

Réseaux informatiques

RXI

Chapitre 10

TCP et UDP

1^{ère} partie



Objectifs d'apprentissage

- Savoir expliquer l'utilisation du protocole UDP
- Savoir décrire l'utilisation des numéros de port
- Comprendre les bases des mécanismes de TCP
 - Etablissement et terminaison de connexions
 - Numéros de séquence
 - Acquittement
 - Retransmission
 - Contrôle de congestion



La couche Transport

- Communication de bout en bout
- Services offerts à la couche supérieure
 1. Service sans connexion
 - Protocole UDP
 2. Service avec connexion
 - Protocole TCP
 - Fiable (garantie de délivrance)
 - Délivrance dans l'ordre de l'émission
 - Régulation de la vitesse (contrôle de congestion)

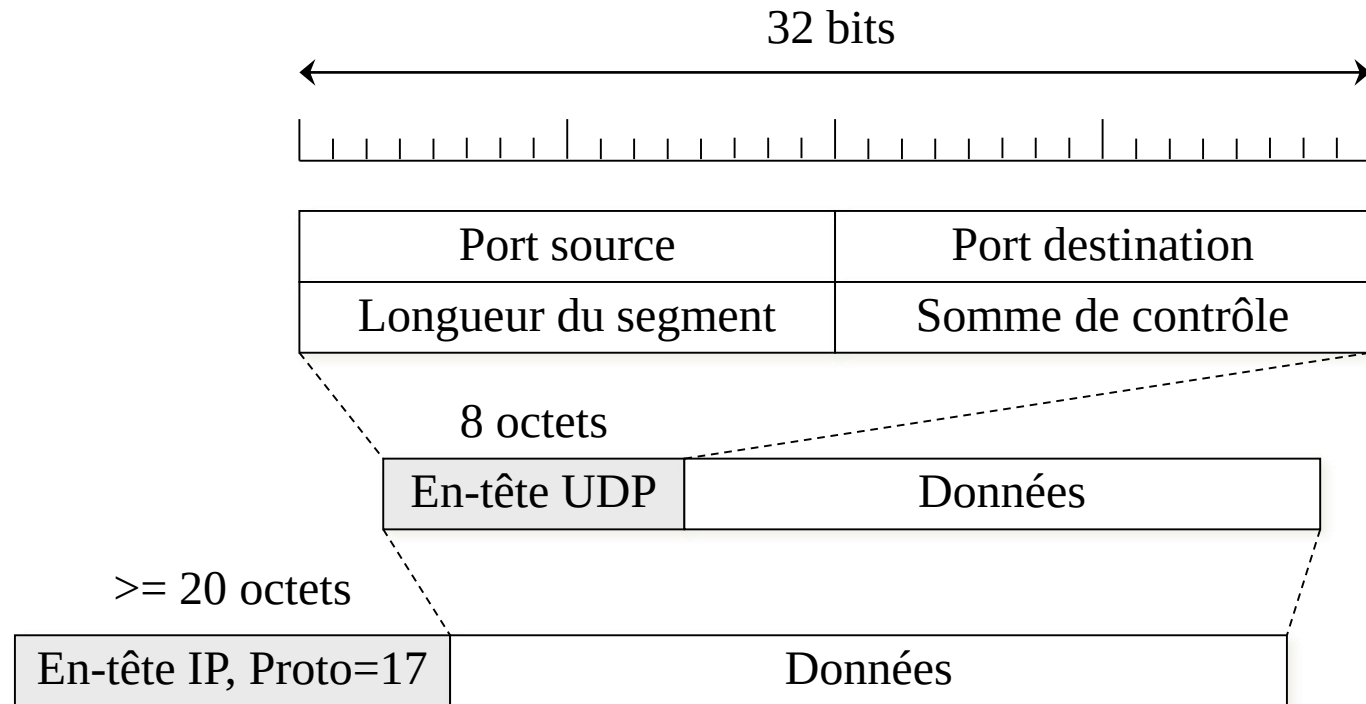


Le protocole UDP

- Fonctionnalités
 - Adressage d'applications à l'aide de numéros de port
 - Contrôle d'erreur optionnel (obligatoire avec IPv6)
 - Transmission non fiable
 - Sans connexion
 - Sans acquittement ou retransmission
 - Sans contrôle du débit
- Utilisé pour
 - Les transmissions multimédia (tolèrent quelques pertes)
 - Les transmissions multicast (multicast toujours sans connexion)
 - Les échanges courts, comme DNS (pas de connexion)



