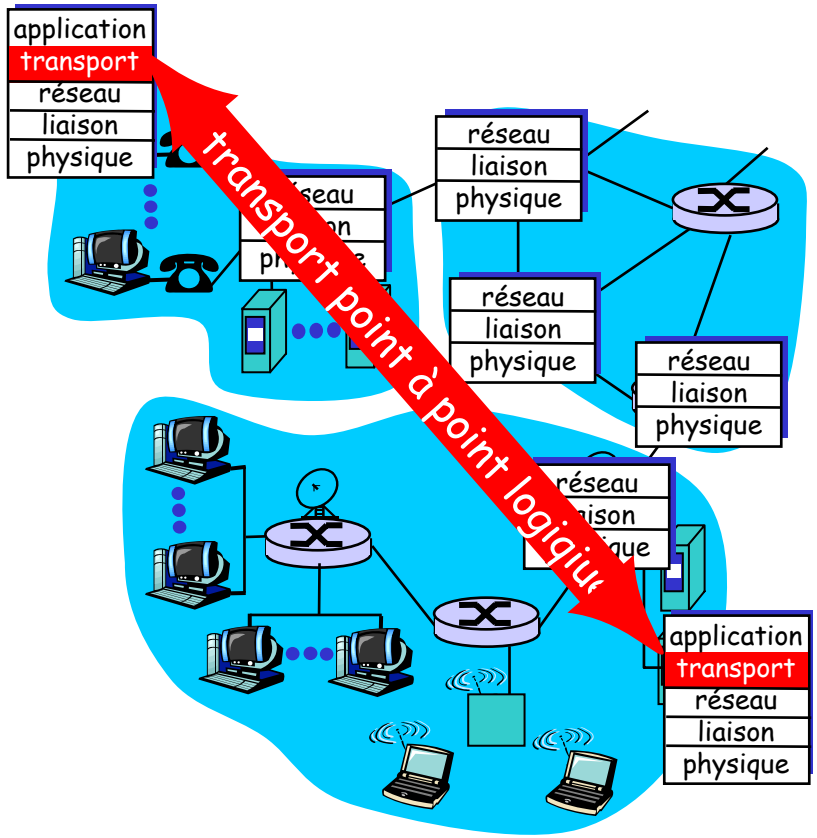
Couche Transport

TCP/UDP

- 1 Services de la couche transport
- 2 Multiplexage et démultiplexage
- 3 Transport sans connexion: UDP
- 4 Principes du transfert fiable de données
- 5 Transport orienté connexion: TCP

Services et protocoles transport

- fournit *une communication logique* entre les processus applicatifs exécutés sur des sites différents
- Les protocoles transport s'exécutent sur les sites
  - émetteur: divise le message en segments, les passe à la couche réseau
  - récepteur: reforme le message à partir des segments obtenus de la couche réseau
- Plus d'un protocole de transport disponible pour les applications
  - Internet: TCP et UDP



Transport vs. réseau

- *Couche réseau:* communication logique entre sites
- *Couche transport :* communication logique entre processus
  - Construit sur les services de la couche réseau
  - Enrichit le service de la couche réseau

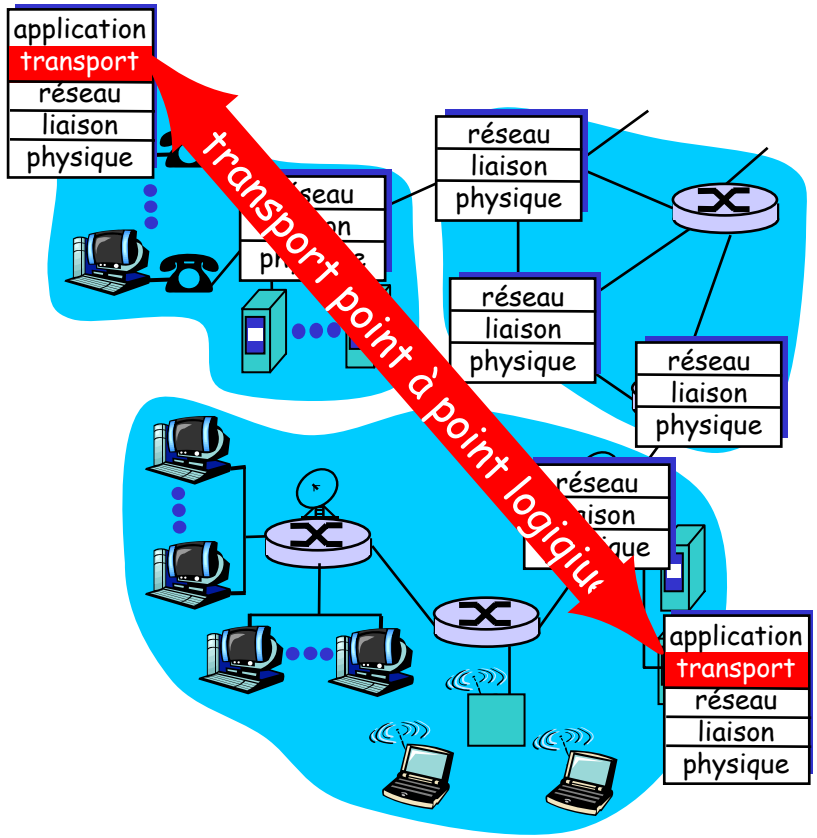
analogie:

5 enfants envoient des lettres à 5 autres enfants

- processus = enfants
- messages appli= lettres dans les enveloppes
- sites = maisons
- protocole transport= Ann et Bill
- Protocole réseau = le service postal

protocoles internet transport

- fiable, délivrance dans l'ordre (TCP)
  - Contrôle de congestion
  - Contrôle de flux
  - Connexion
- Non fiable, dans le désordre: UDP
  - extension du protocole "best-effort" IP
- services non disponibles:
  - garantie des délais
  - garantie largeur de bande



- 1 Services de la couche transport
- 2 Multiplexage et démultiplexage
- 3 Transport sans connexion: UDP
- 4 Principes du transfert fiable de données
- 5 Transport orienté connexion: TCP

