Le protocole TCP : Transmission Control Protocol

(RFC 793,1122, 1323, 2018 et 2581)

TCP : Transmission Control Protocol

- Protocole TCP, Transmission Control Protocol
- Fiabilité 100 %
  - non-duplication
  - ordonnancement
- Contrôle d'erreurs
  - Garantie l'intégrité
- Orienté connexion :
  - Ouverture/fermeture explicite de la connexion

TCP : Transmission Control Protocol

- Full-duplex
  - Transfert bidirectionnel de données sur une connexion
- Contrôle de flux
  - L'émetteur ne submerge pas le récepteur
- Contrôle de congestion :
  - l'émetteur adapte son débit d'envoi de paquets à l'état du réseau

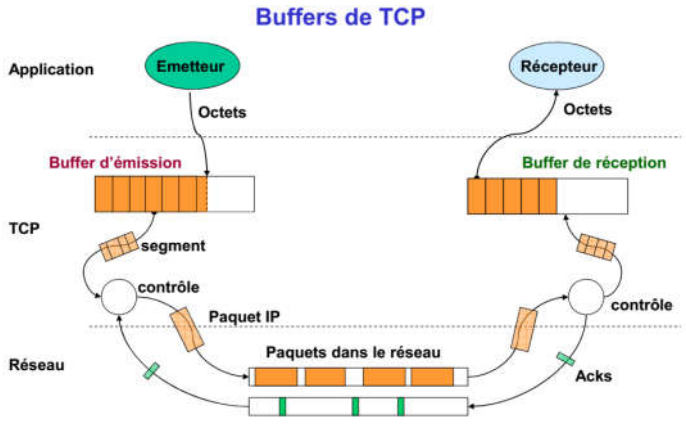
TCP : Transmission Control Protocol

- Fenêtre d'émission:
  - Les mécanismes de contrôle de flux et de congestion règlent la taille de la fenêtre
- Tampons de réception et d'émission

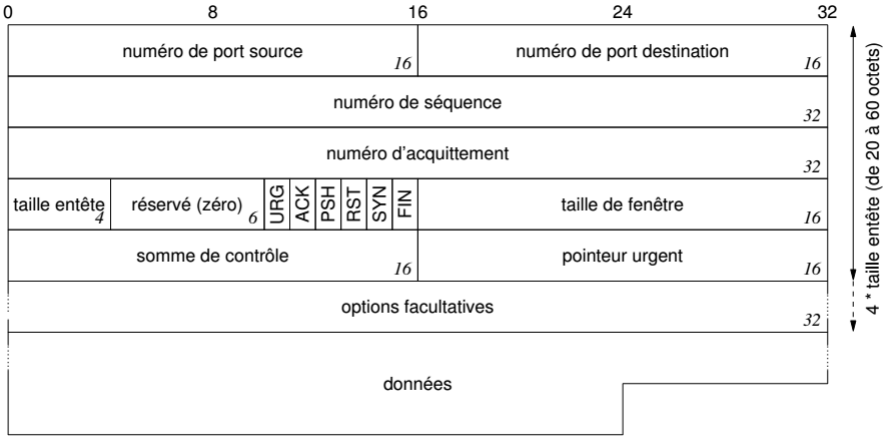
TCP : Transmission Control Protocol

- TCP est orienté flux d'octets
  - le processus émetteur "écrit" des octets sur la connexion TCP
  - le processus récepteur "lit" des octets sur la connexion TCP
- TCP ne transmet pas d'octets individuels
  - en émission :
    - TCP "bufferise" les octets jusqu'à en avoir un nombre raisonnable
    - TCP fabrique un segment et l'envoie

TCP : Transmission Control Protocol



Format du segment TCP



Format du segment TCP

- Port Source (Source Port) :
  - Taille : 2 octets
  - indiquent le ports utilisé par l'application source.
- Port Destination (Destination Port) :
  - Taille : 2 octets
  - Numéro de l'application à laquelle les données sont destinées

Format du segment TCP

- Numéro de séquence (Sequence Number) : il permet de repositionner les données transportées par le segment dans le flux d'octets à reconstituer du côté réception.

Format du segment TCP

- Numéro de séquence d'émission: numéro du premier octet de données dans le segment
  - sauf pour un segment avec SYN=1.
    - Si le bit SYN est à 1 (c.-à-d. si le segment est une demande de connexion), le numéro de séquence signale au destinataire que le prochain segment de données qui sera émis commencera à partir de l'octet Numéro de séquence d'émission + 1.

Format du segment TCP

- Numéro d'acquittement (Acknowledgment Number) : indique le numéro du prochain octet attendu par le destinataire.
  - Si dans le segment reçu FIN=1 et #Seq= x, alors le #Ack renvoyé est x+1 (x est interprété comme étant le numéro de l'octet FIN et que cet octet est acquitté).
- TCP n'acquitte pas les segments mais les octets reçus

Format du segment TCP

- Taille entête ou offset de données (Offset Data) :
  - Il contient la longueur de l'en-tête du segment TCP.
  - L'Offset indique la position relative où commencent les données.
  - Il est exprimé en unité de 32 bits
    - des options (de taille variable) peuvent être intégrées à l'entête et des données de bourrage peuvent être rajoutées pour rendre la longueur de l'entête multiple de 4 octets.
  - Sa valeur est comprise entre 5 et 15

Format du segment TCP

- Drapeaux (Flags)
- Qui contenait six valeurs lors de la conception du protocole en 1981
  - URG, ACK, PSH, RST, SYN, FIN
  - et auquel on a ajouté deux drapeaux qui jouent un rôle dans le contrôle de congestions
    - CWR (Congestion Window Reduced)
    - ECE (ECN Echo). ECN signifiant Explicit Congestion Notification

Format du segment TCP

- Drapeaux (Flags)
  - URG= 1 si le pointeur de données urgentes est valide et = 0 sinon.
  - ACK=1 indique que le numéro d'acquittement est valide et il peut être pris en compte par le récepteur. ACK = 0 si l'accusé de réception est non valide.
  - PSH= 1 indique que les données doivent être remises à l'application dès leur arrivée et de ne pas les stocker dans une file d'attente.

