Université de Strasbourg

ECOLE D'INGENIEURS TELECOM PHYSIQUE STRASBOURG

cnrs

TCP/IP

Couche Transport : Retransmissions & TCP

Pascal Mérindol

merindol@unistra.fr
http://dpt-info.u-strasbg.fr/~merindol/

Ce cours est construit sur la base de plusieurs supports pédagogiques parmi lesquels les cours de Jean-Jacques Pansiot, Antoine Gallais et Julien Montavont. L'usage de ce support ne peut être qu'académique.

Plan

- **Problématiques**
- **Retransmissions**
  - stop & wait
  - continue ou sélective
  - Contrôle de flux
- **La couche transport & TCP**
  - généralités
  - formats & ouverture de connexion
  - contrôle de congestion

Vers un transfert fiable

- **Assurer l'intégrité des données échangées**
- **Niveaux de fiabilité**
  - application critique (vie ou mort) : contrôle d'un avion, une centrale nucléaire, ...
  - application «assez critique» (argent) : données bancaires, ...
  - application peu critique (discussions) : forums, blog, ...

- **Un support physique est-il fiable / exempt d'erreur ?**
  - erreurs au décodage liées à des problèmes :
    - de bruit, interférences, etc.
    - de synchronisation.

Comprendre et réparer une erreur

- **Taux d'Erreur Binaire = BER pour Bit Error Rate**
  - #bits érronnés par quantité de bits échangés
    - 0110100110 -> 0110**01**01**0**0 => BER de 3/10 (BER pour un LAN ~ 10^{-9})
  - erreurs indépendantes ou en rafale

- **Erreur de contenu**
  - détection / correction par **redondance**
  - taux d'erreur résiduel << BER ~ acceptable pour le protocole supérieur

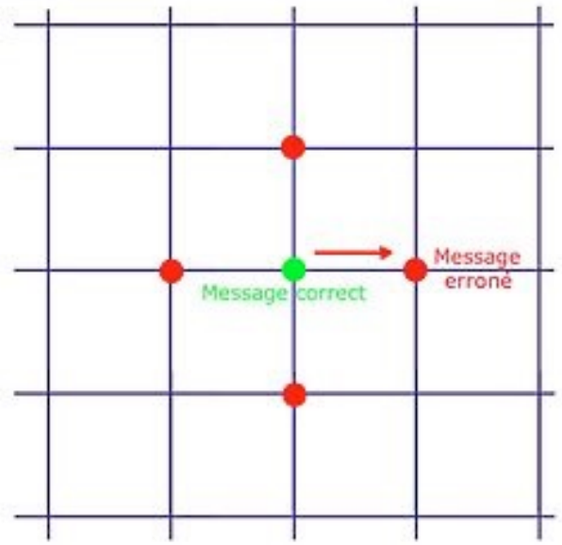
- **Erreurs «sur arrivées» ~ erreurs dans l'échange**
  - pertes : congestion ou rejet du paquet par erreur sur la contenu
  - déséquencements : acquittements, numérotation, timers => **retransmission** !

Fonctionnement général

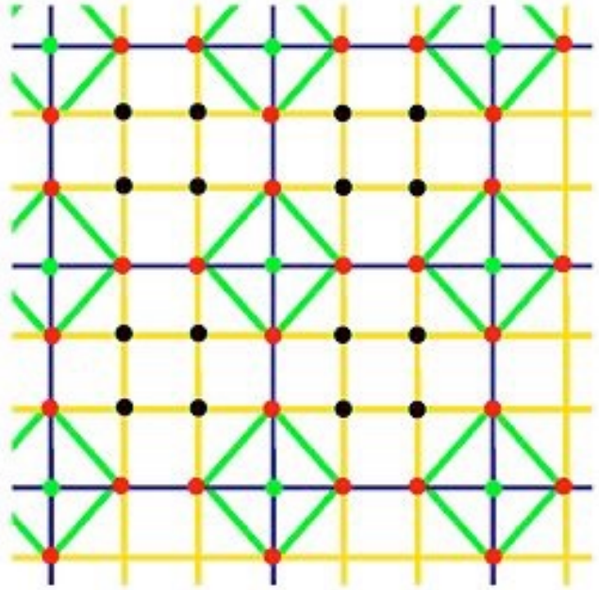
- **Où se produisent les erreurs et qui les contrôle ?**
  - Problèmes de transmission
    - Erreur sur contenu due au canal => traité dans la couche liaison et au dessus
  - Problème de pertes sur congestion
    - buffer des routeurs limité => traité par couche réseau (signalisation) ou TCP
    - Les erreurs de contenu peuvent provoquer des pertes traitées aux niveaux supérieurs

- **De la détection à la retransmission**
  - => erreur détectée et trame éliminée à la couche liaison (via CRC par ex.)
  - => paquet IP contenu dans la trame perdu (couche réseau)
  - => segment TCP contenu dans le paquet IP perdu
    - Retransmissions assurées par la couche transport

Contrôles d'erreurs sur contenu



Code sans redondance


Code correcteur

Redondance : ajouter de l'information de contrôle

- **Codes détecteurs et éventuellement correcteurs d'erreurs**
  - ajout de bits de contrôle (redondance) aux bits d'information
  - Rendement du code = #bits d'information / #bits total
  - compromis entre rendement du codage et taux d'erreurs résiduel
- **Premiers exemples simples :**
  - codes à répétition
    - on envoie x fois le bloc d'information
    - rendement : 1/x
  - bit de parité
    - on ajoute à chaque symbole un bit tel que la somme de tous les bits modulo 2 soit 0 (code à parité paire) ou 1 (code à parité impaire)
    - rendement : |symbole-1|/|symbole|

Somme de contrôle (checksum) : exemple simple

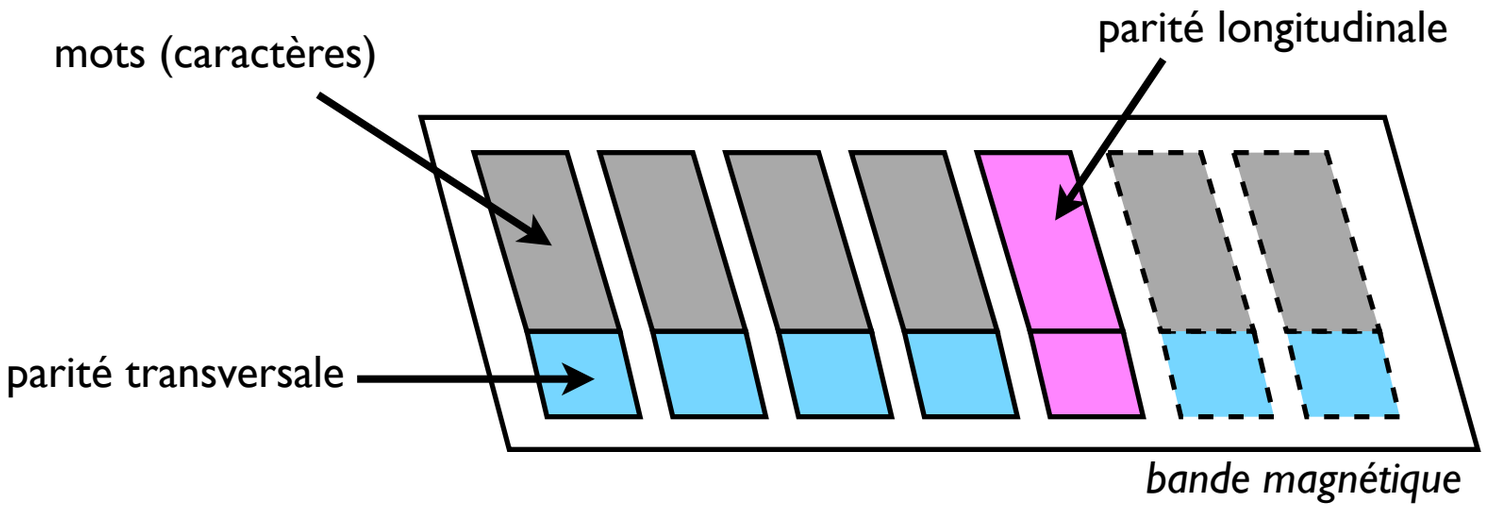
- **Calculer une somme de contrôle encodé sur n bits**
- **S = Addition des valeurs des blocs de m bits, Checksum = S modulo 2^n**
  - Exemple : « émettre : 011010100101010101010010 »
  - Checksum par bloc d'octet : m = 8 bits
  - 3 blocs : [01101010] [01010101] [01010010]
    - 01101010 en base 2 = 106 en base 10
    - 01010101 en base 2 = 85 en base 10
    - 01010010 en base 2 = 82 en base 10
  - S = 106 + 85 + 82 = 273, et soit n = 8
  - Checksum = 273 modulo 2^8 = 17 : soit 00010001 en base 2
  - à transmettre : 011010100101010101010010**00010001**

Compromis rendement/pouvoir

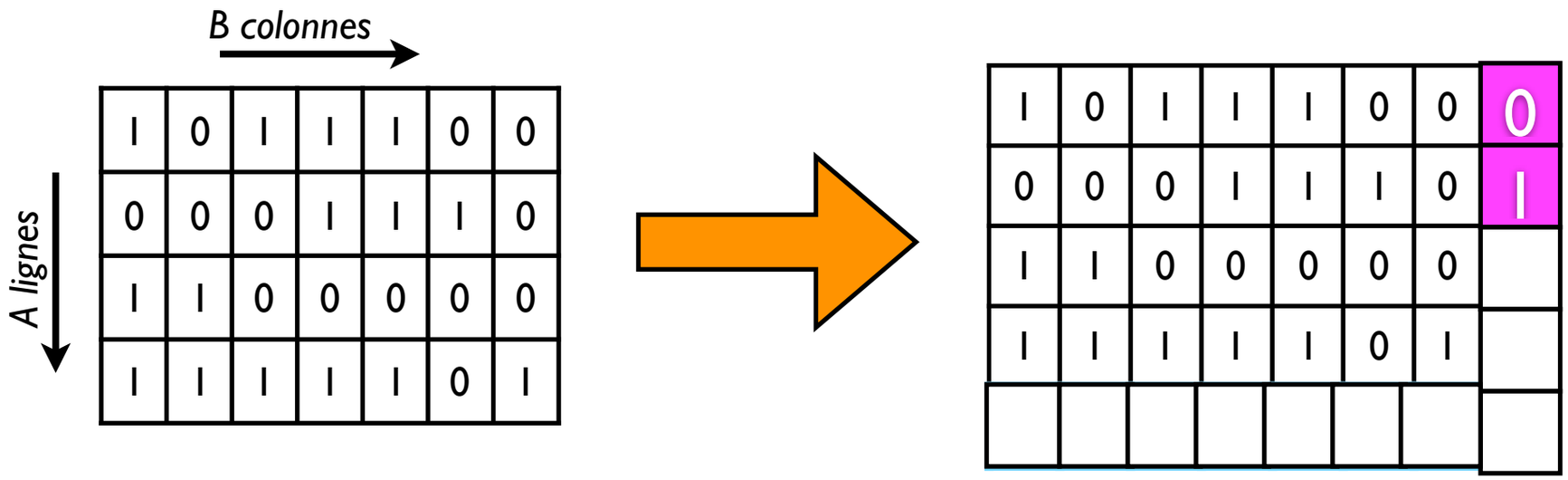
- **Exemple : Parité longitudinale et transversale**
  - on complète le bit de parité par caractère en réalisant un contrôle de parité longitudinal sur l'ensemble des caractères
  - utilisé pour les bandes magnétiques
  - permet de corriger une erreur ou de détecter jusqu'à trois erreurs

- **Recherche de codes à bon rendement et haut pouvoir...**
  - ...détecteur (voire correcteur !)
    - =>la théorie des codes
    - la facilité de codage et décodage par matériel/logiciel est un critère important !
      - décodage en temps réel des données reçues

Parité transversale + longitudinale



Envoi de la suite : 10111000001110110000011111101



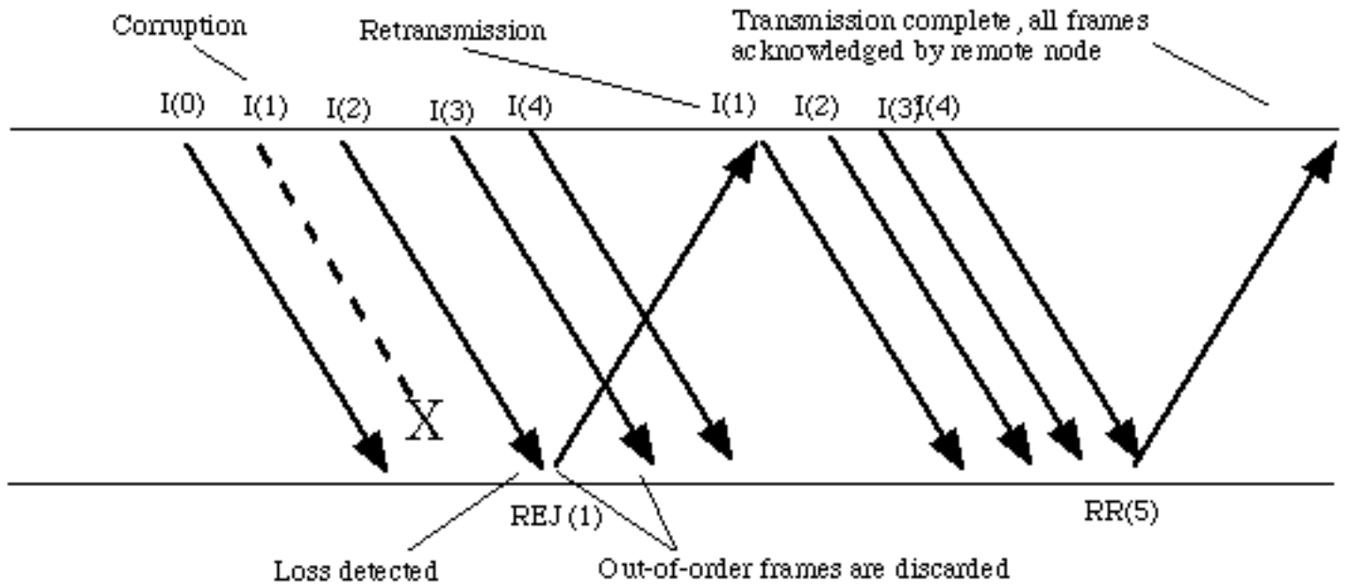
rendement ?

$$\frac{A.B}{(A+1).(B+1)} = 0{,}7$$

Code détecteur et correcteur

Il s'agit juste d'un rapide aperçu très naif…
à vous de creuser la théorie des codes (linéraires, polynomiaux, fontaines, etc)

Retransmission

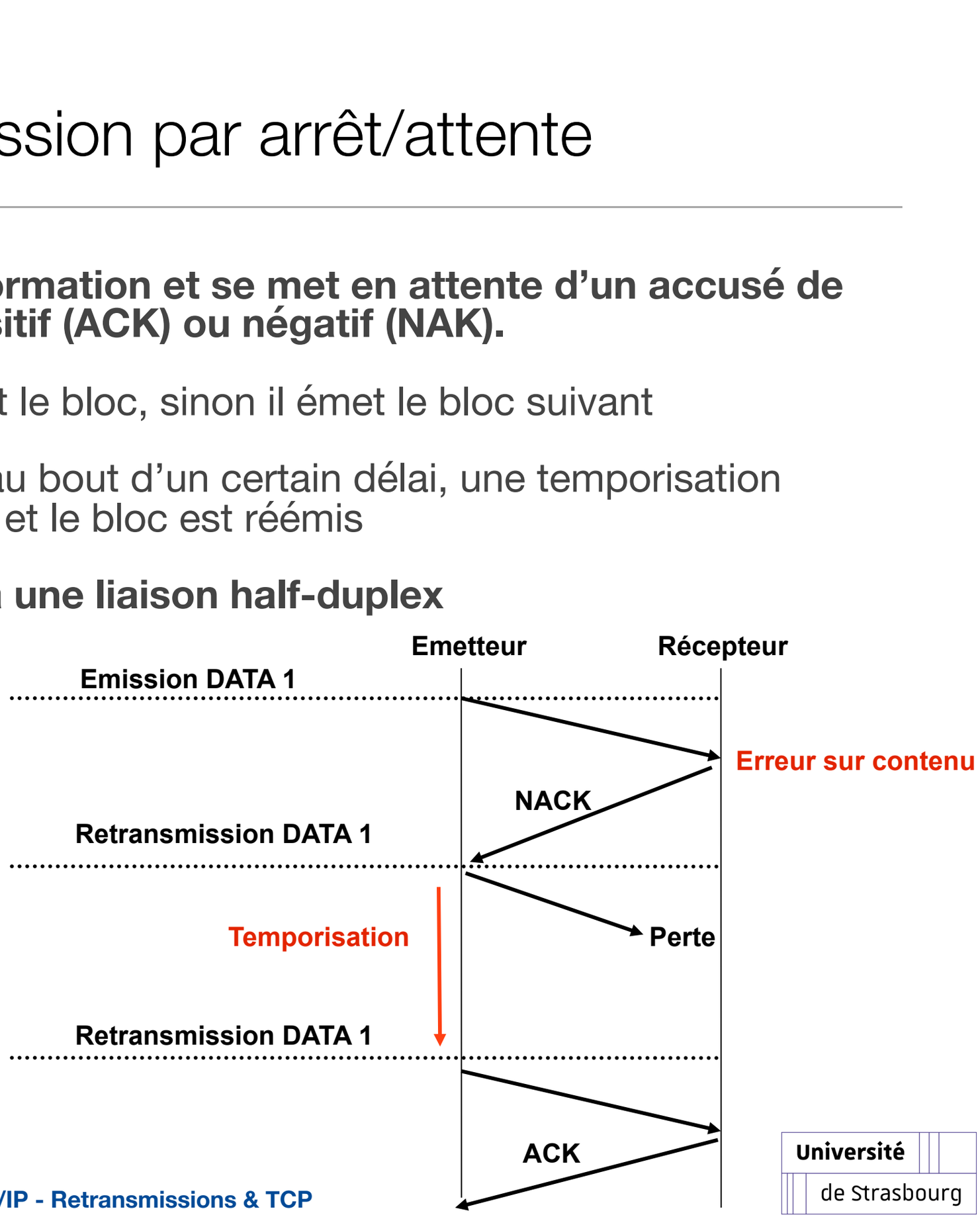


Erreurs «sur arrivée»

- **Limites des codes détecteurs/correcteurs**
  - les erreurs sur contenu ne sont pas toutes corrigées
  - les codes correcteurs à haut pouvoir correctif sont coûteux en codage/décodage
    - => limite des solutions purement FEC (**Forward Error Correction**)
- **Technique de retransmission**
  - ARQ **Automatic Repeat reQuest**
  - mise en oeuvre par un protocole assurant la fiabilité
    - protocole de liaison, de transport (TCP), …
  - permet la correction d'une erreur « sur contenu » détectée (ex. CRC invalide) mais non corrigée = assimilée à une perte
    - une erreur « sur arrivée » (ex. déséquencement, trou dans la numérotation)
      - Ex: perte due à une congestion dans Internet

Stop & Wait : retransmission par arrêt/attente

- **L'émetteur envoie un bloc d'information et se met en attente d'un accusé de réception (ou acquittement) positif (ACK) ou négatif (NAK).**
  - Si l'accusé est négatif il réémet le bloc, sinon il émet le bloc suivant
  - Si l'acquittement n'arrive pas au bout d'un certain délai, une temporisation arrive à échéance (« timeout ») et le bloc est réémis
- **méthode simple, bien adaptée à une liaison half-duplex**
- **Performance ?**