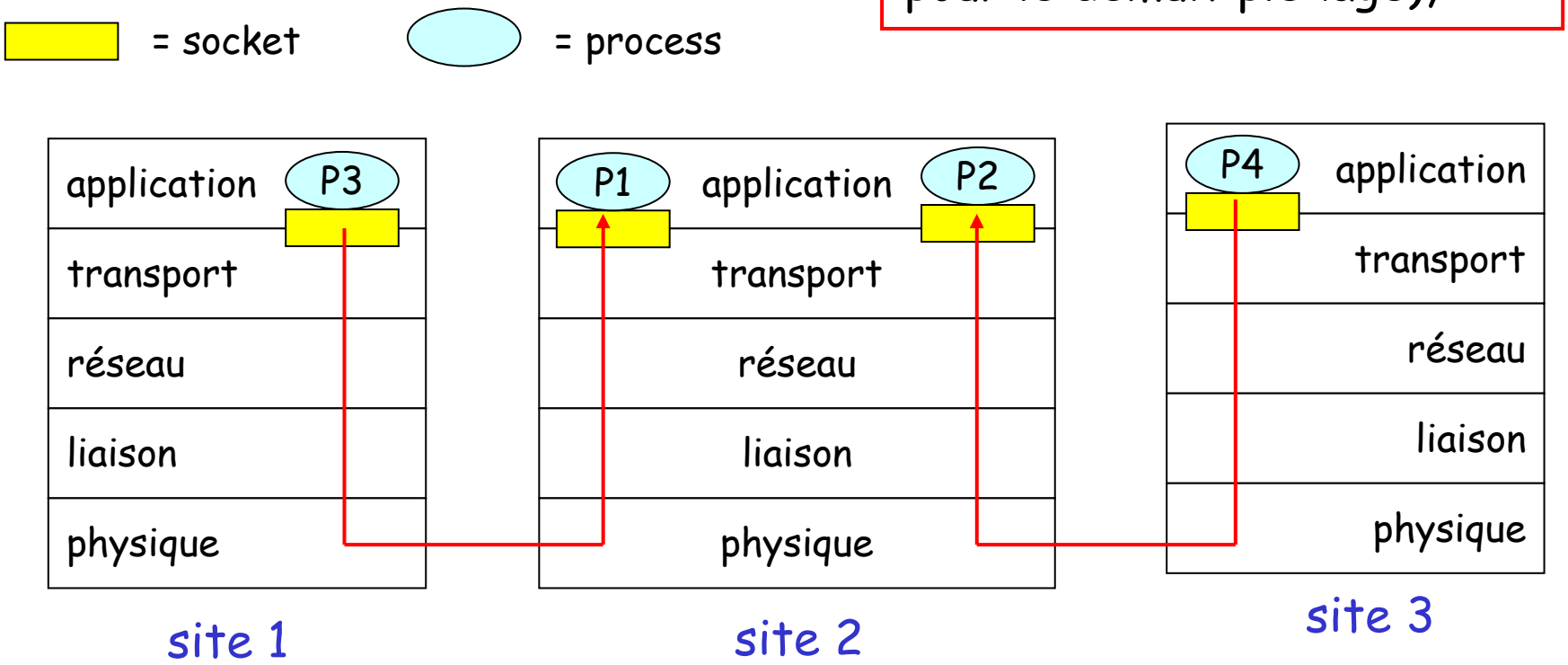
Multiplexage/démultiplexage

Demultiplexage sur le site rcpt :

livre les segments reçus à la bonne socket

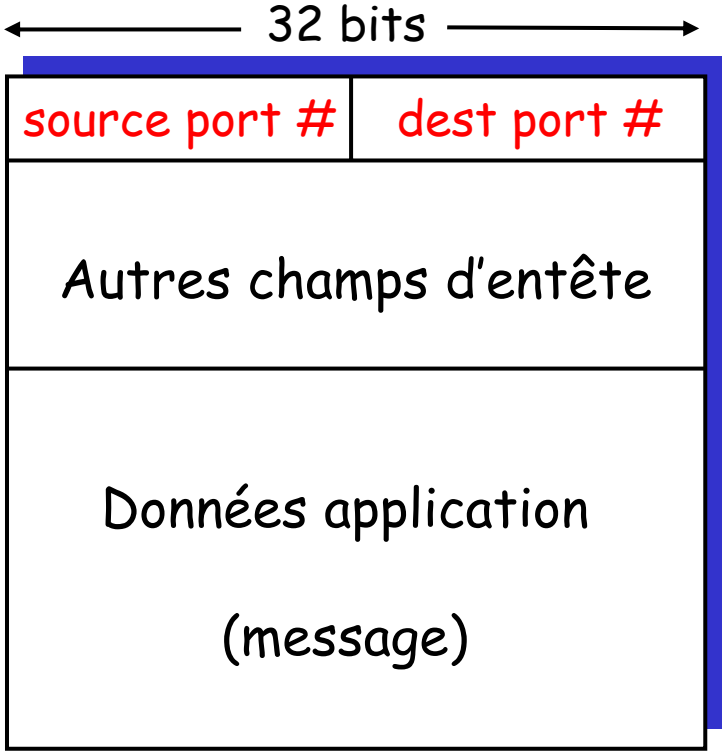
Multiplexage sur le site emetteur:

Assembler les données de plusieurs sockets, ajouter un entête (utilisé ensuite pour le demultiplexage),



Principe du démultiplexage

- Le site reçoit des datagrammes
  - chaque datagramme a l'adresse IP de l'émetteur et du destinataire
  - chaque datagramme (couche réseau) contient un segment (couche transport)
  - chaque segment a le numéro de port de l'émetteur et du destinataire
    (les numéros de port sont spécifiques à chaque applications)
- Les sites utilisent les adresses IP et les numéros de port pour diriger les segments vers la socket appropriée



Format segment TCP/UDP

Démultiplexage sans connexion

- Crée des sockets avec des numéros de port:

```
DatagramSocket mySocket1 = new
   DatagramSocket(99111);
```

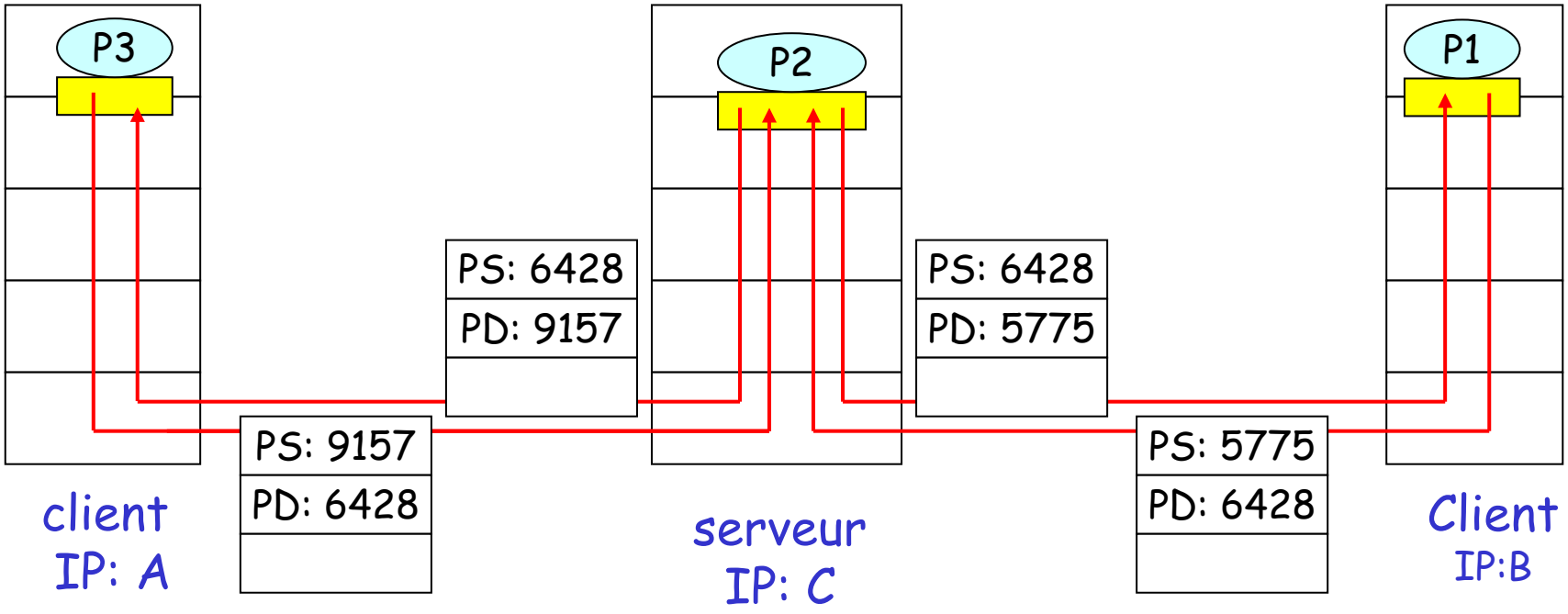
- Socket UDP identifiée par un couple:

