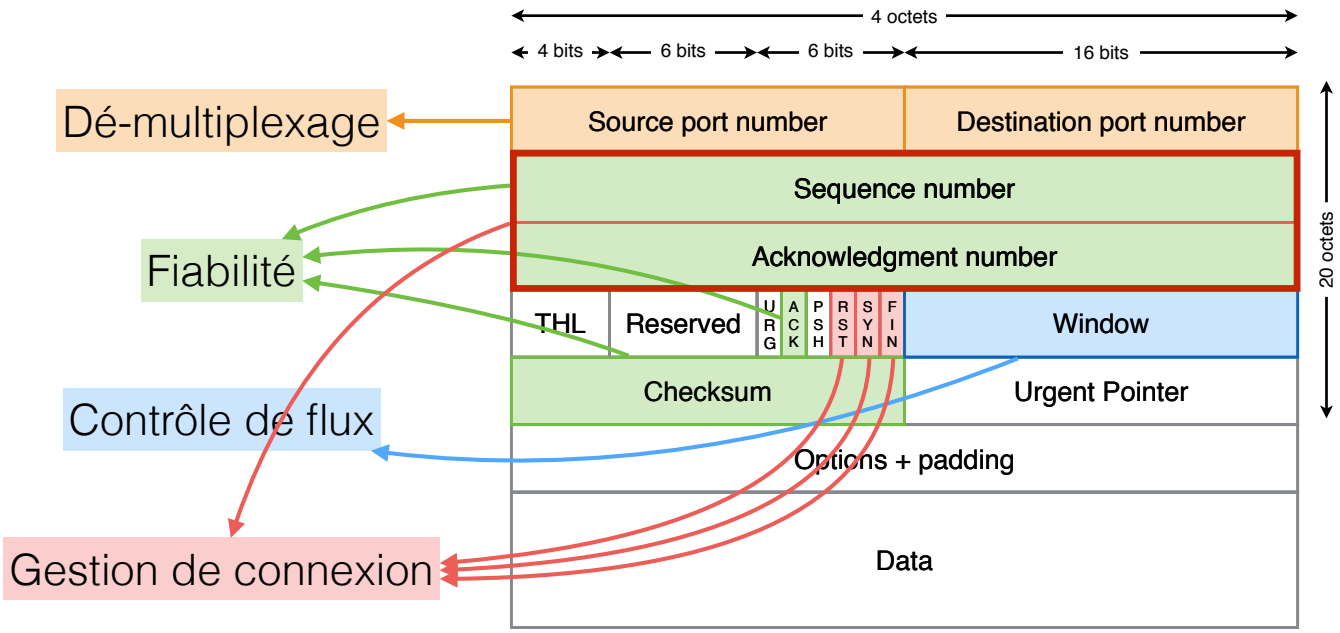
La couche transport dans Internet (cont')

UE LU3IN033 Réseaux
2021-2022

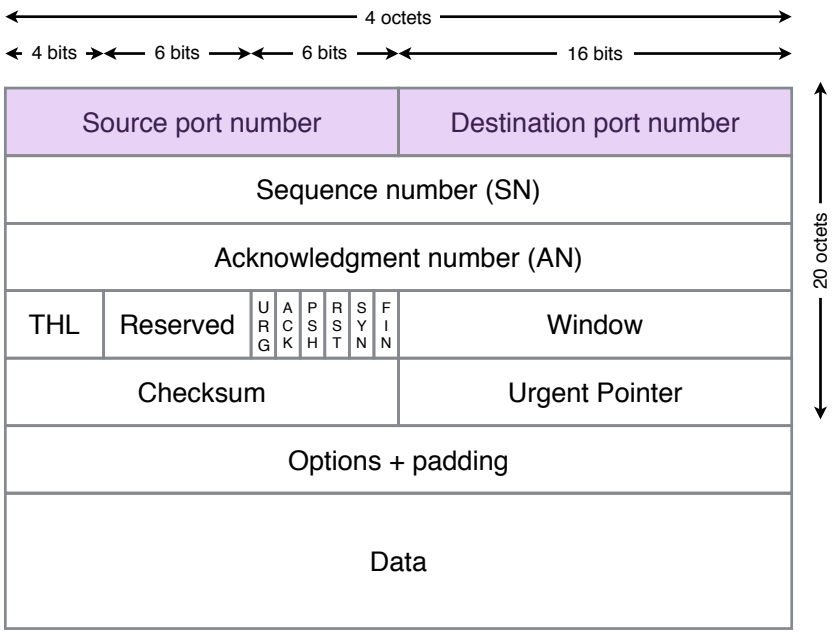
Prométhée Spathis
promethee.spathis@sorbonne-universite.fr

Entête TCP



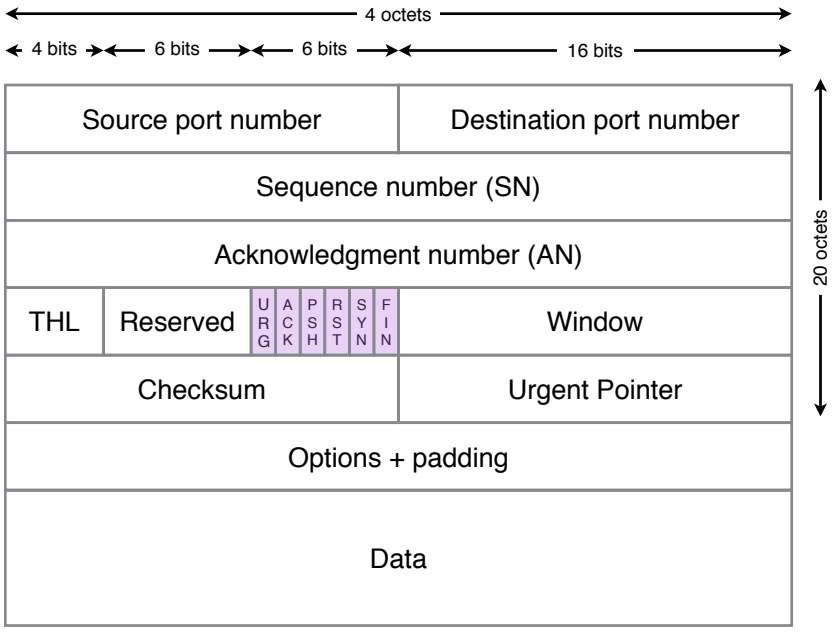
Numéros de port

- Numéro de port source
  - identifie le processus qui émet le segment
- Numéro de port destination
  - identifie le processus à qui est destiné le segment
- Numéro de port client
  - valeur arbitraire
  - laissé au choix de l'OS
- Numéro de port serveur
  - selon le type de service
    - 80 : Serveur Web
    - …