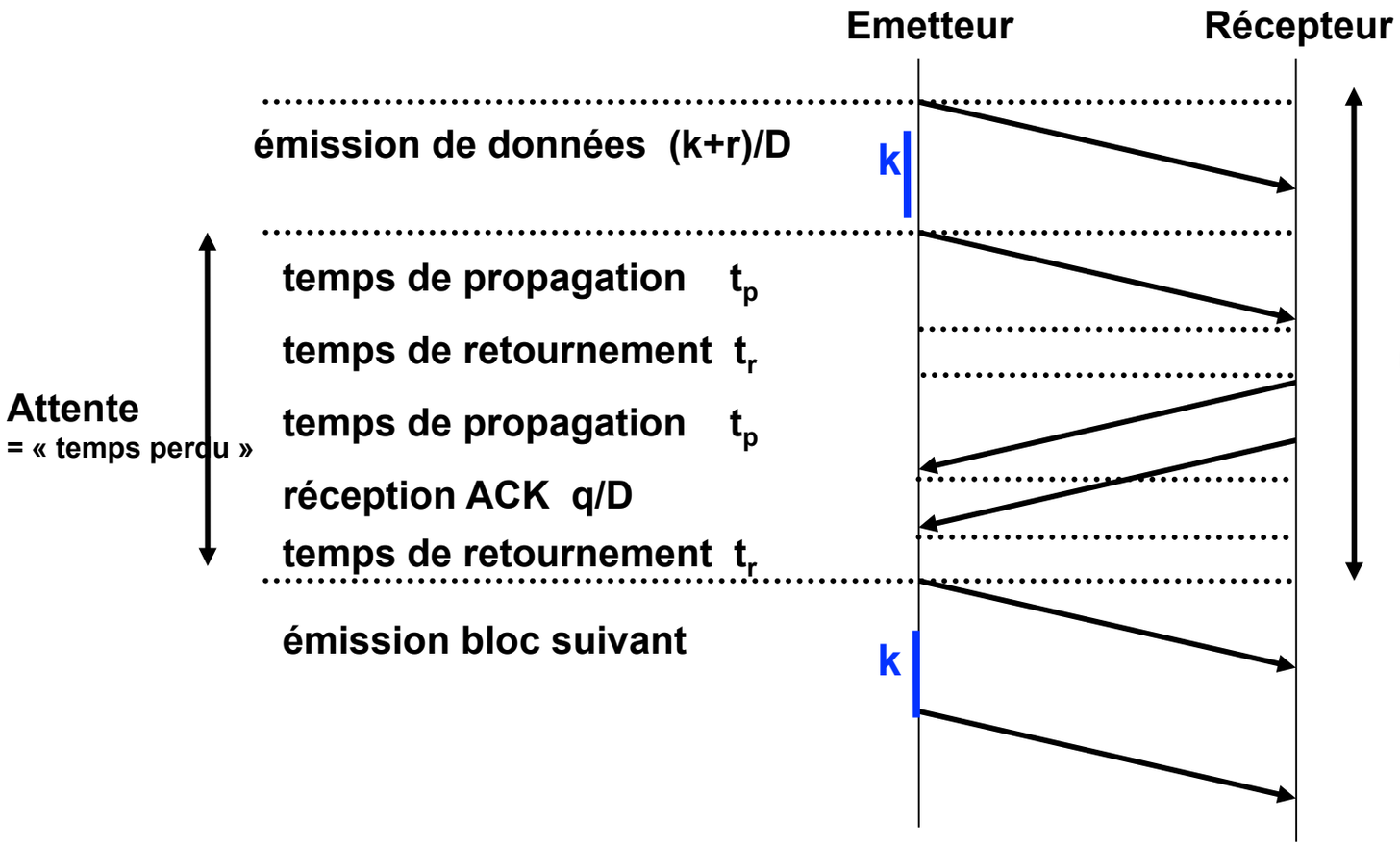


Stop & Wait : analyse

- **Calcul du rendement obtenu par un transfert d'information entre deux stations**
  - D : débit binaire de la liaison
  - k : nombre de bits d'information utile d'un bloc
  - r : nombre de bits d'entête (code détecteur, numérotation, …)
  - T_e : taux d'erreurs par bit (supposées indépendantes)
  - t_p : temps de propagation du signal entre l'Emetteur et Recepteur
  - t_r : temps de retournement des modems et calcul (souvent négligeable)
  - q : taille de l'accusé de réception (en général q << k+r)

- **En l'absence d'erreur, la durée de transmission d'un bloc est de**
  - $\theta = (k + r + q) / D + 2. (t_p + t_r)$
    - débit efficace $d = k / \theta$; rendement $= k / \theta . D = k / ((k + r + q) + 2. (t_p + t_r) .D)$

Schéma Temporel Stop & Wait : analyse sans pertes



Attention : ici on prend la durée d'émission (k+r)/D en compte sur l'illustration

En l'absence d'erreur, le rendement est de : d / D = k / (θ . D) (diminue avec t_p)

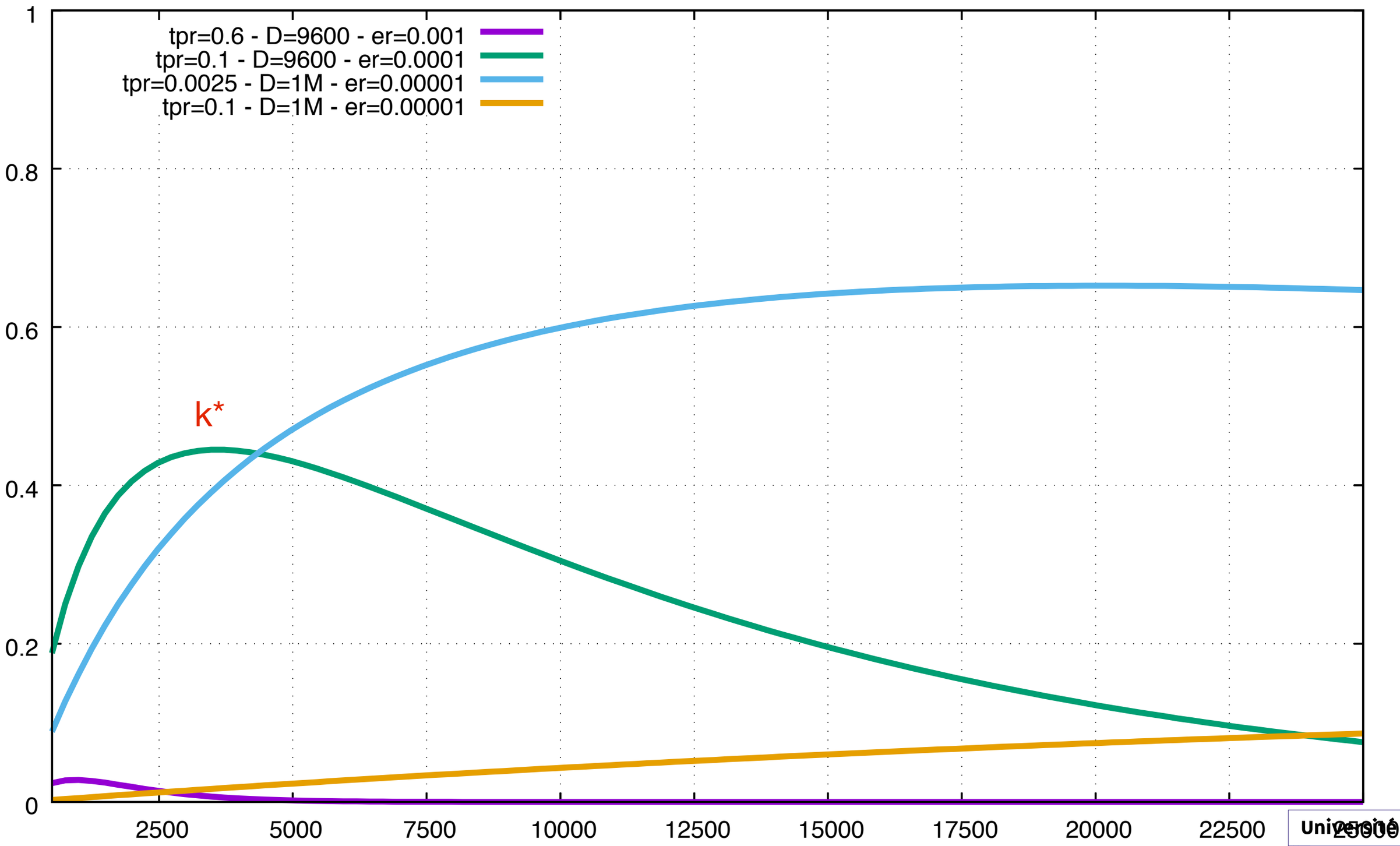
Stop & Wait : avec erreurs

- **Hypothèses :**
  - erreurs indépendantes et toutes détectées
  - pas d'erreur sur les acquittements
- **Probabilité de transmission d'un bloc sans erreur :**
  - $P_c = (1 - T_e)^{k+r}$
- **Probabilité qu'un bloc soit correctement transmis au i^{ème} essai - erreurs des i-1preds**
  - $(1 - P_c)^{i-1} . P_c$
    - → probabilité d'avoir une durée de transmission de i. θ
- **Durée moyenne θ_m nécessaire au transfert correct d'un bloc :**
  - $\theta_m = \Sigma_{i \geq 1}\ i. \theta. P_c. (1 - P_c)^{i-1} = \theta / P_c = \theta / (1 - T_e)^{k+r}$

Stop & Wait : avec erreurs

- **Débit efficace moyen :**
  - $d = k / \theta_m = (k / \theta). (1 - T_e)^{k+r}$
- **Le débit efficace varie avec la longueur des blocs**
  - à partir d'une certaine valeur de T_e ($T_e > 0$), il existe une valeur k* représentant la taille des blocs à partir de laquelle le débit efficace diminue
- **Exemple**
  - trame HDLC (protocole de liaison) r = 48, q = 48
  - Courbe du rendement suivant k avec
    - $T = 2. (t_p + t_r) = 0.2s, 1.2s, 0.005s$
    - D = 9600b/s et 1Mb/s
    - $T_e = 10^{-3}, 10^{-4}, 10^{-5}$

Rendement avec erreurs : Stop & Wait / k (+ T,D,T_e)



Stop and Wait : perte d'un ACK ?

- **Les acquittements peuvent se perdre, le lien peut être coupé**
- **⇒ retransmission sur délai de garde (timeout)**

Algorithme Emetteur	Algorithme Récepteur
1. envoyer B	réception B′
2. armer T	si B′ correct (checksum)
3. si réception ACK	envoyer ACK, lire
4. goto bloc suivant	sinon
5. sinon si réception NAK	envoyer NAK
6. recommencer	
7. sinon si déclenchement délai T	
8. recommencer	

- **Problème : si perte de ACK ⇒ Récepteur doit distinguer ré-émission et nouveauté**

Bit alterné : gestion perte ACK

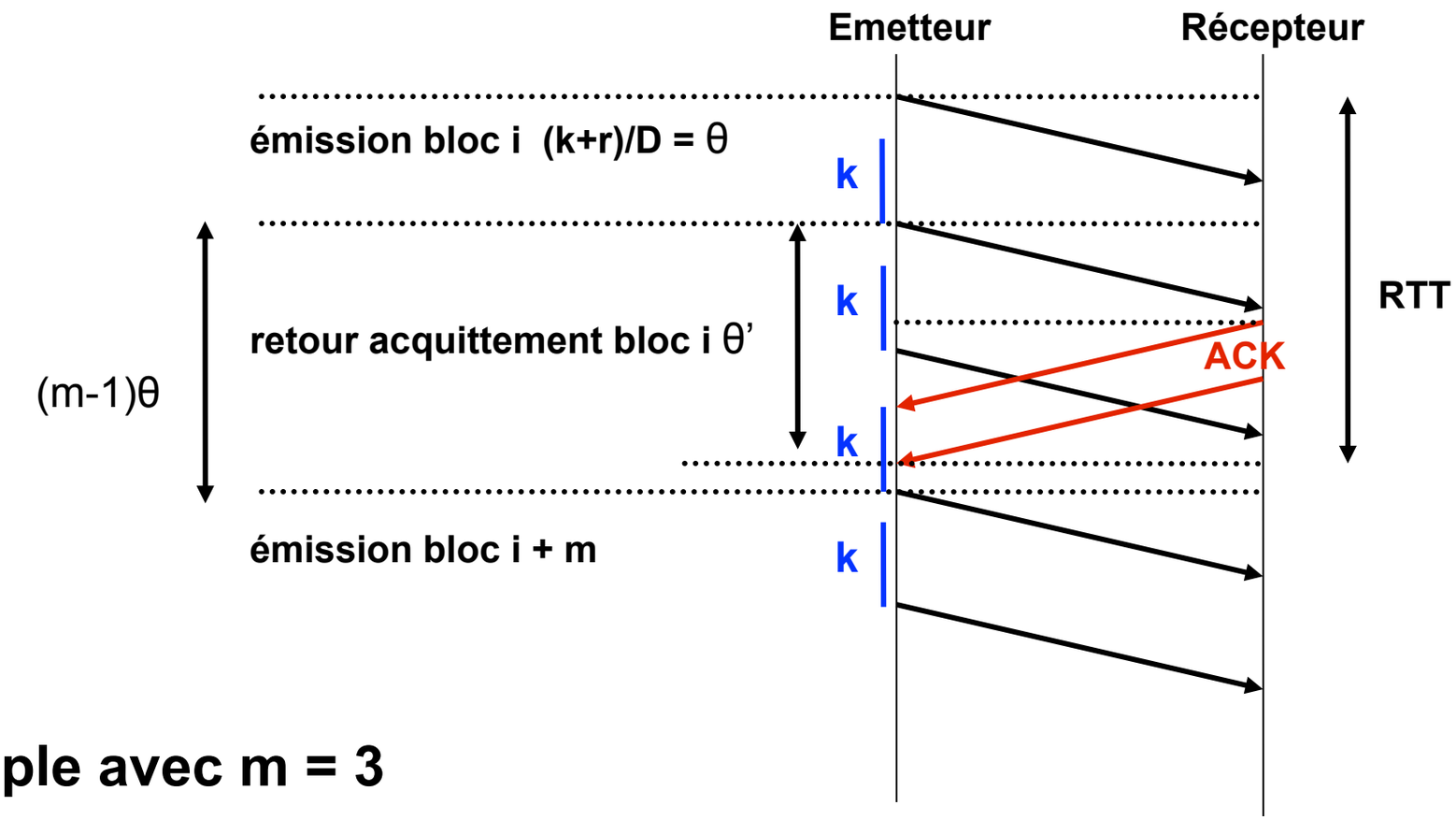
```
          // Initialement i = 0 , des deux côtés

   Algorithme Emetteur                    Algorithme Récepteur
1. envoyer B,i                            si réception B',i
2. armer T                                     si B'correct
3. si réception ACK                               envoyer ACK et lire
4.    i := ¬i                                     i := ¬i
5.    goto bloc suivant                        sinon
6. sinon si réception NAK                         envoyer NAK
7. recommencer                          sinon si réception B', ¬i
8. sinon si déclenchement délai T                 envoyer ACK
9.    recommencer
```

Quand le récepteur reçoit un bloc dupliqué B', ¬i (/ à l'attente du récepteur), il l'acquitte pour débloquer l'émetteur mais ne le « lit » pas (car déjà lu)

Retransmission continue ou sélective

- **Mécanisme d'anticipation**
  - Lorsque les sites sont éloignés, le temps de propagation n'est pas négligeable => la source envoie plusieurs blocs de données avant de se bloquer dans l'attente d'un acquittement => anticipation à l'émission
  - Si le degré d'**anticipation *m*** est suffisant, en l'absence d'erreur, il y a transmission des blocs sans interruption dans un sens. Cela se fait simultanément avec le retour des accusés de réception dans l'autre sens => *nécessite une* ***liaison full duplex***
  - Durée de transmission d'un bloc :
    - $\theta = (k + r) / D$
  - L'acquittement arrive au bout d'un temps (en négligeant t_r) :
    - $\theta' = q / D + 2.\ t_p$
  - Pour qu'il n'y ait pas d'attente entre les blocs :
    - $\theta' \leq (m - 1).\ \theta$

Anticipation en Emission : schéma temporel



Exemple avec m = 3

- **On ne perd plus de temps à attendre les ACKs**

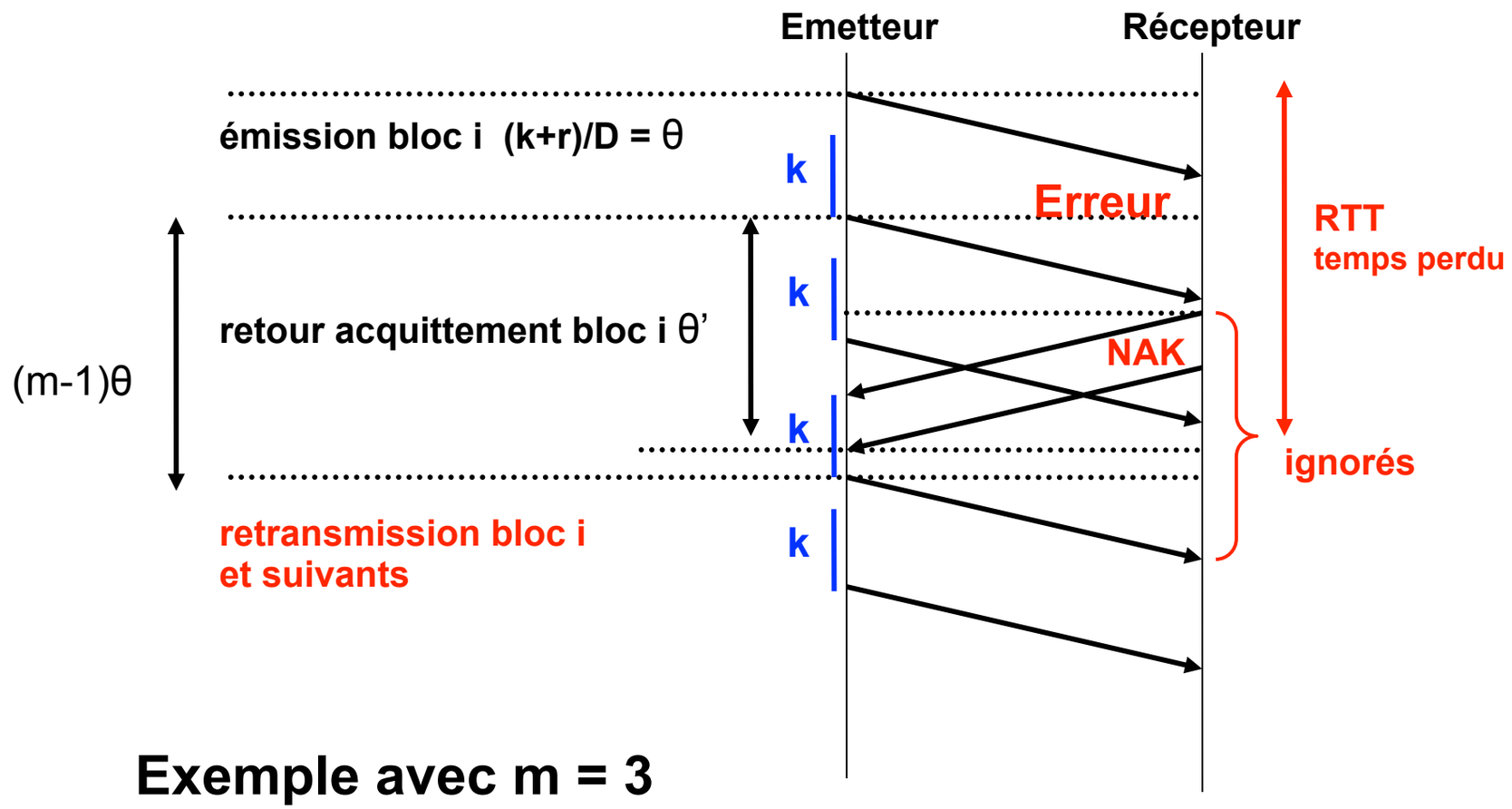
Anticipation : Mise en oeuvre

- **La source dispose d'une séquence de numéros de bloc à émettre appelée fenêtre d'émission (FE). Les numéros dans la fenêtre d'émission indiquent les blocs émis et non encore acquittés**
- **Dès que la source reçoit un acquittement correspondant au premier numéro de la fenêtre, celle-ci est décalée d'une position**
  - fenêtre glissante ou *sliding window*
- **Le puits de données possède une séquence de numéros de bloc qu'il peut recevoir, appelée fenêtre de réception (FR). Si le puits reçoit un bloc dont le n° $\notin$ FR, alors le bloc est refusé (ignoré)**
- **Dès que le puits reçoit le bloc dont le numéro est le premier n° de FR, celle-ci est décalée d'une position**
- **La taille de FR dépend de la façon dont le puits accepte les données**
  - dans l'ordre : |FR| = 1
  - ou le désordre |FR| > 1

Go back N : Evaluation retransmission continue

- **La retransmission est reprise à partir du bloc erroné**
- **Le puits accepte les données uniquement dans l'ordre :**
  - | FR | = 1 : pas d'anticipation en réception
- **Taille de FE optimale : m ≥ ⌈1 + θ' / θ ⌉**
- **S'il se produit une erreur de transmission alors la durée pour transférer correctement le bloc est (m + 1).θ**
- **Durée moyenne θ_m nécessaire au transfert correct d'un bloc :**
  - $\theta_m = \Sigma_{i \geq 0} (i.m + 1). \theta. P_c. (1 - P_c)^i = \theta . (1 + m. (1 - P_c) / P_c)$
- **Débit efficace :**
  - $d = k / \theta_m = k (D / (k + r)) (1 + m. (1 - P_c) / P_c)^{-1}$

