Niveau Transport
"Transport Layer"

I) Problèmes et solutions au niveau transport
II) Exemple des protocoles et services de transport dans l'INTERNET

Niveau Transport
"Transport Layer"

Chapitre I

Problèmes et solutions au niveau transport

Généralités: choix de conception
Adressage
Gestion des connexions
Transfert de données

Objectifs du niveau transport

- **Offrir un service de transmission d'informations** pour les besoins d'un utilisateur de session.
  - **Communication de processus à processus** (de bout en bout),
  - **Selon des objectifs de qualité** de service: transport fiable, multimédia
  - **Efficace** : en performances.
  - **Indépendant** de la nature des réseaux traversés: LAN, MAN, WAN
- **Importance du niveau transport**
  - **La couche transport n'est pas une couche de plus** dans une architecture de réseau.
  - **C'est la couche qui résume toutes les couches associées à la transmission** (les couches basses).
  - **C'est la base** sur laquelle repose une application répartie.

Nécessité de la couche transport: 1) Problème d'adressage

- **Niveaux liaison et réseau** : adresses utilisées = Adresses d'équipements physiques (contrôleur de communication, NIC).
  - Exemple : adresses Ethernet, adresses X25, adresses IP.
- **Niveau transport** (communication de bout en bout) : adresses utilisées = adresses d'applications (processus).
- **Toute solution** visant à utiliser les adressages physiques pour faire communiquer des processus est plus ou moins "bricolée".
  - Exemple : Utilisation d'adresses X25 ou Ethernet 'étendues'
  - On ne peut pas faire dépendre la structure de l'adresse d'une application de la nature du réseau qui permet de l'atteindre.
- **Le niveau transport doit offrir un adressage en propre:**
  - On utilise une adresse "logique d'activité" (adresse transport)
  - On réalise la mise en correspondance d'une adresse transport avec une "adresse physique d'hôte" qui est support de l'application.
  - L'adresse de transport est indépendante du réseau utilisé.

Nécessité de la couche transport: 2) Gestion de connexions et QOS

- **Protocole avec connexion**
  - **Permet la spécification de qualité de service** selon les besoins des applications.
  - **Le transport adapte aux besoins des applications** les niveaux physique, liaison et réseau.
  - **Exemples : TCP, RTP et ISO 8072**
- **Protocole sans connexion**
  - **Chaque donnée circule** en utilisant essentiellement des adresses de transport pour atteindre le destinataire.
  - **Protocole simplifié** pour des communications **rapides.**
  - **Exemples : UDP et ISO 8072 ad1**

Exemple de qualité de service: Le contrôle d'erreurs

- **Un utilisateur d'une architecture de réseaux ne doit pas avoir à se préoccuper des erreurs de transmission.**
  - Taux d'erreur résiduel acceptable (erreurs non détectées, non corrigées). Exemple 10^{-12}/bit pour des données informatiques de criticité normale.
  - La plupart des services de transmission offerts par les services réseaux n'offrent pas cette qualité.
    - Niveau liaison sans contrôle d'erreur : Ethernet, ATM.
    - Niveau réseau avec contrôle : X25 problème de panne de commutateur.
    - Niveau réseau à datagrammes sans contrôle d'erreur : IP.
- **Le niveau liaison ou le niveau réseau sont insuffisants pour les besoins en contrôle d'erreurs des applications.**
- **On reporte au niveau transport le lieu principal de correction des erreurs de transmission.**

Choix de conception du niveau transport

- **Services offerts:** selon les options de conception d'une couche transport.
  - **Gestion des connexions** avec négociation de qualité de service.
  - **Fragmentation** (segmentation).
  - **Groupage** (concaténation).
  - **Multiplexage / éclatement** des connexions de transport sur des connexions de réseaux.
  - **Contrôle d'erreur, de flux, de séquence**.
  - **Propriétés de qualité** de service temporelle.
  - **Contrôle de congestion** (du réseau par contrôle d'admission au niveau transport).

Analyse des concepteurs du transport Internet TCP/UDP

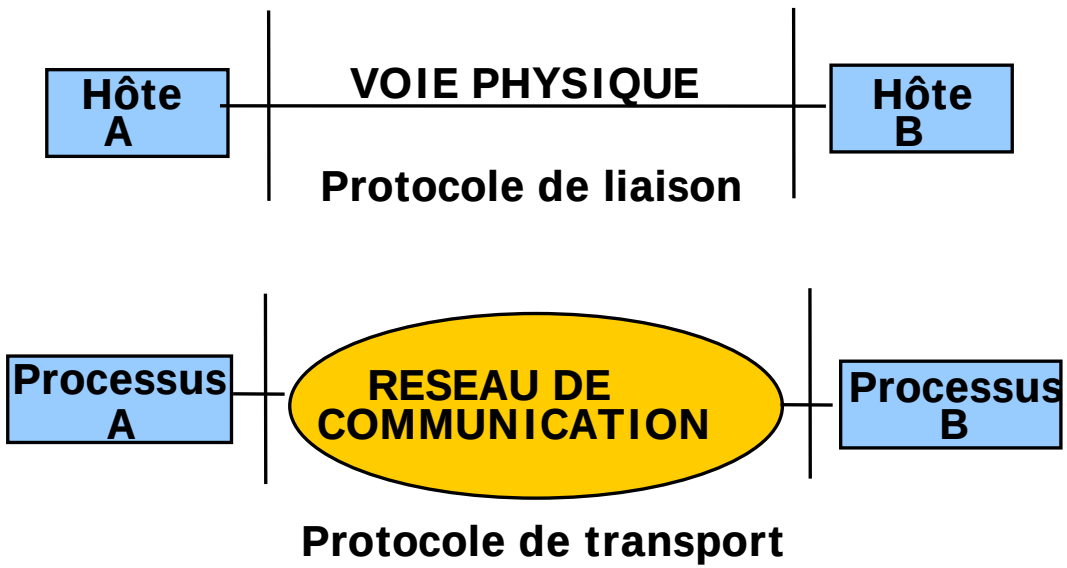
- **TCP : un protocole complet pour des données informatiques.**
  - **Résout les problèmes non traités par IP** (contrôle d'erreur, flux, séquence)
  - **=> TCP un transport fiable**
  - **=> Utilisable pour tous les types de réseaux.**
  - **Pas de négociation initiale**: fonctions toutes disponibles.
  - **Peu adapté** aux réseaux aux fonctionnalités élevées (X25).
  - **Beaucoup de traitements à effectuer**: solution lente.
  - Une solution très **optimisée** pour le service rendu.
- **UDP : un protocole très simplifié**
  - Pour aller vite: réalise uniquement un adressage transport.
- **RTP : un protocole de transport à QOS.**
  - Satisfaction de contraintes temporelles pour le multimédia.

Analyse des concepteurs du transport OSI

- **Etude de l'écart** entre le service à fournir par la couche transport et le service qu'elle obtient de la couche réseau.
- **Permet de déterminer** les fonctions à mettre en oeuvre au niveau de la couche transport.
  - **Meilleur est le service offert par le réseau** (surtout en termes de contrôle d'erreur, livraison en séquence)
  - **Plus simples sont les fonctions** du protocole de transport.
- **Conséquences**
  - **Définition d'un protocole à plusieurs classes (5 classes).**
  - **Classes de transport de base** (classe 0, 1, pour des niveaux réseaux performants) à **complexe** (classe 4 , très complète pour des niveaux réseaux médiocres).
  - **Protocole complexe** (volume de fonctions), négociation d'ouverture de connexion complexe => difficile à implanter.

Particularités des problèmes de réalisation du niveau transport

- **Le service de transport** parait **voisin** du service de **liaison**.
  - Communication en **point à point**.
  - Qualité de service comprenant **contrôle d'erreur et de flux.**
- **Pourrait-on alors utiliser au niveau de la couche transport un protocole de liaison ?**
  - Le problème principal : la différence des moyens de transmission utilisés



Modes de pannes supportées

- **Modes de pannes franches et transitoires dans les systèmes de communication :**
  - **Panne franche** : coupure totale des communications.
  - **Panne transitoire** : pertes aléatoires de message.

- **Au niveau physique comme au niveau réseau** : les deux niveaux supportent des coupures ou des pertes de messages.

= > Peu de différences dans les traitements de niveau liaison et de niveau transport.

Modes de pannes temporelles

- **Panne temporelle** : Délai de transmission anormalement élevé par rapport aux spécifications.
- **Niveau physique** : capacité de stockage du canal très faible (délai de transmission, délai de propagation)
  - => **le temps de transmission** d'une trame est **déterministe**.
- **Niveau réseau** : Le réseau peut **stocker des paquets** pendant un certain temps et ne les **délivrer qu'après ce délai**.
  - **Transmissions à longue distance** avec **surcharge et congestion**.
  - **Des paquets peuvent arriver après un long délai.**
  - => **Le temps de transmission** d'un paquet est **aléatoire (non déterministe).**
  - **Problème important au niveau transport : les "vieux paquets".**

Problèmes de causalité (déséquencement)

- **Niveau physique :**
  - Une voie de communication physique **a un comportement "causal":** les messages émis dans un certain ordre arrivent dans le même ordre.
  - **=> Les suites binaires ne peuvent se dépasser.**

- **Niveau réseau :**
  - Le réseau (en mode datagramme) peut **déséquencer ou dupliquer** les paquets transmis.
    - En raison des **algorithmes de routage** dans les réseaux à datagrammes.
    - En raison des **problèmes de délai de propagation** (déjà évoqués)
  - **=> Les paquets sont possiblement déséquencés.**

Problèmes et solutions au niveau transport

Problèmes d'adressage au niveau transport

Nature des adresses de transport
Adresse = Nom local du processus

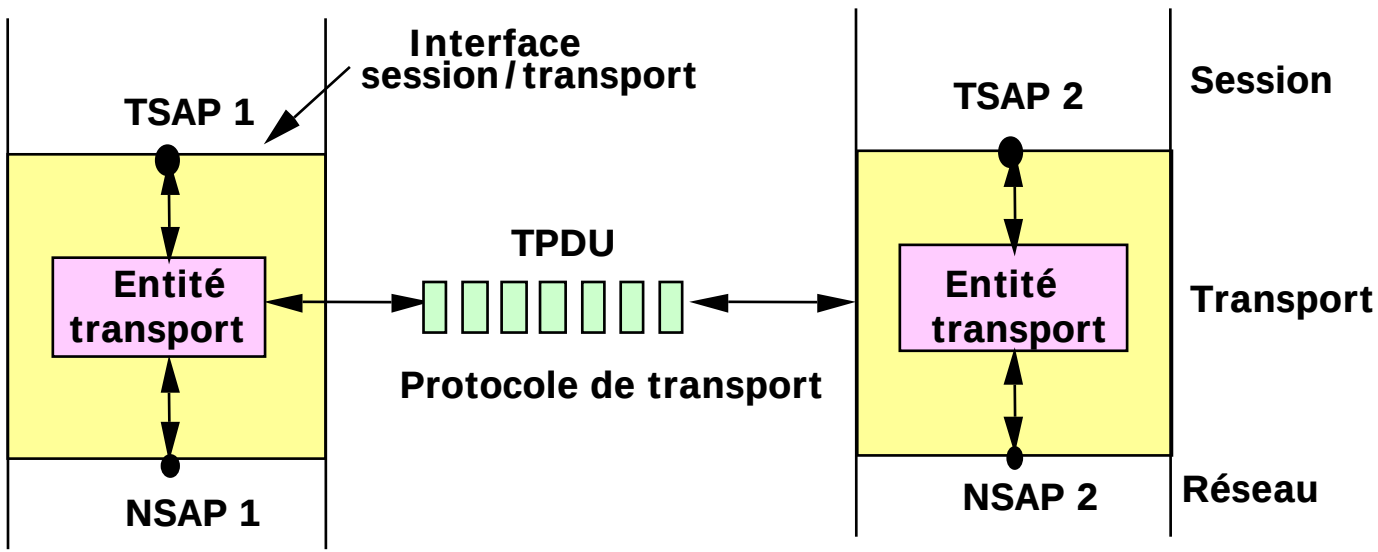
- **Nom local de processus** (source ou destinataire): nom du processus dans le systèmes d'exploitation de l'émetteur ou du destinataire.
- **Solution très médiocre pour résoudre le problème d'adressage :** variantes de syntaxe des noms de processus sur différentes machines.
  - **Gestion des noms** selon les différentes syntaxes.
  - **Difficulté de migration** d'une machine à une autre.
  - **Difficulté de remplacement** d'un processus en cas de panne (changement du nom).
  - **Difficulté pour la répartition de charge** : service rendu par plusieurs instances équivalentes d'un même code.

Nature des adresses de transport Adresse = nom global réseau

- **Nom global : L'adresse d'une source ou d'un destinataire est définie dans une syntaxe réseau unique pour toutes les machines.**
  - **Exemple : en TCP un entier sur 16 bits.**
- **Solution indépendante de la** syntaxe des noms de processus sur différentes machines.
- **Nom global = nom d'une file d'attente de message.**
  - **Besoin de plusieurs communications pour un processus**
  - **Indirection des messages** par un point de passage intermédiaire auquel se raccrochent les processus usagers.
  - **Terminologies pour les files de messages : Point d'accès de service transport, TSAP** "Transport Service Access Point"**,**
    - **Boite à lettre, Port, Porte, "Socket" ou prise.**

Détermination des adresses des processus communicants

- **L'ouverture d'une connexion de transport suppose la connaissance des adresses**
  - **TSAP** ("Transport Service Access Point") (Internet numéro de port) : sélection du processus qui rend le service.
  - **NSAP** ("Network Service Access Point") (Internet adresse IP) : sur quel site se trouve ce processus => Le routage réseau se charge ensuite de le trouver.
- **Problème de mise en correspondance TSAP <-> NSAP.**



Adressage dans une communication client-serveur

- **Scénario d'une communication**
  - **1 / L'utilisateur B (le serveur)** se rattache au TSAP **b** et se met en attente d'une arrivée de connexion.
  - **2 / L'utilisateur A (le client)** qui désire établir une connexion de transport avec B émet une requête de connexion sur le destinataire (NSAP b, TSAP b) en indiquant son adresse d'appelant (NSAP a, TSAP a).
  - **3 / Le serveur et le client** peuvent dialoguer.
- **Le scénario marche parce que B était en attente sur le TSAP** b et que A est supposé connaître b par son adresse.
- **Comment A peut-il connaître cette adresse :**
  - Problème de liaison analogue de l'édition des liens dans les programmes.
  - Solutions de liaison statique ou dynamique.

1) Solution de liaison statique

- **Connaissance des adresses codée dans les programmes**
  - **Tous les utilisateurs de B** ont la connaissance du TSAP de B.
  - **Analogie avec les numéros de téléphone réservés**: 15 , 18.
  - **On installe de façon définitive** un service sur un adresse donnée.
- **Principe simple mais limité.**
  - **Les serveurs dont l'adresse est fixe sont nécessairement assez peu nombreux** => services à caractère générique système.
  - **Gaspillage d'adresses** si l'on utilise une adresse permanente pour un serveur très spécifique ou rarement sollicité.
  - **Il est nécessaire d'utiliser aussi des adresses de TSAP non permanentes**.

2) Liaison dynamique par processus créateur de serveur

- **Notion de processus créateur de serveur (processus "logger")**
  - Processus dont le rôle est de **créer à la demande** des instances de serveur.
  - L'adresse du créateur de serveurs ("logger") est **permanente et connue de tous** les utilisateurs (solution de liaison statique).
- **Fonctionnement avec créateur de serveur sur un hôte NSAP B.**
  - Le client A se connecte au processus créateur et définit le service B à créer:
    - Soit sous la forme d'un nom logique de service (dans une liste)
    - Soit sous forme du nom d'un fichier accessible contenant une image binaire chargeable
  - Le processus créateur alloue une adresse de TSAP libre (**TSAP b**), créé une instance de B et lui attribue l'adresse allouée => il la communique au client A.
  - Le client A se connecte au **TSAP b** pour obtenir le service souhaité.
- **Avantages / inconvénients de la solution**
  - **Solution de désignation, de création et de liaison dynamique complète.**
  - **Ne marche que pour les serveurs pouvant fonctionner au coup par coup** (création à la demande) Exemple : serveur de compilation, serveur d'horloge,....
  - **Inutilisable pour un serveur qui fonctionne en permanence**

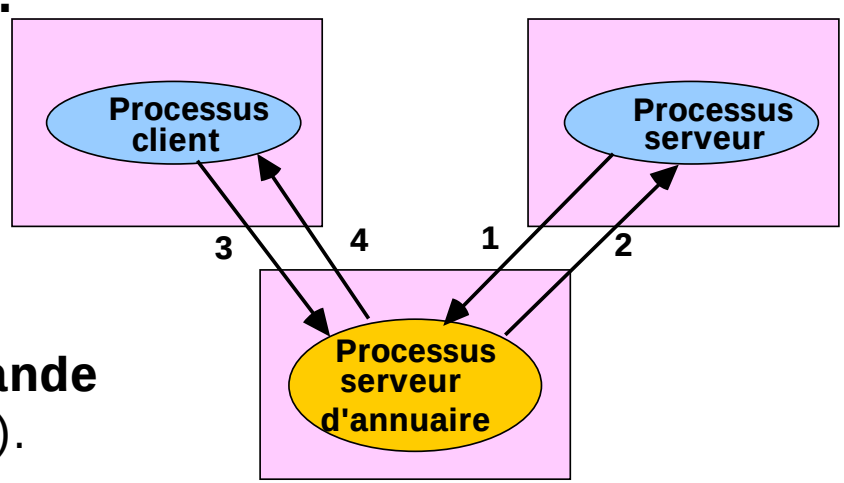
    Exemple : serveur de fichiers.

3) Liaison dynamique par utilisation d'un serveur d'annuaire

- **Principe de la solution**
  - Consiste à faire appel à un processus **serveur d'annuaire**, dont l'adresse (Nsap, Tsap) est permanente et connue de tous (solution statique).
  - Ce processus **gère un annuaire** de services auprès duquel les serveurs peuvent s'enregistrer et qui fournit des renseignements concernant un service:
    - Principalement son **adresse** (Nsap, Tsap)
    - Eventuellement d'autres attributs ou fonctions (protection, description/typage du service, ...).
  - **Solution analogue** de ce que réalisent les renseignements téléphoniques.

Troisième solution : liaison dynamique par utilisation d'un serveur d'annuaire

- **Étapes 1, 2 : Enregistrement dans une base de données des noms de serveurs.**
- 1) B établit une connexion avec le serveur d'annuaire. B émet une requête avec :
  - Le nom du service (chaîne de caractères).
  - L'adresse réseau d'accès au service.
  - L'adresse transport d'accès au service.
  - Tout attribut complémentaire si nécessaire.
- 2) Le serveur d'annuaire **acquitte la demande** d'enregistrement (absence d'homonymie, ...).