Drapeaux TCP

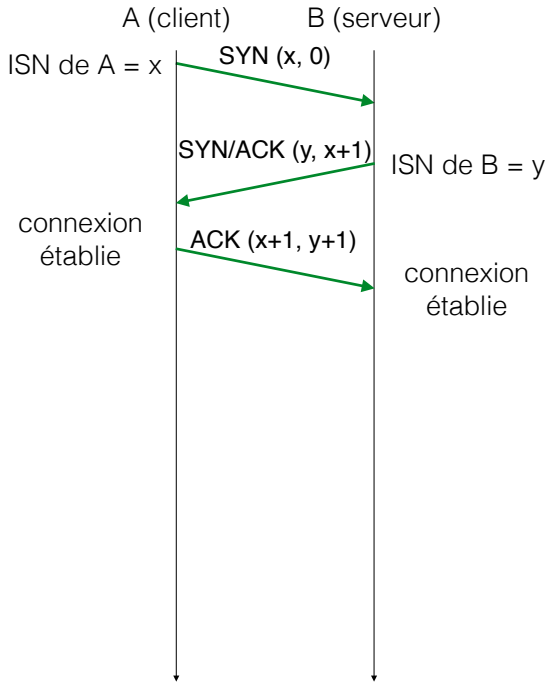
- SYN, FIN, RST, et ACK
  - identifient 4 types de segment
  - les segments de données n'ont pas de drapeau dédié
- SYN
  - ouverture de connexion
- FIN
  - fermeture de connexion
- RST
  - réinitialisation de la connexion
- ACK
  - le champ AN est valide
- PSH
  - les données doivent être lues au plus vite par l'application côté récepteur
- URG
  - nombre d'octets urgents



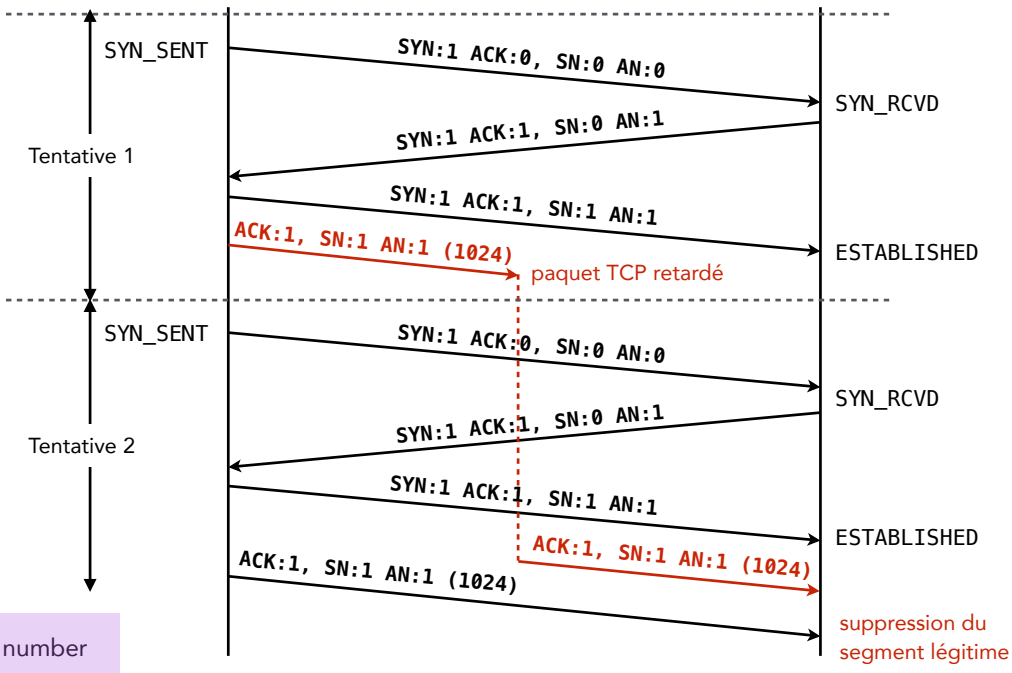
Types de segments TCP: SYN et FIN

Drapeaux SYN et ACK

- Segment SYN
  - Ouverture bilatérale d'une connexion
    - le client envoie un premier SYN auquel le serveur répond par un segment SYN-ACK
    - la connexion TCP est établie une fois le SYN du serveur acquitté par le client
  - Synchronisation des numéros de séquence
    - le champ Sequence Number contient la valeur de l'ISN Initial sequence number
    - choisie aléatoirement
  - Paramétrage de la connexion (options TCP de l'entête des SYN)
- Segments ACK
  - accusent la bonne réception des SYN :
    - en incrémentant le SN du SYN



