

Couche transport

Module FAR

Polytech Montpellier – IG3

David Delahaye

Protocoles de transport – Plan

1. Rôle du transport
2. Le protocole UDP
3. Le protocole TCP

Protocoles de transport - Plan

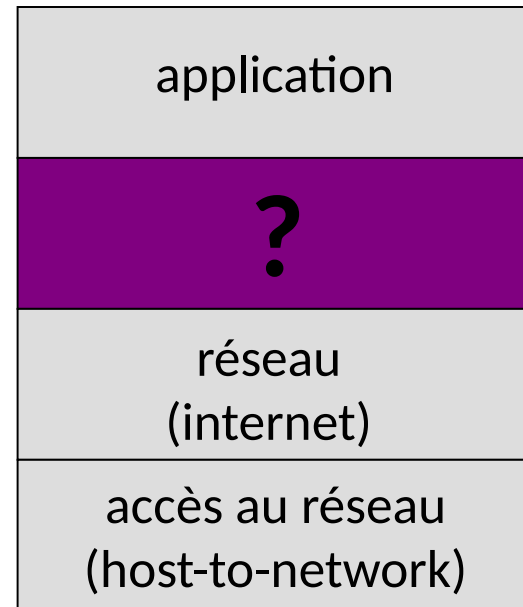
1. **Rôle du transport**
2. Le protocole UDP
3. Le protocole TCP

Problématique

- différentes technologies possibles pour connecter un ensemble de machines

- LAN
- WAN
- inter-réseaux

⇒ service de remise de paquets de machine à machine

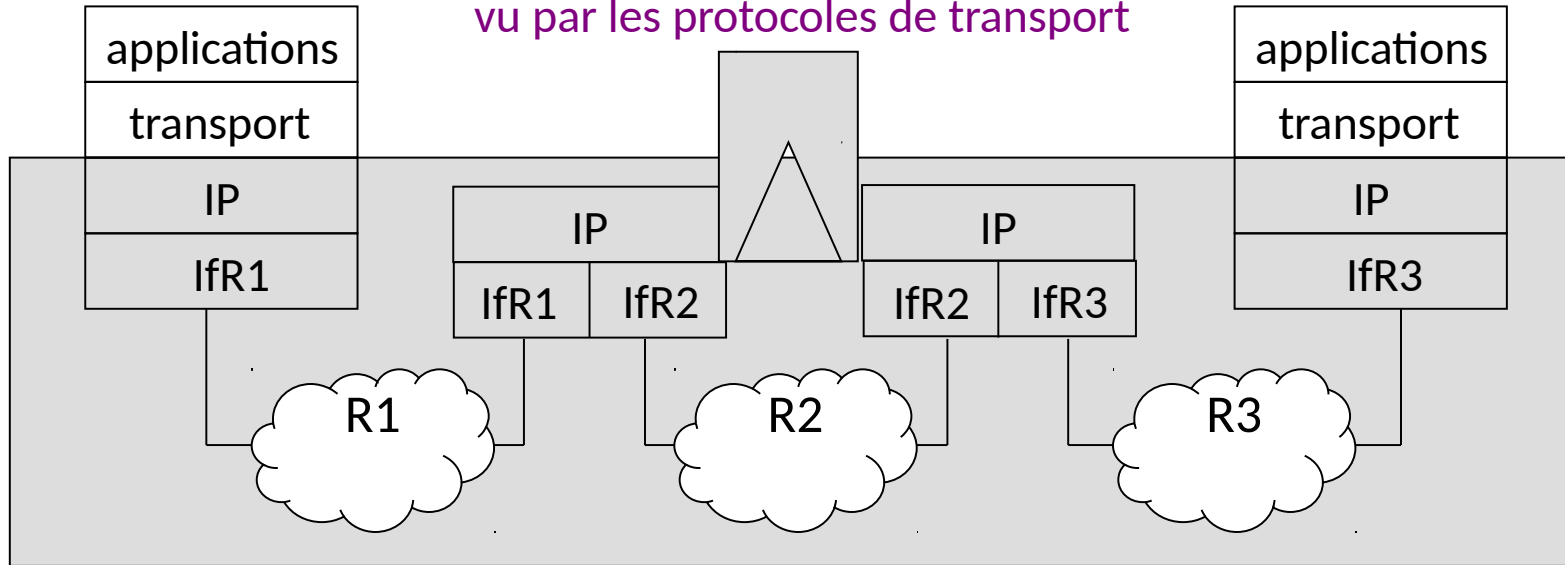


Modèle tcp/ip car 4 couches

- comment passer de ce service à un canal de communication de processus à processus ? (Transport horizontal)

Le système de communication

système de communication
vu par les protocoles de transport



- service fourni par IP
 - routage à travers une interconnexion de réseaux
 - fragmentation/réassemblage
 - service non connecté, *best effort*

Challenges

- prendre en compte le service fourni par la couche réseau
 - pertes de paquets
 - déséquencelements
 - duplications
 - erreurs
 - MTU
 - temps de traversée imprévisibles
 - ...
- prendre en compte les besoins des applications
 - garantie de remise des messages
 - séquencement
 - absence de duplications
 - absence de messages erronés
 - messages de lg quelconque
 - synchronisation entre l'émetteur et le récepteur
 - contrôle de flux par le récepteur sur l'émetteur
 - support de plusieurs applications sur le même hôte
 - ...

Rôle de la couche transport

- transformer les propriétés pas toujours désirables du réseau en un service de haut niveau souhaité par les applications

La couche transport va faire une abstraction (rendre transparent) des caractéristiques et contraintes de la couche réseau vus précédemment.

- plusieurs déclinaisons de protocoles de transport
 - UDP
 - TCP
 - ...

Protocoles de transport - Plan

1. Rôle du transport
2. **Le protocole UDP**
 - le (dé)multiplexage
 - la notion de port
 - le datagramme UDP
 - autres fonctions ?
3. Le protocole TCP

Le protocole UDP

- User Datagram Protocol

Ex : dans un immeuble, il y a plusieurs boîte aux lettres. Il faut donc savoir identifier quelle boîte aux lettres est destinataire.

