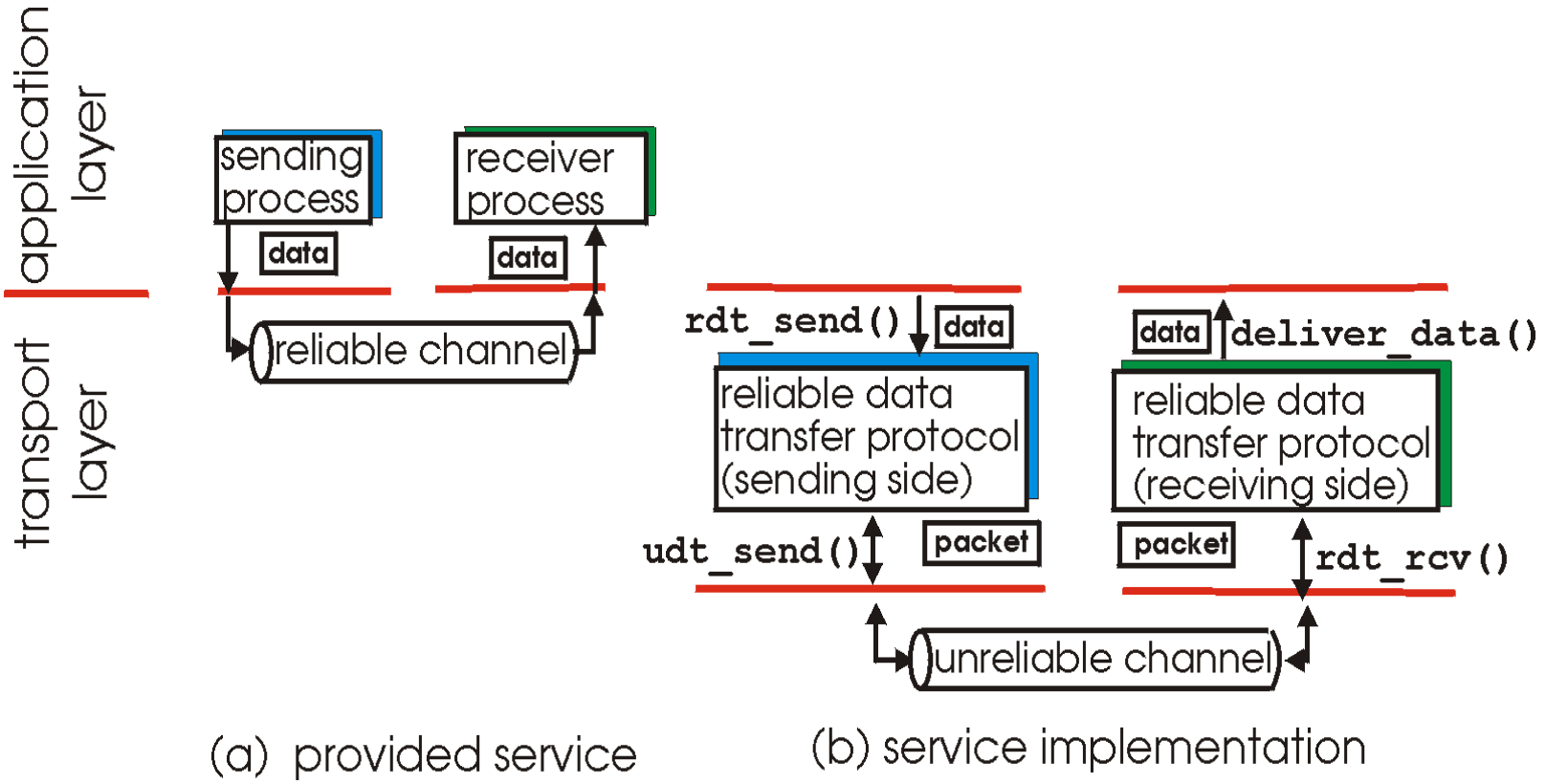
INSTITUT SUPERIEUR INFORMATIQUE ISI
المعهد العالي للإعلامية

الجمهورية التونسية
وزارة التعليم العالي و البحث العلمي و التكنولوجيا
جامعة تونس المنار
المعهد العالي للإعلامية

Chapitre 4

Etude de la fiabilité des protocoles de transport

Principes de transfert de données fiable

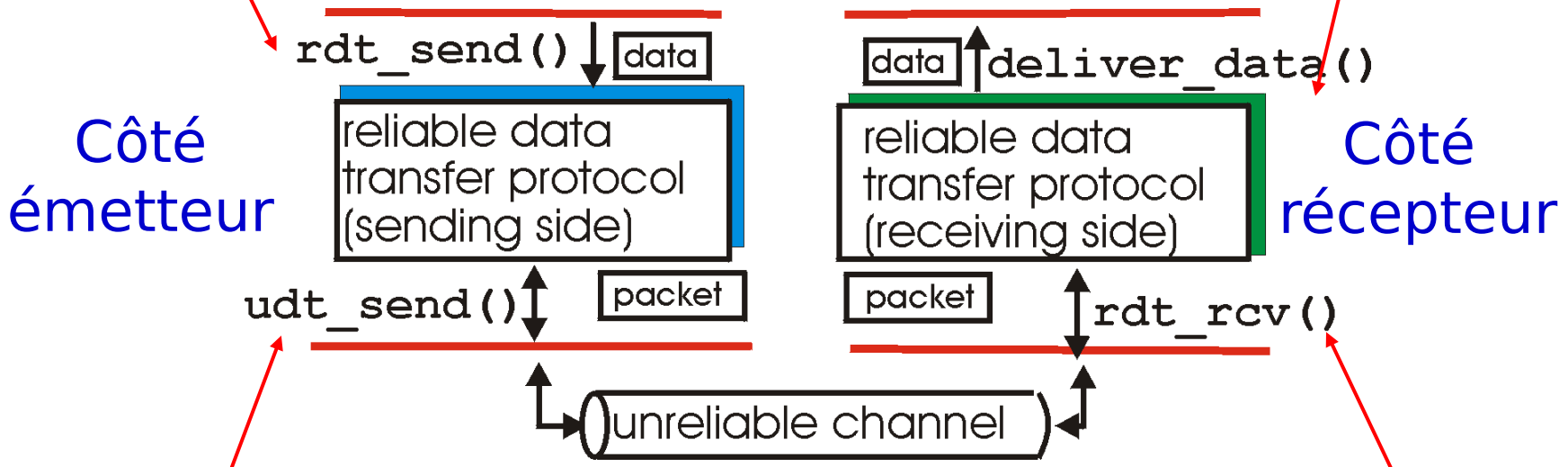


- Les caractèristiques d'un canal non fiable détermine la complexité du potocole de transfert de données fiable" reliable data transfer protocol (rdt)"

transfert de données fiable

`rdt_send()`: suite à un appel par dessus, (e.g., application.), remet les données à expédier à la couche supérieur du pôle destinataire

`deliver_data()`: appelé par **`rdt`** pour délivrer les données à la couche supé



`udt_send()`: appelé par rdt,
pour transférer le paquet à travers un canal non fiable

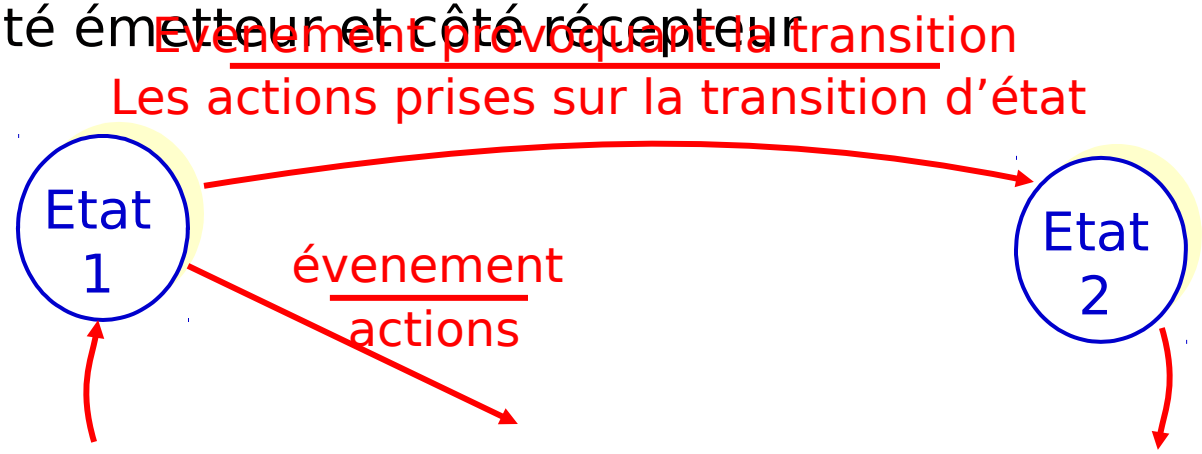
`rdt_rcv()`: appelé quand un paquet arrive du côté destinataire

transfert de données fiable

Nous allons

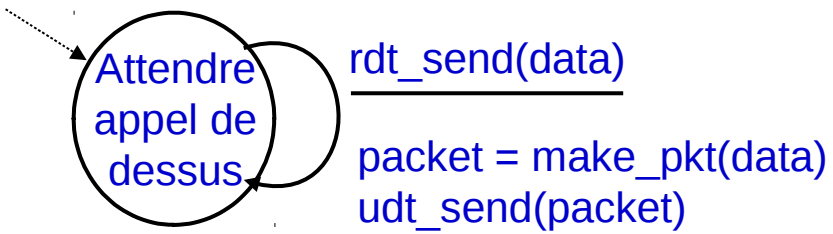
- étudier avec une démarche incrémetale la modélisation des protocoles de transfert de données fiables (rdt)
- considérer uniquement un transfert unidirectionnel
  - mais les infos de contrôle vont suivre les deux directions
- utiliser des machines à états finis (MEF) pour modéliser rdt côté émetteur et côté récepteur