Format du segment TCP

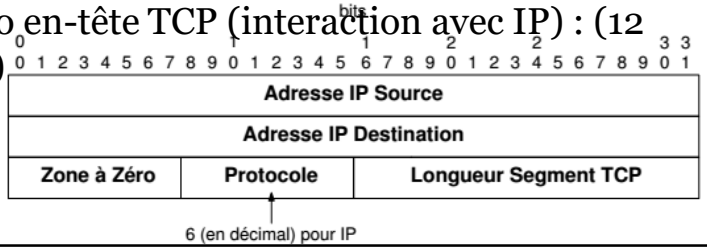
- Drapeaux (Flags)
  - RST= 1 pour demander la réinitialisation d'une connexion.
  - SYN= 1 et ACK= 0 servent à demander l'établissement de connexion.
  - SYN= 1 et ACK= 1 servent à accepter une demande de connexion.
  - FIN= 1 indique que l'émetteur n'a plus de données à émettre et demande de rompre la connexion de son côté.

Format du segment TCP

- Taille de fenêtre (Window) : indique le nombre d'octets que le destinataire peut recevoir.
- si Fenêtre= *F* et que le segment contient un Numéro d'acquittement = A, alors le récepteur accepte de recevoir les octets numérotés de *A* à *A + F -1*.

Format du segment TCP

- Somme de contrôle : pseudo–en-tête + en-tête + données
- Comme pour UDP mais obligatoire
  - (bourrage éventuel 1 octet à 0) + pseudo en-tête TCP
- Pseudo en-tête TCP (interaction avec IP) : (12 octets)

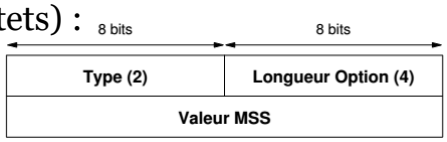


Format du segment TCP

- Pointeur d'urgence: indique l'emplacement des données urgentes dans un segment.
  - Il est valable uniquement si le bit URG=1.
  - Il contient le numéro d'octet qui suit immédiatement les données urgentes.
- Options (taille variable) : options nécessitant des traitements particuliers.
  - L'option la plus importante est certainement l'option MSS

Segment TCP : option MSS

- Format (4 octets) :



- Utilisée éventuellement pendant la phase de connexion (uniquement)
- Chaque côte indique la taille max des données des segments qu'il veut recevoir, appelée MSS (Maximum Segment Size) :
  - souvent MTU - 40 (en-têtes IP et TCP sans option)
  - dépendant de la taille des buffers de réception
  - par défaut 536 (576 octets pour le datagramme IP)

Segment TCP : option MSS

- Difficile à choisir pour l'Internet :
  - si trop petit, perte d'efficacité
  - si trop grand, risque de fragmentation
- L'idéal est le plus grand tel qu'aucun datagramme n'est fragmenté

TCP : la connexion

- Une connexion de type circuit virtuel est établie avant que les données ne soient échangées : appel + négociation + transferts
- Une connexion = une paire d'extrémités de connexion
- Une extrémité de connexion = couple (adresse IP, port)
  - Exemple de connexion : ((124.32.12.1, 49156), (19.24.67.2, 21))
- Une extrémité de connexion peut être partagée par plusieurs autres extrémités de connexions (multi-instanciation)

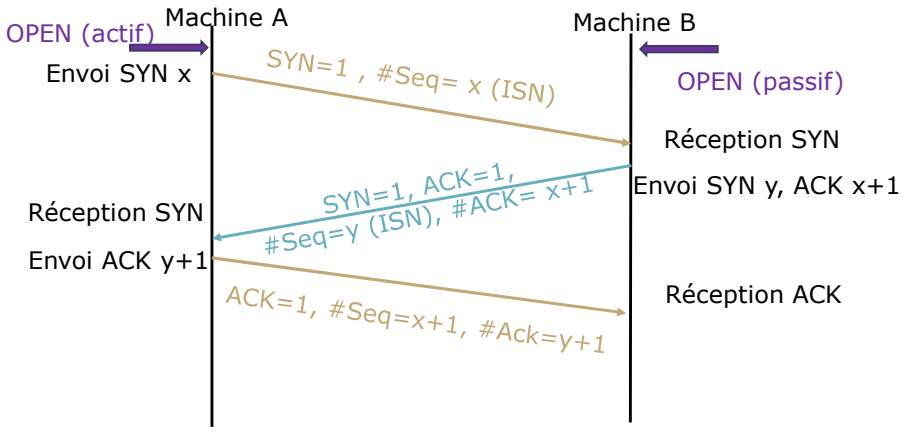
TCP : la connexion

- La mise en œuvre de la connexion se fait en deux étapes :
  - une application (extrémité) effectue une ouverture passive en indiquant qu'elle accepte une connexion entrante
  - une autre application (extrémité) effectue une ouverture active pour demander l'établissement de la connexion

Ouverture et fermeture de connexion TCP

- TCP définit deux modes d'ouverture :
  - mode passif : TCP est en attente d'une demande d'ouverture en provenance d'un autre système (défini ou non).
  - mode actif, TCP adresse une demande de connexion à un autre système.
- Trois phases
  - Ouverture (three way handshake)
    - Envoi de SYN (ouverture active)
    - Réponse avec SYN également (ouverture passive)
  - Transfert
  - Fermeture

