Couche Transport : TCP & UDP

Thierry Vaira

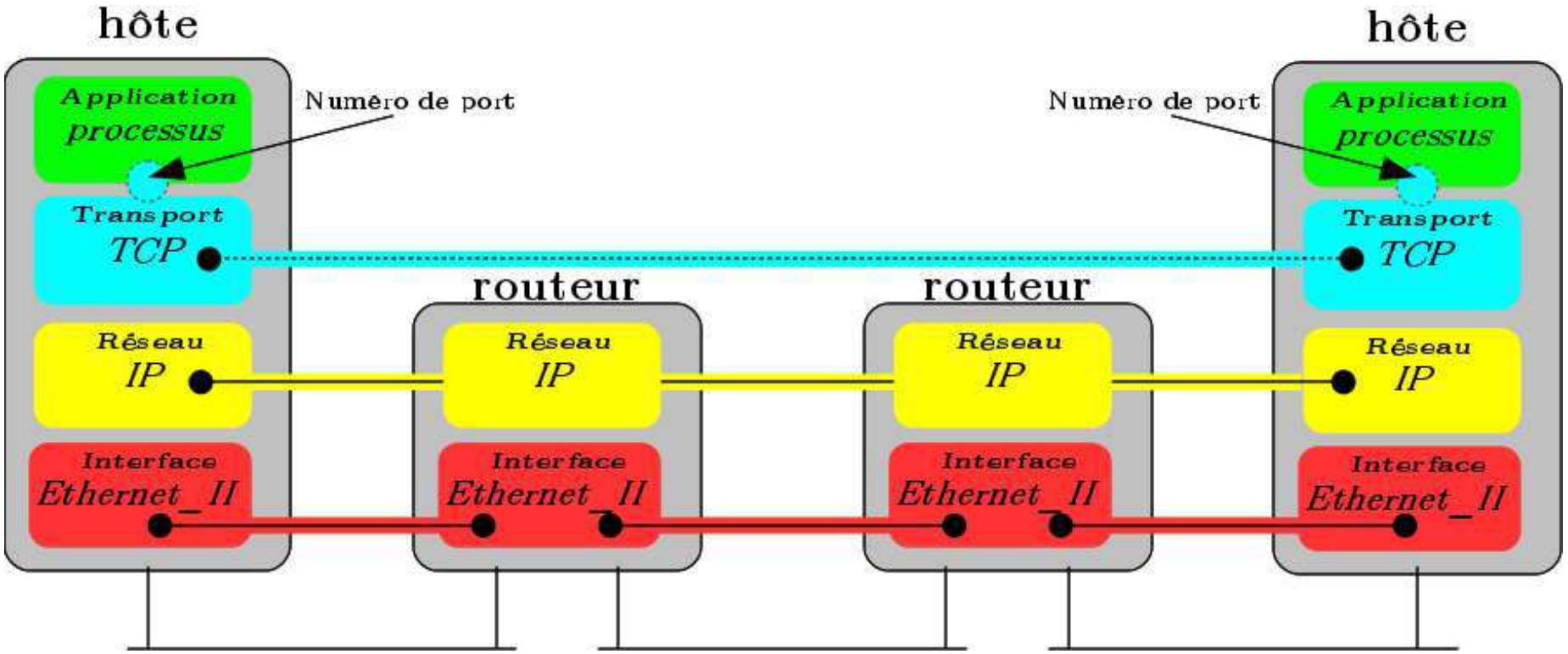
BTS IRIS Avignon

v0.1 13 novembre 2011

Sommaire

La couche Transport

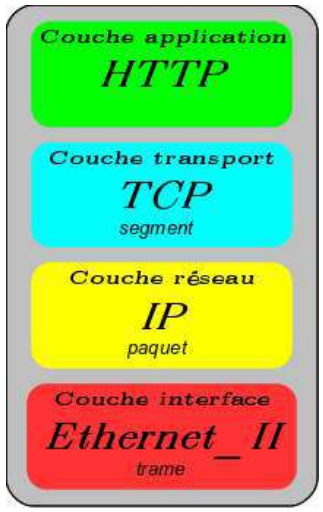
- Elle est responsable du transport des messages complets de bout en bout (soit de processus à processus) au travers du réseau.



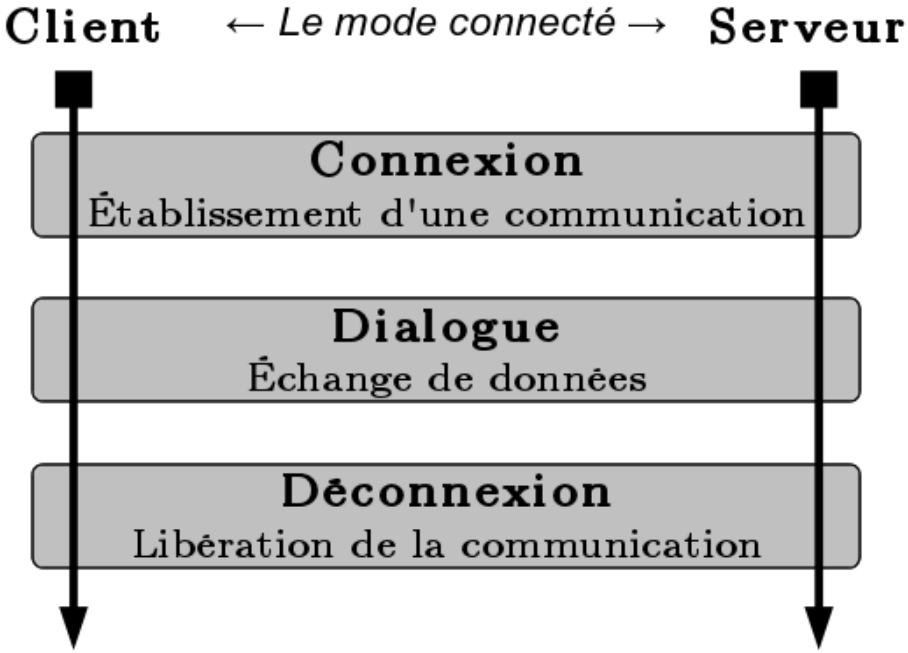
Exemple : le protocole de transport TCP

Le protocole TCP

- TCP (*Transmission Control Protocol*) est un protocole de transport **fiable**, en **mode connecté** (RFC 793) qui assure la **transport des données de la couche Application de bout en bout** (d'un processus à un autre processus).