(adresse IP dest, numéro port dest)

- Quand un site reçoit un segment UDP:
  - contrôle le numéro de port du dest dans le segment
  - Dirige le segment UDP vers la socket attachée à ce numéro de port
- Les datagrammes IP avec des adresses IP et/ou des numéros de port d'émetteurs différents sont dirigés vers la même socket

Sans connexion

```
DatagramSocket serverSocket = new DatagramSocket(6428);
```

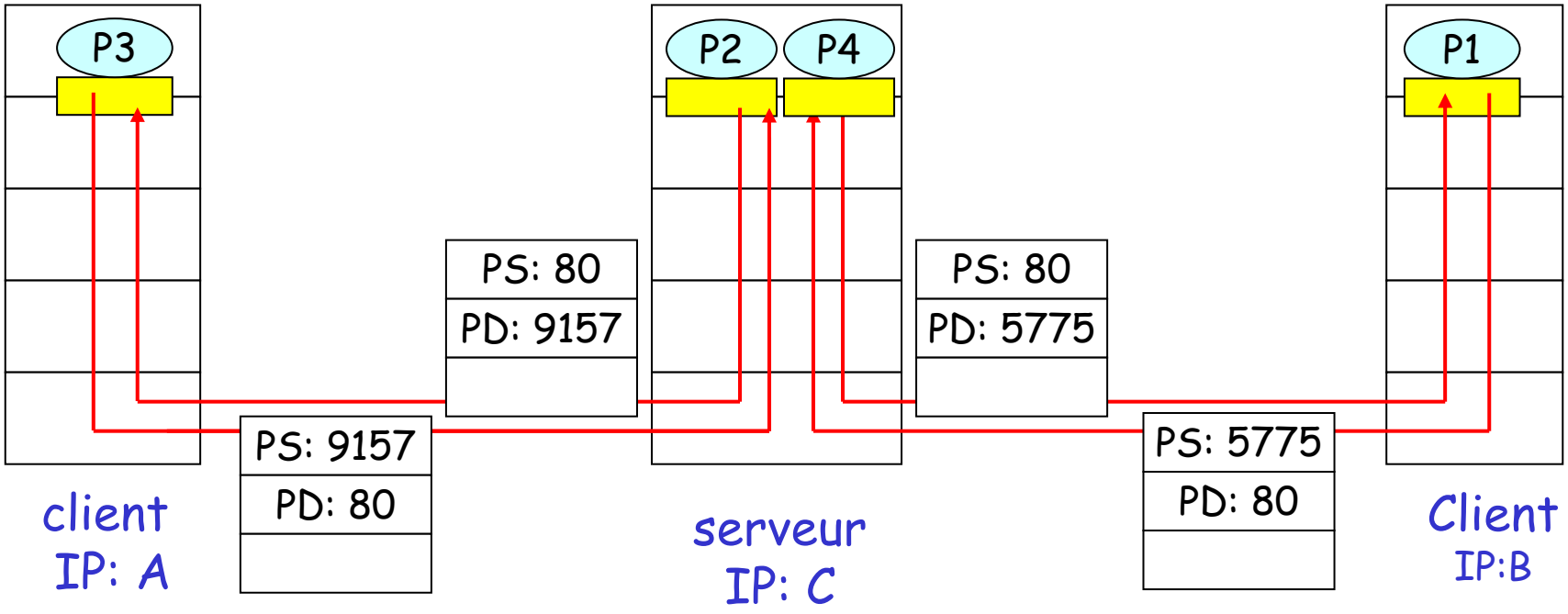


PS fournit "l'adresse de retour"

Démulptiplexage orienté connexion

- Socket TCP identifiée par un 4-uplet:
  - Adresse IP source
  - Numéro de port source
  - Adresse IP dest
  - Numéro de port dest
- Le site rcpt utilise tous ces champs pour diriger le segment vers la socket appropriée
- Un site serveur peut avoir plusieurs sockets TCP simultanément:
  - Chaque socket est identifiée par son propre 4-uplet
- Les serveurs WEB ont des sockets différentes pour chaque client connecté

démultiplexage orienté connexion



- 1 Services de la couche transport
- 2 Multiplexage et démultiplexage
- 3 Transport sans connexion: UDP
- 4 Principes du transfert fiable de données
- 5 Transport orienté connexion: TCP

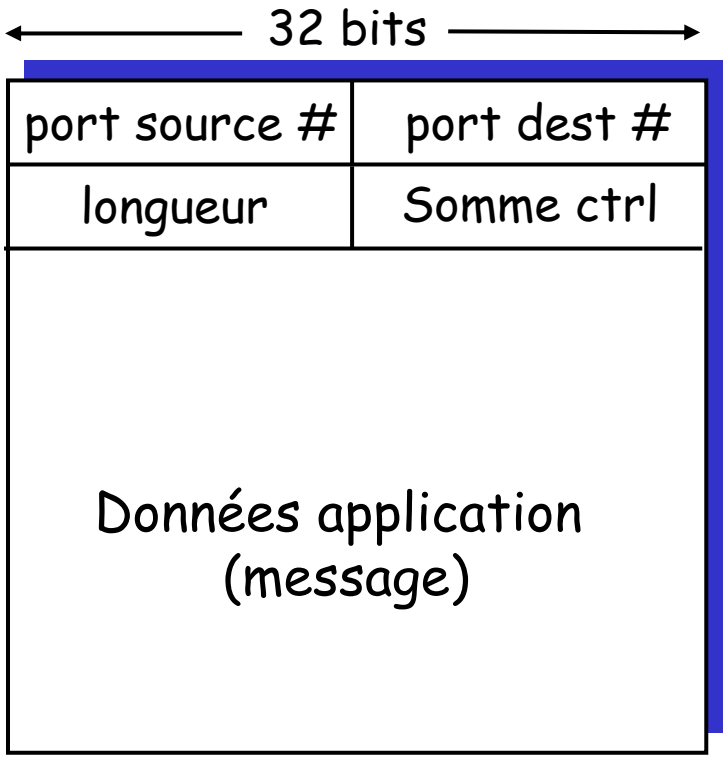
UDP: User Datagram Protocol [RFC 768]

- Protocole de transport simple, sans valeur ajoutée
- Service "best effort", les segments UDP peuvent être:
  - Perdus
  - Livrés dans le désordre aux applications
- *Sans connexion:*
  - Pas de "poignée de main" entre émetteur et récepteur
  - Chaque segment UDP est traité indépendamment des autres

pourquoi UDP?