- se contente d'étendre

- le service de remise d'hôte à hôte à
un service de remise de processus à processus

- Problème

- plusieurs applications peuvent tourner simultanément sur un hôte

⇒ il faut donc pouvoir les identifier de façon non ambiguë

⇒ introduction d'un niveau supplémentaire de **multiplexage**

- cf. le champ type d'Ethernet qui identifie à qui doit être délivré le contenu du champ de données de la trame (**Adresse Mac**)
- cf. le champ protocol de IP qui identifie à qui doit être délivré le contenu du champ de données du datagramme (**Adresse ip**)

Multiplexage/démultiplexage

Rappel : au niveau ip, on dit datagramme. Transport on dit


- Problème
 - comment identifier un processus (une application) ?
- solution 0
 - on peut identifier **directement** un processus sur une machine par son *pid* (*process identifier*)
 - ☹ malheureusement possible que si l'OS est un OS distribué "fermé" qui alloue un *pid* unique à chaque processus

Multiplexage/démultiplexage

- solution Pour différencier les applications d'un même hôte
 - on peut identifier **indirectement** un processus par une référence abstraite (*abstract locater*) appelée **port**
 - un processus source envoie un message sur son port
(Ex : un port pr chat boîte aux lettres)
 - un processus destinataire reçoit le message sur son port
 - port ~ boîte aux lettres
- réalisation de la fonction de (dé)multiplexage
 - champ **port** (de la) **source** du message
 - champ **port** (de la) **destination** du message

Il a dans l'en-tête les 2 champs.

Multiplexage/démultiplexage

- champs *port* codés sur 16 bits
 - 65 535 valeurs différentes \Rightarrow insuffisant pour identifier tous les processus de tous les hôtes de l'Internet
 - les ports n'ont pas une signification globale
 - signification restreinte à un hôte (Signification locale des 65535 valeurs et non pas universel)
 - \Rightarrow un processus est identifié par son port sur une machine donnée
- \Rightarrow clé de démultiplexage de UDP = (port, )

Multiplexage/démultiplexage

- comment un processus connaît-il le port de celui à qui il souhaite envoyer un message ?
 - modèle de communication *émetteur* client/*destinataire* serveur
 - une fois que le client a contacté le serveur, le serveur connaît le port du client
- comment le client connaît-il le port du serveur ?
- le serveur accepte des messages sur un port connu de tous (*well-known port*)
 - ex : pompiers : 18, police : 17, SAMU : 15
 - ex : DNS : 53, Telnet : 23, HTTP : 80

Numéros de port

- 3 catégories
 - ports **well-known** : de 0 à 1023
 - alloués par l'IANA (« Internet Assigned Numbers Authority »)
 - sur la plupart des systèmes, ne peuvent être utilisés que par des processus système (ou root) ou des programmes exécutés par des utilisateurs privilégiés
 - ports **registered** : de 1024 à 49 151
 - listés par l'IANA
 - sur la plupart des systèmes, peuvent être utilisés par des processus utilisateur ordinaires ou des programmes exécutés par des utilisateurs ordinaires

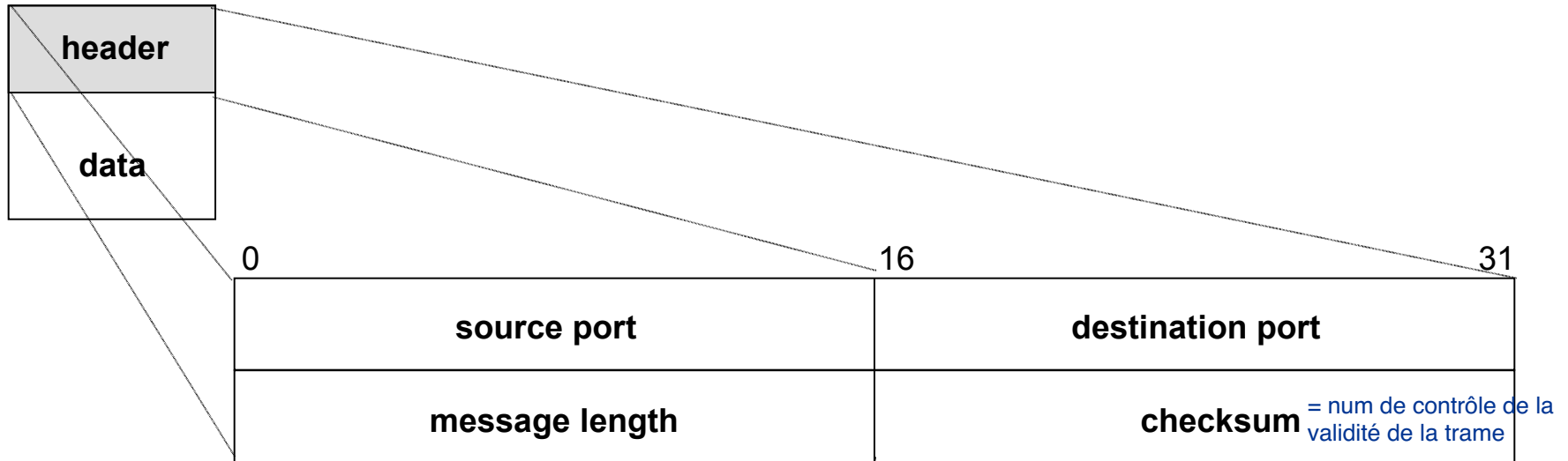
Numéros de port

- ports **dynamic/private** : de 49 152 à 65 535
 - alloués dynamiquement
 - Ces numéros de port privés éphémères sont généralement affectés de façon dynamique à des applications clientes lorsqu'une connexion est initiée.
 - Il est relativement rare pour un client de se connecter à un service par le biais d'un port dynamique ou privé (sauf en Peer to peer).

Les ports *well-known*

<u>No port</u>	<u>Mot-clé</u>	<u>Description</u>
7	ECHO	Echo
11	USERS	Active Users
13	DAYTIME	Daytime
37	TIME	Time
42	NAMESERVER	Host Name Server
53	DOMAIN	Domain Name Server
67	BOOTPS	Boot protocol server
68	BOOTPC	Boot protocol client
69	TFTP	Trivial File Transfert Protocol
123	NTP	Network Time Protocol
161	SNMP	Simple Network Management Protocol

Le datagramme UDP



- fonctionnalités autres que le (dé)multiplexage ?
 - pas de contrôle de flux
 - pas de fiabilité
 - détection d'erreurs ?
 - fragmentation ?