- **Étapes 3, 4: Liaison entre un client et un serveur.**
- 3/ A établit une connexion de transport avec le processus serveur d'annuaire.
  - Requête spécifiant le nom logique du service dont il désire connaître l'adresse.
- 4/ Réponse du serveur : l'adresse demandée. A se déconnecte de l'annuaire.

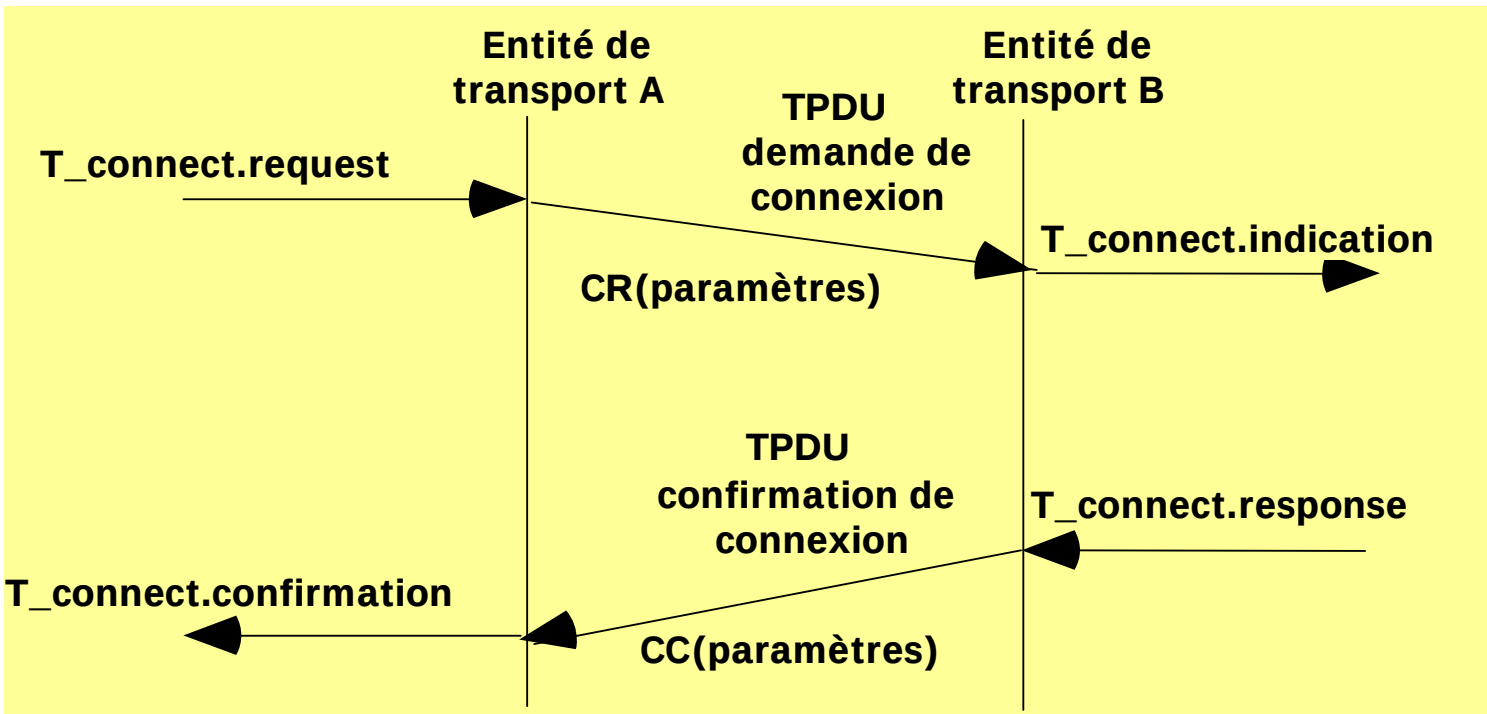
- A n'a plus qu'à établir la connexion avec le serveur.

Problèmes et solutions au niveau transport

Problèmes de gestion des connexions au niveau transport

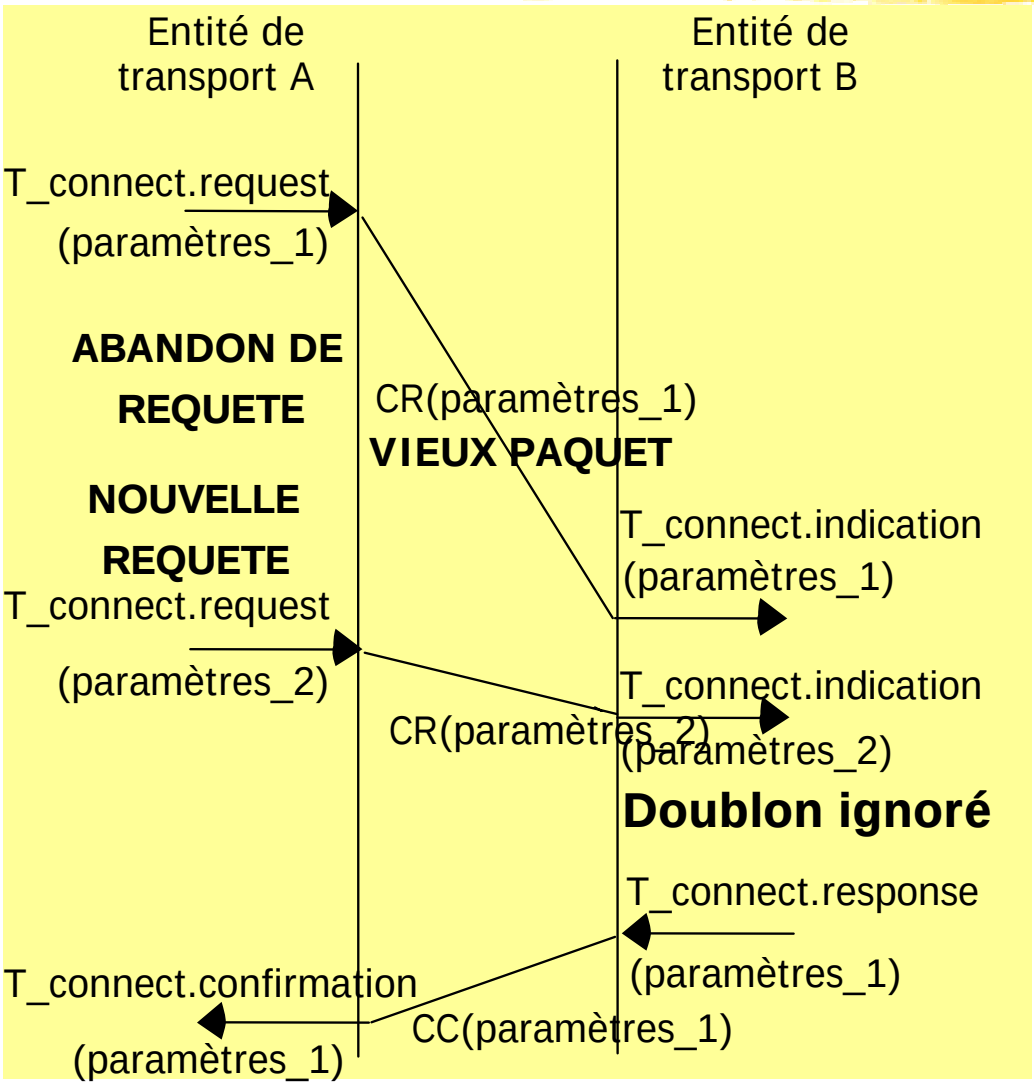
A) Problèmes d'ouverture de connexion
Rappel du schéma de base

- **Rappel : L'accord confirmé de base ('hanshake simple')**



- **Problème:** le réseau peut mémoriser des paquets pendant un temps très long (éventuellement non borné dans l'hypothèse asynchrone).

Exemple 1 : comportement incorrect du à un délai élevé



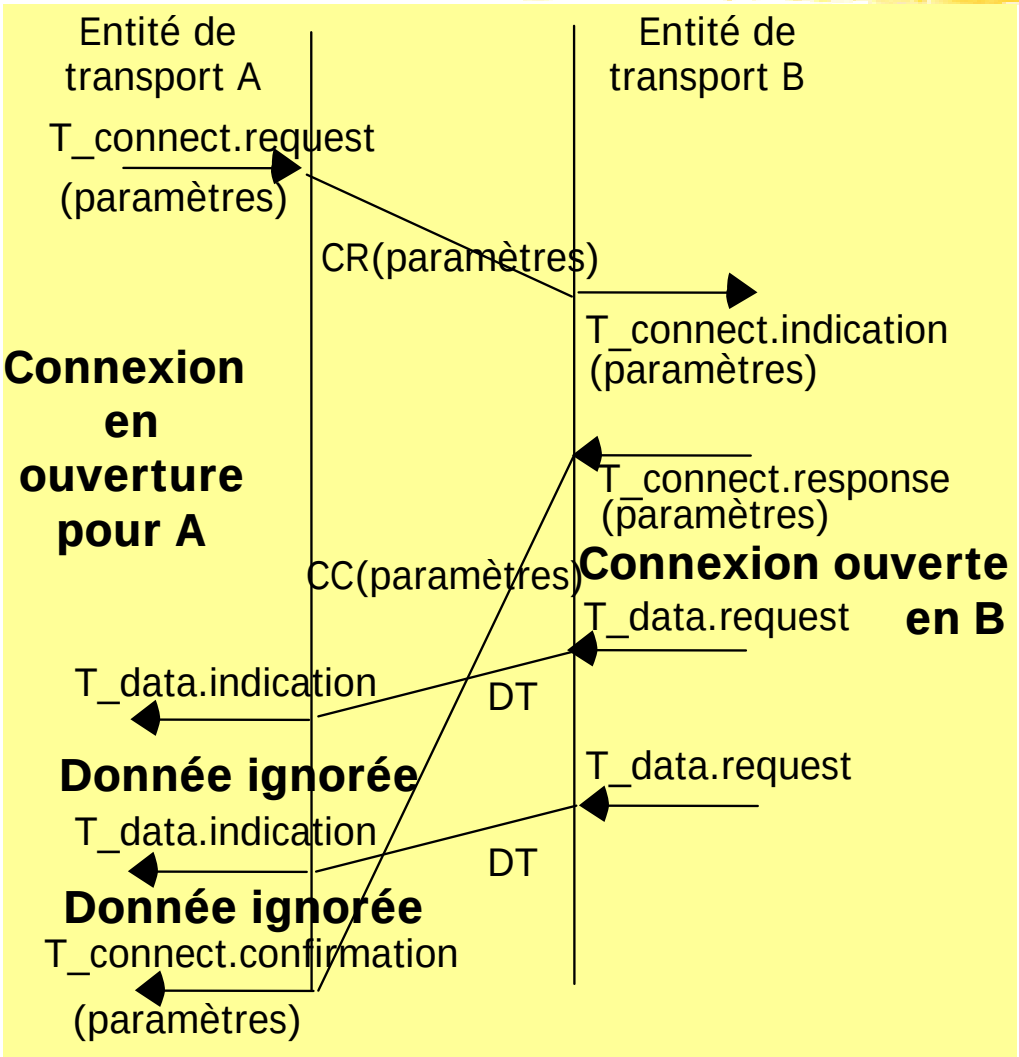
- **Seul responsable de l'erreur** le délai élevé de transmission du premier paquet.
- **Il n'y a pas d'erreur** de transmission et **la séquence des paquets est respectée**.
- On voit que le problème provient surtout **de l'absence d'identification des connexions.**

- **Solution à apporter:**

  Notion de référence de connexion

  et mécanisme de gel des références

Exemple 2 : comportement incorrect du au déséquencement



- **Responsable** le séquencement des paquets **non respecté.**
- D'où une connexion **considérée comme ouverte par un site B** et **non ouverte par l'autre** A (demi-ouverte).
- **La situation peut durer longtemps** si le paquet CC arrive très retardé (ou même n'arrive jamais).

- **Solution à apporter:**

  Ouverture de connexion confirmée en trois étapes ("three way handshake")

Notion de référence de connexion

- **Besoin de créer des références de connexion :**
  - **Un moyen de désigner** une connexion de transport : une communication entre un émetteur et un destinataire.
  - **Un usager peut gérer sur un même TSAP** plusieurs connexions simultanément avec le même ou plusieurs usagers.
- **Problème posé :**
  - Générer à la demande des références uniques pour de nombreux couples d'usagers avec de nombreuses connexions ouvertes.
- **Solution :**
  - Pour chaque site associer un entier unique pour le site et le nom du site : notion d'estampille ('timestamp').
  - Former la référence unique à partir du couple des estampilles émetteur destinataire.
- **Comment créer des entiers uniques ?**

Solutions pour les références : 1) Adresses TSAP à usage unique

- **Référence entière basée sur l'adresse de TSAP**
  - On doit utiliser une adresse de TSAP différente à chaque connexion
  - On s'oblige à ne gérer qu'une connexion par TSAP.
  - Chaque fois qu'on a besoin d'une nouvelle référence de connexion, on crée une nouvelle adresse de transport.
- **Il ne peut plus y avoir d'ambiguïté** entre messages circulant sur des connexions différentes ou en cours d'établissement.
- **Lorsqu'une connexion est fermée**, toutes les messages qui lui étaient associés sont inutilisables (sans destinataire).
- **Solution coûteuse** en adresses de TSAP.
- **Solution qui rend très difficile la liaison** (découverte de TSAP).

Solutions pour les références : 2) Références volumineuses

- **Utilisation d'une référence de connexion longue** contenant des informations de différenciation pour la rendre unique.

- **A) Référence basée sur la date et l'heure d'ouverture de connexion**.
  - Solution correcte chaque référence est différente.
  - Nécessite une gestion correcte du temps absolu.
- **B) Référence basée sur un numéro de séquence.**
  - **Utilisation séquentielle** d'un espace de références grand => il ne repasse par les mêmes numéros qu'avec une faible probabilité.
  - **Nécessite une sauvegarde en mémoire stable** du dernier numéro généré car en cas de panne on ne doit pas réutiliser le même numéro (démarrage à 0 au boute).
- **C) Tirage aléatoire d'une référence** dans un grand espace de numérotation tel qu'on ne peut tirer deux fois la même référence qu'avec une faible probabilité (exemple sur 32 bits ou mieux sur 64 bits).
  - Nécessite un bon générateur d'entiers : avec probabilité faible de collision.
  - Solution probabiliste : on n'est jamais sur de l'absence de collision.
  - La référence unique aléatoire peut-être utilisée comme base de numérotation en séquence des messages.

Solutions pour les références : 3) Référence courte avec gel

- **Utilisation d'une référence de connexion courte réutilisable**.
  - **En général un entier qui correspond à une entrée** dans la table des descripteurs de connexions.
  - **A chaque fois qu'une connexion doit être ouverte** on utilise une entrée libre de la table.
  - **L'index entier est sur n bits, 2^n est petit** et il y r**éutilisation fréquente** des références par des connexions successives.
  - **Il ne faut pas qu'il y ait d'ambiguïté** entre les messages de la précédente connexion utilisant l'entrée et ceux de la nouvelle connexion.
- **Solution** : **gel de référence**
  - **Une référence ne peut être réutilisée** pendant une période suivant la fermeture d'une connexion **= > Gel de la référence**.
  - **Le gel permet à tous les paquets en circulation** sur la connexion fermée d'être reçus et purgés.
  - **On doit disposer d'un mécanisme de limitation de la durée de vie** dans le réseau ('time to live') pour armer un délai de garde correspondant au gel.

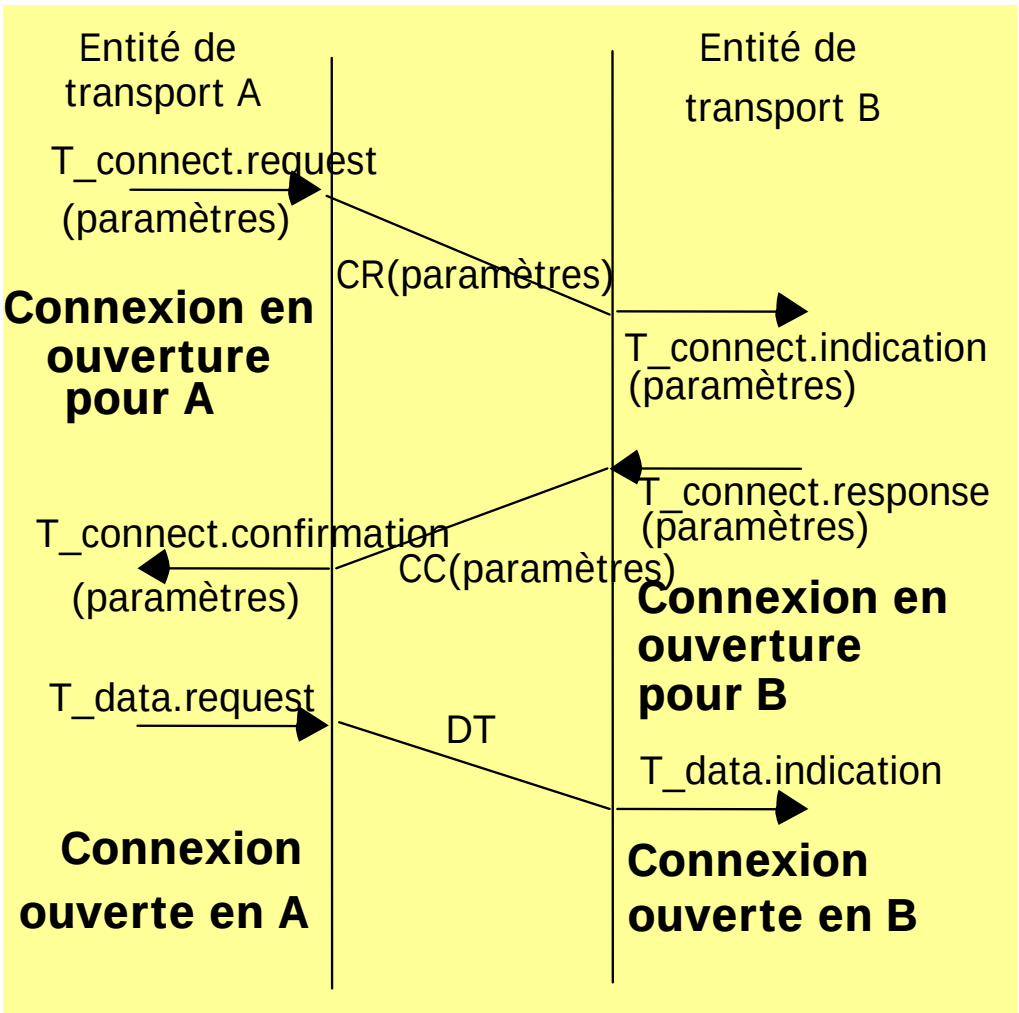
La solution en TCP

- **Utilisation d'une solution ‘hybride’**.
- **La référence est basée sur une horloge ‘temps réel’**
  - Cette horloge est très lointainement reliée au temps absolu.
- **Il faut plutôt considérer que l’horloge de TCP fonctionne comme un générateur de nombres aléatoires**
  - Solution de génération d’entiers probabilistes sur 32 bits.
  - La référence est utilisée comme base de numérotation en séquence des messages.
- **Crainte des collisions entre numéros :** utilisation en plus d’un mécanisme de **gel de référence**
  - **Notion de durée de vie** des paquets qui permet de dimensionner le gel.