Anticipation en émission : schéma temporel avec une erreur

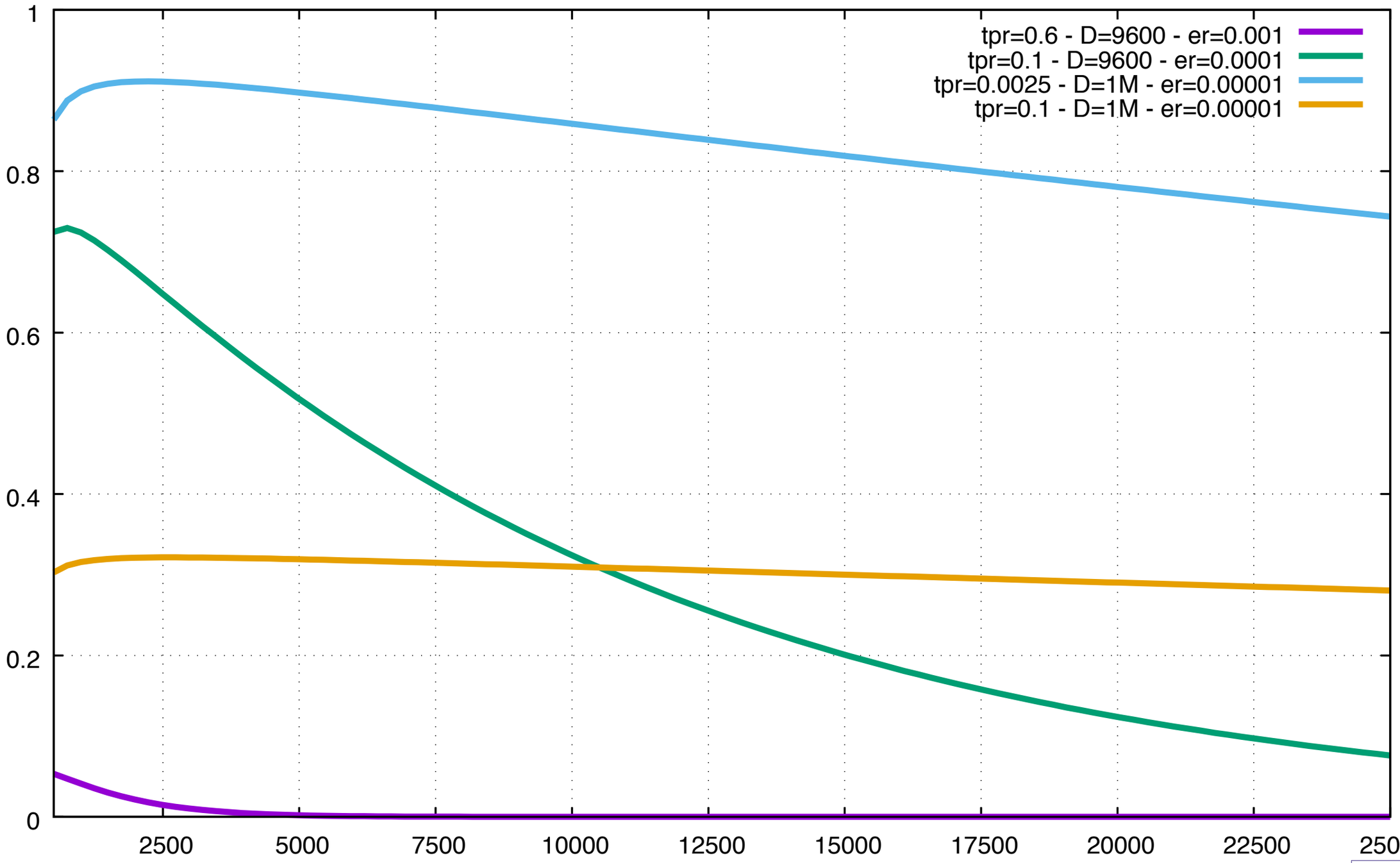


Exemple avec m = 3

Go Back N : Rendement

- **Le rendement R = d/D vaut**
  - $(k/(k+r)) * P_c /(P_c + m * (1 - P_c)) // P_c = (1-T_e)^{k+r}$
- **1er facteur tend vers 1 quand k → ∞**
- **$m = 1 + \lceil (2\ t_p * D + q) / (k + r) \rceil$**
  - m tend vers 2 quand k → ∞
    - pour k petit m ~ D. t_p
- **2ème facteur tend donc vers 0 quand k → ∞**
- **Plus D. t_p est grand, plus une erreur vaut cher : « Long Fat Pipe »**
- **Taille Fenêtre Emission**
  - m trames ≥ 2 t_p * D bits
- **Le récepteur est simple : accepte une seule trame et la livre (pas de tampon)**
- **TCP de base (sans SACK) voisin de ce modèle**

Rendement avec erreurs : Go Back N / k (+ T,D,T_e)

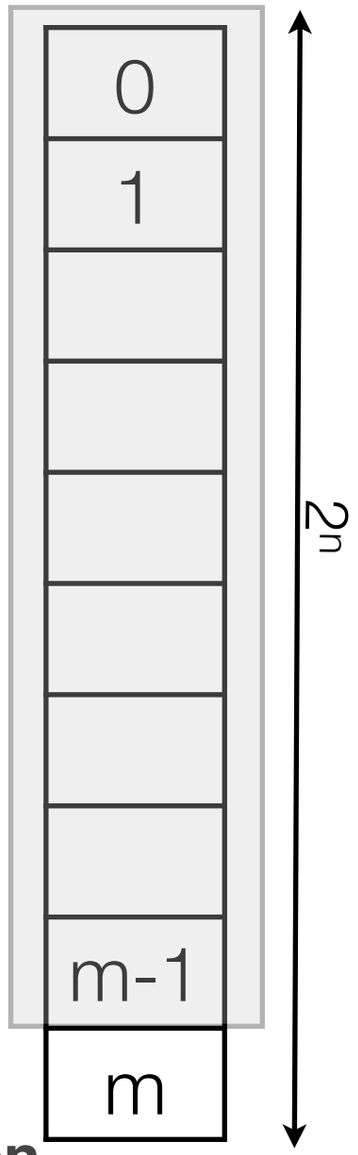


Retransmission continue : numérotation

- **Généralement blocs numérotés modulo 2^n (n bits)**
  - bit alterné n = 1
  - Relation entre m = | FE | , | FR | = 1 et le modulo pour la numérotation des données
  - Supposons m ≥ 2^n *: moins de numéro que la taille de FE*
  - émetteur envoie blocs 0 à 2^n -1, impossible de distinguer les deux scénarios suivants :
    - scénario 1
      - les 2^n blocs se perdent => émetteur renvoie bloc 0 sur timeout => récepteur accepte bloc 0 et renvoie ACK0
    - scénario 2
      - les 2^n blocs arrivent, sont lus et acquittés par récepteur mais les 2^n ACK se perdent => émetteur renvoie bloc 0 sur timeout
        => **récepteur accepte bloc 0 à la place du bloc 2^n** et renvoie ACK0
    - => bloc dupliqué

Retransmission continue : numérotation…jusqu'à ?

- **Le problème est qu'à un instant donné le récepteur peut attendre**
  - un des blocs de FE (y compris le premier)
  - le bloc qui suit FE
    - => m + 1 blocs différents
  - Les numéros de ces m+1 blocs doivent être ≠
  - Si $m < 2^n$
    - le récepteur attend un bloc b i.e : $i < b < i + 2^n$ ($i \in FR$)
      - avec $i \in FE$ et $|FE| < 2^n$ => pas de duplication

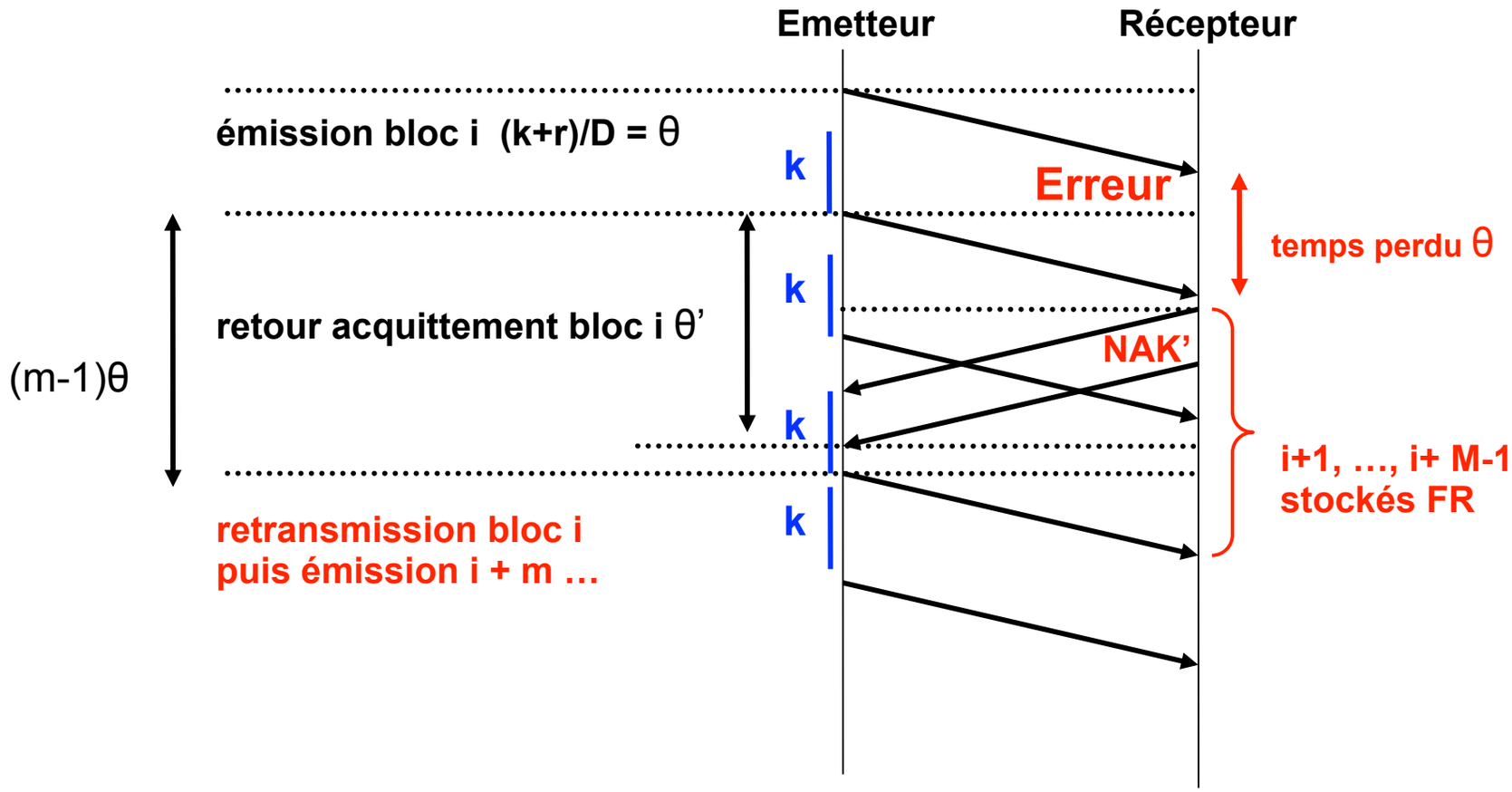


- **Go Back N : pas de duplication ssi $2^n > m$**
  - au minimum : $2^n = m+1$ => **$n = \log_2(m+1)$ bits pour éviter la duplication**

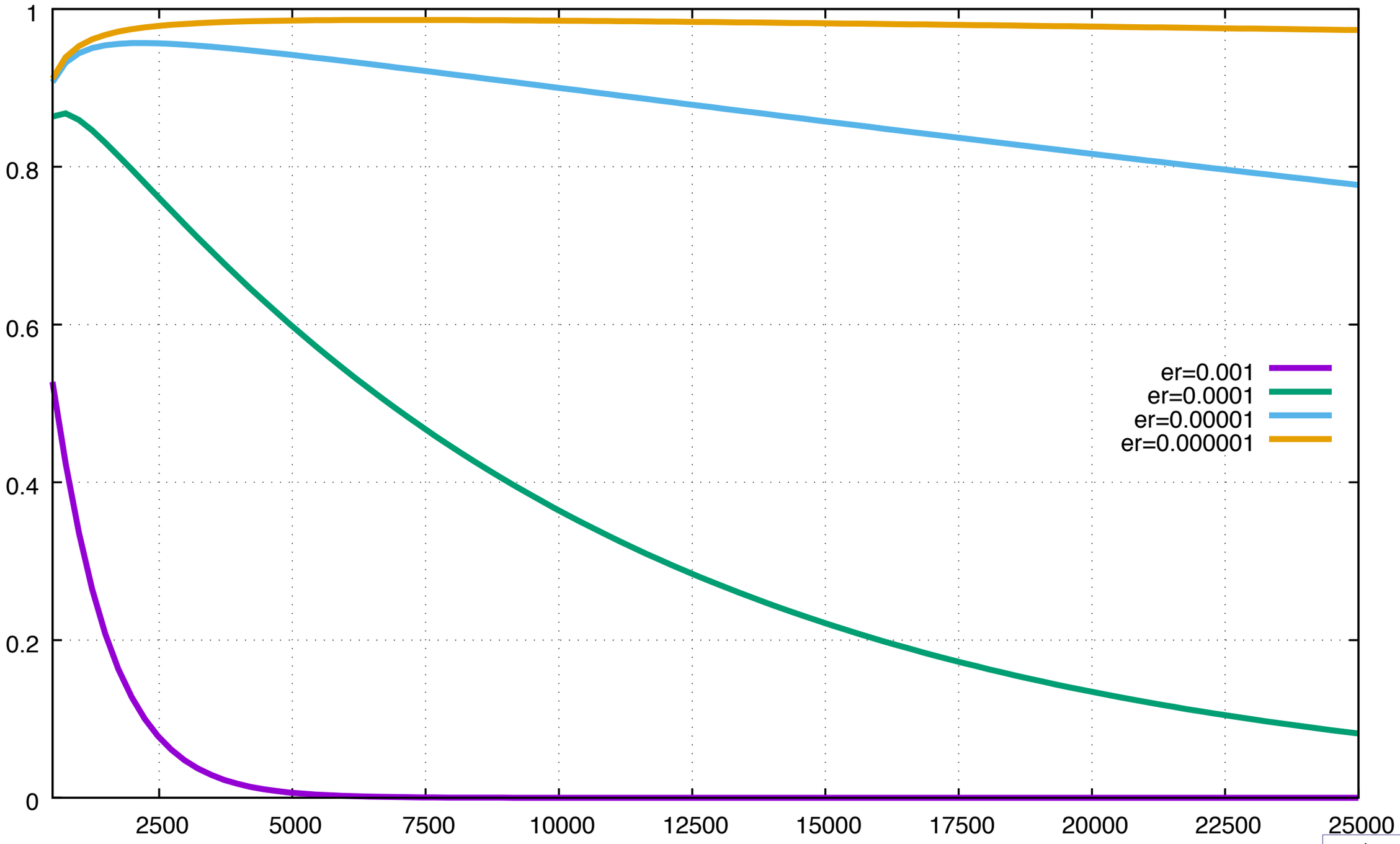
Retransmission sélective

- **En cas d'erreur : NAK' (SREJ avec HDLC ou SACK avec TCP) ou timeout**
  - la source ne retransmet que le bloc erroné
  - le puits stocke les données dans le désordre
    - tant que les blocs ∈ FR : |FR| > 1
    - *a priori* livraison dans l'ordre à la couche supérieure (ex : TCP)
  - Durée moyenne d'émission d'un bloc : $\theta_m = \theta / P_c$ (idem retransmission avec stop&wait)
    - durée d'émission d'un bloc : $\theta = (k + r) / D$ (idem retransmission continue)
  - transmission continue=pdt l'attente des ACK d'autres blocs sont transmis et acceptés
    - FE et FR «assez» grands
    - débit efficace : $d = k / \theta_m = D \,.\, P_c \,.\, k / (k + r)$; rendement : $d / D = P_c \,.\, k / (k + r)$
- **Méthode efficace *même si* D et t_p élevés** : d/D indépendant de D et de t_p
- **Nécessite une mémoire suffisante coté récepteur : |FR|>1**

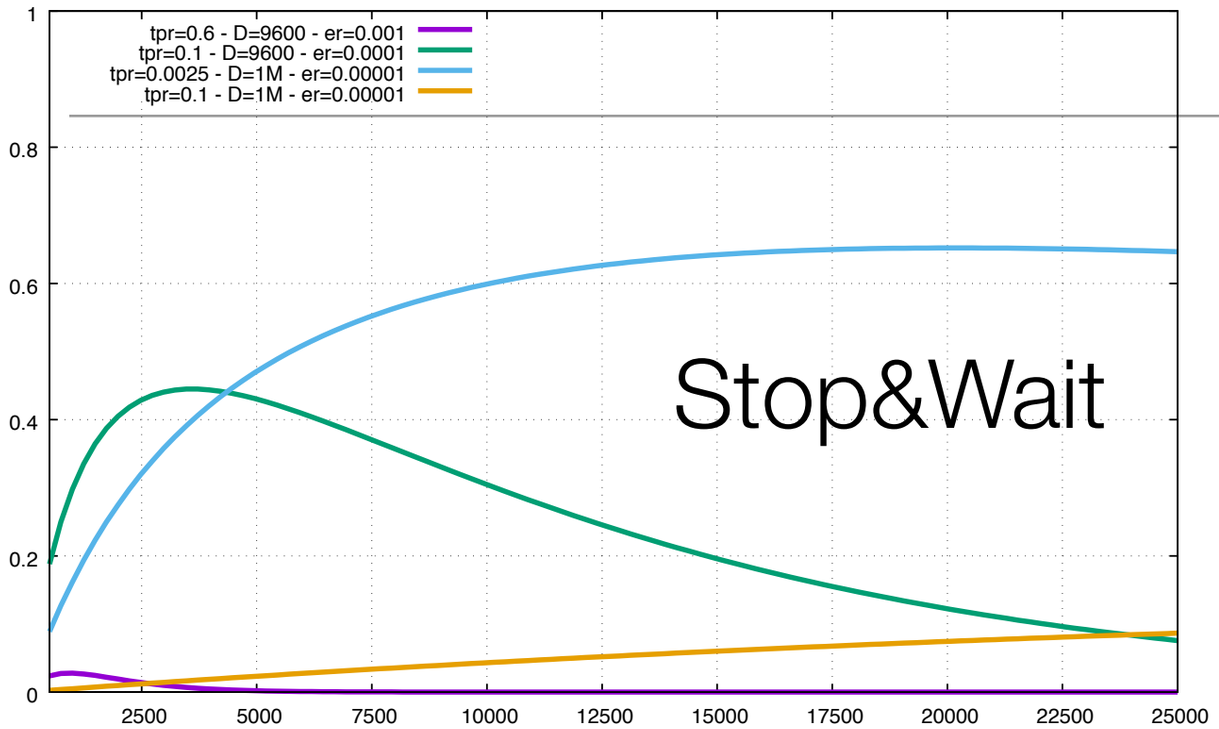
Retransmission sélective : schéma temporel avec une erreur

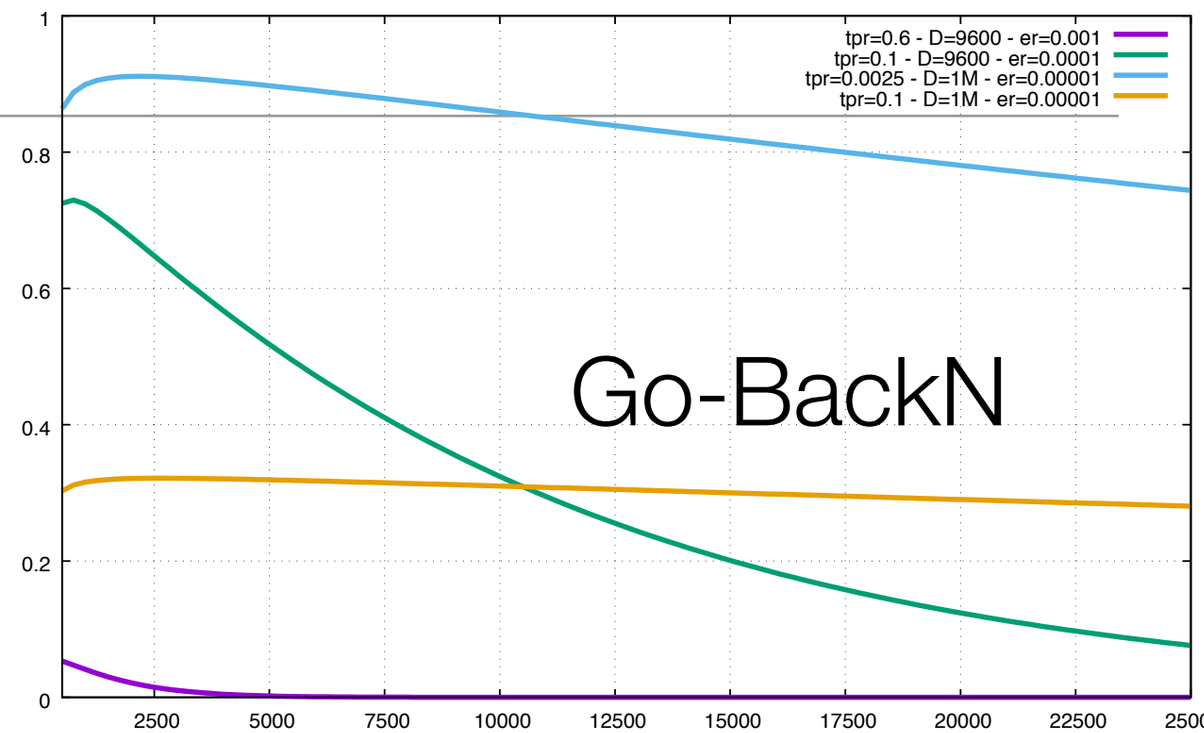


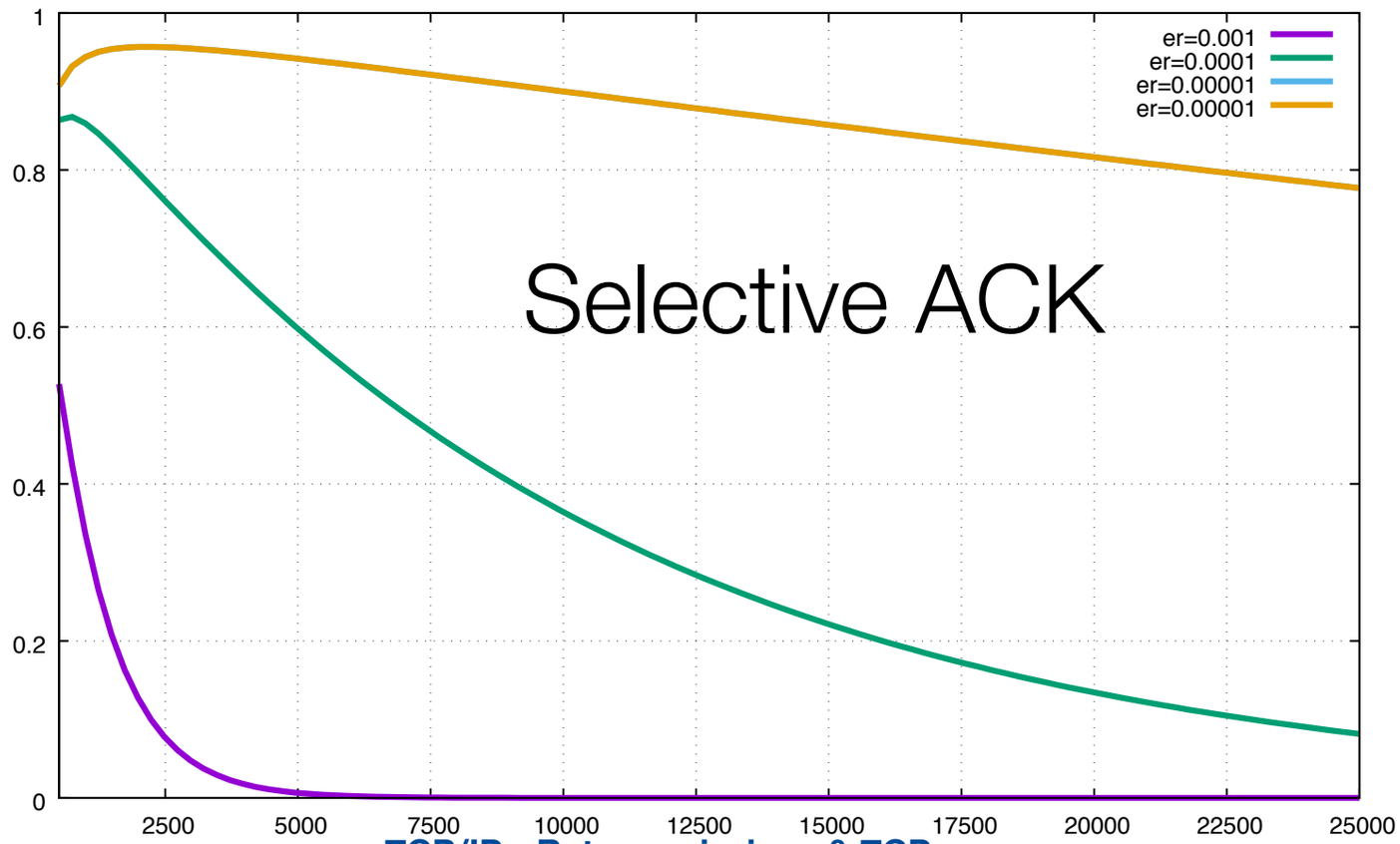
Rendement avec erreurs : retransmission sélective / k (+ T_e)