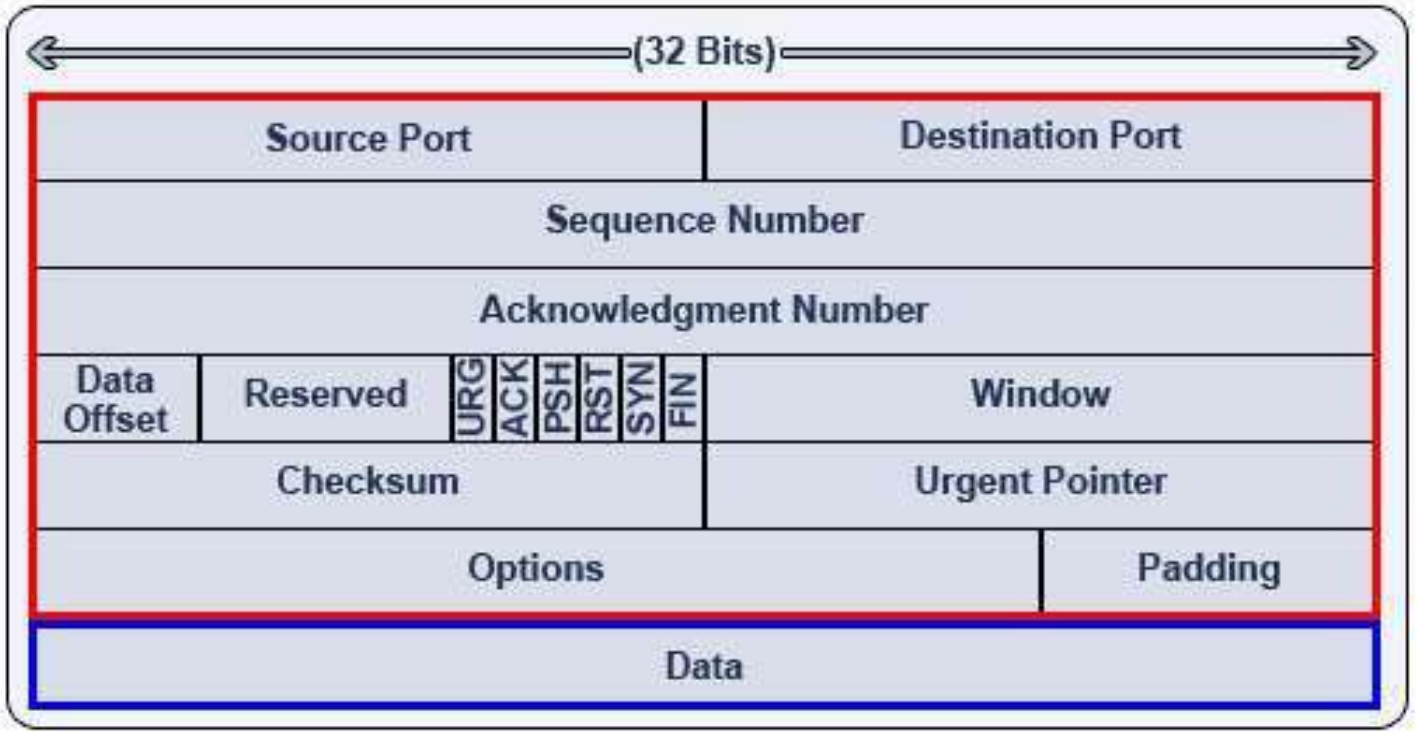
Modèle DoD



- La couche Transport (TCP et UDP) utilise les **numéros de port** comme technique d'adressage des bouts d'une communication.

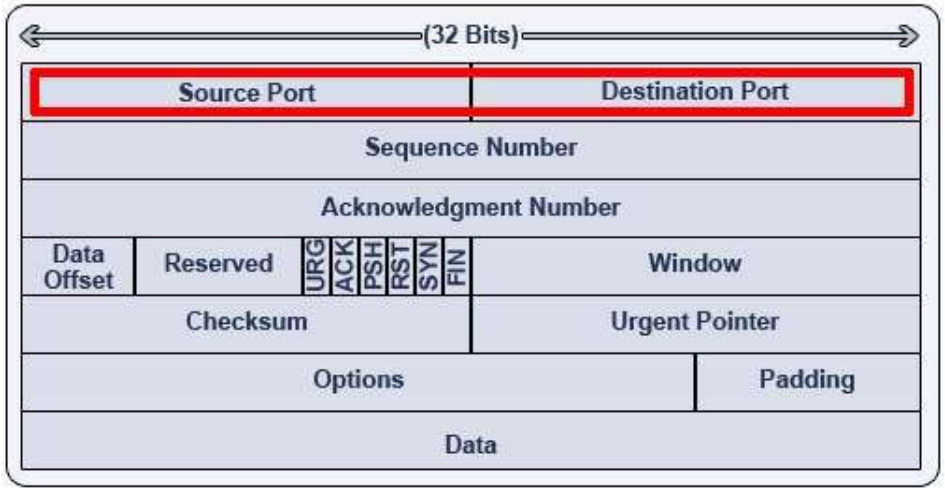
Segment TCP

Un segment TCP est constitué d'un **en-tête (*header*)** et des **données (*data*)** :



Numéro de port

- Un **numéro de port** sert à identifier l'application (un processus) en cours de communication par l'intermédiaire de son protocole de couche application (associé au service utilisé) :
  - Pour chaque port, un numéro lui est attribué codé sur **16 bits** (soit $2^{16} = 65536$ ports distincts)
  - L'attribution des ports est faite par le système d'exploitation, sur demande d'une application.



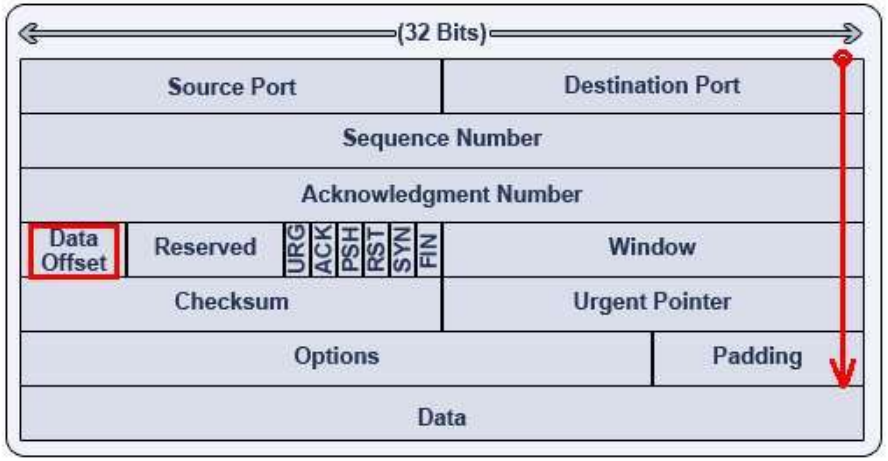
Attribution connue des numéros de ports : `# more /etc/services`

Extraire les données d'un segment

- Pour extraire les données reçues d'un segment, on utilisera le champ ***Data Offset*** qui contient la longueur (en mots de 4 octets) de l'en-tête TCP.

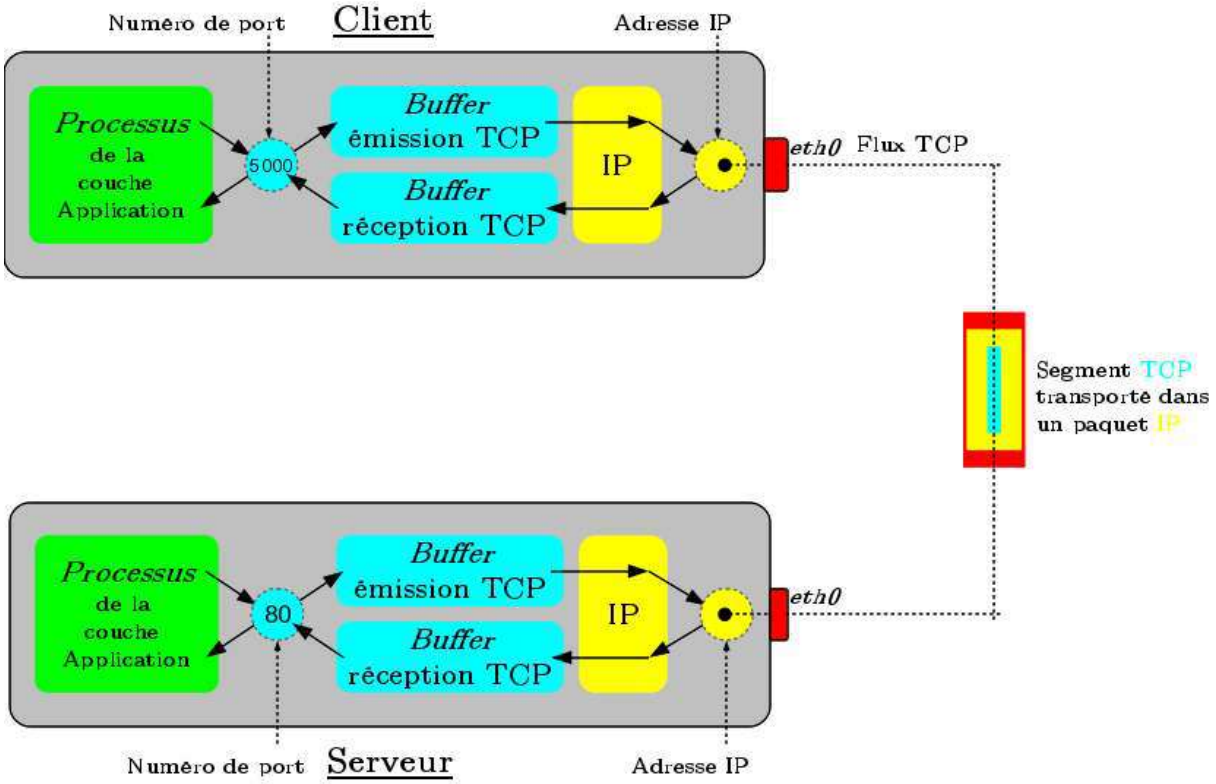
Exemple

Si le champ *Data Offset* contient la valeur **5** : l'en-tête TCP aura une longueur de $5x4 = 20$ octets. Cela correspond à un en-tête standard sans options. Le maximum sera de 60 octets (15 x 4 = 60).