Ouverture de connexion en trois étapes ("three way handshake")

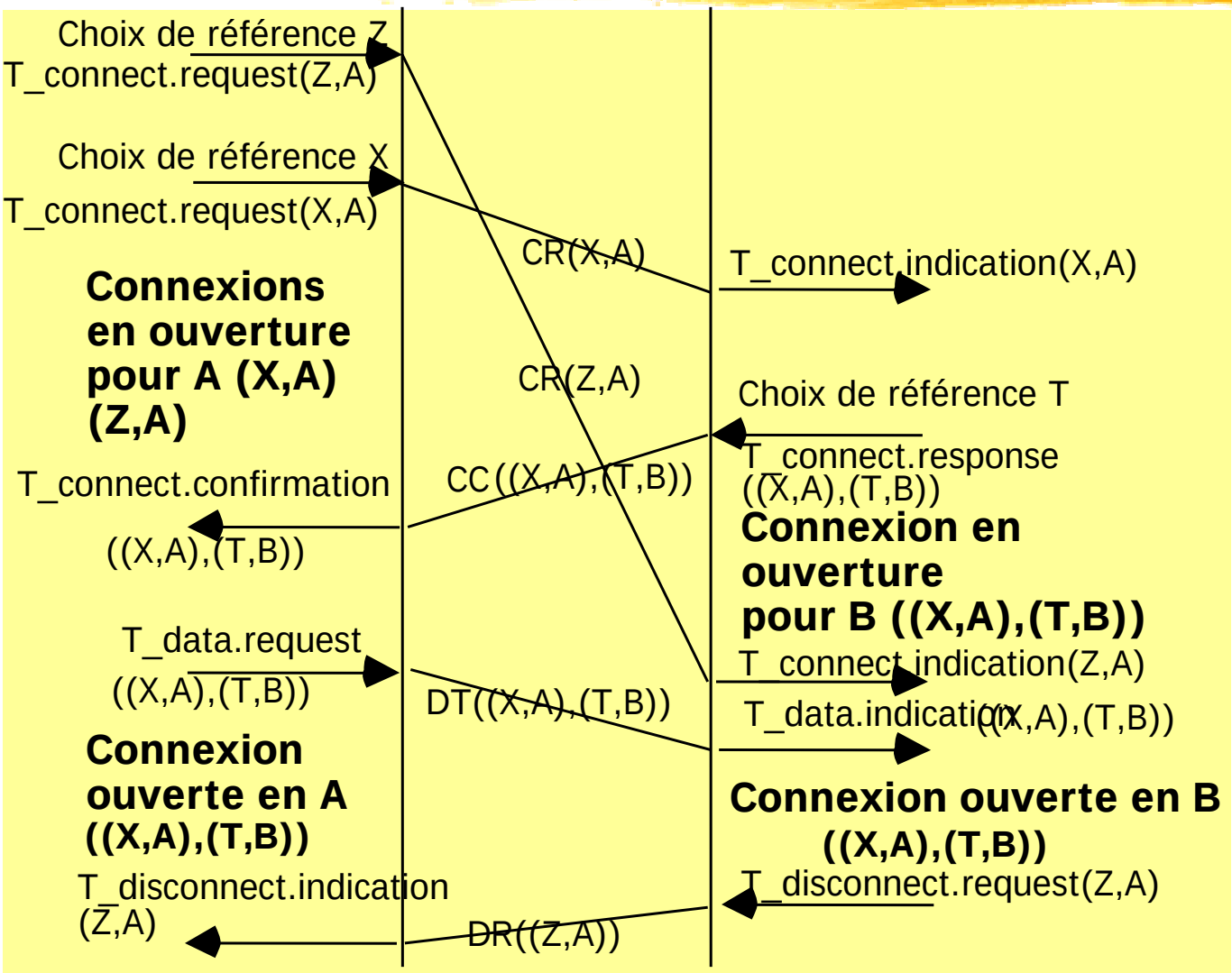
- **Problème posé**: perte d'unités de données de protocole en phase d'ouverture (ouverture de connexion incomplète à cause de messages perdus, anciens ou déséquencés).
- **L'objectif visé**: Avant de commencer à transférer effectivement des données on doit atteindre un état tel que:
  - le demandeur de l'ouverture sait que le destinataire accepte l'ouverture,
  - le destinataire sait que le demandeur sait qu'il accepte l'ouverture de connexion.
- **Solution 1**
  - Ignorer les TPDU ("Transport Protocol Data Unit") de données qui arrivent avant la confirmation d'ouverture connexion => des pertes de données, alors que le réseau est parfaitement fiable.
  - Ceci ne définit pas comment on atteint l'ouverture confirmée (la connexion peut rester indéfiniment en ouverture).
- **Solution 2**
  - Stocker les TPDU de données qui arrivent avant la confirmation de connexion, afin de les présenter lorsque la connexion est établie.
  - On peut avoir à stocker de gros volumes de données.
  - Il n'y a toujours pas de définition de la confirmation d'ouverture.

L'ouverture confirmée en trois étapes



- **Principes**
- **On interdit** à l'entité appelée d'envoyer des données (de considérer que la connexion est ouverte)
- **L'appelé ne peut répondre** qu'une TPDU de confirmation.
- **Jusqu'à ce qu'il ait reçu** une TPDU qui lui confirme la bonne réception de sa TPDU de confirmation de connexion par l'appelant.
- **On utilise pour cela une troisième TPDU**
- Selon le protocole une donnée si l'appelant a des données à envoyer ou une TPDU spécifique (troisième message d'ouverture).

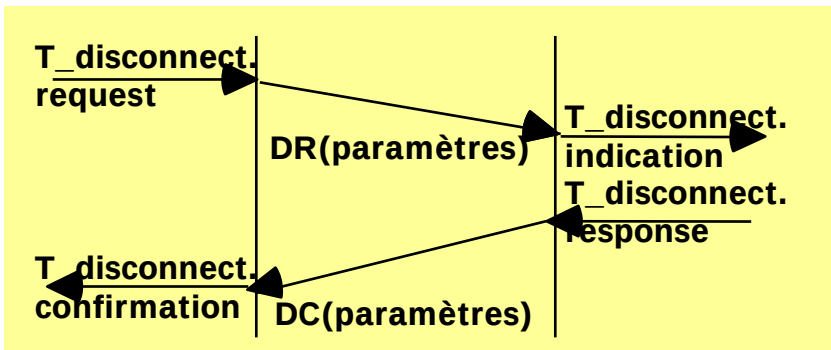
Exemple d'échange confirmé en trois étapes avec références



- **Remarques:**
- Il ne peut y avoir d'ambiguïté entre les connexions référencées Z et X dont l'ouverture concurrente est en cours.
- Le site B a le choix d'accepter ou de rejeter (ici par un disconnect) l'une des deux connexions.

B) Problèmes de fermeture de connexion

- **Fermeture de connexion = atteindre un consensus** sur la connaissance de la libération.
  - Si la connexion doit être libérée A et B doivent atteindre (assez rapidement) un état ou ils décident **tous les deux que la connexion est fermée**:
  - Pour éviter **l'existence indéfinie de connexions** demi-ouvertes
    - L'un a décidé de fermer et l'autre ne le sait pas.
  - Pour terminer les échanges dans une situation claire du point de vue applicatif.
- **Solution de base: accord confirmé**
  - T_disconnect.request, DR
  - T_disconnect.indication
  - T_disconnect.response, DC
  - T_disconnect.confirmation.



- **Ce protocole ne marche pas en présence de tous types de délais de transmission et de pertes de messages.**

Problèmes de consensus : le problème des deux armées

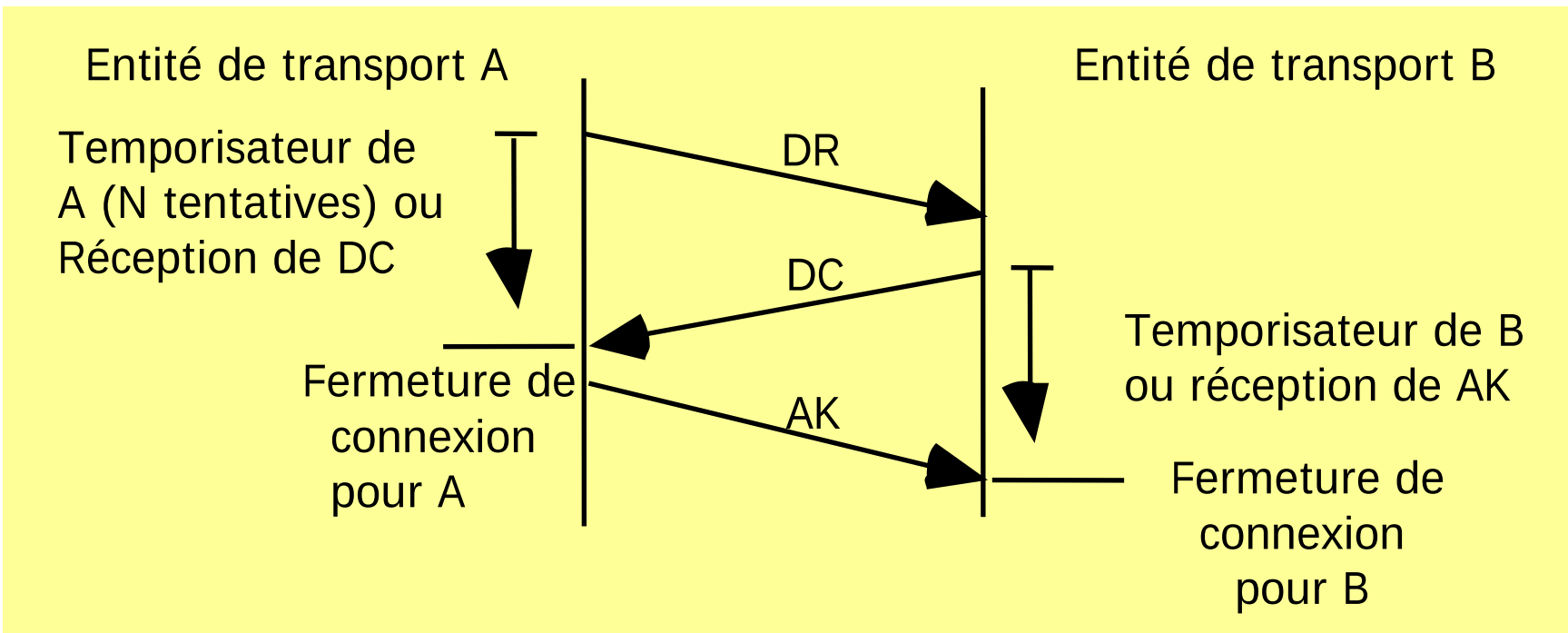
- **La formulation d'un problème de consensus entre deux entités communicantes** analogue au problème de déconnexion:
  - Une armée A campe dans une vallée.
  - L'armée B (son adversaire) est séparée en deux corps d'armées B1 et B2 qui occupent deux collines distinctes séparées par l'armée A.
  - A est plus forte que chacun des deux corps d'armées B, mais A est moins forte que B dans le cas ou les deux corps B1 et B2 attaquent de façon coordonnée.
  - Le seul moyen pour coordonner l'attaque des B est l'usage de messagers qui traversent la vallée et qui sont susceptibles de mettre très longtemps, d'être capturés ... en raison des ennemis
- **Solution asynchrone au problème des deux armées**
  - **Existe-t-il un protocole déterministe :** qui marche dans tous les cas.
  - **En mode message asynchrone :** quels que soient les pertes ou les retards (délais de transmission non bornés).
  - **Permettant aux commandants B de gagner avec certitude :** de se mettre d'accord sur l'heure d'attaque ? (de résoudre le problème du consensus à deux).

Impossibilité d'une solution déterministe en mode asynchrone

- **Présentation informelle du mode de raisonnement.**
- **Recherche d'une solution à deux messagers.**
  - Le commandant de B1 envoie au commandant de B2 le message "Attaque demain« .
  - Le commandant de B2 reçoit le message et fait répondre "D'accord".
  - Ce dialogue n'est pas concluant, car le commandant de B2 ne peut pas savoir avec certitude si sa réponse a bien été reçue (non perdue ou retardée après la date proposée).
- **Recherche d'une solution à trois messagers.**
  - On utilise à un protocole en trois étapes.
  - B1 doit accuser réception de la réponse de B2 à sa proposition.
  - Alors, c'est le commandant de B1 qui ne peut pas savoir si sa confirmation est bien arrivée.
  - S'il y a perte de la confirmation, B2 ne bougera pas, B1 attaquera seul et sera vaincu.
- **Recherche d'une solution à N messagers.**
  - Avec N messagers, le même problème se pose : l'émetteur du Nième message n'est jamais sur de savoir si son message a été reçu ou non.
  - Pour toute solution à N messagers comme le dernier message peut ne pas arriver il ne peut donc être essentiel pour la solution et on doit pouvoir s'en passer.
  - Donc on peut enlever les derniers messagers les uns après les autres.
  - Comme il n'existe pas de solutions à deux messagers il n'existe pas de solution (raisonnement par récurrence).

Solution probabiliste et synchrone pour la déconnexion de transport

- **Solution probabiliste :** une solution qui ne marche pas en toutes circonstances de pannes et de délais de transmission.
- **Solution synchrone :** basée sur des délais de transmission bornés permettant de régler des délais de garde.
- **Schéma général** d'une solution:

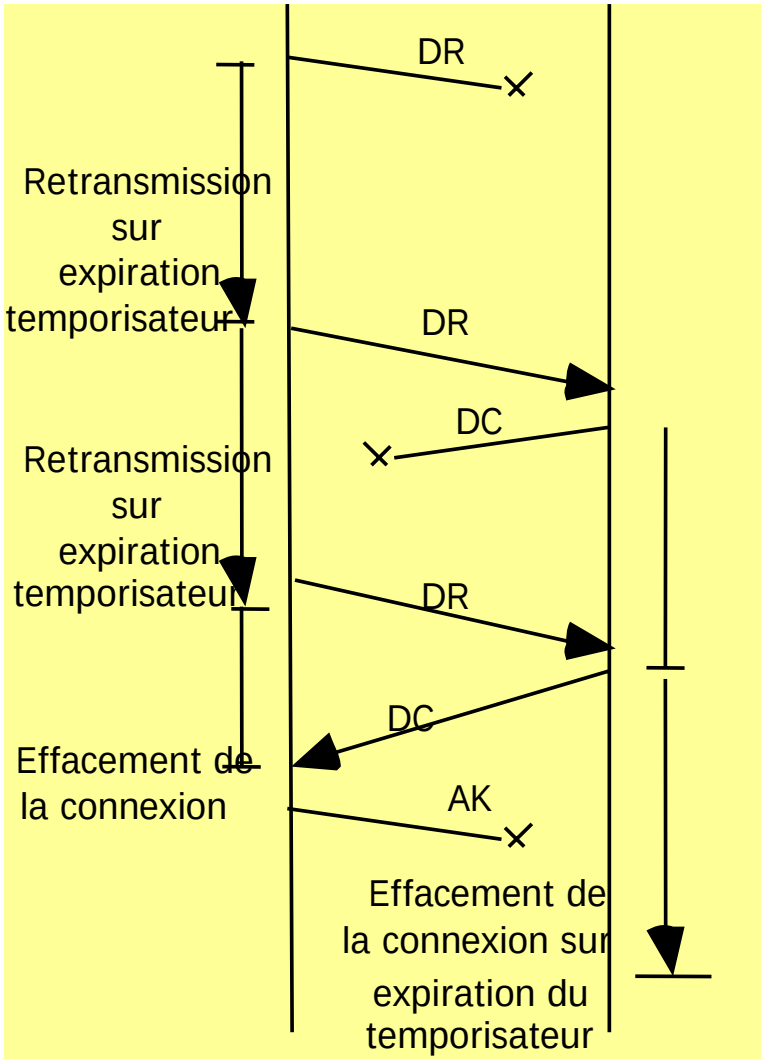


Détails du fonctionnement de la déconnexion

- **Action de A**
  - L'entité de transport A désirant mettre fin à une connexion envoie une TPDU de demande de **déconnexion DR** et arme un temporisateur.
  - A échéance, si elle n'a pas reçu de réponse, elle **retransmet la TPDU.**
  - Au bout de **N retransmissions** sans réponse ou sur réception d'une confirmation DC, l'entité **A ferme la connexion.**
- **Action de B**
  - L'entité de transport B recevant la demande de déconnexion **répond par une TPDU de confirmation** de déconnexion et arme un temporisateur.
  - A **échéance** du temporisateur ou sur réception d'un **acquittement** de sa confirmation, elle **ferme** effectivement la connexion (elle enlève de sa table des connexions ouvertes les informations concernant la connexion considérée).
  - L'entité de transport A initiatrice de la libération **accuse réception de la confirmation** en envoyant une TPDU d'acquittement et en fermant la connexion de son côté.

Analyse du protocole de déconnexion

- **Cas pour lesquels le protocole fonctionne**
  - Basés sur la réussite de certaines communications et sur l'existence de délais de garde.
  - Permettant de régler des temporisateurs en fonction des délais de propagation des messages sur le réseau.
- **Cas pour lequel le protocole ne fonctionne pas**
  - Il existe une probabilité non nulle pour que les demandes de déconnexion ou leurs réponses se perdent toutes.
  - A ferme la connexion sans que B le sache ou B ferme la connexion sans que A le sache => Notion de connexion demi-fermée.