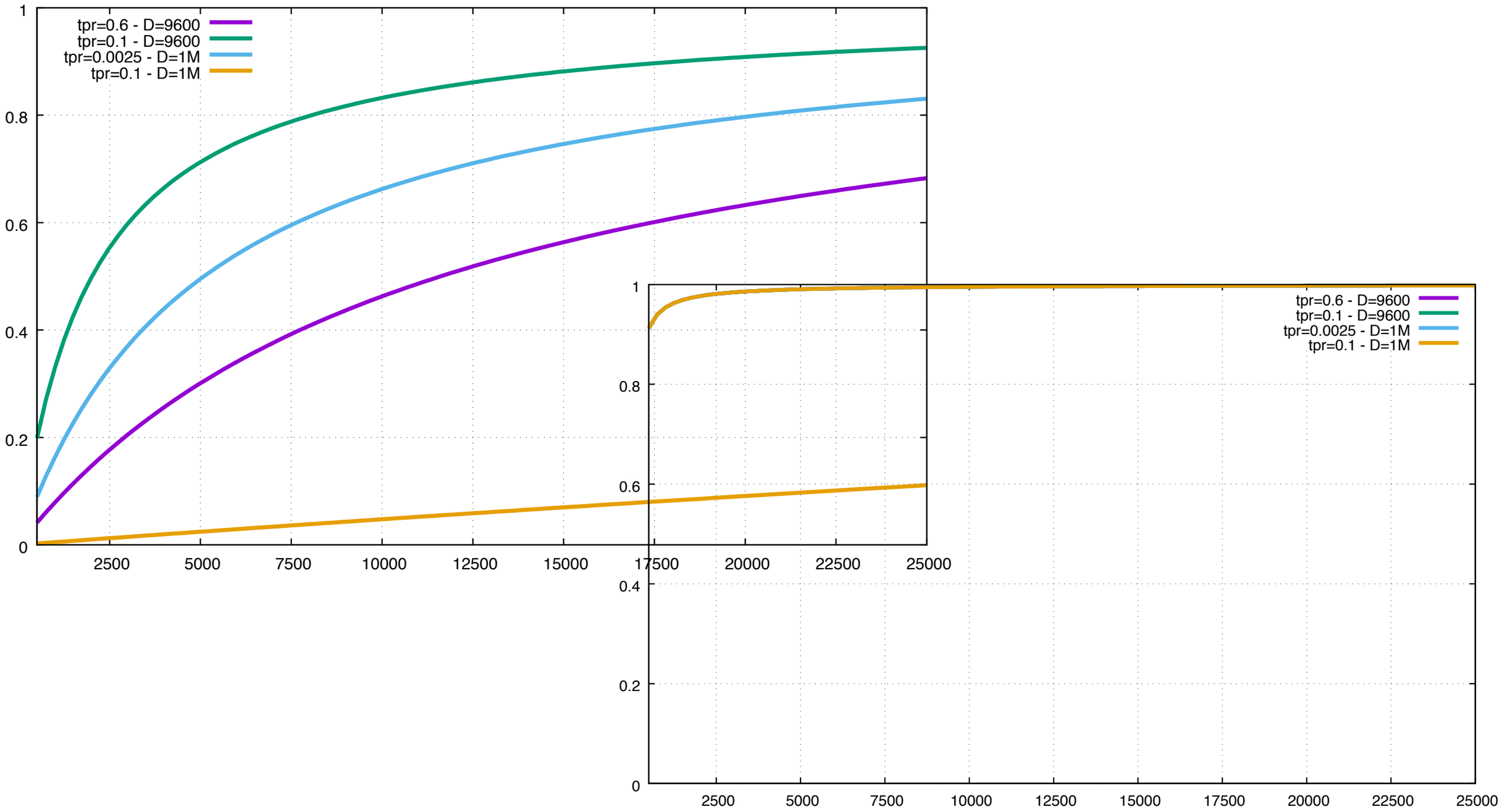
Comparaisons (avec erreurs)







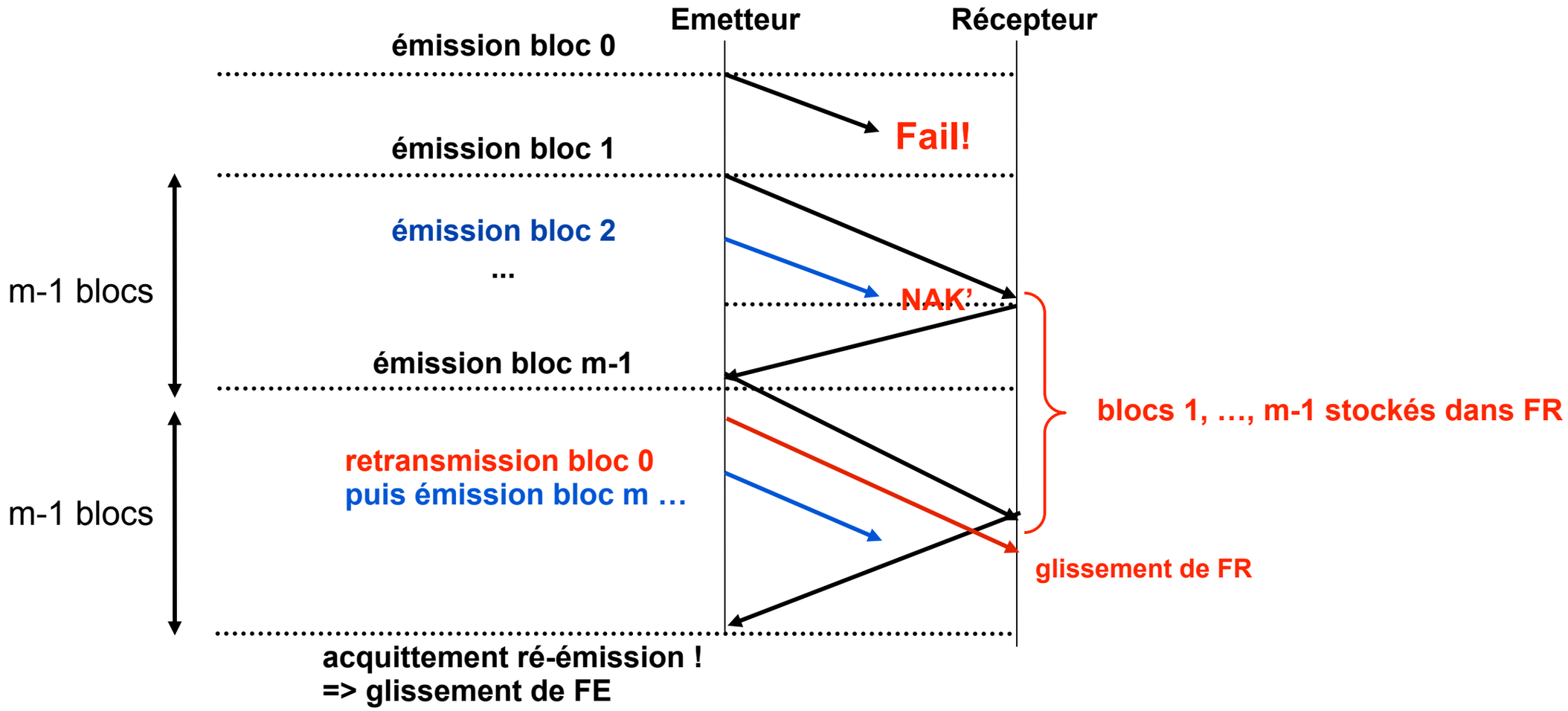
Comparaisons…et sans erreurs ?



Retransmission sélective : taille de fenêtre

- **Pour avoir rendement maximal**
- **En l'absence d'erreur, il suffit que (idem Go Back N)**
  - $|FE| = m = 1 + \lceil (2\ t_p * D + q) / (k + r) \rceil$
  - et $|FR| = 1$: car blocs arrivent dans l'ordre
- **Si une erreur/perte isolée**
  - $|FE| = 2m - 1$: émission en attendant acquittement ré-émission
  - $|FR| = m$: accepter les blocs qui suivent l'erreur
- **Si deuxième erreur/perte sur le même bloc (déjà retransmis)**
  - $|FE| = 3m - 2$ et $|FR| = 2m - 1$
- **Si T_e élevé et blocs longs : retransmissions multiples**
  - => rendement élevé seulement si |fenêtres| (buffers) de grande taille

Taille des fenêtres : schéma temporel avec une erreur



Retransmission sélective : numérotation

- **Numérotation modulo 2^n (n bits)**
- **m taille de FE et l taille de FR**
- **Supposons m + l > 2^n**
  - émetteur envoie blocs 0 à m - 1
  - m blocs bien reçus et acceptés
  - Récepteur attend blocs m , … , m + l -1 : attend numéros m, …, 2^n -1, 0, …, (m + l - 1) (mod 2^n)
  - m acquittements se perdent
  - timeout coté émetteur : il renvoie les m blocs 0, … , m - 1
  - récepteur reçoit bloc 0 et l'accepte comme bloc 2^n
- **=> bloc dupliqué alors que pour l'émetteur même symptôme que perte de m blocs**
- **Pas de duplication ssi m + l ≤ 2^n**

Contrôle de flux : régulation du débit (adapter au récepteur)

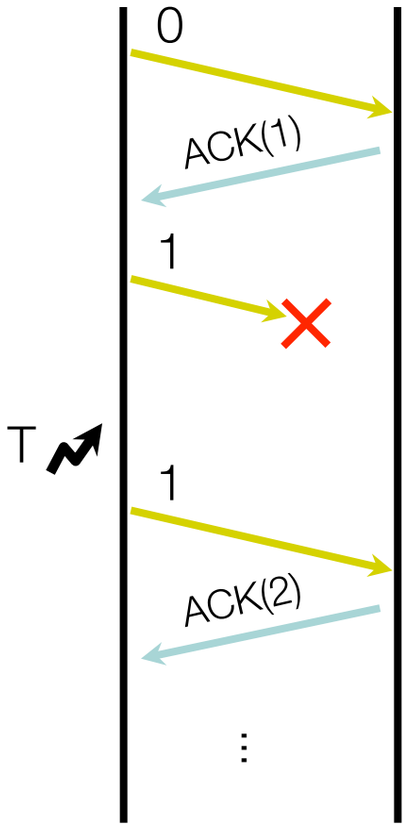
- **En pratique, le récepteur possède un nombre limité de tampons**
  - blocs reçus et non consommés par l'application
- **Contrôle de flux : le récepteur régule le débit de l'émetteur**
  - Contrôle hardware par ligne séparée
    - exemple liaison imprimante parallèle
      - codage des données transparent
  - Message spécifique de contrôle
    - Exemple : Protocole en mode caractères XON / XOFF
    - Fonctionne en tout ou rien
  - Mécanisme de fenêtrage (crédit)
    - le crédit=#données que le récepteur accepte de recevoir (/ buffers dispos)

Contrôle de flux : mécanisme de fenêtrage / crédit

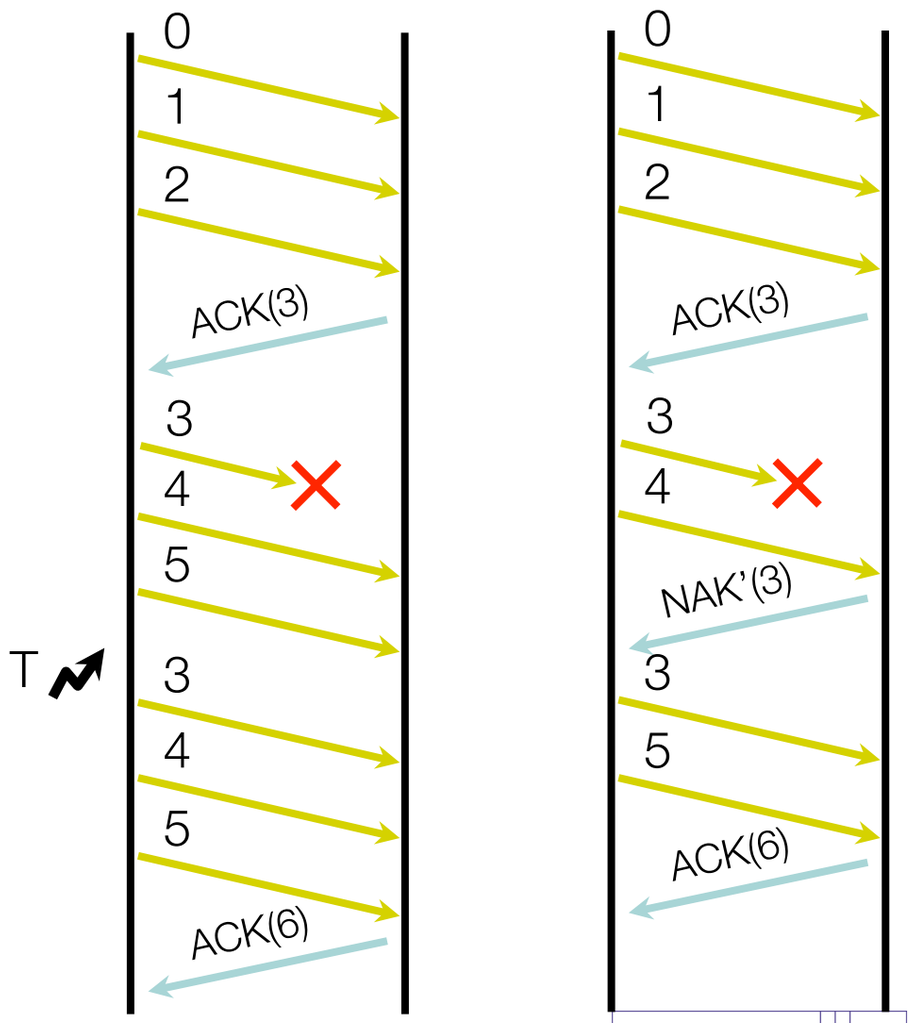
- **L'exemple du contrôle de flux de TCP**
- **Chaque acquittement contient un crédit W (= nombre d'octets)**
  - paramètre Window des messages TCP
- **L'émetteur a une fenêtre d'émission FE**
  - FE contient les données envoyées mais non encore acquittées
  - FE a une taille maximale limitée par W
  - Quand la fenêtre est pleine :
    - l'émetteur est bloqué et doit attendre
      - des acquittements (qui font coulisser la fenêtre)
      - et/ou une augmentation du crédit (taille de FE)

- **Le récepteur**
  - Quand il reçoit un paquet de données envoie
  - Un acquittement et la valeur de W
  - (choisie en fonction ressources disponibles)
- **A réception d'un ACK, l'émetteur**
  - Fait coulisser le début de FE
  - Adapte la taille de FE à W

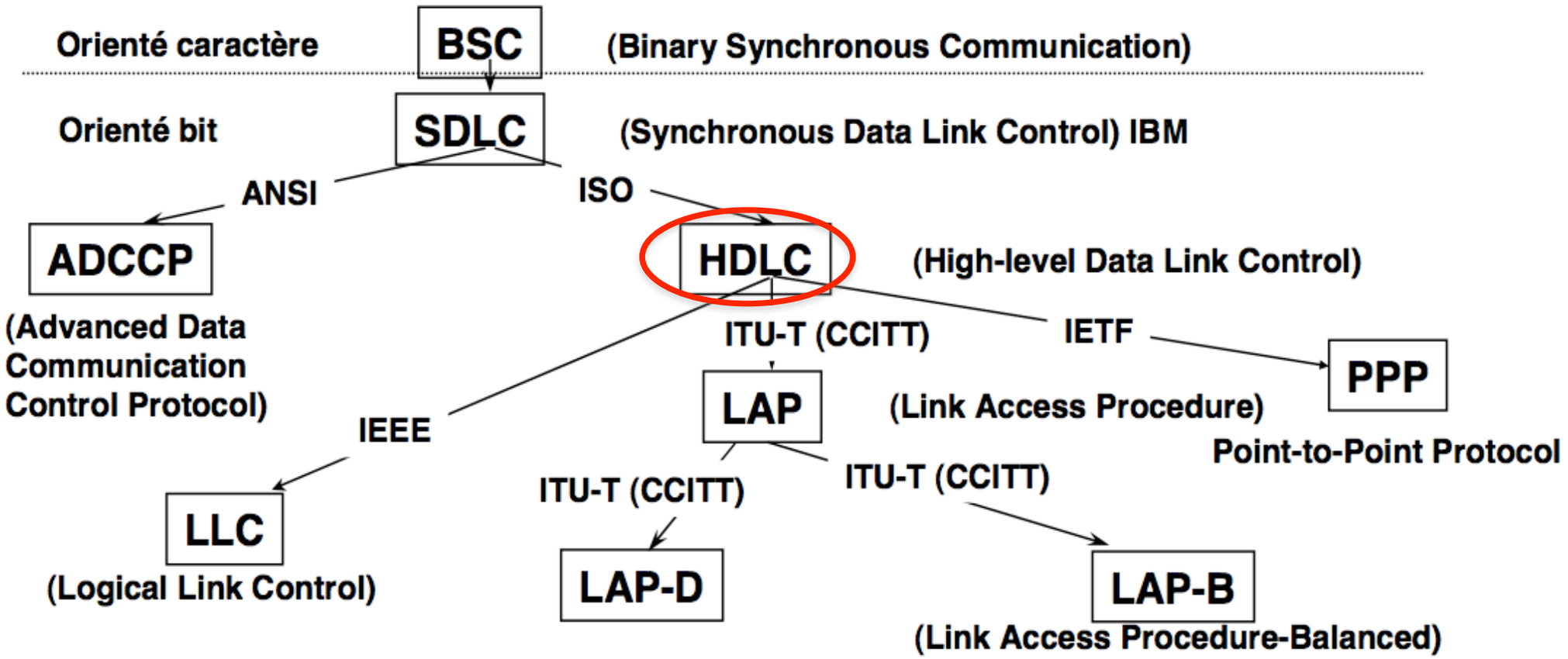
Retransmission : bilan



- **Plusieurs méthodes aux rendements différents**
  - Stop & Wait
  - Go Back N : Retransmission continue
  - Retransmission sélective
- **S'adapter au contexte !**



On est à quel niveau ? «L'ancêtre HDLC...»