Les données d'un segment

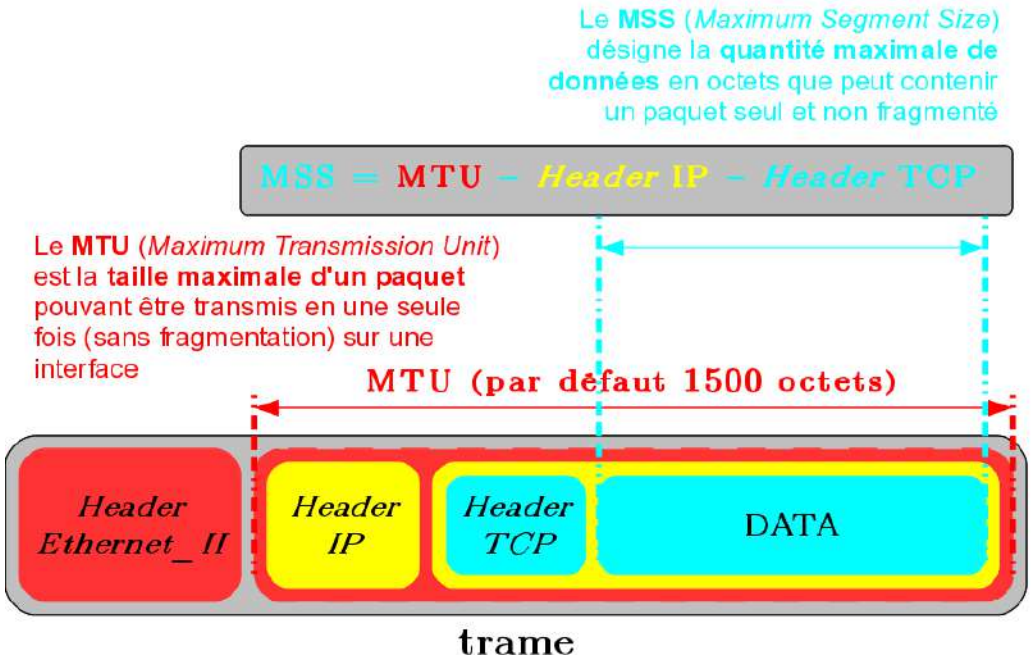
- Une communication TCP est **bidirectionnelle *full duplex*** et orientée **flux d'octets**.
- Les données en octets sont préalablement découpées en **segments**.



- Un segment TCP sera transporté au travers du réseau par un paquet IP, lui-même transmis par une trame sur le réseau local.

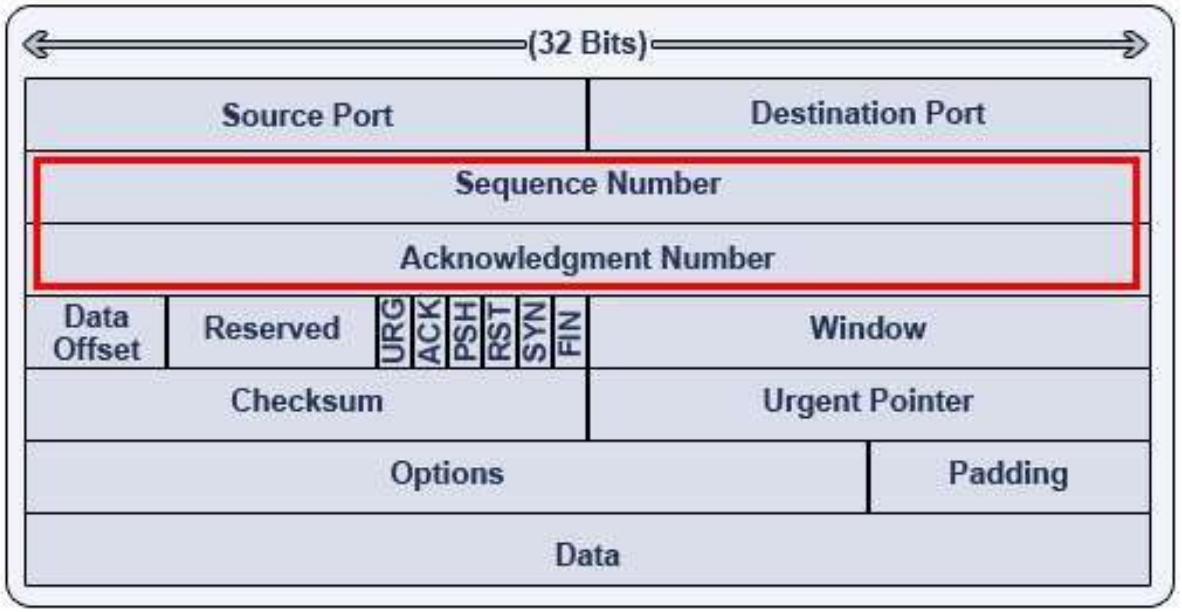
Taille des segments

- Le processus de couche application lit ou écrit dans des *buffers* (tampons).
- La couche TCP décide quand envoyer ou délivrer les données d'un segment (sauf cas particuliers : voir *flags* PSH et URG).
- En théorie, la taille d'un segment TCP est limité par la taille d'un paquet IP (64Ko). Mais TCP a intérêt à envoyer des segments de taille maximale en évitant toutefois la fragmentation IP qui est coûteuse.