Format des datagrammes UDP

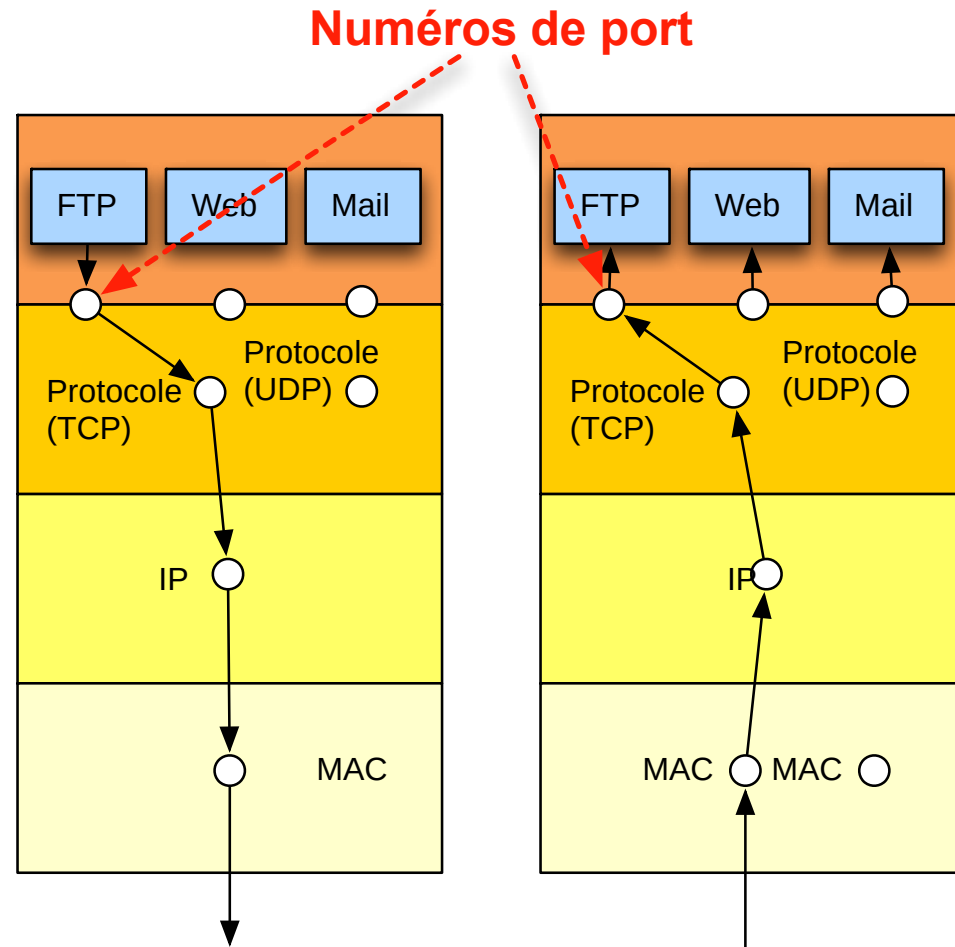


- Seulement 4 champs dans l'en-tête UDP, 8 octets en total
- Longueur maximum d'un datagramme: 65'535 octets



Numéro de ports

- Les numéros de ports sont les adresses de la couche Transport
- Permettent d'adresser plusieurs applications sur la même machine
- Entiers sur 16 bits
- Utilisés par TCP et UDP
- TCP et UDP peuvent réutiliser les mêmes ports



Les ports TCP / UDP

- Deux types de numéros de ports
 - Ports bien connus (1-1023)
 - Assignés à des applications standard
 - Typiquement utilisés par les applications serveur

Service	Port	Protocole
SSH	22	TCP
SMTP	25	TCP
DNS	53	UDP
HTTP	80	TCP

- Ports éphémères (1024 – 65535)
 - Typiquement utilisés par les applications client
 - Assignés dynamiquement par TCP/UDP



Exemples de code – serveur UDP en (pseudo-)C

```
int main(void) {  
    // Create UDP socket  
    int32_t sock = socket(AF_INET, SOCK_DGRAM, 0);  
  
    ...  
  
    // Bind socket to local port 1100  
    struct sockaddr_in sockaddr;  
    sockaddr.sin_port = htons(1100);  
    bind(sock, &sockaddr, ...);  
  
    ...  
  
    while(1) {  
        // Receive messages and obtain client address  
        nbytes = recvfrom(sock, data, 1023, 0, &clientaddr, ...);  
  
        ...  
    }  
}
```



Exemples de code – client UDP en (pseudo-)C

```
int main(void) {  
    // Create socket  
    int32_t sock = socket(AF_INET, SOCK_DGRAM, 0);  
    ...  
    // Fill destination address structure  
    struct sockaddr_in serveraddr;  
    serveraddr.sin_port = htons(1100);  
    inet_pton(AF_INET, "193.134.1.2", &serveraddr.sin_addr);  
    ...  
    // Send a message  
    char request[] = "This is a request";  
    nbytes = sendto(sock, request, ..., &serveraddr,...);  
    ...  
}
```