Analyse du protocole de déconnexion

- **Solution synchrone (basée sur le temps) et probabilistes aux connexions demi-fermées**
  - On utilise un **délai de détection d'inactivité** de la connexion : si aucune TPDU n'a été reçue pendant ce délai, une entité est autorisée à fermer la connexion de son côté **(solution synchrone).**
    - Exemples : A sur émission de DR ou B sur émission de DC (temporisateurs)
  - Cette solution amène à fermer des connexions qui ne devraient pas l'être uniquement **parce que les temps de réponse sont trop élevés** par rapport aux délais de garde **(solution probabiliste).**
  - Pour que ça fonctionne il faut que les messages aient un temps de transmission borné (**transmission à délai borné ou "synchrone"**) qui permet de régler le délai de garde d'inactivité et qu'on accepte une probabilité (qui doit être très faible) de fermer des connexions.
- **Conclusion**
  - **Dans le monde des réseaux en présence de pannes (pertes de messages) et délais de transmission, on ne construit de solutions industrielles que probabilistes et synchrones.**
  - **La probabilité d'échec doit rester acceptable.**

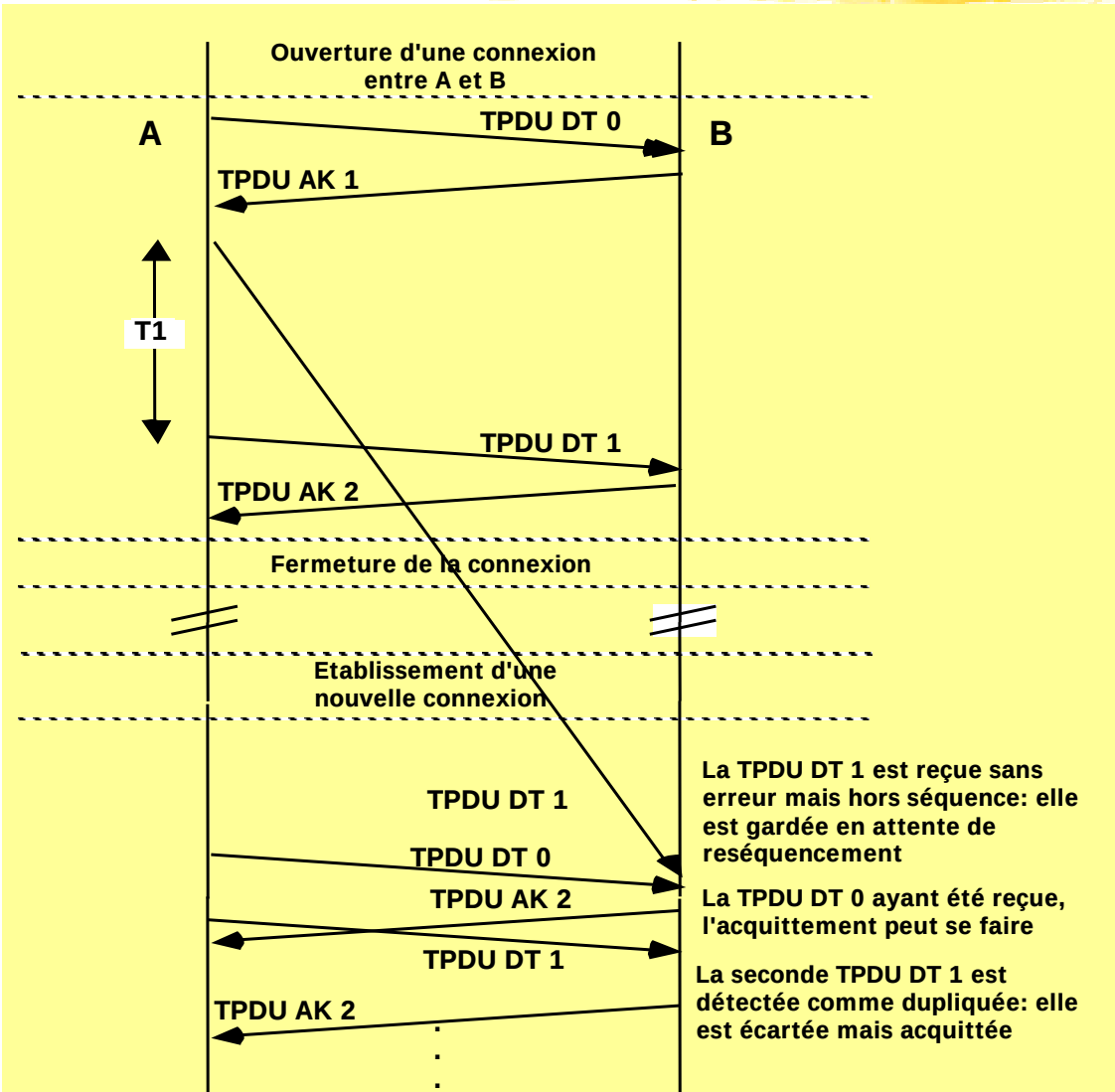
Problèmes et solutions au niveau transport

Problèmes de transmission des données au niveau transport

Contrôle d'erreur au niveau transport

- **Code détecteur d'erreur**
  - On utilise plutôt une somme de contrôle (de type parité) rapide à calculer par programme (par rapport à un code polynomial).
- **Numéros de séquence**
  - La livraison en séquence assuré par une numérotation des TPDU.
- **Acquittements positifs et temporisateurs**
  - Le contrôle d'erreur est assuré comme au niveau liaison par des protocoles à acquittement positif et retransmission sur délai de garde.
- **Solutions identiques à celles des protocoles de liaison.**
- **Difficulté du contrôle d'erreur de transport**
  - Problèmes posés par les vieux paquets et les dé-séquencements.

Exemple 1 : Interférence entre paquets sur des connexions successives



Solutions possibles

- Références de connexion dans les données.

=> Identification des paquets.

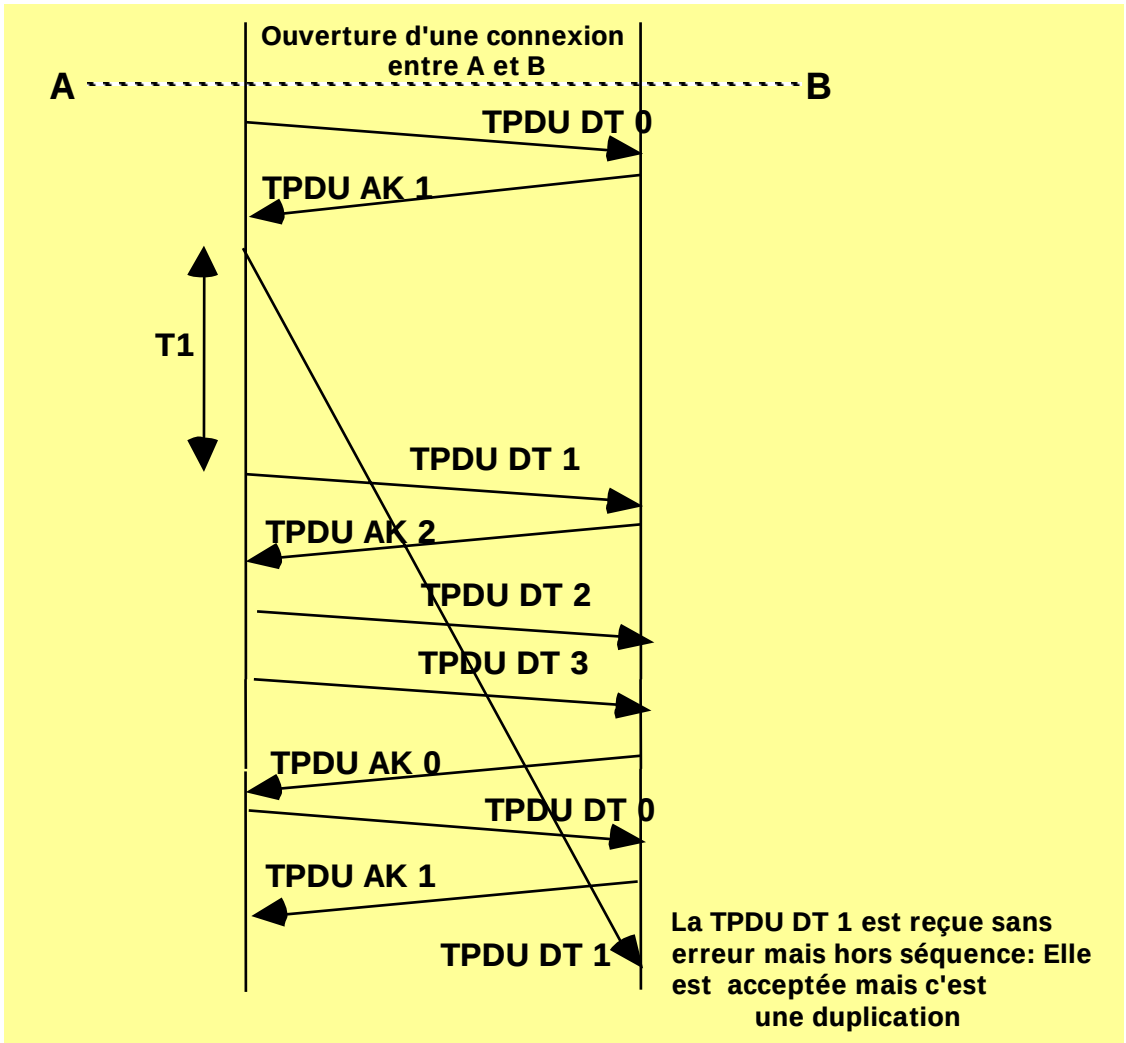
- Utilisation du gel des références.

=> Pas de réception dans une connexion ultérieure.

- Changement de l'espace des numéros de séquence utilisés.

=> Acceptation dans une nouvelle fenêtre de réception.

Exemple 2 : Interférence entre paquets de la même connexion



Solution

- Utilisation d'un numéro de séquence de grande taille (par exemple sur 32 bits ou mieux sur 64 bits).
- La probabilité de réutilisation à court terme d'un même numéro pendant une connexion doit être très faible.

Contrôle de flux au niveau transport

- **Problème de contrôle de flux.**
  - Adaptation de la vitesse du processus émetteur à celle du récepteur.
  - En l'absence de contrôle de flux, des unités de données sont détruites à leur arrivée, faute de tampons libres.
- **Problèmes posés par le contrôle de flux de transport.**
  - Le réseau sous-jacent achemine plus ou moins vite les informations selon sa charge.
  - Le site récepteur prend en compte les messages au niveau transport plus ou moins rapidement selon sa charge.
  - Les traitements d'application prennent plus ou moins de temps selon les données à donner.
  - **=> Très grande variabilité des vitesses de réception.**
  - **=> Nécessité d'un mécanisme de régulation très adaptatif**
- **Remplacement des fenêtres de taille fixe par des fenêtres de taille variable.**

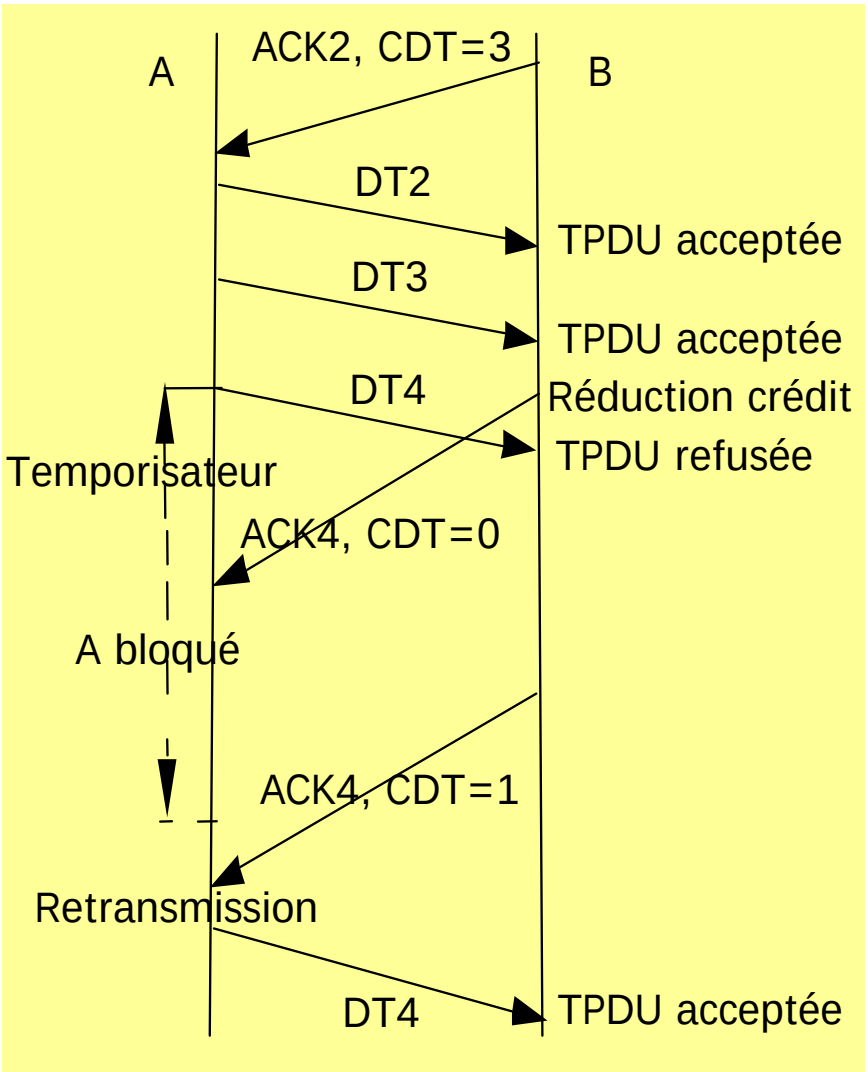
Contrôle de flux par crédit variable

- **Notion de crédit CDT** :
  - **Une zone supplémentaire** dans les données émises par le récepteur et qui sont porteuses d'un acquittement positif AK ('piggybacking').
  - **C'est le nombre d'unités de données que l'émetteur peut émettre en anticipation** (que le destinataire est prêt à recevoir) à partir du numéro d'acquittement porté dans le segment.
  - Le récepteur **accepte** les messages de **numéro AK** au **numéro AK+CDT-1**.
- **Définit un contrôle de flux par fenêtre glissante de taille variable** car le récepteur peut modifier le crédit en fonction de ses capacités.
  - Le récepteur peut accroître la taille de la fenêtre dynamiquement
  - Le récepteur peut réduire la taille de la fenêtre dynamiquement.
- **Remarque**
  - **Le crédit est donné par la définition en absolu** des numéros de séquence autorisés : autorisation d'émettre dans la fenêtre de AK à AK + CDT - 1.
  - **La définition relative par autorisation d'envoi de CDT messages en plus** est impossible puisque les crédits doivent pouvoir être répétés en cas de perte.

    => A chaque répétition **on accorderait l'envoi de CDT messages en plus.**

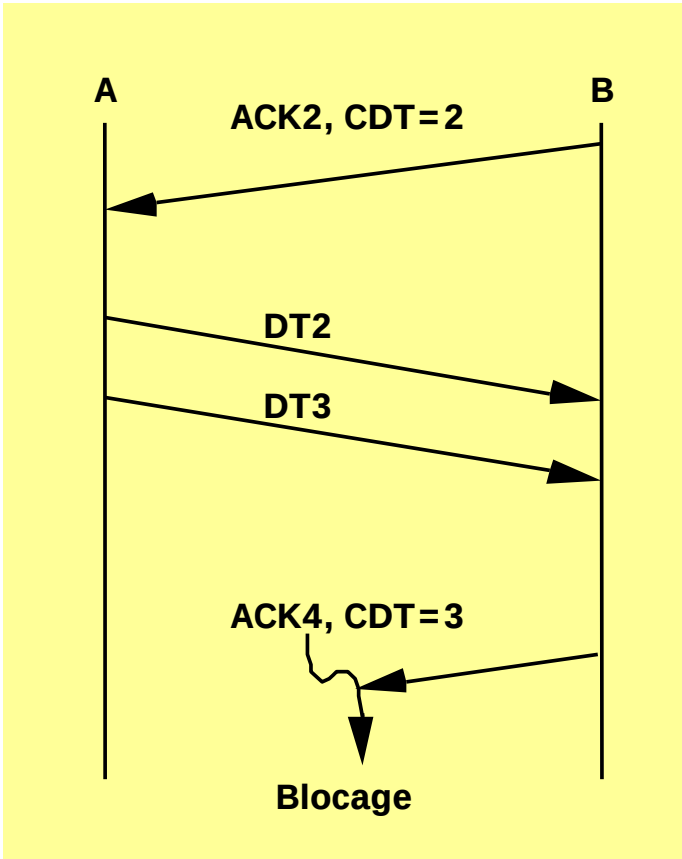
Problèmes de gestion des crédits : 1) Réduction de crédit

- **Problème** : la réduction de crédits peut entraîner la perte de TPDU de données.
- **Exemple:**
  - **Perte de TPDU données** à la suite d'une réduction de la limite supérieure de fenêtre.
  - **Bien que le réseau soit fiable.**
  - **La TDPU DT 4 arrive hors fenêtre** puisque le destinataire B a réduit entre temps le crédit d'émission de l'expéditeur A.
- **Solution:**
  - Retransmission comme si la donnée avait été bruitée.
  - Utilisation d'acquittements positifs et de temporisateur.



Problèmes de gestion des crédits : 2) Interblocage de perte de crédit

- **Problème** : La perte d'un message de crédit peut conduire à un interblocage de message.
- **Exemple :**
  - Interblocage du à une perte de message porteur de crédit.
- **Solution:**
  - Répétition des messages porteurs de crédits.



