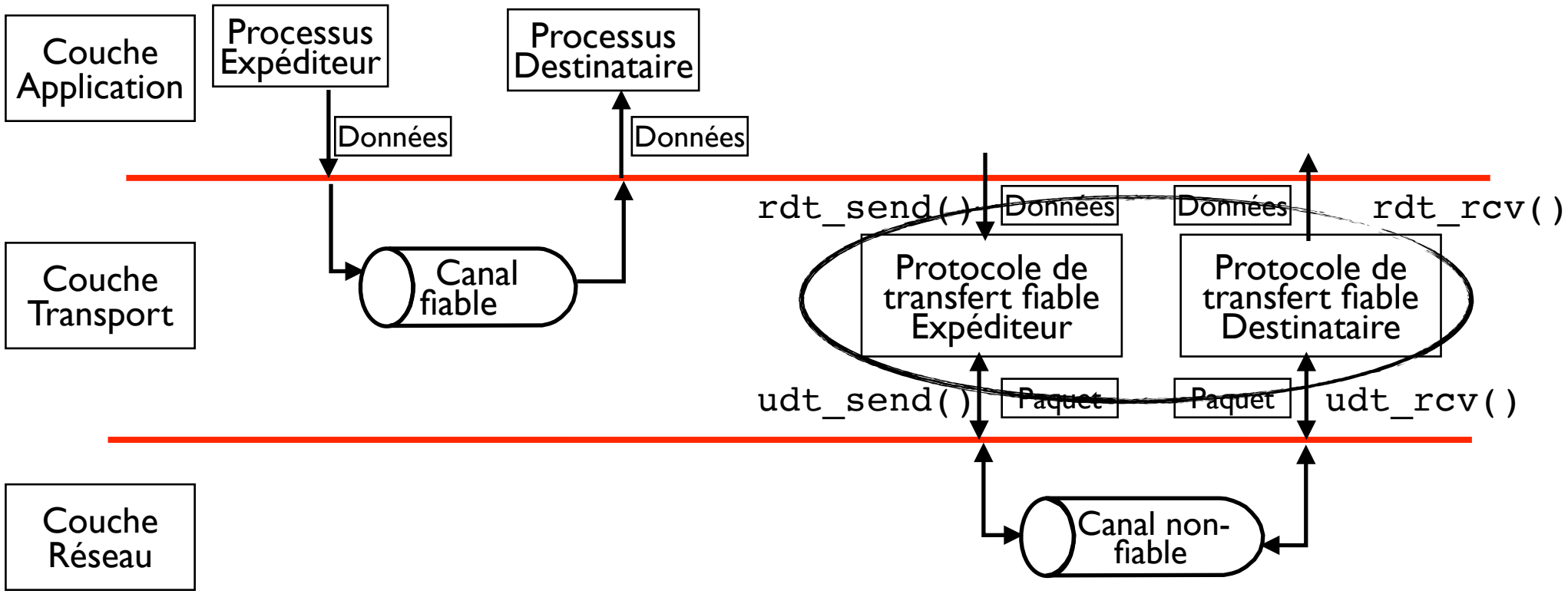
Suite couche Transport

Plan

1. Services de la couche Transport
2. Multiplexage et démultiplexage
3. Transport sans connexion : UDP
4. Principes du transfert de données fiable
5. Transport orienté connexion : TCP
6. Principes du contrôle de congestion
7. Contrôle de congestion TCP
8. Limites de TCP & Résumé

Transfert fiable sur un canal non-fiable ?

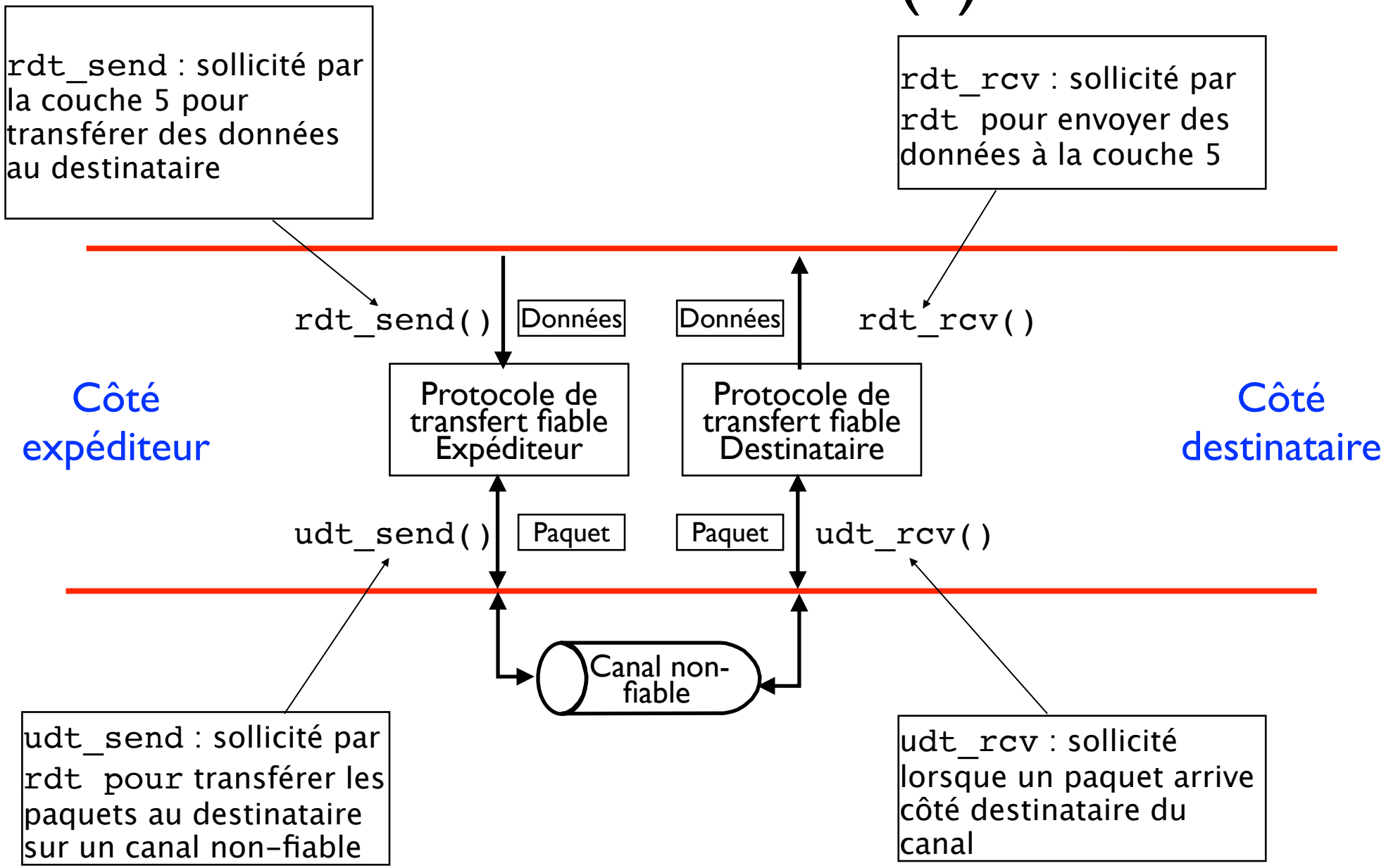
- rdt = "reliable data transfer" (fiable)
- udt = "unreliable data transfer" (non-fiable)
- `rdt_send()` ∈ couche 5
- `udt_send()` ∈ couche 3 (IP Best effort)
- Comment construire `rdt` à partir de `udt` ?



a) Service voulu

b) Service existant

Principes d'un transfert de données fiable (2)

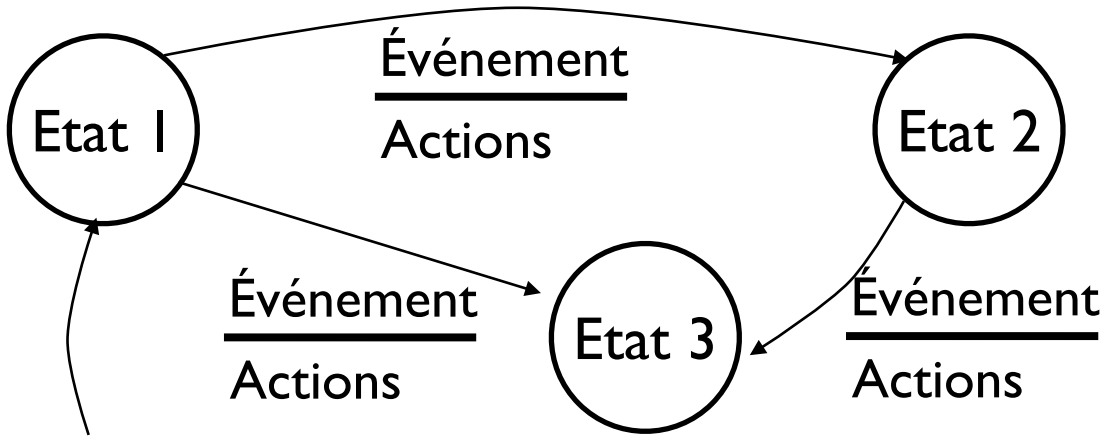


- La complexité du protocole rdt dépend des caractéristiques du canal

Principes d'un transfert de données fiable (3)

- Construction d'un protocole de transfert de données fiable
  - *"reliable data transfer"*
  - appellation : rdt x.x
- Considère uniquement le transfert de données unidirectionnel
  - Les paquets de contrôle circulent eux dans les deux sens
- Description par des automates à nombre d'états finis