Ouverture et fermeture de connexion TCP



***ISN** : **I**nitial **S**equence **N**umber*

Ouverture et fermeture de connexion TCP

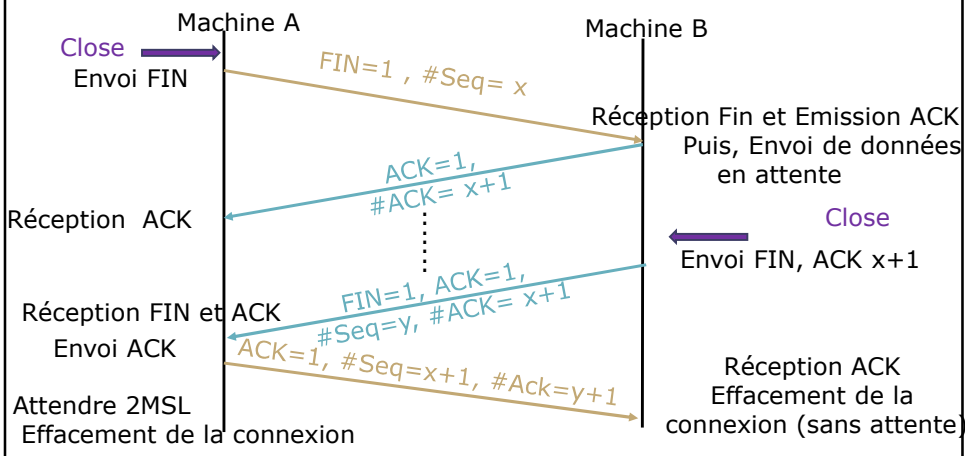
Pourquoi choisir des numéros de séquences aléatoires à l'établissement de connexion

- afin de distinguer des connexions courtes successives
- Renforcer la sécurité
  - Supposons que B a confiance en A à partir de l'@IP de A
    - e.g., accepter les demandes de création de compte issues de A
  - Un attaquant M veut se faire passer pour A
    - M ne souhaite pas recevoir des données issues de B mais seulement lui envoyer des données pour créer un compte sur B
  - M peut-il établir une connexion avec B en se faisant passer pour A?

Ouverture et fermeture de connexion TCP

- M peut-il établir une connexion avec B en se faisant passer pour A?
  - Si M ne peut pas intercepter les messages entre A et B (pour connaître les numéros de séquence), il ne pourra pas se faire passer pour A une fois la connexion entre A et B établie, à condition que A et B choisissent des numéros de séquences aléatoires.

Ouverture et fermeture de connexion TCP



Ouverture et fermeture de connexion TCP

- L'attente pendant 2MSL (Maximum Segment Lifetime) permet le cas échéant de retransmettre le dernier Ack.
  - En effet, si le dernier Ack se perd, le correspondant va le réclamer.
- Une connexion TCP ne peut utiliser les adresses IP et numéros de port d 'une connexion dans l'état 2MSL
- La destruction explicite de la connexion permet de réutiliser le port en toute quiétude.

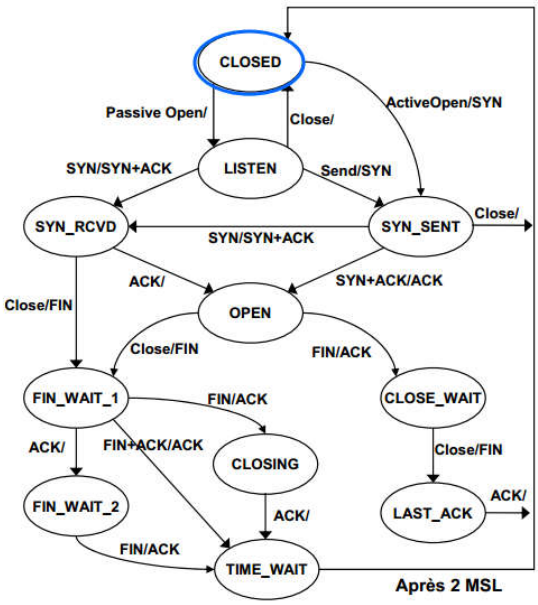
Ouverture et fermeture de connexion TCP

	A sends	B sends
1	SYN, **seq=0**	
2		SYN+ACK, seq=0, **ack=1** (expecting)
3	ACK, **seq=1**, ack=1 (ACK of SYN)	
4	"abc", **seq=1**, ack=1	
5		ACK, seq=1, **ack=4**
6	"defg", **seq=4**, ack=1	
7		seq=1, **ack=8**
8	"foobar", **seq=8**, ack=1	
9		seq=1, **ack=14**, "hello"
10	**seq=14**, ack=6, "goodbye"	
11,12	**seq=21**, ack=6, FIN	seq=6, **ack=21** ;; ACK of "goodbye", crossing packets
13		seq=6, **ack=22** ;; ACK of FIN
14		seq=6, **ack=22**, FIN
15	**seq=22**, ack=7 ;; ACK of FIN	

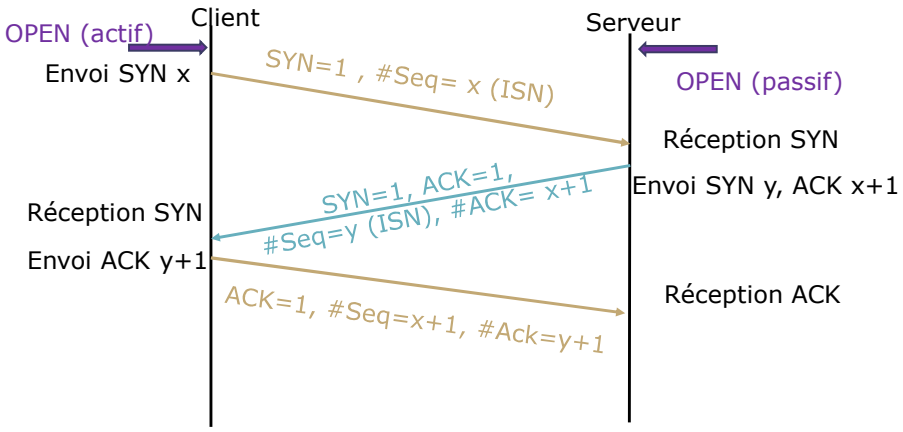
Fermeture brutale d'une connexion

- Drapeau RST à 1
- Signifie qu'il a y eu un problème grave
- Connexion libérée immédiatement
- Données non traitées/segments retardés sont perdus
- Applications sont informées
- Utilisé aussi pour refuser une demande de connexion