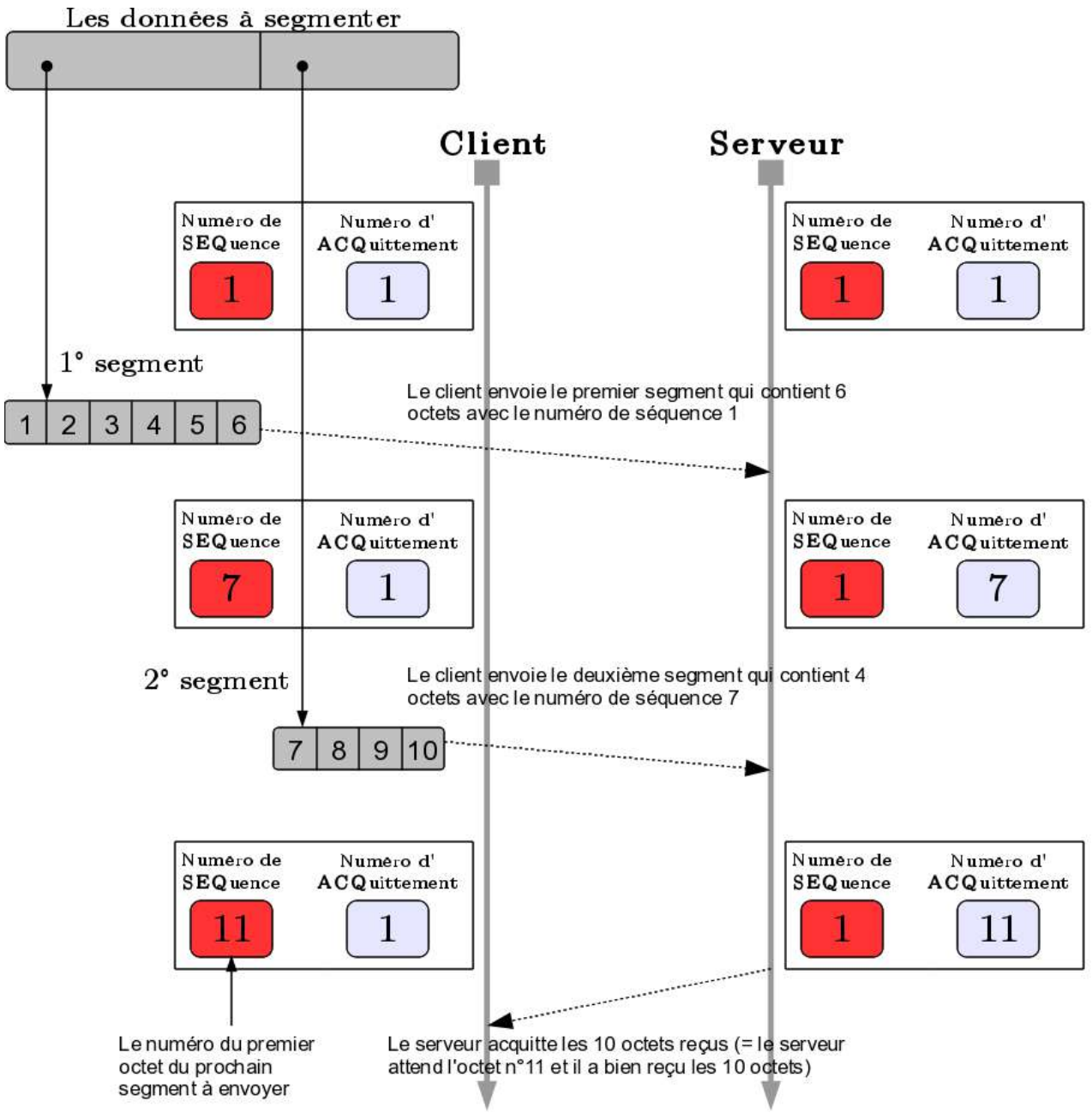


Séquencer les données à envoyer (1/2)

- Le protocole TCP comptabilise tous les octets transmis.
- Le champ ***Sequence Number*** indique la place du **premier octet de données du segment**.
- Le champ ***Acknowledgment Number*** indique le **prochain octet attendu** par l'émetteur du segment.
- Il y a une numérotation indépendante pour chaque sens de la connexion et elle ne commence pas forcément à 0.



Séquencer les données à envoyer (2/2)

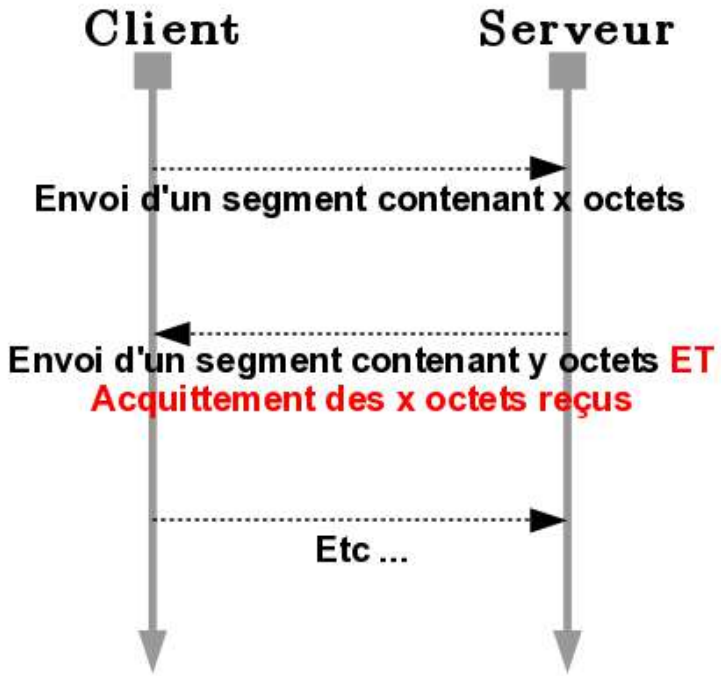


Fiabilité de la transmission

- La fiabilité de la transmission est assurée par un mécanisme baptisé *Positive Acknowledgement with Re-transmission* (**PAR**).
- L'émetteur démarre une alarme (*timeout*) à chaque envoi de segment :

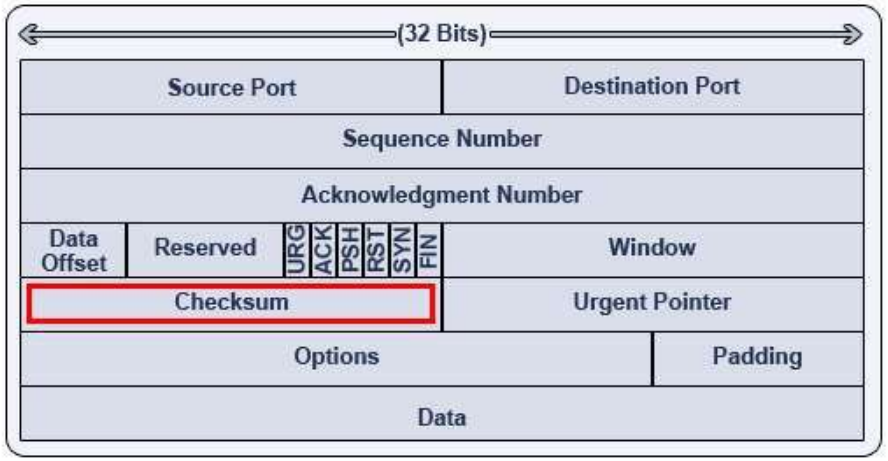
  `si alarme expire avant l'arrivée d'un acquittement`
  `alors retransmission des données du segment`

- Utilisation de la technique appelée *piggybacking* ("porter sur le dos") :



Checksum

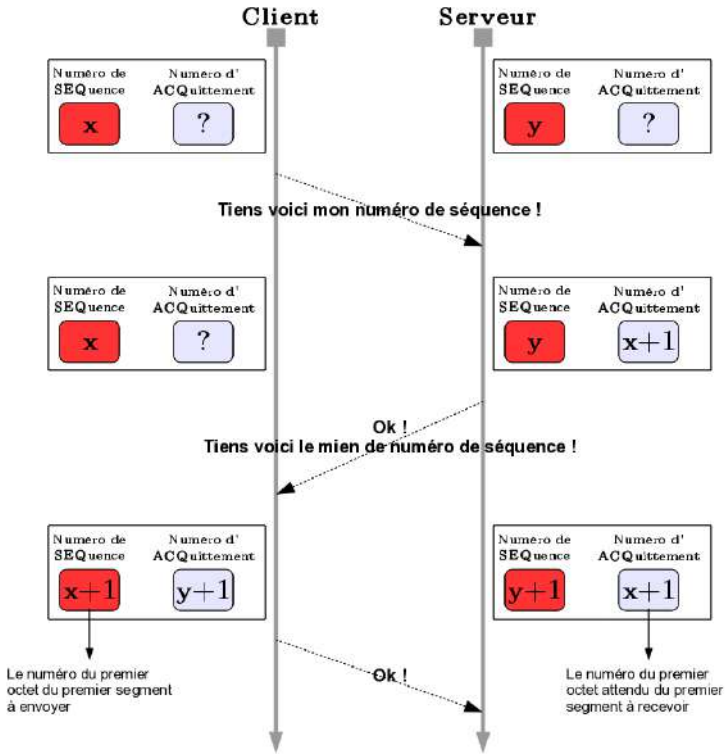
- Le champ ***Checksum*** est codé sur **16 bits** et permettra de valider le segment TCP reçu :
  - Le *Checksum* est constitué en calculant le complément à 1 sur 16 bits de la somme des compléments à 1 des octets du segment (en-tête + données) pris deux par deux (mots de 16 bits).
  - Un pseudo en-tête de 12 octets est constitué et pris en compte dans le calcul. Ce pseudo en-tête comporte les adresses IP sources et destinataires, le type de protocole et la longueur du message TCP (en-tête + données).