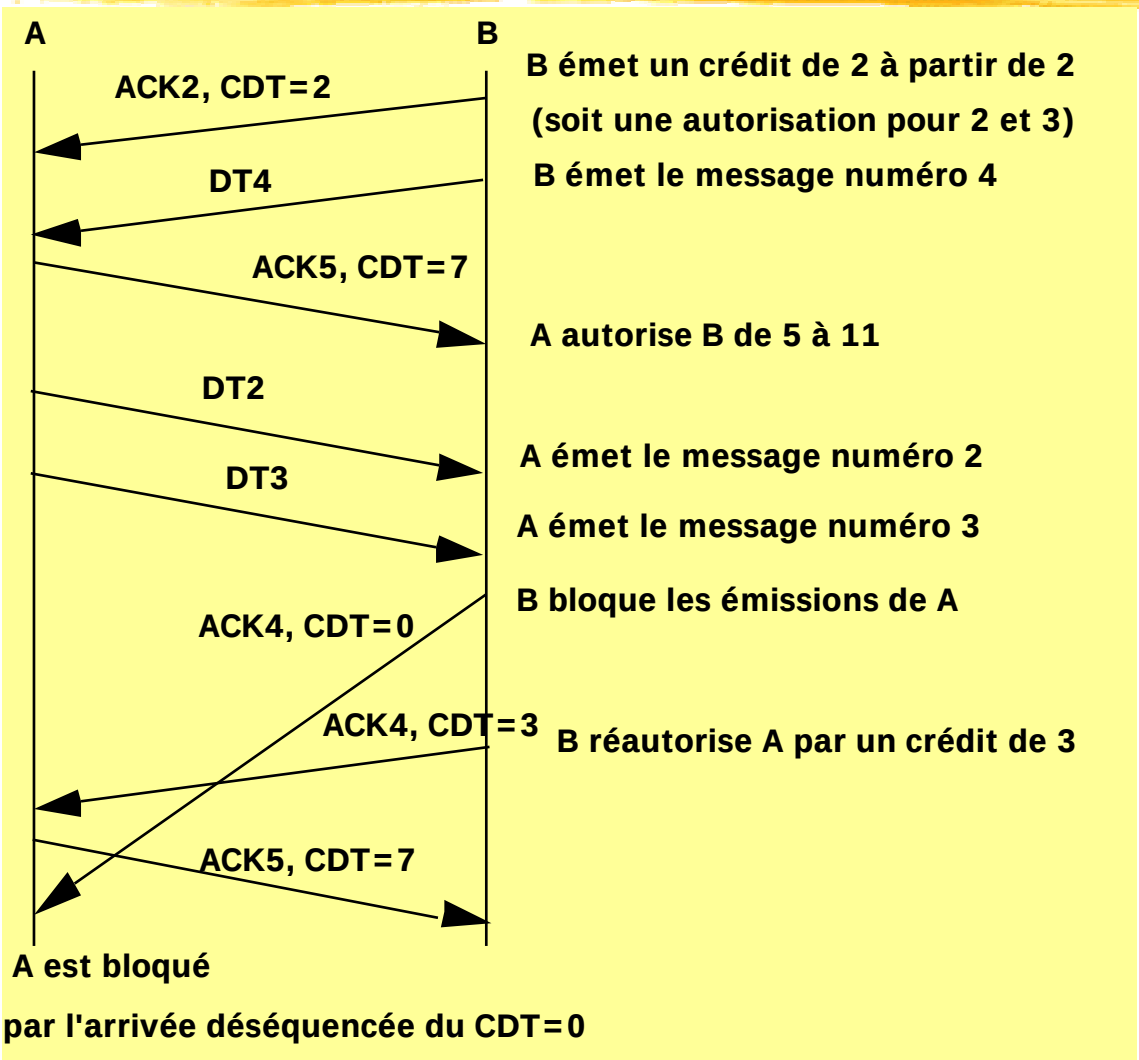
Problèmes de gestion des crédits
Solutions à la perte de crédit

- **1- Le récepteur répète périodiquement** ses messages d'augmentation de crédit (sur temporisateur).
  - Trafic périodique d'acquittements avec crédit (ou "Idle Data Transfer").
  - => Tôt ou tard le site distant reçoit une copie du dernier crédit.

- **2- Le récepteur arme un délai de garde TW** après chaque message d'acquittement/crédit et répète le message à échéance uniquement si l'émetteur n'a pas repris ses émissions entre temps (il considère qu'il est perdu).
  - **Si TW est trop faible:** on risque de réexpédier des acquittements/crédits à un rythme élevé pendant un temps important, ce qui consomme inutilement les ressources du réseau.
  - **Si TW est très élevé:** on rallonge la période de blocage de l'expéditeur quand ce dernier à des TPDU à émettre.

3) Interblocage de déséquencement de crédits

- **Problème** : Le déséquencement des crédits peut conduire à des interblocages.
- **Exemple :**
  - Un crédit ayant dépassé l'autre l'émetteur est bloqué.
- **Une solution:**
  - Utiliser des numéros de séquence pour les augmentations de crédits.
  - Délivrer les augmentations dans l'ordre d'émission.



Problèmes et solutions au niveau transport

Conclusion

Conclusion: Conception des protocoles de transport

- **Problèmes de transport** : finalement assez compliqués.
- **On dispose de solutions correctes** en mode point à point pour les données informatiques classiques.
- **Différents problèmes restent posés** :
  - **Adaptation des solutions existantes** à la montée en débits (gigabits, térabits, ...)
  - **Communication de niveau transport en diffusion** (multipoint).
  - **Communications de transport avec qualité de service temporelle** (multimédia).

Niveau Transport
"Transport Layer"

Chapitre II

Exemple des protocoles et services de transport INTERNET

UDP "User Datagram Protocol"
TCP "Transmission Control Protocol"
Un service pour TCP et UDP: les sockets

Exemple des protocoles et services de transport INTERNET

UDP

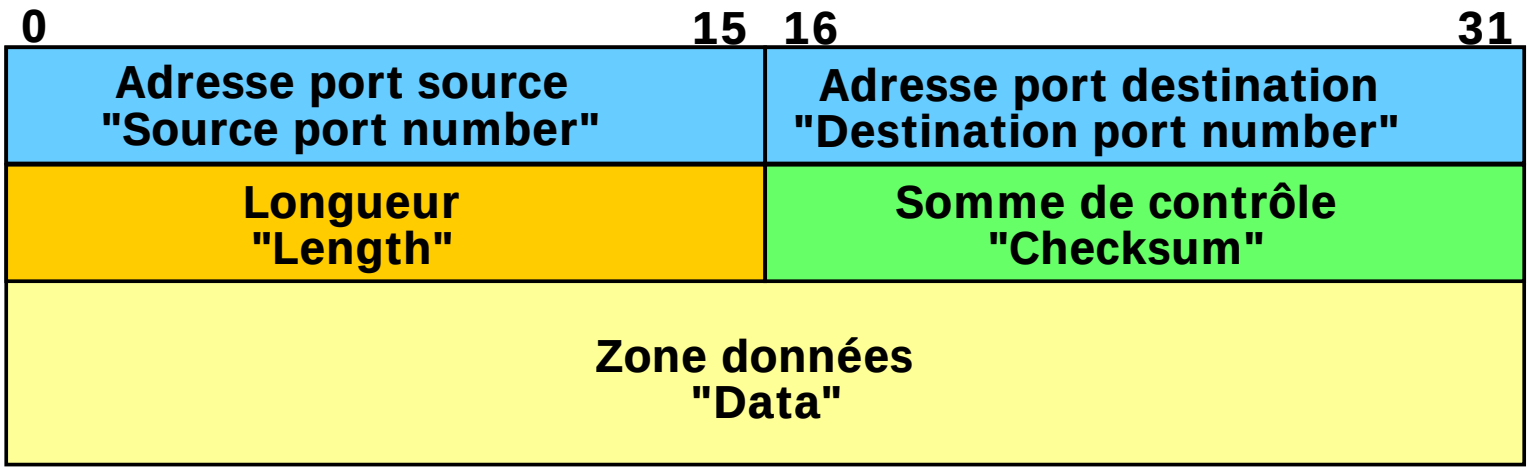
"User Datagram Protocol"

Choix de conception UDP

- **UDP permet l'émission de datagrammes IP en utilisant un protocole très simplifié de niveau transport.**
- **Sans connexion**
  - Pour des relations courtes.
  - **Efficace** (en termes de performances).
  - Simplification des développements de codes applicatifs réseaux.
- **Avec adressage de niveau transport**
  - Identification des **"ports"**

- **UDP implante donc très peu de mécanismes**
  - Adresses de ports (points d'accès de service de transport)
  - Somme de contrôle d'entête (optionnelle)
  - Aucun contrôle de séquence, contrôle d'erreur, ni contrôle de flux.

Format du segment UDP



- Structure très simplifiée (génération et analyse rapide)
- Peu encombrante (8 octets)
- Dédiée essentiellement au problème d'adressage de niveau transport.

Détails concernant les différents champs

- **Zone source port et destination port**
  - Numéros de port identifiant l'utilisateur source et l'utilisateur destinataire (16 bits).
  - Le numéro de port source est optionnel (si non renseigné il est mis à zéro).
- **Zone longueur**
  - Longueur totale en octets du message.
  - Redondant avec la longueur de paquet IP.
  - => La longueur est optionnelle (dans ce cas on la met à zéro).
- **Zone somme de contrôle**
  - Le champ "checksum" couvre la partie entête et la partie données.
  - Il est calculé comme pour l'entête IP par un ou exclusif sur des groupes de 16 bits et complémenté à un.
  - Si la longueur des données est impaire un remplissage par des zéros est effectué sur le dernier octet.
  - => La somme de contrôle est optionnelle (dans ce cas on la met à zéro).

Exemple des protocoles et services de transport INTERNET

TCP

"Transmission Control Protocol"

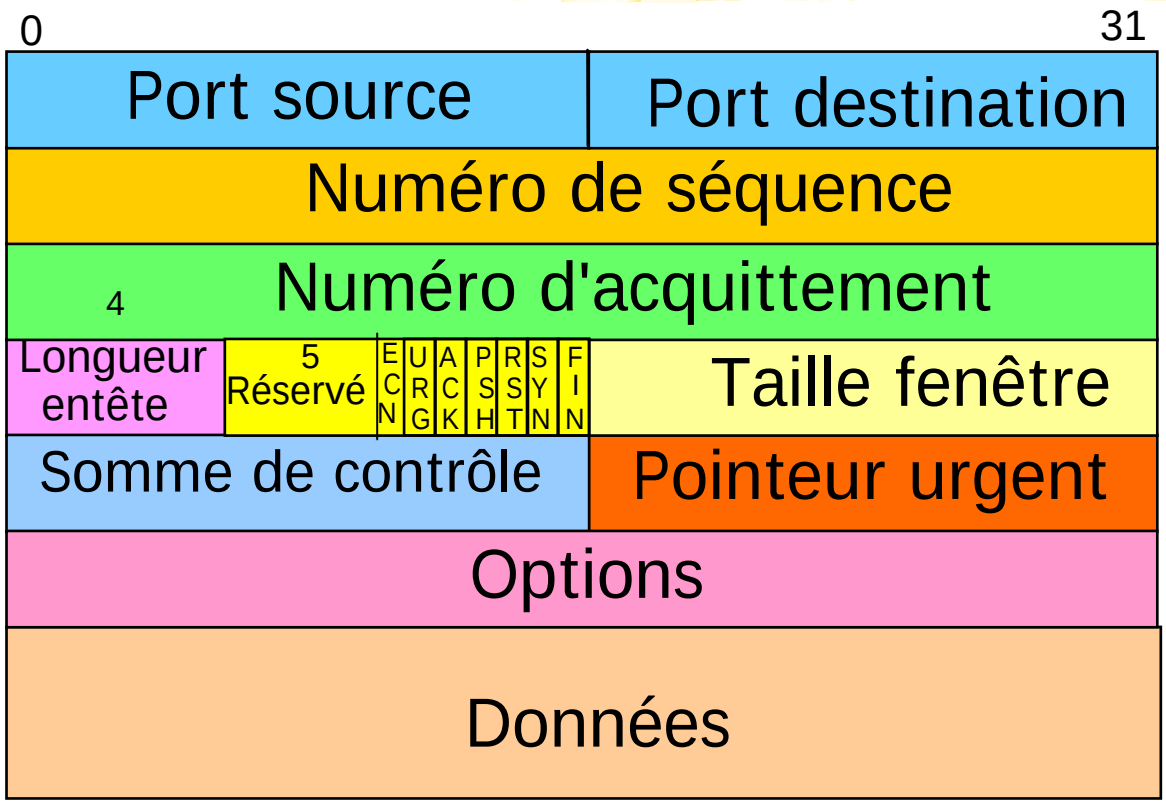
Choix de conception TCP

- **TCP a été conçu en fonction de IP qui est de qualité de service médiocre.**
  - Le réseau sous-jacent (IP) peut **perdre, altérer, dupliquer**, **déséquencer** les paquets.

- **TCP permet un transport de données orienté octets**
  - En mode **connecté, bidirectionnel.**
  - Avec **contrôle d'erreur** (transport fiable).
  - Avec **contrôle de séquence.**
  - Avec **contrôle de flux.**
  - Avec **contrôle de congestion**

- **Existence de trois phases principales**
  - **Ouverture** de connexion
  - **Transfert** de données
  - **Fermeture** de connexion

Format du segment TCP



- Un seul format de TPDU pour tous les mécanismes du protocole.
- Un en-tête relativement long : au minimum 20 octets

Détails concernant les différents champs (1)

- **Numéro de port source ("Source port")**
  - Entier sur 16 bits: identifie le port émetteur.
- **Numéro de port destination ("Destination port")**
  - Entier sur 16 bits: identifie le port récepteur.
- **Numéro de séquence ("Sequence Number") sur 32 bits**
  - Si le bit SYN est non positionné c'est le numéro de séquence du premier octet de données dans la TPDU
  - Si le bit SYN est positionné c'est le numéro de séquence initial (ISN): le premier octet de données a pour numéro ISN+1.
- **Numéro d'acquittement sur 32 bits.**
  - Numéro de séquence de l'octet que l'entité s'attend à recevoir.
- **Longueur de l'en-tête sur 4 bits**
  - En nombre de mots de 32 bits.
  - Nécessaire puisque l'en-tête comporte des options de longueur variable.
- **Zone réservée de 5 bits.**

Détails concernant les différents champs (2)

- **Sept drapeaux (1 bit) déterminent la présence de certains champs** dans le segment TCP et en fait donnent un type (ou plusieurs types) au segment: donnée, acquit positif, établissement ou libération de connexion...

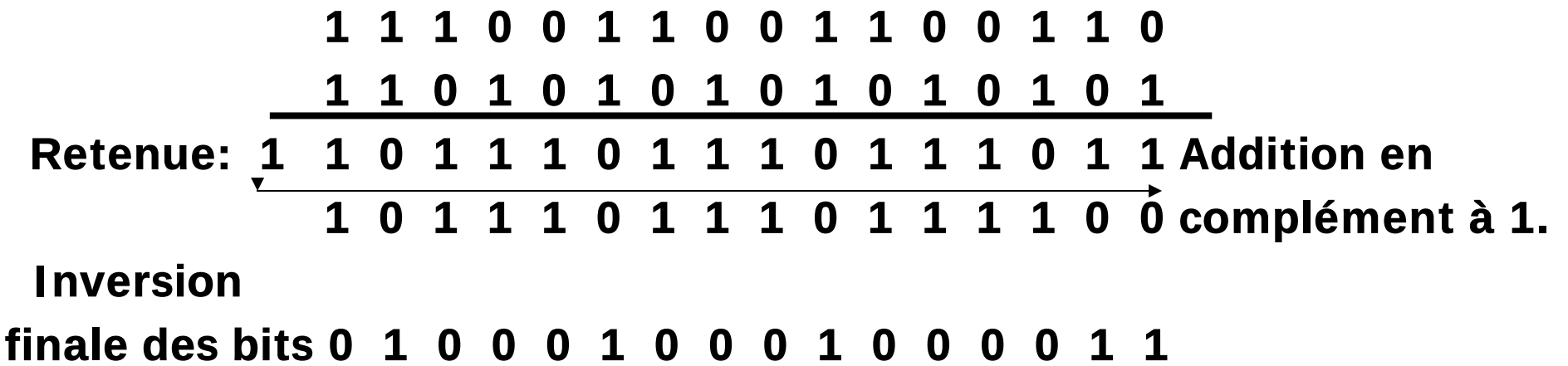
- **URG** ("Urgent") : à 1 si la zone donnée urgente est présente (le pointeur urgent sur la fin de la zone de donnée urgente doit être positionné).
- **ACK** ("Acknowledgment") : à 1 si le champ "numéro d'acquittement" est présent. Il sert aussi à l'acquittement du numéro de séquence initial.
- **PSH** (Push) : une option de service qui demande la transmission immédiate des données en instance dans un segment porteur du bit push. A l'arrivée le destinataire doit délivrer également immédiatement.
- **RST** ("Reset") : l'émetteur ferme abruptement la connexion.
- **SYN** ("Synchronize") : synchronise les numéros de séquence (utilisation principale à l'établissement des connexions).
- **FIN** ("Fin") : l'émetteur n'a plus rien à transmettre et libère la connexion.
- **ECN** ("Explicit Congestion Notification") : notification de congestion.

Détails concernant les différents champs (3)

- **Taille Fenêtre** ('Window Size') (16 bits)
  - Le crédit (taille de la fenêtre de réception) exprimé en nombre d'octets.
  - Mesuré à partir du numéro d'acquittement inséré dans le segment.
- **Somme de contrôle** ('Checksum') (16 bits)
  - Sert à la détection d'erreurs.
- **Pointeur urgent** ('Urgent pointer') (16 bits)
  - Le pointeur urgent pointe sur la fin des données urgentes placées nécessairement à partir du début du segment.
  - Les données urgentes peuvent être suivies par des données normales.
- **Zone options**
  - **Permet l'échange d'informations protocolaires** en extension de l'entête (taille maximum de la zone option 44 octets).
  - **Un code sur un octet distingue les différentes options** (de l'ordre de 25/30 options définies par des RFC).

Approfondissements 1) : calcul de la somme de contrôle UDP/TCP

- **Checksum :** calcul sur le segment décomposé en groupes de 16 bits : calcul **simple, faisable en logiciel** comme en **IP.**



- **Calcul portant sur le segment et un 'pseudo entête IP'.**
  - Les adresses IP source et destination, la longueur, le type.
  - Pour sécuriser la donnée utilisateur mais aussi les principales informations protocolaires (TCP et une partie de l'entête IP).
  - Violation de l'indépendance entre couches.

Approfondissements 2) : Différences Push et Urg

- **Mode push** dans une primitive de service
  - **Force l'émission immédiate** du segment porteur des données en mode push (le segment comporte le bit push).
  - **Les données circulent normalement.**
  - **A destination:** le segment doit être délivré immédiatement.
  - **Exemple:**
    - Terminal en écho local on peut mettre le bit push sur un segment lorsqu'on a atteint le retour chariot.
    - Terminal en écho distant : on doit faire push sur chaque caractère.
- **Mode données urgentes.**
  - **Un bloc urgent** est défini à l'intérieur du segment (utilisation du bit URG et du champ pointeur urgent).
  - **Les données devraient circuler le plus rapidement possible** : mécanismes de QOS ou de priorité IP.
  - Le destinataire devrait recevoir les données dès l'arrivée.

Approfondissements 3) : Exemples de zones options (1)

- **Option ‘End of option list’** (code=0) : fin de la liste des options TCP.
- **Option ‘No operation’** (code=1) : pour remplissage et alignement sur des frontières de mots de 32 bits.
- **Option MSS ('Maximum Segment Size')** sur 4 octets: Taille maximum des segments (code=2).
  - Annonce de la taille max en octets que le TCP est capable de traiter (dans un segment d’ouverture avec bit syn=1).
  - Absence de MSS : TCP doit toujours accepter 512 octets de données soit un MSS par défaut de 512 + 20 + 4 = 536.
- **Option ‘Window Scale’** 10 octets : facteur d’échelle fenêtre (code = 3)
  - Dans les réseaux haut débit la valeur du crédit (taille de la fenêtre sur 16 bits 64K) est trop petite.
  - Sur un octet cette option définit à l’ouverture de connexion un facteur multiplicatif à appliquer à la zone fenêtre.
  - La valeur utilisée est 2^n ou n est le facteur d’échelle.
  - Soit en fait un décalage de n bits sur la valeur de la taille de fenêtre.