Etat: Quand sur cet "Etat", l'Etat suivant est determiné uniquement par l'événement suivant

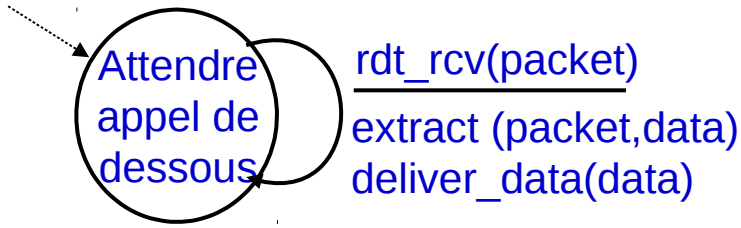


Rdt1.0: protocole de transfert fiable à travers un canal totalement fiable

- à travers un canal totalement fiable
  - pas d'erreurs sur les bits
  - pas de perte de paquets
- MEF séparées pour l'émetteur et pour le récepteur:



Emetteur

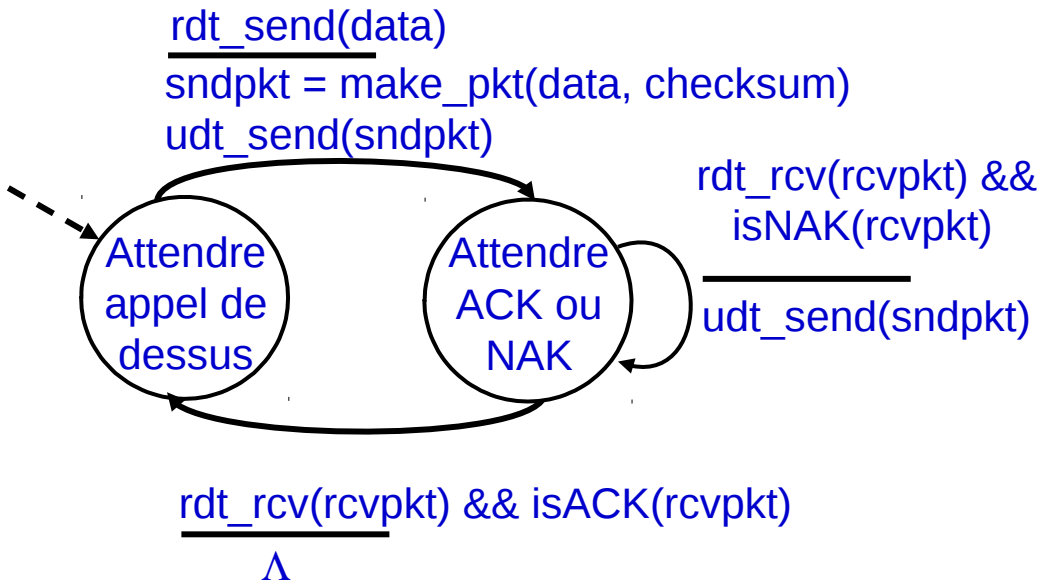


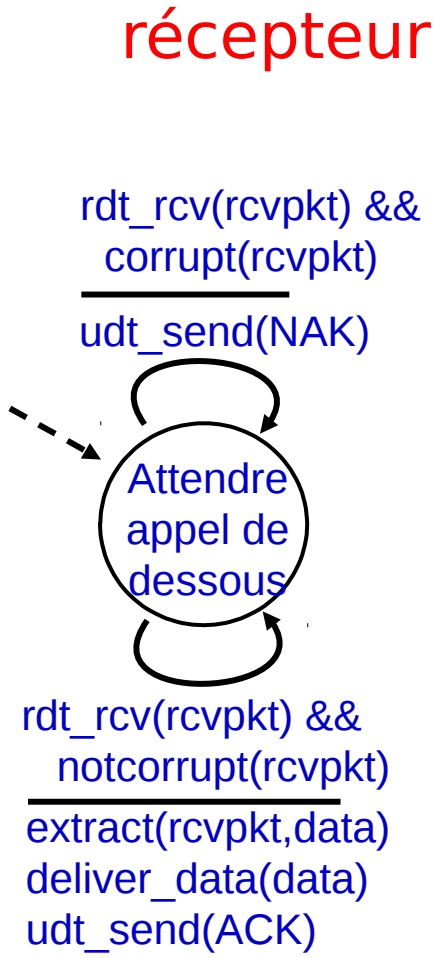
Récepteur

Rdt2.0: canal avec erreur sur les bits

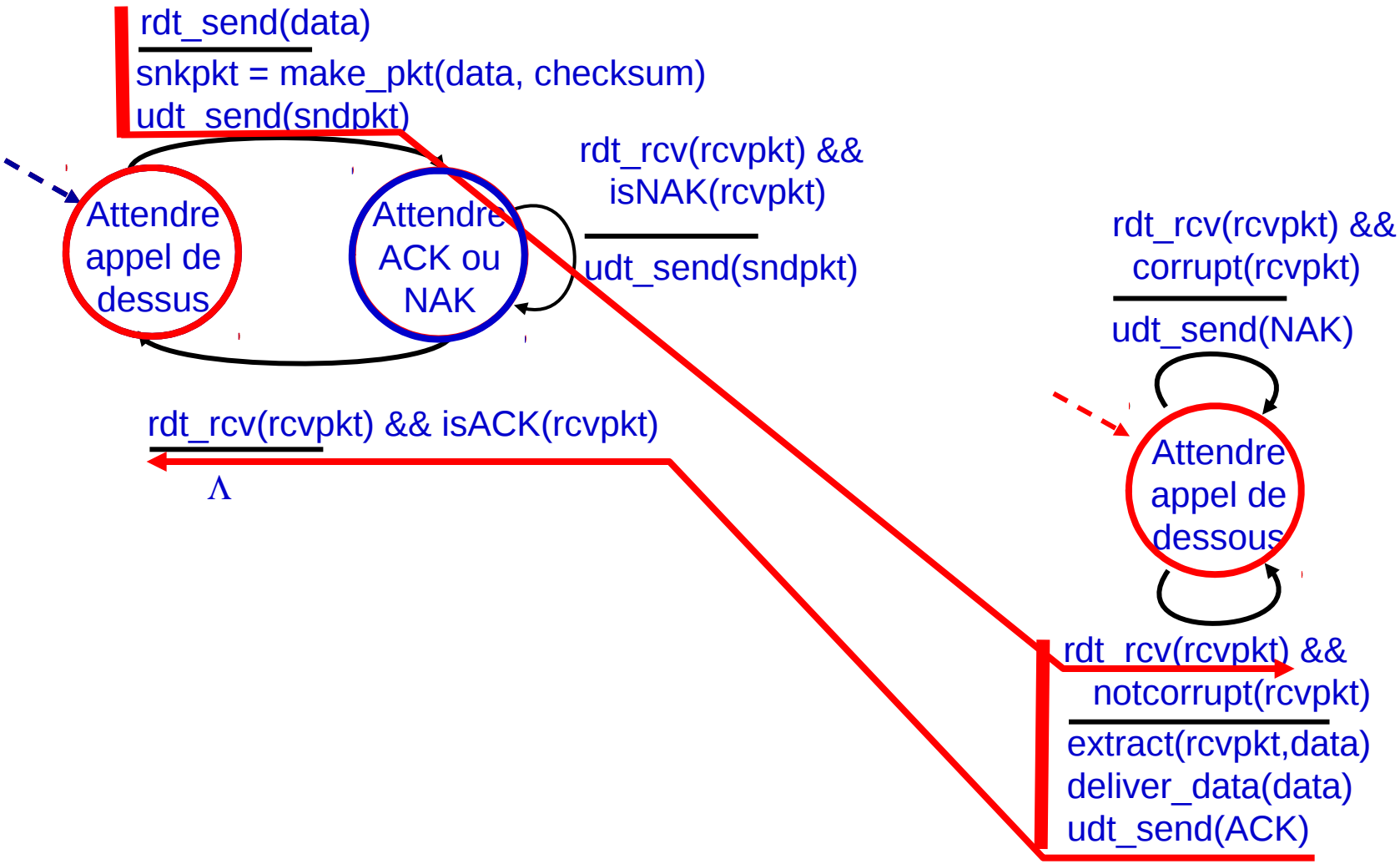
- La canal peut altérer des bits sur le paquet
  - checksum pour détecter des erreurs sur le bits
- *Question*: Comment recouvrir ce type d'erreur:
  - *acknowledgements (ACKs):* Le récepeur dit explicitemet à l'émetteur que le paquet arrive sans erreur
  - *negative acknowledgements (NAKs):* Le récepeur dit explicitemet à l'émetteur que le paquet arrive avec erreur
  - L'emetteur retransmet le paquet quand il reçoit un NAK
- nouveaux mécanismes dans **`rdt2.0`** (par rapport à **`rdt1.0`**):
  - détection d'erreur
  - retour du récepteur: msgs de contrôle (ACK,NAK) rcvr->sender