- Un exemple de fonction calculant le checksum TCP est fourni sur le site : www.frameip.com

Synchronisation des numéros de séquence

- Pour assurer le bon fonctionnement du séquencement des données, le client et le serveur doivent d'abord **synchroniser** leurs numéros de séquence initiaux.
- Cette synchronisation est réalisée lors de l'établissement de la communication (la phase de **connexion**).

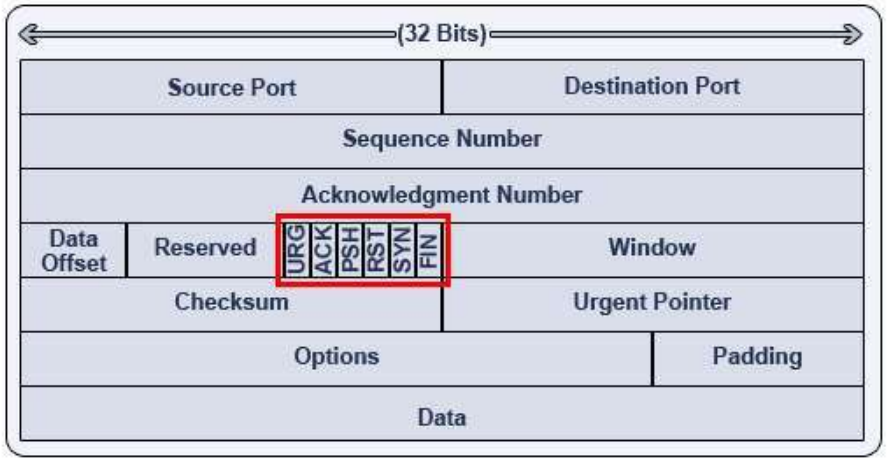


Frères des Ecoles Chrétiennes
Saint-Jean-Baptiste de la Salle - Avignon

Les drapeaux

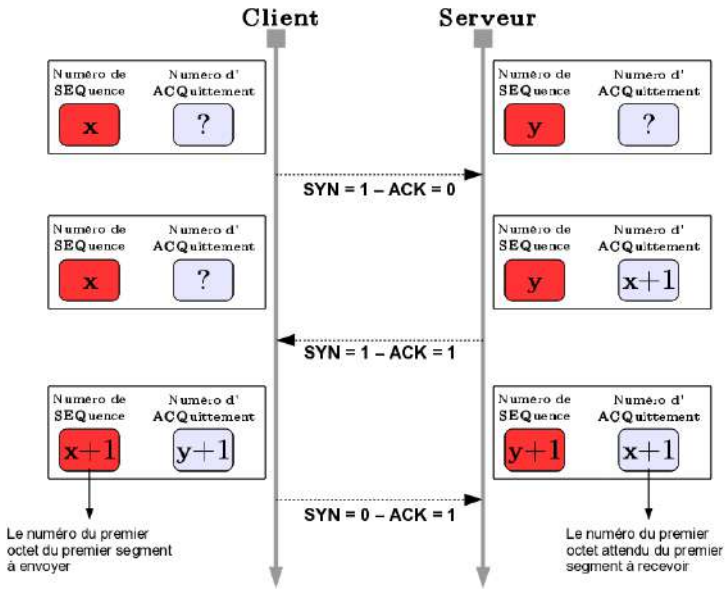
- Les **6 drapeaux** (*flags*) sont essentiels dans la gestion d'une communication TCP :

- 0x20 ← **URG** : valide le champ *Pointeur Urgent*
- 0x10 ← **ACK** : valide le champ *Acknowledgment Number*
- 0x08 ← **PSH** : indique au récepteur de délivrer immédiatement les données en attente
- 0x04 ← **RST** : demande au récepteur une réinitialisation de la connexion ou met fin à une demande
- 0x02 ← **SYN** : demande une synchronisation du *Sequence Number* (connexion)
- 0x01 ← **FIN** : l'émetteur demande une déconnexion



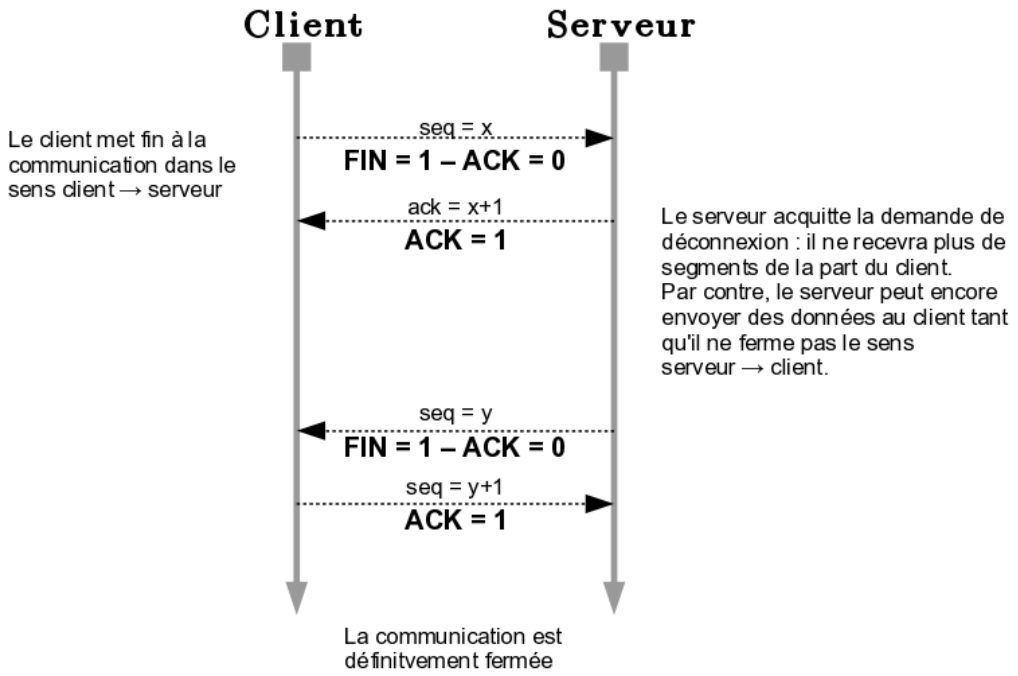
Connexion TCP

- Une ouverture active de connexion TCP est établie “en trois temps“ (*Three Way Handshake*).
  1) Le client TCP initialise la connexion en envoyant un segment incluant un SYN (*SYNchronize sequence numbers*) et un numéro de séquence x.
  2) Le serveur TCP lui répond par un segment avec les drapeaux SYN et ACK (*Acknowledgement*) avec un numéro d'acquittement x+1 et son numéro de séquence y.
  3) Le client TCP termine la connexion avec le *flag* ACK et le numéro d'acquittement y+1. Il peut déjà envoyer des données en même temps.