Détection d'erreurs

- UDP effectue du contrôle d'erreur sur la trame elle même. Si un bit de la trame saute, le Checksum sera mauvais et la pile IP le détectera et ne le délivrera pas à l'application.
- UDP n'offre aucune garantie ni d'acheminement ni de séquençement.
- Si un paquet se perd en cours de route ou bien si un paquet arrive avant un autre (déséquençement), ni IP ni UDP ne peuvent le détecter. C'est à l'application de le gérer si elle le veut/peut.

Fragmentation

- en théorie
 - les messages UDP peuvent être fragmentés par IP
- en pratique
 - Message tjr contenu ds un seul segment udp
 - la plupart des applications utilisant UDP limitent leurs messages à 512 octets
 - ⇒ pas de fragmentation
 - ⇒ pas de risque de message incomplet

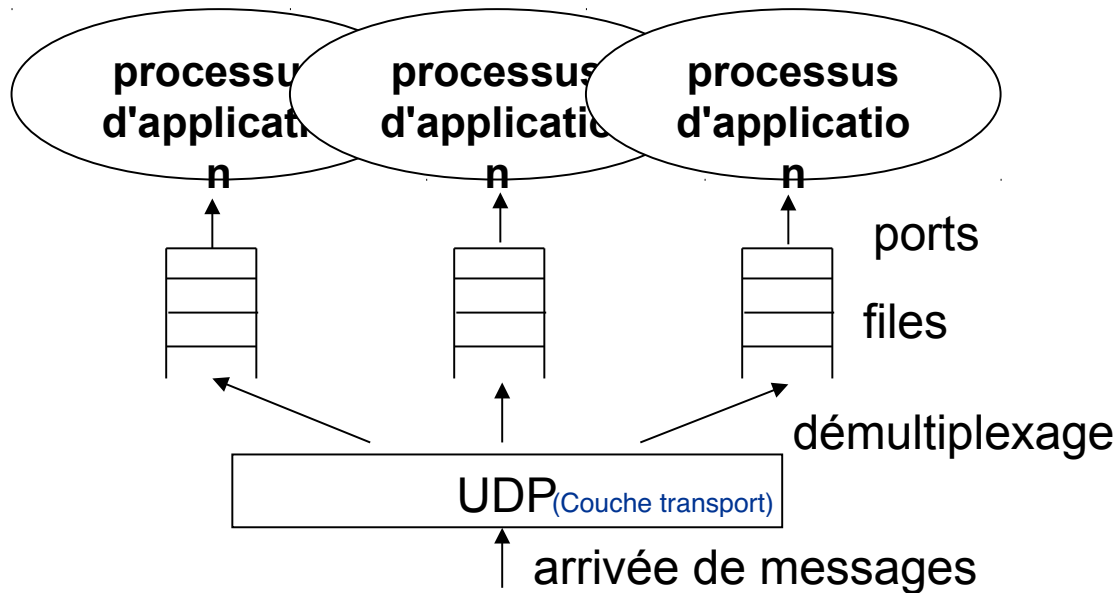
Utilisation

Pour des petits messages et quand on a besoin de recevoir rapidement

- Utilisé quand il est nécessaire soit de transmettre des données très rapidement, et où la perte d'une partie de ces données n'a pas grande importance, soit de transmettre des petites quantités de données, là où la connexion « 3-WAY » TCP serait inutilement coûteuse en ressources.
- Protocoles DHCP/BOOTP, DNS, SNMP, TFTP , xPL.
- Voix sur IP, streaming.
- Jeux en réseau (FPS).

Implémentation des ports

- port = référence *abstraite*
⇒ l'implémentation diffère d'un OS à l'autre !
- en général, un port = une **file de messages**



- quand un msg arrive, UDP l'insère en fin de file
 - si la file est pleine, le msg est rejeté
- quand le processus veut recevoir un msg, il le retire de la tête de la file
 - si la file est vide, le processus se bloque jusqu'à ce qu'un msg soit disponible

Message dans un seul canal puis distribué dans plusieurs canaux (à destination de plusieurs applications) en fct du port

Protocoles de transport - Plan

1. Rôle du transport
2. Le protocole UDP
3. **Le protocole TCP**
 - transport vs. liaison
 - flux d'octets et segments
 - le segment TCP
 - l'établissement de connexion
 - la libération de connexion
 - le contrôle de flux
 - le contrôle d'erreur

Le protocole TCP

- Transmission Control Protocol
- Protocole assez complexe
- 95 % de tout le trafic Internet
- HTTP/HTTPS, SMTP/POP3/IMAP, FTP
- Offre un service de remise
 - en mode connecté
 - fiable
 - Full-duplex (information dans les deux sens)
 - de flux d'octets

Le protocole TCP

- Met en œuvre des mécanismes de
 - (dé)multiplexage
 - gestion de connexions
 - contrôle d'erreur
 - contrôle de flux
 - contrôle de congestion

Comparaison avec une liaison de données

Ettd = terminaux (ordis...)
Etd = équipement (routeur...)

1. gestion des connexions

- la liaison est bâtie sur un canal physique reliant toujours les 2 mêmes ETTD
 - TCP supporte des connexions entre 2 processus s'exécutant sur des hôtes quelconques de l'Internet
- ⇒ phase d'établissement plus complexe

2. le RTT (*Round Trip Time*) est

- pratiquement constant sur une liaison
 - varie en fonction de l'heure de la connexion et de la "distance" séparant les 2 hôtes
- ⇒ dimensionnement du temporisateur de retransmission

Comparaison avec une liaison de données

3. les unités de données

- ne se doublent pas sur le support de transmission
 - peuvent se doubler et peuvent être retardés de façon imprévisible dans le réseau Car réseau maillé (routeurs partout)
- ⇒ TCP doit prévoir le cas de (très) vieux paquets réapparaissent

4. les buffers de réception

- sont propres à *la* liaison
 - sont partagés entre toutes les connexions ouvertes
- ⇒ TCP doit adapter le mécanisme de contrôle de flux

Comparaison avec une liaison de données

5. une congestion

- du lien sur une liaison de données ne peut pas se produire sans que l'émetteur ne s'en rende compte

Si les paquets ne peuvent pas arriver à destination ds le protocole tcp, l'émetteur doit alors être prévenu

