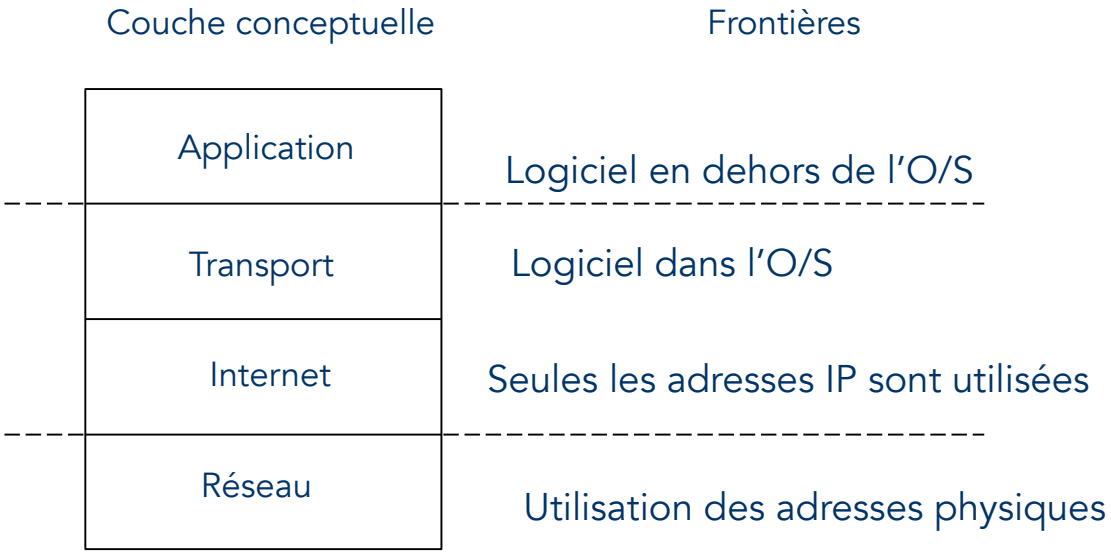
Couche Transport

- Les protocoles de transport
- Numéros de port
- UDP
- TCP

Les protocoles de transport

Fontières cruciales dans le modèle TCP/IP



Adressage pour les applications

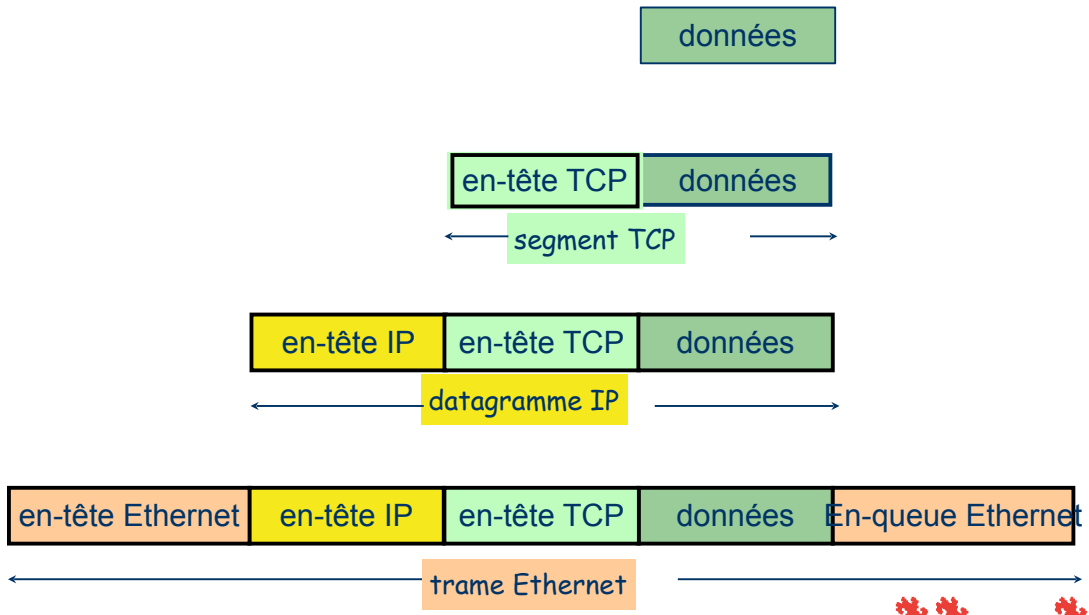
Sur un même hôte on trouve plusieurs applications

- il faut donc pouvoir les identifier de façon non ambigüe
- introduction d'une nouvelle adresse
  - @MAC identifie l'émetteur et le récepteur de la trame
  - @IP identifie la source et destination du datagramme IP
  - vers quelle application ??

identifiée directement par son *pid* (*process identifier*) ??

Encapsulations successives

- cas application / TCP / IP / Ethernet



Couche Transport

- Les protocoles de transport
- Numéros de port
- UDP
- TCP

La couche transport

But : fournir un transport de données fiable, et performant (coût) de la source vers la destination indépendamment du support physique ou du réseau.

Fonctions de la couche transport

- **Multiplexage et démultiplexage**
- [**Détection d'erreurs**] : puisque IP n'est pas fiable
- [**Établissement/fermeture d'une connexion**]
- [**Segmentation**] : découper un message en paquets
- [**Contrôle de flux**] : pour éviter les débordements
- [**Contrôle de congestion**]

Les protocoles de transport

L'Internet a 2 principaux protocoles de transport :

UDP

(User Datagram Protocol)

Service **sans connexion**

TCP

(Transmission Control Protocol)

- **Service fiable de bout en bout** sur un réseau non fiable.
- Service **orienté connexion**
- **Adaptation** dynamique aux caractéristiques de trafic du réseau et robustesse aux pannes.