- **On est bien au niveau 2 (liaison) mais...**
  - ajdh HDLC => LLC sans fiabilité
  - retransmissions gérées au niveau 4 : TCP

HDLC en vrai : trames de supervision codés sur 4 bits

- **00 : trame d'acquittement RR -> Receive ready (ACK)**
- **01 : trame d'acquittement négatif REJ -> Reject (NACK)**
  - Détection d'une erreur de transmission avec mode Go back N
  - Indique le n° de la 1ère trame incorrecte
- **10 : trame de contrôle de flux RNR -> Receive Not Ready (W=0)**
  - Acquittement de toutes les trames reçues
  - Indique un problème temporaire côté récepteur (ex : mémoire tampon pleine)
- **11 : trame de rejet sélectif SREJ -> Selective Reject (NAK')**
  - Demande la re-transmission d'une trame (Retransmission Sélective)

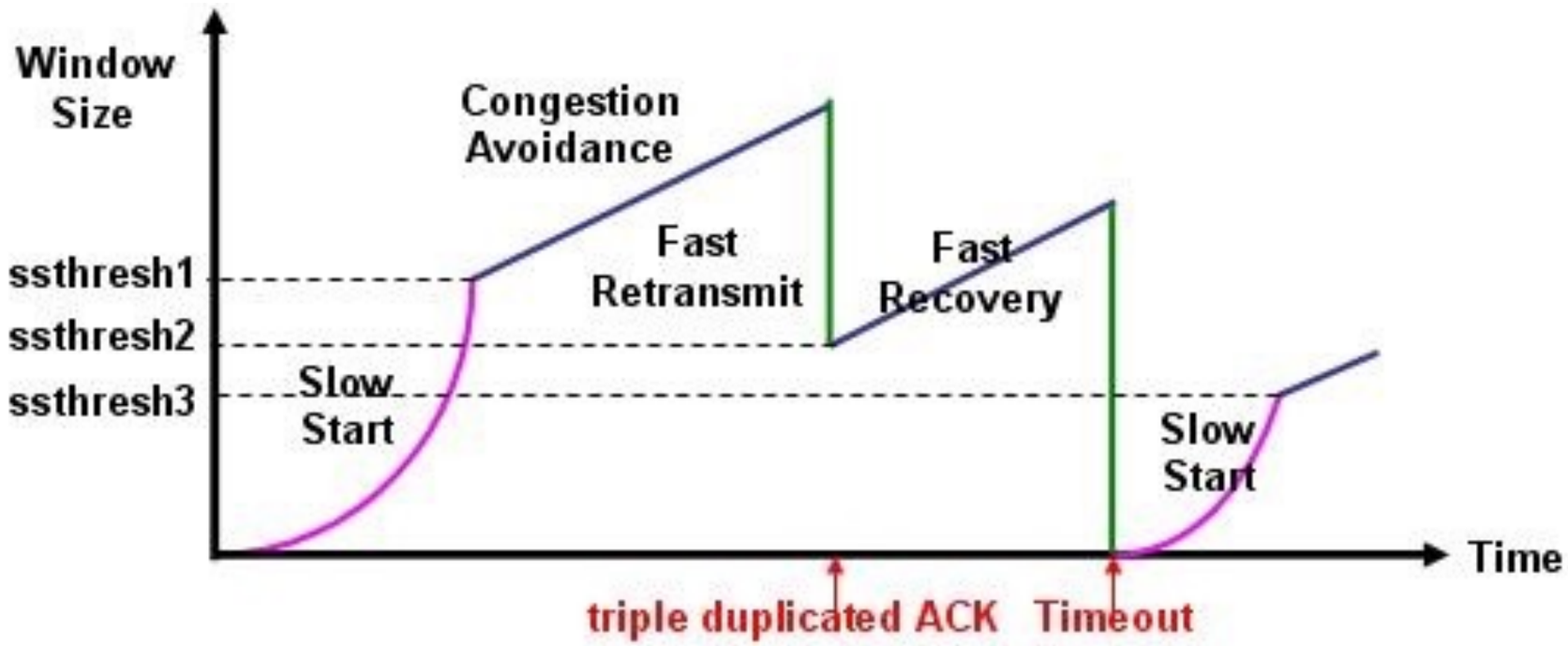
HDLC

Flag 1	Address 1	Control 2	Type 2	Data (Variable)	FCS 4	Flag 1

PPP

Flag 1	Address 1	Control 2	Type 2	Data (Standardized)	FCS 4	Flag 1

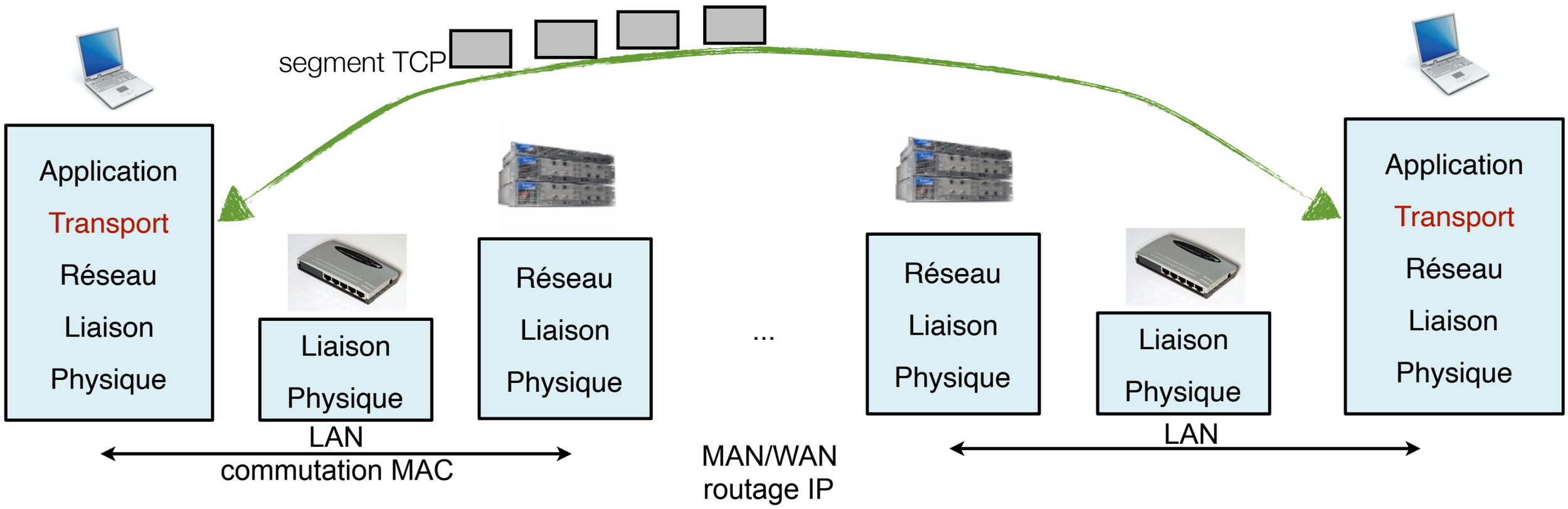
La couche transport & TCP



Le Service Transport

- **Service fourni aux couches supérieures**
  - Assure l'acheminement de messages (TPDU) entre deux applications distantes avec certaines qualités (fiabilité, ordonnancement, délai, …)
  - « cache » les réseaux sous-jacents aux applications
  - Plusieurs services possibles
    - orienté connexion, fiable (ex. TCP)
    - sans connexion, pas fiable (ex. UDP)
    - Transactionnel (sans connexion, fiable)
    - Fiabilité partielle (ex : compromis fiabilité / délai)
  - négociation éventuelle d'options lors de l'établissement d'une connexion transport

Vue d'ensemble



- Protocoles Transport d'origine «IETF»
  - ✓ User datagram protocol (UDP)
  - ✓ Transmission Control Protocol (TCP)
  - ✓ Datagram Congestion Control Protocol (DCCP)
  - ✓ Real-time transport protocol (RTP)
  - ✓ Stream Control Transmission Protocol (SCTP)

Problématique de bout en bout

- **Fiabilité, contrôle de flux, …**
- **Le service sous-jacent utilisé est un réseau et non un lien physique :**
  - adressage explicite
  - établissement de connexion complexe (multiplexage)
  - problème d'errance des données dans le réseau
    - doublons, substitutions, déséquencements (paquets dans le désordre)
    - les délais peuvent être grands et variables
      - mémoires des routeurs intermédiaires

- **Gestion du contrôle de flux et de congestion de bout en bout**
  - difficile (vitesse de réaction, interaction entre plusieurs connexions)

Une connexion à un service

- **Adresser l'application distante**
  - adresse de machine (ex IP) + adresse de point de connexion (exemple port TCP)
  - multiplexage/démultiplexage de connexions
  - Ex TCP : Ident. connexion = @ source + @ destination + port source + port destination
- **Eviter les substitutions**
  - confusion des données d'une ancienne connexion avec celles d'une nouvelle connexion de même identificateur
    - on associe à chq connexion une référence avec modulo très grand (TCP : 32 bits)
    - négociation de la référence lors de la connexion
    - échange tripartite
- **Etre « sûr» que la connexion est établie**
  - « problème des deux armées »

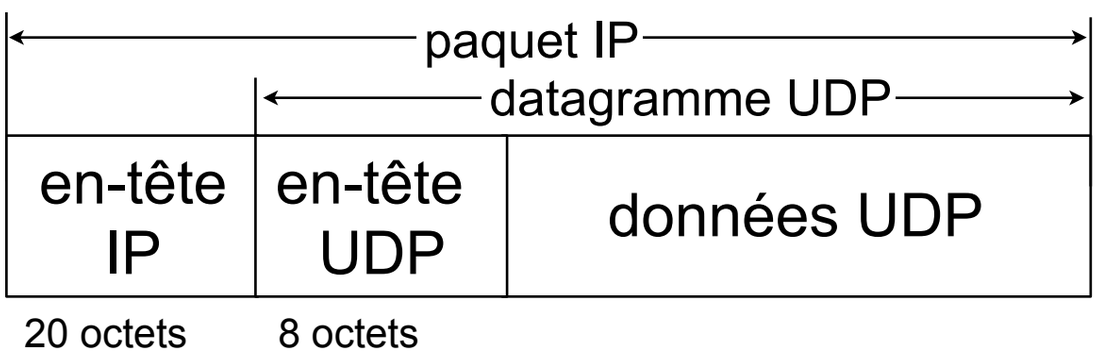
Service transport

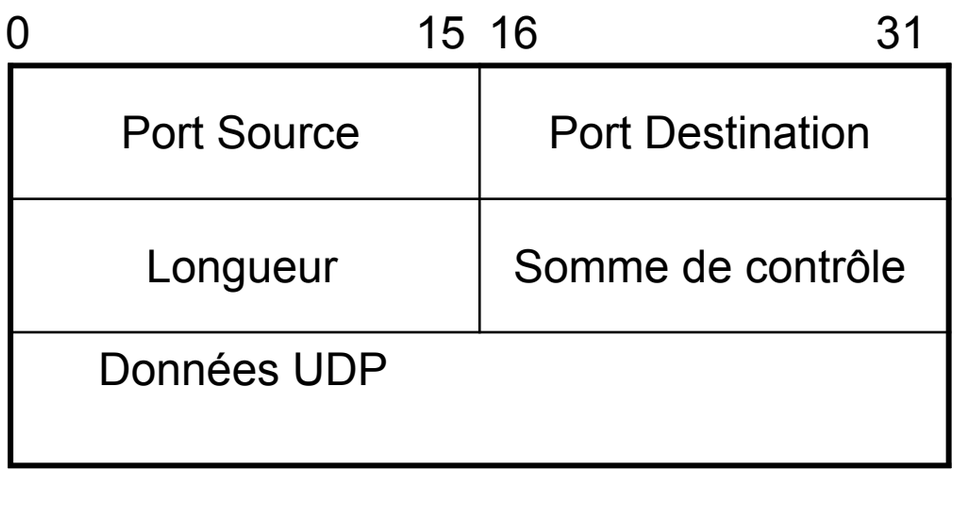
- **Service fourni aux couches supérieures applicatives**
  - ➡ assure l'acheminement des messages selon certains critères qualités:
    - ➡ fiabilité, ordonnancement, délai, etc...
  - ➡ «cache» les réseaux sous-jacents aux applications
  - ➡ négociation éventuelle d'options lors de l'établissement de la connexion

- ✓ **Transmission non fiable, non séquentielle, unicast ou multicast : *UDP***
  - ➡ temps réel
  - ➡ garantissant la bande passante disponible
  - ➡ utilisé par le multicast IP
- ✓ **Transmission fiable, séquentielle, unicast : *TCP***
  - ➡ contrôle de congestion
  - ➡ contrôle de flux
  - ➡ orienté connexion

UDP

- **User Datagram Protocol : Protocole de transport minimaliste**
  - IETF RFC 768
  - Protocole très simple (peu de fonctions)





- **UDP n'offre aucune garantie de fiabilité**
- **UDP se contente d'envoyer les datagrammes de l'application vers la couche IP**

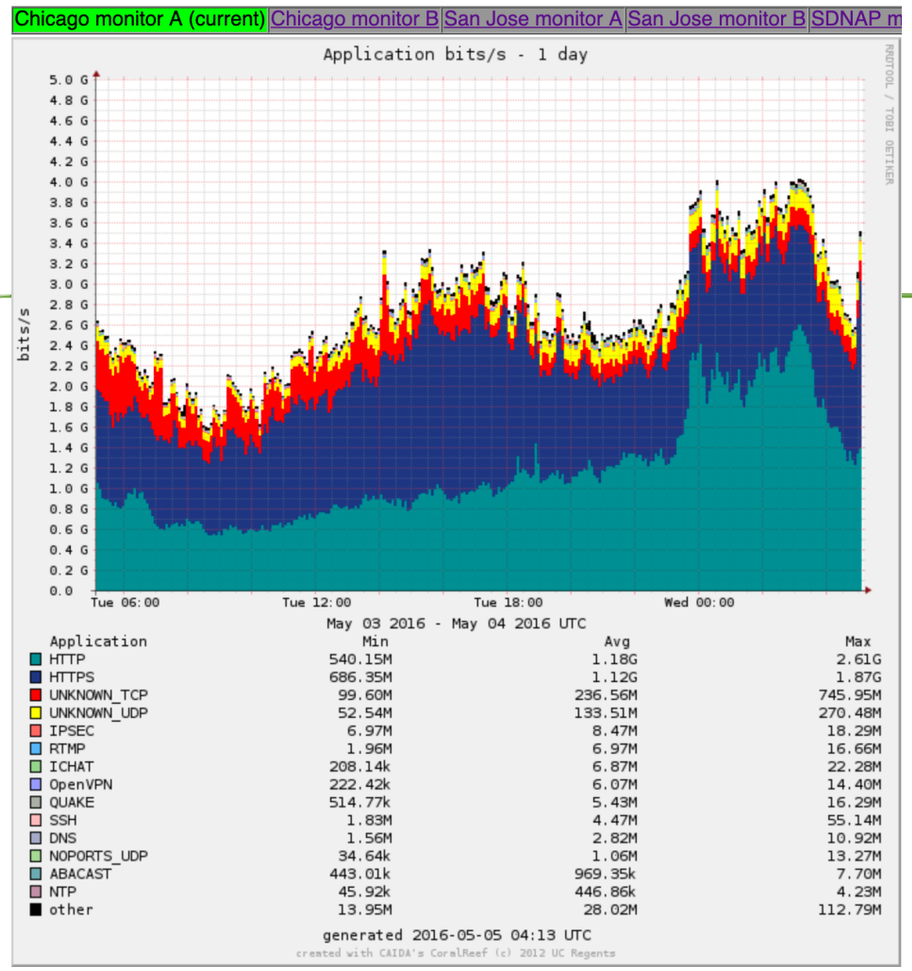
✓Best Effort : les segments UDP peuvent être :
  - perdus, transmis dans le désordre

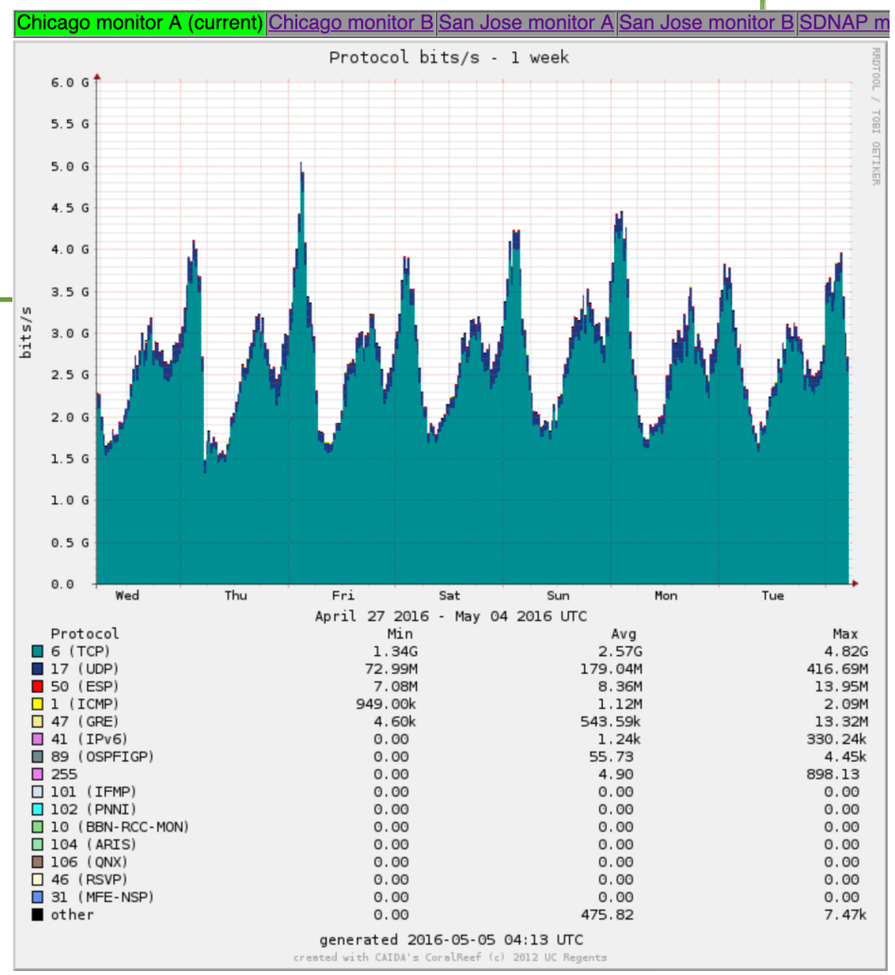
✓ Sans connexion
  - pas de négociation src/dst et chaque segment est traité indépendamment

✓ Avantages
  - pas de délai d'attente de connexion, pas d'état à maintenir, en-tête simplifiée, pas de contrôle de congestion (forts débits sans bridage !)