Initial Sequence Number

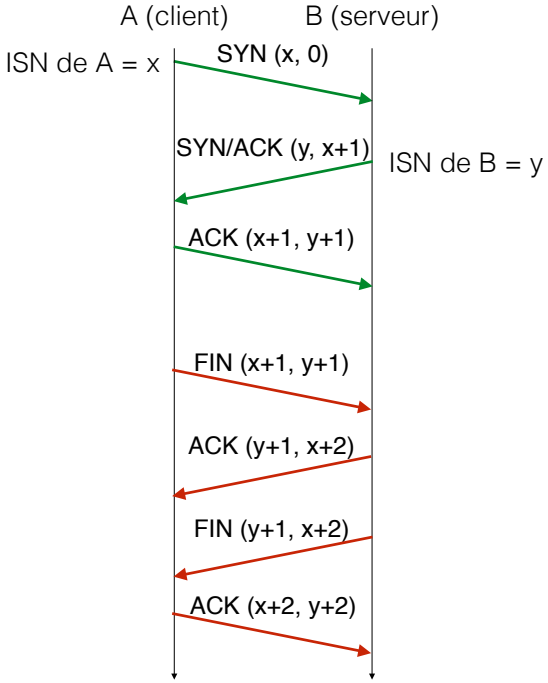


SN : sequence number
AN : ACK number

Types de segments TCP: FIN et ACK

Drapeaux FIN et ACK

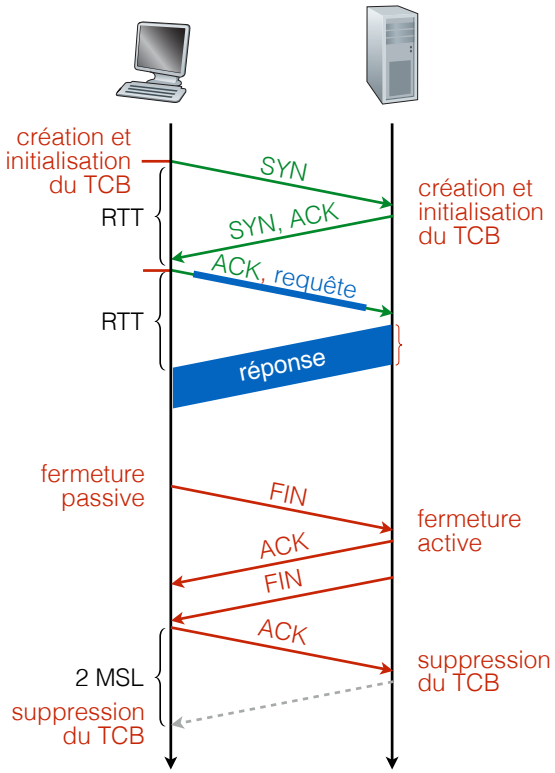
- Segment FIN
  - Fermeture bilatérale de connexion
    - serveur et client envoient un segment FIN
    - la connexion est fermée une fois les deux FIN acquittés
- Segments ACK
  - accusent la bonne réception des FIN :
    - en incrémentant le SN du FIN



Connexion TCP

- Connexion TCP
  - ouverture : poignée de main à 3 voies
    - segments SYN, SYN/ACK, ACK
  - fermetures : poignée de main à 4 voies
    - segments FIN, ACK
- Création et initialisation des TCB côté client et serveur
  - Les TCB Transport Control Block contiennent les informations d'état caractérisant l'échange
    - les identifiants du processus (adresse IP, numéro de port)
    - numéros de séquence des octets reçus en séquence et acquittés
    - les valeurs des fenêtres
- Les TCB sont supprimés après réception des FIN segments (four-way handshake)

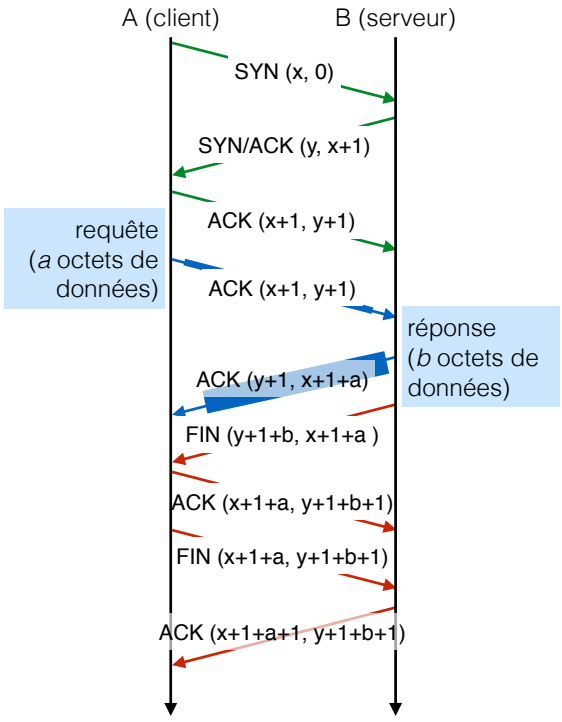
Une connexion TCP est la combinaison des TCB client et serveur et les informations d'état qu'ils contiennent



Types de segments TCP: ACK et données

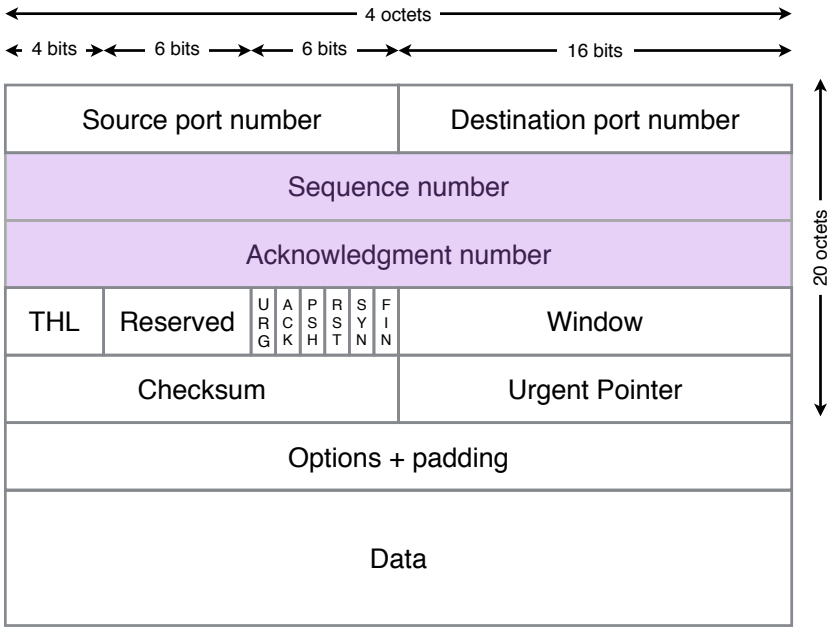
Drapeau ACK

- Segments ACK
  - accusent la bonne réception :
    - des SYN et des FIN
    - des octets de données correctement reçus
  - éventuellement "à cheval sur" (piggyback)
    - des segments SYN, FIN
    - des segments de données
- Segments de données
  - pas de drapeau explicite
  - drapeaux éventuellement positionnés :
    - ACK, PSH, et/ou URG