Adressage pour les applications

on peut identifier indirectement un processus par une référence abstraite (*abstract locater*) appelée **port**

- un processus source envoie un message sur son port
- 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

Adressage pour les applications

champs *port* codés sur 16 bits

- 65 535 valeurs différentes ➔ 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) : un processus est identifié par son port sur une machine donnée
  - ↳ clé de démultiplexage de UDP = (port, hôte)

Adressage pour les applications

- Comment un processus connaît-il le port de celui à qui il souhaite envoyer un message ?

Modèle de communication client/serveur :
(client) requête -> (serveur) réponse
Le serveur connaît le port du client !!

- Comment le client connaît-il le port du serveur?

Il existe :
- des services communs (connus de tous)
- des services spécifiques connus de certains clients

Services connus
pompiers : 18
SAMU : 15

Services connus
DNS : 53
http : 80

Numéros de port

- 3 catégories
  - ports well-known : de 0 à 1023
    - alloués par l'IANA
    - 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
  - ports dynamic/private : de 49 152 à 65 535
    - alloués dynamiquement

Les ports well-known

No port	Mot-clé	Description
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

Couche Transport

- Les protocoles de transport
- Numéros de port
- UDP
- TCP

Le protocole UDP

User Datagram Protocol

- étend le service de remise d'hôte à hôte à un service de remise de processus à processus
- Protocole **orienté messages**
- fournit un service **non fiable** sans connexion - les messages peuvent être perdus, dupliqués ou arriver dans le désordre.
- permet de distinguer différentes destination pour un hôte donné à l'aide des **n° de port** destination et source

Le protocole UDP

Files d'attente UDP

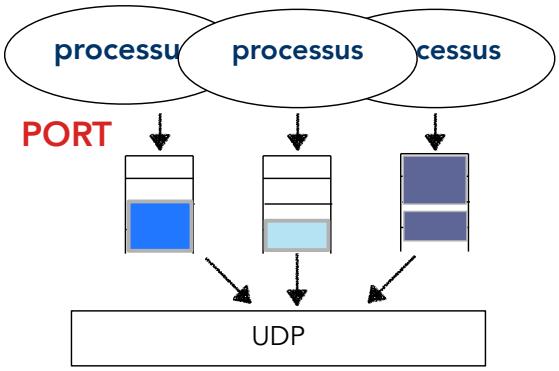
- L'O/S crée une file d'attente interne par PORT
- La taille de la file peut être spécifiée ou changée par l'application
- Quand UDP reçoit un datagramme, il contrôle le n° de port de la destination avec la liste des ports actifs en cours d'utilisation
- Sinon, il envoie un message ICMP "port unreachable error"

Implémentation des ports

PORT = référence ***abstraite***

- l'implémentation diffère d'un OS à l'autre
- en général, un port = une file de messages

L'O/S crée une file d'attente interne par PORT



envoi indépendant de messages

Taille max du message UDP : 65534 octets

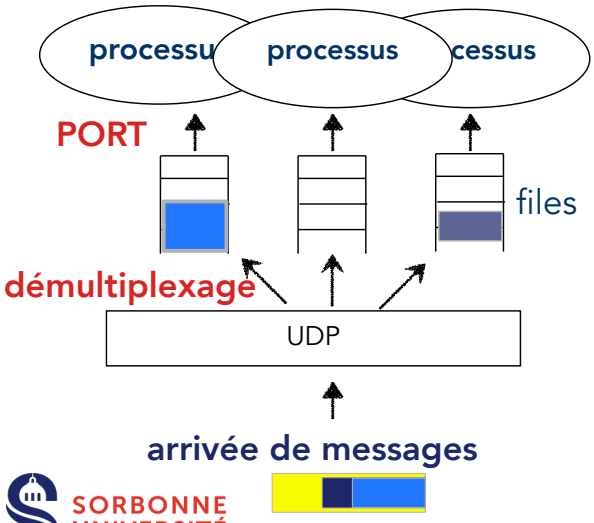
Entête UDP : 8 octets
Entête IP : 20 octets (min)
Si message UDP > 1472 octets
—> **Fragmentation** IP

Taille max. du champ « données » de la trame Ethernet : 1500 octets

Implémentation des ports

PORT = référence ***abstraite***

- l'implémentation diffère d'un OS à l'autre
- en général, un port = une file de messages



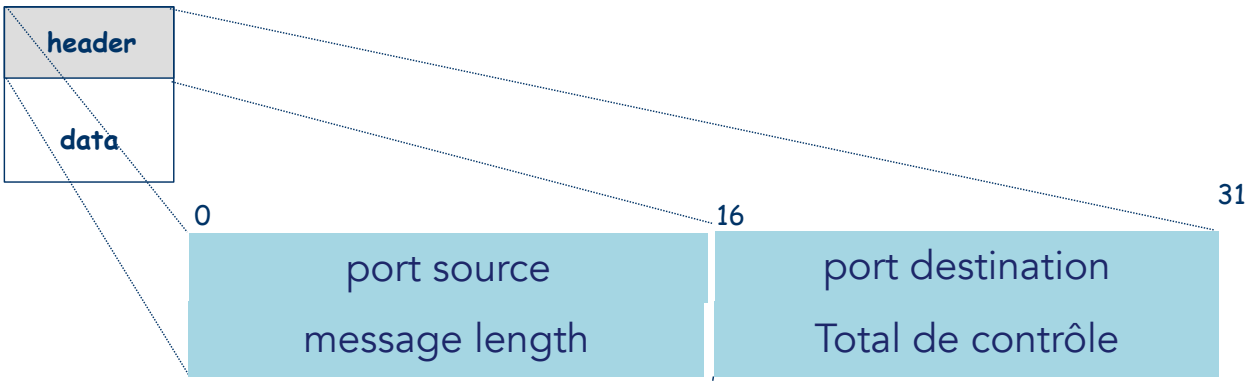
Le processus reçoit un msg

- 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

UDP reçoit un msg

- il l'insère en fin de file
- si la file est pleine, le msg est rejeté
- si le n° port n'est pas trouvé
  —> ICMP « port unreachable »