- Diagramme états/transition

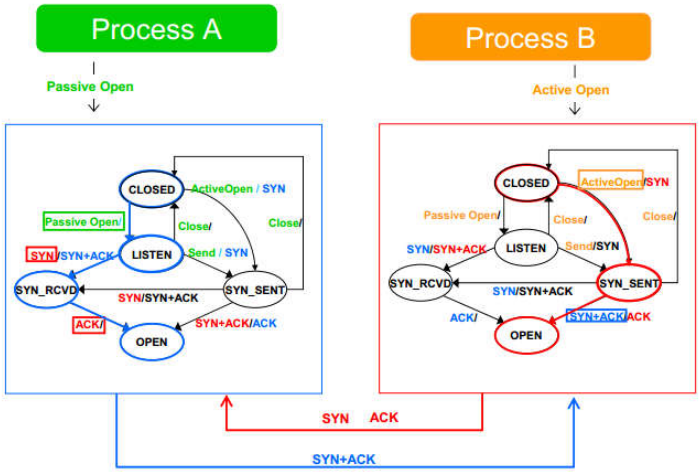


Ouverture et fermeture de connexion TCP

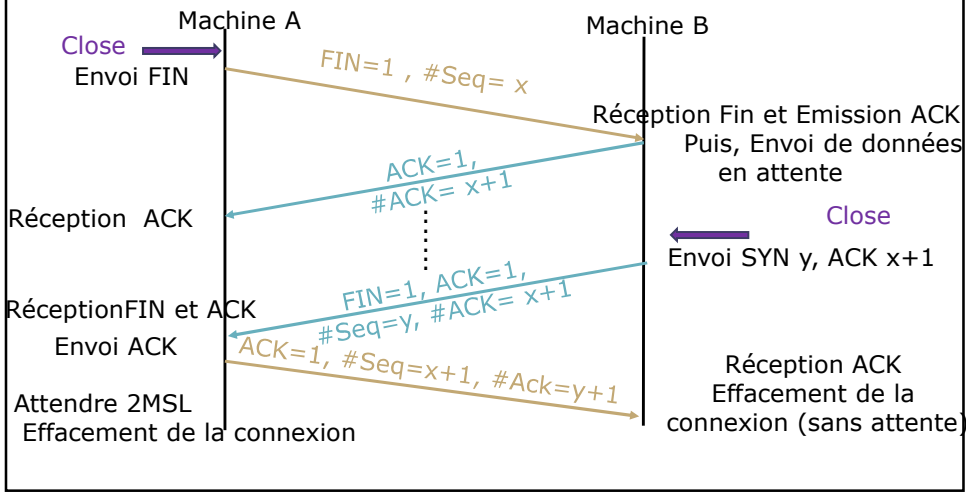


***ISN** : **I**nitial **S**equence **N**umber*

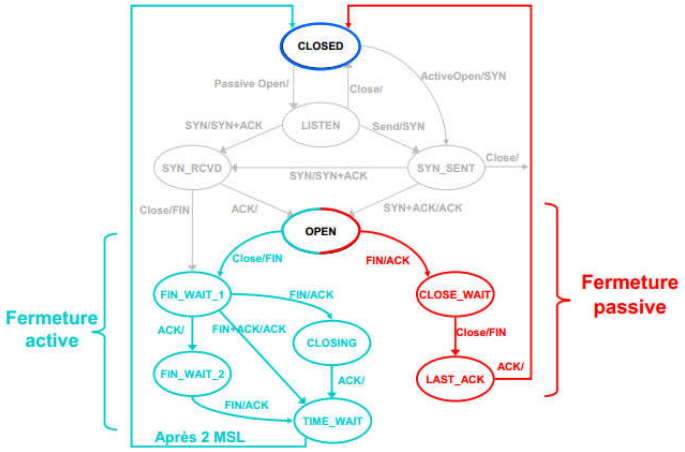
Ouverture d'une connexions



Ouverture et fermeture de connexion TCP



Fermeture d'une connexions



Émission des données et des acquittements

- Dans TCP, les seuls données pouvant être envoyées sont celles se situant dans la fenêtre comprise entre x et x + F avec
  - x les dernières données acquittées
  - F est la taille de la fenêtre TCP.

Émission des données et des acquittements

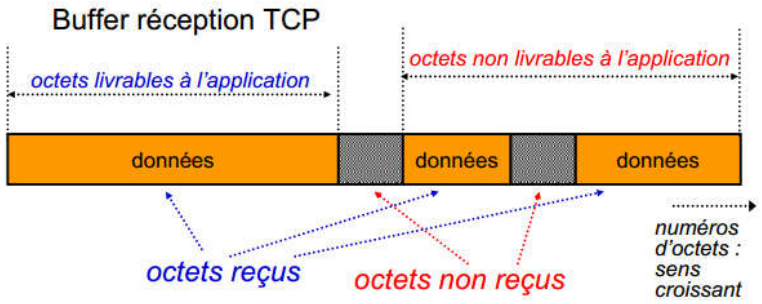
- Une fois la connexion établie, les numéros de séquences initiaux (ISN) ainsi que les tailles maximales de segment (MSS) définis, le transfert de données peut commencer.
- L'entité TCP peut prélever des octets venant de l'application pour composer des segments et les envoyer vers l'entité distante.

Émission des données et des acquittements

- Les segments sont formés de la façon suivante :
  - une chaîne d'octets (de taille inférieure à un MSS) est prélevée du buffer émission, le numéro du premier octet de la chaîne est déposé dans ce segment, le checksum est calculé, puis le segment ainsi formé est confié à IP pour routage sur le réseau,
  - lorsque l'on souhaite faire porter au segment de données un acquittement (piggy-backing), le drapeau ACK est mis à 1 et un numéro d'acquittement est déposé dans le segment avant la phase de calcul du checksum.

Émission des données et des acquittements

- Le récepteur ne délivre à l'application que les octets en séquence.