Déconnexion TCP

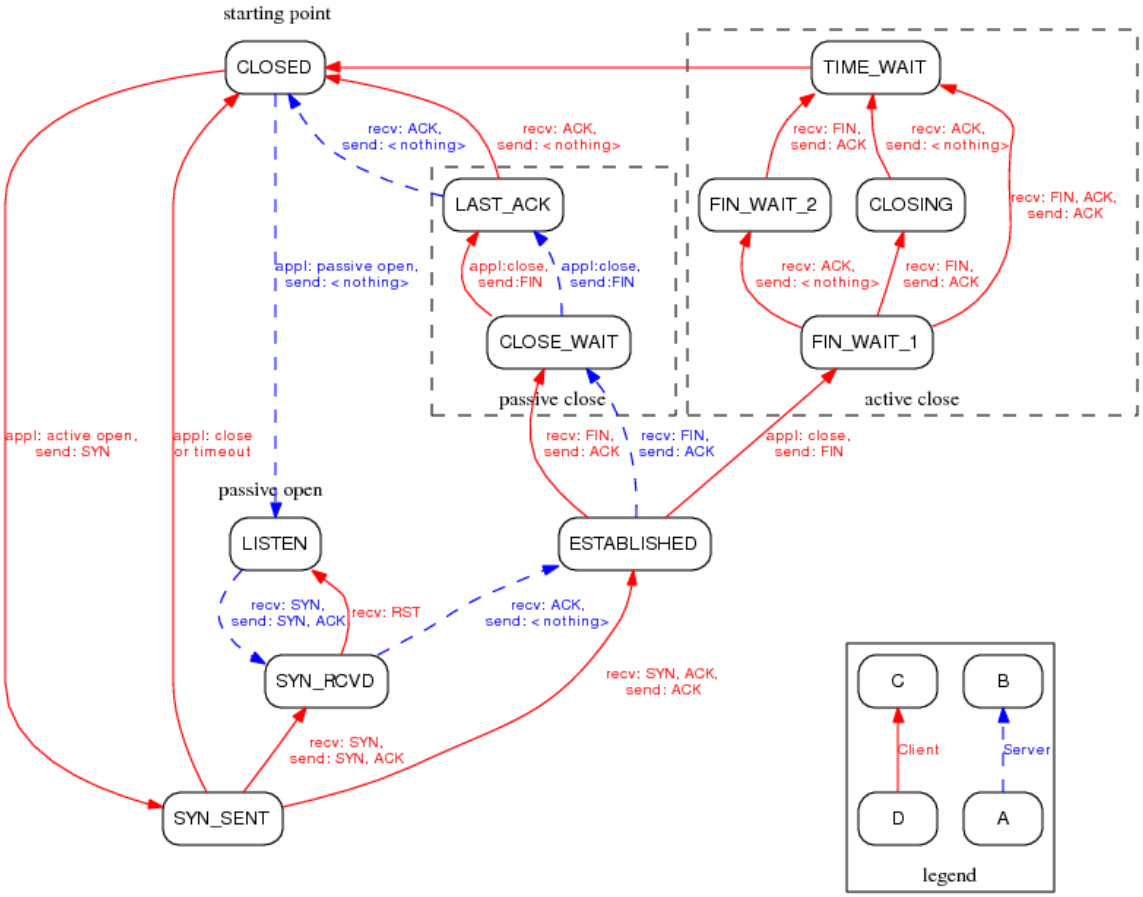
- Une déconnexion TCP se fera “en quatre temps“. La raison est qu’une connexion TCP est *full-duplex*, ce qui implique que les deux directions doivent pouvoir être fermées indépendamment l’une de l’autre.



- Un mécanisme de *reset* (*flag* **RST**) est prévu pour terminer une connexion au plus vite. Ce type d’arrêt est typiquement géré par la couche TCP elle-même quand l’application s’est brutalement interrompue.

États d'une communication TCP

- Une communication TCP comporte de nombreux **états** qui dépendent des *flags* :



Ces états sont observables sous Linux avec la commande : `# netstat -ta`

Contrôle de la communication

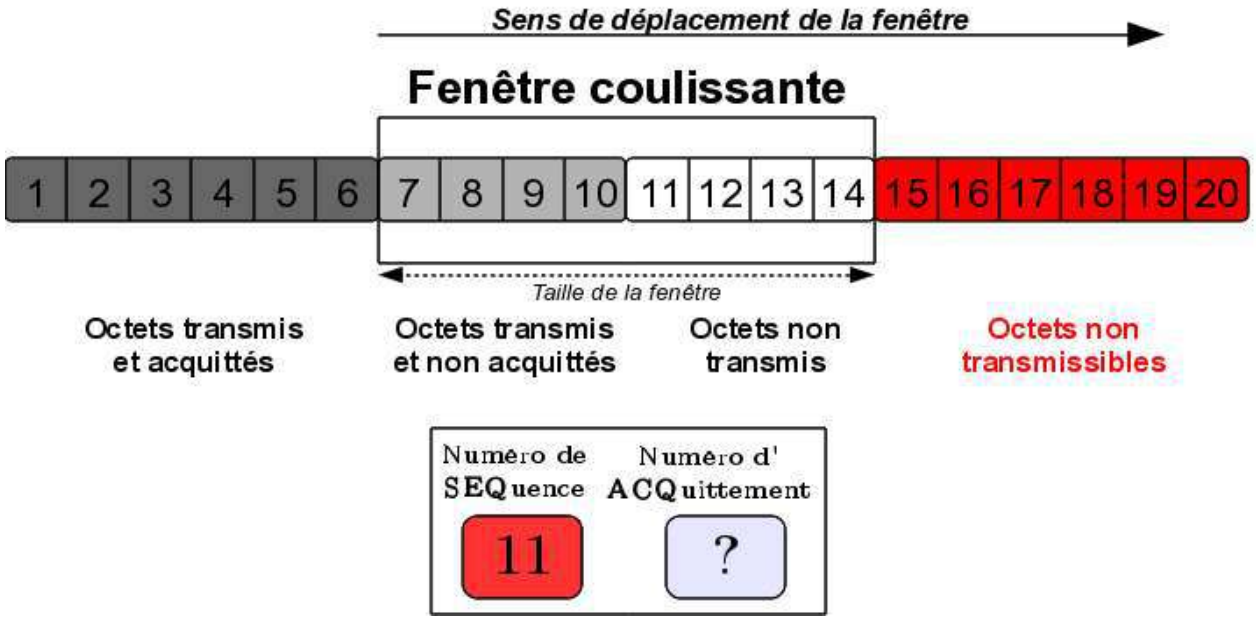
- Constat : l'émetteur ne doit pas saturer le tampon (*buffer*) de réception du récepteur.

- TCP s'appuie alors sur le mécanisme de la fenêtre (*Window*) pour réaliser :
  - Le contrôle de flux et de congestion
  - Le contrôle des erreurs, des pertes, duplication
  - L'optimisation de l'utilisation de la connexion

- Une communication TCP sera considérée comme **fiable** car elle est basée sur :
  - la numérotation des octets (*Sequence Number*)
  - la détection des erreurs (*checksum*)
  - la détection des pertes (*timeout* et acquittements successifs)
  - la récupération des pertes (par retransmission)

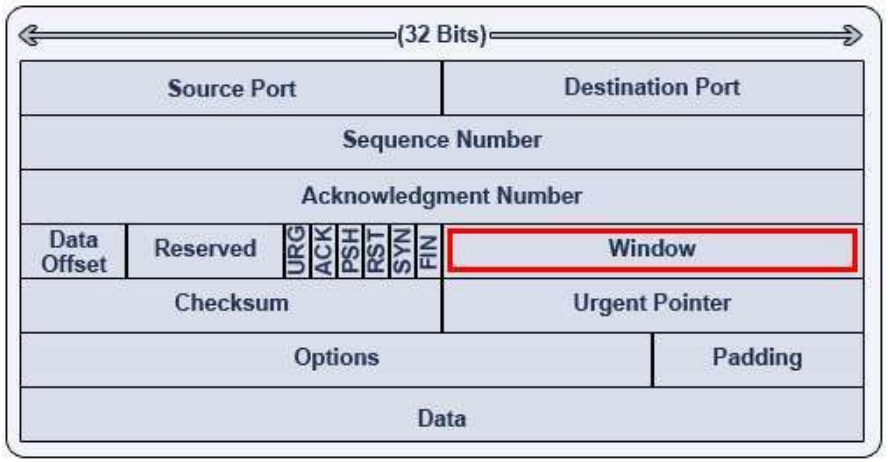
Le mécanisme de la fenêtre

- La fenêtre définit le nombre d'octets pouvant être envoyés par anticipation (sans attendre l'acquittement des octets précedemment transmis). Cela dépend de la capacité du *buffer* (tampon) du récepteur.
- TCP utilise le mécanisme de la **fenêtre coulissante** ou glissante (*sliding window*) : nombre d'octets maximum pouvant être émis sans attendre d'acquittement.