- du réseau peut se produire

⇒ TCP va mettre en œuvre un contrôle de congestion

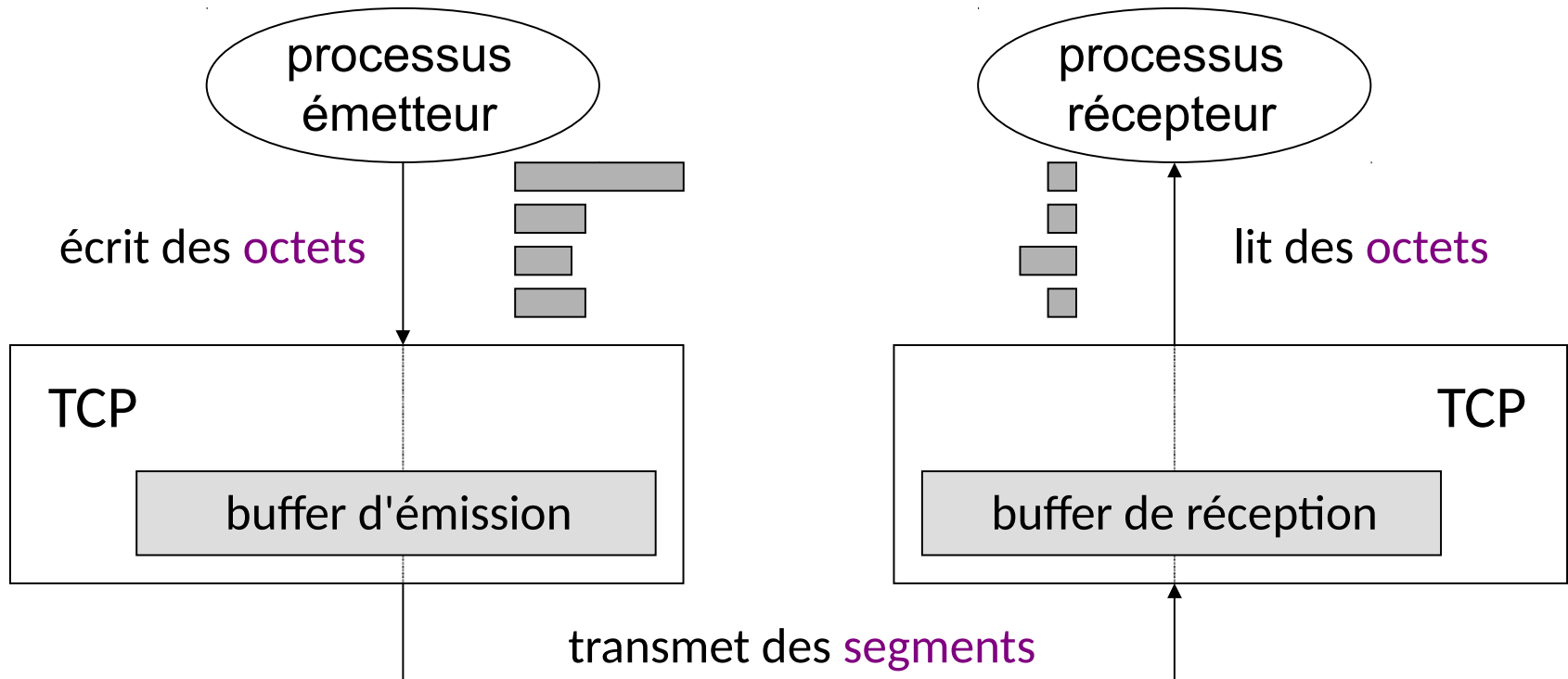
Byte-oriented

- TCP est orienté flux d'octets
 - le processus émetteur « écrit » des octets sur la connexion TCP
 - le processus récepteur « lit » des octets sur la connexion TCP

Byte-oriented

- TCP ne transmet pas d'octets individuels
 - en émission
 - TCP bufferise les octets jusqu'à en avoir un nombre raisonnable = il va attendre d'avoir assez d'octet pour faire le transfert.
 - TCP fabrique un **segment** et l'envoie
 - en réception
 - TCP vide le contenu du segment reçu dans un buffer de réception
 - le processus destinataire vient y lire les octets à sa guise