- Pas d'établissement de connexion (qui ajoute des délais)
- Simple: sans état connexion à la source et au receveur
- Entête de segment petit
- Pas de contrôle de congestion: UDP peut circuler aussi vite que possible

- Utilisé pour les flux d'appli multimedia
  - Tolérantes aux pertes
  - Sensibles à la vitesse
- Autres usages
  - DNS
- Transfert fiable sur UDP: la fiabilité est ajoutée par la couche applicative
  - Correction d'erreur spécifique à l'application



Format de segment UDP

Somme de contrôle UDP

But: détecter les "erreurs" (ex: bits inversés) dans le segment transmis

Emetteur

- Traite le segment comme une séquence d'entiers 16-bit
- Somme de contrôle: addition du contenu du segment
- L'émetteur met la somme de contrôle dans le champ ad hoc du segment UDP

Récepteur:

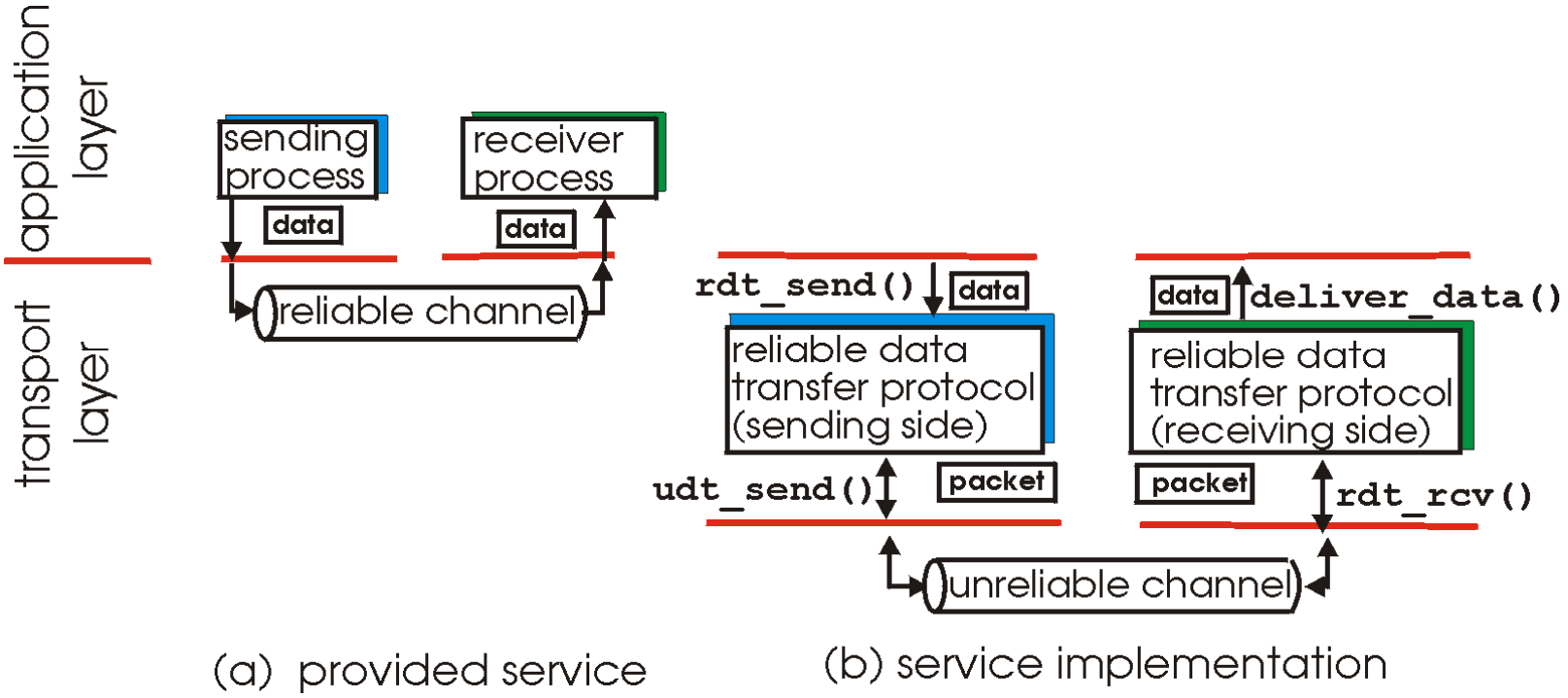
- Calcule la somme de contrôle du segment reçu
- Contrôle si la somme calculée correspond au champ ad hoc du segment reçu:
  - NON - erreur détectée
  - OUI – pas d'erreur détectée

Mais peut être présente ….

- 1 Services de la couche transport
- 2 Multiplexage et démultiplexage
- 3 Transport sans connexion: UDP
- 4 Principes du transfert fiable de données
- 5 Transport orienté connexion: TCP

Principes du transfert fiable

- important dans les couches applications, transport, liaison
- un des sujets importants pour les réseaux



(a) provided service (b) service implementation

- Les caractéristiques du canal non fiable déterminent la complexité du protocole de transfert fiable (reliable data transfer protocol (rdt))

Idée:

- Si le canal peut inverser les bits: somme de contrôle pour le détecter
- *Correction des erreurs*:
  - *Accusé de réception (ACKs):* le receveur dit explicitement que le paquet reçu est OK
  - *Accusé de réception négatif (NAKs):* le receveur dit explicitement qu'il y a une erreur dans le paquet reçu
  - L'émetteur retransmet le paquet si nécessaire

Et si les ACK/NACK sont corrompus?

- L'émetteur n'a plus de retour du récepteur
- S'il retransmet: duplication possible

Que faire?

- L'émetteur ACK/NAK les ACK/NAK du récepteur? Et si ces ACK/NAK sont perdus?
- Retransmettre, même si le paquet a été reçu sans erreur?