Rappel : ports réservés

- **Assure le multiplexage des applications (pour une IP donnée)**
- **Ports réservés (attribué par l'IANA) : 1-1023**
  - ➡ exemples :
    - ✓ 7 : echo
    - ✓ 21 : FTP
    - ✓ 22 : SSH
    - ✓ 23 : telnet
    - ✓ 25 : SMTP
    - ✓ 80 : HTTP
- **1024-5000 : ports éphémères (clients)**
- **5000+ : serveurs**





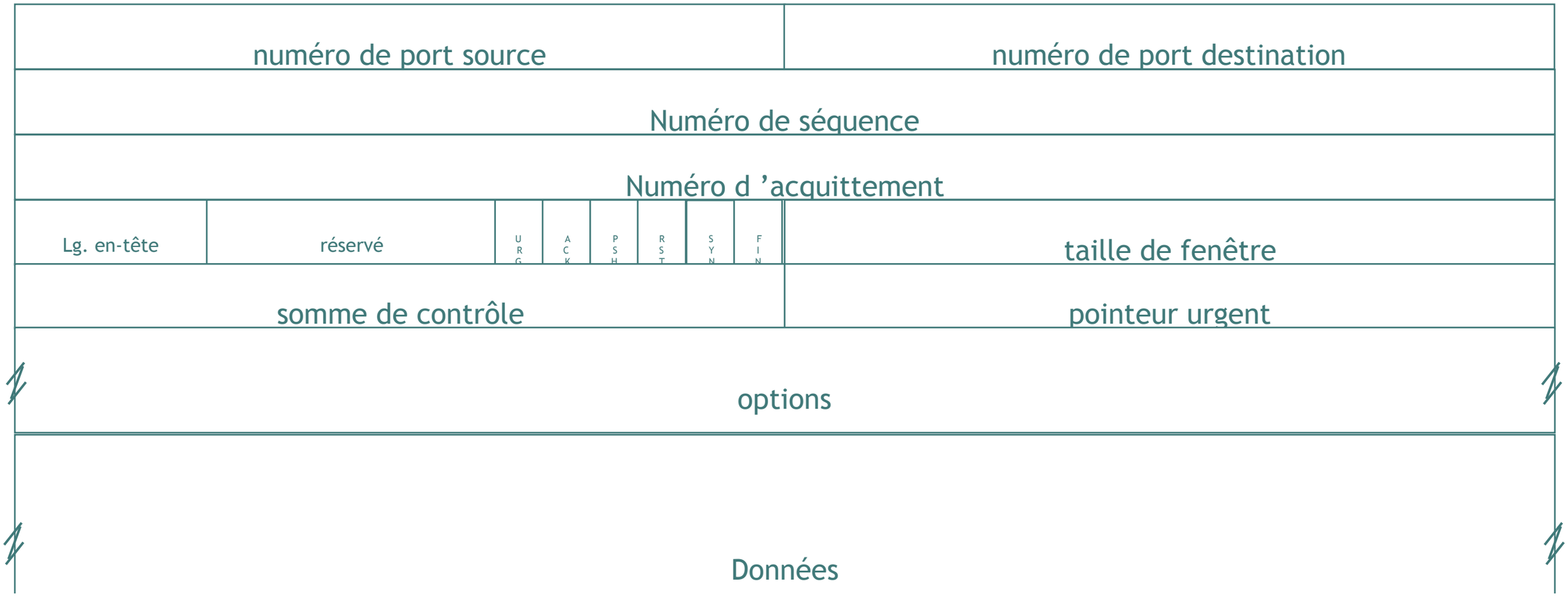
TCP : Généralités

- **Transmission Control Protocol**
  - ✓ RFC 793, 1122, 1323, ...
  - ✓ Constante évolution (en particulier pour le contrôle de congestion)

  - ✓ **Modèle de service TCP**
    - ➡ orienté connexion, bidirectionnelle
    - ➡ fiable : gère les pertes, ré-ordonne les données
    - ➡ assure un contrôle de flux entre émetteur et récepteur
    - ➡ connexion TCP est identifiée par deux extrémités
      - ➡ adresses des sockets émetteur et destinataire
    - ➡ Données véhiculées sous forme de flot d'octets
      - ➡ pas de délimitation des messages de bout en bout

- ✓ ~ 90 % du trafic sur Internet (http, transfert de fichiers, ...)
- ✓ Plus complexe qu'UDP mais plus sûr :)

En-tête d'un segment TCP

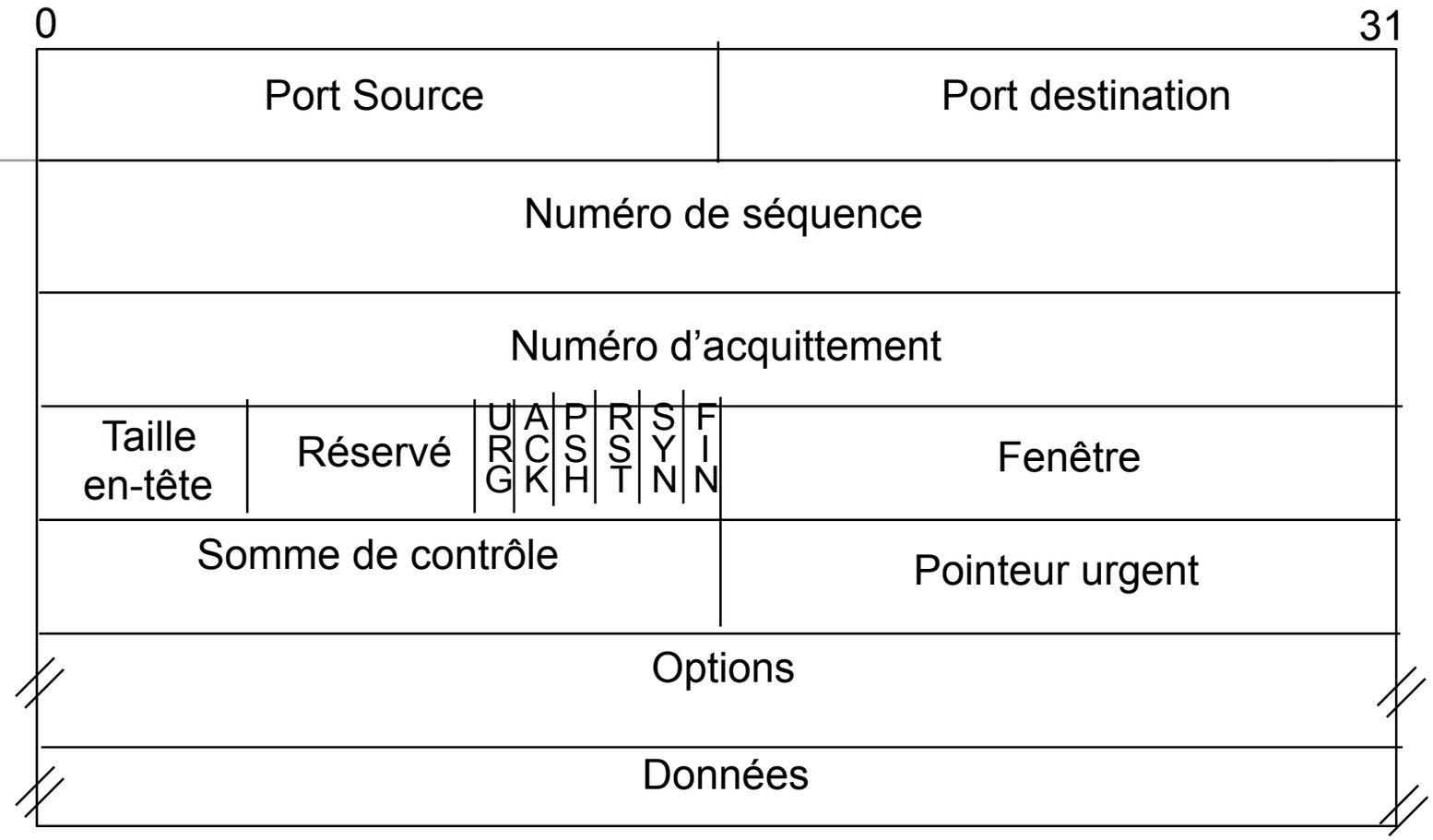


- ✓ **Les ports sources et destination identifient l'application émettrice et réceptrice**
- ✓ **Chaque numéro de séquence ou d'acquittement est relatif à un octet de données**

➡ N° de séquence = N° du premier octet du segment (/ flux de données émetteur)

➡ N° d'acquittement = N° du premier octet du segment attendu (piggy backing)

En-tête TCP (2)



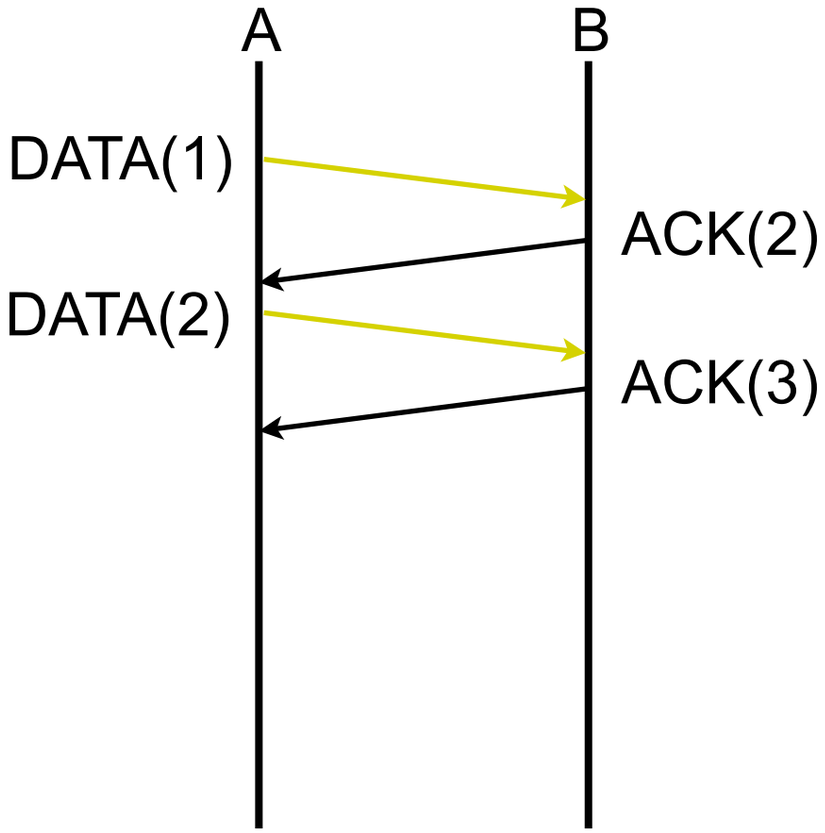
- ✓Taille de l'en-tête : donne la longueur de l'en-tête en mots de 32 bits (5 mots max + options)
- ✓URG : pointeur urgent doit être exploité
- ✓ACK : le numéro d 'acquittement est valide
- ✓PSH : Le récepteur doit passer les données le plus rapidement possible à l 'application
- ✓RST : Réinitialisation de la connexion
- ✓SYN : Synchronisation des numéros de séquence pour initialiser une connexion
- ✓FIN : Fin de l 'envoi de données
- ✓Fenêtre : nombre d'octets que le récepteur peut encore recevoir à partir du N° d'acquittement
- ✓Pointeur urgent : position de la fin des données urgentes
- ✓Le contrôle de flux est fourni par chaque extrémité en annonçant une taille de fenêtre

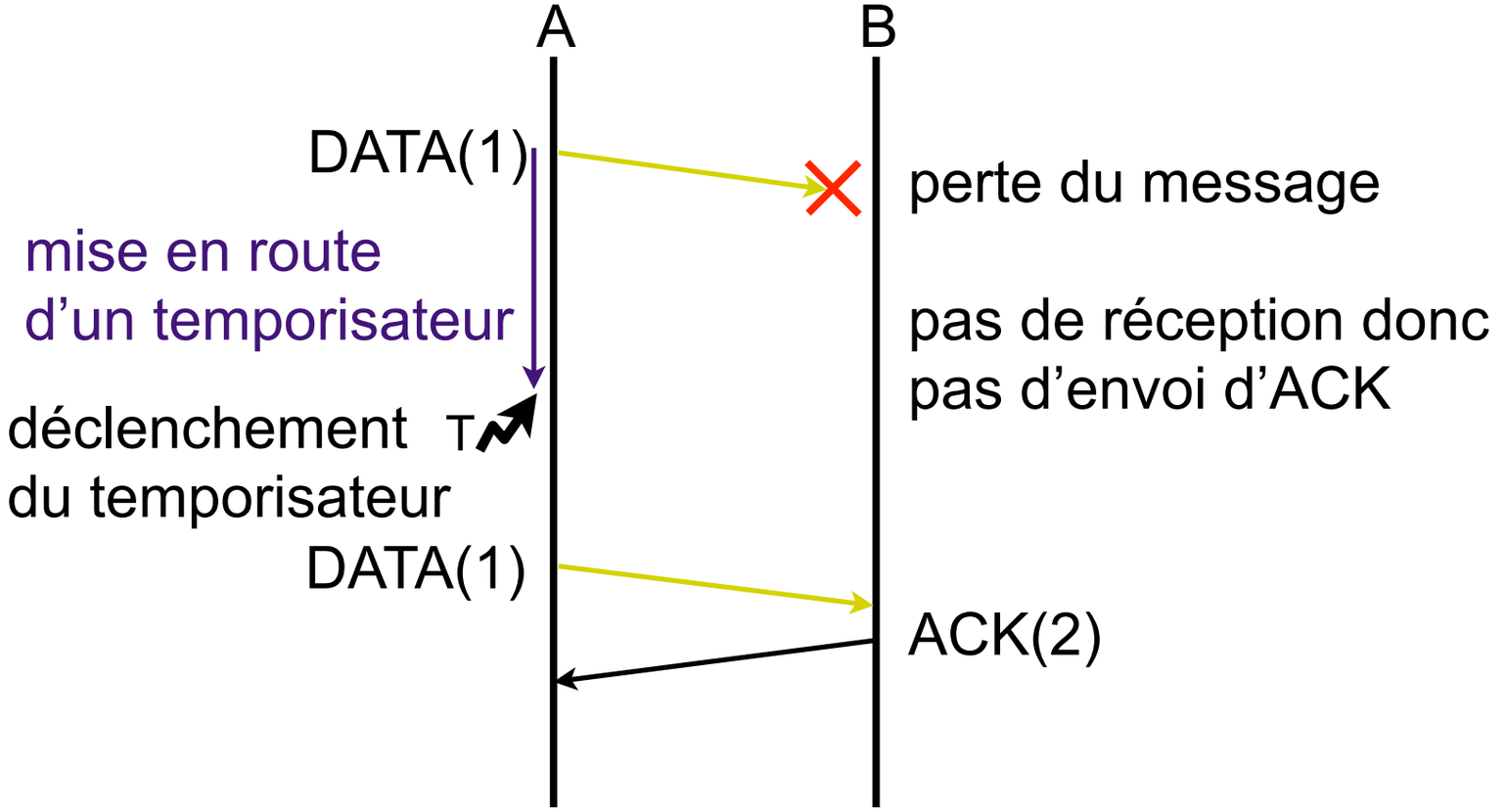
Options TCP

- ✓ MSS : taille maximale du segment que l'émetteur accepte de recevoir
- ✓ timestamp : pour calculer la durée d'un aller/retour (RTT)
- ✓ sack : acquittement sélectif
- ✓ wscale : facteur d'echelle ("shift") pour augmenter la taille de la fenêtre au delà des 16 bits du champ WINDOW (65535 max par défaut)
  - ➡ Si > 0, la taille de la fenêtre est de 65535×2^{shift}
- ✓ nop : Les options utilisent un nombre quelconque d'octets, par contre les paquet TCP sont toujours alignés sur une taille de mot de quatre octets
  - ➡ complétion des mots (bourrage/padding)

TCP (rappel HDLC) : gestion des pertes

- Acquittements & Timers

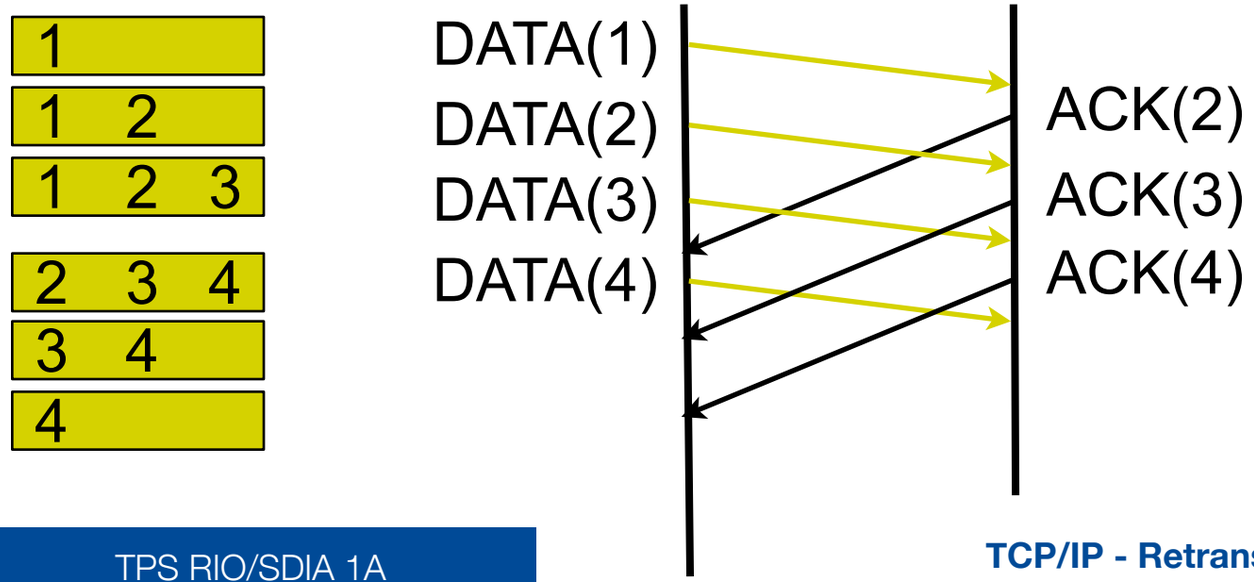


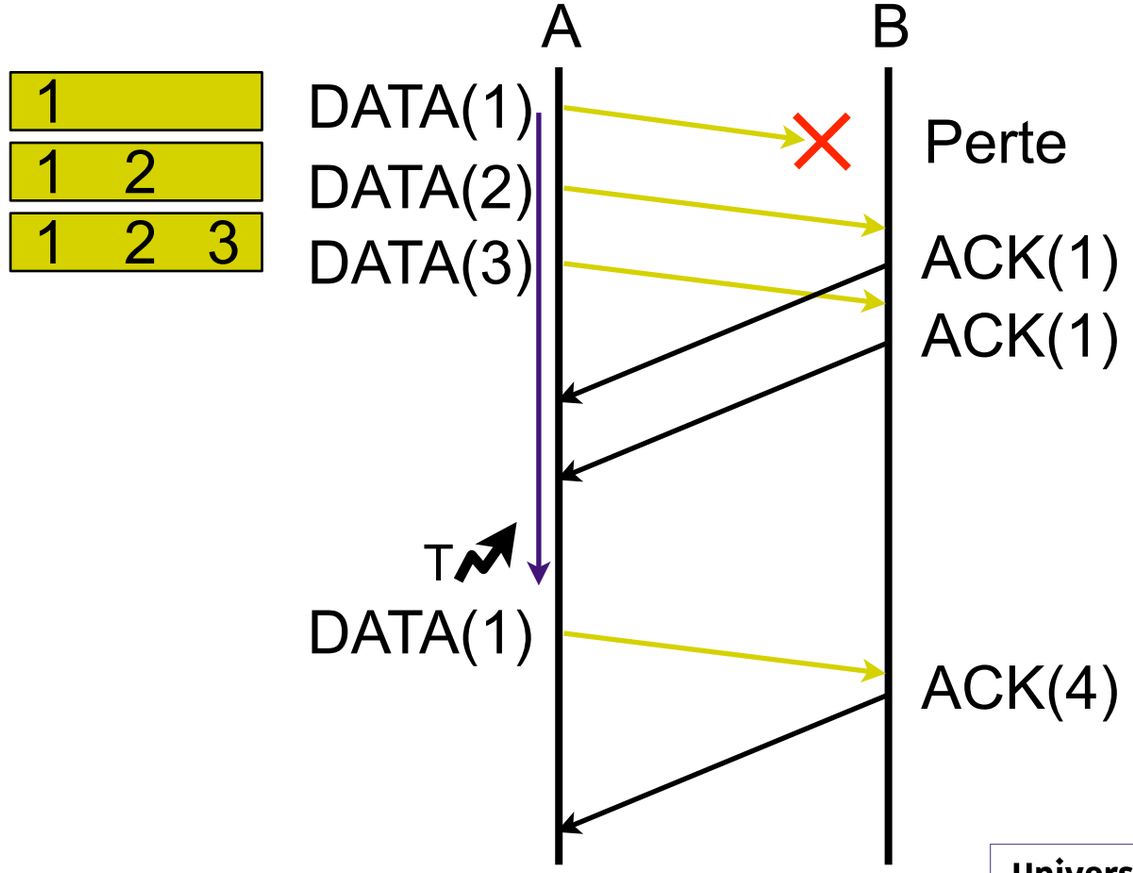


TCP (rappel HDLC) : fenêtre glissante

- ***Comment accélérer l'échange de segment ?***
  - ➡ ***Anticiper en utilisant des fenêtres !***

- ✓ La fenêtre d'émission indique quels sont les messages que l'émetteur peut envoyer
- ✓ La taille de la fenêtre correspond au nombre de messages que l'émetteur peut envoyer sans attendre les ACK !