L'état suivant est déterminé uniquement par le prochain événement

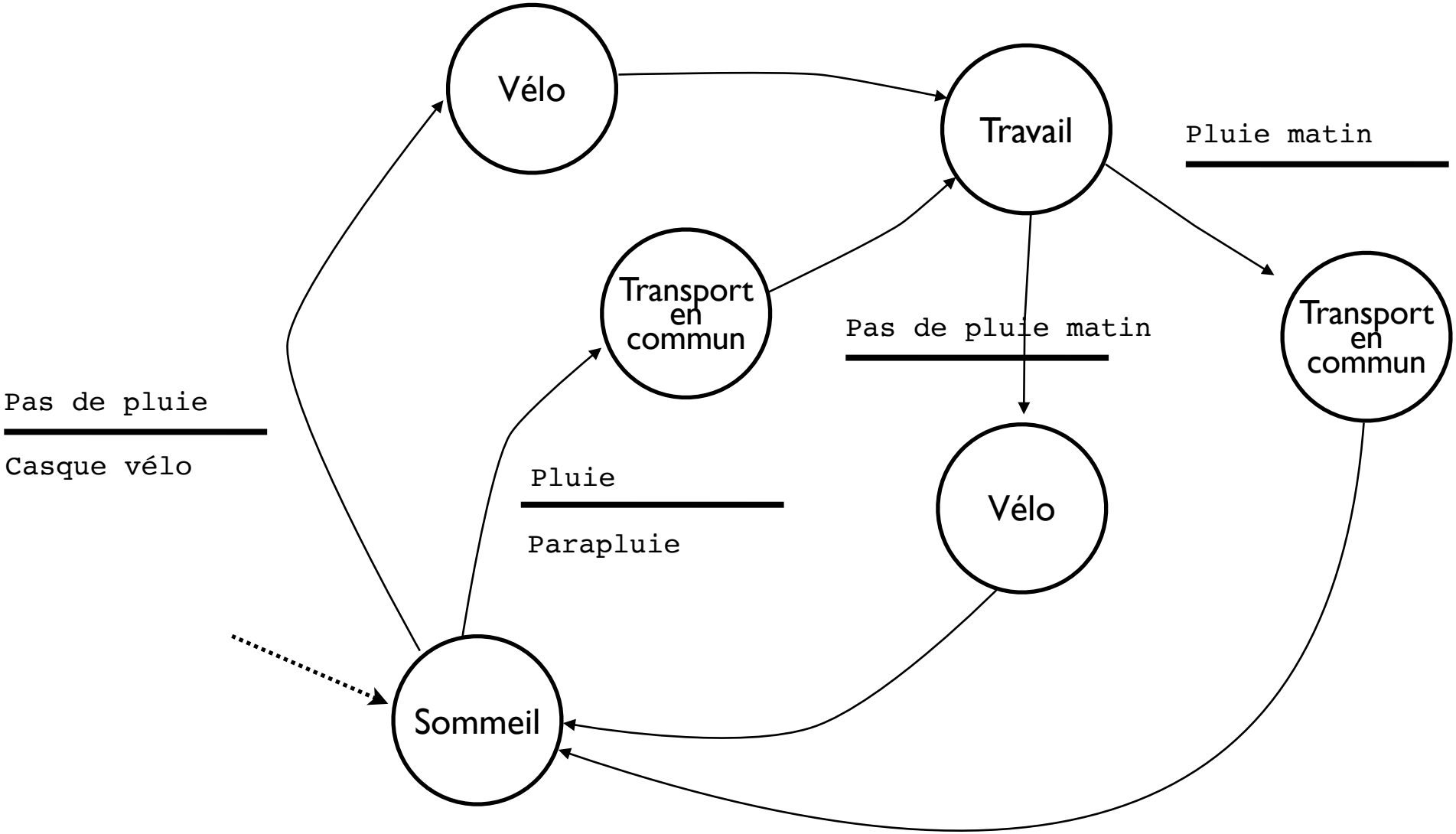


Les événements causent les transitions d'état

Les actions sont prises lors des transitions d'état

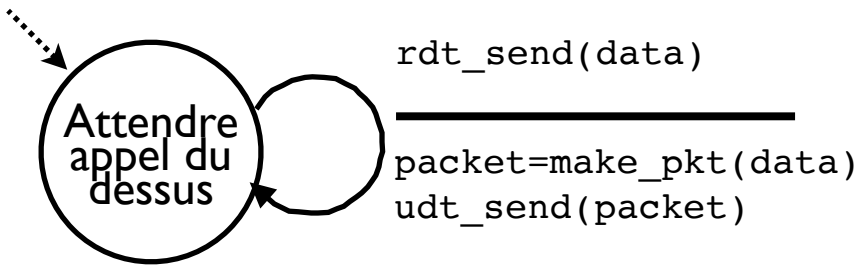
Exemple d'automate

- Modélisation des journées de boulot d'un collègue…

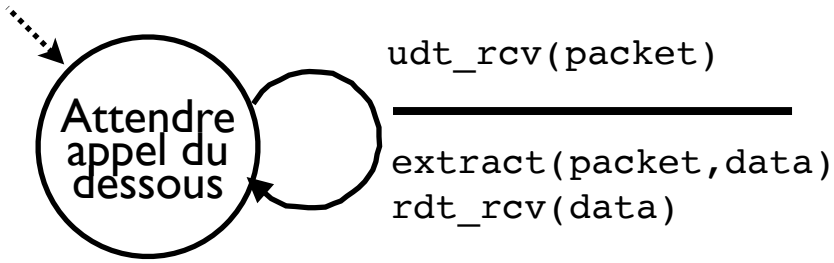


rdt1.0 : transfert fiable sur canal fiable

- Hypothèse : canal totalement fiable
  - pas d'erreur bits
  - pas de perte de paquets
- Automates différents pour l'expéditeur et le destinataire
  - l'émetteur envoie les données dans le canal inférieur
  - le destinataire reçoit les données du canal inférieur



Expéditeur



Destinataire

rdt2.0 : canal avec erreurs binaires

- Hypothèse : canal introduit des erreurs bits dans les paquets
- Comment se rétablir de ces erreurs ?

  (a) Détection d'erreurs

  - Somme de contrôle

  (b) Accusé de réception

  - ACK : acquittement - le destinataire informe l'expéditeur que le paquet a été bien reçu
  - NAK : acquittement négatif - le destinataire informe l'expéditeur que le paquet comporte une erreur
  - Boucle de retour

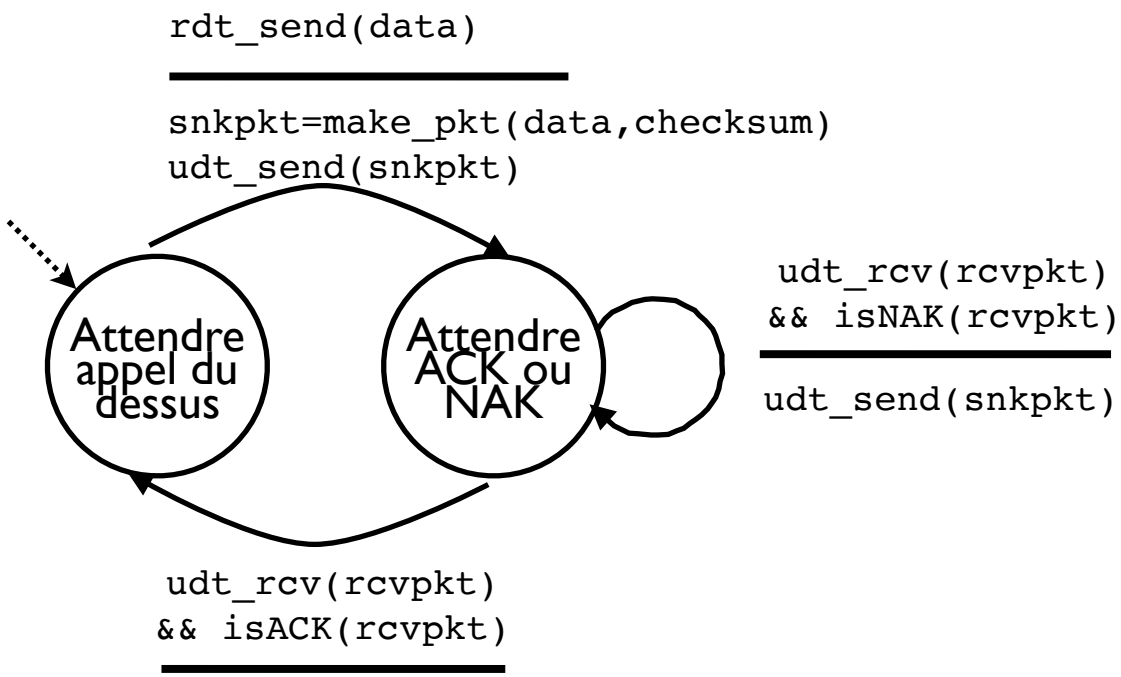
  (c) Retransmission des paquets erronés

  - À la réception d'un NAK

- Protocole "Send and wait"

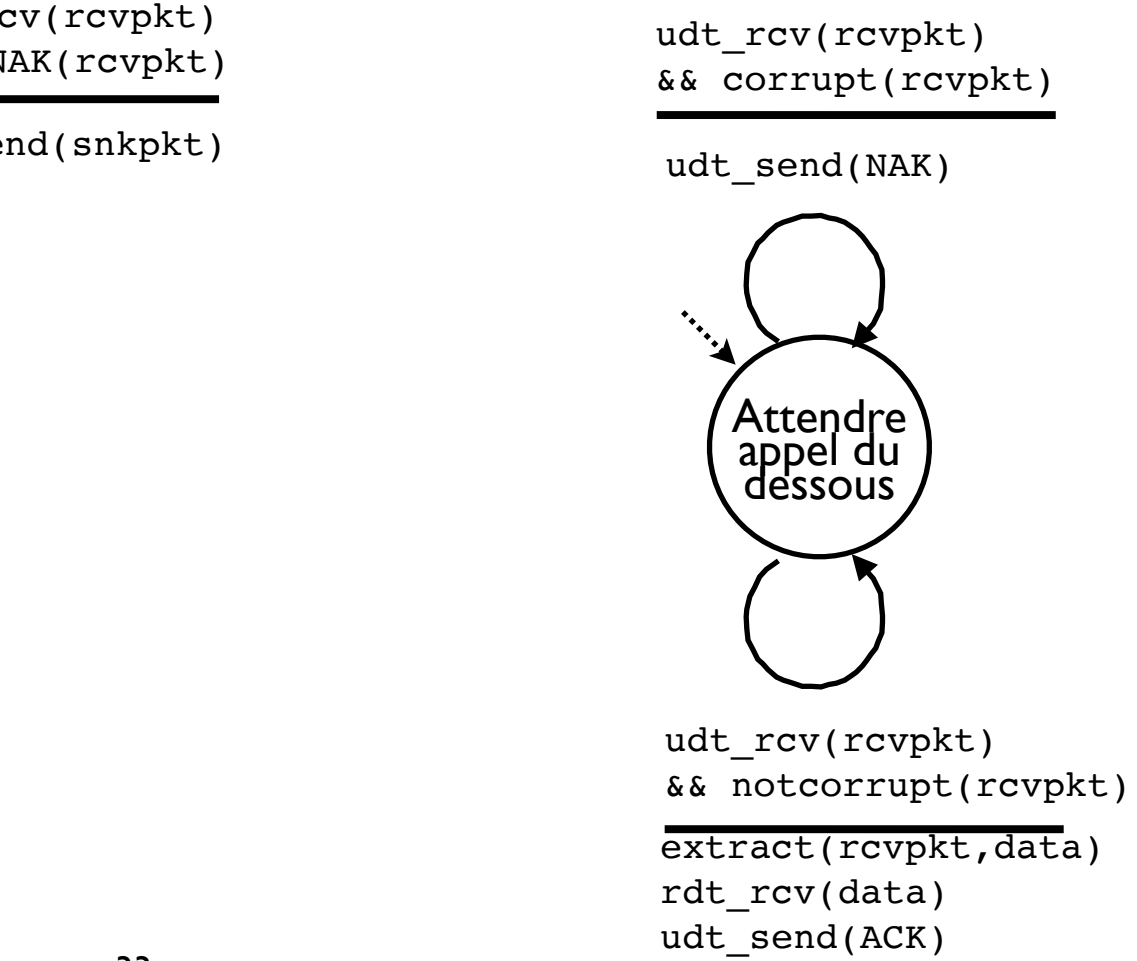
  - l'expéditeur émet un paquet ... puis se met en attente d'un ACK/NAK