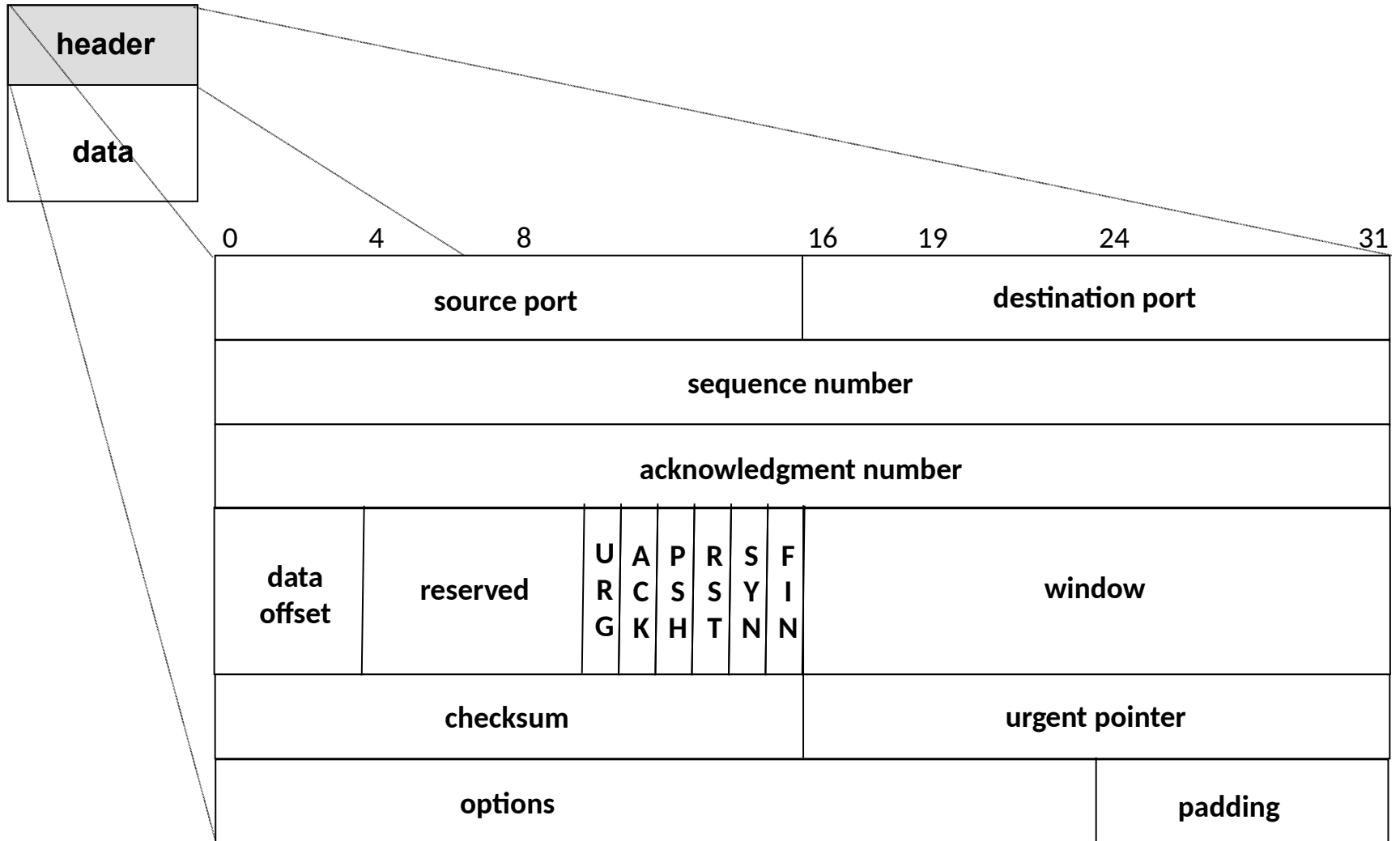
Byte-oriented



Construction d'un segment

- Quand est-ce que TCP décide d'envoyer un segment ?
 1. il a MSS (MaximumSize Segment) octets de données à envoyer
 - $MSS = MTU - \text{en-tête IP} - \text{en-tête TCP}$
MTU = Taille maximale de la trame ethernet
 2. le processus lui demande explicitement D'envoyer le segment quelque soit sa taille
 - fonction *push*
 3. le temporisateur expire
 - pour éviter d'attendre trop longtemps MSS octets

Le segment TCP



options

Les champs de l'en-tête TCP

- **source port** : identifie le processus source sur la machine source
- **destination port** : identifie le processus destinataire sur la machine destinataire
- **sequence number** : N° du 1er octet de données du segment (sauf si SYN=1 : ISN)
- **acknowledgment number** : acquitte tous les octets de données de N° strictement inférieur
- **data offset** : lg de l'en-tête en mots de 32 bits
- **reserved** : 6 bits à 0
- **URG** : mis à 1 pour signaler la présence de données urgentes
- **ACK** : mis à 1 pour indiquer que le champ acknowledgment number est significatif
- **PSH** : mis à 1 pour signaler la fin d'un message logique (push)
- **RST** : mis à 1 pour réinitialiser la connexion (panne, incident, segment invalide)
- **SYN** : mis à 1 pour l'établissement de la connexion
- **FIN** : mis à 1 pour fermer le flux de données dans un sens
- **window** : # d'octets de données que le destinataire du segment pourra émettre
- **checksum** : obligatoire, calculé sur la totalité du segment et sur le pseudo en-tête
- **urgent pointer** : pointe sur la fin (comprise) des données urgentes
- options : MSS, ...
- **padding** : alignement de l'en-tête sur 32 bits