TCP : ouverture/fermeture d'une connexion

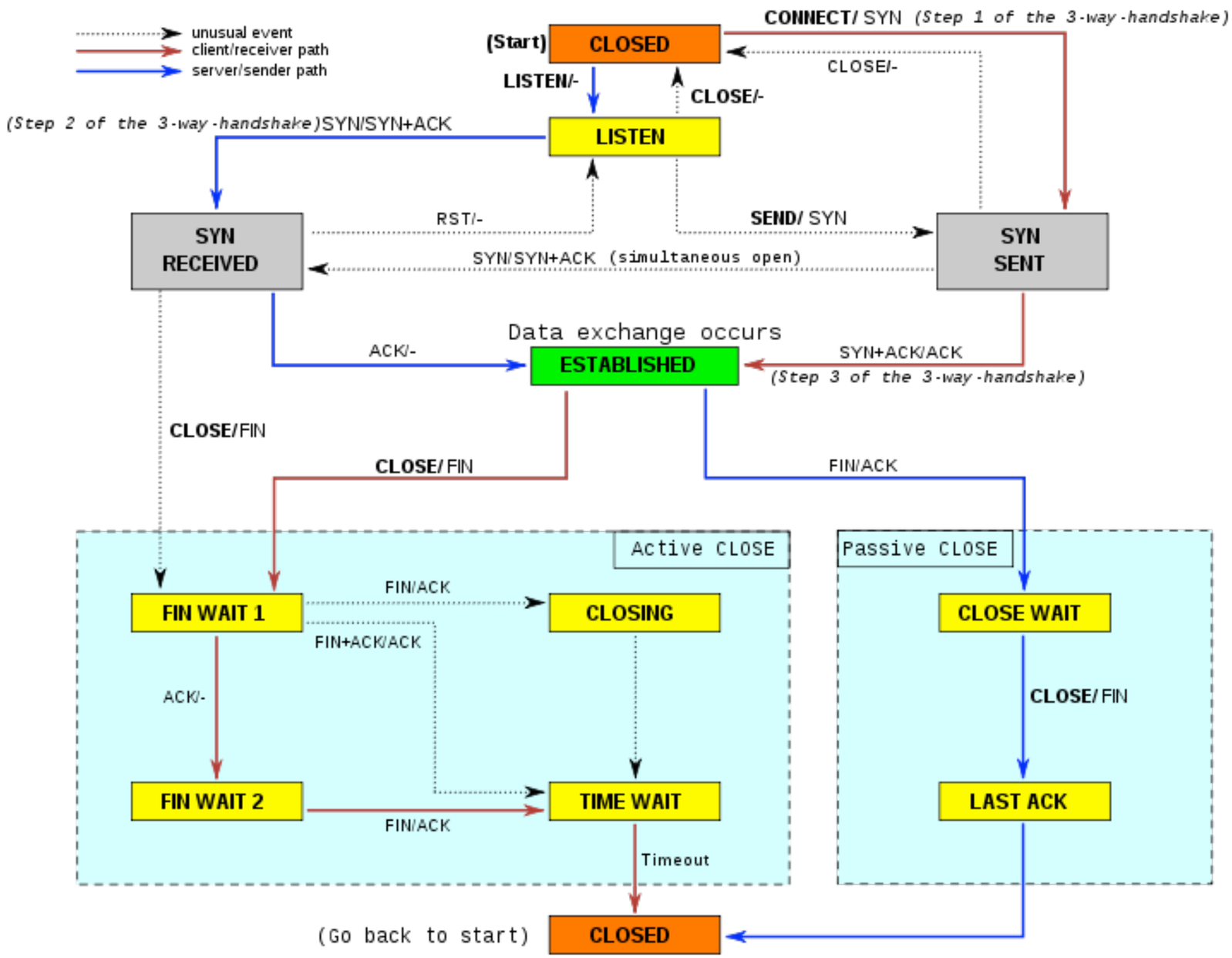
- **Etablissement d'une connexion**
  - ➡ le côté qui envoie le premier SYN effectue une ***ouverture active*** (svnt le client)
  - ➡ l'autre extrémité effectue une ***ouverture passive*** (généralement le serveur)
  - ➡ l'envoi d'un SYN consomme un numéro de séquence
  - ➡ chaque extrémité choisi son numéro de séquence initial

- **3 segments TCP pour ouvrir une connexion**

- **4 segments TCP pour fermer une connexion**
  - ➡ on parle de ***semi-fermeture***

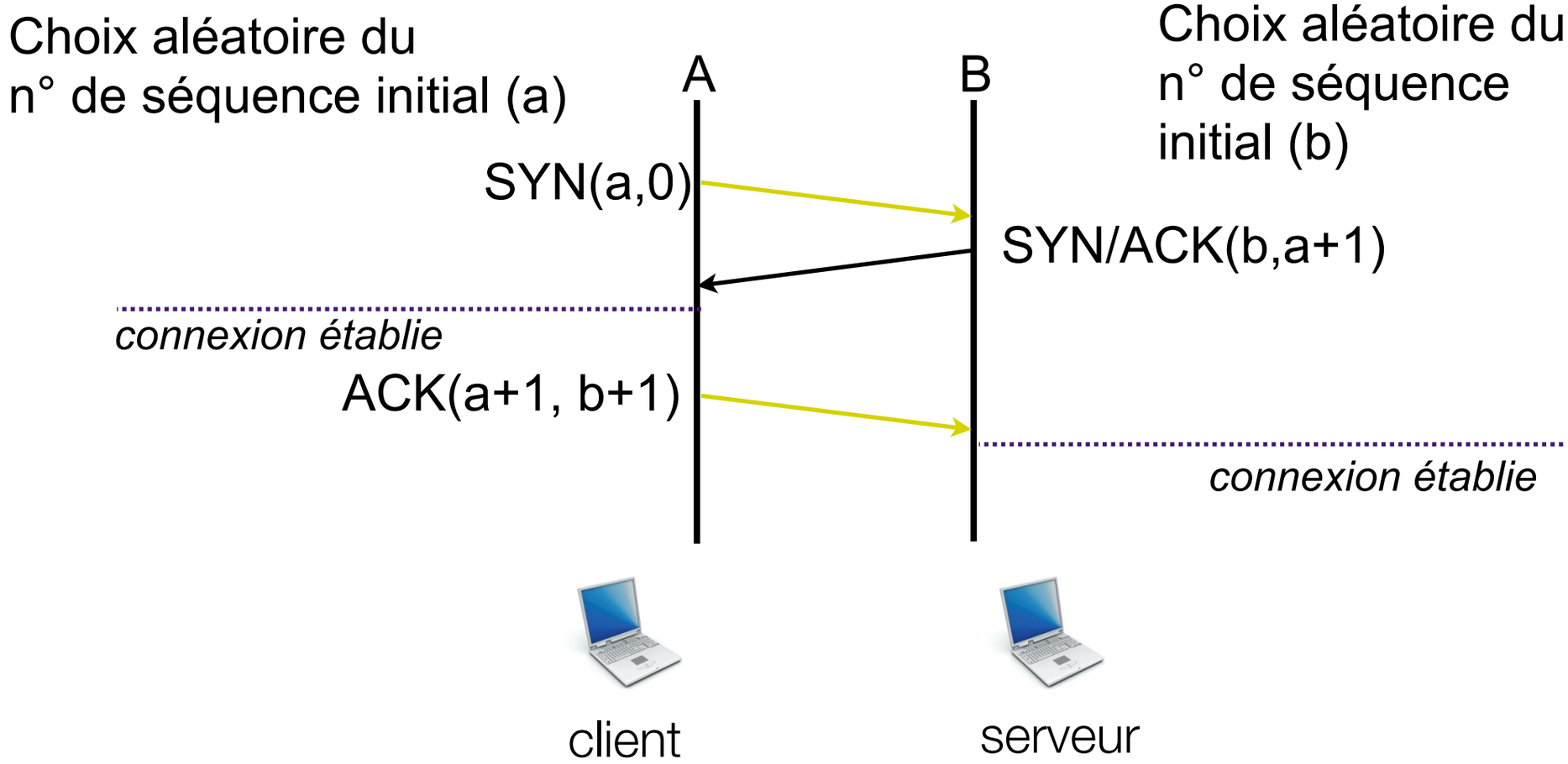
- Une connexion TCP est full-duplex et chaque direction peut être arrêtée indépendamment
- Chaque extrémité envoie un message FIN
  - ➡ réception d'un message FIN ➥ plus de données émises dans cette direction

TCP : établissement et fermeture

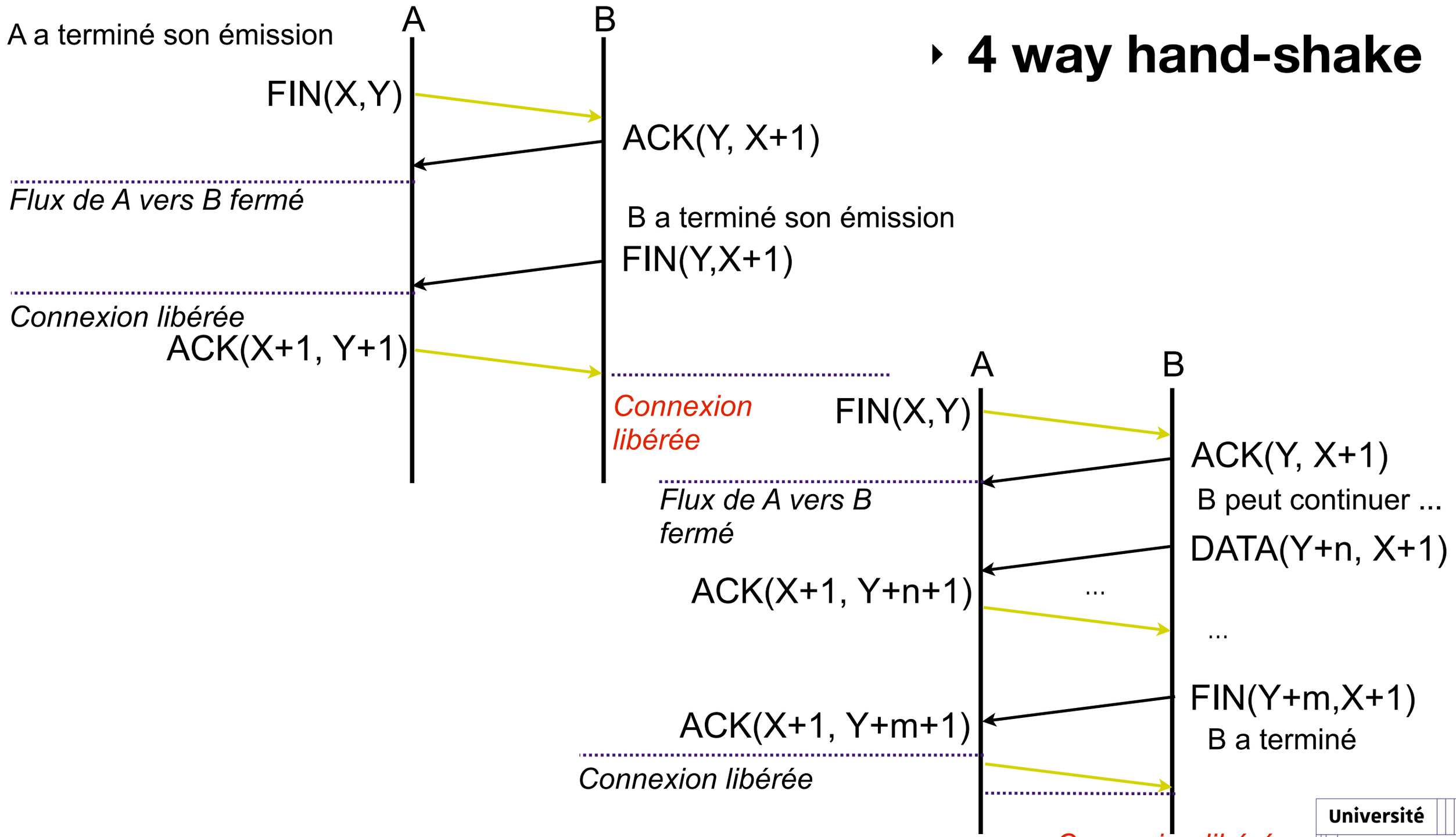


TCP : établissement connexion

- **3 way hand-shake**



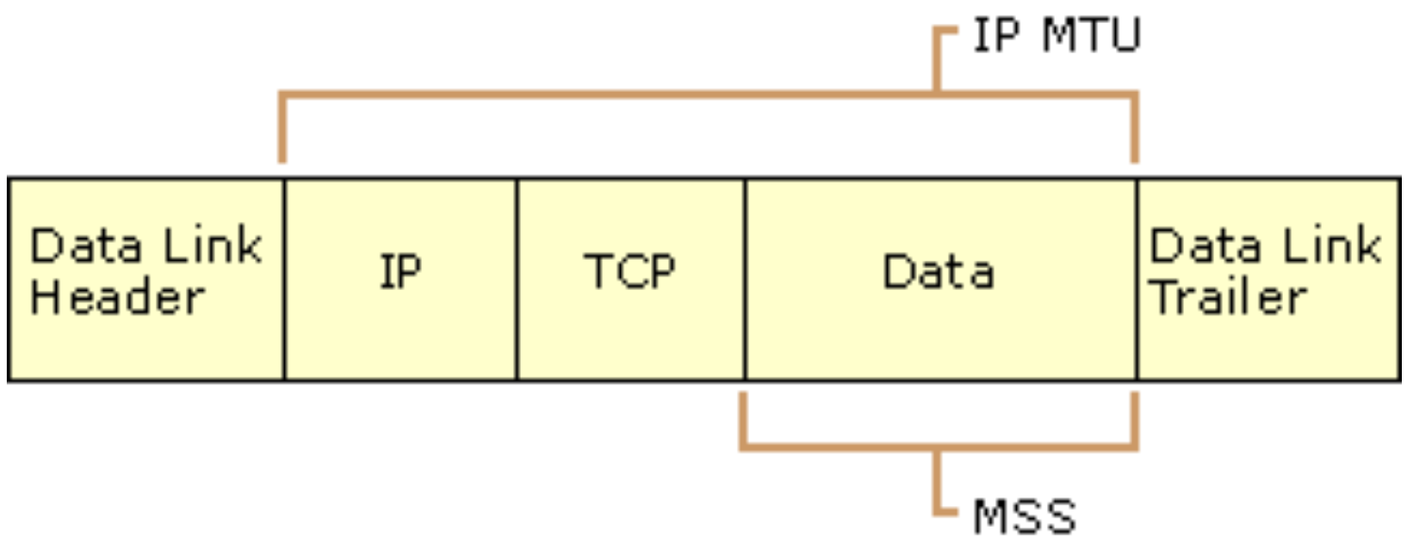
TCP : fermeture connexion



- **4 way hand-shake**

TCP : MSS

- Taille maximale d'un segment :
  - ✓ Maximum Segment Size
    - ➡ plus grand morceau de données que TCP enverra à l'autre extrémité
  - ✓ lors d'une connexion, chaque extrémité peut annoncer sa MSS
  - ✓ l'option MSS ne peut apparaître que dans un segment SYN
  - ✓ la valeur par défaut est 536
  - ✓ en général, les implémentations de TCP choisissent le MSS le plus grand possible tout en évitant la fragmentation ➥ efficacité maximum



Transfert de données et contrôle de flux

- **Transfert bidirectionnel simultané**
- **Transfert de données**
  - la gestion des fenêtres d'anticipation est liée :
    - aux réceptions d'acquittements
    - au rythme de lecture / écriture des applications

- **L'émetteur peut envoyer les octets compris entre**
  - dernier N° ACK (+ grand modulo 2^{32}) reçu et
  - dernier N°ACK reçu + dernier W reçu
- **W permet donc au récepteur de ralentir émetteur**
  - W = « crédit émission »

Blocages ?

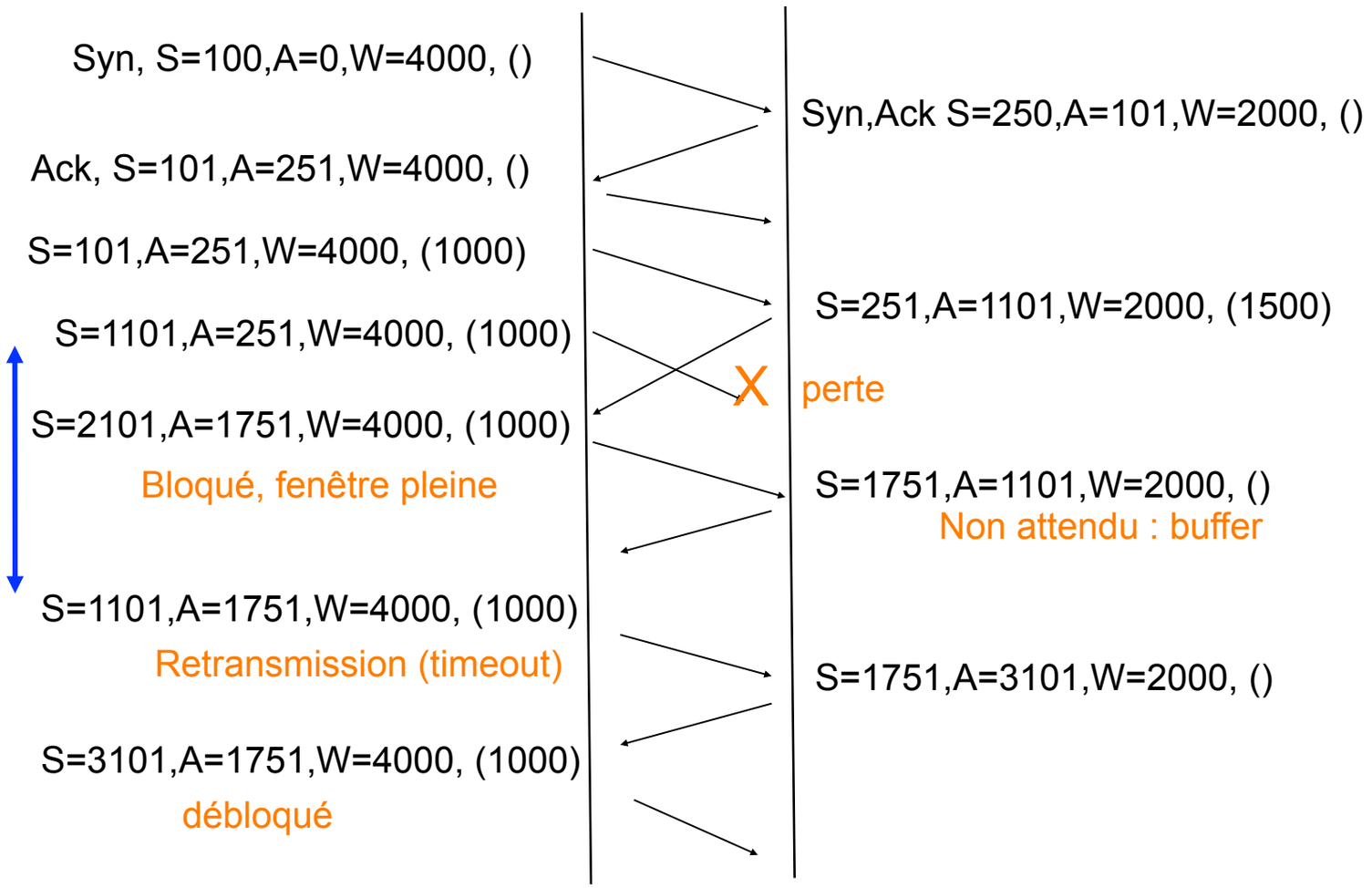
- **Quand W = 0, l'émetteur n'envoie plus de données sauf dans deux cas :**
  - données urgentes
  - segment d'1 octet : oblige le récepteur à ré-annoncer le prochain octet attendu et la taille de la fenêtre
    - => évite les interblocages si l'annonce d'une taille de fenêtre > 0 s'est perdue

- **Emetteurs : pas tenus de transmettre immédiatement les données qui proviennent des applications**
- **Récepteurs : pas tenus d'acquitter le plus rapidement possible**
  - Souvent un acquittement tous les 2 segments

Estimation RTT

- **Fiabilité**
  - récepteur accepte dans l'ordre et acquitte (mais peut mémoriser dans le désordre)
  - émetteur arme délai de retransmission RTO
    - associé au plus ancien octet non acquitté
    - Déclenchement RTO => retransmission segment le plus ancien
- Efficacité dépend estimation RTO (très variable suivant connexion)
- Principe
  - RTT calculé pour chaque paquet acquitté : **`dernierRTT`**
  - **`NouveauRTT`** = a*__`AncienRTT`__ + (1-a)*__`dernierRTT`__
  - RTO = b * **`NouveauRTT`** (0 < a < 1, b > 1)
  - Algorithme de Karn : ne pas tenir compte des paquets retransmis

Exemple : connexion + échange



TCP : Contrôle de flux

- **Débit standard avec TCP standard**
  - ➡ W < 64Ko
  - ➡ au plus W par RTT (dépendance au RTT !)
    - ✓ RTT = 1ms (LAN) : 64Ko/ms ➡ 64Mo/s ≈ 500Mb/s
    - ✓ RTT = 640ms (satellite) : 64Ko / 640ms ≈ 800Kb/s
- Intérêt du paramètre Wscale (Window Scale)
  - ➡ Wscale = n ➡ multiplie W par 2^n

- **Emetteur est bloqué dans 2 cas :**
  - ➡ W = 0 (récepteur saturé) ➡ attente de recevoir un segment avec W > 0
  - ➡ Fenêtre d'émission pleine
    - W octets envoyés et non encore acquittés
      - ✓ si perte d'un ACK : attendre ACK suivant (cumulatif)
      - ✓ si perte données : attendre RTO (Retransmission TimeOuts) et retransmettre
      - ✓ si aucune erreur : attendre ACK (au plus RTT)
        - peut indiquer que W trop petit