rdt2.0: specification de la MEF

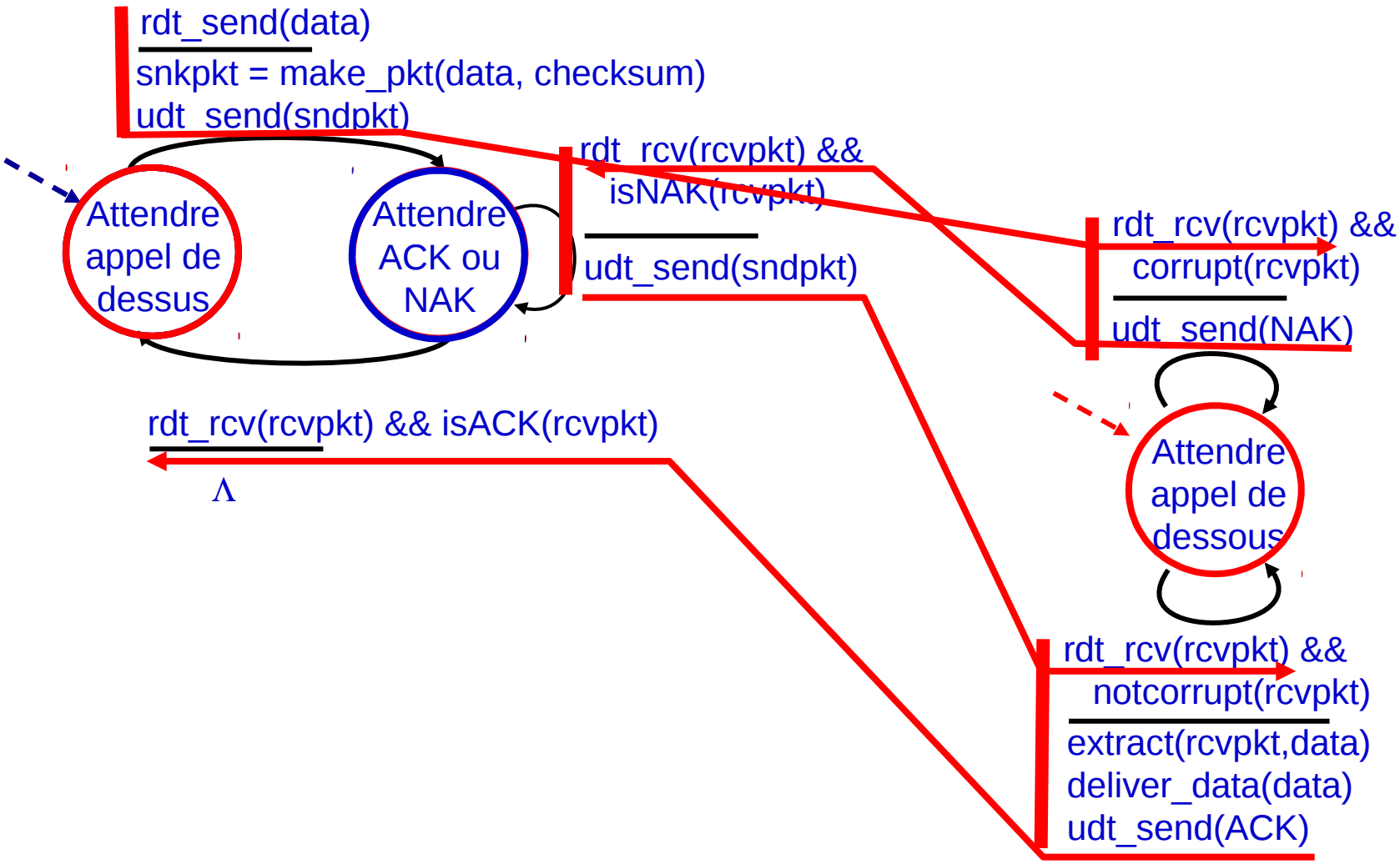




rdt2.0: opérations sans erreurs



rdt2.0: scénario avec erreur



rdt2.0 a une faille fatale!

Qu'arrive t-il si les ACK/NAK sont corrompus?

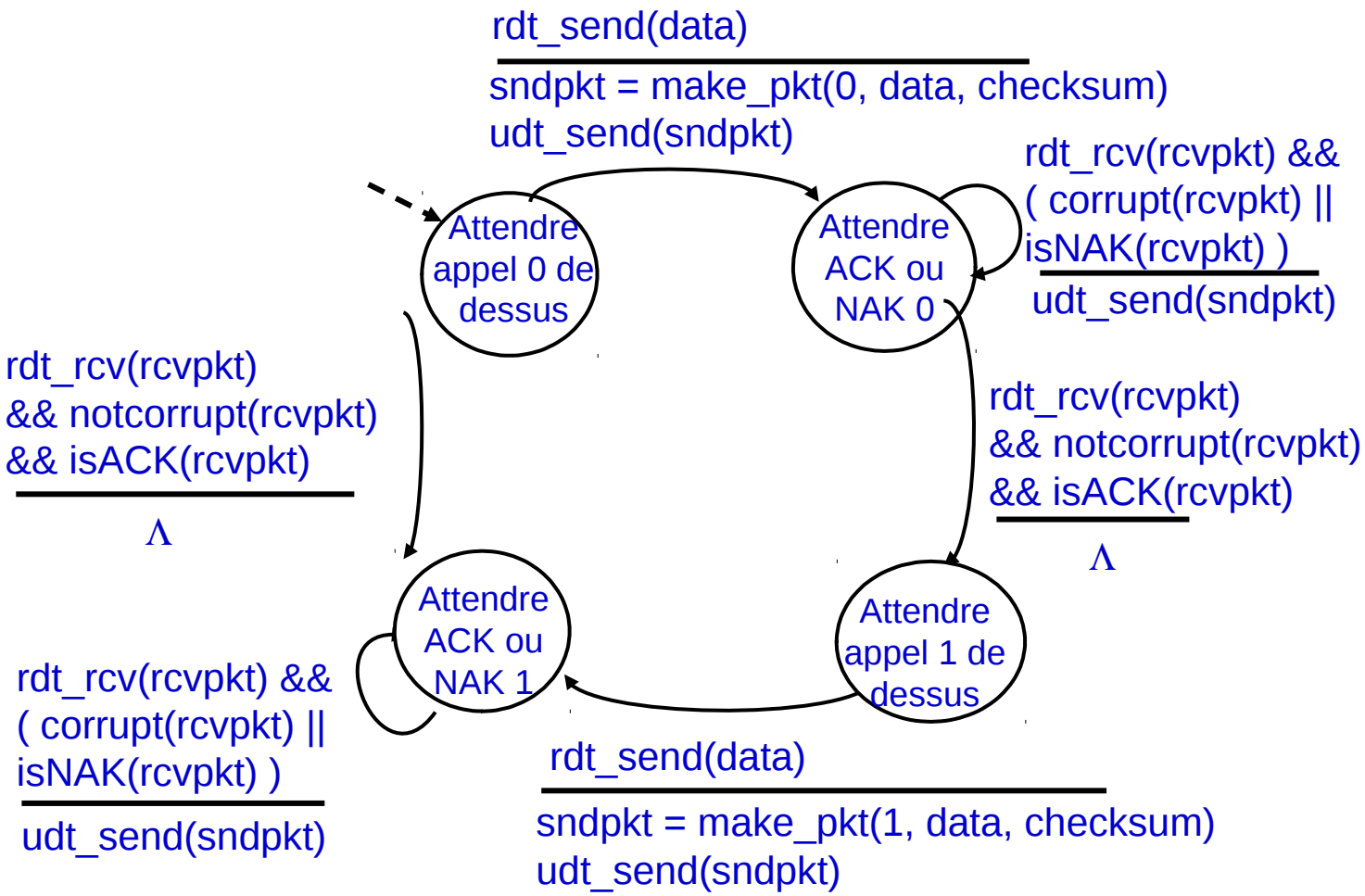
- L'émetteur ne peut pas savoir si le récepteur a bien reçu le paquet ou non!
- Il ne peut pas juste retransmettre le paquet : possiblité de duplication

Traitement de la dupplication:

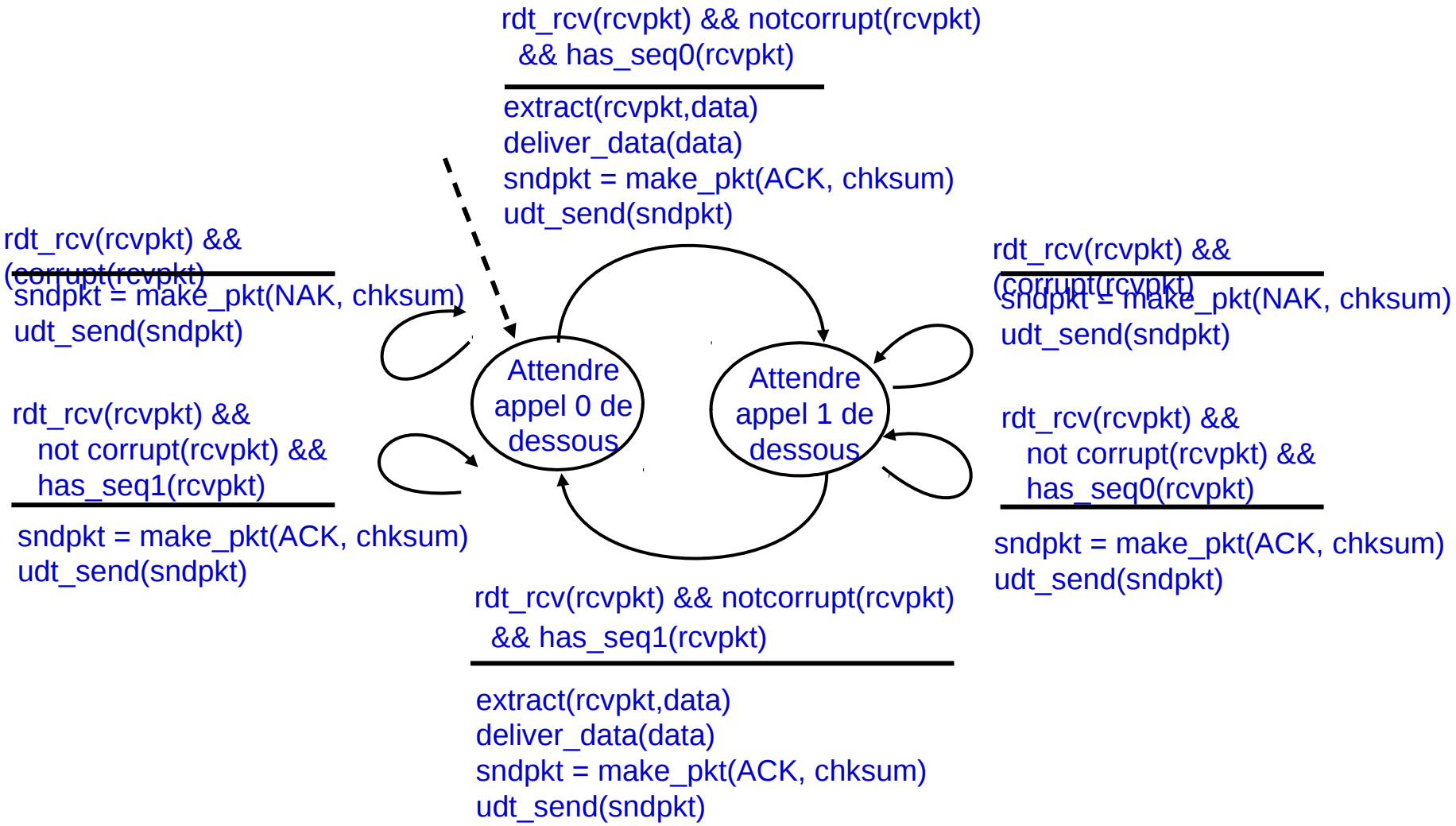
- L'émetteur ajoute un *numéro de séquence* pour chaque paquet
- L'émetteur retransmet le paquet courant si les ACK/NAK sont confus
- Le récepteur élimine (ne délivre pas à la couche de dessus) le paquet dupliqué

stop and wait
L'émetteur envoie un paquet, puis attend une réponse du récepteur

rdt2.1: Emtteur traite les ACK/NAKs confus



rdt2.1: Récepteur traite les ACK/NAKs confus



rdt2.1: discussion

Emetteur:

- num de seq ajouté au paquet
- deux nums (0,1) sont suffisant. pourquoi?
- Doit vérifier si ACK/NAK est corrompu
- deux états de transition
  - L'état doit "mémoriser" si le paquet courant a le num de seq 0 ou 1

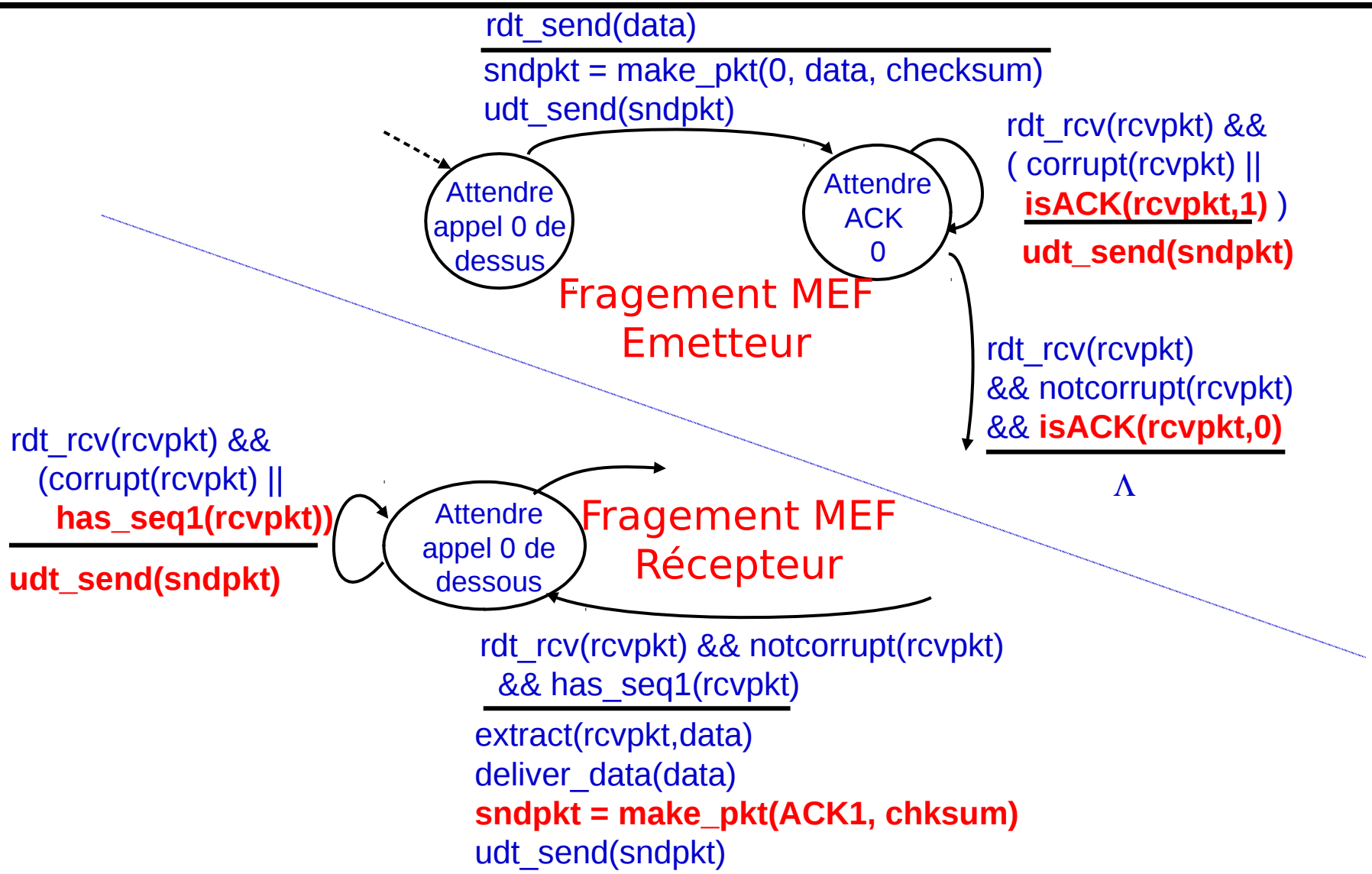
Récepteur:

- doit vérifié si la paquet reçu est dupliqué
  - L'état indique est ce que le num de séq du paquet 0 ou le 1 est expecté
- note: Le récepteur ne peut pas savoir si l'ACK/NAK précédent est bien reçu ou non par l'émetteur

rdt2.2: protcole sans NAK

- Même fonctionnalité que rdt2.1, utilisant uniquement des ACKs
- Au lieu d'un NAK, le récepteur envoie un ACK pour le dernier paquet reçu correctement
  - Le récepteur doit inclure explicitement le num de séq du paquet à aquitter
- un ACK dupliqué pour l'émetteur est équivalent à un NAK: *retrasmission du paquet courant*

rdt2.2: Fragments de la MEF Emetteur/Récepteur



rdt3.0: canal avec erreurs et pertes

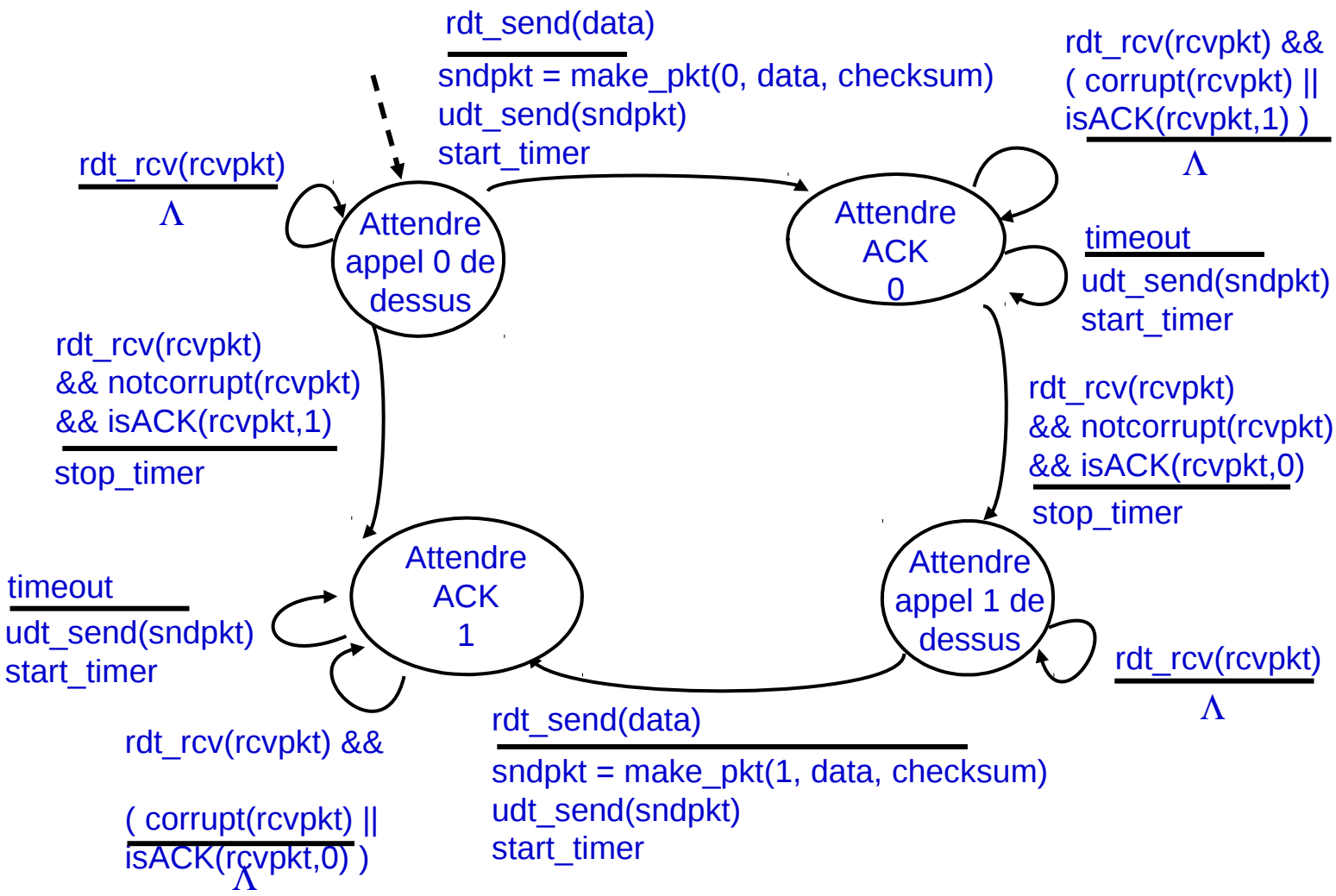
Nouvelle hypothèse: le canal peut perdre des paquets (données ou ACKs)

