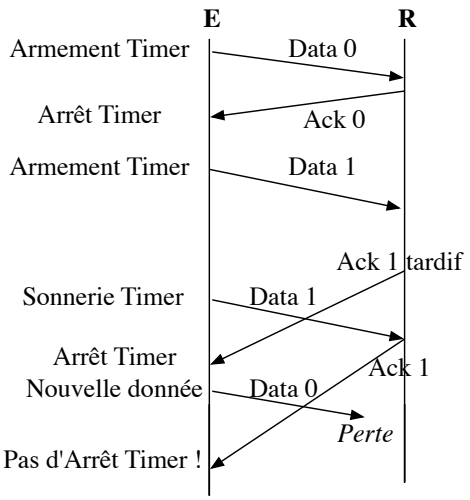
```
Autoriser_demande_CS; NoEmis=0;
tant que vrai
   Suivant événement
        Si Demande_emis_CS (paquet);
                Buffer = [NoEmis, paquet];
                 Emission_CI (Buffer);
                (NoEmis ++) mod 2 ;
                Interdire_demande_CS;
                Armer_Timer;
        Si Arrivée_CI (paquet)
                si paquet = ACK et NoACK=(NoEmis + 1)  mod 2
                        Autoriser_demande_CS;
                        Arrêter_timer;
        Si Expiration Timer
                Emission_CI (Buffer); Armer_Timer;
```

Algorithme du bit alterné

- **Récepteur**

```
Interdire_demande_CS; NoAttendu=0;      /* Numéro de la  donnée attendue */
tant que vrai
   Suivant événement
        Si Demande_recept_CS (paquet);
                paquet= Buffer;
                Interdire_demande_CS;

        Si Arrivée_CI (paquet)
                Si Entete(paquet)=NoAttendu
                        Buffer=donnée(paquet);
                        Autoriser_demande_CS;
                        Emission_CI (ACK avec NoAttendu);
                        (NoAttendu ++) mod 2;

                Sinon /*Doublon*/
                        Emission_CI (ACK avec ( NoAttendu+1) mod 2);
```


Exemple Bit Alterné



Contrôle de flux et récupération des erreurs en même temps

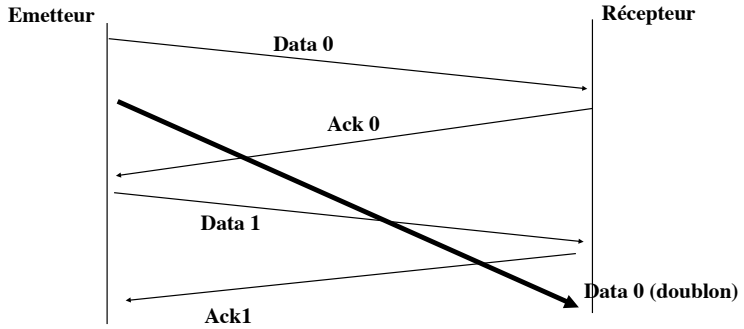
- **Possibilité d'une double utilisation des acquittements pour le contrôle de flux et la récupération des erreurs :**
  - On envoie l'ACK seulement au moment de la libération du buffer
  - Si récepteur très lent (ou bloqué) il y a des re-émissions de paquets par l'émetteur dues au contrôle de flux et non à la récupération d'erreur, ce qui ne sert à rien et charge le réseau inutilement
  - Donc intéressant si timer assez grand pour que l'on soit rarement en position de re-émission due au contrôle de flux
- **Il vaut mieux utiliser 2 types d'ACK pour effectuer les deux contrôles:**
  - Emission d'un ACK_ERREUR à la réception d'un paquet
  - Arrêt du timer de re-émission à la réception des ACK_ERREUR
  - Emission d'un ACK_FLUX à la lecture par la couche supérieure côté récepteur
  - Autorisation d'émission à la couche Supérieure à la réception d'un ACK_FLUX

Réglage du timer de re-émission

- **Liaison de donnée:**
  - Temps de transfert constant: délai de propagation + délai d'émission
  - Durée du timer légèrement supérieur au temps de transfert donnée + temps transfert de l'ACK
- **Transport:**
  - Latence très variable
  - Dépend de la charge du réseau
  - Le temps de traversée des routeurs varie en fonction de la charge
  - Solution: durée de timer calculé dynamiquement en fonction du temps d'aller-retour courant (voir protocole TCP)