Le protocole TCP

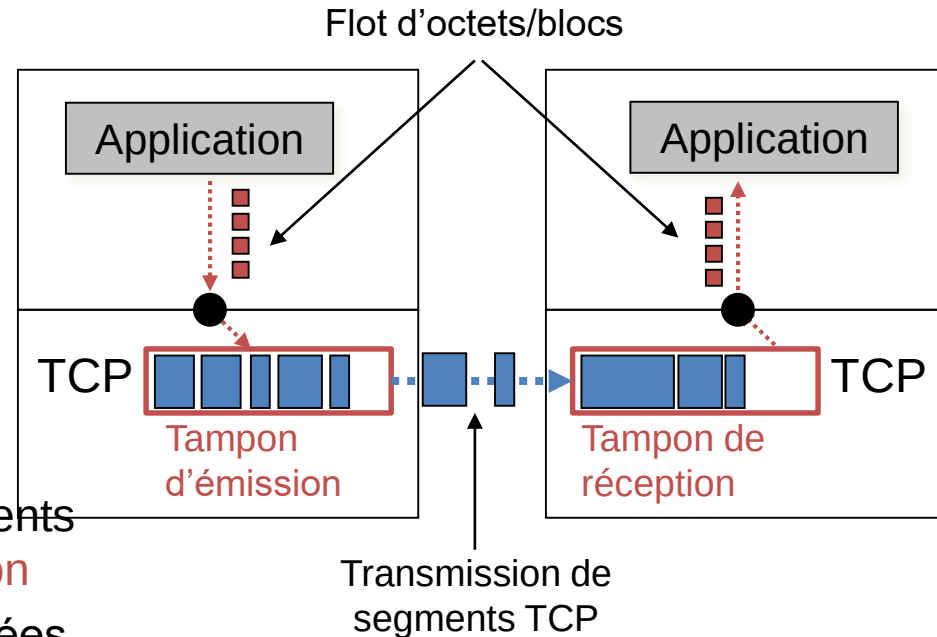
- TCP: *Transmission Control Protocol*
- Fonctionnalités principales
 - Transmission fiable de bout en bout
 - Numéros de séquence
 - Acquittements
 - Retransmission
 - Etablissement et terminaison de connexions
 - Connexions bidirectionnelles et point-à-point
 - Régulation du débit
 - Contrôle de flux : adaptation de la vitesse au récepteur
 - Contrôle de congestion : adaptation de la vitesse au réseau



Transfert tamponné à flot d'octets

- TCP offre à la couche supérieure un service à **flot d'octets**

1. L'application passe des blocs de données à TCP
2. TCP les met dans un **tampon d'émission**
3. TCP regroupe les données en **segments** à transmettre
4. Le récepteur TCP place les segments reçus dans un **tampon de réception**
5. L'application lit des blocs de données, sans tenir compte de segments



- La délimitation des messages de l'application n'est pas respectée**

On peut dire « *send(data) != recv(data)* » : les données reçues ne correspondent pas aux données envoyées