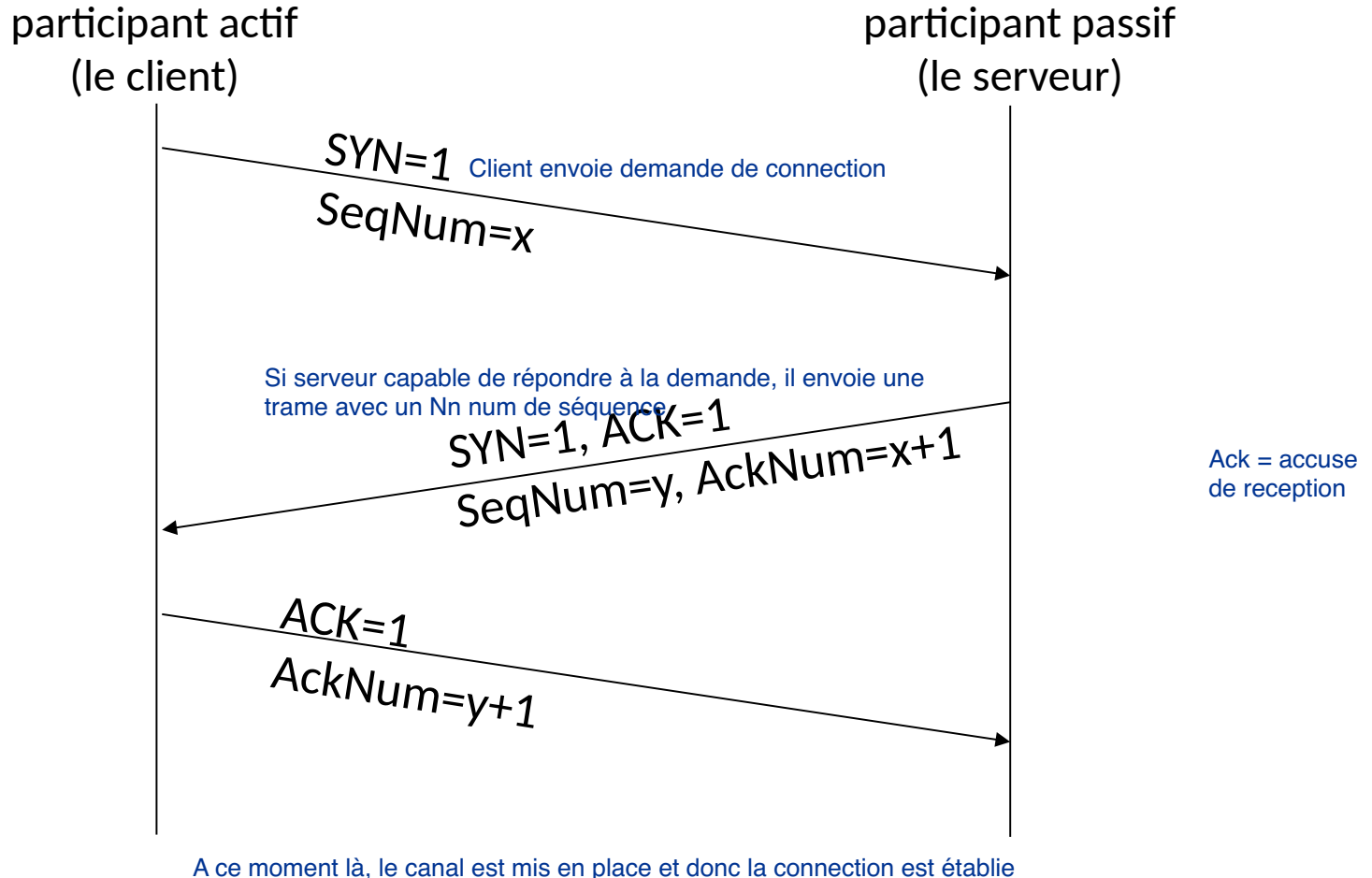
Les ports *well-known*

No port Mot-clé Description

20	FTP-DATA	File Transfer [Default Data]
21	FTP	File Transfer [Control]
23	TELNET	Telnet
25	SMTP	Simple Mail Transfer
37	TIME	Time
42	NAMESERVER	Host Name Server
43	NICNAME	Who Is
53	DOMAIN	Domain Name Server
79	FINGER	Finger
80	HTTP	WWW
110	POP3	Post Office Protocol - Version 3
111	SUNRPC	SUN Remote Procedure Call

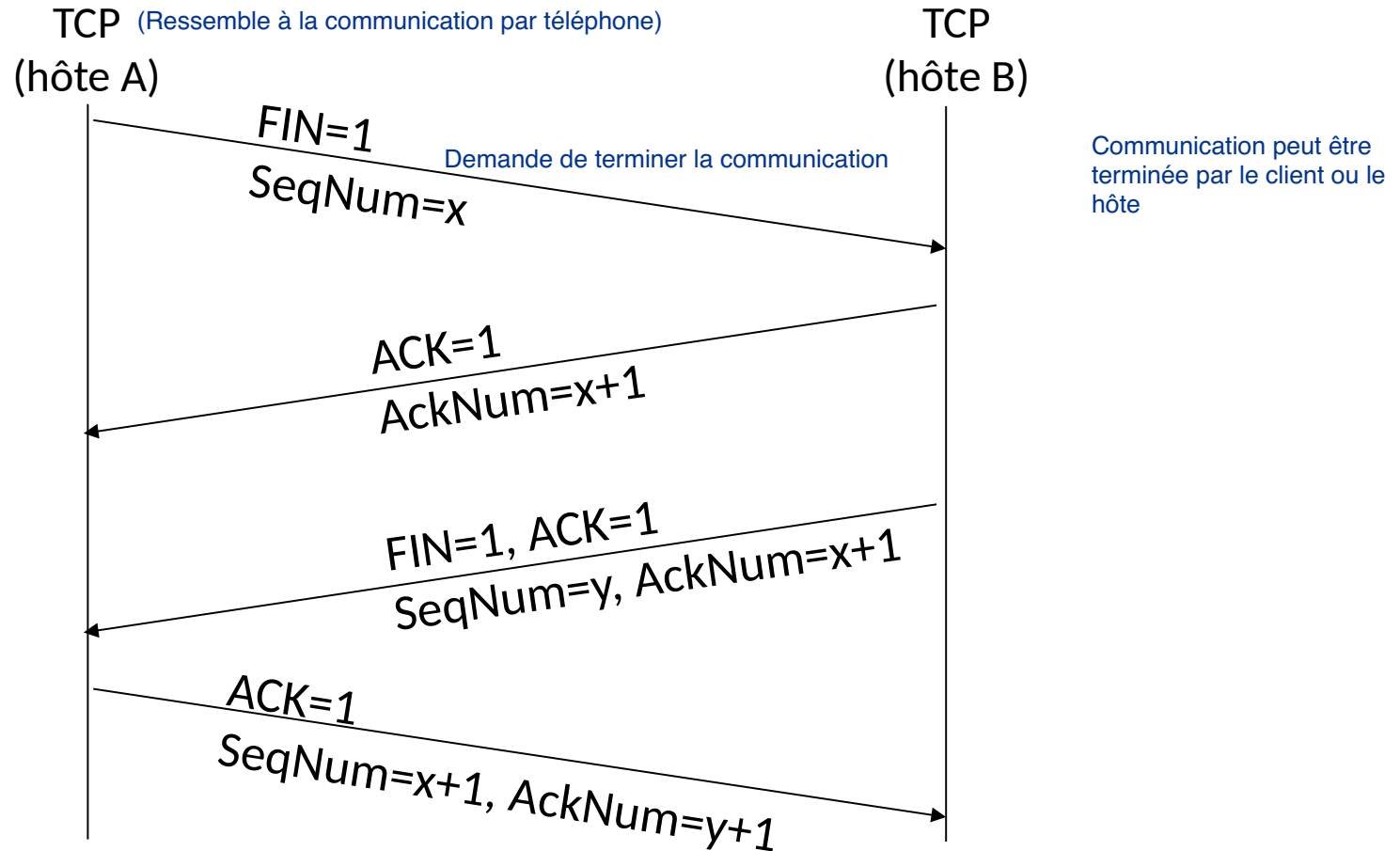
L'établissement de connexion

- les 3 segments échangés



La libération de connexion

- les 2 sens de transmission sont fermés séparément



Transfert de données

- TCP assure un transfert de bout en bout **fiable**

⇒ Contrôle de flux

⇒ Contrôle d'erreur

Transfert de données

Ici seq= seqNum et ack = ackNum

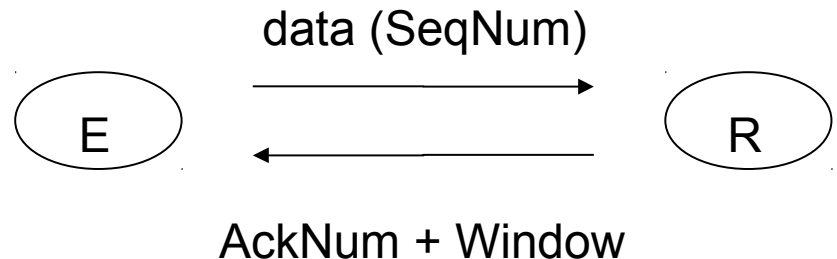
- Exemple de transfert de données
 - Hôte A → hôte B : un octet, avec Seq = 43 et Ack = 79 ;
 - Hôte B → hôte A : segment ACK avec Seq = 79 et Ack = $43 + 1 = 44$;
 - Hôte A → hôte B : segment ACK avec Seq = 44 et Ack = $79 + 1 = 80$.