Mise en défaut du protocole du bit alterné

- **En cas de déséquencement des paquets**

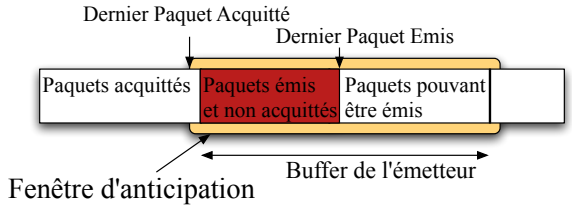


Plage de numéro de séquence ≥ Max-double + 2

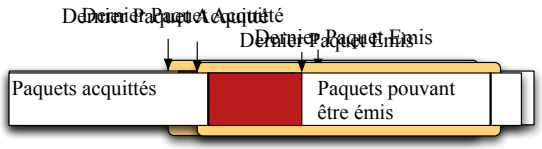
Fenêtre d'anticipation pour la récupération des erreurs

- Un numéro de séquence est associé aux données et acquittements.
- Un timer est associé à chaque trame émise pour une re-émission éventuelle
- Un paquet d'acquittement est envoyé pour chaque paquet de donnée
- Comme pour le contrôle de flux on se permet d'émettre un certain nombre de paquet avant la réception d'acquittement afin d'augmenter l'efficacité du protocole
- Un tampon en émission est nécessaire pour le stockage des paquets en attente d'une re-émission à la sonnerie d'un timer (la taille du tampon en émission est égale à la taille de la fenêtre)
- On ne s'occupe plus par la suite du contrôle de flux mais on pourrait "mélanger" les deux protocoles comme précédemment pour le bit alterné et le protocole "envoyer et attendre"

Fenêtre d'anticipation pour la récupération des erreurs



- A la réception d'un acquittement la fenêtre «glisse»:

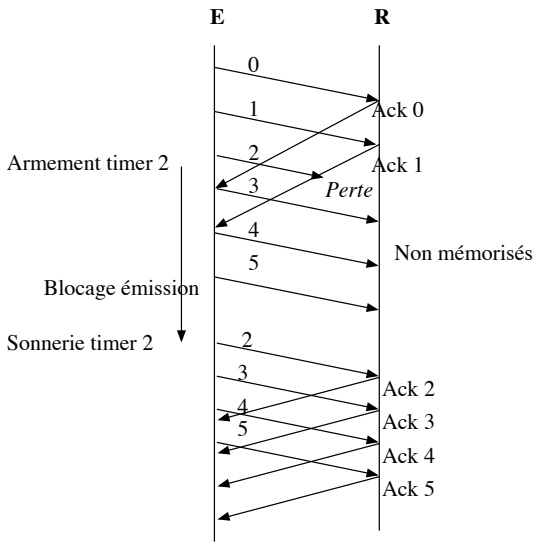


Première stratégie: le Go-Back-N

- Le plus simple possible
- Quand il y a une erreur le récepteur refuse toutes les paquets suivants
- Un timer par paquet
- L'émetteur re-émet tout depuis un paquet non acquitté
- Un acquittement indique la réception de tous les paquets jusqu'au numéro qu'il porte (acquittements "cumulatifs")

Exemple Stratégie Go-Back-N

- **Taille de la fenêtre = 4**
- **Re-émission inutile des paquets 3, 4, 5 et 6**



Algorithmes Go-Back-N: récepteur

- Il n'y a pas de contrôle de flux (le récepteur suit le rythme) un seul buffer peut suffire

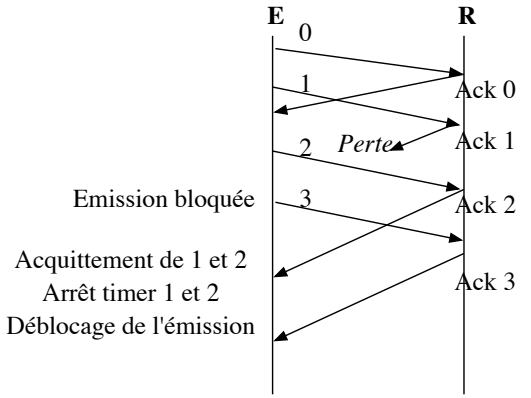
```
Interdire_demande_CS;
NoAttendu=0;
tant que vrai
   Suivant événement
        Si Demande_recept_CS (paquet);
                paquet= Buffer;
                Interdire_demande_CS;

        Si Arrivée_CI ([No, paquet])
                si No == NoAttendu alors
                 Buffer=paquet;
                    Emission_CI ([NoAttendu, Ack]);
                  (NoAttendu ++ ) mod TAILLE_FENETRE;
                    Autoriser_demande_CS;
```