Numérotation des segments (1)

- Numéro de séquence
  - signification différente selon le type de segment
- Numéro d'acquittement
  - accuse la réception d'un segment en incrémentant la valeur du NS :
    - de 1 si segment SYN, FIN ou RST
    - du nombre d'octets reçus si segment de données



Numérotation des segments (2)

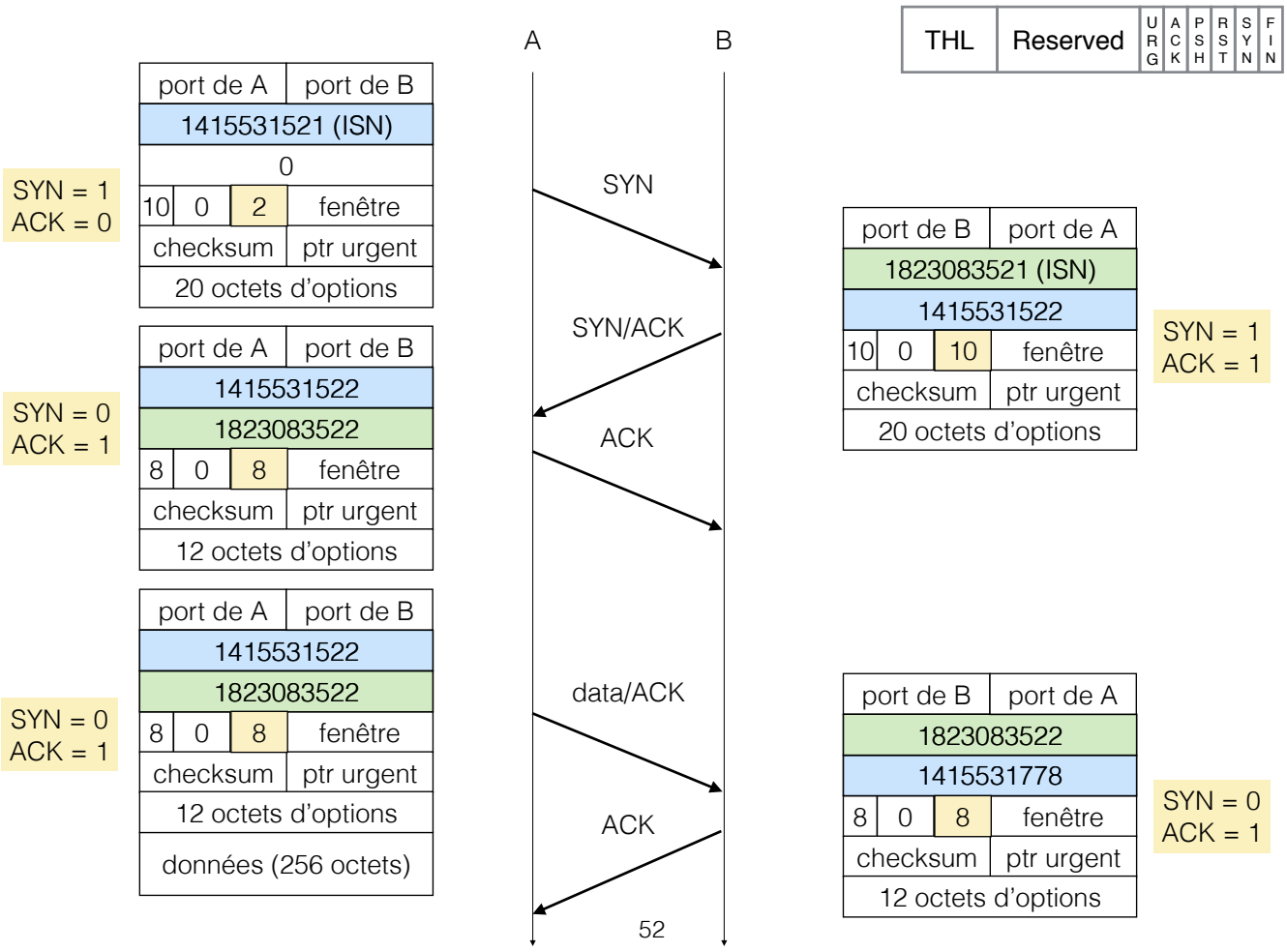
Champ Sequence Number

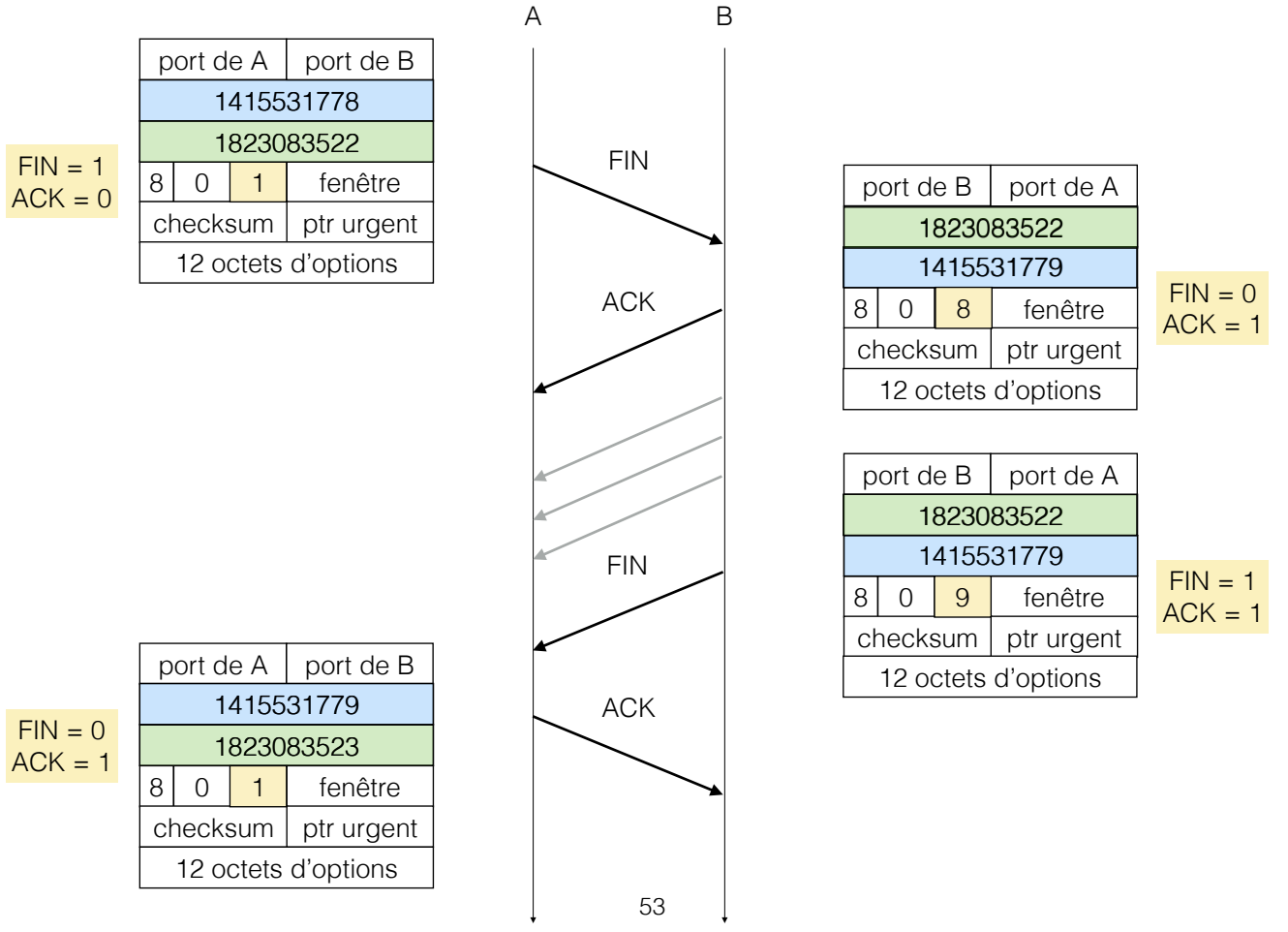
- TCP numérote les octets de données en séquence
  - la valeur initiale est choisie aléatoirement
- Segments SYN
  - le champ Sequence Number contient la valeur du numéro de séquence initiale (ISN)
    - Un SYN est acquitté par un segment ACK dont le AN incrémente la valeur de l'ISN
- Segments de données
  - le champ Sequence Number contient le numéro de séquence du premier octet de données transporté
- Segments FIN
  - la valeur du champ Sequence Number d'un segment FIN permet d'identifier le segment ACK retourné pour accuser la bonne réception du segment FIN
    - Un FIN est acquitté par un segment ACK dont l'AN incrémente le SN du segment FIN

Numérotation des segments (3)

Champ Acknowledgment Number

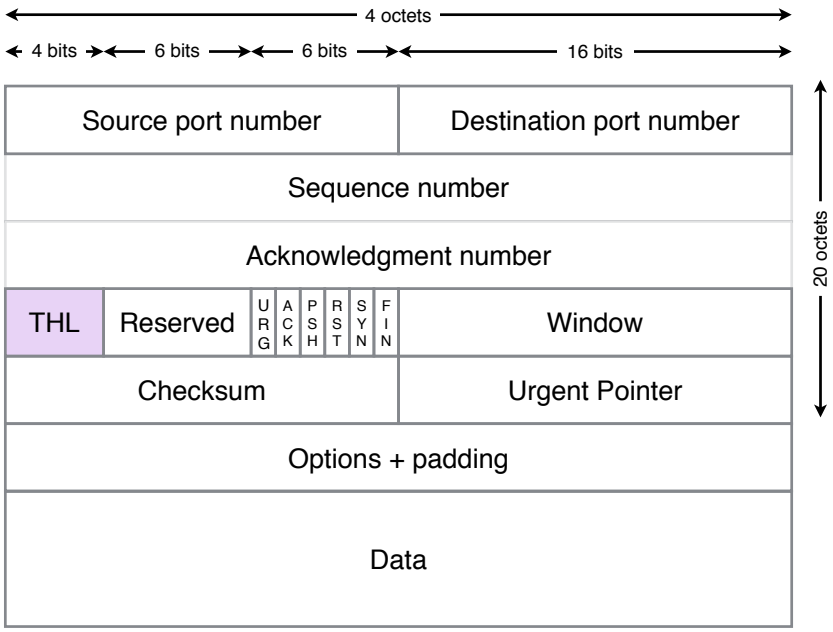
- Un segment TCP accuse les segments reçus si le drapeau ACK est positionné