Transfert de données

- Piggybacking : utiliser le même segment pour envoyer des données et acquitter des données reçues.
- Avantages : améliore l'efficacité, meilleure utilisation de la bande passante.
- Inconvénients : le récepteur conditionne l'envoi du ACK à l'envoi de données. Si pas de données à envoyer, il envoie un ACK au bout d'un certain temps (« Receiver timeout »).

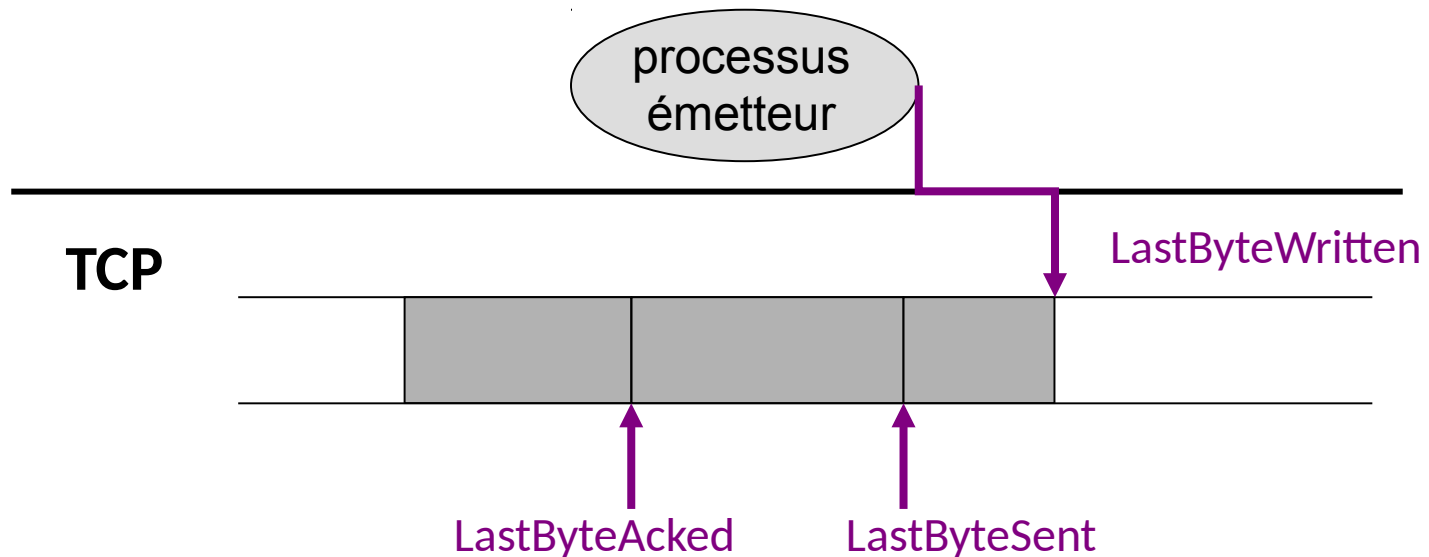
Le contrôle de flux

- une entité réceptrice TCP dispose de ressources (buffers) en nombre limité
 - la taille de la fenêtre reflète la disponibilité des buffers de réception
Fenêtre de taille variable : s'élargir quand nouveaux messages arrivent et rétrécit quand libération messages.
 - une entité TCP gère un nombre de connexions variable
- ⇒ fenêtre dynamique
- progression de la fenêtre par acquittement et crédit
 - champ *window*
 - indique le # d'octets de données que l'entité est prête à recevoir



Buffer d'émission

- en émission, un buffer stocke
 - les données envoyées et en attente d'acquittement
 - les données passées par le processus émetteur mais non encore émises



Buffer d'émission

Ici c'est le numéros d'octets cad leur position dans le message d'origine

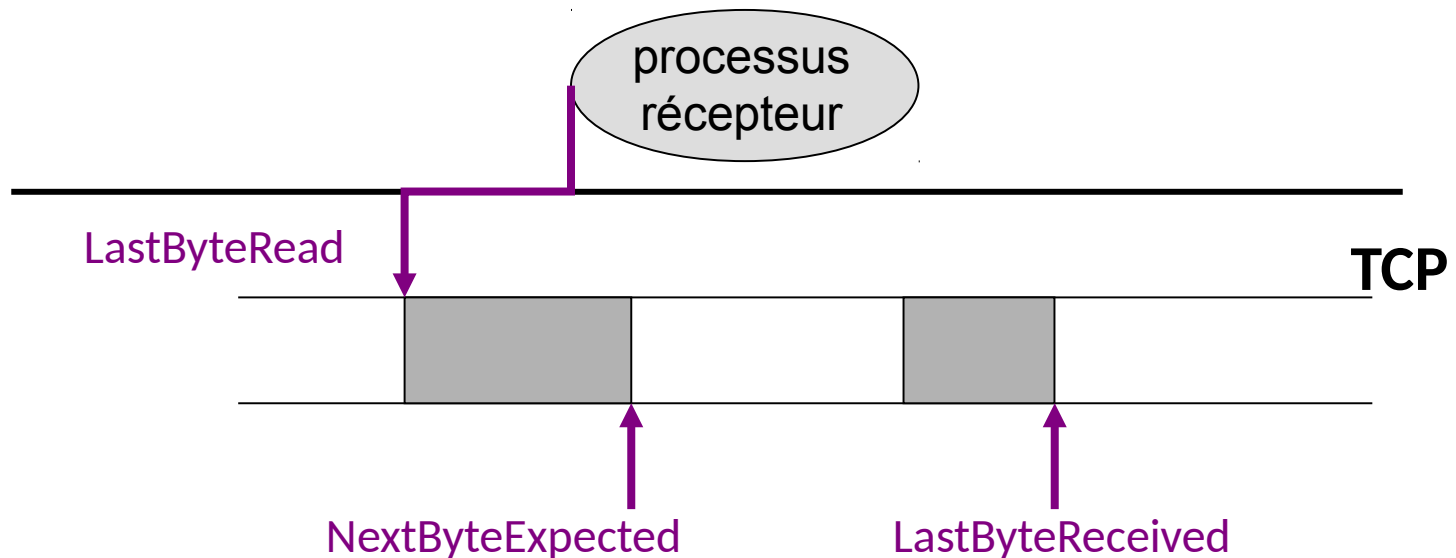
- remarque : on a toujours
 - $\text{LastByteAcked} \leq \text{LastByteSent} \leq \text{LastByteWritten}$
- TCP vérifie à tout moment que
 - $\text{LastByteSent} - \text{LastByteAcked} \leq \text{AdvertisedWindow}$
= taille de la fenêtre
- TCP calcule une fenêtre effective
 - $\text{EffectiveWindow} \leftarrow \text{AdvertisedWindow} - (\text{LastByteSent} - \text{LastByteAcked})$