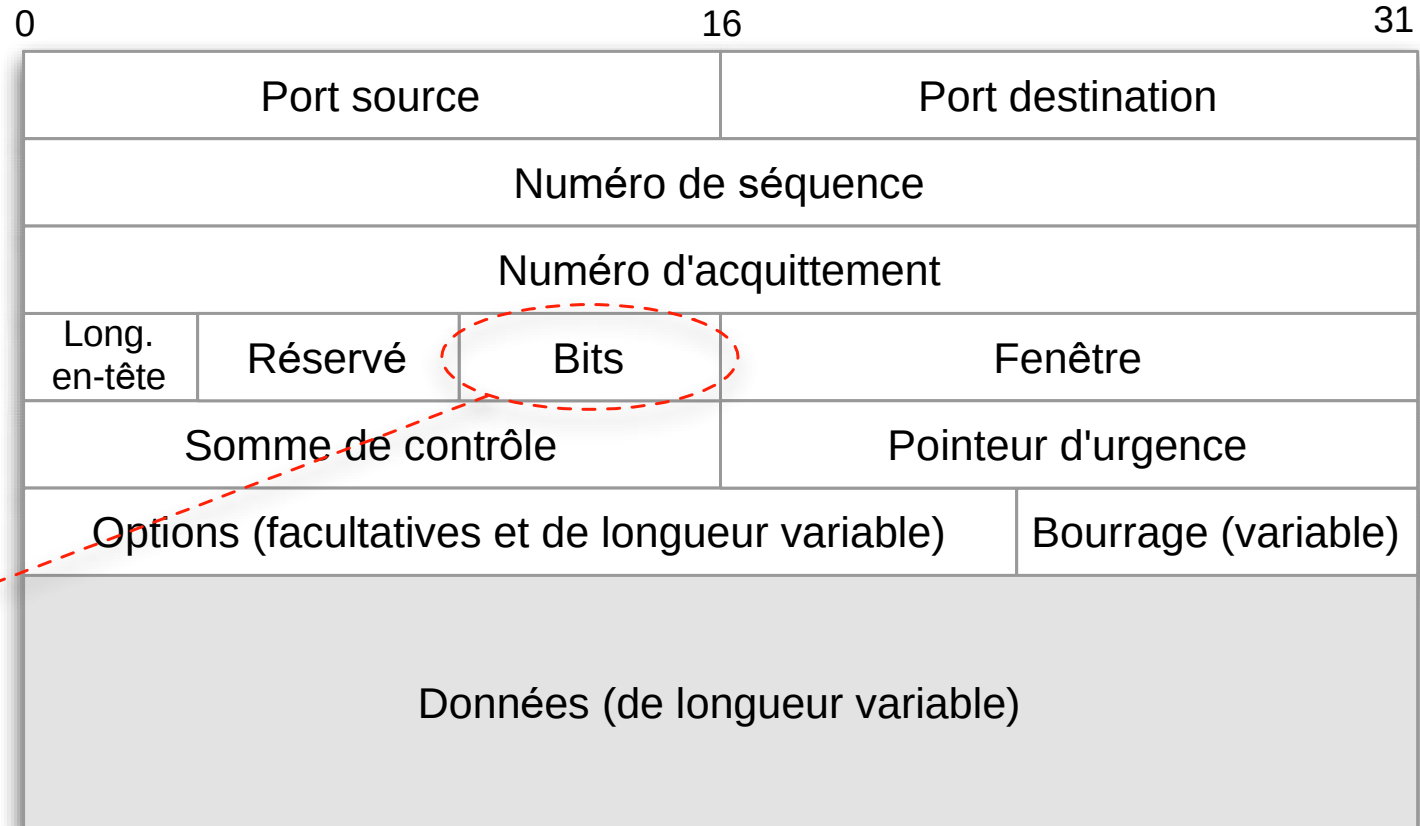


Transmission fiable

- Garantie de délivrance des données, dans l'ordre de l'émission
- Mécanismes nécessaires
 1. Numéros de séquence
 2. Acquittement
 3. Retransmission