Le datagramme UDP



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

Le checksum UDP

- Calcul optionnel avec IPv4, obligatoire avec IPv6
- portée :
  - en-tête UDP
  - champ de données UDP
  - *pseudo-header (une partie de l'entête IP)*
    - champ Protocole IP (8 bits cadrés à droite sur 16 bits)
    - champ IP ***@source*** (32 bits)
    - champ IP ***@dest***. (32 bits)
    - champ UDP ***length*** (16 bits)

UDP est indissociable de IP !

Le checksum UDP

- algorithme de calcul

Principe

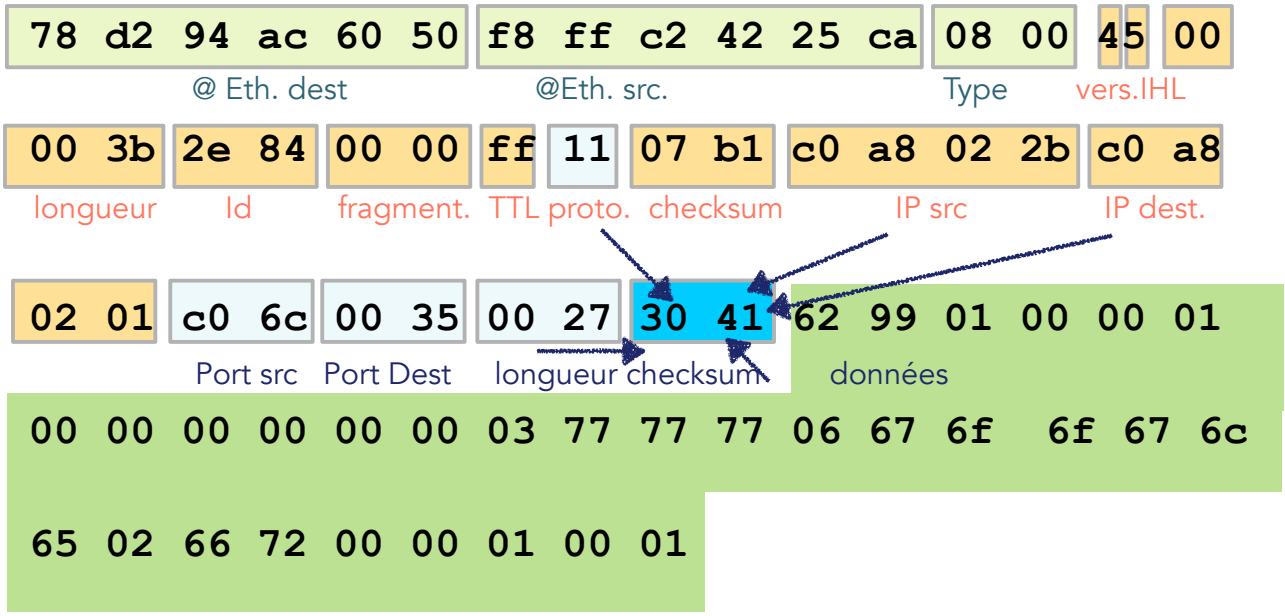
- le champ ***checksum*** est initialement mis à 0
- la suite à protéger est considérée comme une suite de mots de 16 bits
- les mots de 16 bits sont additionnés un à un, modulo 65 535
- le checksum est le complément à 1 (inverse bit à bit) de la somme trouvée

↳ le récepteur fait la somme modulo 65 535 de tous les mots concernés et vérifie qu'il obtient FF FF ou 00 00

☺ implémentation logicielle simple

☹ moins puissant qu'un CRC

Datagramme UDP



UDP - résumé

- UDP
  - entête 8 octets (20 octets pour TCP entête)
  - pas de connexion à établir et maintenir
  - primitive détection d'erreur
  - pas de séquencement
  - pas de contrôle de congestion