Exemples de zones options (2)

- **Option ‘TCP SACK permitted’** 2 octets : (code = 4)
  - TCP peut fonctionner en mode rejet sélectif.
  - Dans le cas ou des accusés de réception sélectifs sont autorisés. Cette option met en place le mécanisme.
- **Option ‘Timestamp’** 10 octets : datation (code = 8)
  - **Problème** de la mesure du temps d’aller retour (RTT Round Trip Time): les acquittements groupés sont retardés.
  - **Fonction principale de l’option**: faire écho d’une date sur 32 bits pour qu’un émetteur mesure correctement le RTT.
    - Deux estampilles : **TSVAL** la date de l’émission du présent segment ,
    - **TSECR** (‘echo reply’) la date qui était portée par le segment de donnée dont ce segment est un acquittement positif (bit ACK=1).

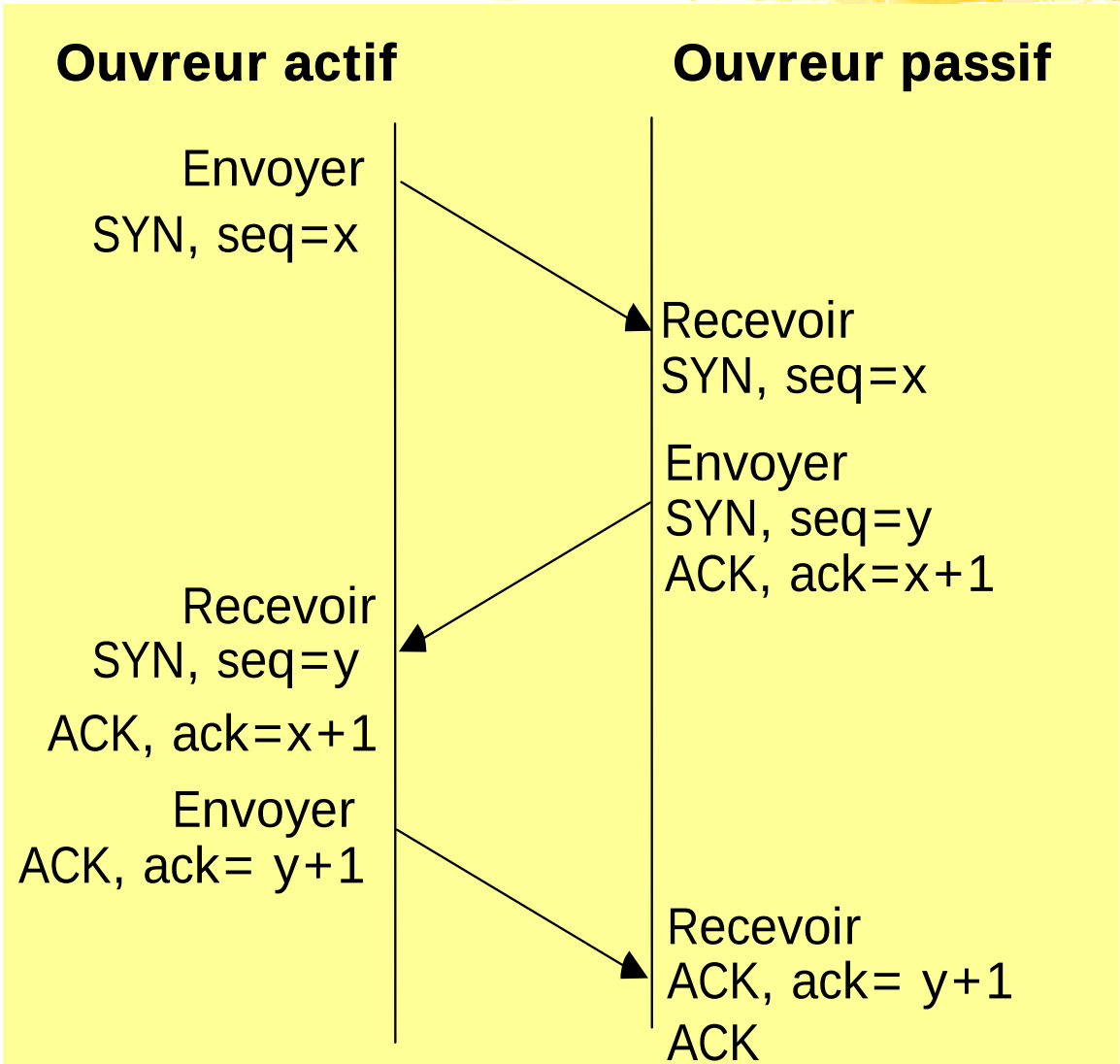
Protocole d'ouverture de connexion en TCP : Principes généraux (1)

- **Ouverture de connexion en trois messages** : "three way handshake" (pour des raisons de fiabilité).
- **Chaque extrémité doit choisir un numéro de séquence** initial ISN.
  - **Ce numéro sur 32 bits est envoyé à l'autre extrémité.**
  - **Chaque numéro de séquence** initial choisi par l'un des communicants doit être **acquitté par l'autre**.
  - **Le couple des numéros de séquence sert de référence initiale** à la connexion.
  - **La norme recommande de choisir les numéros de séquence "au hasard" selon les bits de faible poids d'une horloge** et de ne pas réutiliser ces numéros avant un délai (gel de référence, purge des "vieux paquets" retardés dans le réseau et qui pourraient être mal interprétés).

Protocole d'ouverture de connexion en TCP : Principes généraux (2)

- **Une seule connexion TCP** est établie entre deux ports.
  - TCP ignore les **requêtes** de connexion **ultérieures.**
- **Ouverture de connexion normale** : on distingue le demandeur initial de la connexion (**ouvreur actif**) et l'accepteur (**ouvreur passif**).
- **Cas possible d'ouverture simultanée** : on a deux initiateurs simultanément actifs et l'on doit traiter ce cas particulier.

Diagramme de messages d'ouverture de connexion TCP



Message 1

- Demande de connexion avec SYN=1, un numéro de séquence initial X, et ACK=0 indiquant que le champ d'acquittement n'est pas utilisé.

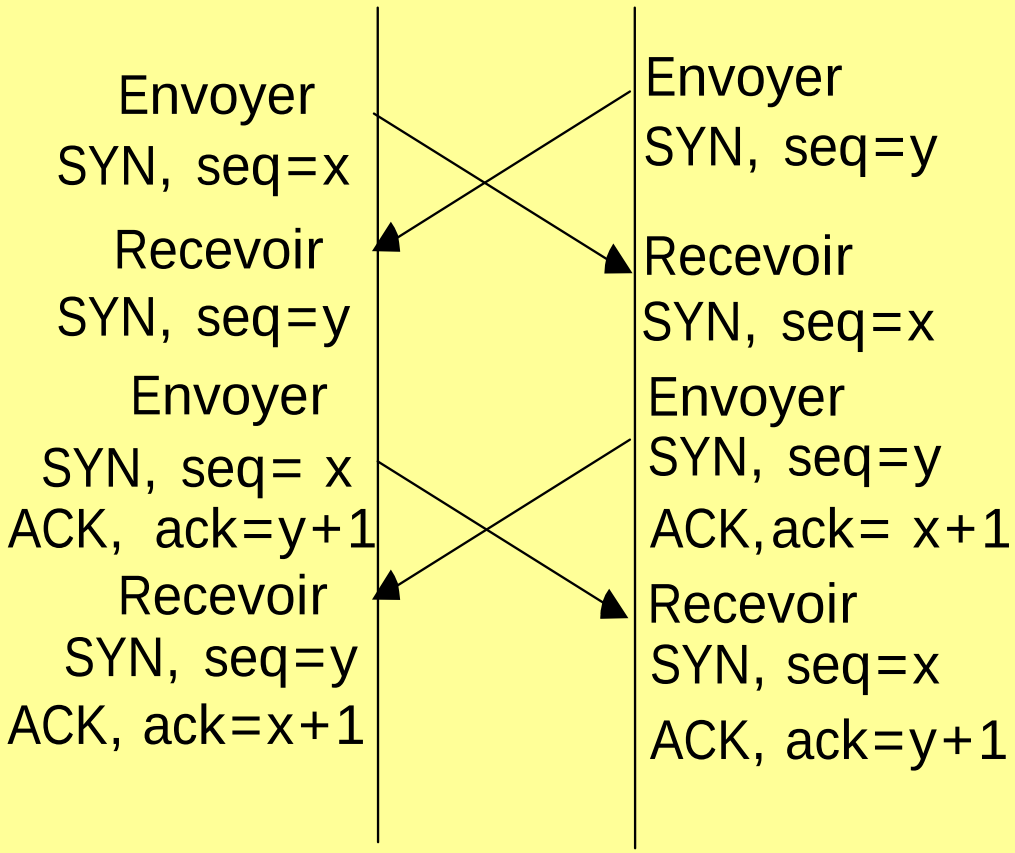
Message 2

- Confirmation de connexion avec SYN=1, un numéro de séquence initial Y et ACK=1 avec le numéro X+1 acquittant le numéro proposé X.

Message 3

- Acquittement du numéro Y proposé avec ACK=1 et un numéro d'acquittement Y+1.

Ouverture de connexion simultanées

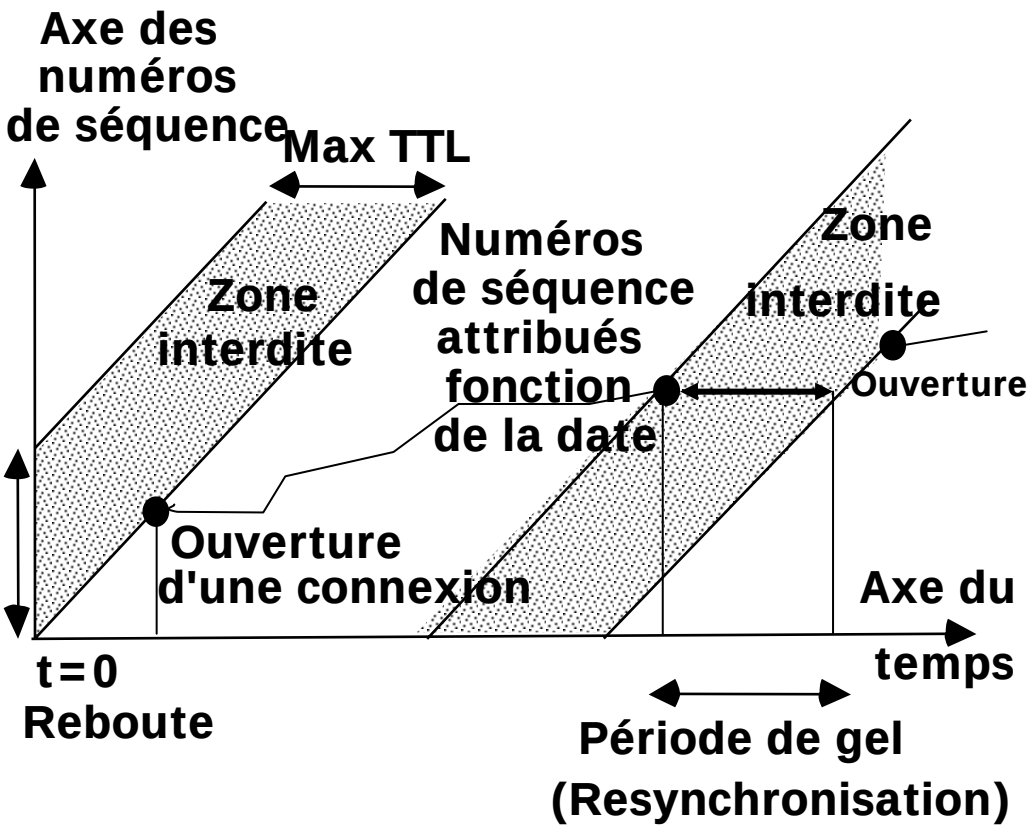


- **Collision d'appel** : deux messages SYN concurrents.
- La norme indique **qu'une seule connexion** doit-être ouverte.
- C'est le cas ici : les deux sites ont négociés en **quatre messages une connexion de référence unique (x,y).**

Approfondissement: Choix des numéros de séquence

- **Des connexions successives doivent recevoir des numéros de séquence initiaux différents et "aléatoires"**.
  - Pour différencier les messages ayant circulé sur les deux connexions
- **Pour des numéros de séquence sur 32 bits** le RFC 793 TCP suggère d'utiliser une **horloge sur 32 bits**.
  - **Horloge TCP:** un compteur incrémenté toutes les 4 micro-secondes ce qui le fait recycler environ en 4 heures ("wrap around").
  - **L'incrémentation n'est pas faite un par un** (trop d'interruptions) : mais selon une période qui peut être assez longue (exemple 1/2 seconde => incrémentation de 128000).
  - **Les implantations diffèrent de façon significative** sur le choix de ces valeurs.

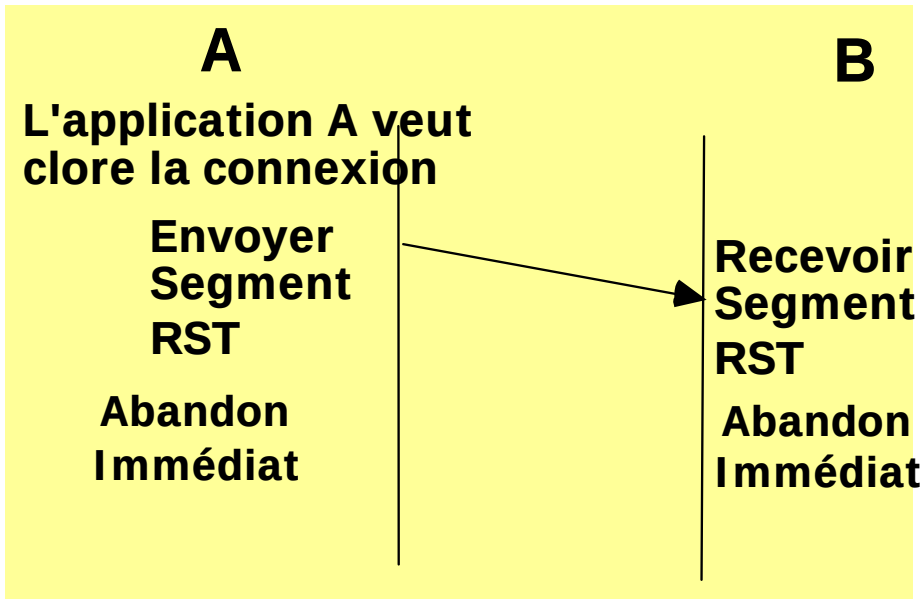
Choix des numéros de séquence : Règles de choix



- **Deux numéros de séquence identiques** correspondant à des messages **différents de connexions différentes** successives entre les mêmes ports ne doivent pas être attribués.
- **Le TTL ("Time To Live")** d'un paquet dans le réseau de communication permet de régler le délai de la zone interdite (durée du gel)..
- **Les numéros de séquence dans la zone interdite ne peuvent être attribués** car des paquets ayant ces numéros peuvent être en transit.

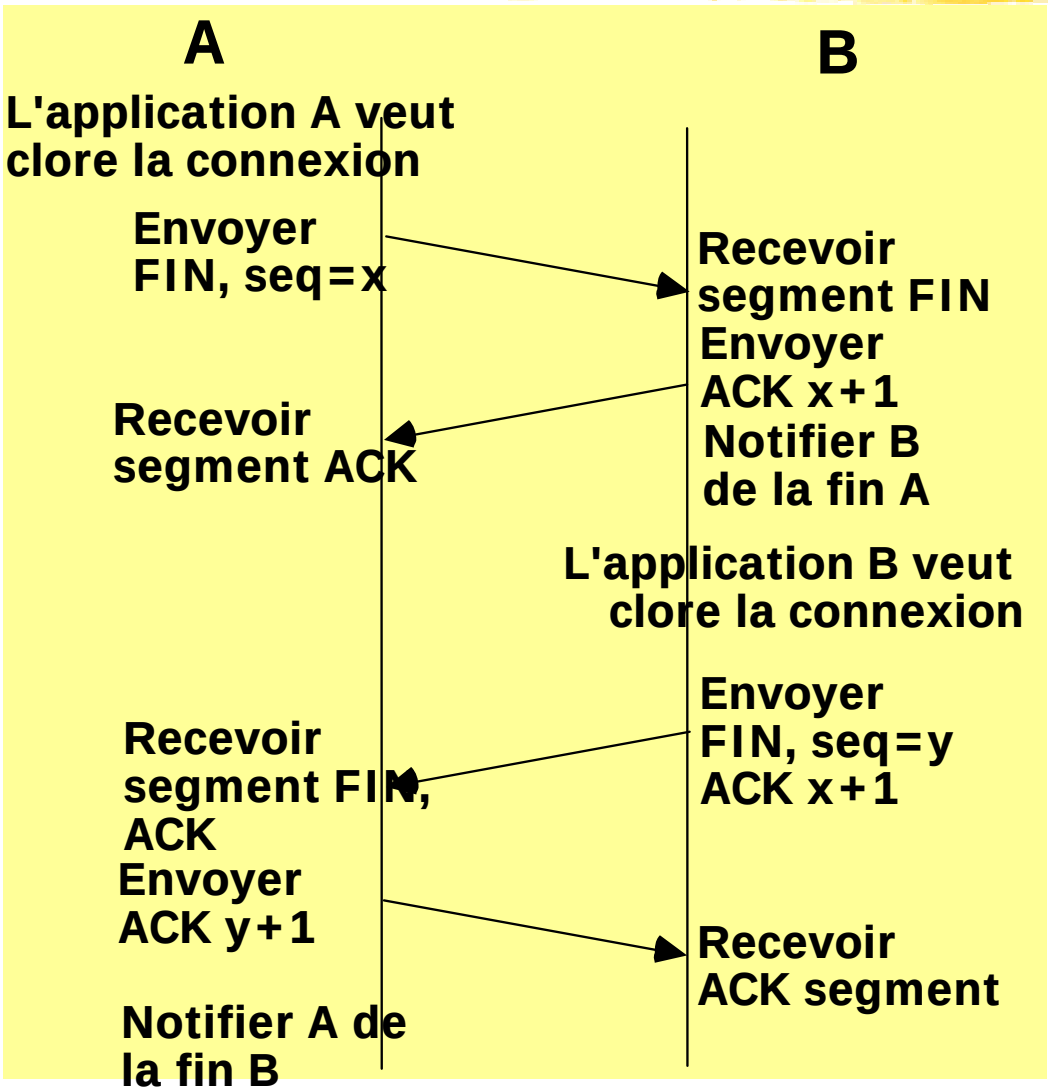
1) Protocole de fermeture de connexion: Libération abrupte

- **TCP Connection reset**
  - **Lorsqu'un segment est transmis avec le bit RST** : une fermeture inconditionnelle (immédiate) de la connexion est réalisée.