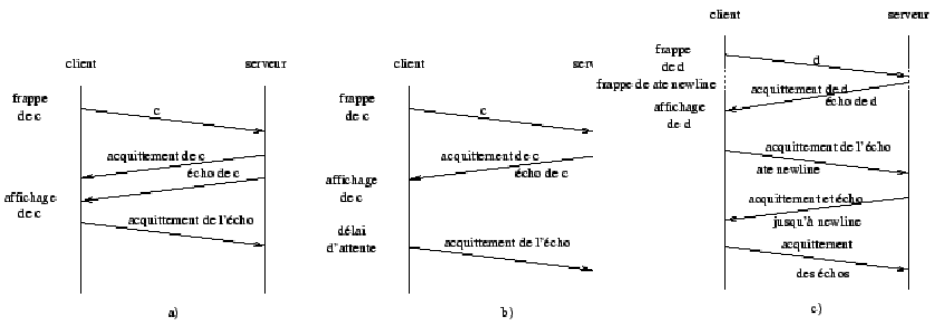
Émission des données et des acquittements

- Plusieurs règles réagissent l'envoi des acquittements:
  - un acquittement est retardé par défaut s'il n'y a pas de données à émettre
  - lorsque des données urgentes sont reçues, un acquittement est envoyé
  - lorsqu'un paquet hors séquence est reçu, un acquittement est envoyé
  - un acquittement n'est pas retardé de plus de 200 ms
  - dès que des données sont disponibles, l'acquittement est envoyé

Algorithme de Nagle

- Idée : il est pénalisant d'envoyer des segments contenant 1 seul octet de données (par ex. terminal virtuel)
- principe de Nagle : si l'appli. écrit octet par octet
  - envoi du premier octet dans un segment
  - accumulation dans le tampon des octets suivants tant que le premier octet n'est pas acquitté
  - envoi des octets accumulés dans un seul segment
  - attente de l'acquittement pour envoyer le segment suivant…
- Peut être désactivé dans certains cas (X-Window : Il faut envoyer les mouvements de souris de la souris de suite et non les tamponner. . .)

Algorithme de Nagle



Algorithme de Nagle

- L'activation de l'algorithme de Nagle empêche aux données envoyées d'être émises dès qu'elles sont produites, afin d'éviter la production de plusieurs petits paquets. Pour cela, un paquet n'est envoyé que dans les conditions suivantes :
  - il contient suffisamment de données pour remplir complètement un paquet TCP
  - le précédent paquet a été acquitté.

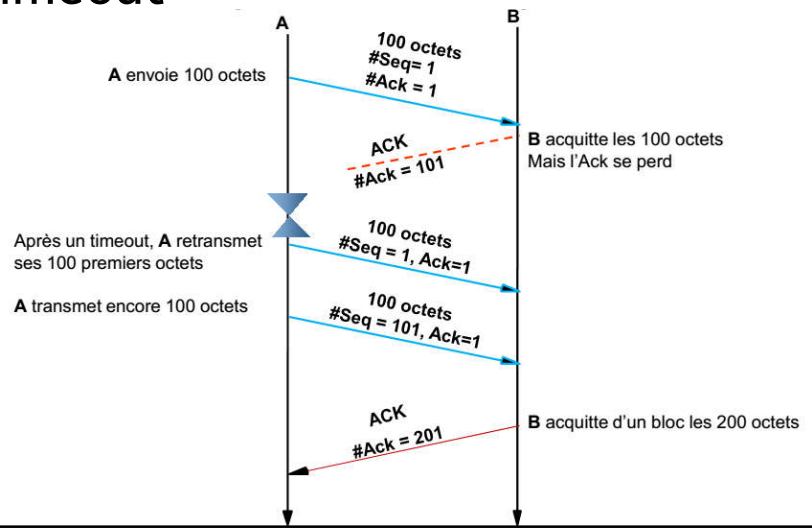
Gestion des erreurs

- Le protocole TCP dispose de plusieurs mécanismes pour gérer les erreurs :
  - Les paquets TCP disposent d'une somme de contrôle.
    - Les paquets reçus avec une erreur sont détruits.
  - Les paquets dupliqués sont détectés grâce au numéro de séquence.
  - Les paquets arrivant dans le désordre peuvent être réordonnés grâce au numéro de séquence.

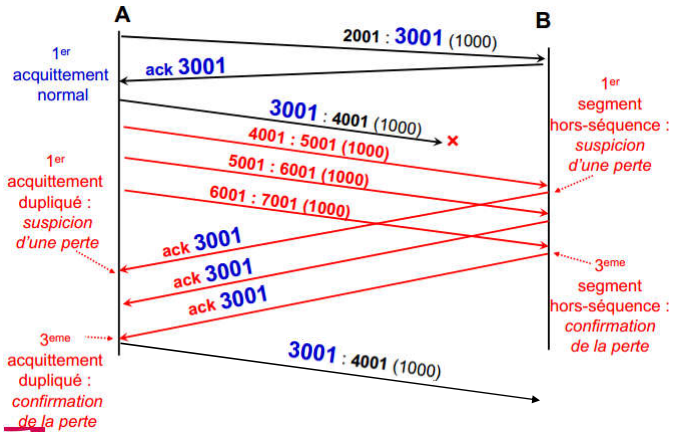
Gestion des erreurs

- Principe de base du contrôle d'erreur
  - Emission de segment suivi d'un armement de temporisateur (timer)
  - Utilisation de numéro de séquence en émission
  - On numérote les octets et non les segments
  - Si un Ack est reçu avant le déclenchement du timer : OK
  - Deux mécanismes de retransmission utilisés conjointement :
    - Retransmission sur Timeout
    - Retransmission rapide : Après la réception de trois Ack portant le même numéro d'acquittement inférieur à celui attendu,

Exemple de retransmission sur Timeout



Exemple de retransmission rapide



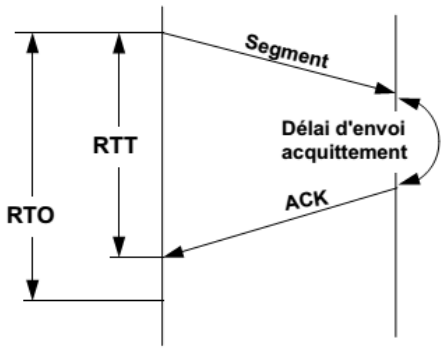
Détermination du RTO

- RTO (Retransmission Timeout)
  - à chaque envoi d'un paquet de données, une horloge propre est lancée
  - si l'horloge expire, le paquet est retransmis
- Impact de cette valeur sur les performances
  - valeur trop petite : des retransmissions inutiles ont lieu
  - valeur trop grande : attente trop importante entre deux retransmissions