- La valeur du Acknowledgment Number est calculée en incrémentant la valeur du SN du segment reçu :
  - de 1, si le segment reçu est un SYN, un FIN ou un RST
  - du nombre d'octets de données correctement reçus, si le segment reçu est un segment de données
- Les ACK sont éventuellement piggybackés
  - aux segments SYN, FIN et aux segments de données
- Le SN d'un segment ACK vide non piggybacké n'est pas pertinent
  - sa valeur reprend la valeur du prochain SN attendu le récepteur sans le consommer
  - le segment de données suivant reprend cette valeur pour son SN


THL
Reserved
URG
ACK
PSH
RST
SYN
FIN
A
B
SYN
SYN/ACK
ACK
data/ACK
ACK
port de A
port de B
1415531521 (ISN)
0
10
0
2
fenêtre
checksum
ptr urgent
20 octets d'options
SYN = 1
ACK = 0
port de B
port de A
1823083521 (ISN)
1415531522
10
0
10
fenêtre
checksum
ptr urgent
20 octets d'options
SYN = 1
ACK = 1
port de A
port de B
1415531522
1823083522
8
0
8
fenêtre
checksum
ptr urgent
12 octets d'options
SYN = 0
ACK = 1
port de B
port de A
1823083522
1415531778
8
0
8
fenêtre
checksum
ptr urgent
12 octets d'options
SYN = 0
ACK = 1
port de A
port de B
1415531522
1823083522
8
0
8
fenêtre
checksum
ptr urgent
12 octets d'options
données (256 octets)
SYN = 0
ACK = 1
52