Traiter les dupliquas:

- L'émetteur ajoute un numéro de *séquence* à chaque paquet
- L'émetteur retransmet le paquet courant si ACK/NAK corrompu
- Le récepteur ignore (ne livre pas) les paquets reçus en double

stop et attend

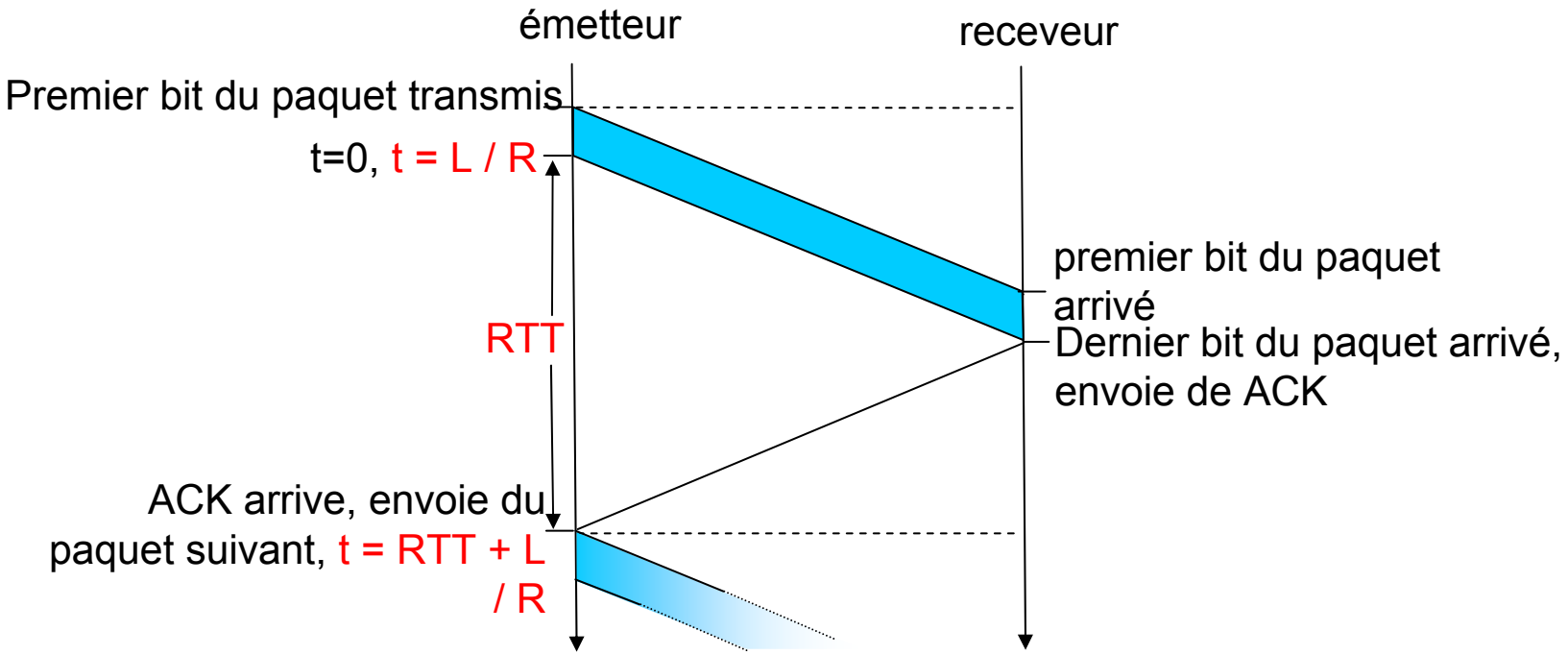
L'émetteur envoi un paquet et attend la réponse du récepteur

Et les pertes?

Approche: l'émetteur attend un temps "raisonnable" l'ACK/NACK

- Retransmission si l'ACK n'est pas reçu à temps
- Si paquet (ou ACK) simplement retardé (et pas perdu):
  - La retransmission créera un dupliqua, l'usage du numéro de séquence évite la livraison multiple
  - Le receveur doit indiquer le numéro de séquence du paquet pour lequel il envoie ACK
- Nécessite l'évaluation du temps

stop-et-attend

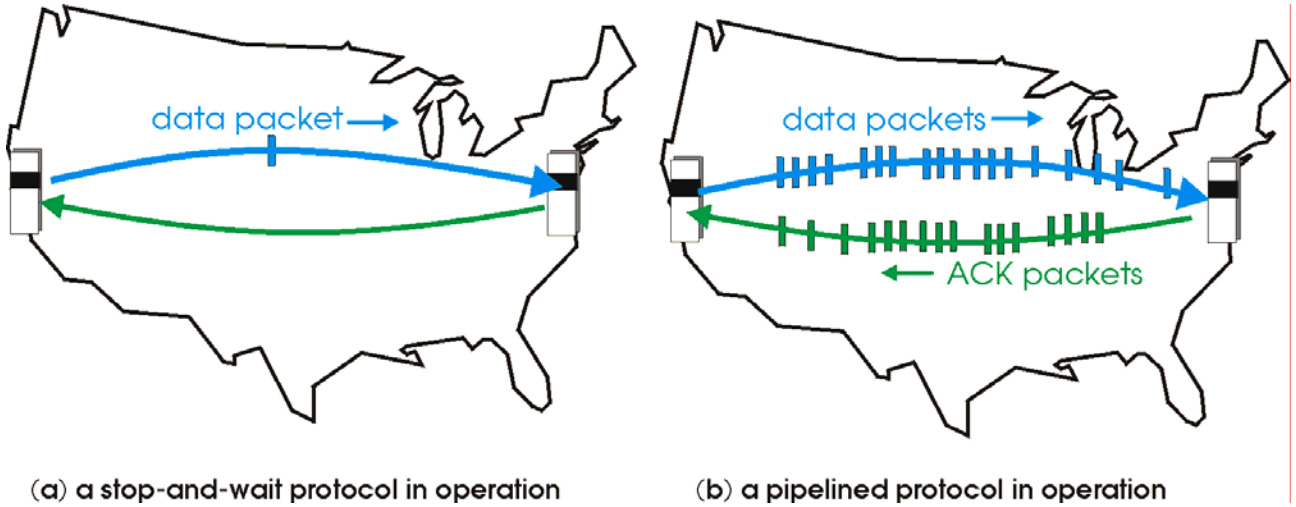


$$U_{\text{émetteur}} = \frac{L / R}{RTT + L / R} = \frac{.008}{30.008} = 0.00027 \text{ microseconds}$$

protocoles en pipeline (avec anticipation)

Pipelining: l'émetteur envoie "avec anticipation" plusieurs paquets, çà acquitter par le récepteur

- La plage des numéros de séquence doit être étendue
- Tampons chez l'émetteur et/ou le récepteur