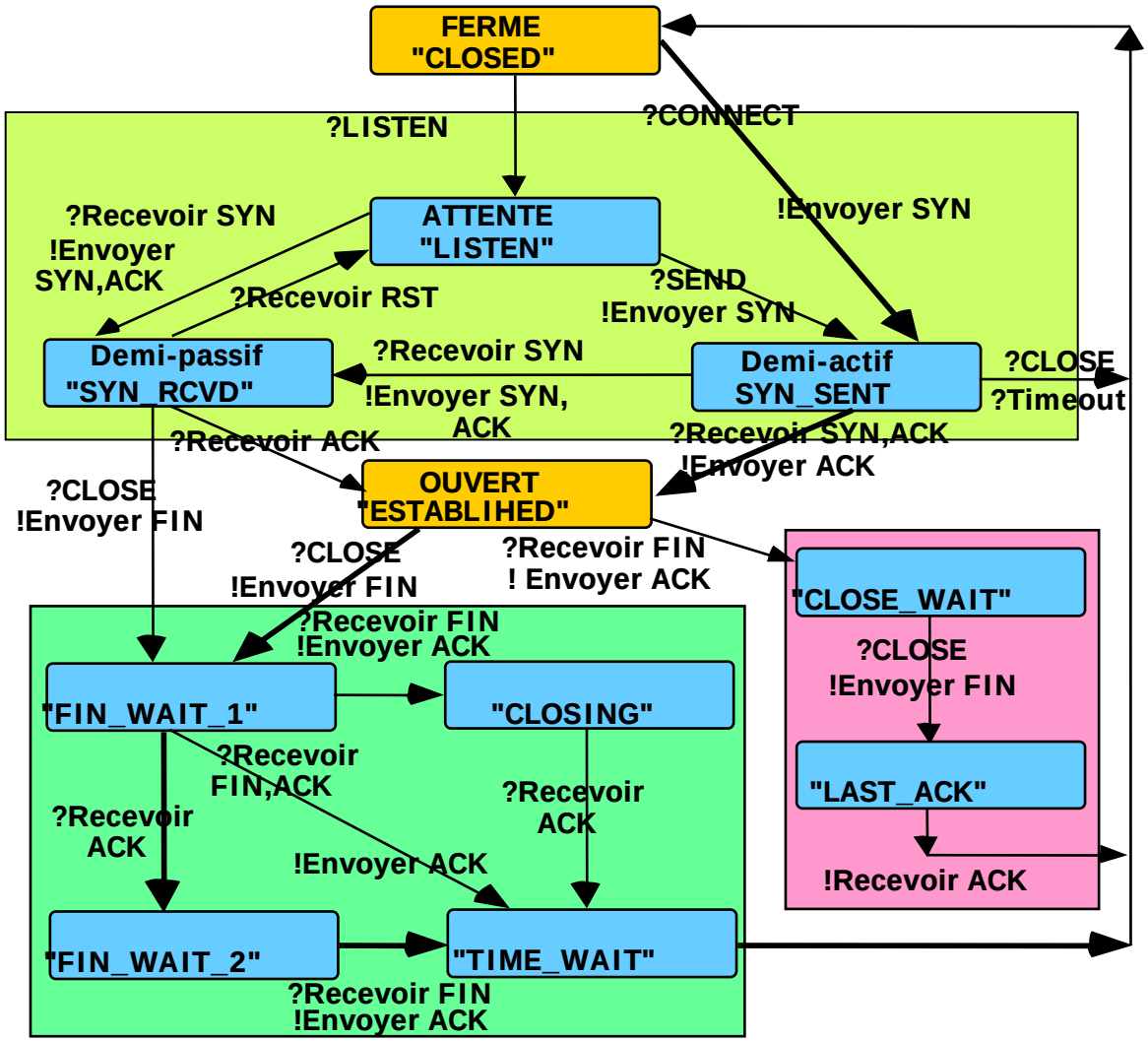
- **Le récepteur abandonne la connexion**.
- **Il notifie à l'application** la fin immédiate de la connexion
- **L'émetteur, le récepteur libèrent tout l'espace de travail** occupé par les segments en cours de transmission.

2) Protocole de fermeture de connexion: Libération négociée



- **TCP Graceful close**
  - **Chaque utilisateur doit générer** sa propre requête de déconnexion.
  - **On garantit la livraison correcte** de toutes les données échangées sur la connexion.

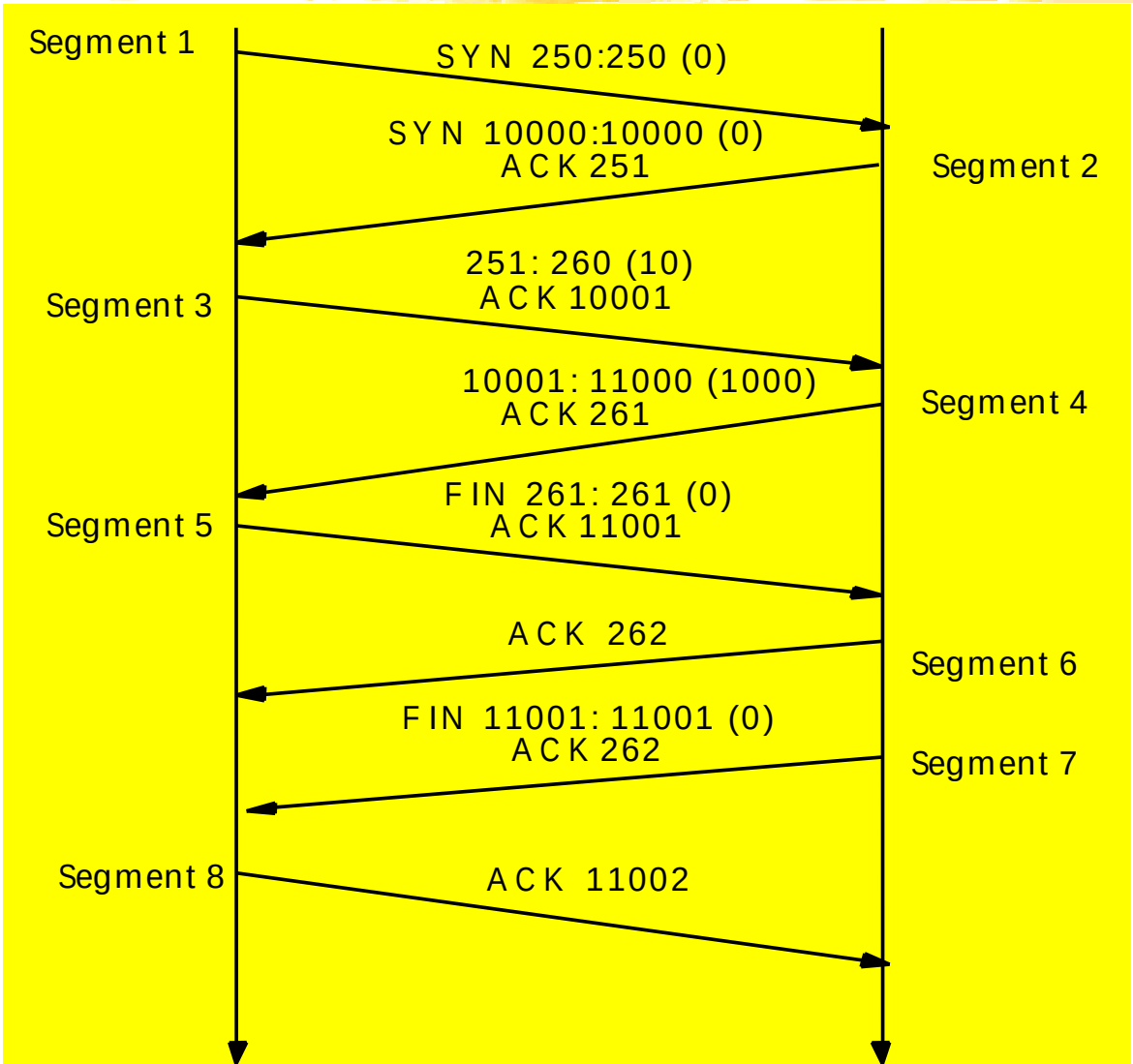
TCP : Automate de connexion/déconnexion



Transfert des données en TCP

- **Structuration des échanges par octets ("stream").**
  - Les blocs d'octets consécutifs à transmettre sont placés dans des segments TCP.
  - TCP décide de l'envoi d'un segment indépendamment des primitives d'émission en effectuant **du groupage** ou de la **segmentation** pour envoyer un segment de taille optimale compte tenu de la gestion des fenêtres de contrôles de flux et de congestion.
- **Contrôle de séquence**
  - Chaque octet est numéroté sur 32 bits.
  - Chaque segment porte le numéro de séquence du premier octet de données.
- **Contrôle de flux**
  - Basé sur le crédit variable (taille de la fenêtre 'window size')
  - Crédit s'exprimant en nombre d'octets.
- **Contrôle d'erreur**
  - Le contrôle d'erreur utilise une stratégie **d'acquittement positif** avec **temporisateur** et **retransmission**.
  - **Son originalité** réside dans l'algorithme de calcul de la valeur du temporisateur de retransmission : la valeur du temporisateur armé lors d'une émission dépend du délai d'aller-retour mesuré pendant une période récente.

Exemple d'échange de segments connexion/deconnexion en TCP

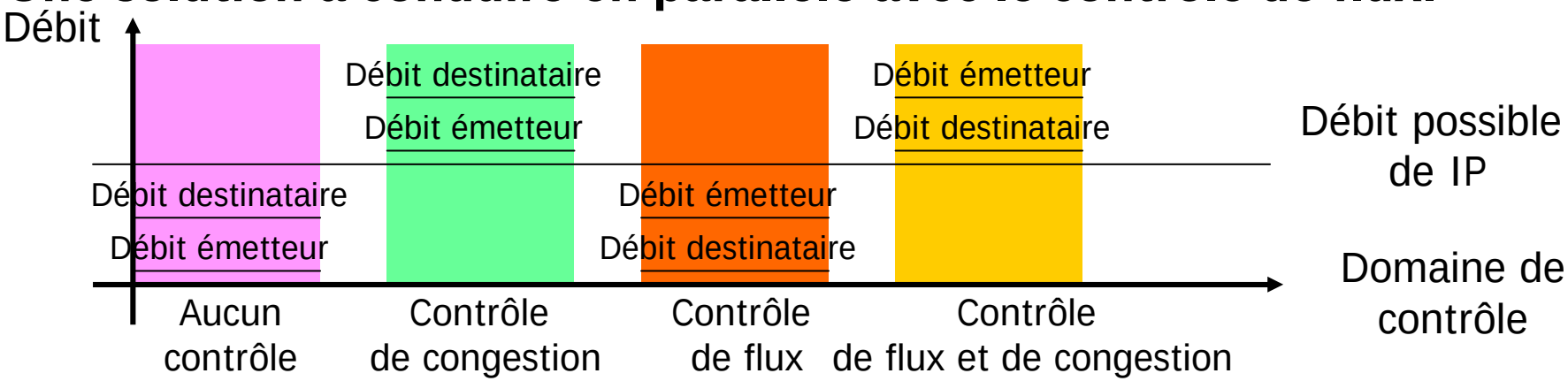


TCP "Transmission Control Protocol"

Approfondissements TCP

TCP: Approfondissements
1) Contrôle de congestion

- **Contrôle de congestion réalisé par TCP** : mais qui s'applique en fait au niveau réseau IP (c'est IP qui devrait faire ce travail).
- **Congestion TCP :** une solution basée sur un contrôle d'admission dans IP
  - TCP ne doit pas soumettre un trafic supérieur à celui de IP.
  - S'il y a des pertes de segments TCP suppose qu'elles sont dues à de la congestion (pas au bruit) => TCP ralentit son débit.
- **Une solution à conduire en parallèle avec le contrôle de flux.**

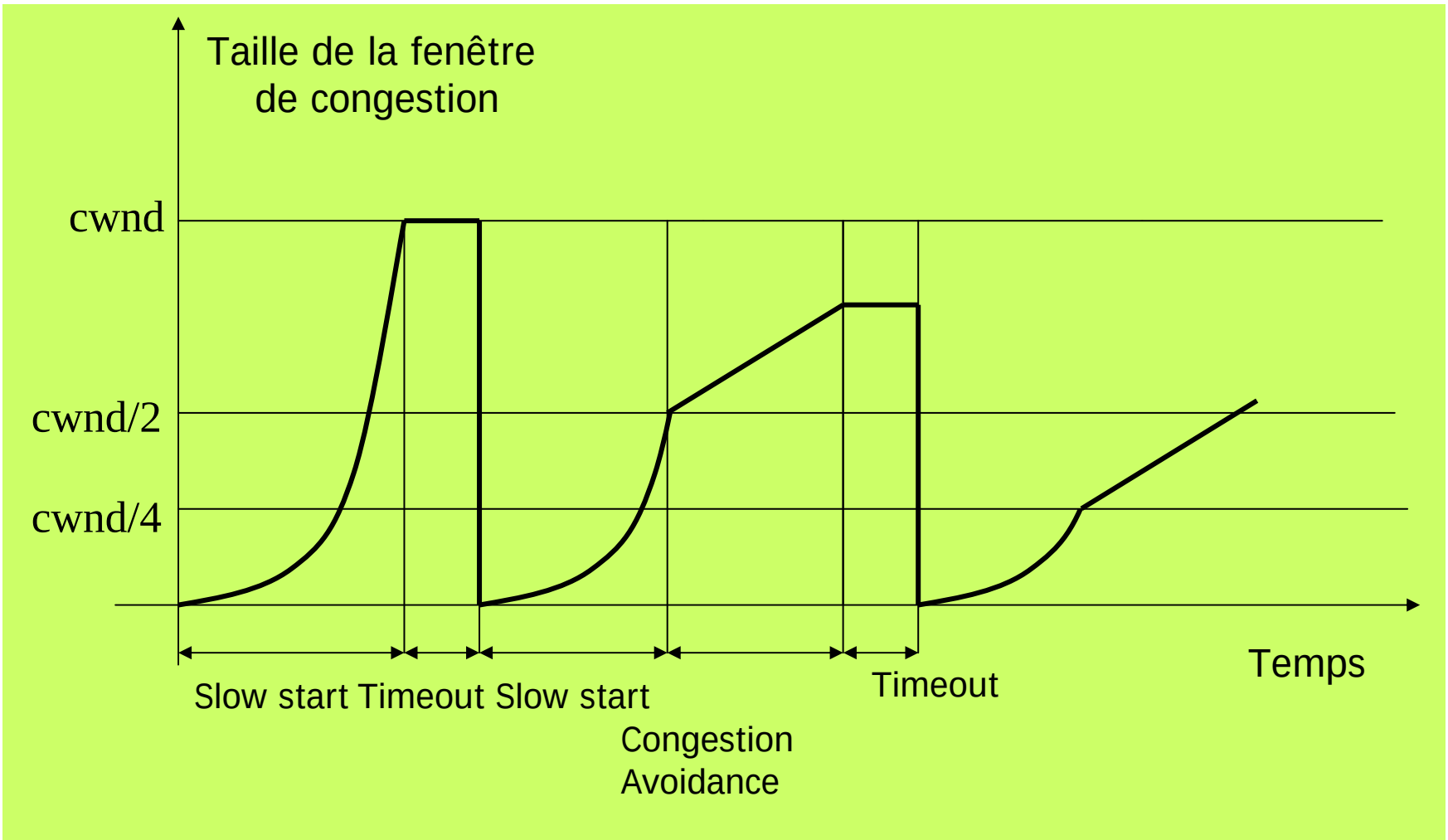


- **=> Une fenêtre pour le contrôle de flux.**
- **=> Une fenêtre pour le contrôle de congestion.**
- **=> La fenêtre réellement utilisée : la plus petite des deux fenêtres.**

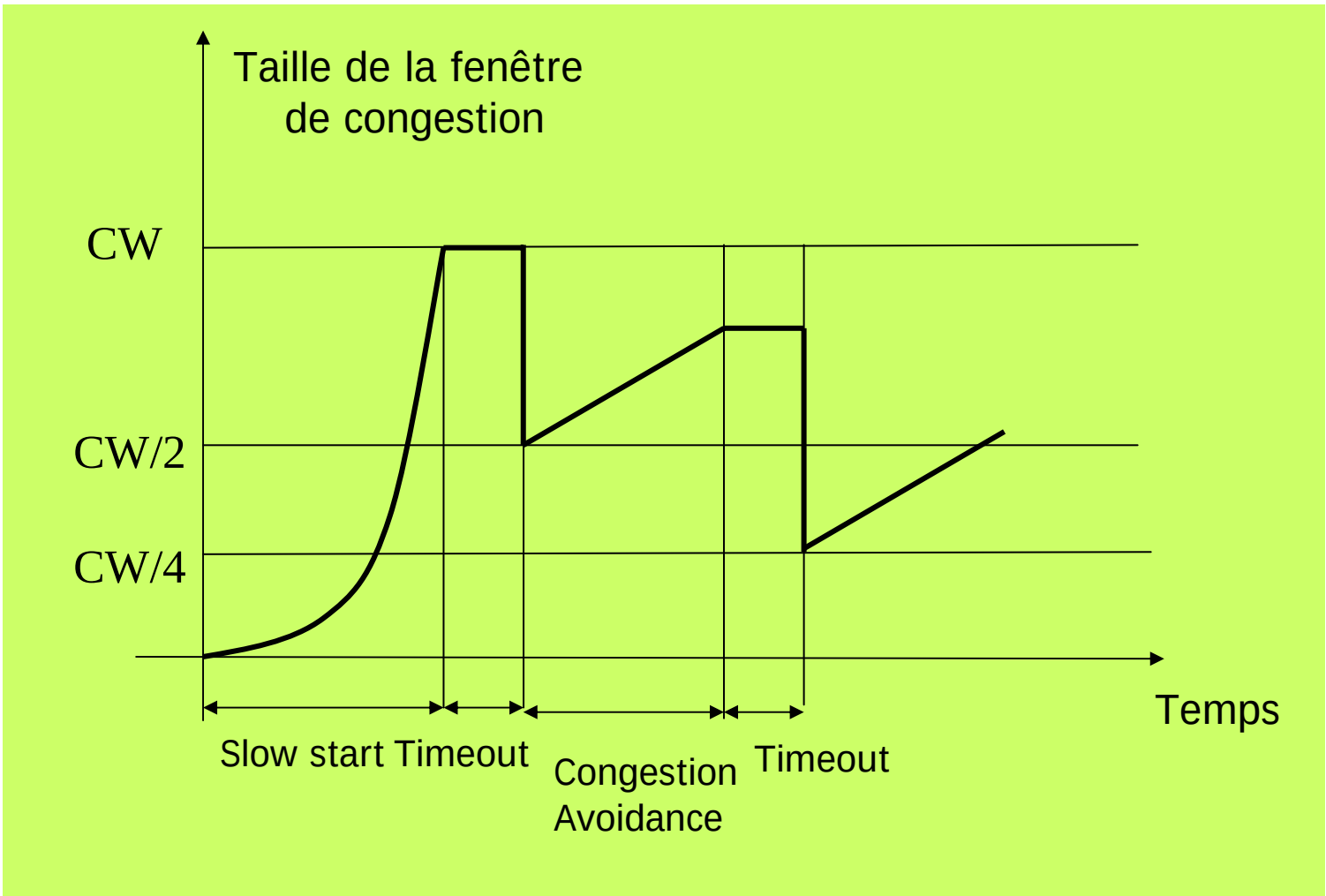
Dimensionnement de la fenêtre de congestion : départ lent 'slow start'

- **Recherche du débit maximum du réseau en TCP Tahoe**
- **A) Démarrage avec croissance exponentielle.**
  - Fenêtre de congestion initiale de taille 1 : un segment de MSS octets.
  - A chaque acquittement positif (transmission réussie): on double la taille de la fenêtre en transmettant 1, 2 , 4, 8, ... segments.
  - Jusqu'à un échec de transmission (pas d'acquittement) pour une valeur cwnd: on considère qu'il est du au phénomène de congestion.
  - On repart en démarrage lent jusqu'à la moitié de la valeur soit cwnd/2.
- **B) Recherche fine avec une croissance linéaire.**
  - Phase baptisée évitement de congestion: **'congestion avoidance'.**
  - Lorsque la fenêtre atteint la moitié de sa taille: reprise de la croissance en augmentant par incréments de 1.
- **Optimisation :** ne pas refaire la phase de slow start (TCP Reno).