Format des paquets TCP

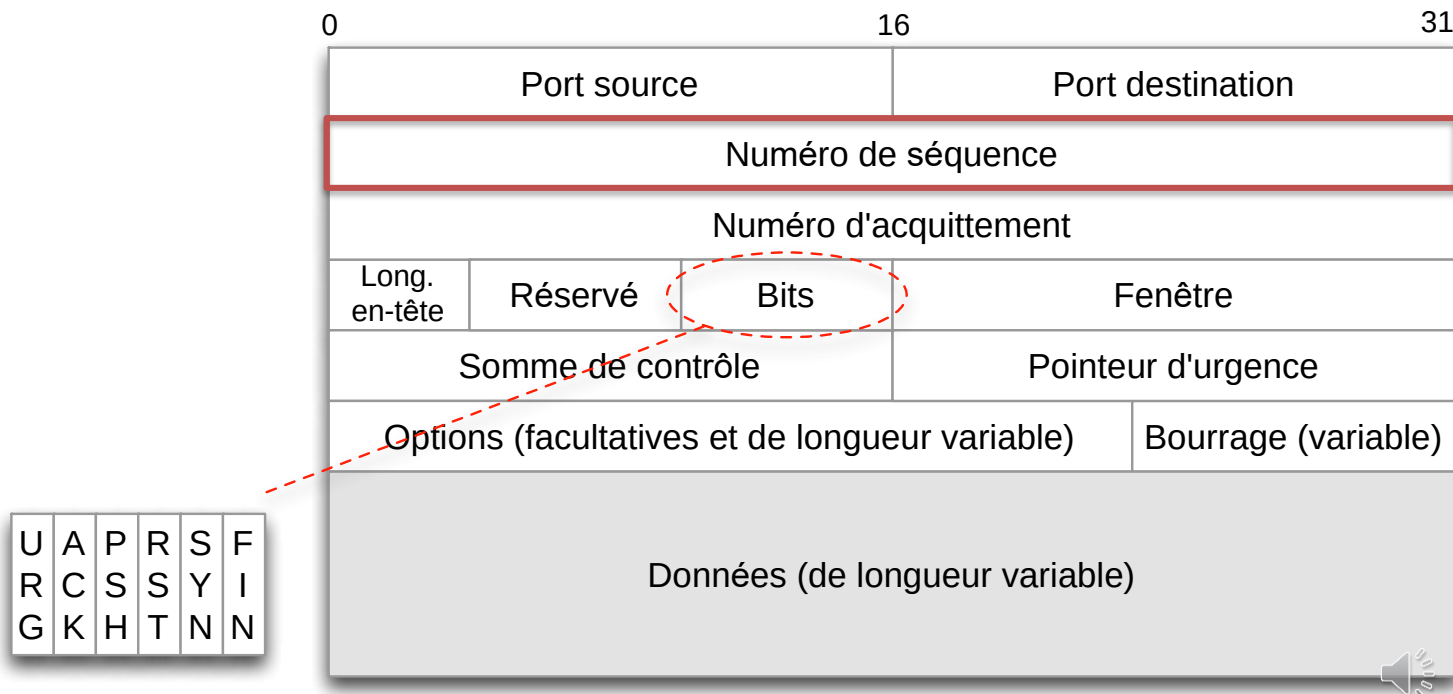


U	A	P	R	S	F
R	C	S	S	Y	I
G	K	H	T	N	N



Numéros de séquence

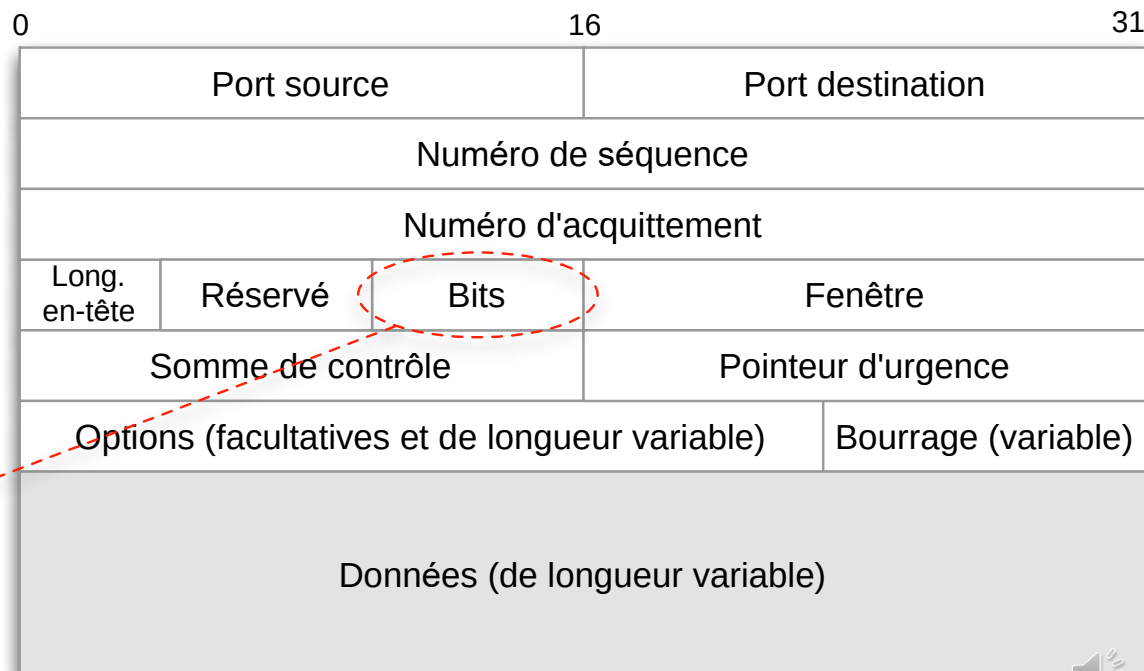
- Les octets sont numérotés, non pas les segments
- L'émetteur indique le numéro du premier octet des données du segment