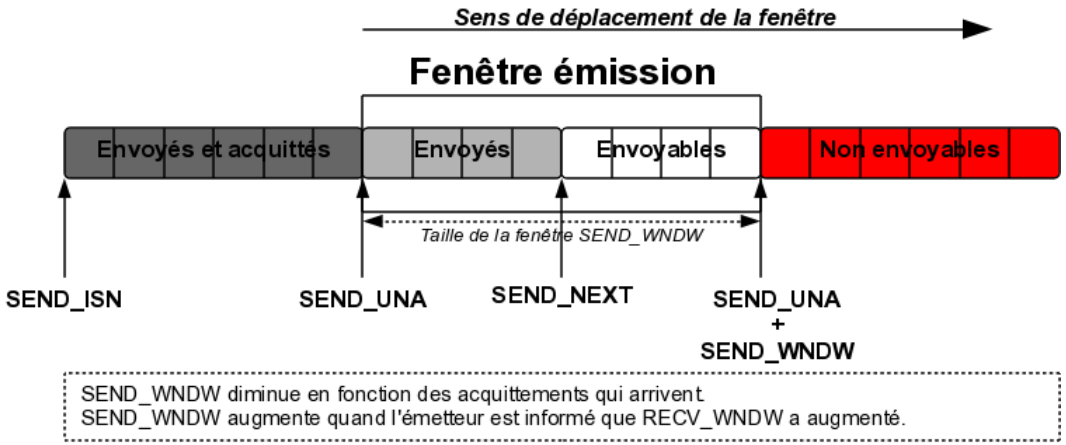
Window (1/2)

- Le champ ***Window*** (Fenêtre) est codé sur 16 bits et correspond au nombre d'octets à partir du numéro d'acquittement que le récepteur est capable de recevoir.
- Le destinataire ne doit donc pas envoyer les segments après son numéro de séquence + taille de la fenêtre.
- Cela permet aussi au récepteur de recevoir des segments hors séquence (trou) et de profiter des délais d'attente pour réorganiser les données.

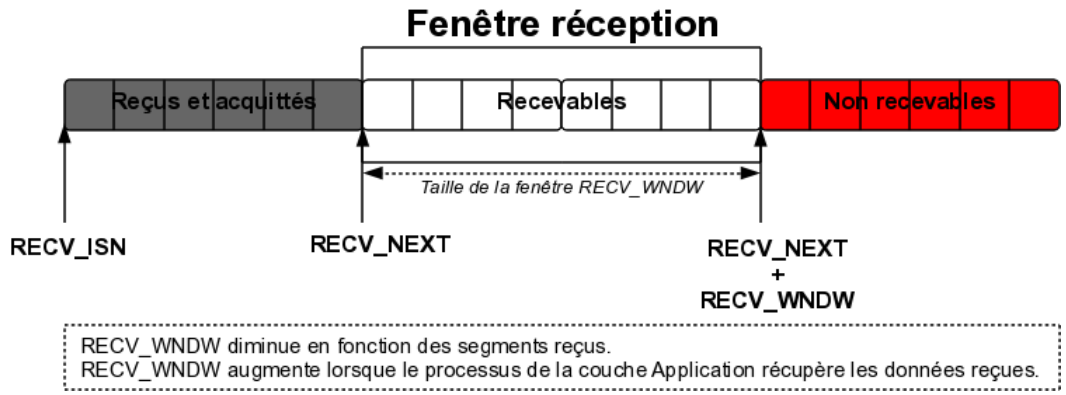


Window (2/2)

- Chaque machine gère localement ses fenêtres pour chaque sens de transmission (fenêtre d'émission et fenêtre de réception)

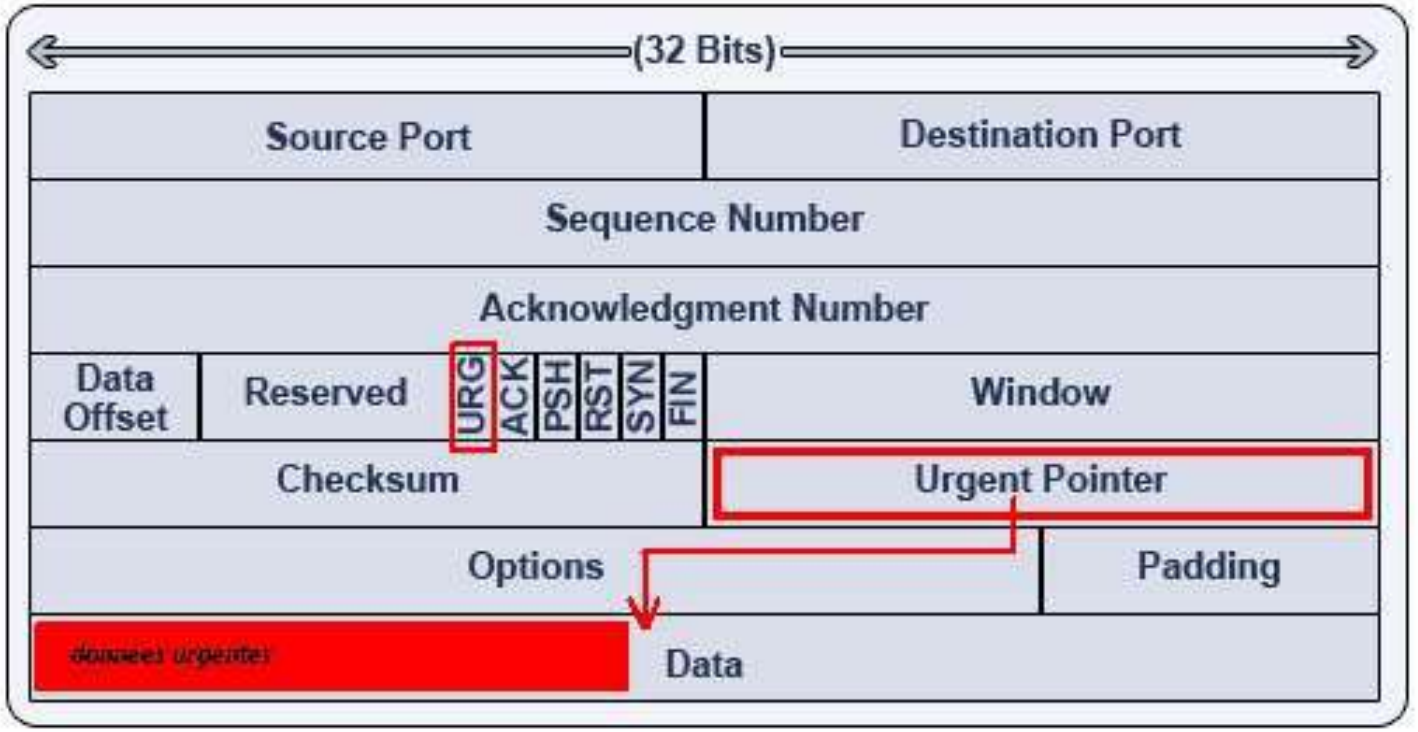



```
SI (SEND_NEXT - SEND_UNA == SEND_WNDW)
ALORS émission stoppée
FSI
```



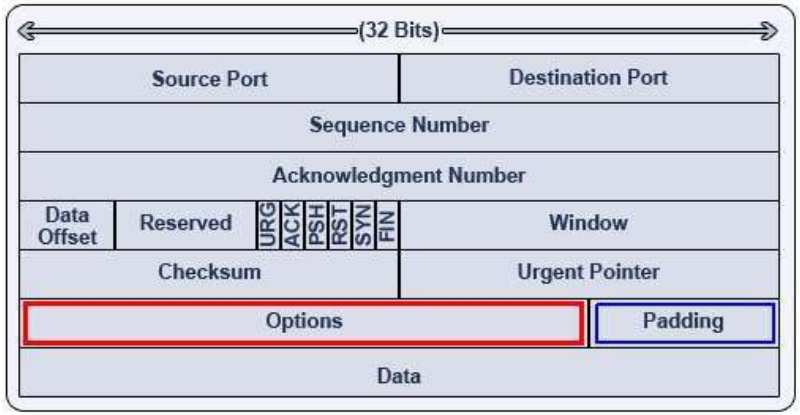
Données urgentes

- Cela permet de transmettre des données sans retard, qui doivent être traitées de manière urgente :
  - Le *flag* **URG** indiquera de prendre en compte le champ ***Urgent Pointeur***.
  - Les données urgentes sont toujours en tête du segment. Le ***Urgent Pointeur*** pointera donc sur la première donnée normale (non urgente).