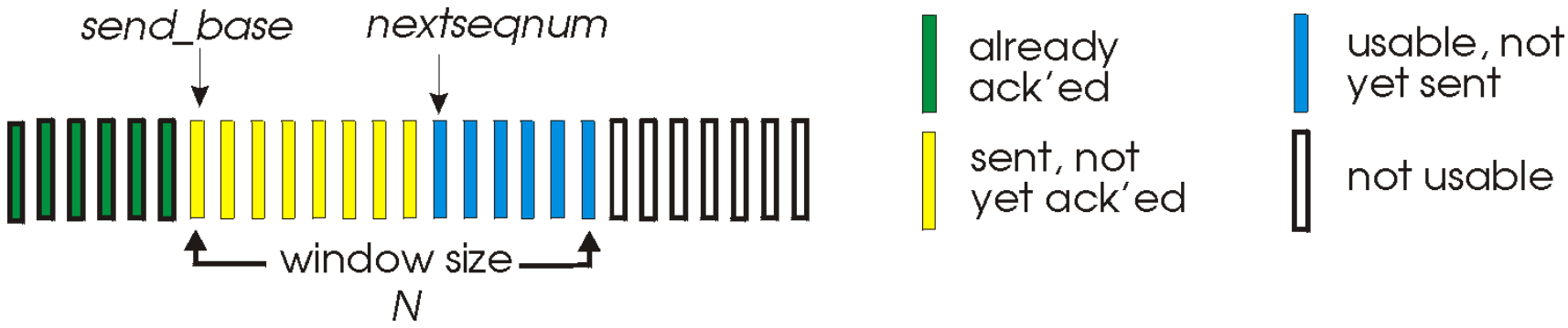
(a) a stop-and-wait protocol in operation (b) a pipelined protocol in operation

- Deux types de protocoles en pipeline: *go-Back-N, selective repeat*

Go-Back-N (fenêtre glissante)

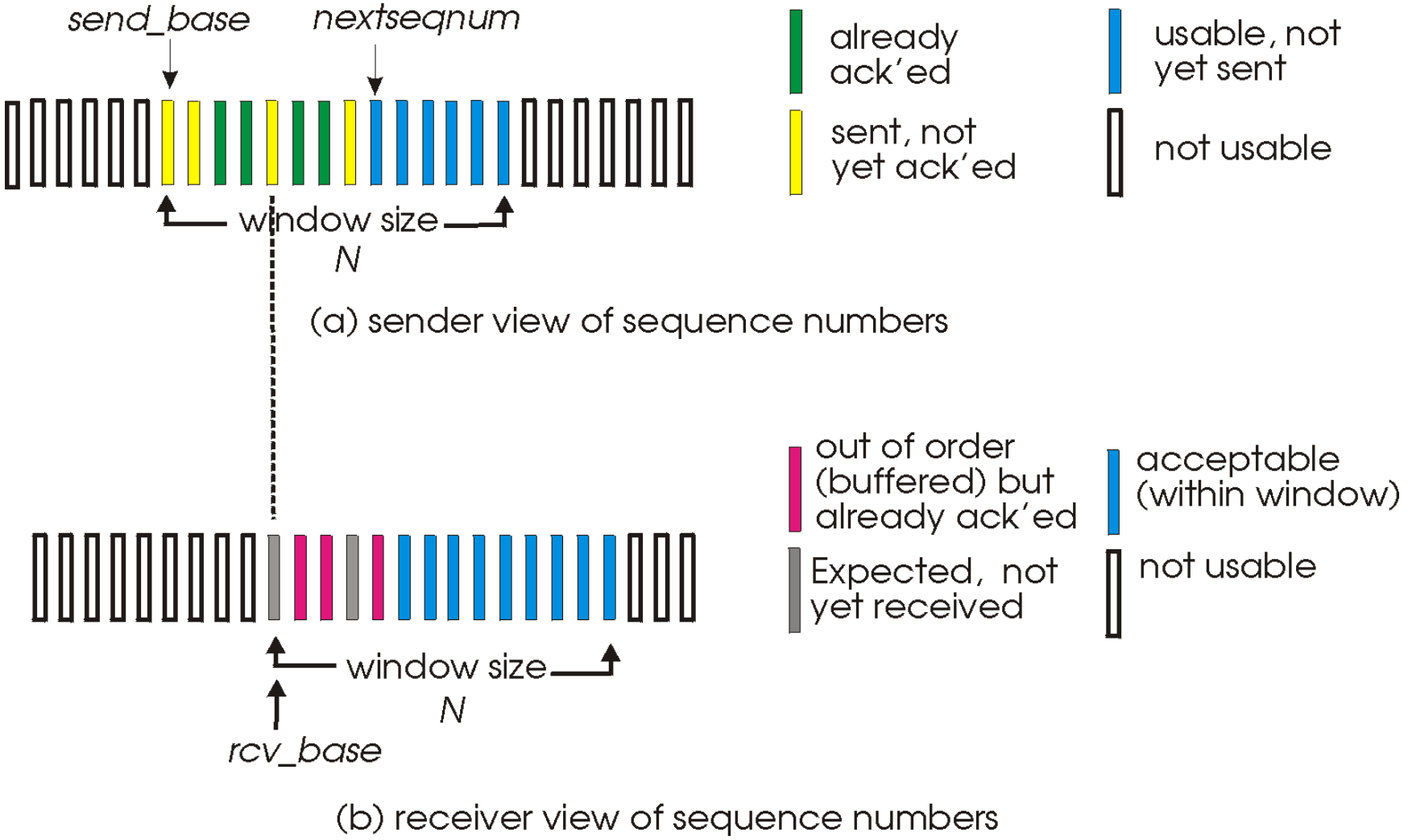
Emetteur

- "fenêtre" jusqu'a N paquets consécutifs non acquittés,



- ACK(n): ACKs acquiite tous les paquets jusqu'à n "ACK cumulatif"
- *timeout(n):* retransmet le paquet n et tous les paquets envoyés ayant un numéro de séquence supérieur à n

Selective repeat: fenêtre chez l'émetteur et le récepteur

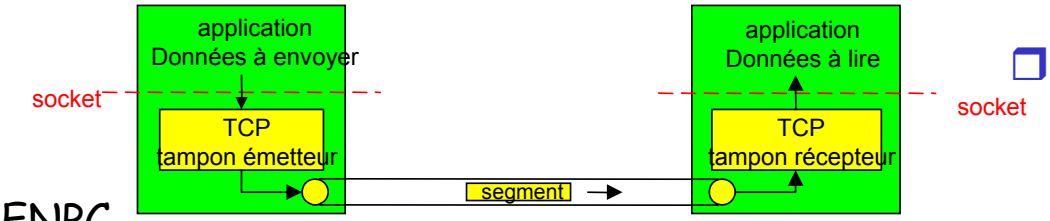


- 1 Services de la couche transport
- 2 Multiplexage et démultiplexage
- 3 Transport sans connexion: UDP
- 4 Principes du transfert fiable de données
- 5 Transport orienté connexion: TCP