rdt2.0 : description

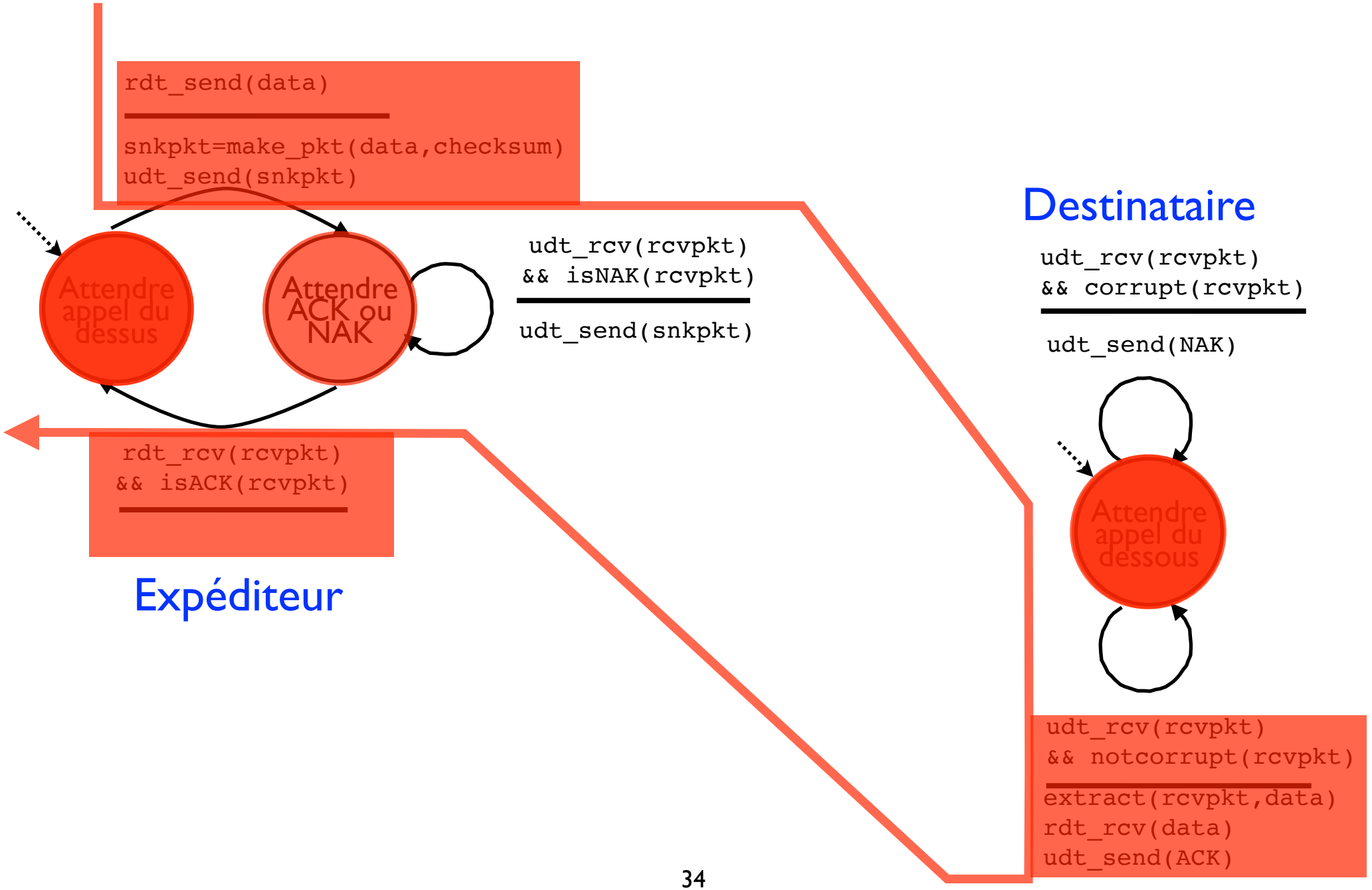


Expéditeur

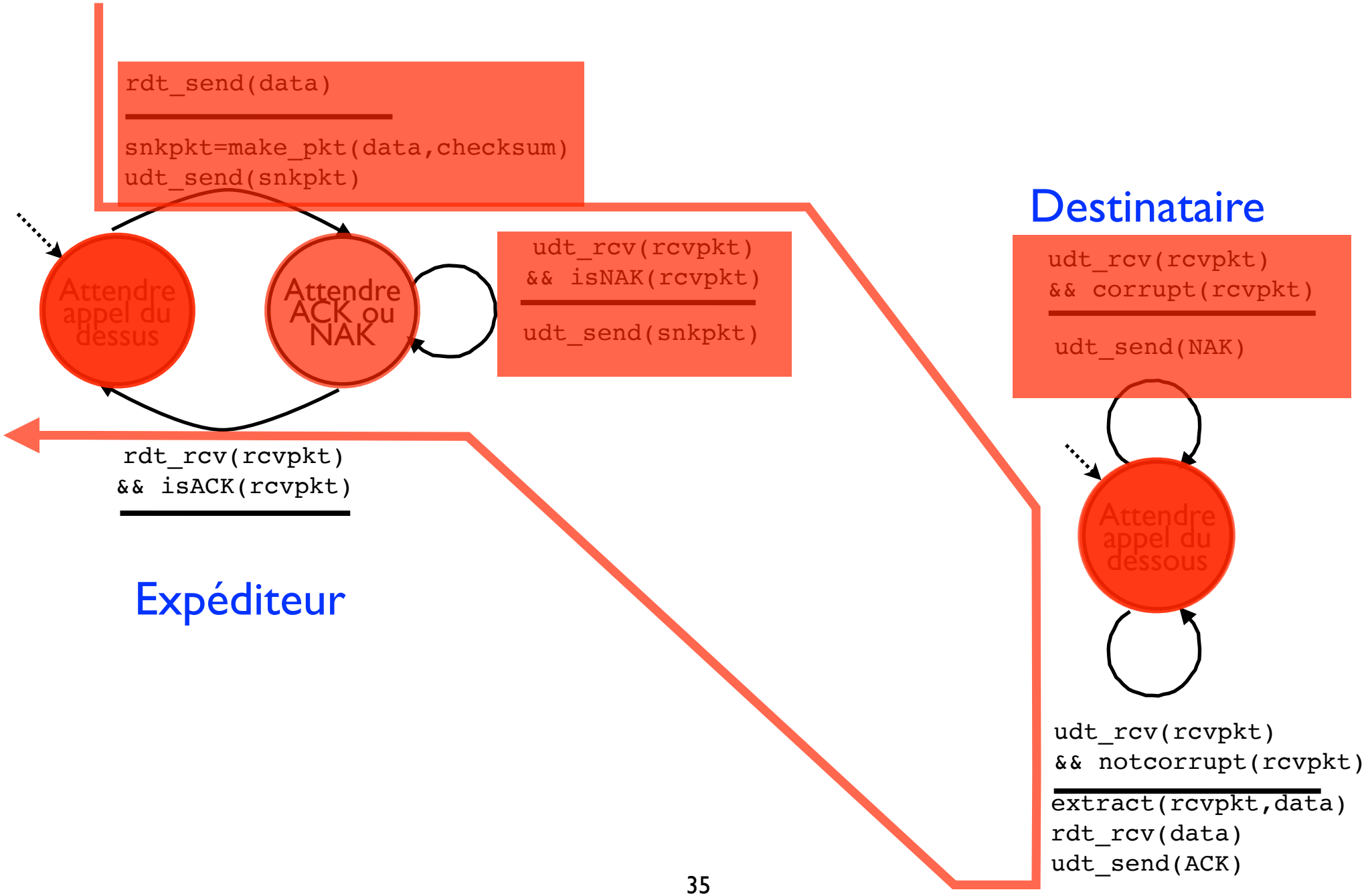
Destinataire



rdt2.0 : fonctionnement sans erreur



rdt2.0 : fonctionnement avec erreur



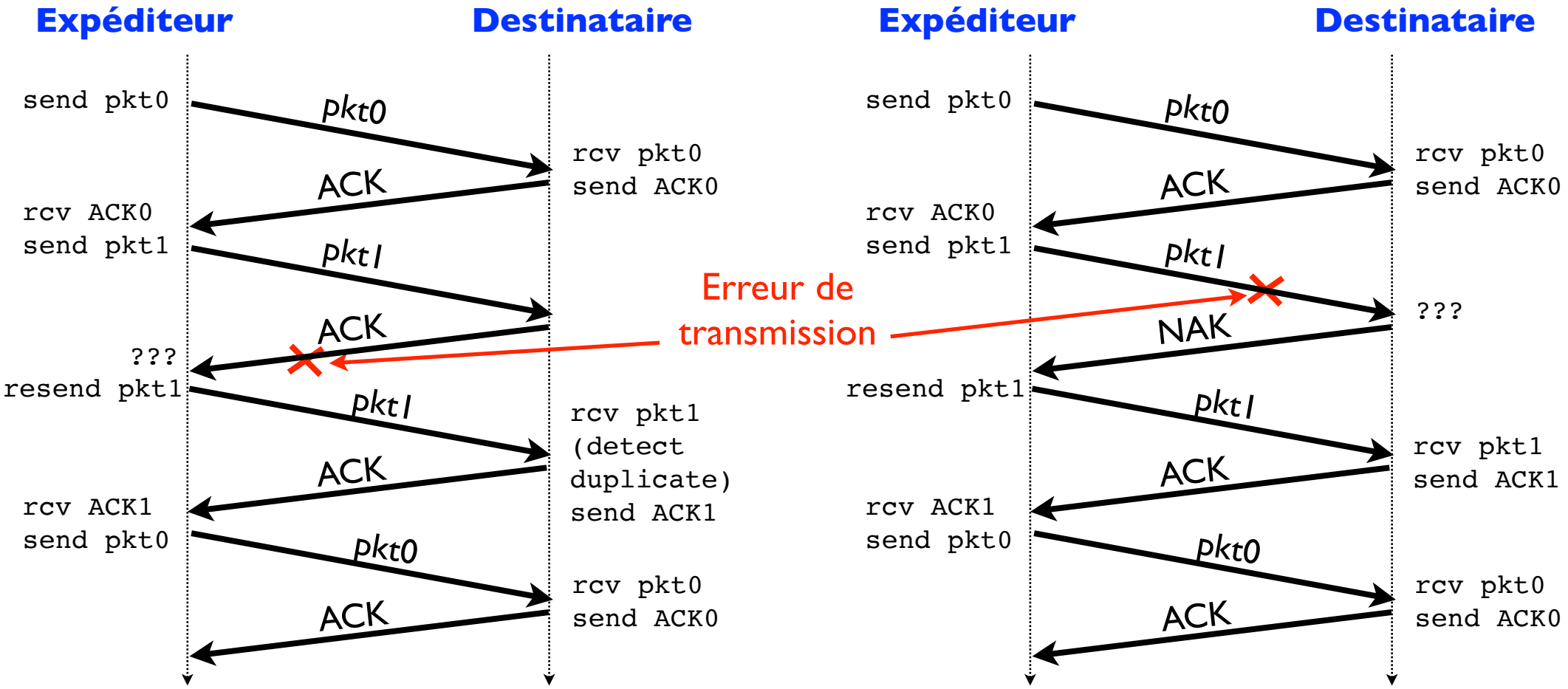
rdt2.0 : quel est le problème ?