Couche Transport

- Les protocoles de transport
- Numéros de port
- UDP
- TCP

Le protocole TCP

Transmission Control Protocol

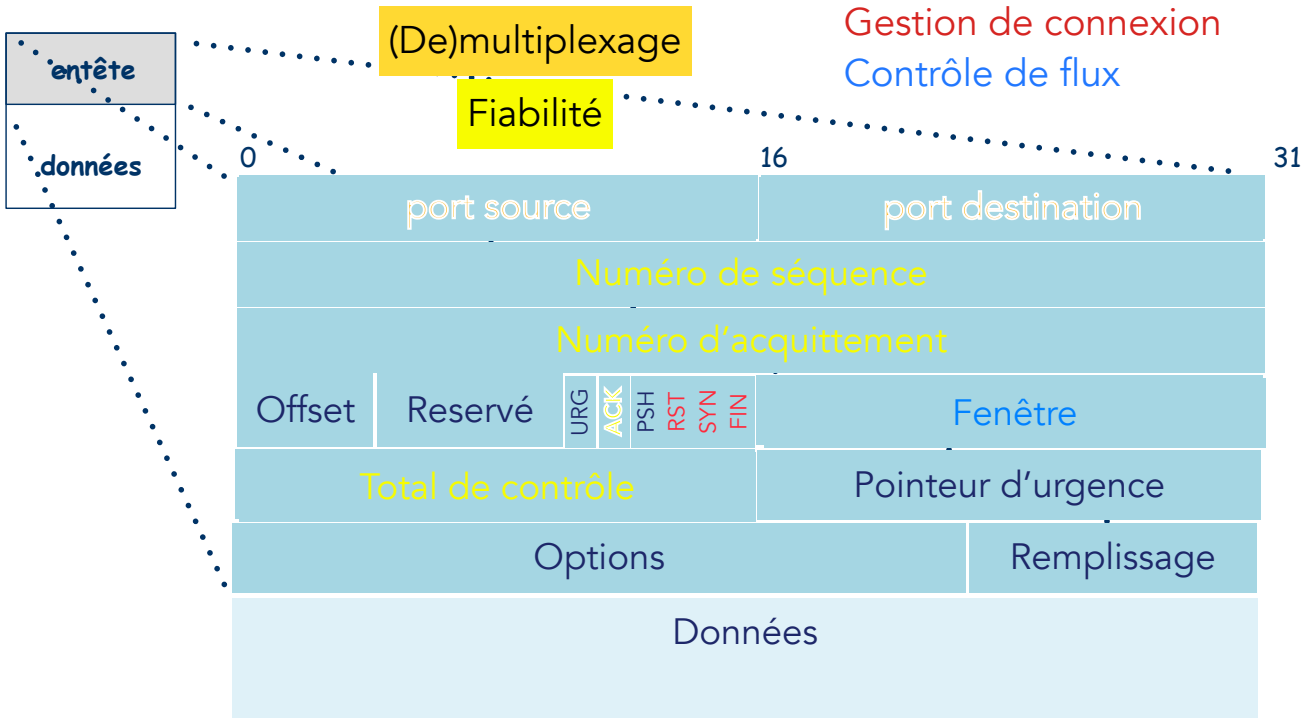
offre un service de remise
- en mode connecté
- fiable
- full-duplex
- de flux d'octets

met en œuvre des mécanismes de
- (dé)multiplexage
- gestion de connexions
- segmentation
- fiabilité
- contrôle de flux
- contrôle de congestion

Les ports

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
80	HTTP	WWW
110	POP3	Post Office Protocol - Version 3

Le segment TCP



Le segment 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 (codé sur 4 bits)
- reserved : 6 bits à 0
- Flags :
  - URG : mis à 1 pour signaler la présence de données urgentes
  - ACK : mis à 1 pour indiquer que le champ acknowledment 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
- fenêtre : # d'octets de données que le destinataire du segment pourra émettre
- Total de contrôle : obligatoire, calculé sur la totalité du segment et sur le pseudo en-tête
- pointeur urgent : pointe sur la fin (comprise) des données urgentes
- options : MSS, …
- remplissage : alignement de l'en-tête sur 32 bits

Le segment TCP

Les options TCP sont de la forme : `TYPE-[LONGUEUR-DONNEES]`

- **EOOL** : T=0, L=1 - End of Option List
- **NOP** : T=1, L=1 - les options utilisent un nombre quelconque d'octets mais doivent être alignés sur un multiple de 4 octets. « No Operation »
- **MSS** (Maximum Segment Size) : T=2, L=4, D=taille max du segment que l'émetteur peut recevoir.
- **Timestamp** : T=8, L=10, D=Deux horodatages de 4 octets chacun pour synchroniser les horloges entre hôtes.
- **wscale** : T=3, L=3, D=facteur d'échelle (« shift ») pour augmenter la taille de la fenêtre au-delà des 16 bits du champ WINDOW (> 65535). (Ex. si « shift » vaut 0, la taille de la fenêtre reste 65535, s'il vaut 1, la taille de la fenêtre est de 131072 octets).
- **SACK-Permitted** : T=4,L=2 - Autorise l'utilisation des « selective Acknowledgment » pour indiquer les paquets reçus qui ne sont pas en séquence.

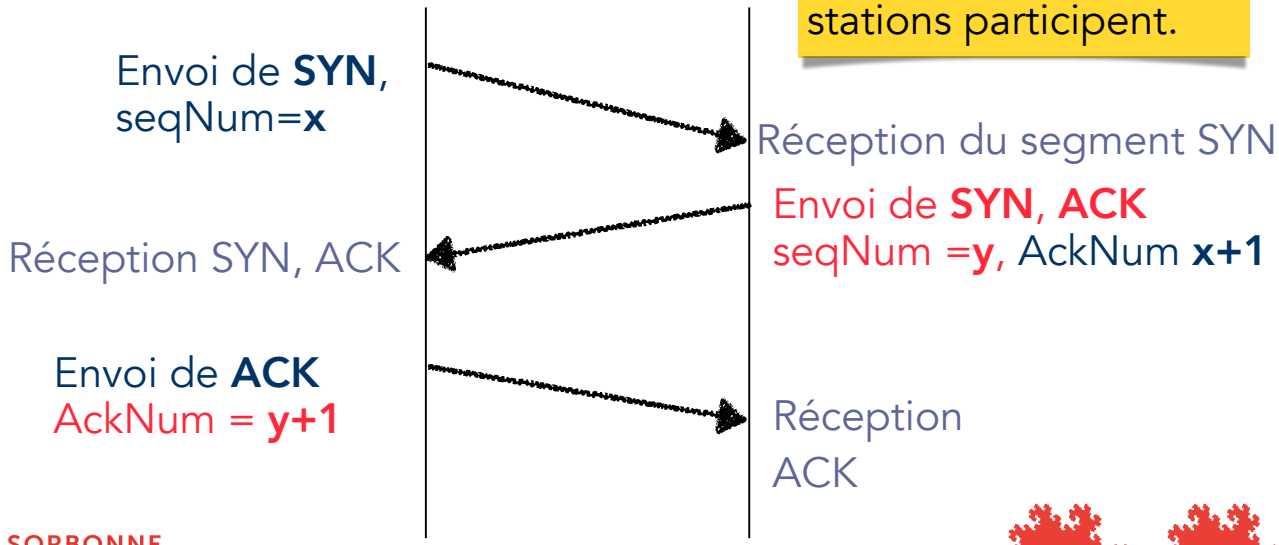
Le segment TCP

```
78 d2 94 ac 60 50  f8 ff c2 42 25 ca  08 00  45 00
   Eth dest           Eth src          Type  v4IHL

00 40  00 00  40 00  40  06  a4 ef  c0 a8 02 2b  68 12
Lentgh  Id no  Frag.  TTL Prot.chcksm  IPsrc      IPdest

6a e3  dd 5f  01 bb  ad a1 21 23  00 00 00 00  b0 02
       Portsrc Portdest   seq. N°     ACK n°   offset Flags

ff ff  07 6f  00 00  02 04 05 b4  01  03 03 06  01 01
window chcksm Urg ptr MSS L taille  Nop Win.scale Nop Nop

08 0a 29 75 c2 6f  00 00 00 00  04 02  00  00
opt L. Timestamp(S)  (R)        sack   EOL Padding
```