- le checksum, num de seq, ACKs, et la retransmission sont nécessaires mais non suffisants

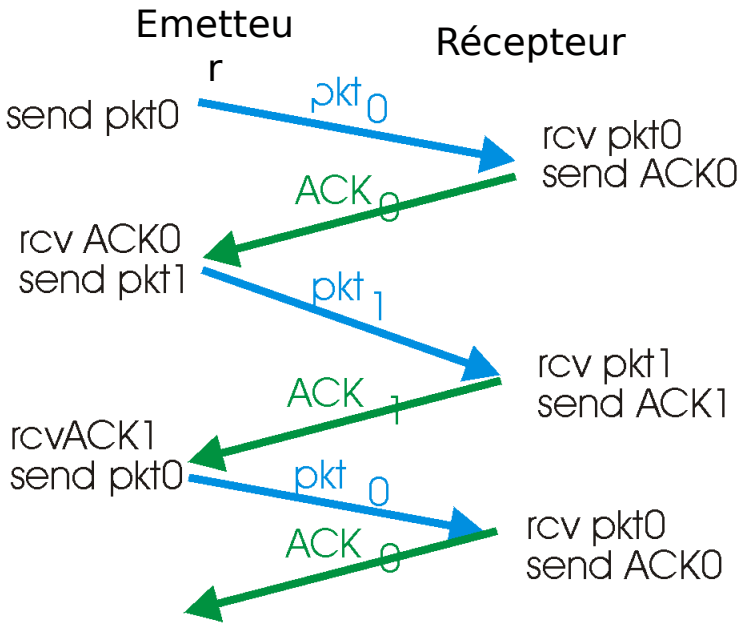
Approche: L'émetteur attend l'ACK sur un intervalle de temps résonable

- Il retransmet le paquet si l'ACK n'arrive pas sur cet intervalle du temps
- Si le paquet (ou l'ACK) met un retard (mais non perdu):
  - La retransmission dupplique le paquet, mais l'utilisation du num de seq peut remédier à la dupplication
  - Le récepteur doit spécifier le num du séq du paquet qui vient d'être aquitté
- Nécessite l'utilisation d'un timer

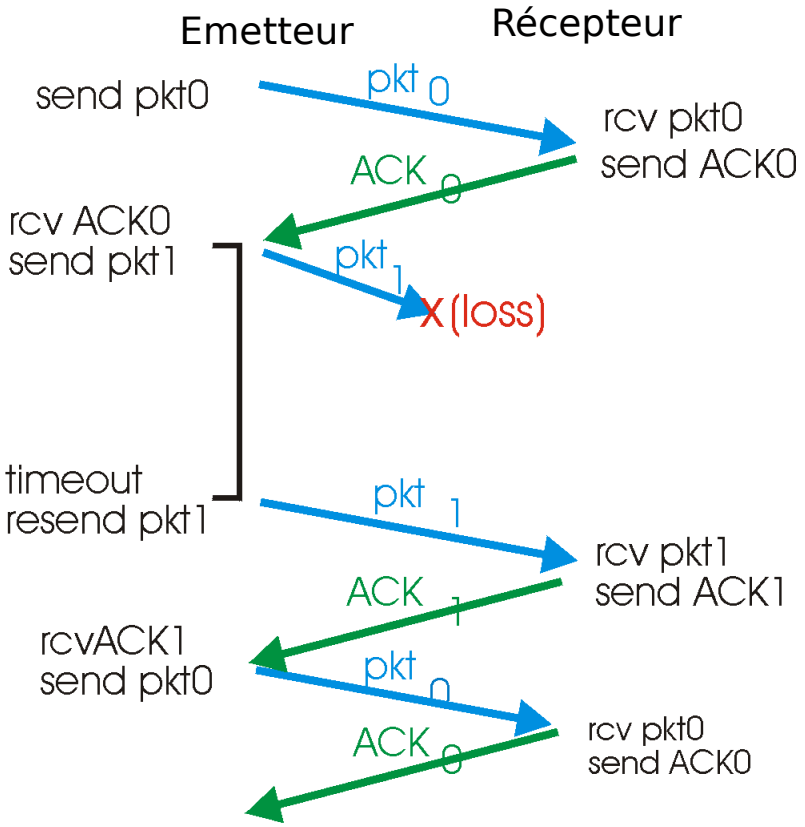
rdt3.0 : Emetteur



rdt3.0 : Exemple

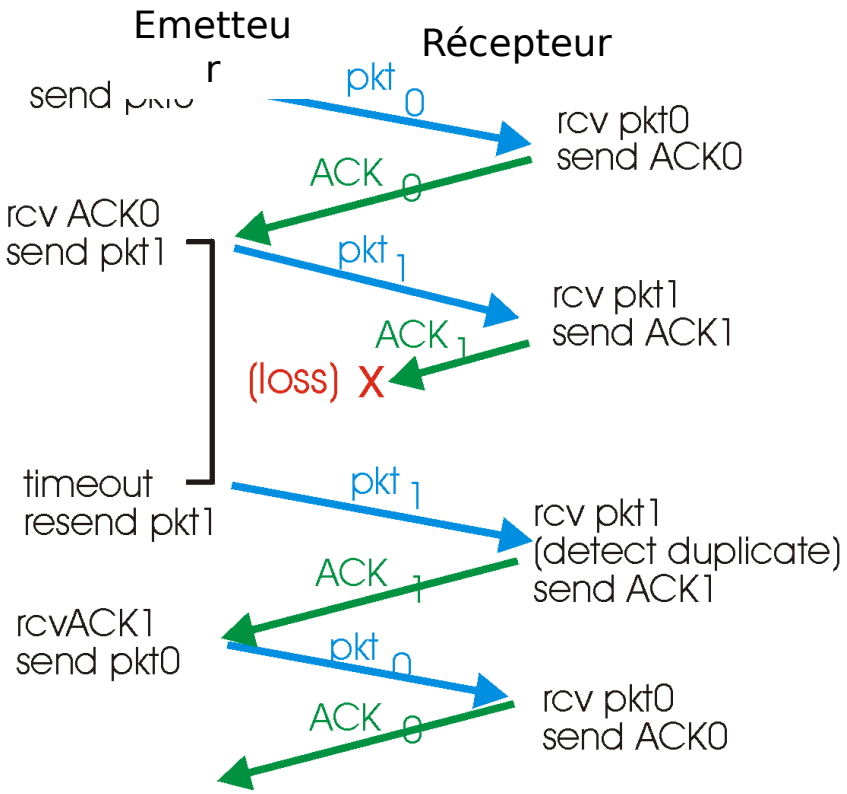


(a) Opération sans perte

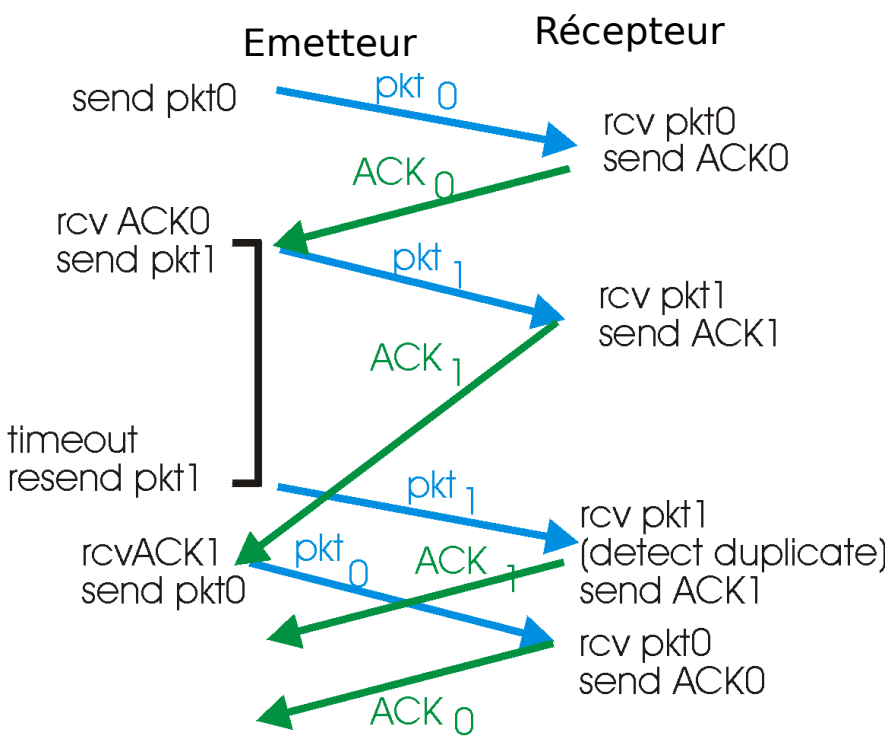


(b) perte d 'un paquet

rdt3.0 Exemple (suite)