Détermination du RTO

- RTT (Round Trip Time) : temps entre l'envoi d'un paquet et la réception de son accusé
- TCP fait une estimation du RTT (temps aller/retour) et ajuste dynamiquement la valeur du temporisateur (RTO) en fonction de cette estimation
  - exemple : RTO = 2*RTT

Détermination du RTO



Détermination du RTO

- Algorithme simplifié de Jacobson : un RTT lissé est d'abord calculé en effectuant une pondération entre la précédente estimation et la dernière mesure du RTT :
  $RTT_{lissé} = (1-\alpha)\ RTT_{t-1} + \alpha\ RTT_{mesuré}$
  - α : facteur de lissage compris entre 0 et 1 permettant de donner plus ou moins d'importance à la dernière mesure du RTT (une valeur de α proche de 1 sera utilisée dans un réseau peu stable avec des temps de transmission très variables)

Détermination du RTO

- Le RTO est ensuite calculé en multipliant le RTT lissé par un coefficient de variance de délai β ayant une valeur recommandée de 2 :
  $RTO = \beta\ RTT_{lissé}$
- Jacobson à montré par la suite qu'un calcul basé sur la déviation moyenne lissée D (approximation de la déviation standard) donnait de meilleurs résultats pour d'importantes variations du RTT :
  $D = (1-\beta)\ D + \beta\ |RTT_{mesuré} - RTT_{lissé}|$
  $RTO = RTT_{lissé} + 4D$

Problème avec l'algorithme de Jacobson

- Problème : Quand TCP retransmet un segment et puis il reçoit un acquittement : est-ce que cet Ack correspond au segment initial ou bien au segment retransmis ?
  - TCP n'a aucun moyen de distinguer les deux cas.
- Associer l'Ack au segment le plus ancien peut conduire à une surestimation du $RTT_{mesuré}$ si plusieurs tentatives de retransmissions ont été effectuées.
- Associer l'Ack au dernier segment émis peut conduire à une sous-estimation du $RTT_{mesuré}$.

Algorithme de Karn

- Une des solutions utilisées dans la pratique pour remédier au problème avec l'algorithme de Jacobson est connue sous le nom d'algorithme de Karn :
  - Ne pas mettre à jour la valeur de RTT en cas de retransmission
  - Mais augmenter RTO , selon la formule :

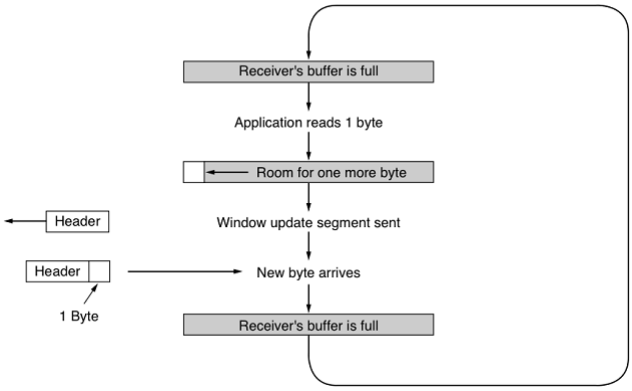
  RTO = g * RTO_dernier (généralement g vaut 2)

Syndrome de la fenêtre stupide

- Soit une application qui ne lit les données qu'octet par octet (ou en petite quantité)
- L'émetteur envoie suffisamment d'octets pour remplir le tampon de réception
- Le TCP récepteur envoie une taille de fenêtre nulle
- Lorsque l'application lit un octet, TCP enverra une taille de fenêtre de 1
- L'émetteur peut alors envoyer 1 octet !
- Puis taille de fenêtre 0
- etc.

➔ gaspillage de bande passante

Syndrome de la fenêtre stupide



Eviter la fenêtre stupide

- Côté récepteur : (Algorithme de Clark) n'annoncer une réouverture de la fenêtre que lorsque sa taille est .
  - soit égale à la moitié du tampon de réception
  - soit égale au MSS
- Côté émetteur : (Algorithme de Nagle) retarder le plus possible l'envoi
  - si des données n'ont pas été acquittées, placer les nouvelles données en tampon
  - lorsqu'un ACK arrive, envoyer ce qu'il y a (même si le segment n'est pas plein)

même en cas de PUSH !

Contrôle de flux et de congestion

- Contrôle de flux :
  - évite que les destinataires ne soit engorgés par des sources trop rapides
- Contrôle de congestion :
  - évite la transmission de segments sur le réseau alors qu'il est congestionné
  - optimise le taux d'utilisation des ressources du réseau
- un mécanisme de deux fenêtres est utilisé pour contraindre l'émission des segments

Contrôle de flux et de congestion

- Fenêtre (variables) :
  - fenêtre de congestion (cwnd) : représente le nombre d'octets pouvant être transmis sans risque de congestionner le réseau
  - fenêtre de flux (rwnd) : représente le nombre d'octets pouvant être transmis sans provoquer de débordement dans le buffer du récepteur.
  - snd.wnd = min [cwnd, rwnd]

Contrôle de flux dans TCP

- « Fenêtre glissante »
  - Idée : l'émetteur peut envoyer des données sans avoir reçu de confirmation de tout ce qu'il a déjà émis… tant que le récepteur peut absorber les nouvelles données !
- Piloté par le récepteur
  - Champs de l'en-tête TCP
    - Taille de la fenêtre (16 bits)
    - N° de ACK (32 bits)

Contrôle de flux dans TCP

- Le récepteur confirme l'arrivée de données ⇒ la fenêtre se ferme
  - Champ N° de ACK
- Le récepteur lit des données (acquittées) ⇒ la fenêtre s'ouvre
  - Champ Taille de la fenêtre
- Fermeture + ouverture = la fenêtre se déplace « vers la droite » (glisse)