Algorithmes Go-Back-N: émetteur

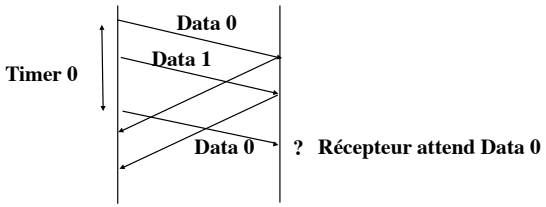
```
Autoriser_demande_CS; NoEmis=0; /*Prochain No de paquet à émettre*/
NoAcquitAtt=0; /*No d'Acquittement Attendu*/
tant que vrai
   Suivant événement
        Si Demande_emis_CS (paquet);
                Emission_CI ([NoEmis, paquet]);
                Mémorisation dans buffer de [NoEmis, paquet];
                ArmerTimer (NoEmis); NoEmis ++;
                Si (NoEMis-NoAcquit) = TAILLE_FENETRE alors Interdire_demande_CS;
        Si Arrivée_CI (paquet) /* Acq portant le numéro NoPaquetRecu */
                si NopaquetRecu ≥ NoAcquitAtt alors /*Acq cumulatif*/
                        Autoriser_demande_CS;
                            Répéter pour No= NoAcquitAtt à NoPaquetRecu
                                    Arrêter Timer (No);
                            NoAcquitAtt = NoPaquetRecu + 1;
         Si Sonnerie Timer(NoTimer)
                Répéter pour No= NoTimer à NoEmis-1
                        Emission_CI ([No, paquet]);  /*pris dans le buffer d'émission */
                        Armer Timer (No);
```


Intérêt des Acquittements cumulatifs

- **Exemple:**
- **Taille de la fenêtre = 3**
- **Ack 2 acquitte 1 et 2**



- **Relation Taille de la fenêtre/No de Séquence pour que ce protocole fonctionne correctement:**
  - Taille_fenetre + 1 ≤ Plage_Numero_sequence
  - Sinon le récepteur ne peut pas différencier un doublon d'une nouvelle donnée
  - Exemple : Taille_fenetre= Plage_Numero_sequence =2

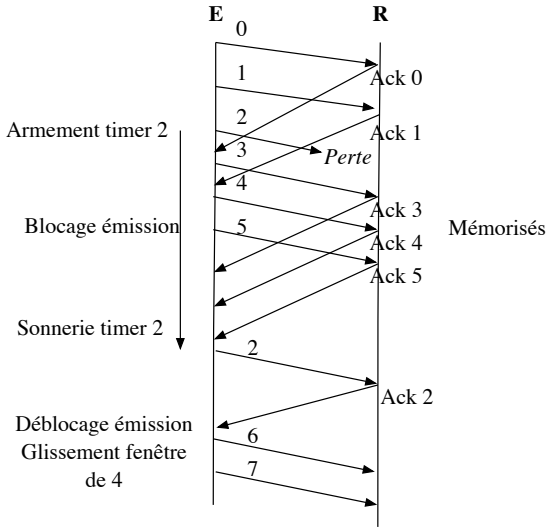


Solution à re-émission sélective

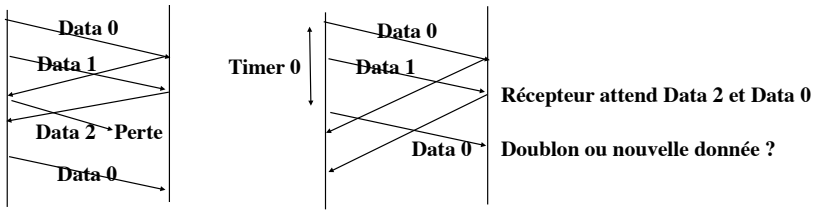
- But : éviter les re-émissions inutiles du Go-Back-N
- Un acquittement par paquet
- Un acquittement n'acquitte qu'un seul paquet à la fois
- Un timer par paquet
- Le récepteur mémorise les trames qui ne sont pas "en séquence"
- L'émetteur "avance" sa fenêtre d'émission jusqu'au dernier paquet acquitté en séquence

Exemple re-émission sélective

- **Taille fenêtre= 4**



- **Relation Taille de la fenêtre/No de Séquence pour que ce protocole fonctionne correctement:**
  - **Exemple : Taille_fenetre= 2 et Plage_Numero_sequence =3**