(c) perte d 'un ACK



(d) timer expire avant
l'arrivée de l 'ACK

Performance de rdt3.0

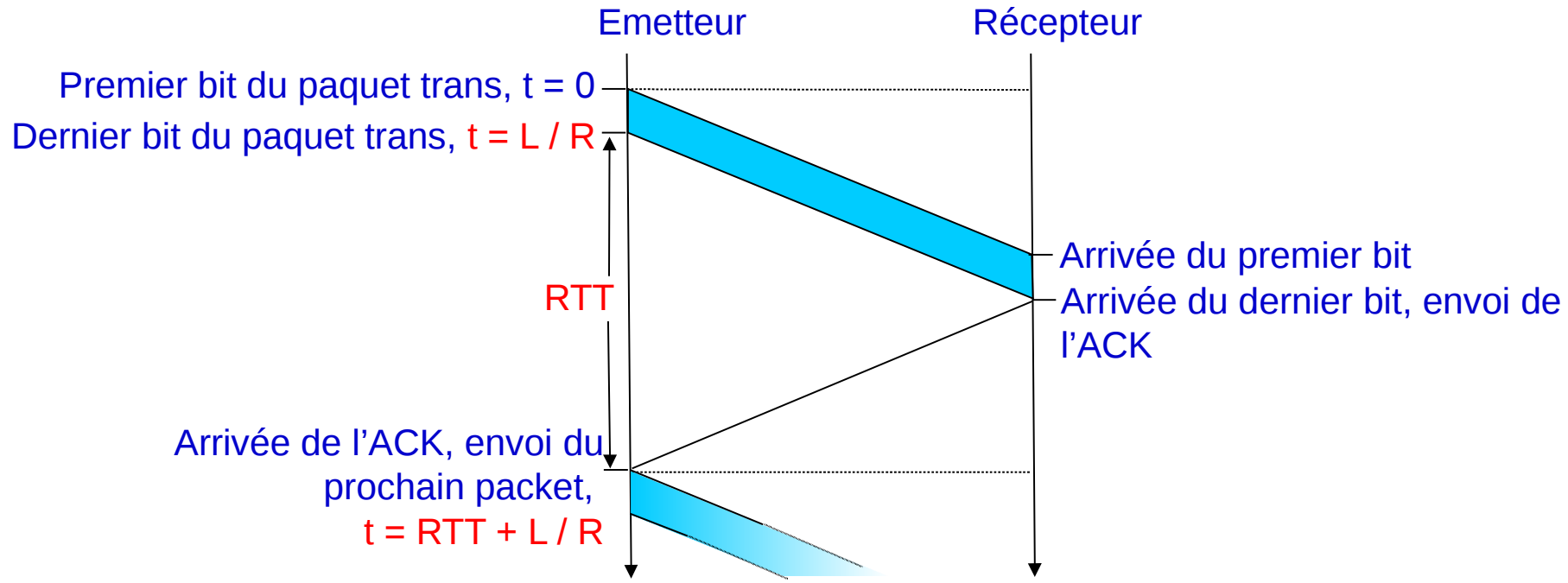
- rdt3.0 fonctionne, mais les performances sont mauvaises
- example: un lien à 1 Gbps, délai de bout-en-bout est 15 ms, taille du packet 1Koctets :

$$T_{transmi} = \frac{L \text{ (long du paquet en bits)}}{R \text{ (taux de transmission, bps)}} = \frac{8kb/pkt}{109\ b/sec} = 8 \text{ microsec}$$

$$U_{Emett} = \frac{L / R}{RTT + L / R} = \frac{.008}{30.008} = 0.00027$$

- U Emett: utilisation – la proportion du temps d'occupation pour l'émission
- Paquet de 1Koctets chaque 30 msec -> 267Kbits/sec à travers un lien de 1 Gbps
- Le protocole réseau limite l'utilisation des ressources physiques!

rdt3.0: stop-and-wait

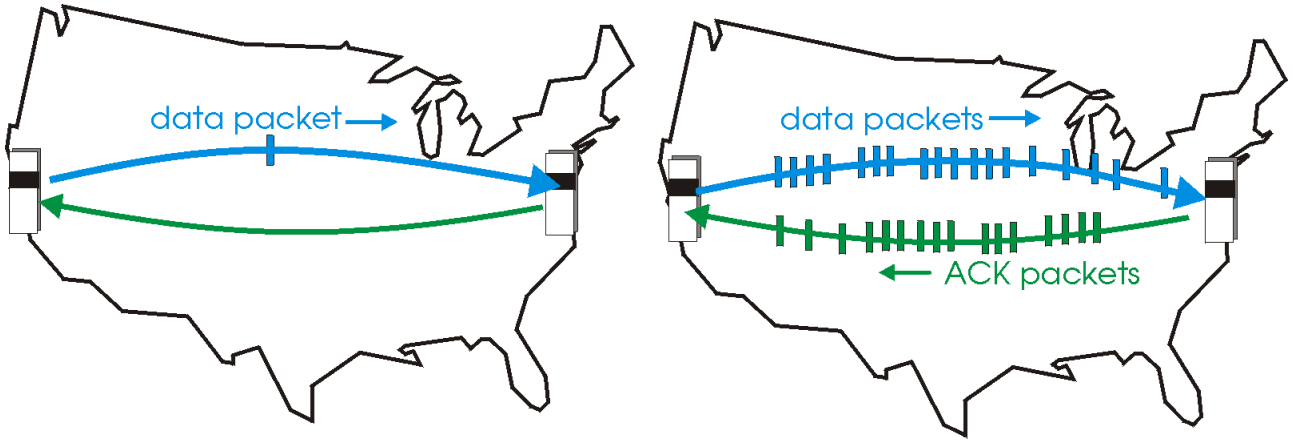


$$U_{Emett} = \frac{L / R}{RTT + L / R} = \frac{.008}{30.008} = 0.00027$$

Protocole Pipeliné

Pipelinage: L'émetteur se permet d'envoyer plusieurs paquets avant de recevoir l'ACK du premier

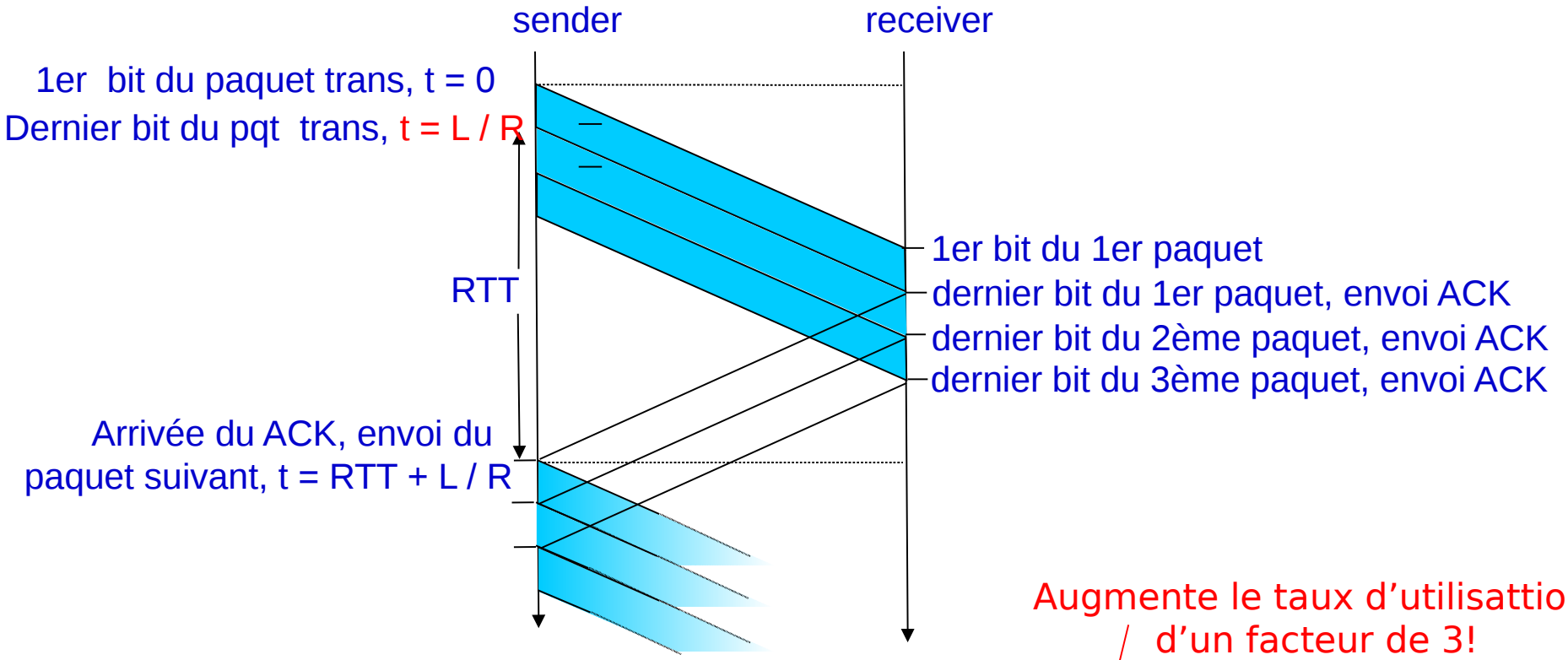
- L'intervalle des num de séquence doit augmenter
- Utilisation de mémoire Tampon au niveau de l'émetteur et/ou récepteur