Buffer d'émission

- en parallèle, TCP doit s'assurer que le processus émetteur ne sature pas son buffer
 - si le processus veut écrire y octets et que $(\text{LastByteWritten} - \text{LastByteAcked}) + y > \text{MaxSendBuffer}$
TCP bloque l'écriture

Buffer de réception

- en réception, un buffer stocke
 - les données dans l'ordre non encore lues par le processus récepteur
 - les données déséquencées



Buffer de réception

- remarque : on a toujours
 - $\text{LastByteRead} < \text{NextByteExpected} \leq \text{LastByteReceived} + 1$
- TCP vérifie à tout moment que
 - $\text{MaxRcvBuffer} \geq \text{LastByteReceived} - \text{LastByteRead}$
- TCP communique une fenêtre mise à jour
 - $\text{AdvertisedWindow} \leftarrow \text{MaxRcvBuffer} - (\text{LastByteReceived} - \text{LastByteRead})$

Buffer de réception

- au fur et à mesure que les données arrivent
 - TCP les acquitte si tous les octets précédents ont été reçus
 - LastByteReceived glisse vers la droite \Rightarrow rétrécissement possible de la fenêtre

Réouverture de fenêtre

- Problème
 - la fenêtre peut avoir été fermée par le récepteur
 - un ACK ne peut être envoyé que sur réception de données (approche *smart sender/dumb receiver*)

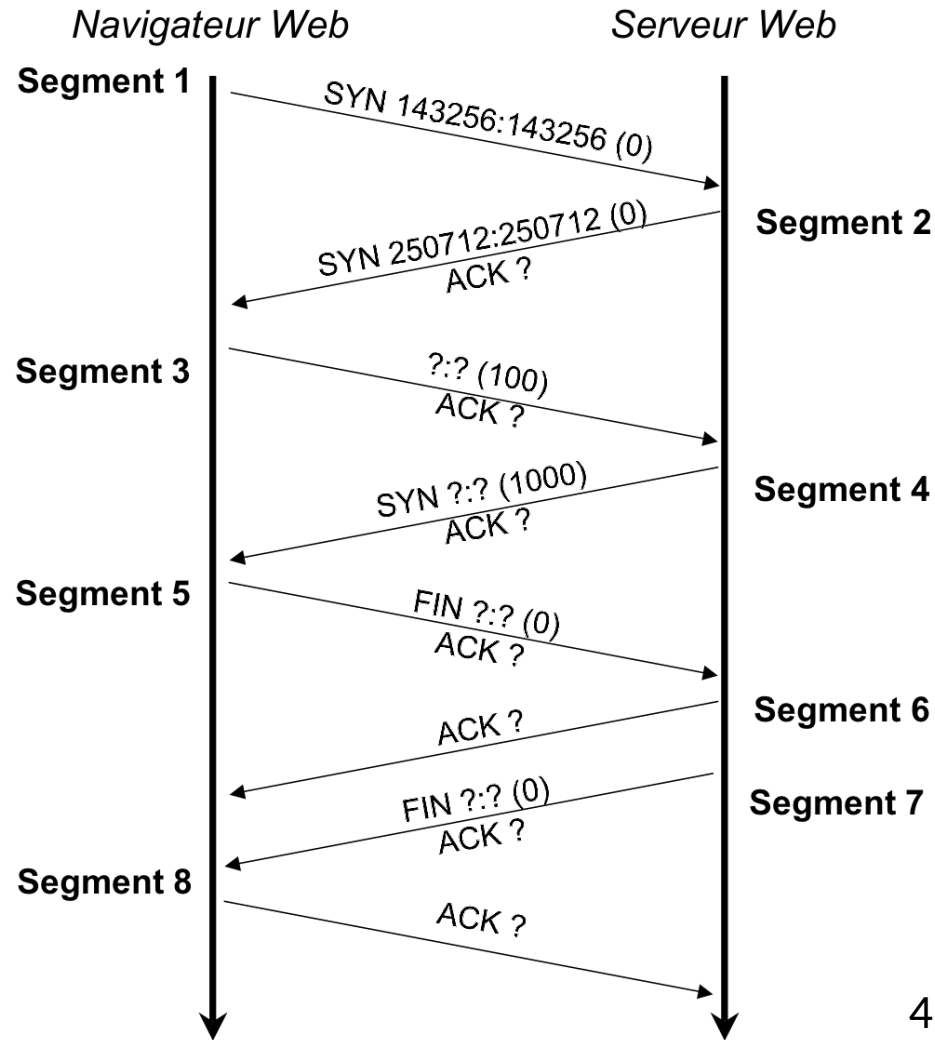
⇒ comment l'émetteur peut-il se rendre compte de la réouverture de la fenêtre ?
- solution
 - lorsque l'émetteur reçoit une AdvertisedWindow à 0, il envoie périodiquement un segment avec 1 octet de données pour provoquer l'envoi d'un ACK

Contrôle d'erreur

- repose sur
 - le champ Checksum
 - le champ SequenceNumber
 - des acquittements positifs (*dumb receiver* \Rightarrow pas d'acquittements négatifs)
 - un temporisateur de retransmission
 - des retransmissions
 - RTT variable
- \Rightarrow dimensionnement dynamique du temporisateur

Exercice

- Compléter le schéma suivant :



Exercice

- Donner les numéros de séquence des deux premiers segments, du 5ème segment et du dernier segment pour :
 - ➔ Un fichier de 500 000 octets émis par A
 - ➔ MSS = 1000 octets
 - ➔ La numérotation des octets commence à 0