Acquittement

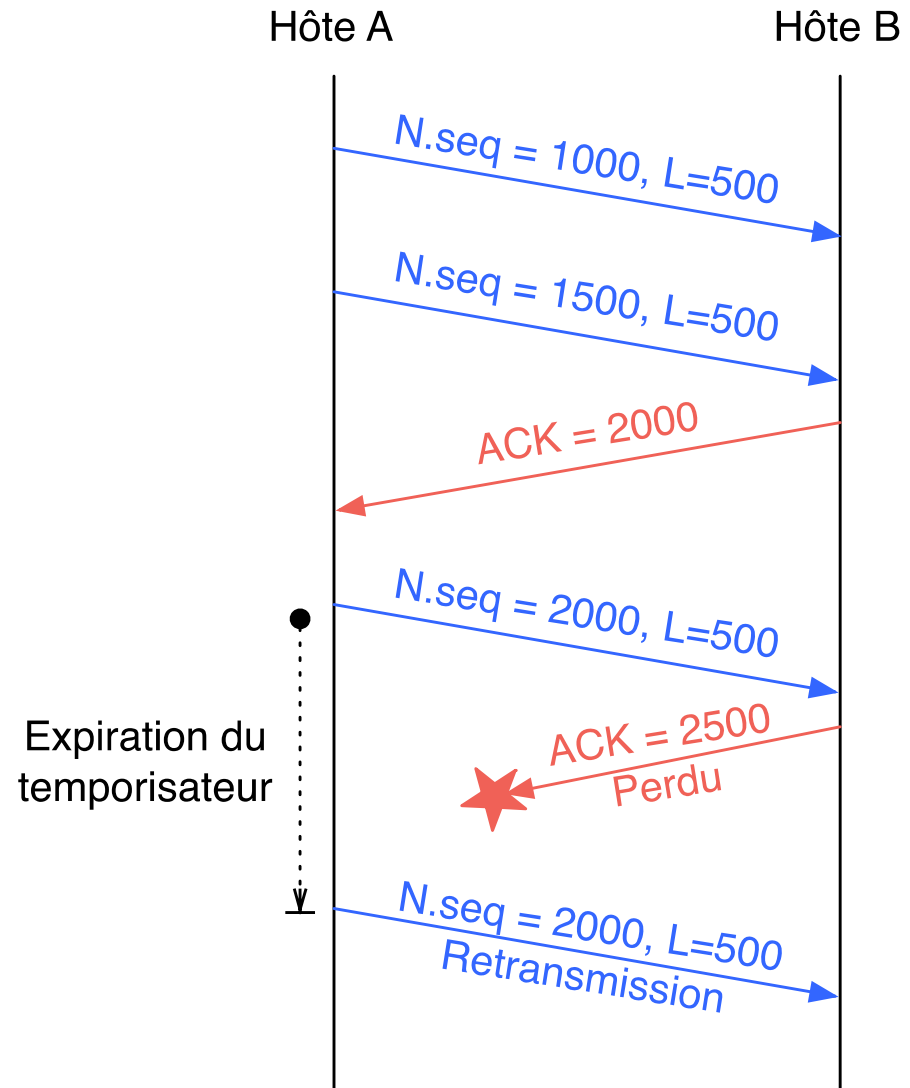
- Le numéro d'acquittement indique **le prochain octet attendu** par le récepteur
- Un acquittement peut être envoyé séparément ou combiné avec des données
- Les acquittements sont **cumulatifs** :
 - Un acquittement confirme la réception de toutes les données jusqu'à ce point