Options

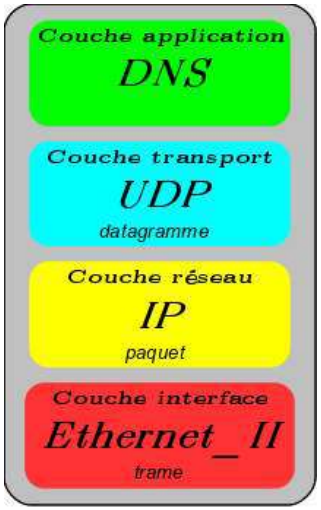
- Les **Options** peuvent occuper un espace de taille variable (ou nulle) à la suite de l'en-tête TCP. Les plus utilisées sont :
  - **mss** : au moment de la connexion, chaque partie annonce son MSS.
  - **timestamp** : utilisée pour calculer la durée d'un aller-retour (**RTT**, *Round Trip Time*).
  - **wscale** : facteur d'échelle qui permet de surpasser la limite des 16 bits du champ *Window*.
  - **sack** : utilisation des acquittements sélectifs.



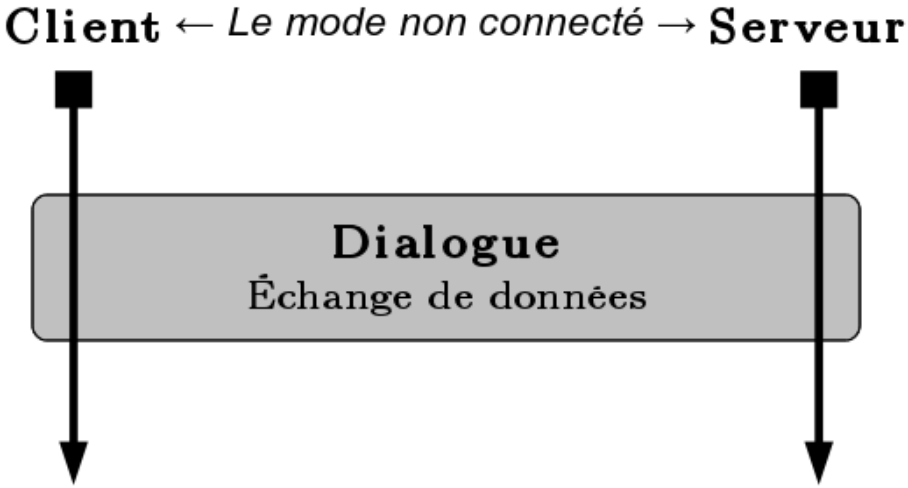
- Pour s'assurer que l'en-tête est toujours un multiple de 32 bits, il est parfois nécessaire d'ajouter du ***padding* (bourrage)**.

Le protocole UDP

- UDP (*User Datagram Protocol*) est un protocole souvent décrit comme étant **non-fiable**, en mode **non-connecté** (RFC 768), mais **plus rapide** que TCP. Il assure lui aussi la transmission des données de bout en bout (d'un processus à un autre processus).



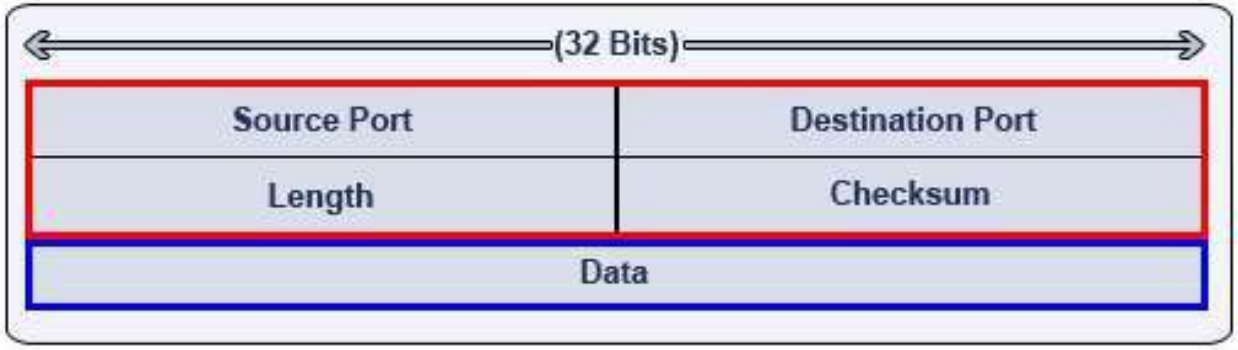
Modèle DoD



- Les protocoles UDP (et TCP) utilisent les **numéros de port** comme technique d'adressage des bouts d'une communication.

Datagramme UDP

- La communication UDP est basée sur un couple ***Source Port*/*Destination Port***, chacun codé sur **16 bits** soit $2^{16} = 65536$ ports distincts. L'attribution des ports est faite par le système d'exploitation, sur demande du processus de la couche Application.
- Le champ ***Checksum*** est codé sur **16 bits** et permettra de valider le datagramme UDP reçu : son calcul est identique à celui de TCP.



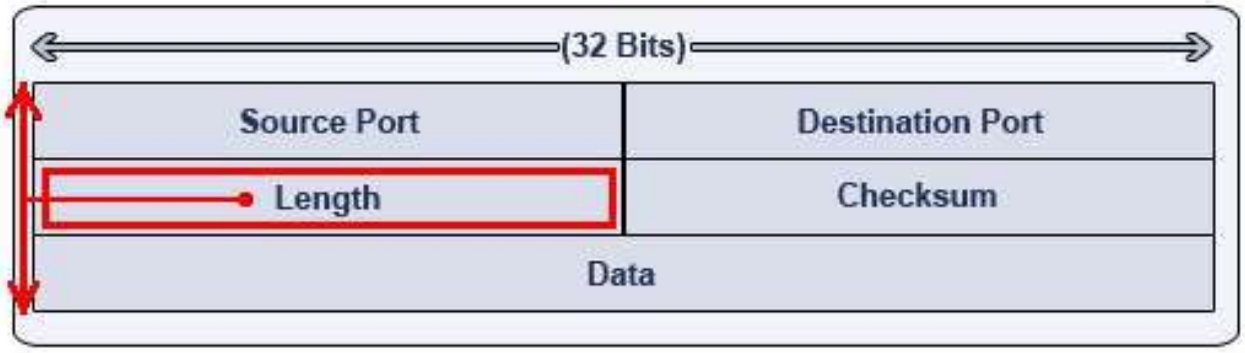
- Pour extraire les **données (*Data*)**, l'opération est simple et rapide car l'**en-tête** UDP est de **taille fixe** : il fait **8 octets**.

Taille d'un datagramme

- Le champ ***Length*** étant codé sur 16 bits, un datagramme aura une longueur totale maximale de $2^{16} = 65535$ octets (entête + data).

En pratique, on constate que :

- La plupart des systèmes limitent la taille des datagrammes à 8Ko (8192 octets).
- De nombreux protocoles de la couche Application utilisent un datagramme de 512 octets.



Caractéristiques

Avantage : UDP est un protocole rapide car

- simple (pas de connexion, pas d'états entre le client et le serveur)
- économique en bande passante (en-tête de 8 octets)
- sans contrôle de congestion donc UDP peut émettre sans attendre

Inconvénient : UDP est un protocole non fiable car

- pas d'acquittement donc pas de garantie de bon acheminement
- pas de détection de pertes donc pas retransmission possible
- pas de contrôle de flux et donc risque de saturation des *buffers*
- pas de séquencement donc les datagrammes peuvent être traités dans le désordre

90% des services d'Internet utilisent TCP !

Utilisation

Pour les applications multimédias car :

- tolérance aux pertes
- sensible au débit
- multicast possible
- Exemples : RTSP/RTP/RTCP/VoIP

Autres applications utilisant UDP car :

- faible volume de données
- pas besoin d'un service fiable
- Exemples : DNS, SNMP, BOOTP/DHCP

Pour réaliser des transferts fiables en UDP, il faut ajouter des mécanismes de reprise sur erreurs au niveau de la couche Application.