- Si un segment de données arrive erroné 
- Mais que se passe-t-il si un ACK ou NAK arrive erroné ? 
- L'expéditeur ne sait pas si son paquet est correctement arrivé au destinataire
- Solution : simplement retransmettre le dernier paquet envoyé ?
  - Exemple : transmettre le code 4972 par téléphone
  - 4 "OK"; 9 "OK"; 7 "Pas OK"; 7 "OK"; 2 "#&@" ???
  -
  -
- Solutions possibles
  -
  -

rdt2.1 : numéro de séquence (1)

- Ajouter un numéro de séquence dans le segment
  - Qui identifie chaque segment
  - à la réception d'un ACK erroné, le segment est retransmis
  - si un duplicata arrive, le segment est supprimé
- Pour un protocole "Send and wait"
  - combien de bits sont nécessaires ?
  -
  -

rdt2.1 : numéro de séquence (2)



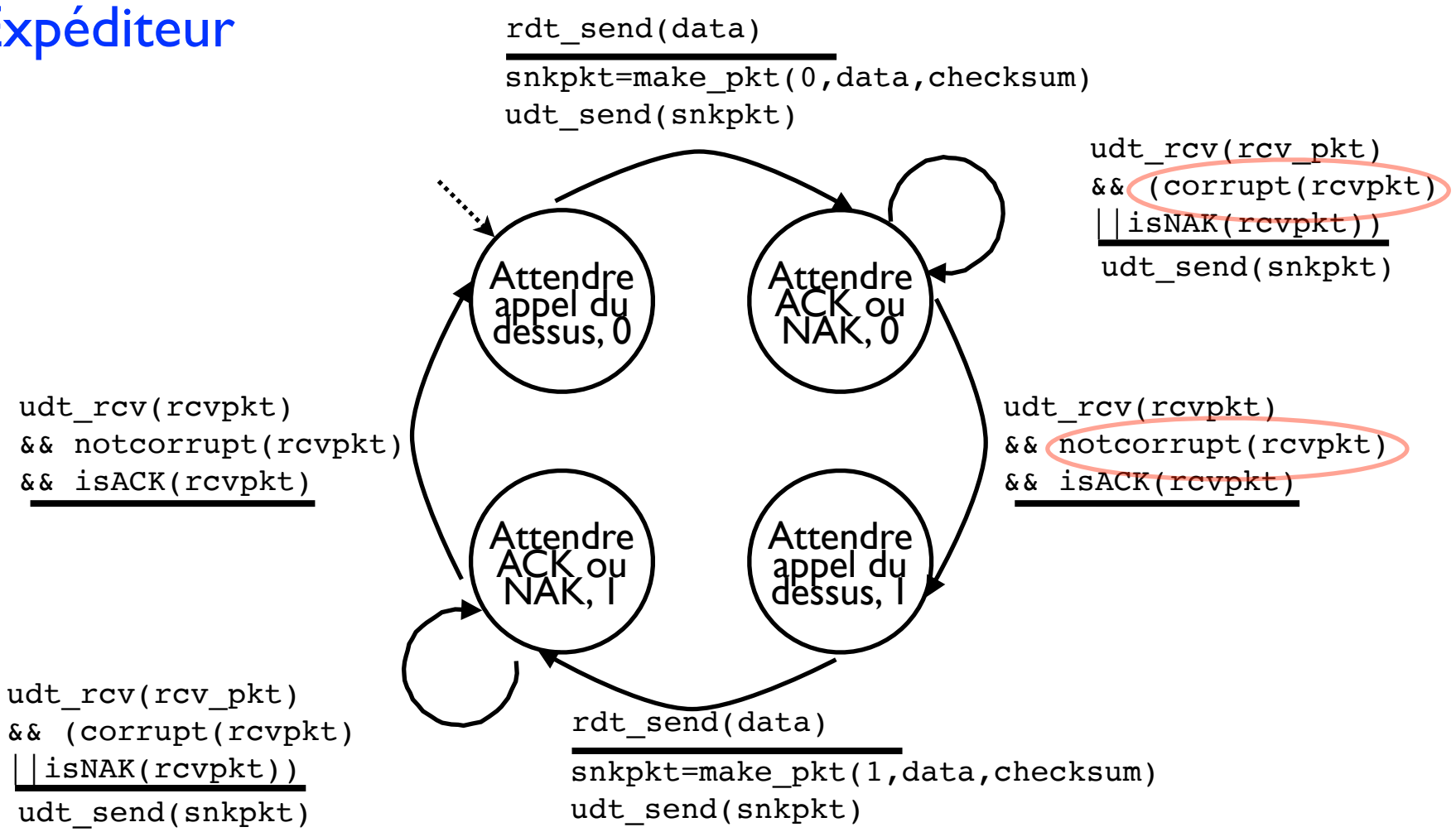
a) Erreur dans l'ACK

b) Erreur dans le segment

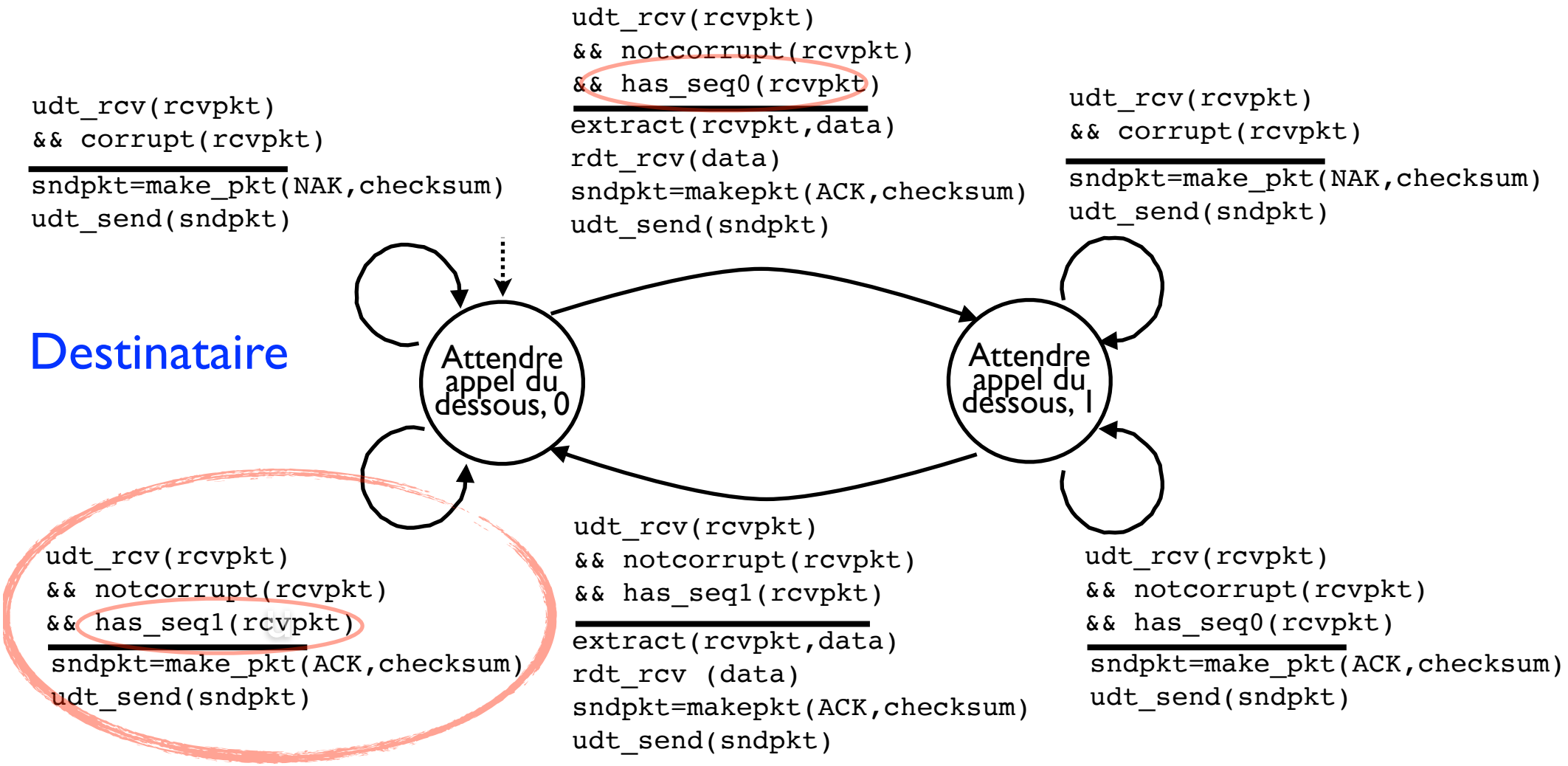
rdt2.1 : expéditeur gère les ACK/NAK erronés

- On double le nombre d'états des automates
  - pour mémoriser si le numéro de séquence du paquet courant vaut 0 ou 1

Expéditeur



rdt2.1 : destinataire gère les ACK/NAK erronés



rdt2.2 : un protocole sans NAK

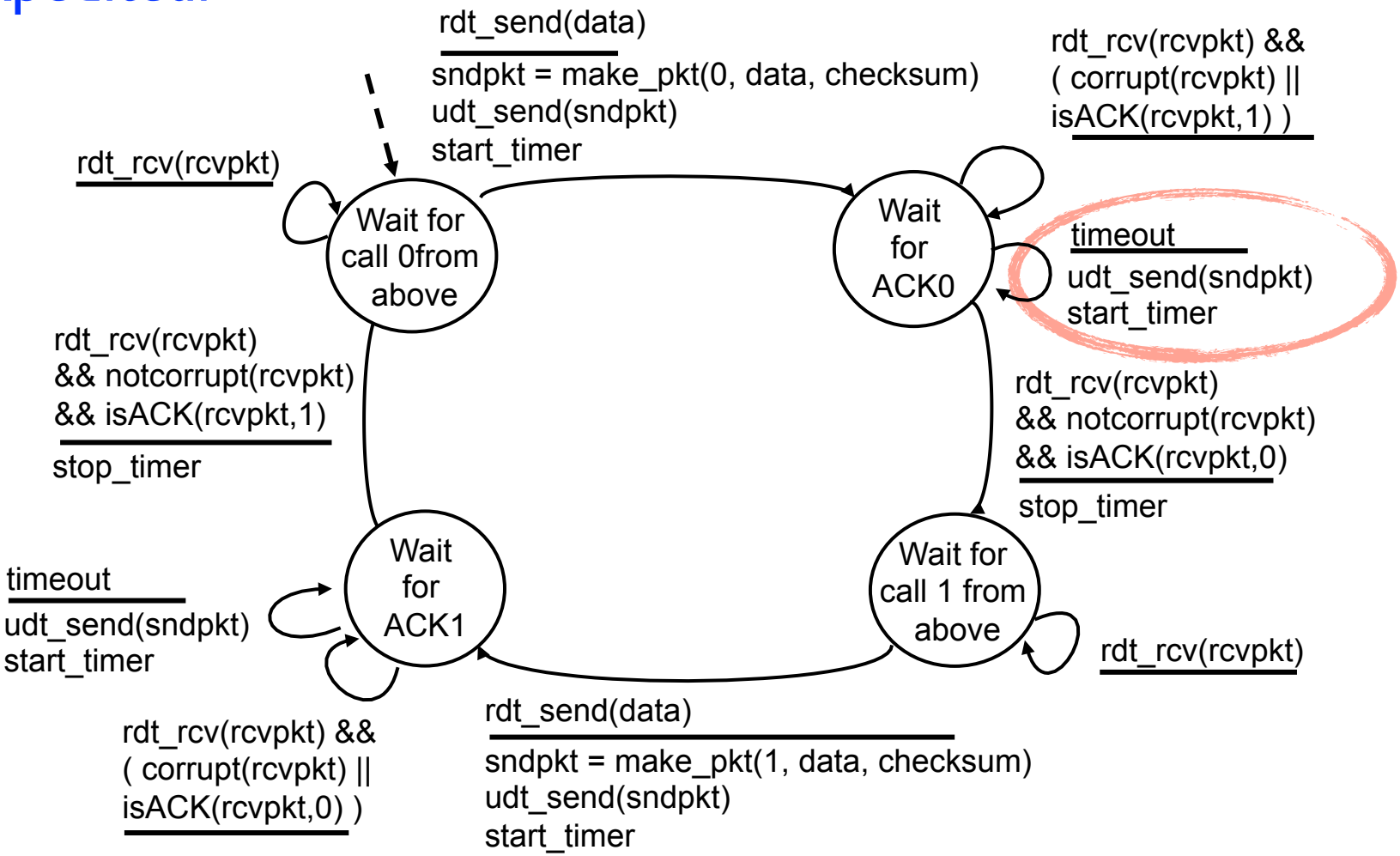
- Faire comme rdt2.1 mais uniquement avec des ACKs ?
- Au lieu d'un NAK...
  - ... le destinataire envoie un ACK associé au dernier segment correctement reçu
- Les ACKs doivent être numérotés
  - le destinataire doit explicitement inclure le #séquence du paquet dont il accuse la bonne réception
- Recevoir **2 ACK identiques ⇔ recevoir 1 NAK**
  - et donc déclenche la retransmission du paquet courant