A
B
FIN
ACK
FIN
ACK
port de A
port de B
1415531778
1823083522
8
0
1
fenêtre
checksum
ptr urgent
12 octets d'options
FIN = 1
ACK = 0
port de B
port de A
1823083522
1415531779
8
0
8
fenêtre
checksum
ptr urgent
12 octets d'options
FIN = 0
ACK = 1
port de B
port de A
1823083522
1415531779
8
0
9
fenêtre
checksum
ptr urgent
12 octets d'options
FIN = 1
ACK = 1
port de A
port de B
1415531779
1823083523
8
0
1
fenêtre
checksum
ptr urgent
12 octets d'options
FIN = 0
ACK = 1
53

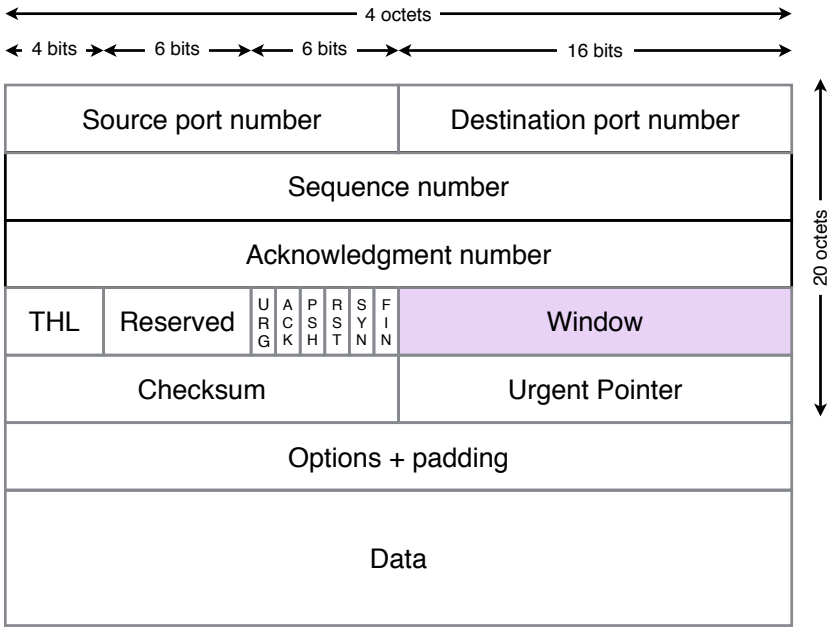
Champ THL Transport Header Length

- THL
  - taille de l'entête exprimée en mots de 32 bits (4 octets)
- Taille fixe
  - THL = 0x5 (0101) : 5
    - 20 octets
    - entête sans option
- Taille max
  - THL = 0xF (1111) : 15
    - taille totale 60 octets
    - options TCP 40 octets



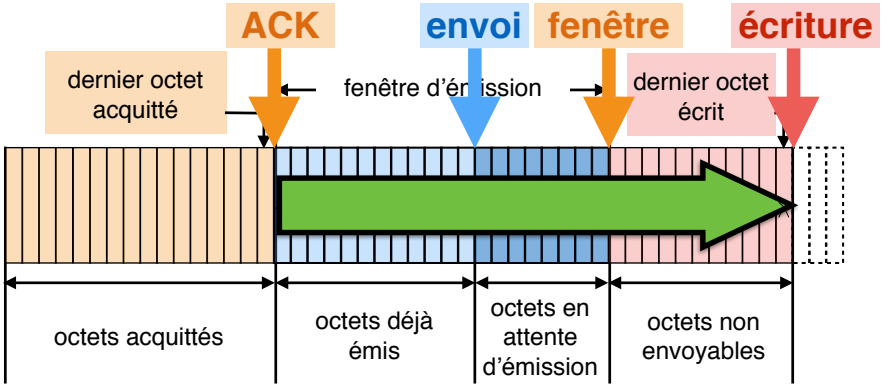
Champ Window

- Window
  - utilisé pour le contrôle de flux
  - indique la taille des tampons libres en réception de l'émetteur du segment



Contrôle de flux

- Le contrôle de flux évite les pertes découlant de l'engorgement des récepteurs
  - Un récepteur TCP stocke les octets reçus dans un tampon mémoire …
  - … le temps que l'application vienne les lire
  - L'absence de tampon en réception provoque la suppression des octets en excès
- Mécanisme de fenêtrage
  - La quantité d'octets que peut envoyer un émetteur est déterminée par sa fenêtre d'émission
  - La taille de la fenêtre d'émission est déterminée :
    - par la valeur du champ window contenu dans l'entête des segments que retourne le récepteur
    - cette valeur représente la quantité de ses tampons libres en réception



Champ Checksum

paquet IP

4	IHL	TOS	Total Length
Identifier		R DF MF	Fragment offset
TTL	Protocol	Header checksum	
Source IP address			
Destination IP address			