Le protocole TCP

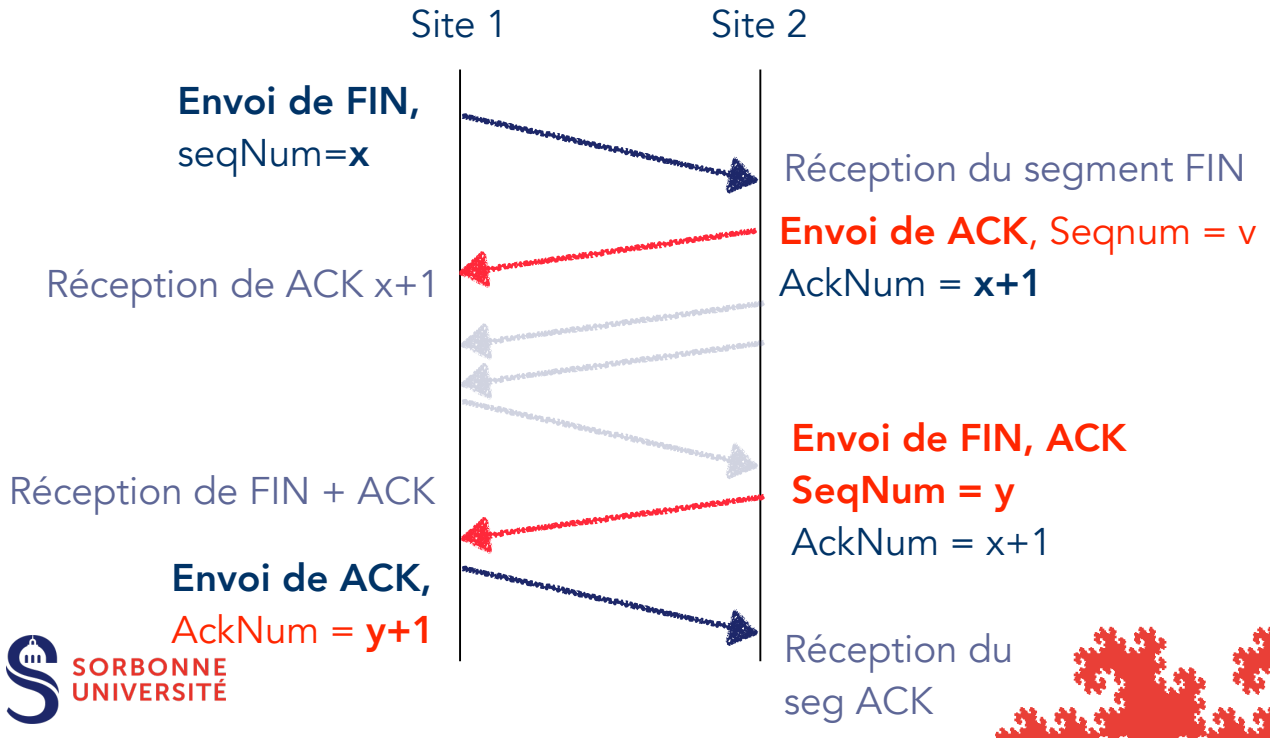
Ouverture de connexion

TCP est un protocole **orienté connexion** qui requiert que les deux stations participent.