Protoole stop-and-wait Protoole pipeliné

- Deux formes génériques de protocole pipelinés: *go-Back-N, répétition sélective*

Pipelinage: augmenter l'utilisation



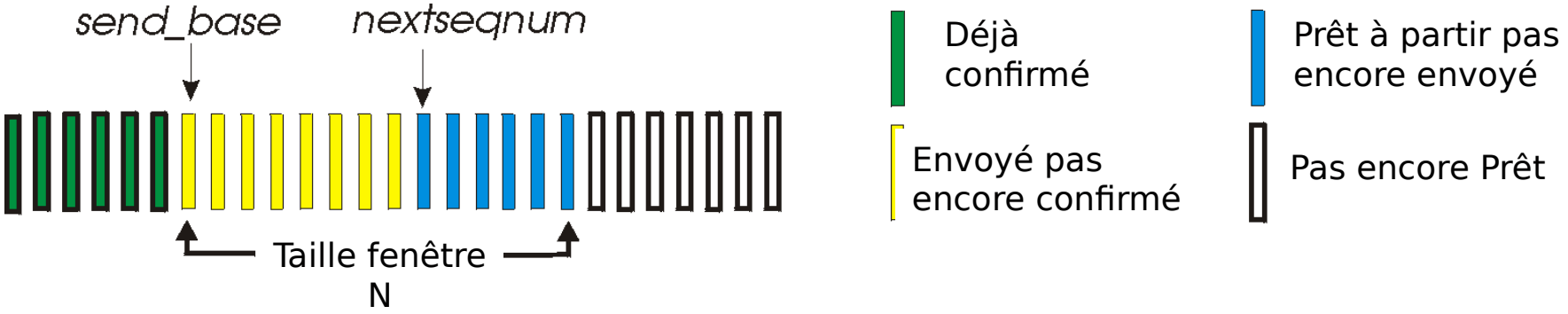
Augmente le taux d'utilisattion d'un facteur de 3!

$$U_{sender} = \frac{3 * L / R}{RTT + L / R} = \frac{.024}{30.008} = 0.0008$$

Go-Back-N “protocole de fenêtre glissante”

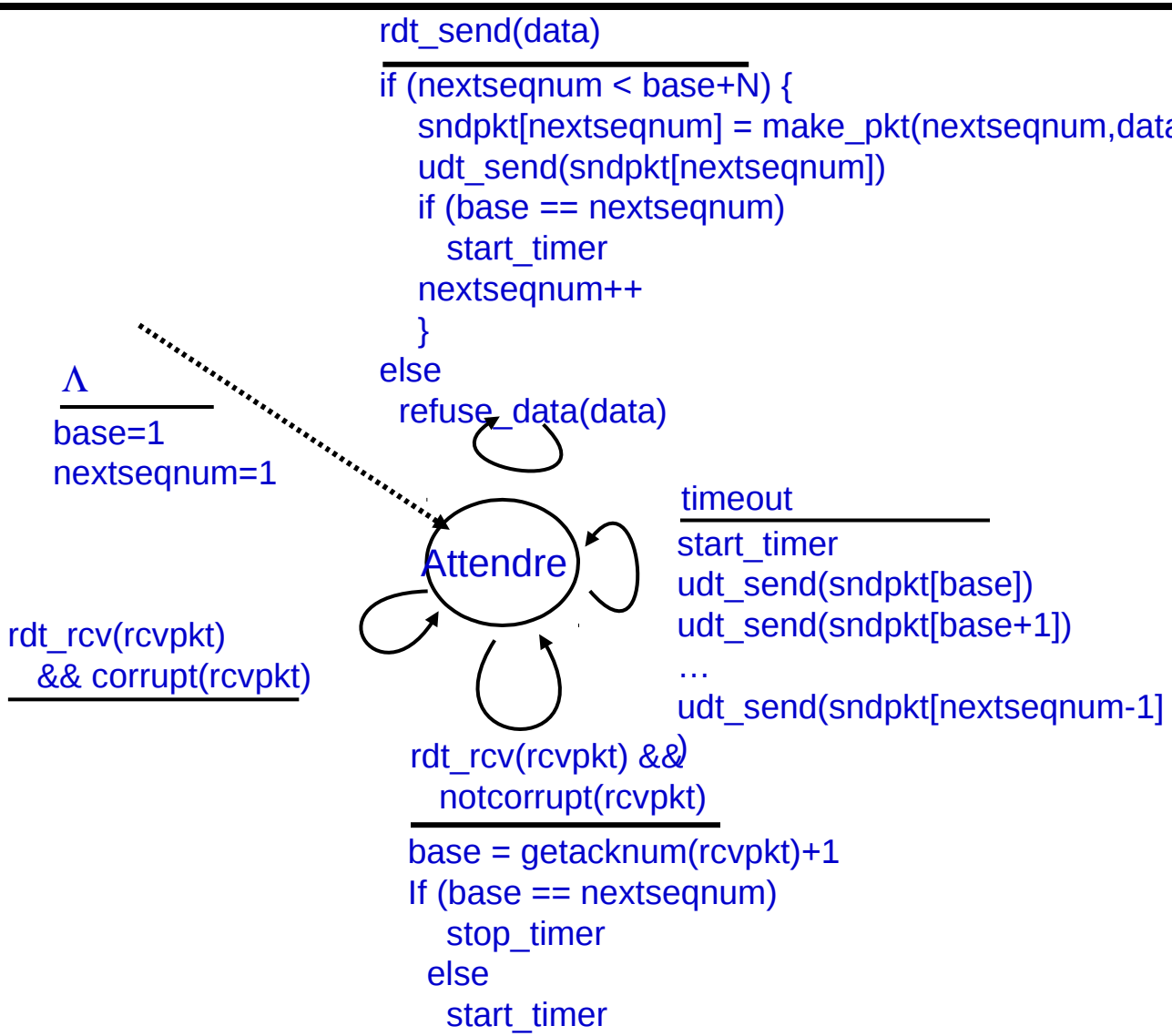
Emetteur:

- taille du champ num de séqu est de k-bits à l’en-tête
- “Fenêtre” de taille N : est le nombre consécutifs de paquets sans accusés de réception et qui sont permis à l’émission

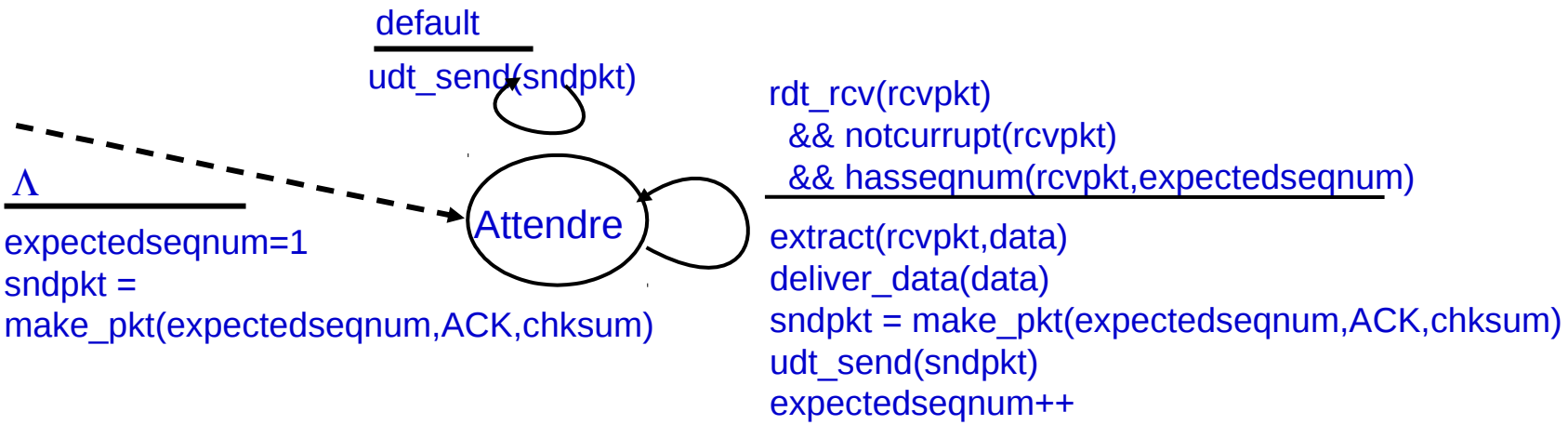


- ACK(n): numérotation des ACKs sera interprété comme un ACK cumulatif
- Actionner un timer si le nextseqnum=send_base
- *timeout(n):* retransmettre le paquet *n* et tous les paquets de num de séq sup à *n* en respectant la limite de la fenêtre glissante