rdt3.0 : canal avec erreurs et pertes

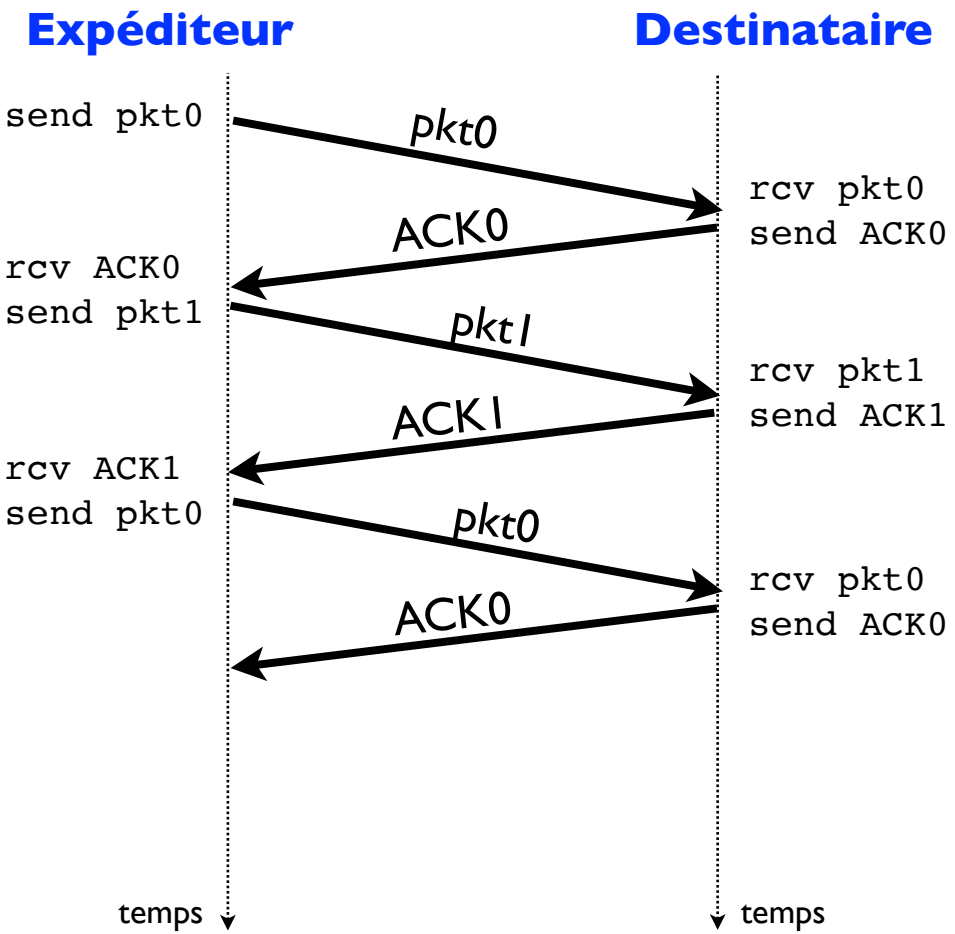
- Hypothèse : canal peut perdre des paquets (données et ACKs)
  - somme de contrôle, #séquence, ACK, retransmission : pas suffisants !
- Que faut-il de plus ?
  -
  -
  -
  - Si le paquet ou l'ACK arrive simplement trop tard ???
    - paquets dupliqués mais #séquence pour y répondre
  - Le destinataire doit spécifier le #séquence du paquet qu'il acquitte

rdt3.0 : canal avec erreurs et pertes

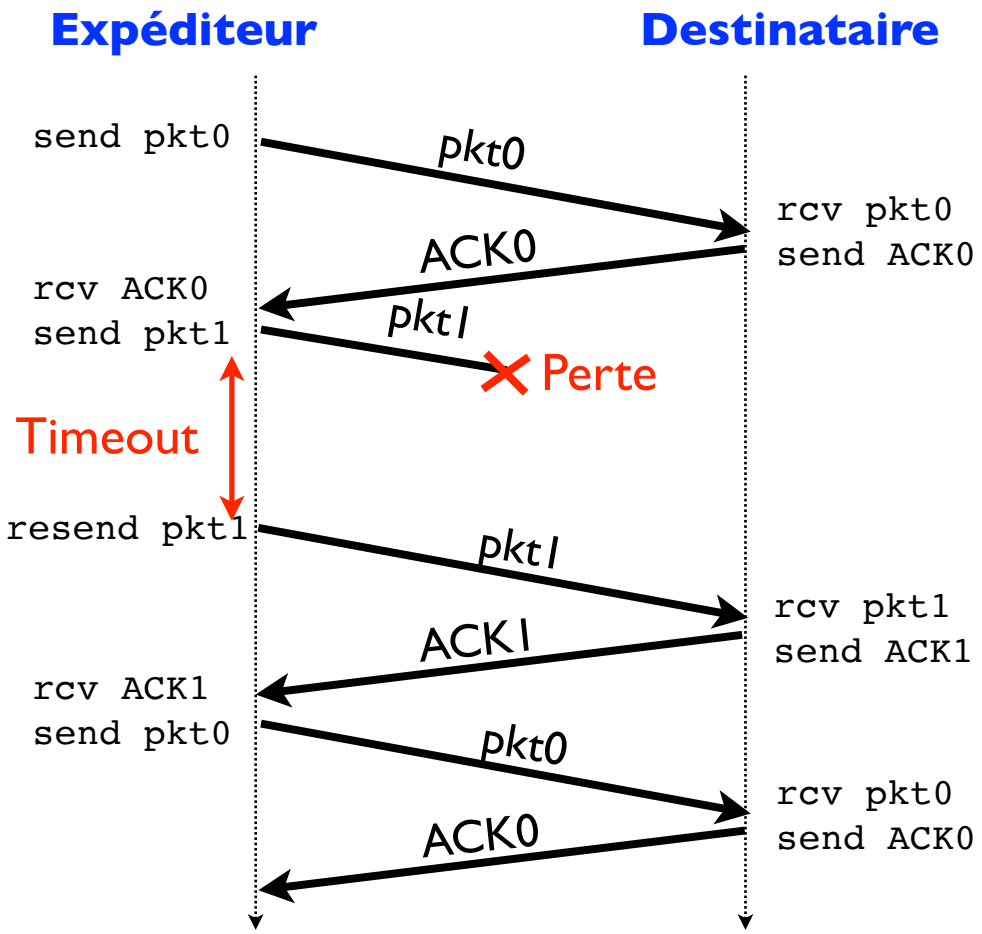
Expéditeur



rdt3.0 en action (1)

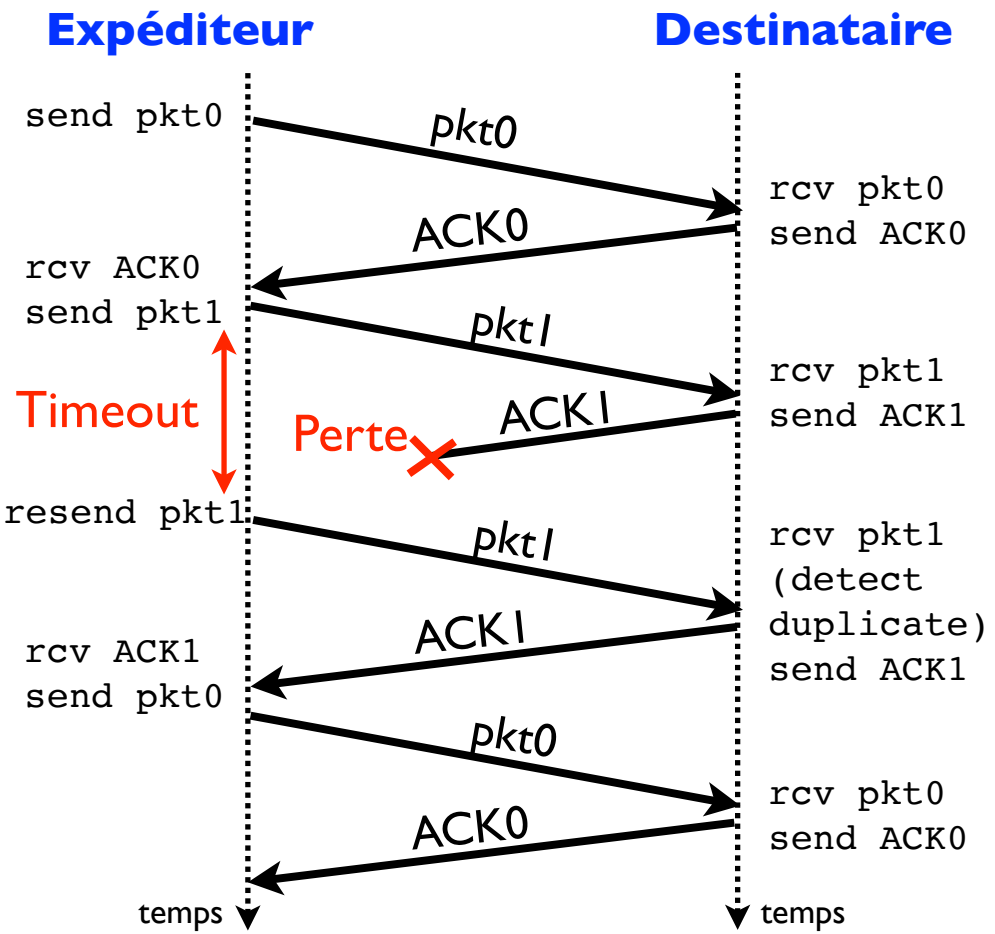


a) Sans perte

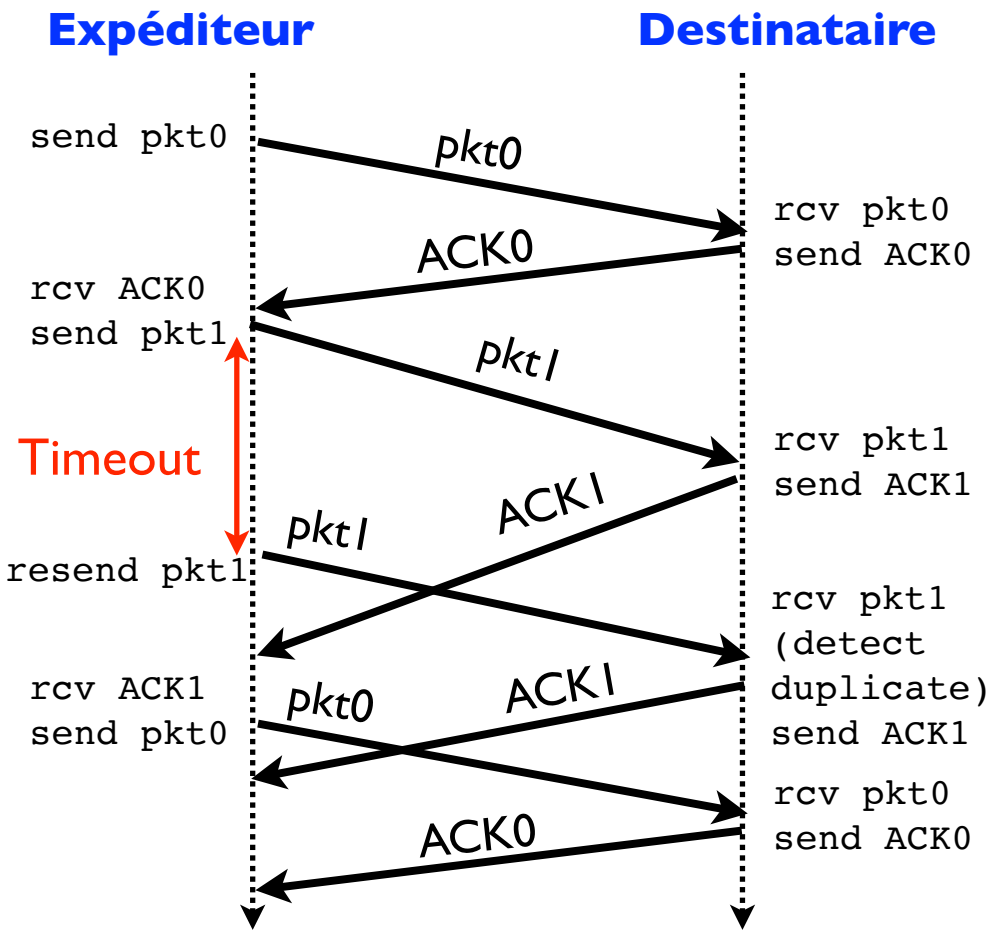


b) Paquet perdu

rdt3.0 en action (2)



c) ACK perdu



d) Expiration prématurée

Pièces maîtresse d'un protocole de transport de données fiable ("rdt")