TCP: Résumé

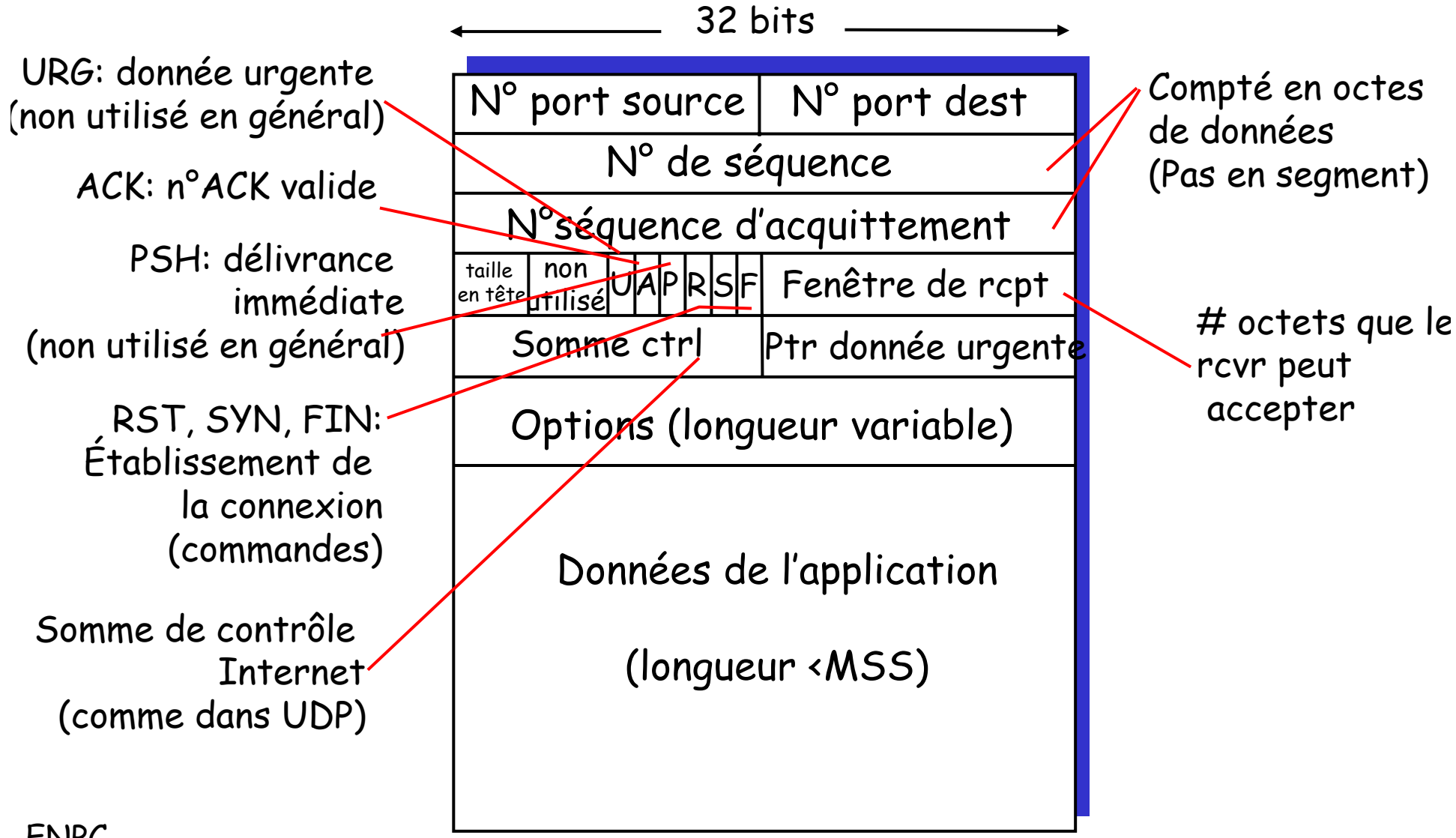
RFCs: 793, 1122, 1323, 2018, 2581

- point à point:
  - Un émetteur,un receveur
- fiable, flux ordonné d'octets
  - Pas de limites des messages
- En pipeline:
  - Largeur de la fenêtre affectée par les contrôles de flux et de congestion
- *tampons: émetteur et récepteur*



- Bi-directionnel:
  - Flux bi-directionnel dans la même connexion
  - MSS: (maximum segment size) taille maximum d'un segment pour les données applicatives
- Orienté connexion:
  - Poignée de main (échange de message de contrôle), initialise états émetteur et récepteur avant l'échange des données
  - Aucun élément de la connexion TCP dans les différents éléments du réseau entre l'émetteur et le rcpt
- Contrôle de flux:
  - L'émetteur ne débordera pas le récepteur

TCP structure d'un segment



TCP: Gestion de la connexion

Rappel: l'émetteur et le récepteur établissent une "connexion" avant l'échange de données

- initialise les variables TCP:
  - N° de seq.
  - tampons, info pour le contrôle de flux (ex: **RcvWindow**)

- *client:* initie la connexion

```
Socket clientSocket = new
 Socket("hostname","port

 number");
```

- *serveur:* contacté par le client

```
Socket connectionSocket =
 welcomeSocket.accept();
```

3-poignée de main (Three way handshake):

Pas 1: l'hôte client envoie un segment TCP SYN au serveur

- Indique son n° de séquence initial
- Pas de données

Pas 2: l'hôte serveur reçoit SYN et répond avec un segment SYNACK

- Le serveur alloue ses tampons
- Spécifie son n° de séquence initial

Pas 3: le client reçoit SYNACK, répond avec un segment ACK qui peut contenir des données

TCP gestion de la connexion

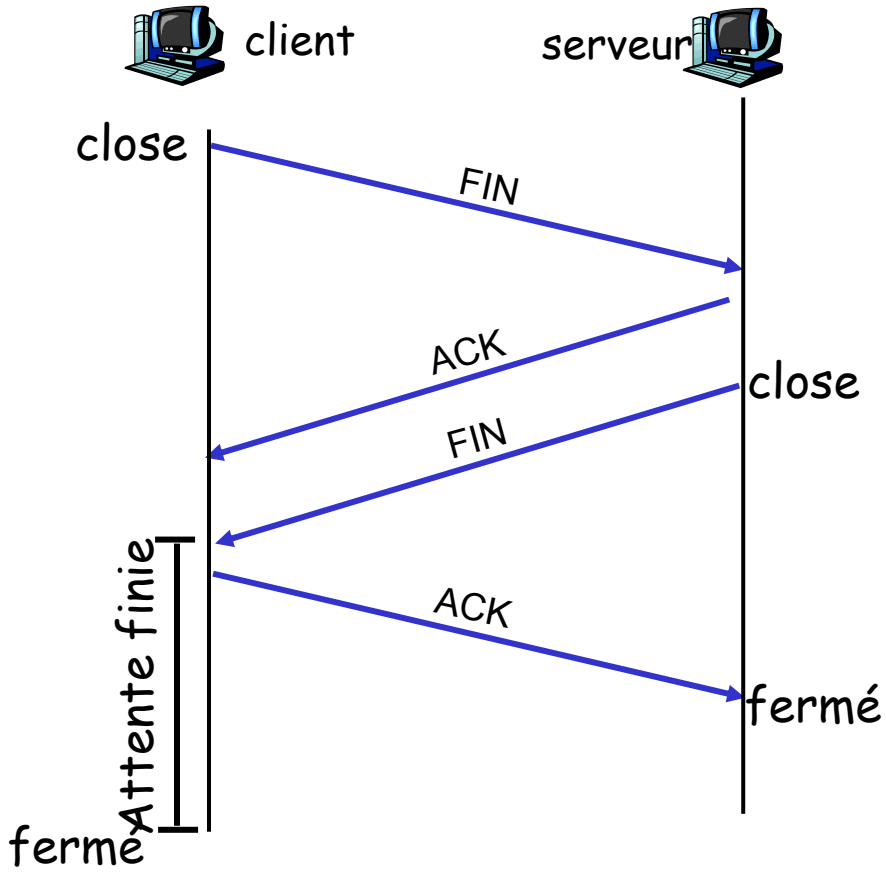
Fermer une connexion:

Le client ferme la socket:

```
clientSocket.close();
```

pas 1: le coté client envoie un segment TCP FIN au serveur

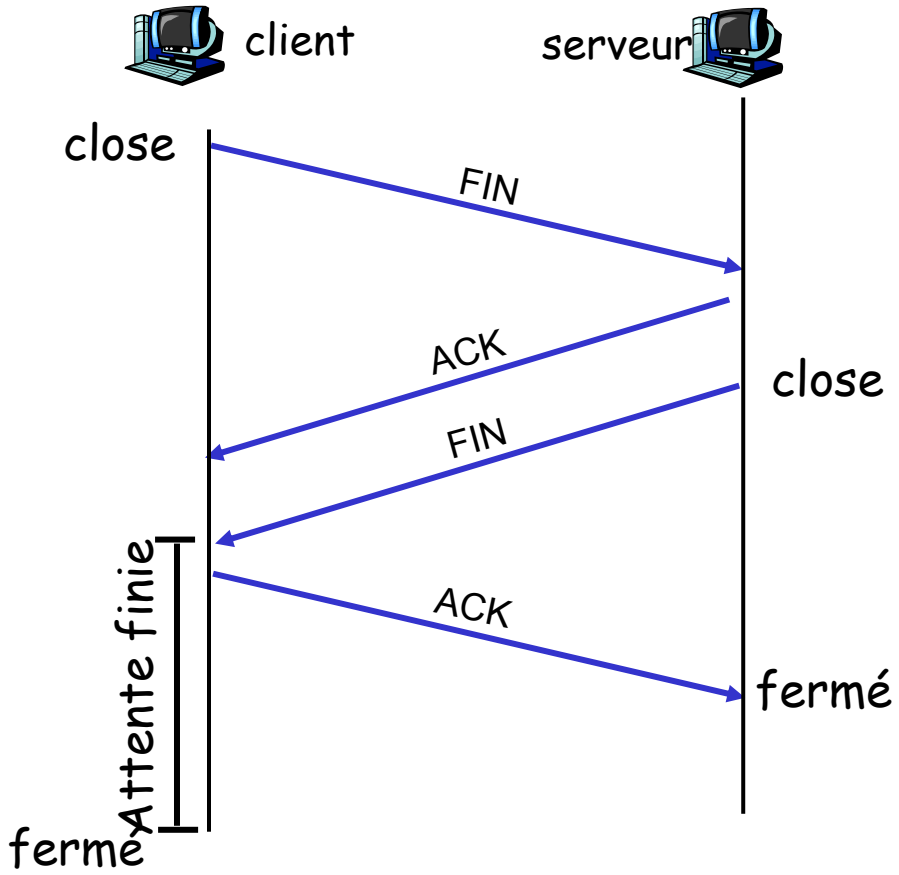
pas 2: le serveur reçoit FIN, répond avec ACK. Il ferme la connexion et envoie FIN.



Pas 3: le client reçoit FIN, répond avec ACK.

- Entre dans une attente finie. Il répondra avec ACK à tout FIN reçu

Pas 4: le serveur reçoit ACK. La connexion est fermée.

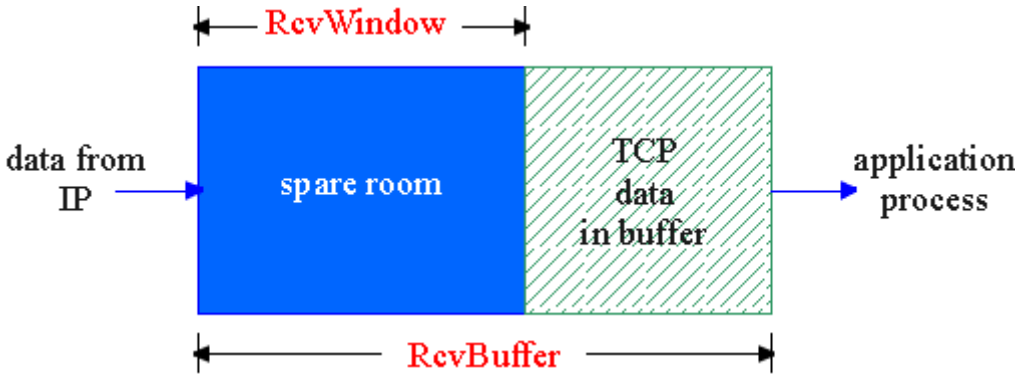


TCP: Transfert fiable

- TCP crée un service fiable au dessus du service non-fiable d'IP
- Protocole en pipeline
- Acks groupés
- TCP utilise un seul timer pour la retransmission
- Retransmissions effectuées si:
  - Un évt timeout
  - Acks dupliqués

TCP: contrôle de flux

- Le coté receveur de TCP a un tampon de réception:



- Le processus applicatif peut être lent dans sa lecture des données du tampon

Contrôle de flux

L'émetteur ne fait pas déborder le tampon de rcpt en transmettant trop vide

- Service d'alignement des débits: aligner le débit de l'émetteur au débit de lecture de l'application réceptrice

Contrôle de congestion

Congestion:

- Informellement: "trop de sources envoient trop de données trop vite pour que le *réseau* qui les gère"
- Différent du contrôle de flux
- Manifestations:
  - Perte de paquets (débordements des tampons dans les routeurs)
  - Délais longs (attente dans les tampons des routeurs)
- Un problème important pour les réseaux

Approches pour le contrôle de la congestion

2 grandes approches à l'égard du contrôle de la congestion:

Gestion par les systèmes terminaux:

- Sans info-en-retour explicite du réseau
- Congestion déduite par les systèmes terminaux en observant les pertes et les délais
- Approche choisie par TCP

Gestion assistée par le réseau:

- Les routeurs donnent de l'info-en-retour aux systèmes terminaux
  - Un seul bit indique la congestion (SNA, DECbit, TCP/IP ECN, ATM)
  - Explicite la vitesse de transmission de l'émetteur