GBN: MEF étendue de l'émetteur



GBN: MEF étendue du récepteur

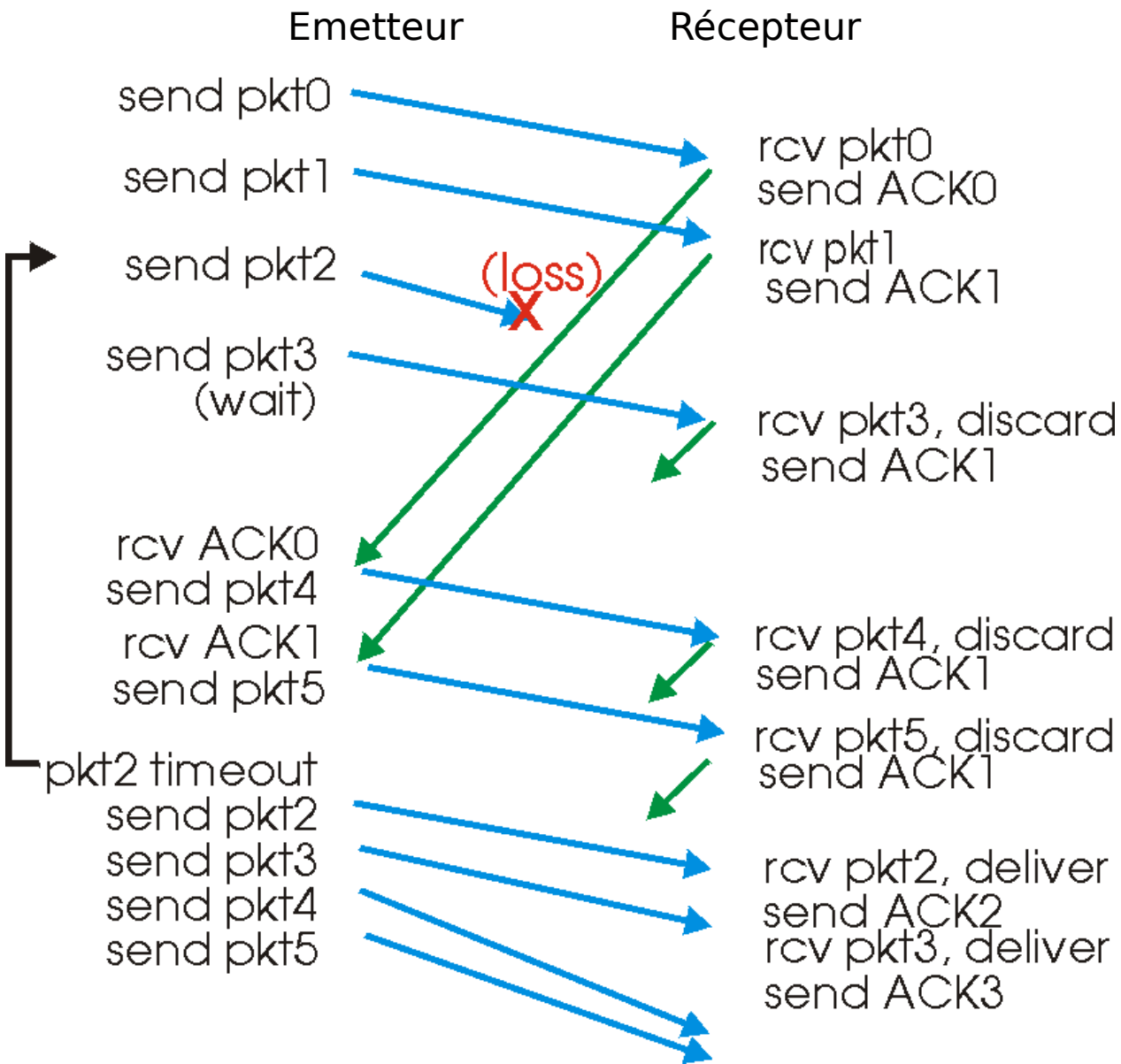


ACK-uniquement: Envoi ACK pour les paquets reçus correctement avec le plus grand num de séquence

- peut générer des ACKs duppliqués
- nécessite uniquement la mémorisation du **`expectedseqnum`**

- paquets non dans l'ordre:
  - écarter (ne pas mettre en buffer)!
  - Ré-envoi des paquets-ACK avec le num de seq le plus élevé dans l'ordre

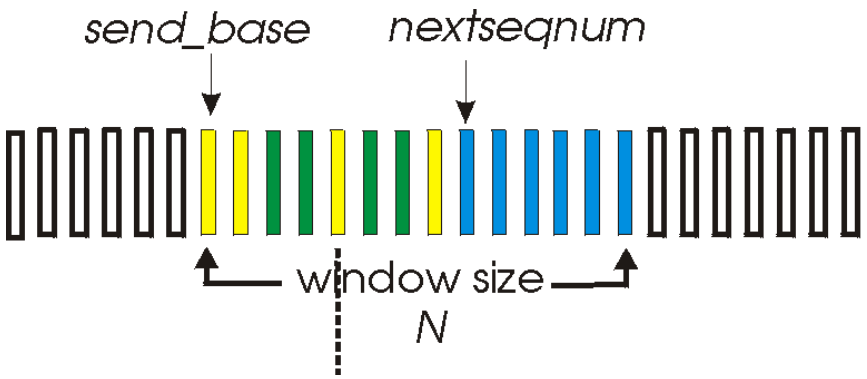
GBN : Exemple

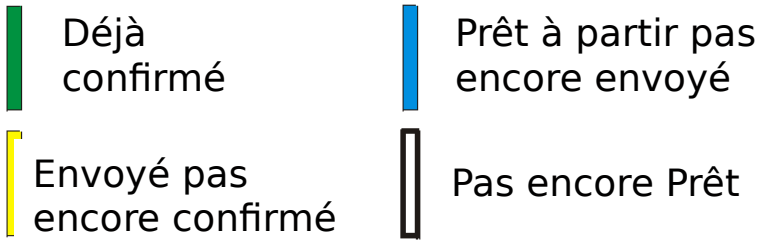

Emetteur
Récepteur
send pkt0
send pkt1
send pkt2
(loss)
send pkt3
(wait)
rcv ACK0
send pkt4
rcv ACK1
send pkt5
pkt2 timeout
send pkt2
send pkt3
send pkt4
send pkt5
rcv pkt0
send ACK0
rcv pkt1
send ACK1
rcv pkt3, discard
send ACK1
rcv pkt4, discard
send ACK1
rcv pkt5, discard
send ACK1
rcv pkt2, deliver
send ACK2
rcv pkt3, deliver
send ACK3

Répétition sélective (Selective Repeat)

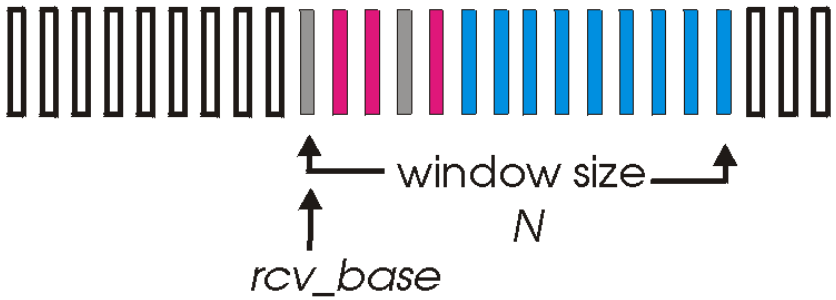
- Le récepteur aquitte d'une façon individuelle tous les paquets reçus correctement.
  - Un buffer de paquets est nécessaire pour mettre éventuellement les paquets à délivrer à la couche application dans l'ordre
- L'émetteur retranmet uniquement les paquets pour lesquels l'ACK n'est pas reçu
  - L'émetteur active le timer pour chaque paquet non acquitté
- Fenêtre d'émission
  - N num de séquence consecutif
  - limte encore le nombre de paquets émis sans avoir reçus d'ACK

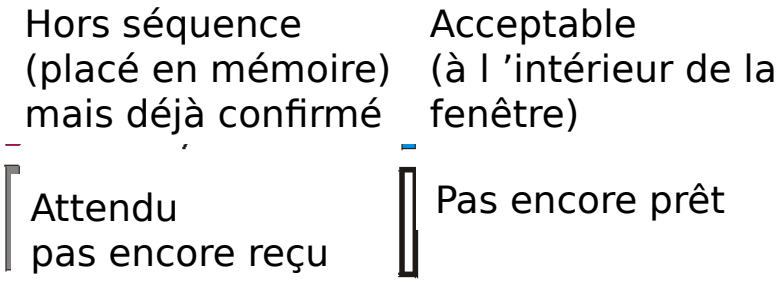
Répétition sélective : Fenêtre d'émission et de réception





(a) Vue de l 'émetteur des numéros de séquence





(b) Vue du récepteur des numéros de séquence

Répétition sélective

Emetteur

Donnée de la couche supérieure :

- Si le num de seq suivant est dans sa fenêtre, envoi du paquet

Expiration du temps imparti (n):

- retransmission du paquet n

réception d'un ACK(n) dans [sendbase,sendbase+N]:

- marquer paquet n comme reçu
- Si num de seq de ce paquet est égal à send_base, la fenêtre se déplace jusqu'au paquet non confirmé de plus faible numéro
- Les paquets en attente en buffer seront transmis si la fenêtre couvre leur num de séq

Récepteur

paquet n dant [rcvbase, rcvbase+N-1]

- Envoi de ACK(n)
- S'il n'est pas dans l'ordre: mise en buffer
- Dans l'ordre (=rcv_base): délivrer à la couche supérieur (+ tous les autres paquets en buffer qui suivent dans l'odre), avancer la fenêtre jusqu'au paquet non reçu encore

paquet n dans [rcvbase-N,rcvbase-1]

- Envoi de ACK(n)

dans les autres cas:

- ignorer le paquet

Répétition sélective : Exemple