C	E	U	A	P	R	S	F
W	C	R	C	S	S	Y	I
R	E	G	K	H	T	N	N



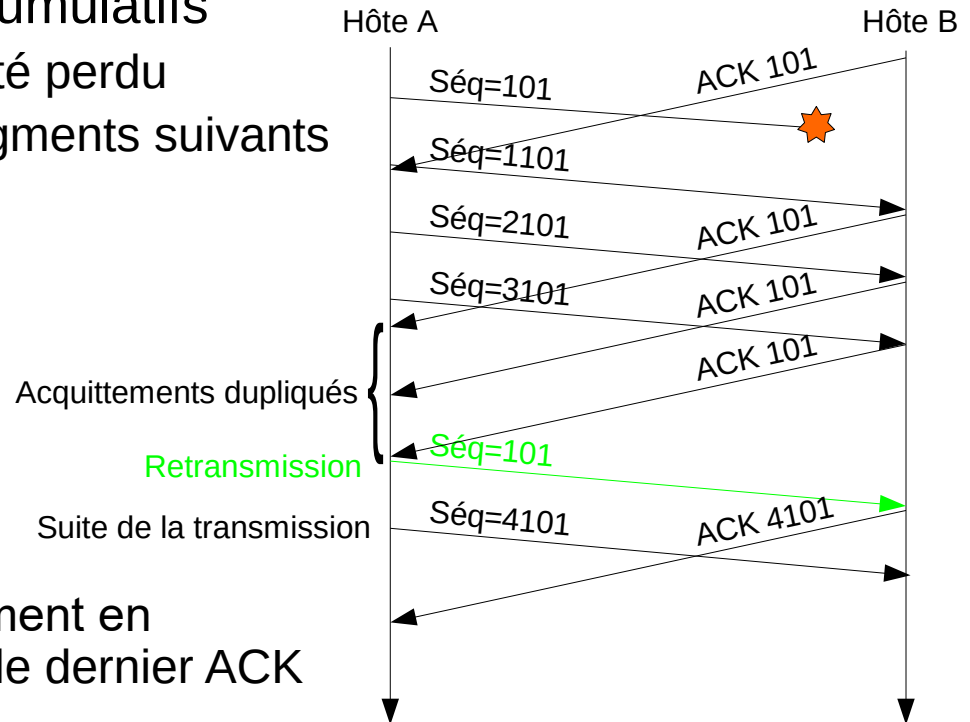
Retransmission après timeout

- L'émetteur gère un temporisateur de retransmission pour chaque segment envoyé
- Si le temporisateur expire avant la réception d'un acquittement, le segment est retransmis
- TCP a un mécanisme pour adapter la durée du temporisateur



Retransmission rapide

- Problème des acquittements cumulatifs
 - Un segment intermédiaire a été perdu
 - Comment signaler que les segments suivants sont arrivés ?



- **Solution :**
 - Lors de la réception d'un segment en désordre, le récepteur répète le dernier ACK ('**ACK dupliqué**')
 - Si l'émetteur reçoit **3 ACK dupliqués**, il retransmet le segment **sans attendre un timeout**



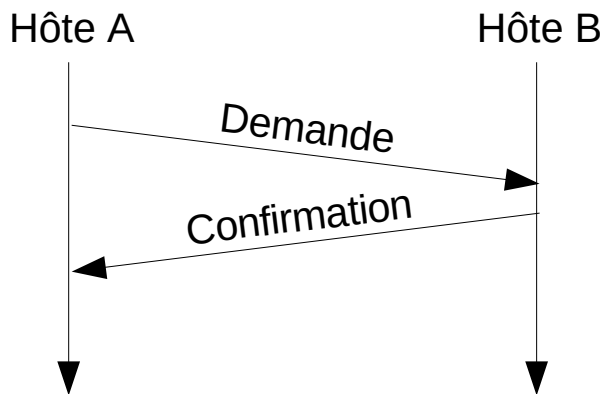
Acquittements et retransmission

- En résumé, TCP détecte la perte de paquets par deux événements
 - Timeout du temporisateur de retransmission
 - Aucun acquittement n'a été reçu
 - C'est-à-dire plusieurs paquets ont été perdus
 - Signale typiquement une congestion sévère
 - Trois acquittements dupliqués
 - Un ou plusieurs paquets ont été perdus
 - Mais quelques paquets sont arrivés correctement
 - Signale une congestion légère



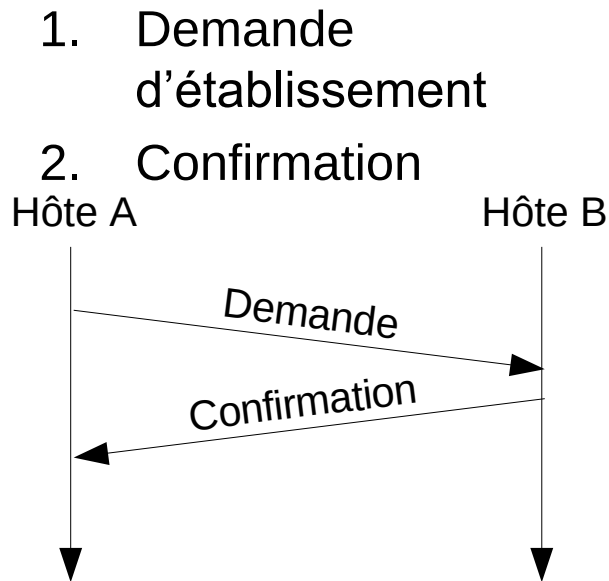
Etablissement de connexions

- C'est simple ? Non !
- Protocole simpliste :
 - Etablissement de connexion en deux temps
 1. Demande d'établissement
 2. Confirmation