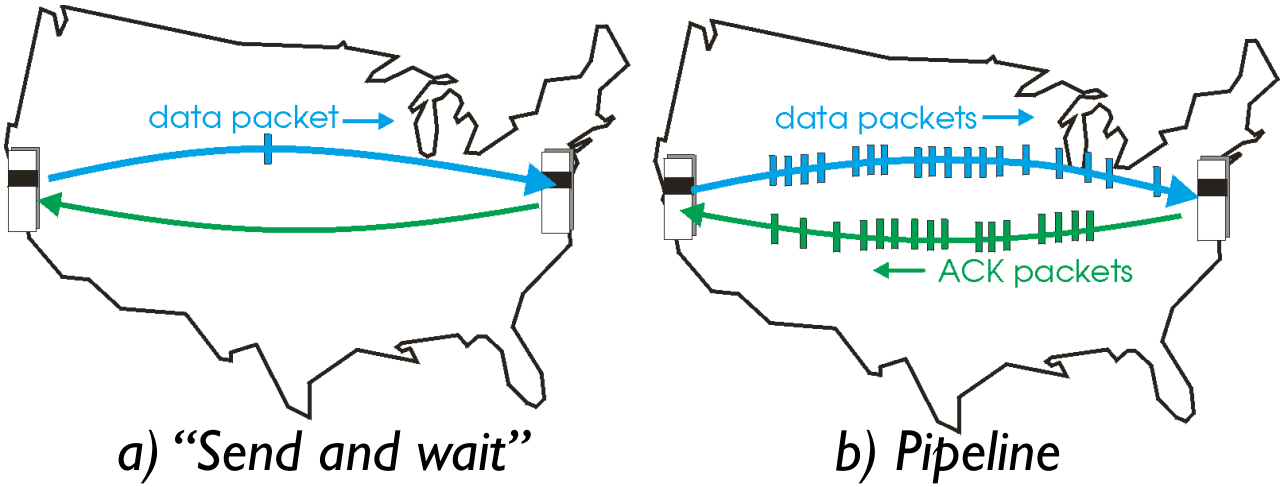
1. somme de contrôle
   - détecter les erreurs
2. accusés de réception (ACK & NAK)
   - boucle de contrôle
3. #séquence
   - détecter duplicata, erreurs sur les ACKs & pipeline (cf suite)
4. "timer" (temporisateur)
   - pertes de paquets

Performances de rdt3.0

- rdt3.0 protocole fonctionnel mais performances ☹
  - à cause de l'approche "send and wait"
- Exemple
  - lien 1Gb/s → C, délai propagation 15ms, paquet 1Ko → L
  - U : utilisation du lien (proportion de temps à émettre) → efficacité
  - D_{utile} : débit utile
  - $t_{émission}$ =
  - U =
  - D_{utile} =
    - A comparer avec les 1 Gbps !
- Fonctionnel mais pas performant

Protocoles à anticipation (1)

- Pipeline
- Expéditeur peut transmettre plusieurs paquets à la suite sans attendre des accusés de réception
- Ce nombre de paquets = fenêtre d'anticipation



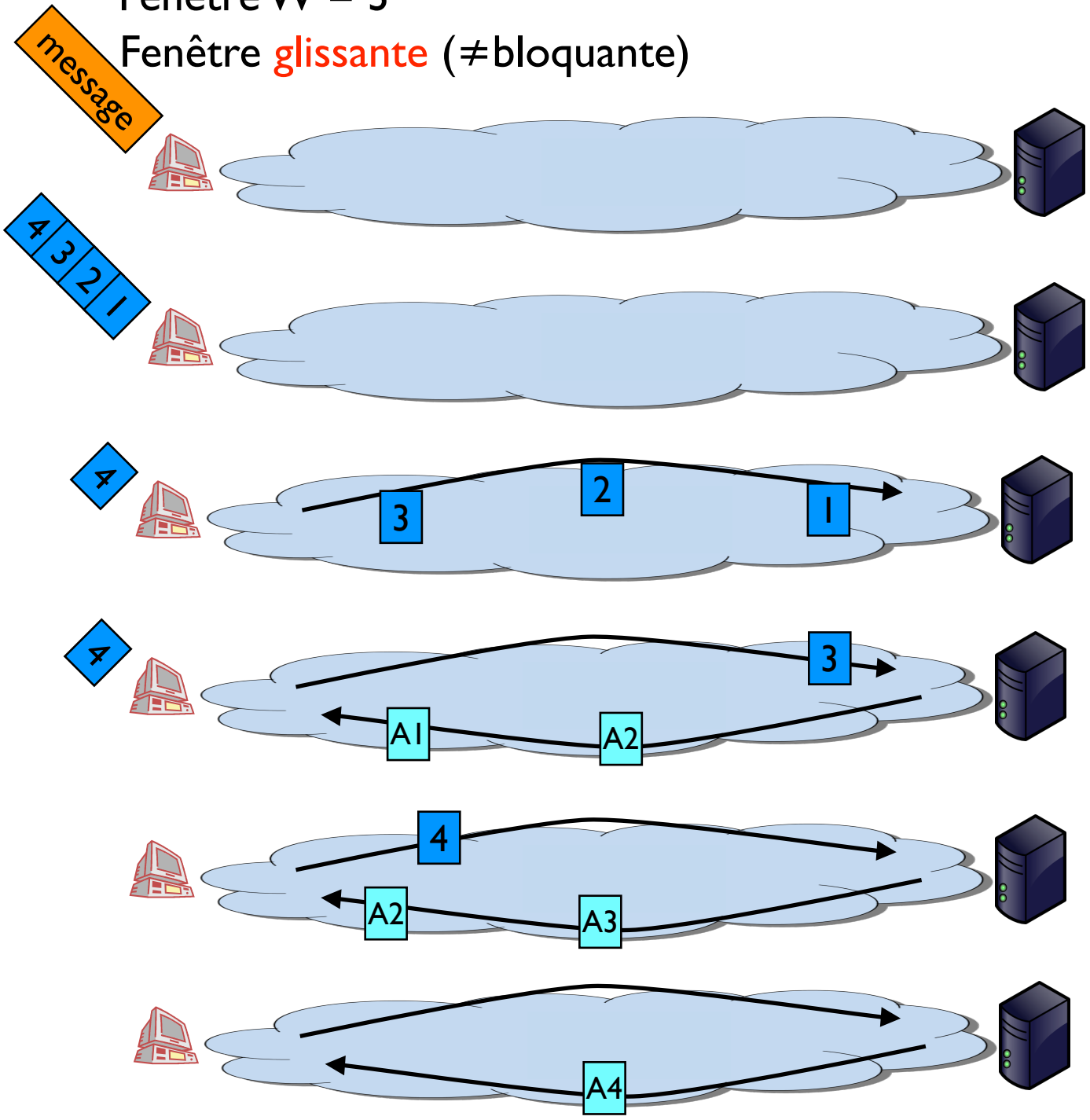
a) "Send and wait" *b) Pipeline*

- Taille supposée de la fenêtre d'anticipation ?

Protocoles à anticipation (2)

Fenêtre W = 3

Fenêtre glissante (≠bloquante)