pseudo entête

Source IP address		
Destination IP address		
0	Protocol	TCP segment total length

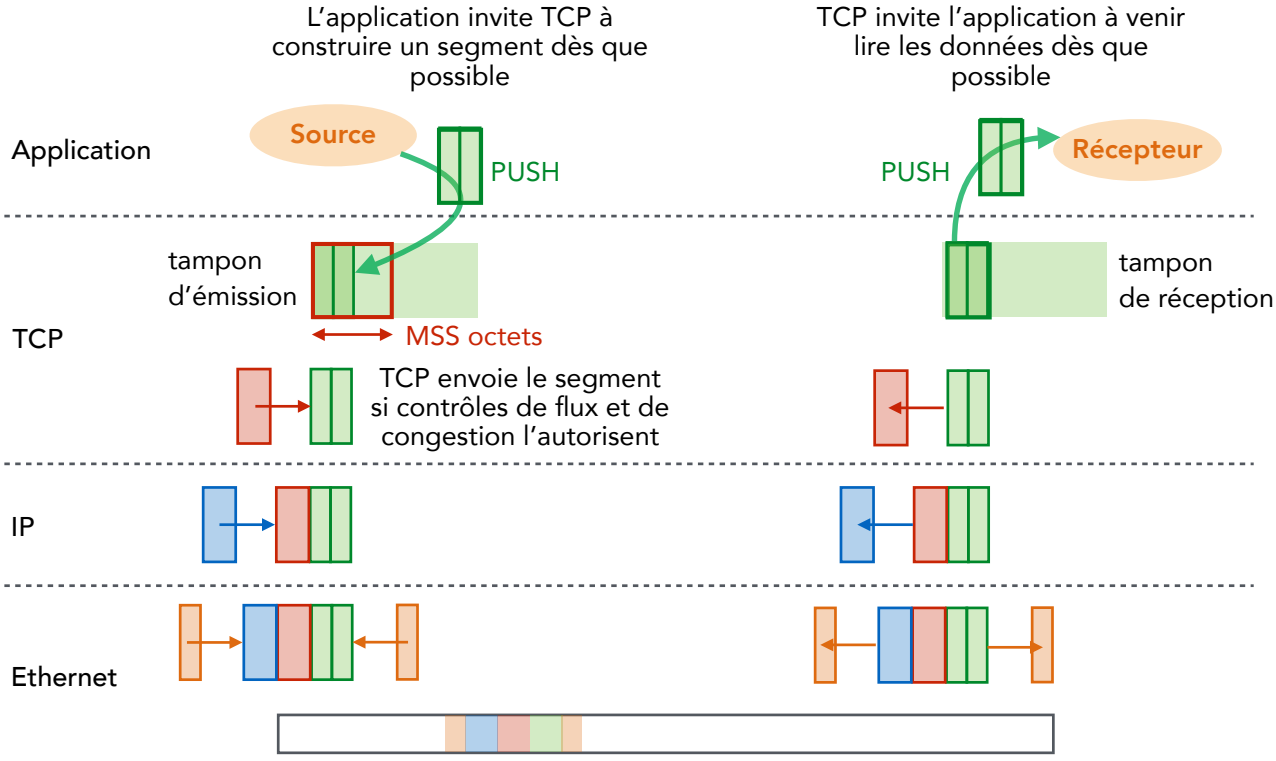
segment TCP

Source port number								Destination port number
Sequence number								
Acknowledgment number								
THL	Reserved	URG	ACK	PSH	RST	SYN	FIN	Window
Checksum								Urgent Pointer
Options + padding								
Data								

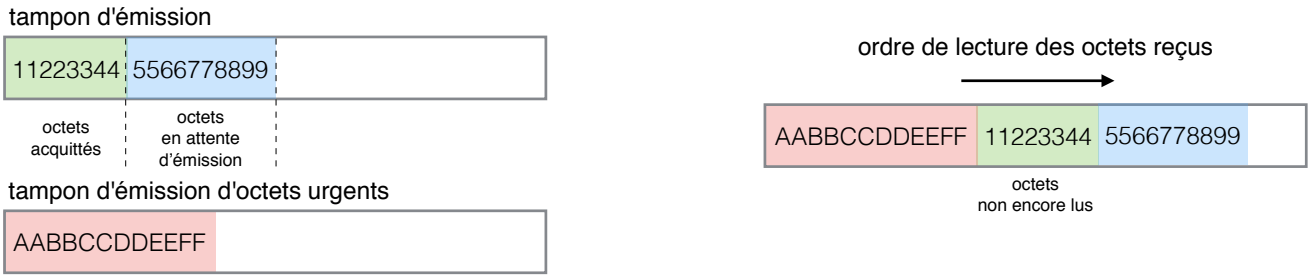
Drapeaux PUSH et URG

- Drapeau PUSH
  - côté émetteur :
    - l'application invite TCP à construire un segment sans attendre d'octets supplémentaires
  - côté récepteur :
    - l'application est invitée à lire les octets reçus dès que possible
- Drapeau URG et champ Urgent pointer (16 bits)
  - nombre d'octets urgents transportés dans la charge utile du segment (et les suivants)
  - ces octets sont lus en priorité par l'application réceptrice …
  - … sans attendre que les octets précédemment reçus soient préalablement lus

Drapeau PUSH

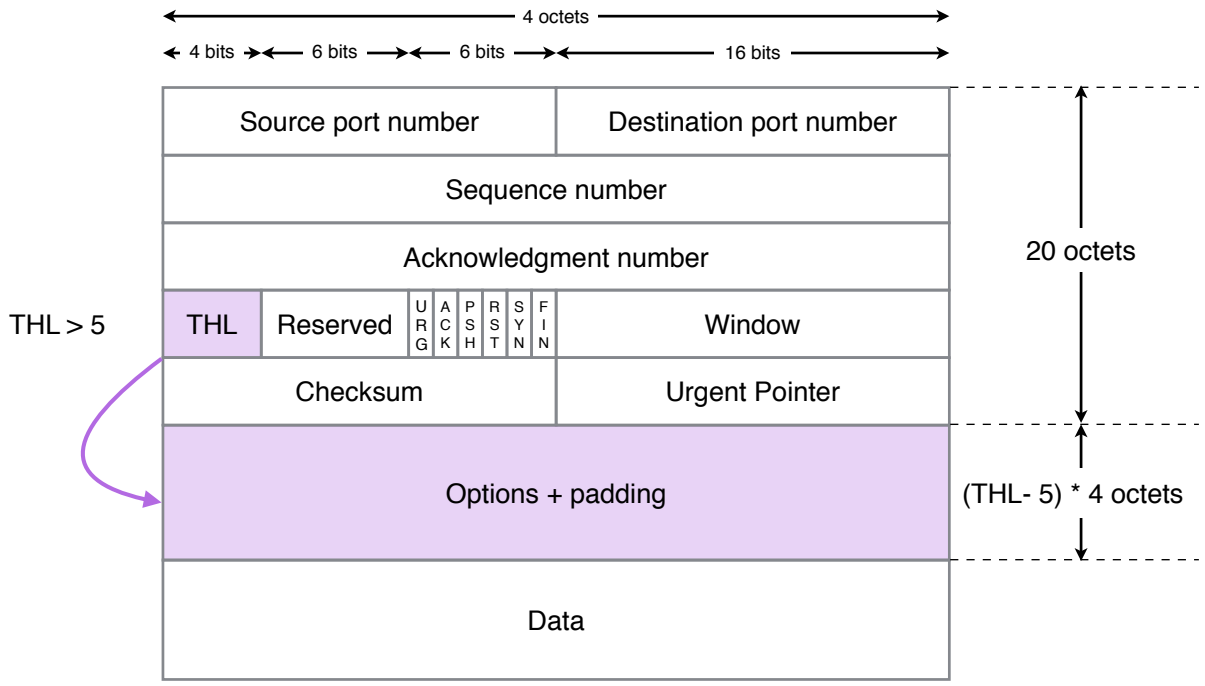


Drapeau URG et champ Urgent Pointer



Source port number							Destination port number
Sequence number							
Acknowledgment number							
THL	Reserved	URG	ACK	PSH	RST	SYN	FIN	Window
Checksum							Urgent Pointer: 0x0005
A	A	B	B	C	C	D	D
E	E	F	F	5	5	6	6
7	7	8	8	9	9		