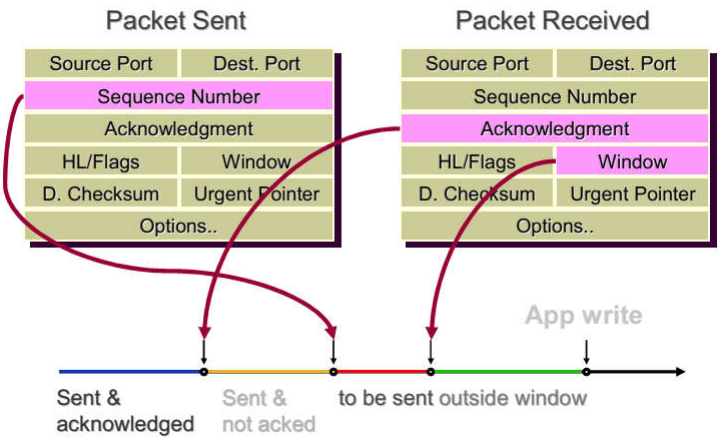
Contrôle de flux dans TCP

- Fenêtre glissante : évolution dans le temps

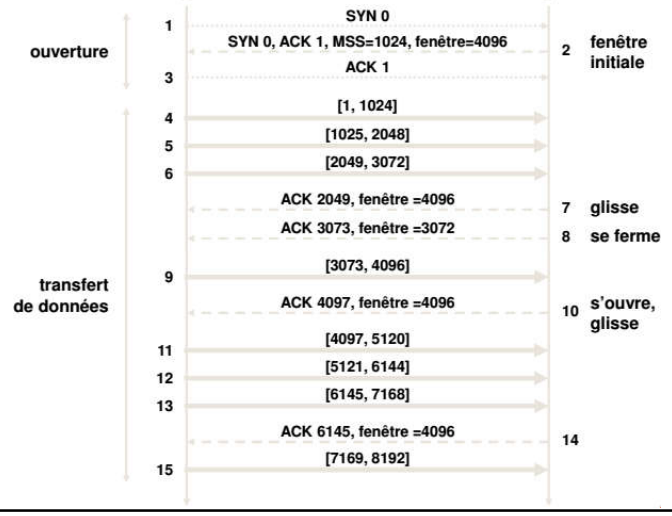
Principe de la fenêtre glissante

- Chaque TCP annonce le nombre d'octets qu'il est disposé à recevoir, à compter du n° dans le champ ACK

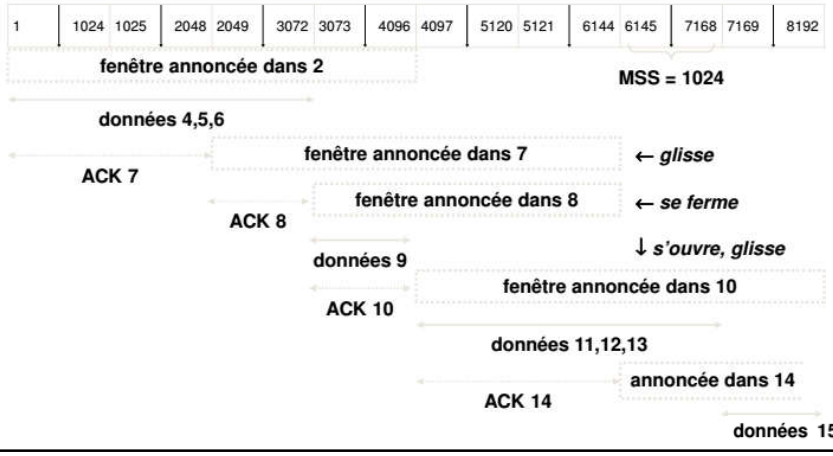
Contrôle de flux dans TCP



Fenêtre glissante : exemple



Fenêtre glissante : exemple



TCP : mécanismes de contrôle de congestion

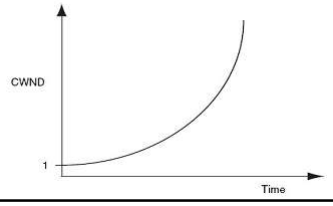
- En général, la perte d'un paquet est la seule information sur l'état du réseau
- Le contrôle de congestion est géré exclusivement par l'émetteur
  - le récepteur ne fait que renvoyer des accusés de réception
- Les algorithmes basiques de contrôle de congestion supportés par TCP sont [RFC 2581] :
  - Démarrage lent (slow start)
  - Évitement de congestion (congestion avoidance)
  - Retransmission rapide (fast retransmission)
  - Recouvrement rapide (fast recovery)

Démarrage lent (slow start)

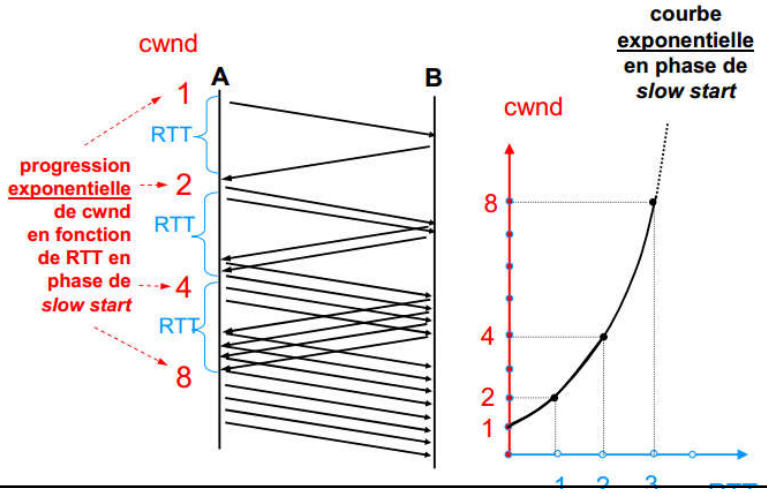
- Initialisation
  - Seuil de démarrage lent (ssthresh)
    - Sa valeur initiale est laissé à l'implémentation
  - cwnd = 1 MSS

Démarrage lent (slow start)

- But : retrouver rapidement la bande passante disponible
- cwnd++ à chaque non dup ack reçu (cwnd *= 2 à chaque RTT), RFC 2581
  - croissance exponentielle
- Si atteinte ssthresh :
  - on entre en congestion avoidance