1. message à envoyer

2. divisé en 4 segments

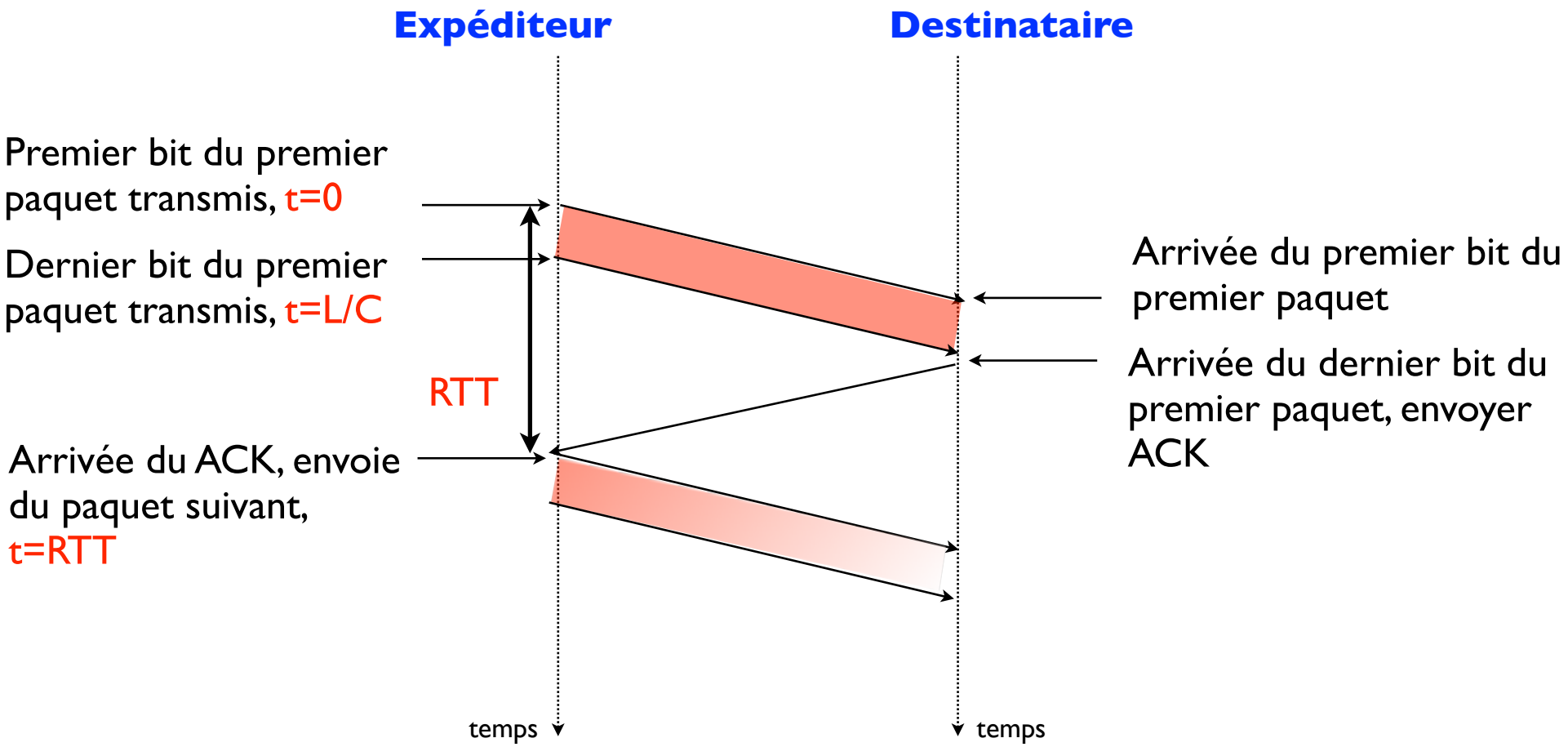
3. la fenêtre bloque le 4ème segment

4. les segments se "transforment" en ACK

5. l'arrivée de ACK 1 déclenche le départ du segment 4

6. le transfert prend fin

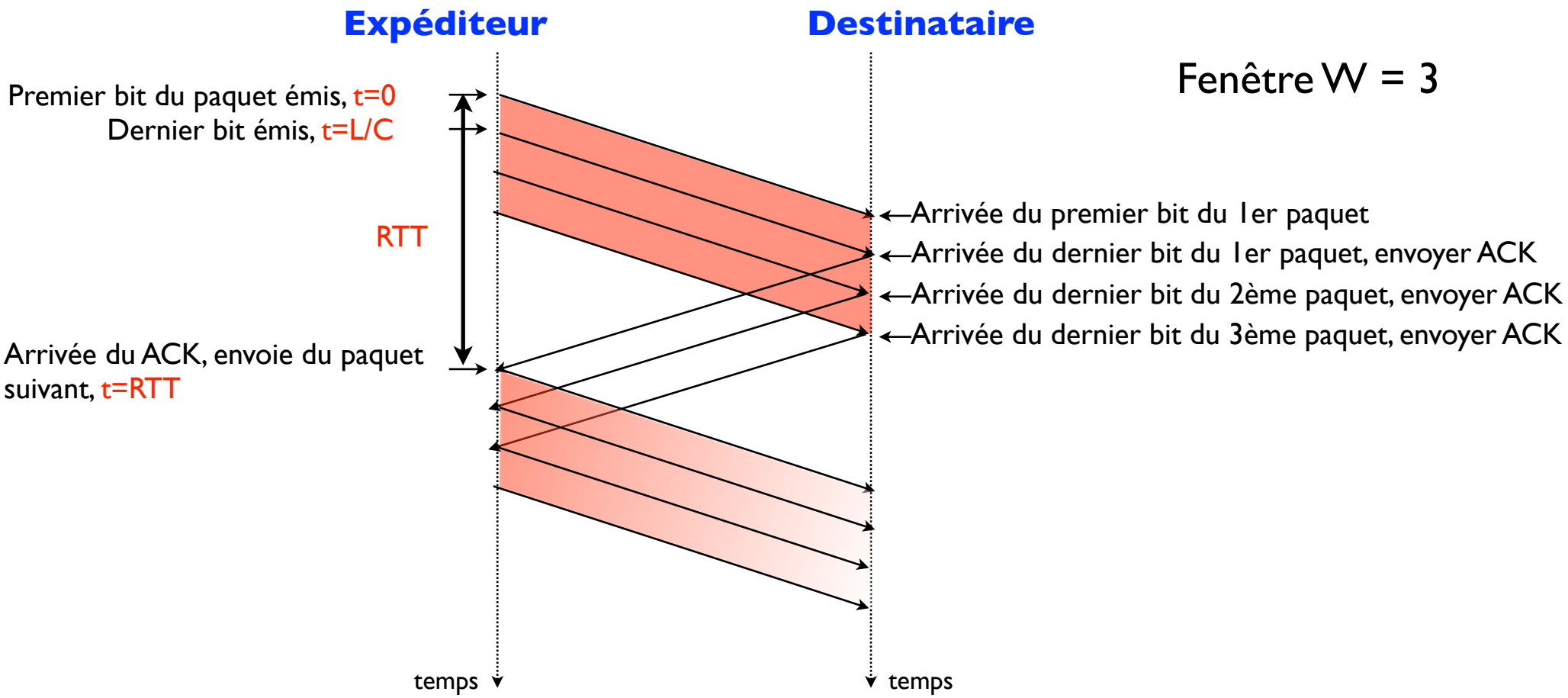
Performances de rdt3.0 (2)



Efficacité de "Send and wait"

U =

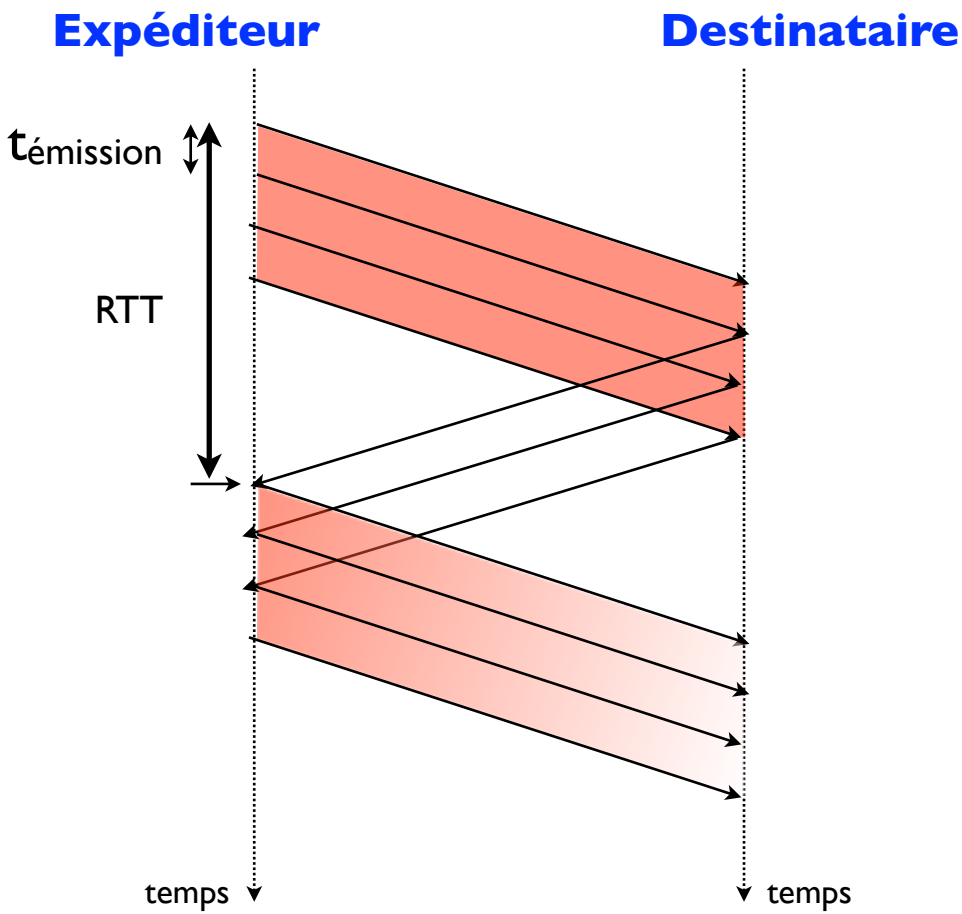
Performances avec fenêtre d'anticipation



Efficacité avec "Pipeline" de taille W

U =

Taille optimale de la fenêtre d'anticipation



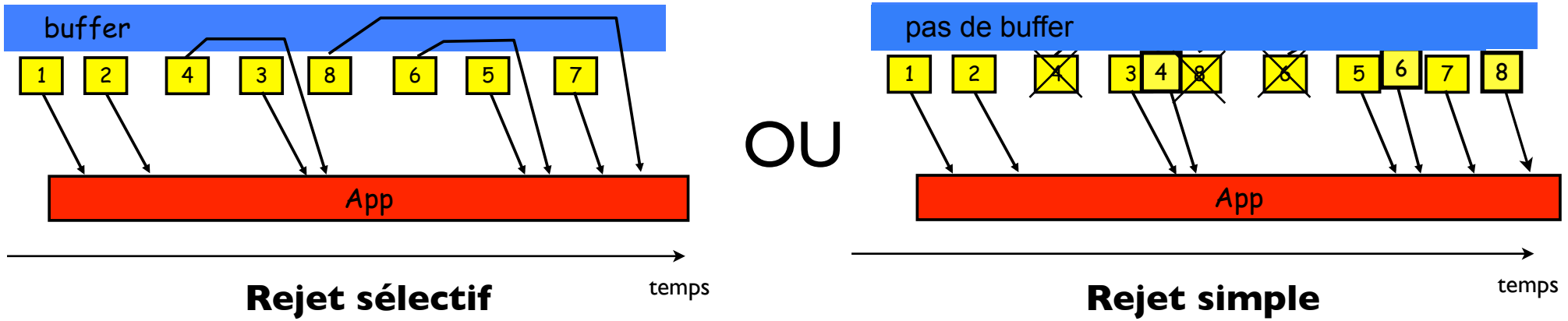
- $t_{émission}$ = temps d'émission d'un paquet
- Taille optimale de la fenêtre telle que U = 1
  - W^* =

Protocoles à anticipation (3)

- Une meilleure utilisation des ressources réseaux 
- Mais ... 
  -
  -
  -
- Buffer émission : nécessaire ?
  -
- Buffer réception : nécessaire ?
  -

Buffer de reséquencement

- Arrivées déséquencées des segments : 1, 2, 4, 3, 8, 6, 5, 7



- Rejet Sélectif ("Selective Repeat")
  - La couche Transport maintient un buffer par connexion TCP
    - les paquets peuvent entrer déséquencés
    - et attendent jusqu'à être transmis dans l'ordre à la couche Application
- Rejet Simple ("Go-Back N")
  - La couche Transport ne maintient pas de buffer
    - Seuls les paquets bien séquencés sont acceptés

Rejet simple ou sélectif ?