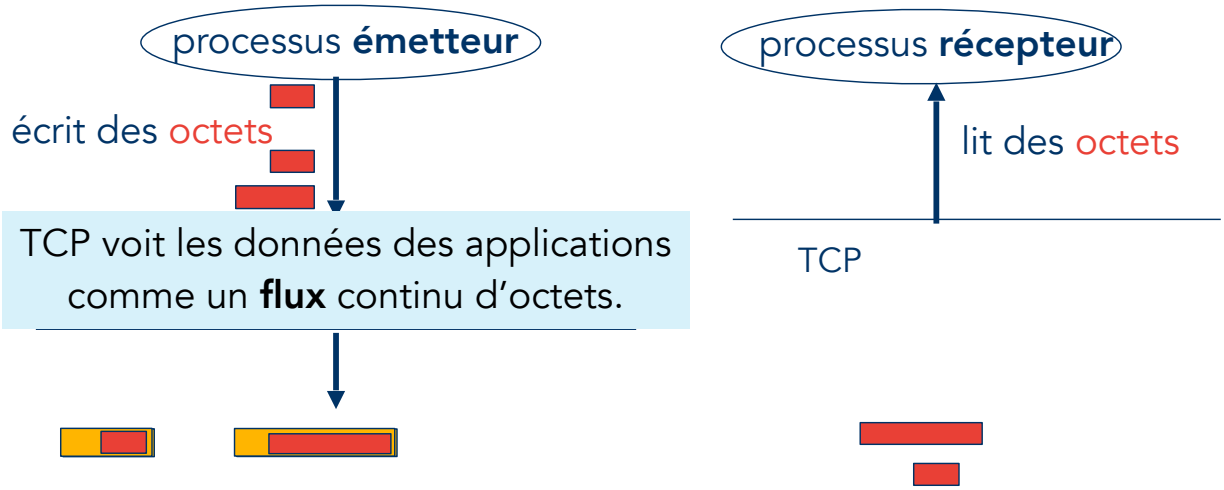
Le protocole TCP

Fermeture d'une connexion



Le protocole TCP

Protocole orienté flux d'octets



TCP divise le flux de données en segments TCP
1 segment TCP -> 1 datagramme IP.

Il faut **segmenter** les données

Le protocole TCP

Segmentation

- Pas d'obligation d'envoyer les données dès qu'elles arrivent et tous les segments de la connexion n'ont pas nécessairement la même taille.
- Plusieurs choix possibles pour déterminer quand envoyer les données :
    1. Attendre un peu (500ms) pour essayer de regrouper les données (et ACK)
    2. Attendre qu'un segment ait atteint la MSS (algorithme de Nagle).
    3. Quand il s'agit de données Urgentes

Le protocole TCP

Taille du segment

- Il est préférable que la taille maximum du segment (**MSS**) soit la plus grand possible mais il faut éviter les fragmentations dans les couches inférieures.
- La valeur du MSS est négociée à l'ouverture de la connexion.

Le protocole TCP

Numérotation des segments

Emetteur

- Les segments envoyés sont de longueur variable
- Les segments *retransmis* n'ont pas nécessairement la même taille que le segment original donc les ACKs ne peuvent pas faire référence aux n° de segments.

récepteur :

- récolte un flux d'octets de données
- reconstruit une copie exacte du flux envoyé

La numérotation fait référence à une position dans un flux utilisant des numéros d'octets.

Le protocole TCP

Gestion des buffers

Emission

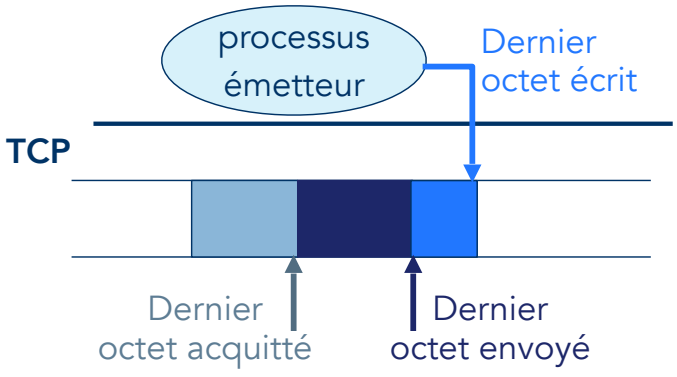
Le buffer stocke

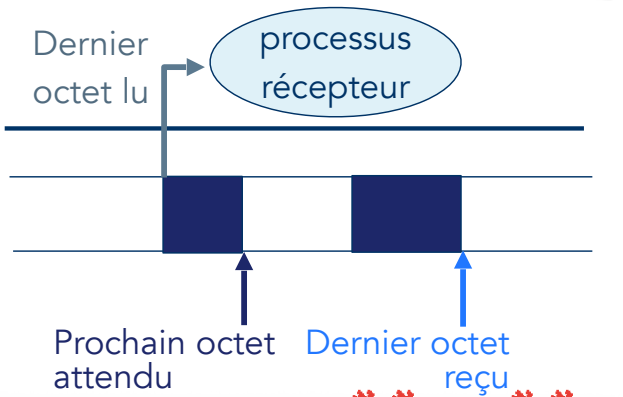
- les données envoyées en attente d'ACK
- les données non encore émises

Réception

le buffer stocke

- les données dans l'ordre non encore lues par le processus récepteur
- les données déséquencées





Dernier_octet_Lu < Prochain_octet_attendu ≤ dernier_octet_reçu +1

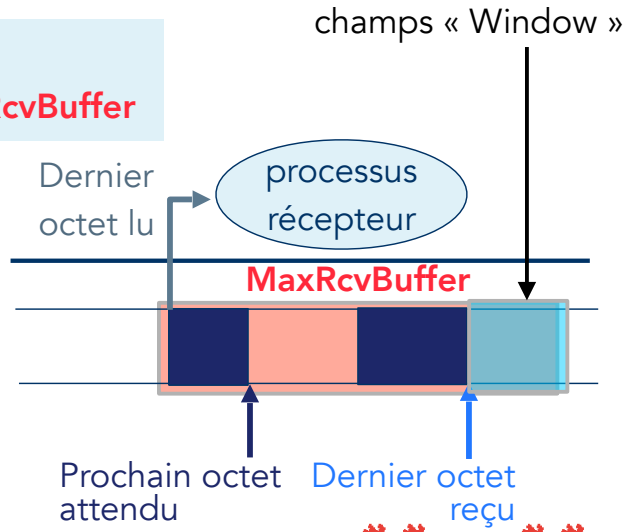
Le protocole TCP

TCP informe de la taille de buffer disponible

Window < MaxRecvBuffer – (Dernier octet reçu – Dernier octet lu)

A tout moment

Dernier octet reçu – dernier octet Lu ≤ **MaxRcvBuffer**



Le protocole TCP

Fiabilité

Comment fournir un transfert fiable si le système de communication sous-jacent ne l'est pas !