  - **2 * Taille_fenetre ≤ Plage_Numero_sequence**

    Sinon le récepteur ne peut pas différencier un doublon d'une nouvelle donnée

Optimisations

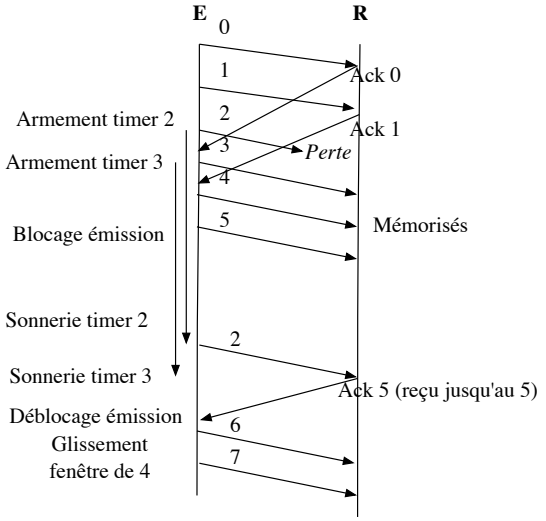
- **Dans la re-émission sélective la perte d'un acquittement est équivalent à une perte de donnée, il a pour conséquence la re-émission du paquet de données**
  - Idée: garder la notion d'acquittement "cumulatif" du Go-Back-N
  - Le récepteur mémorise mais n'acquitte pas les paquets suivants un paquet perdu. On peut alors re-acquitter le dernier paquet reçu (*)
  - Il faut alors limiter les re-émissions successives (Voir exemple transparent suivant)
- **Pour éviter l'attente des timers de reémissions**
  - A la réception d'un ack dupliqué (*) : re-émission du paquet supposé perdu
  - Emission d'acquittements négatifs à la réception de données qui ne sont pas en séquence
    - » Valable si les paquets ne sont pas trop souvent "déséquencés" sur le réseau (voir TCP: acquittements sélectifs)

Exemple re-émission sélective avec acquittements cumulatifs

- **A la sonnerie 3 pour limiter les re-émissions inutiles le paquet 3 n'est pas re-émis car 2 vient de l'être**

Il ne sera re-émis qu'à la réception de l'ack du 2

- **Que se passe-t-il si le 3 s'est perdu aussi ?**

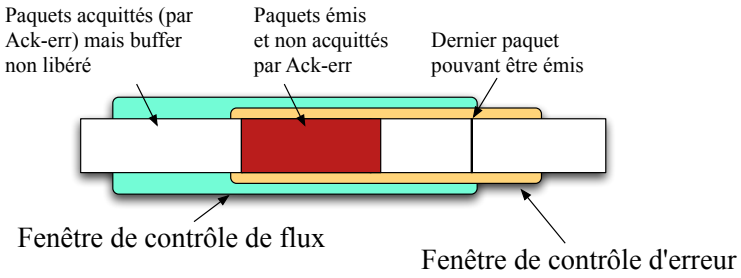


Contrôle de flux et récupération d'erreur à la fois

- On peut mélanger les deux contrôles avec fenêtre d'anticipation
- Deux types d'ACK :
  - Emission d'un ACK_ERREUR à la réception d'un paquet
  - Emission d'un ACK_FLUX à la lecture par la couche supérieure côté récepteur
- Des buffers en réception pour le contrôle de flux
- Des buffers en émission pour le contrôle d'erreur
- Une fenêtre à anticipation pour chacun des deux contrôles
- L'émetteur se bloque suivant la fenêtre la moins "avancée"

Les deux fenêtres du contrôle de flux et de la récupération d'erreur à la fois

- Les deux fenêtres sont indépendantes et avancent suivant chacun des acquittements



- A cet instant l'émetteur est bloqué par la fenêtre du contrôle de flux

Tailles des fenêtres d'anticipation

- Taille de la fenêtre optimale identique pour les deux contrôles dépendant de
  - Temps émission paquet + Temps émission Ack + 2 * temps de propagation
- Donc la taille des buffers doit être identique en émission et réception si le récepteur suit le rythme
- Si le récepteur est lent de façon sporadique on a intérêt à avoir: Buffer réception ≥ Buffer d'émission

Contrôle de flux et récupération d'erreur au niveau transport

- **Temps de transfert très variable**

➡ **timer de re-émission optimal variable**

- Réglage dynamique du timer en fonction des temps d'aller-retour précédents

➡ **taille de la fenêtre optimale très variable**

- On peut essayer de la borner
- On peut avoir à gérer un grand nombre de connexions de transport et donc avoir un potentiel en terme de "buffer" très variable
- On laisse au programmeur le choix de la taille des buffers d'émission et de réception

Contrôle de flux et récupération d'erreur au niveau transport

Le mécanisme de crédit

Pour éviter l'envoi des "ack_flux":

– Chaque paquet d'acquittement (d'erreur) contient un champ crédit qui indique à l'expéditeur le nombre de paquets qu'il peut envoyer.

– par exemple l'acquittement : **(ack = i, crédit = 5)**

indique à l'émetteur qu'il a bien reçu les paquets jusqu'au numéro **i** et qu'il peut encore en recevoir **5** (après celui là)

En d'autre terme le récepteur indique la place libre dans son buffer de réception

–Le protocole TCP de la couche transport utilise cette méthode