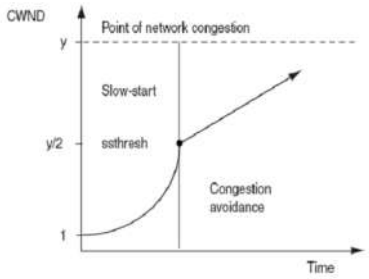


Démarrage lent : Exemple

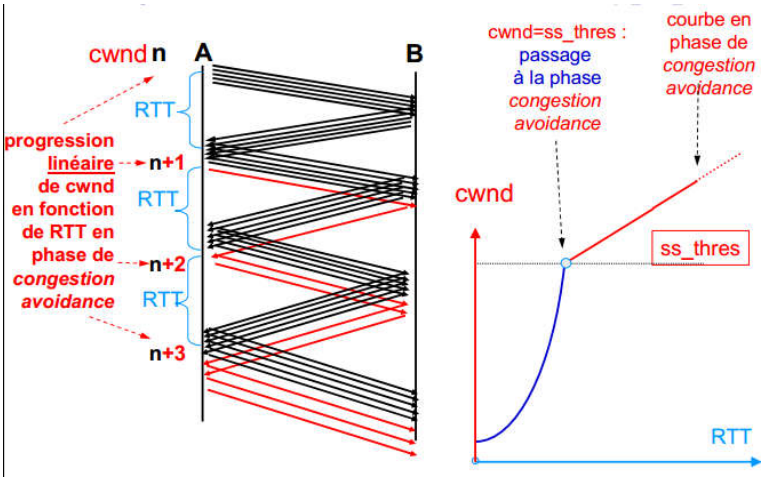


Évitement de congestion (congestion avoidance)

- Utilisé quand cwnd >= ssthresh
  - quand cwnd < ssthresh, c'est le slow start
- But : augmenter le débit en testant gentiment la bande passante disponible
- cwnd++ à chaque RTT
  - croissance linéaire



Congestion avoidance : Exemple



Slow start et Congestion avoidance)

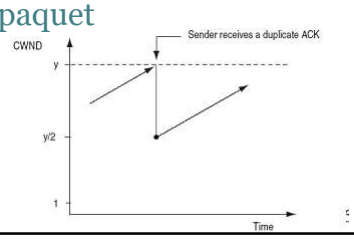
- Si perte par timeout (=> pertes successives) :
  - ssthresh = cwnd / 2
  - entre en slow start avec cwnd=1
- Si perte par 3 dupack (=> perte isolée) :
  - ssthresh = cwnd / 2
  - entre en fast retransmission

Retransmission rapide (fast retransmission)

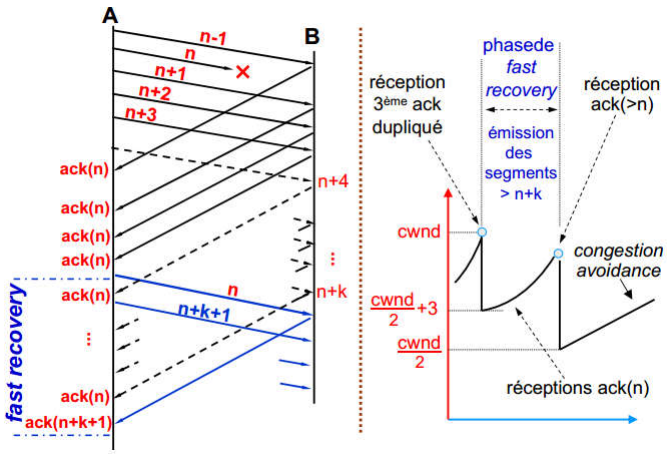
- But : détecter plus rapidement la perte d'un paquet (et le retransmettre)
- si N dup acks, on n'attend plus le timeout, mais :
  - on retransmet le paquet
  - on entre en slow start avec cwnd=1 (Tahoe), ou bien en fast recovery (les autres)

Recouvrement rapide (fast recovery)

- But : conserver une ouverture de fenêtre de congestion suffisante jusqu'à un non dup ack arrive.
- ssthresh = cwnd / 2
- cwnd = ssthresh + 3
  - => envoi éventuel de nouveaux paquets
- Pour chaque dup ack, cwnd++
  - => envoi éventuel d'un nouveau paquet
- Réception d'un non dup ack :
  - cwnd = ssthresh
  - retour au congestion avoidance



Fast recovery : Exemple



TCP : évolution des Contrôle de congestion

- 1974–1980 Protocoles TCP et UDP, les deux sans CC
- 1988 TCP Tahoe, 1er CC, slow start + cong. avoidance + fast ret.
- 1990 TCP Reno, Tahoe + fast recovery, se remet plus rapidement lors d'une perte
- 1994 TCP Vegas, basé sur l'historique du RTT (état des routeurs)
- 1999 TCP NewReno, TCP Reno + adaptation de frec, gère mieux plusieurs pertes
- Options (entre l'émetteur et le récepteur) :
  - 1992 Window scaling et timestamps, gère de grandes fenêtres de réception, resp. mesure les RTTs
  - 1996 SACK, DSACK, gère mieux les pertes en spécifiant exactement les paquets reçus
- 1994 ECN, les routeurs donnent des informations sur la congestion
- 1998–2002 cwnd initial augmente de 1 à 2–4 segments (~4 ko)
- 1999/2003 Algorithme ABC, modification d'un CC, des octets à la place de paquets
- 2000 TFRC, CC basé équation
- CCs réseaux sans fil :
  - 2001 TCP Westwood+, basé sur l'historique du RTT, meilleure utilisation si pertes aléatoires
- CCs pour grands cwnd (réseaux « rapides ») :
  - 2003 High-speed TCP, suivi de 2004 BIC, 2005 CUBIC, 2006 Compound TCP
- 2006 Protocole DCCP, entre UDP et TCP, sans retransmission, choix du CC
- 2011 bufferbloat, CoDel
- 2013 cwnd initial augmente de 2–4 à 10 segments
- 2013 Algorithme PRR, se remet plus rapidement lors d'une perte dans les flux courts (Web)
- 2013 Data Center TCP
- 2017 BBR (Bottleneck Bandwidth and Round-trip propagation time)

Variantes TCP