Mécanismes pour fiabiliser les transferts

- Numéros de séquence
- Checksum
- ACK positif avec retransmission
- ACK négatif avec retransmission
- Utilisation de temporisateurs de retransmission

Le protocole TCP

Fiabilité : Récepteur -> gestion des ACK

Evénement	Action de TCP
Arrivée d'un segment en séquence. Toutes les données précédentes reçues en séquence ont été acquittés	ACK retardé (500ms) pour permettre si possible les ACK groupés.
Arrivée d'un segment en séquence. Toutes les données précédentes reçues en séquence pas encore acquittées	Envoi immédiat d'un ACK groupé.
Arrivée de données « autorisées » mais pas en séquence (détection d'un manque)	Envoi d'un ACK dupliqué (n° du prochain octet en séquence attendu)
Arrivée d'un segment qui comble (partiellement ou complètement) des données manquantes en séquence.	Envoi immédiat d'un ACK indiquant le prochain octet en séquence attendu (ou n° des octets ACK si SACK)

Le protocole TCP

Fiabilité : l'émetteur utilise

- des **acquittements positifs**
- des **acquittent négatifs :** 3 ACK avec le même n° reçus consécutivement sont considérés comme des ACK négatifs
- un **temporisateur de retransmission** déclenché à l'envoi de segment: s'il expire -> retransmission du segment.

Quelle valeur choisir pour le temporisateur de retransmission ?

RTT = α * Old_RTT + (1 - α) * RTT_mesuré

RTO = β * RTT

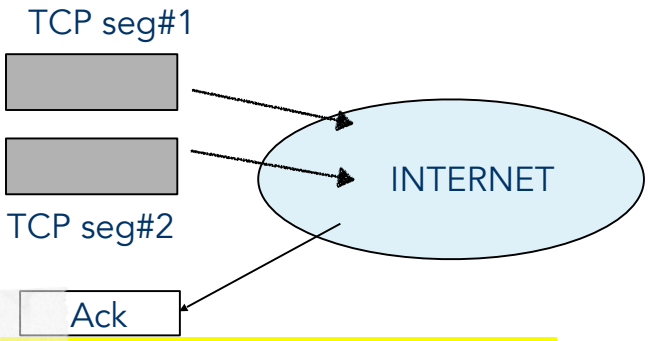
RTT : temps d'aller et retour — α: coefficient de lissage=0,9

RTO : Round Trip Timeout — $\beta > 1$ (β=2)

Le protocole TCP

Ambiguités de TCP

Estimation du RTT =
instant ACK reçu - Instant envoyé



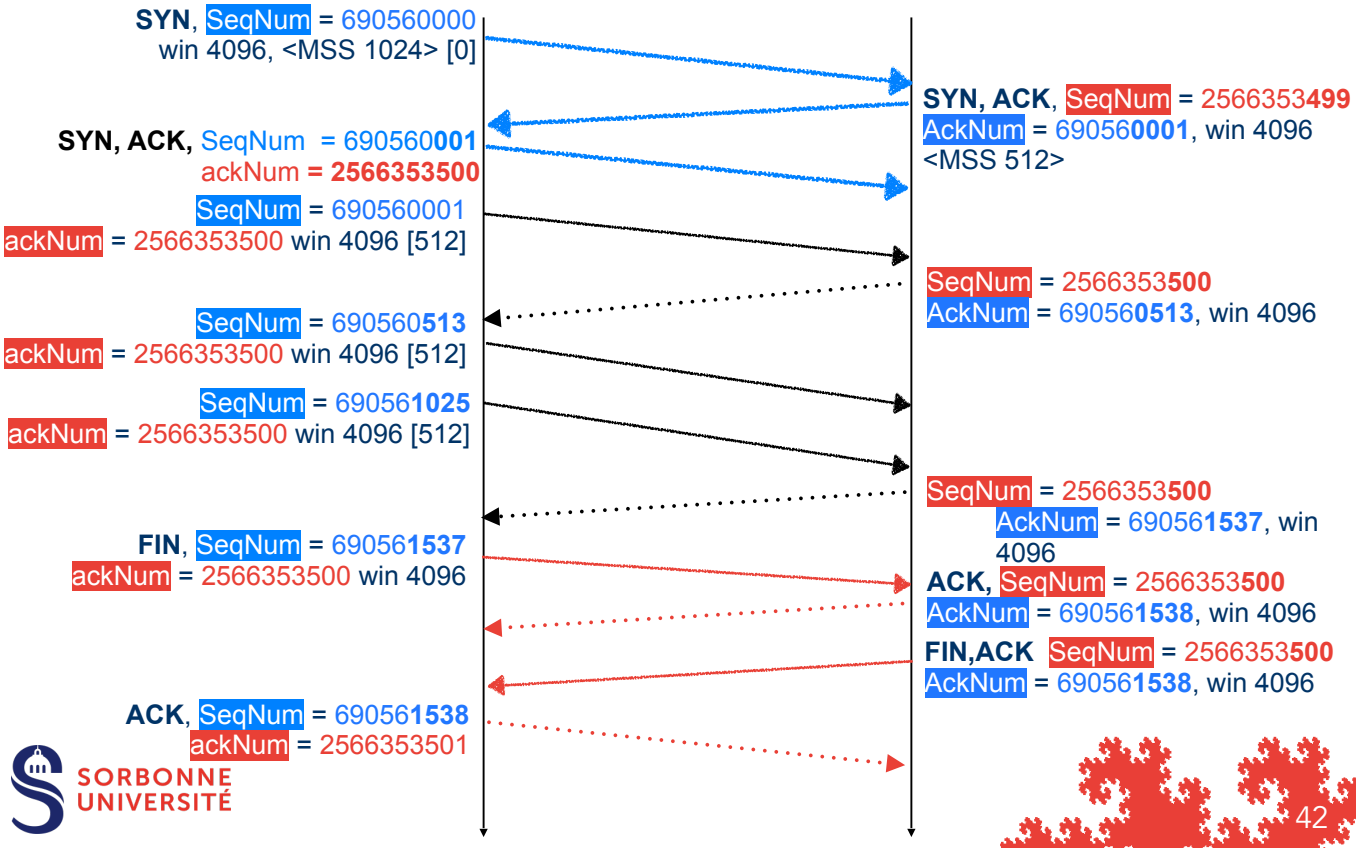
Solution de Karn : pas de mise à jour du RTT en cas de retransmission.