Evolution de la fenêtre de congestion ('slow start'):TCP Tahoe



Evolution de la fenêtre de congestion ('slow start'): TCP Reno



2) Retransmission rapide Fast Retransmit / Fast recovery

- **Technique des acquittements répétés :** pour signaler un segment hors séquence.
  - Une astuce pour remplacer un acquittement négatif
  - Deux acquittements positifs peuvent apparaître sur simple déséquencement par le réseau.
- **Si on a trois acquittements d'un segment :** indication forte qu'un segment qui suit est perdu.
- **Retransmission du segment sans attendre** le délai de garde => **Solution baptisée 'fast retransmit'.**
- **Ne pas ramener la taille de la fenêtre de congestion à 1** pour un démarrage lent 'slow start' : il y a du trafic à écouler
- **Solution baptisée 'fast recovery'.**
- **Autre solution :** acquittements négatifs dans la zone option (TCP avec SACK).

3) Adaptation du temporisateur aux temps de réponse

- **Notion de RTT 'Round Trip Time' :** mesure du délai d'aller retour (émission d'un segment => réception de son acquittement positif).
- **Notion de RTO 'Retransmission Timeout' :** adaptation des délais de garde en fonction des RTT.
- **Possibilités de différentes formules** donnant RTO en fonction de RTT.
- **La méthode de base :**
  - **RTT mesuré** : de préférence sur chaque segment.
  - **RTT lissé** : sRTT = (1 – g) * sRTT + g * RTT
  - **Valeur de RTO** le temporisateur : RTO = 2 * sRTT
  - **Problème:** ne tient compte que de la moyenne de RTT.

La méthode de Karn/Partridge

- **Utiliser la moyenne et la variance.**
- **RTT courant** : ne pas utiliser les mesures de RTT lors d'une retransmission et doubler le RTO à chaque retransmission car on est probablement en congestion (idée de retard binaire).
- **RTT lissé** : sRTT = (1 – g) * sRTT + g * RTT
- **Ecart type du RTT** 'standard deviation' écart par rapport à la moyenne
  D = (1 – h) * D – h * | RTT – sRTT |
- **Valeurs recommandées** pour les coefficients g et h:

  g = 0.125 : prise en compte sur une moyenne de 8 mesures.

  h = 0.25 : prise en compte sur une moyenne de 4 mesures.
- **Définition de RTO le temporisateur** : RTO = sRTT + 4 D
- **Plus RTT est grand et plus grande est sa variation : plus long est le temporisateur.**
- **Autre méthode : Jacobson/Karels.**

4) Gestion de taille des segments 'Silly Window Syndrome'

- **Situation : traitement lent du récepteur.**
  - **Ré ouverture lente** de petits crédits d'émission.
  - **Sur ces petits crédits** => émission de petits segments.
  - **Syndrome de la fenêtre ridicule ('silly')** => Mauvaise utilisation du réseau car les messages sont de petite taille.
- **Solutions : Le groupage (Algorithme de Naggle)**
  - **Le récepteur ne transmet que des crédits importants.**
    - Une fraction significative de l'espace total des tampons récepteurs.
  - **L'émetteur diffère les émissions**
    - Tant qu'un segment de taille suffisante à émettre n'est pas réuni (définition d'un petit segment : relativement au MSS du destinataire).
    - Ou tant qu'il y a des données non acquittées.

5) Protection contre le passage à zéro des numéros de séquence

- **Passage par zéro ('Wrap Around') :** Twrap =2^{31}/débit binaire
  - Ethernet (10Mbps): Twrap = 1700 s (30 minutes)
  - Ethernet Gigabit (1 Gbps): Twrap = 17 secs.
  - Si la durée de vie dans le réseau (MSL Maximum Segment Life) est possiblement supérieure on a un risque d'interférence entre numéros.
- **PAWS : Protection against wrapped sequence numbers.**
- **Utilisation de l'option 'timestamp'** : estampillage TSVAL 32 bits pour la mesure correcte des délais d'aller retour dans la zone option.
- **A partir d'une horloge à la milliseconde**: repassage par 0 en 24 jours.
- **Solution:** combinaison du numéros de séquence habituel et de l'estampille TSVAL pour obtenir un numéro sur 64 bits : comparaison des valeurs sur 64 bits.
- **Le problème de repassage par zéro ne peut se poser que sur 24 jours:** acceptable en relation avec la durée de vie des segments.

Conclusion TCP

- **Un protocole de transport fiable efficace.**
  - Indispensable dans le domaine des réseaux basse et moyenne vitesse.
  - Des optimisations nombreuses : versions Tahoe, Reno et New-Reno, Sack TCP ... toujours étudiées.
- **Problèmes posés**
  - **Passage aux réseaux gigabits en TCP.**
    - Différentes expériences : modifications, dimensionnement pour apporter des réponses à cette question.
  - **Support des applications multimédia**
    - Nécessite la définition de nouveaux protocoles de transports qui respectent les besoins de qualité de service de ces applications (RTP ,....)

Exemple des protocoles et services de transport INTERNET

Un service pour TCP et UDP: les sockets

Généralités interface "socket"

- **Définition en 1982** : une interface de programmation d'applications réseaux (API) pour la version UNIX BSD.
- **Existence de plusieurs autres interfaces réseaux :** TLI, NETBEUI, ...
- **Objectifs généraux**
  - Fournir des moyens de communications entre processus (IPC) **utilisables en toutes circonstances**: échanges locaux ou réseaux.
  - Cacher **les détails d'implantation** des couches de transport aux usagers.
  - Si possible cacher les **différences entre protocoles de transport hétérogènes** sous une même interface (TCP, Novell XNS, OSI)
  - Fournir une interface d'accès qui se rapproche **des accès fichiers pour simplifier la programmation** => En fait des similitudes et des différences importantes entre programmation socket et fichier.

Choix de conception des sockets

- **Une "socket" (prise) est un point d'accès de service pour des couches transport :** essentiellement TCP/UDP mais aussi d'autres protocoles (OSI, DECNET...).
- **Une socket est analogue à un objet** (de communication)
  - **Un type**:
    - Pour quel protocole de transport est-elle un point d'accès de service?
    - Quelle est la sémantique de l'accès au service?
  - **Un nom:** identifiant unique sur chaque site (en fait un entier 16 bits).
  - **Un ensemble de primitives** : un service pour l'accès aux fonctions de transport.
  - **Des données encapsulées :**
    - **un descriptif** (pour sa désignation et sa gestion)
    - **des files d'attente** de messages en entrée et en sortie.

Désignation des sockets

- **Identifier complètement une socket** dans un réseau et pour une couche transport : un couple NSAP,TSAP.

  Exemple Internet TCP: Numéro de port , Adresse IP
- **Gestion par l'IANA**
- **A) Numéros de ports réservés :** numéros de ports réservés pour des services généraux **bien connus** ou **"well-known ports"** (numéros inférieurs à 1023).
  - **Exemples ports UDP** : Echo server: 7, TFTP: 69.
  - **Exemples ports TCP** : Telnet: 23, DNS: 53, HTTP: 80.
- **B) Numéros de ports enregistrés ('registered'):** (entre 1024 et 49151) pour des applications ayant fait une demande.
- **C) Numéros de ports privés ('private') (dynamiques) :** les autres numéros entre 49152 et 65535 qui sont attribués dynamiquement aux sockets utilisateurs (clients).

Choix de conception des sockets avec TCP

- TCP est un transport **fiable** en **connexion** et en mode **bidirectionnel point à point**.
- **Une socket TCP peut être utilisée par plusieurs connexions** TCP simultanément.
- **Une connexion est identifiée par le couple** d'adresses socket des deux extrémités.
- **Un échange TCP est orienté flot d'octets**.
  - Les zones de données qui correspondent à des envois successifs ne sont pas connues à la réception.
  - Pour optimiser TCP peut tamponner les données et les émettre ultérieurement.
  - L'option **"push"** qui permet de demander l'émission immédiate d'un segment.
  - L'option **"urgent"** qui devrait permettre l'échange rapide de données exceptionnelles avec signalement d'arrivée.

Choix de conception des sockets avec UDP

- UDP est une couche transport **non fiable, sans connexion,** en mode **bidirectionnel** et **point à point**.
- **L'adresse UDP d'une socket** (Numéro de port UDP , Adresse IP) sur l'Internet à la même forme que celle d'une socket TCP.
- **Mais les deux ensembles d'adresses sont indépendants** : une communication UDP n’a rien à voir avec une communication TCP.
- Un échange UDP est sans connexion (échange de **datagrammes)**.
- Les zones de données qui correspondent à des envois successifs sont **respectées** à la réception.

Exemple des protocoles et services de transport INTERNET

Les primitives de l'interface socket

Exemple en langage C en UNIX.

Primitive socket

- **Permet la création d'un nouveau point d'accès de service transport:**
  - définition de son type.
  - allocation de l'espace des données.
- **Trois paramètres d'appel**
  - **Famille** d'adresses réseaux utilisées locale, réseau IP, réseau OSI ...
  - **Type** de la socket (du service) sémantique de la communication.
  - **Protocole** de transport utilisé.
- **Un paramètre résultat:** le numéro de descripteur socket.
- **Profil d'appel de la primitive en C**

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
int socket (int famille, int type, int protocole);
```

Approfondissement des paramètres de la primitive socket

- **Paramètre Famille**
  - AF_UNIX : Communication locale (i-node)
  - AF_INET : Communication Internet
  - AF_ISO : Communication ISO

    ....
- **Paramètre Type**
  - **SOCK_STREAM** : Flot d'octets en mode connecté
    (ne préserve pas les limites de l'enregistrement)
  - **SOCK_DGRAM** : Datagramme en mode non connecté
    (préserve les limites de l'enregistrement)
  - **SOCK_RAW** : Accès aux couches basses.
  - **SOCK_SEQPACKET** : Format structuré ordonné
    (protocoles différents de l'Internet)
- **Paramètre Type de protocole**

Valeur	Relation avec le paramètre type
IPPROTO_TCP	SOCK_STREAM
IPPROTO_UDP	SOCK_DGRAM
IPPROTO_ICMP	SOCK_RAW
IPPROTO_RAW	SOCK_RAW

Primitive bind

- **Primitive pour l'attribution d'une adresse de socket** à un descripteur de socket.
- **N’est pas réalisé** lors de la création du descriptif (socket).
  - Un serveur (qui accepte des connexions) doit définir sur quelle adresse.
  - Un client (qui ouvre des connexions) n'est pas forcé de définir une adresse (qui est alors attribuée automatiquement).
- **Profil d'appel de la primitive**

```
#include <sys/types.h>
#include <sys/socket.h>
int bind (    int s,
              struct sockaddr_in *mon_adresse,
              int longueur_mon_adresse        )
```

- **Trois paramètres d'appel**
  - Numéro du descriptif de Socket (s).
  - Structure de donnée adresse de socket Pour internet type sockaddr_in.
  - Longueur de la structure d'adresse.

Approfondissement concernant la primitive bind

- **Descripteur d'adresse de socket**

```
#include <sys/socket.h>
struct sockaddr_in {
    short           sin_family;
    u_short         sin_port;
    struct in_addr  sin_addr;
    char            sin_zero[8];   };
```

- **Un exemple d'exécution de "bind" pour les protocoles Internet.**

```
struct servent *sp
struct sockaddr_in sin

/* Pour connaître le numéro de port */
if((sp=getservbyname(service,"tcp")==NULL)
/* cas d'erreur */

/* Remplissage de la structure sockaddr */
/* htonl convertit dans le bon ordre */
/* INADDR_ANY adresse IP du site local */
sin.sin_family= AF_INET;
sin.sin_port = sp -> s_port;
sin.sin_addr.s_addr=htonl(INADDR_ANY):

/* Création d'une socket internet */
if ((s=socket(AF_INET,SOCK_STREAM,0))<0)
/* cas d'erreur */

/* Attribution d'une adresse */
if (bind(s, &sin, sizeof(sin)) < 0)
/* cas d'erreur */
```

Primitive listen

- **Utilisé dans le mode connecté lorsque plusieurs clients** sont susceptibles d'établir plusieurs connexions avec un serveur.
- **Indique le nombre d'appel maximum attendu** pour réserver l'espace nécessaire aux descriptifs des connexions.
- **La primitive listen est immédiate** (non bloquante).
- **Profil d'appel :** int listen (int s , int max_connexion)

  s : Référence du descripteur de socket

  max_connexion : Nombre maximum de connexions.

Primitive accept

- **La primitive accept** permet de se bloquer en attente d'une nouvelle demande de connexion (donc en mode connecté TCP).
- **Après accept,** la connexion est complète entre les deux processus.
- **Le site qui émet accept exécute une ouverture passive.**
- **Pour chaque nouvelle connexion entrante** la primitive fournit un pointeur sur un nouveau descriptif de socket qui est du même modèle que le descritif précédemment créé.
- **Profil d'appel**

```
#include <sys/types.h>
#include <sys/socket.h>
int accept (    int ns,
                struct sockaddr_in *addr_cl,
                int lg_addr_cl)

ns              : Référence nouvelle socket
addr_cl         : L'adresse du client.
lg_addr_cl: La longueur de l'adresse.
```

Approfondissement concernant les primitives listen et accept

- **Exemple de code UNIX :** pour un serveur qui accepte des connexions successives et qui créé un processus pour traiter chaque client.

```
#include <sys/socket.h>
/* Adresse socket du client appelant */
struct sockaddr_in from;
quelen = ... ;
if (listen (s, quelen) <0 )
        Cas d'erreur
/* On accepte des appels successifs */
/* Pour chaque appel on créé un processus */
if((g=accept(f,&from,sizeof(from)))<0)
        Cas d'erreur
if ( fork ...
/* Processus traitant de connexion*/
```

Primitive connect

- **La primitive connect** (bloquante) permet à un client de demander l'ouverture (**active**) de connexion à un serveur.
- **L'adresse du serveur doit être fournie.**
- **La partie extrémité locale relative au client** est renseignée automatiquement.
- **Ensuite le client ne fournit plus l'adresse du serveur** pour chaque appel mais le descriptif de la socket (qui contient l'adresses serveur).
- **Profil d'appel**

```
#include <sys/types.h>
#include <sys/socket.h>
int connect (  int s,
               struct sockaddr_in *addr_serv,
               int lg_addr_serv)

s              : La référence de la socket
addr_serv      : L'adresse du serveur.
lg_addr_serv : La longueur de l'adresse.
```

Primitives send, recv

- **Les primitives send, recv (bloquantes)** permettent l'échange effectif des données.
- **Le profil d'appel est identique à celui des primitives read et write sur fichiers** avec un quatrième paramètre pour préciser des options de communications.
- **Profil d'appel**

```
#include <sys/types.h>
#include <sys/socket.h>
int send  (int s, char *zone,
           int lg_zone, int options_com)
int recv  (int s,char *zone,
           int lg_zone, int options_com)
s                          : La référence de la socket
zone                       : La zone à échanger.
lg_zone          : La longueur de la zone.
options_com      : Les options (données urgentes , ….)
```

Primitives sendto, recvfrom

- **Les primitives sendto, recvfrom** permettent l'échange des données dans le mode non connecté UDP.
- **On doit préciser l'adresse destinataire** dans toutes les primitives **sendto** et **l'adresse émetteur** dans les **recvfrom**.
- **Profil d'appel**

```
#include <sys/types.h>
#include <sys/socket.h>
int sendto (int s,
            char *zone,
            int lg_zone,
            int options_com,
            struct sockaddr_in *addr_dest,
            int lg_addr)
int recvfrom (int s,
            char *zone,
            int lg_zone,
            int options_com,
            struct sockaddr_in *addr_emet,
            int *lg_addr)
addr_dest  : L'adresse du destinataire.
addr_emet  : L'adresse de l'émetteur.
lg_addr    : La longueur de l'adresse.
```

Primitives shutdown, close

- **Shutdown** permet la terminaison des échanges sur une socket suivi de la fermeture de la connexion :
  - **Profil d'appel** : int shutdown(s , h); Pour la socket s.
  - **h = 0** : l'usager ne veut plus recevoir de données
  - **h = 1** : l'usager ne veut plus envoyer de données
  - **h = 2** : l'usager ne veut plus ni recevoir, ni envoyer.
- **Close :** Permet la fermeture d'une connexion et la destruction du descriptif.
  - Profil d'appel

```
#include <sys/types.h>
#include <sys/socket.h>
int close ( int s )
```

Résumé : Interface socket

- **Fonctionnement en TCP**
  - Serveur.
    socket
    bind
    listen
    accept
    recv, send
    close
  - Client.
    socket
    connect
    recv, send
    close
- **Fonctionnement en UDP**
    socket
    recvfrom, sendto
    close

Bibliographie
Niveau transport

- A.S. Tannenbaum "**Computer Networks**" Prentice Hall
- W.R. Stevens "**TCIP/IP Illustrated, The protocols**" , Addison Wesley
- Internet Web : multiples cours TCP.