- Choix entre rejet simple et sélectif ?
  - Selon la "rareté" des ressources
  - Mémoire rare → rejet simple
  - Réseau difficile (canal peu fiable) → rejet sélectif
- Et TCP, fait-il du rejet simple ou sélectif ?
  - En général il mémorise les paquets déséquencés
  - mais il utilise des ACK cumulatifs

Plan

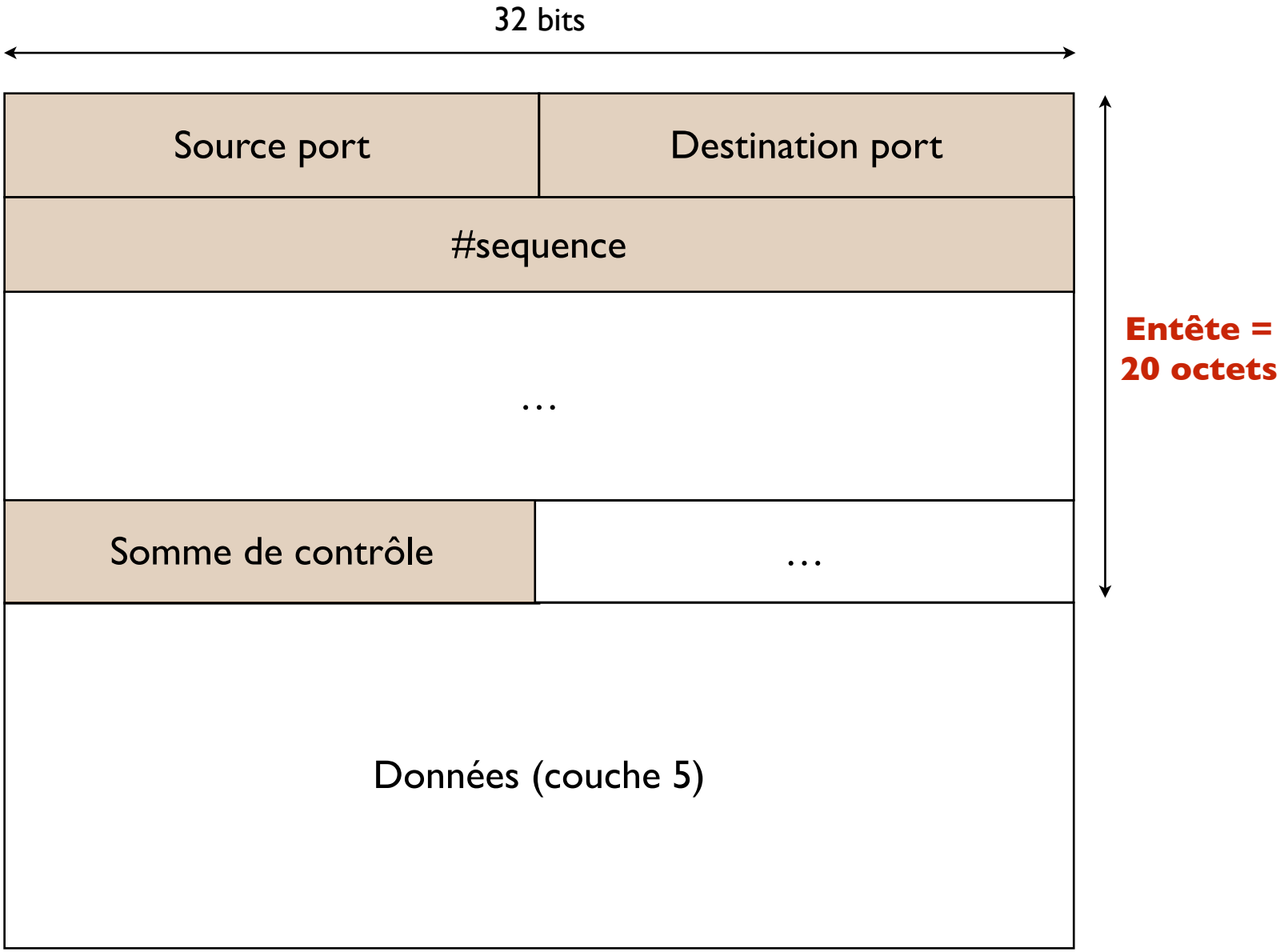
1. Services de la couche Transport
2. Multiplexage et démultiplexage
3. Transport sans connexion : UDP
4. Principes du transfert de données fiable
5. Transport orienté connexion : TCP
6. Principes du contrôle de congestion
7. Contrôle de congestion TCP
8. Limites de TCP & Résumé

Vue d'ensemble de TCP

[RFCs: 793, 1122, 1323, 2018, 2581]

- Orienté connexion
  - échange d'information ("handshaking")
  - expéditeur et destinataire fixent les paramètres du transfert
- Mode duplex
  - les données peuvent circuler dans les deux sens
- Point-à-point
  - entre un expéditeur & un destinataire
- Livraison fiable et séquencée des données
  - buffers d'émission et de réception
  - taille des segments fixée par
    - MSS :"Maximum Segment Size" (hors en-tête)
- Pipeliné
  - la fenêtre d'anticipation est dynamique

Structure simplifiée d'un segment TCP



Due to COVID19 all TCP applications are being converted to UDP to avoid handshake!

Plan

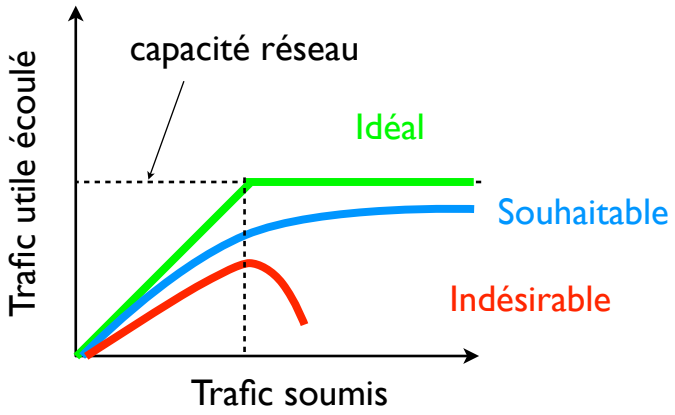
1. Services de la couche Transport
2. Multiplexage et démultiplexage
3. Transport sans connexion : UDP
4. Principes du transfert de données fiable
5. Transport orienté connexion : TCP
6. Principes du contrôle de congestion
7. Contrôle de congestion TCP
8. Limites de TCP & Résumé

Problématique (1)

- Ressources du réseau sont limitées
  - Capacité d'émission des liaisons
  - Capacité de traitement des nœuds
  - Capacité de stockage (buffers) des nœuds
- Lorsque le trafic soumis (charge) est trop important
  - Contention sur les ressources
  - Des files se forment dans les routeurs
  - Retards et pertes de paquets ↗
  - Phénomènes de congestion

Problématique (2)

- Idéalement
  - Si *Trafic utile soumis* < *Capacité réseau*
    - *Trafic utile écoulé* =
  - Sinon (au-delà)
    - *Trafic utile écoulé* =
- En réalité, c'est différent. Pourquoi ?
  -
- Objectif : éviter la zone d'effondrement de performances
- Mais comment ?



Comportements	Causes dégradation	Autres exemples
Idéal		
Souhaitable		
Indésirable		

Plan

1. Services de la couche Transport
2. Multiplexage et démultiplexage
3. Transport sans connexion : UDP
4. Principes du transfert de données fiable
5. Transport orienté connexion : TCP
6. Principes du contrôle de congestion
7. Contrôle de congestion TCP
8. Limites de TCP & Résumé

Contrôle de congestion

- 3 étapes
  1. Comment détecter une congestion ?
  2. Comment réguler le débit d'une source TCP ?
  3. À combien réguler son débit TCP ?

- Les détails peuvent varier selon les versions de TCP
  - mais elles sont compatibles entre elles

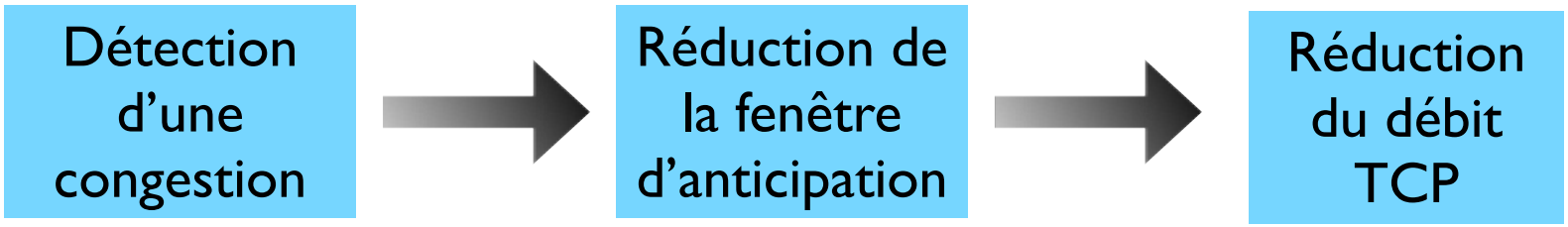
- TCP Tahoe (BSD, 1988)
- TCP Reno (BSD,1990)
- TCP BIC (Kernel Linux jusqu'à 2.6.18, 1988)
- TCP Vegas (1990)
- TCP Scalable (2002)
- TCP New Reno (BSD, 2002)
- HSTCP (2003)
- FAST TCP (2005)
- TCP CUBIC (Kernel Linux depuis 2.6.19, 2005)
- H-TCP (optionnel sur Kernel Linux 2.6)
- Compound TCP (Windows Vista, 2008)

Détecter une congestion

- Comment détecter une congestion ?
  - Sans assistance du réseau
  - Uniquement à partir des terminaux
  - 
    - 
- Hypothèse fondamentale
  - Une congestion provoque des pertes de paquets
  - L'inverse est-il vrai ?
  - 

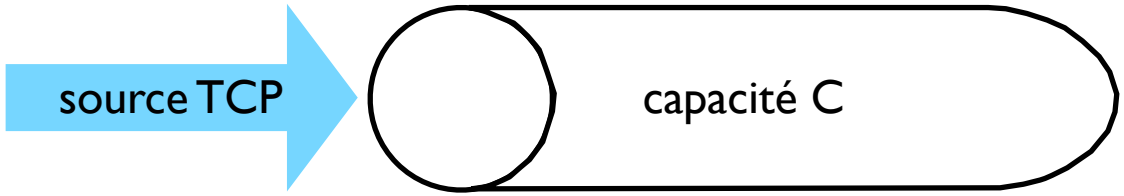
Réguler son débit

- Comment réguler le débit d'émission d'une connexion TCP ?
  - en variant la taille de sa fenêtre d'anticipation

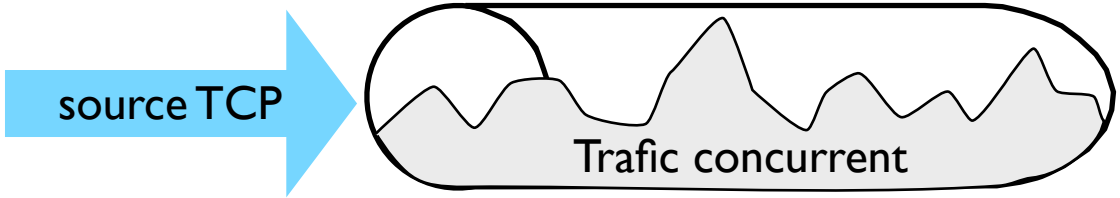


À quel niveau réguler son débit ?

- À combien réguler le débit d'une source TCP ?
- Exemple 1 : un lien dédié de capacité C



  - débit souhaitable pour la source ?
  -
- Exemple 2 : un lien de capacité C partagé à plusieurs



  - débit souhaitable pour la source ?
  -
- Problème : quantité inconnue et dynamique. Comment la découvrir ?

Hausse additive, baisse multiplicative

- Principe : détecter la capacité disponible sur le chemin
  - augmenter progressivement le débit de TCP en agrandissant la taille de sa fenêtre d'anticipation jusqu'à atteindre le débit max supporté
  - comment savoir qu'on a atteint le max ?
- Hausse additive : augmenter la taille de la fenêtre d'anticipation d'1 MSS après chaque RTT
- Baisse multiplicative : diviser la taille de la fenêtre d'anticipation par 2

➡ AIMD : "Additive Increase Multiplicative Decrease"

➡ "Congestion Avoidance"

Résumé

- Principes utilisés dans la couche Transport
  - multiplexage et démultiplexage
  - transfert de données fiable
  - contrôle de flux (M2 seulement)
  - contrôle de congestion
- Principaux protocoles existants dans Internet
  - UDP
  - TCP
    - 95% du volume et 90% des flux sur Internet
    - source : *What TCP/IP Protocol Headers Can Tell Us About the Web*, Smith, F. Donelson, et al., 2001.

Réseaux avancés & Recherche

- M2 SRIV - Réseaux avancés : mécanismes détaillés et performances de TCP
- M2 SRIV - Applications Systèmes & Réseaux : protocoles de transport pour les flux multimédia - RTP / RTCP
- Protocole QUIC
  - basé sur UDP, sécurisé
  - IETF
- Multipath TCP

Fin du Cours
couche Transport