- RTT initial estimé (1s).
- si le tempo expire : RTO = γ RTO (Stratégie du « timer backoff »)

Utilisation d'un horodatage

- Options TCP : définie lors de l'établissement de la connexion
- TSval : horodatage pour les trames émises
- TSecr : réécriture du TSval dans les trames d'ACK -> l'émetteur peut facilement mettre à jour son RTT

Le protocole TCP



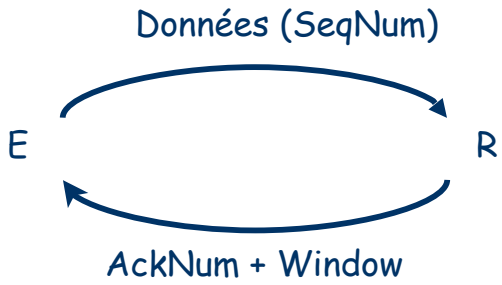
SORBONNE UNIVERSITÉ

Le protocole TCP

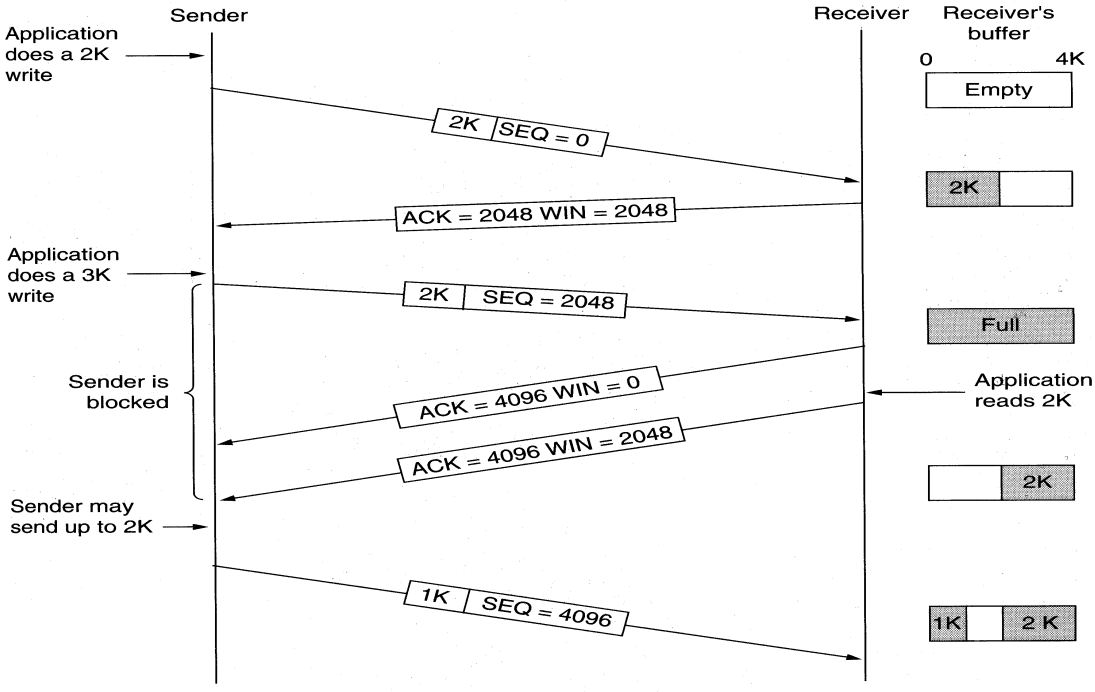
Le contrôle de flux

- une entité réceptrice TCP dispose de ressources (buffers) en nombre limité
- la taille de la fenêtre (champ Window) reflète la disponibilité des buffers de réception
- une entité TCP gère un nombre de connexions variable

Le champ ***Window*** indique le **# max d'octets** de données que R est prêt à recevoir



Le protocole TCP



Le protocole TCP

- Les temporisateurs persistants de TCP
  - Que se passe t-il quand la fenêtre de réception tend vers 0 ?
  - Que se passe t-il si le deuxième ack indiquant que la fenêtre de réception a augmenté est perdue ?
  - L'émetteur utilise un temporisateur persistant et envoie des segments (sondes de fenêtre) pour déterminer si la fenêtre a été agrandie.
  - Les sondes sont déclenchées après 500ms, puis un algorithme de retrait exponentiel est utilisé pour recalculer ce temporisateur : 1,5sec; 1,5*2=3sec; 1,5*4= 6sec ; 1,5*8=12sec

- Une temporisation 'keepalive' permet de terminer une connexion

TCP - résumé

- Entête (20 octets minimum)
- Transfert fiable
- Les données ne sont pas envoyées immédiatement (formation des segments, contrôle de congestion)
- surcoût (retransmission, ACK, ouverture de connexion)
- transmission orientée flux, segmenté par TCP.