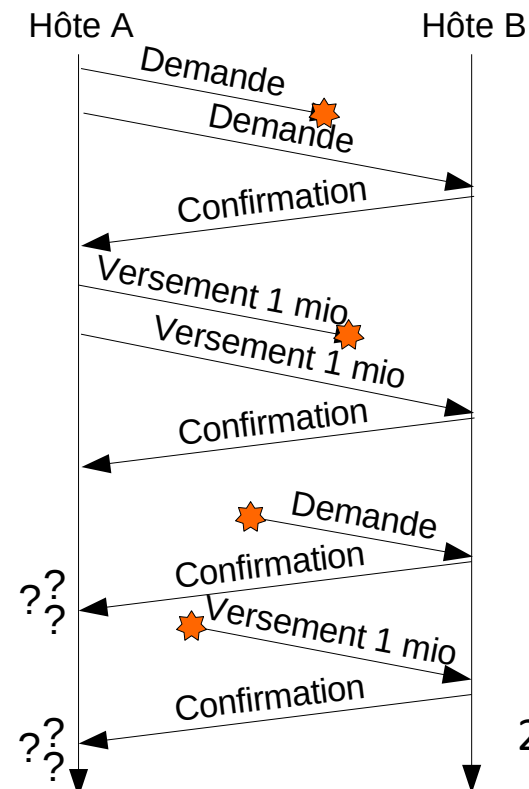
Etablissement de connexions

- C'est simple ? Non !
- Protocole simpliste :
 - Etablissement de connexion en deux temps



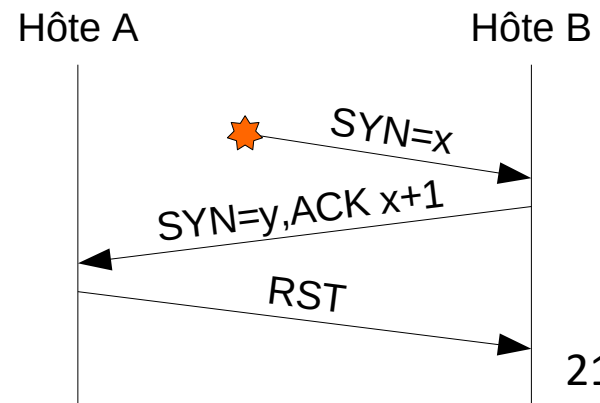
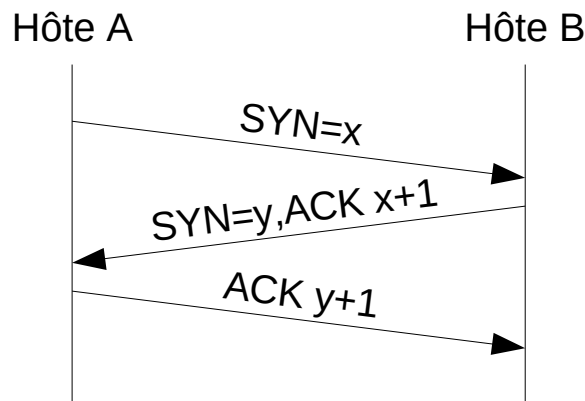
Scénario

- Ordre de versement à une banque
- Certains paquets du client sont retardés et retransmis



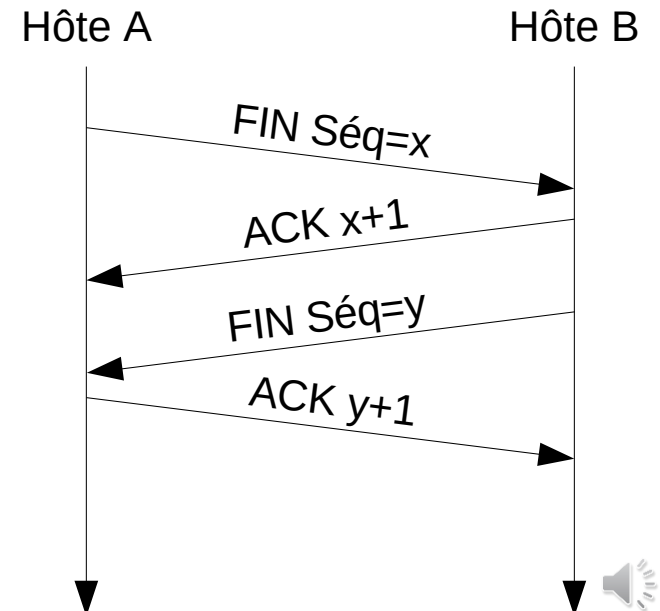
Etablissement de connexions dans TCP

- Le protocole doit tenir compte de la possibilité de doublons
- Mécanisme de TCP : *Three way handshake* (établissement de connexion en trois temps)
 - Les hôtes se mettent d'accord sur les numéros de séquence initiaux
 - Lors d'une erreur, un message RST (RESET) permet d'interrompre l'établissement de connexion



Libération d'une connexion dans TCP

- Les connexions TCP sont bidirectionnelles
 - Libération séparément dans les deux sens
 - Un hôte qui n'a plus de données à transmettre peut fermer la connexion dans une direction
 - L'autre hôte peut continuer à transmettre
 - Deux segments pour libérer un sens de la connexion:
FIN + ACK



Fin partie 1