TCP : Contrôle de congestion

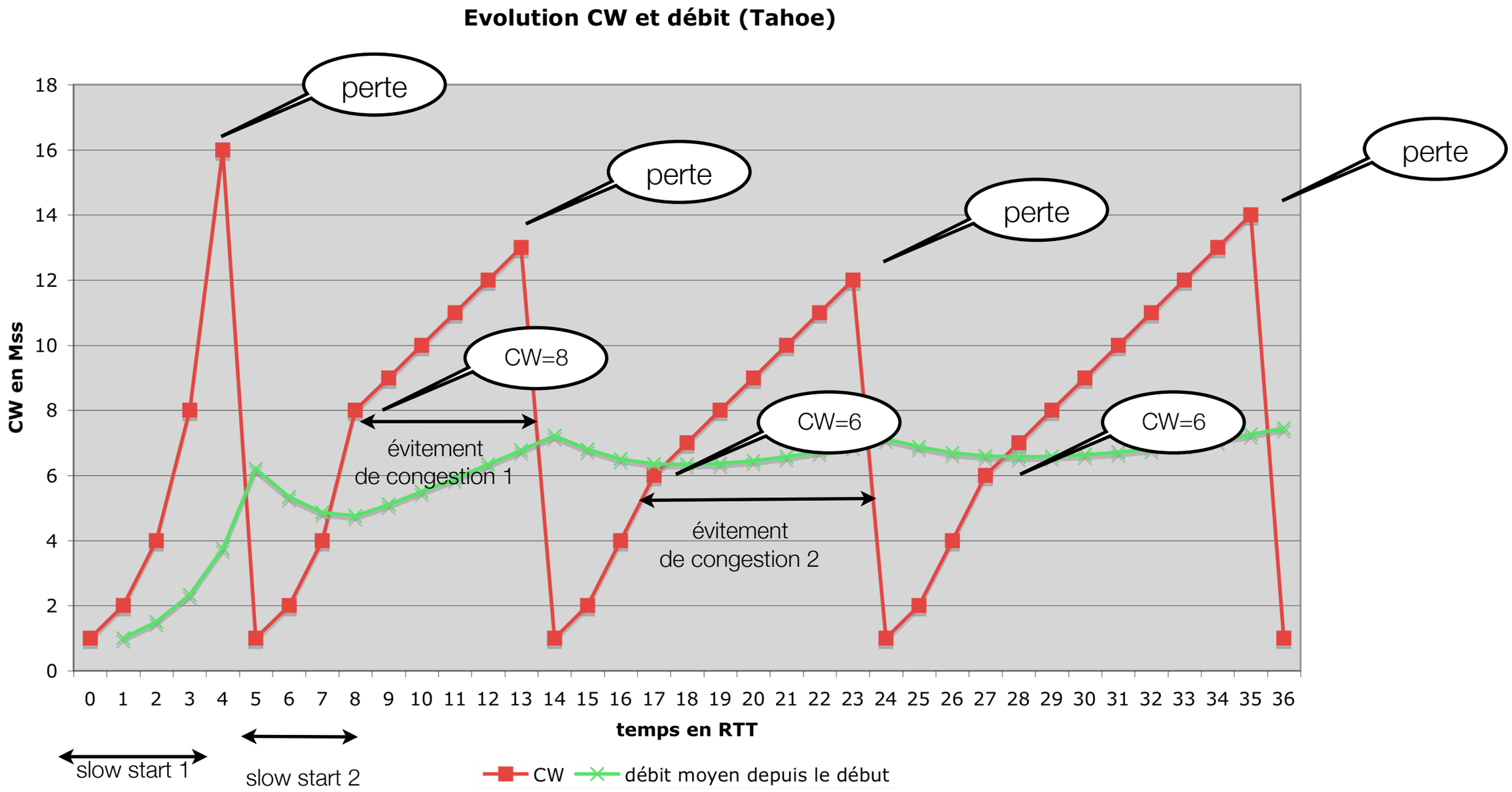
- **Dans l'Internet, contrôle distribué par les stations émettrices**
  - ✓ TCP coté émetteur s'en charge en régulant le débit
    - ➥ variation dynamique de la taille de la fenêtre d'émission
  - ✓ les pertes (et retransmissions) sont interprétés comme signe de congestion dans le réseau

- **Plusieurs fenêtres :**
  - ✓ fenêtre de contrôle de flux
    - ➥ relative à la capacité du récepteur (paramètre W)
  - ✓ fenêtre de congestion
    - ➥ relative à la capacité du réseau (paramètre CW)
  - ✓ la quantité de données envoyés correspond au minimum de ces deux fenêtres
    - ➥ FE = min(W,CW)
    - ➥ en pratique la Fenêtre d'Emission (FE) est aussi limité par la taille des tampons

TCP : Contrôle de congestion

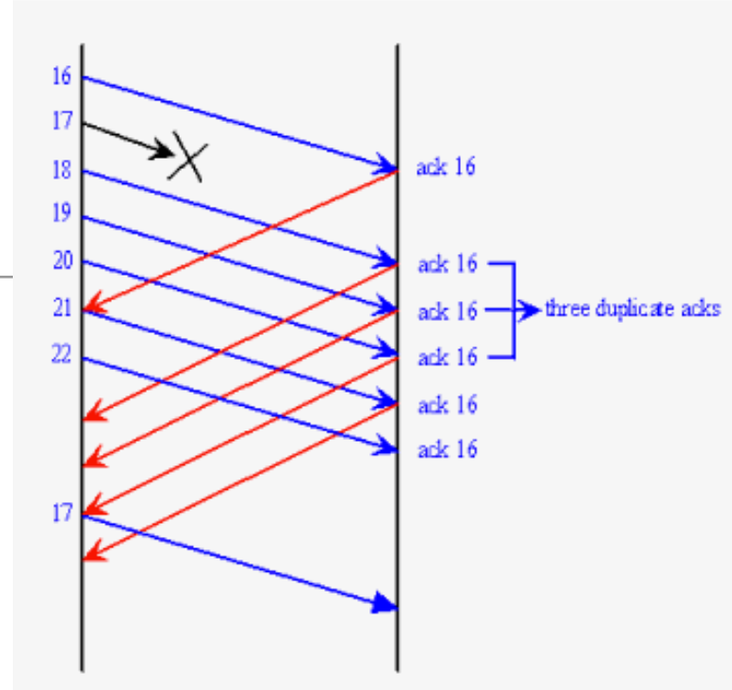
- **TCP basique (Tahoe, années 80)**
  - ➡ débit initial faible (capacité du réseau inconnue) mais augmentation rapide...
  - ➡ La Fenêtre de Congestion (CW) est mesurée en MSS
  - ➡ Algorithme slow start (Jacobson 88)
    - ✓ à l'établissement de la connexion : CW = 1 MSS
    - ✓ l'émetteur envoie un segment de taille maximum
    - ✓ à chaque ACK sans timeout, CW ++ tant que CW ≤ W : tout va bien...
      - ➥ à chaque RTT sans perte, CW double (croissance binaire exponentielle)
  - ➡ sinon lors d'une retransmission (timeout due à une perte de paquet)
    - ✓ on considère qu'il s'agit d'une congestion ! ➥ redémarre en slow start avec CW = 1
  - ➡ Pour minimiser les oscillations : évitement de congestion
    - ✓ quand la fenêtre CW atteint un seuil S
      - ➥ on augmente CW de ~1 MSS par RTT (au lieu de 1 par ACK ➥ linéaire)
      - ➥ calcul de S : initialement S = W_{max} (64Ko), puis à chq retransmission sur timeout
        - ✓ S = CW/2; CW = 1;

TCP : Tahoe



- **«Stabilisation ?» ~ recherche du «toit» ...**

TCP amélioré : Reno

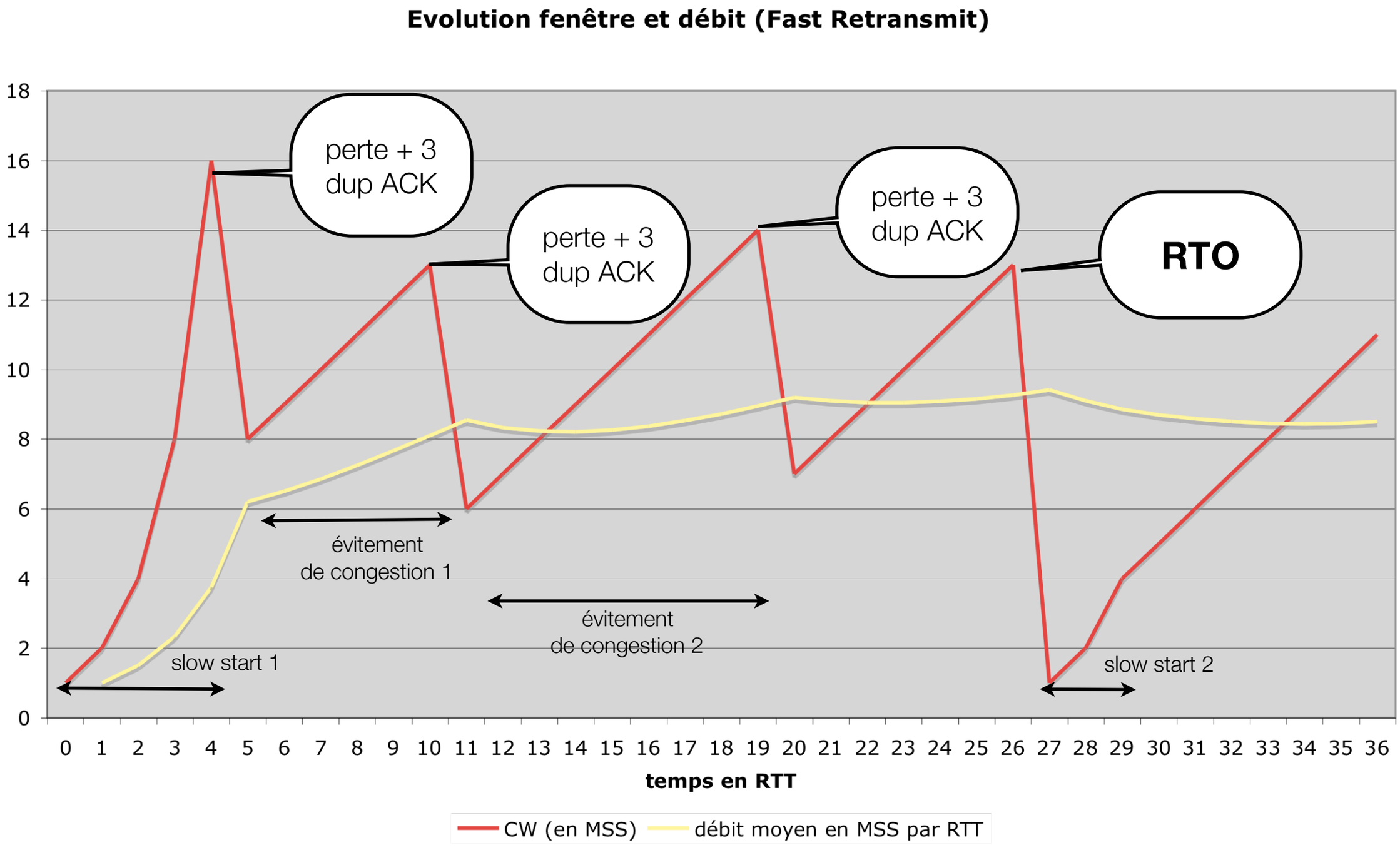


- **Lorsque les pertes surviennent, attente RTO ? peu efficace !**
  - ➡ fast retransmit + fast recovery (TCP Reno)
  - ➡ si 3 ACK dupliqués
    - ✓ 3 paquets sont arrivés après un «trou» (un ou + segments perdus)
    - ✓ ~ réseau congestionné mais “pas trop” (une seul perte ?)
    - ✓ fast-retransmit (RFC 2581) : ré-émettre le paquet «perdu»,
    - ✓ puis fast-recovery :
      - ✓ S = CW/2, CW = S + 3MSS et CW++ pour chaque dup ACK en plus (➡ flight size)
        - ★ après réception d’un «ACK partiel» (au moins le ACK attendu)

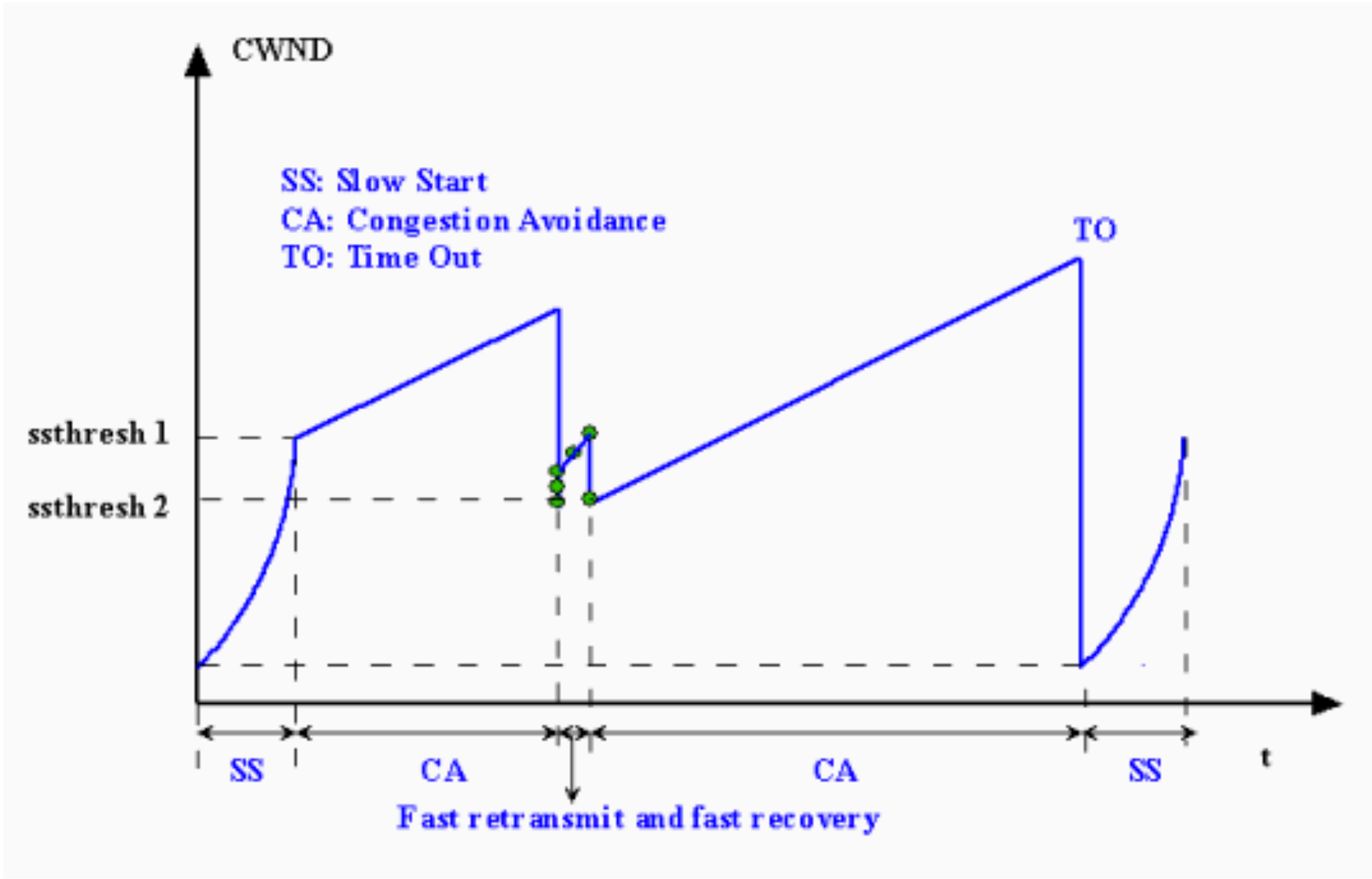
          ➥ Fin fast-recovery:

          CW = S et évitement de congestion classique
  - ➡ si timeout : S = CW/2; CW = 1 => slow start !

TCP : Reno


Evolution fenêtre et débit (Fast Retransmit)
perte + 3 dup ACK
perte + 3 dup ACK
perte + 3 dup ACK
RTO
évitement de congestion 1
évitement de congestion 2
slow start 1
slow start 2
temps en RTT
CW (en MSS)
débit moyen en MSS par RTT

TCP : Reno (en détail)



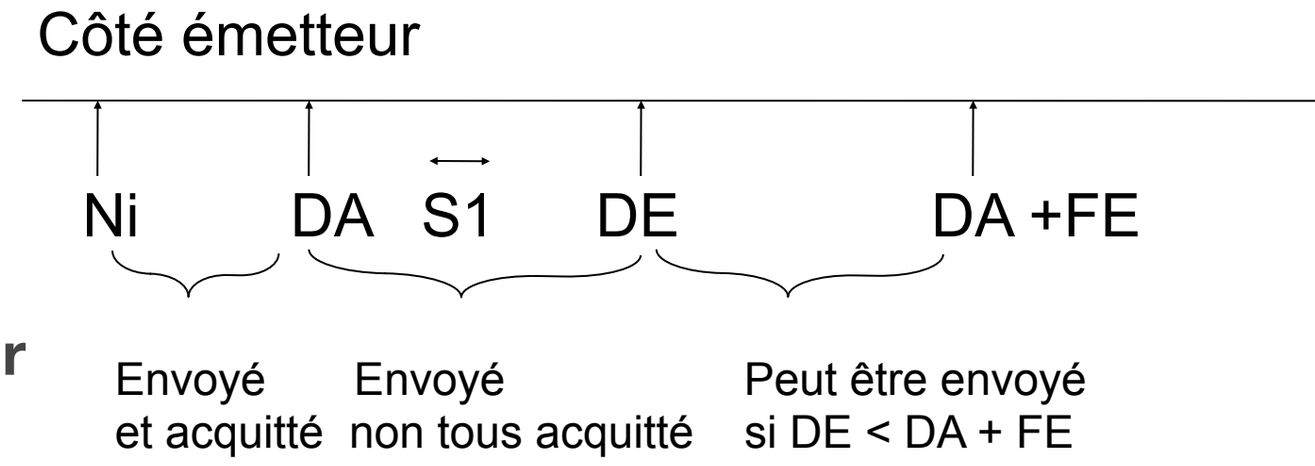
TCP : New Reno & SACK

- **Pertes en rafale** : Problème de performance si plusieurs segments d'une même fenêtre se perdent : quid du ACK partiel ?
- **TCP New Reno**
  - ➡ ne quitte le mode fast-recovery que sur réception d'un ACK pour toute la fenêtre
  - ➡ si réception d'un ACK partiel + n , alors retransmission immédiate du segment suivant le dernier acquitté par cet ACK et réduction de CW -= n
    - ➥ compenser l'augmentation potentielle de CW pendant le mode fast-recovery
- **Retransmission sélective (SACK, RFC 2018)**
  - ✓ négocié à la connexion : option SACK_permitted
  - ✓ si récepteur détecte une perte
    - ➡ N° Seq segment reçu > N° Seq segment attendu
    - ➡ récepteur envoie dup ACK avec
      - ★ option SACK contenant n fois (début, fin) pour n blocs isolés
    - ➡ évite la retransmission continue type GoBackN
    - ➡ permet de déclencher le fast retransmit/recovery

TCP équité

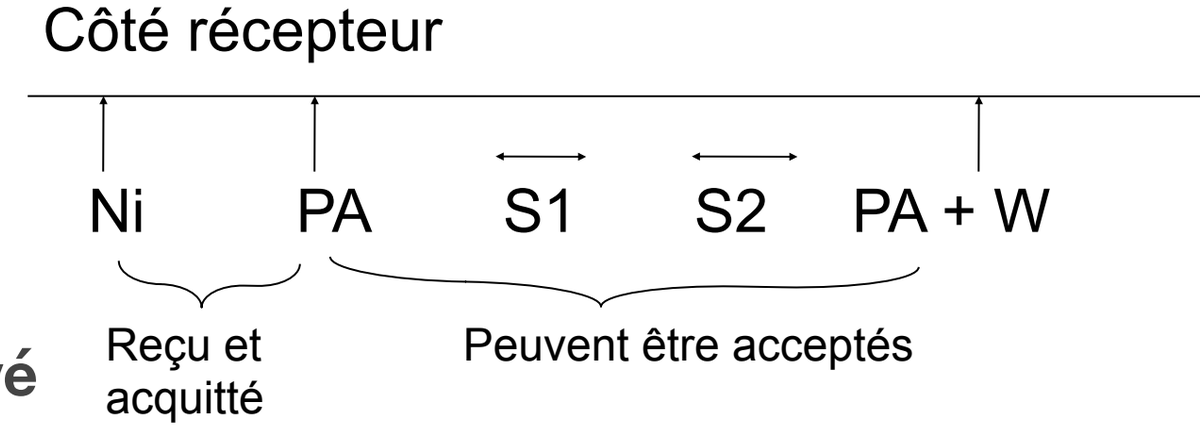
- **En cas de lien congestionné**
  - De débit D
  - Si une seule connexion TCP
    - Débit connexion ≈ D
    - Ex : réseau d'accès ADSL
  - Si n connexions TCP (n > 1)
    - Équité si chaque connexion ≈ D/n
    - En pratique pertes réparties statistiquement
      - => chaque connexion ralentit
    - Mais dépend aussi du RTT
      - RTT longs désavantagés

TCP : bilan coté émetteur



- **Ni : Numéro initial choisi par l'émetteur**
- **DA : dernier N° Ack reçu**
- **S1 : segment acquitté par SACK (0 ou 1 ou plusieurs segments)**
- **DE : dernier octet envoyé. Cas particuliers**
  - DE = DA -1 : aucun octet non acquitté
  - DE = DA + FE : fenêtre pleine, émetteur bloqué
- **FE : fenêtre émission = min(W, CW)**
- **A chaque acquittement nouveau**
  - DA avance, FE est recalculé, RTT re-estimé (et donc RTO recalculé)

TCP : bilan coté récepteur



- **Ni : Numéro initial choisi par l'émetteur**
- **PA : Prochain attendu = dernier N° Ack envoyé**
- **W : Fenêtre réception (fonction des capacités)**
- **S1, S2 blocs reçus dans le désordre (SACK éventuel)**
  - non encore fournis à l'application
- **A chaque segment attendu reçu**
  - PA avance; W est (éventuellement) recalculé; Acquittement éventuellement envoyé
- **A chaque segment non attendu reçu**
  - PA inchangé; stocker segment non attendu; envoyer ACK dupliqué
    - ou SACK

TCP : Synthèse

- **Protocole stable**
  - amélioration algo retransmission/congestion
  - chaque système (Windows, Linux)
    - implémente une ou plusieurs variantes

- **Optimisation pour réseaux très haut débit**
  - Wscale, SACK, Cubic, fastTCP, …

- **Manque d'efficacité si réseau peu fiable**
  - perte ≠ congestion
  - Exemple : réseaux sans fil…par exemple, TCP sur du Wifi