Options TCP (1)



TCP Options (2)

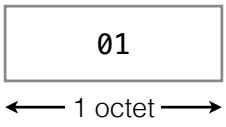
Kind	Length	Meaning	Reference
0x00	–	End of Option List	RFC 793
0x01	–	No-Operation	RFC 793
0x02	0x04	Maximum Segment Size	RFC 793
0x03	0x03	WSOPT - Window Scale	RFC 1323
0x04	0x02	SACK Permitted	RFC 2018
0x05	N	SACK (Selective ACK)	RFC 2018
0x06	0x06	Echo (obsoleted by option 8)	RFC 1072
0x07	0x06	Echo Reply (obsoleted by option 8)	RFC 1072
0x08	0x0A	TSOPT - Time Stamp Option	RFC 1323
0x09	0x02	Partial Order Connection Permitted	RFC 1693
0x0A	0x03	Partial Order Service Profile	RFC 1693
0x0B	–	CC	RFC 1644
0x0C	–	CC.NEW	RFC 1644
0x0D	–	CC.ECHO	RFC 1644
0x0E	0x03	TCP Alternate Checksum Request	RFC 1146
0x0F	N	TCP Alternate Checksum Data	RFC 1146

http://www.iana.org/assignments/tcp-parameters

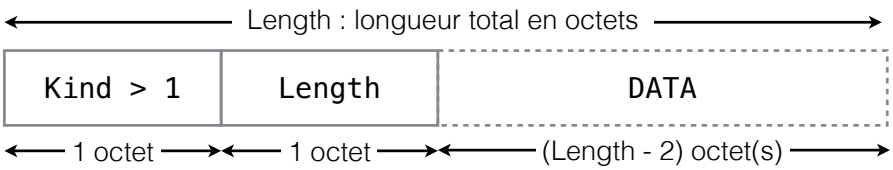
Options TCP (3)

Format de l'option EOL End of Options List (kind = 0)

Format de l'option NOP No OPeration (kind = 1)

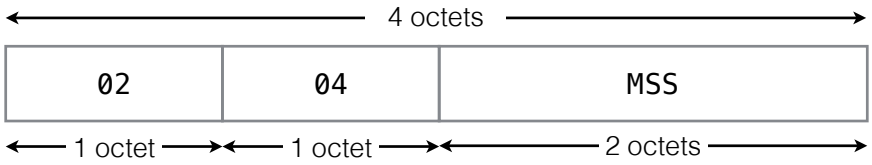


Format des options de type kind > 1

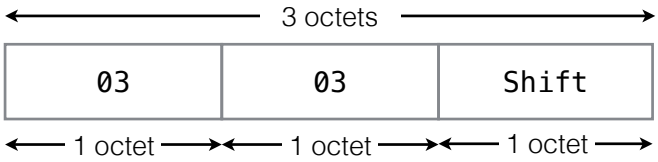


Options TCP (4)

Maximum Segment Size (MSS)

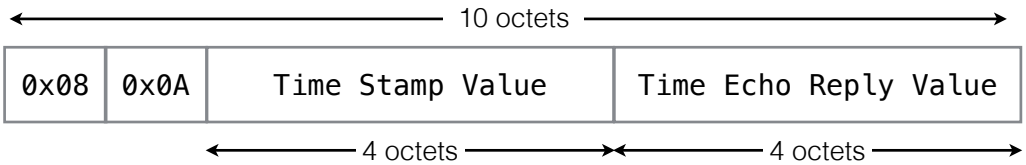


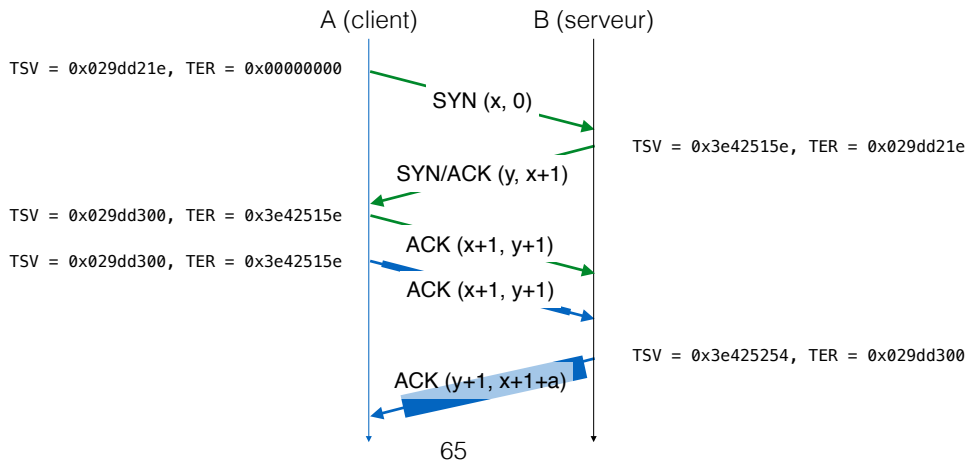
Windows Scale WSopt



Options TCP (5)

Time Stamp (TS)





```
ff ad 13 89 5c 3e 06 6a
00 00 00 00 a0 02 40 00
9c 2e 00 00 02 04 05 a0
01 03 03 00 01 01 08 0a
00 9e 7b c4 00 00 00 00
```

Port source : 0xFFAD, Port destination : 0X1389

Numéro de séquence : 0x5C3E066A

Numéro d'acquittement : 0x00000000

THL : 0xA (40 octets), SYN : 1, Fenêtre : 0x4000

Somme de contrôle : 0x9C2E, Pointeur urgent : 0x0000

Option 1 : Type : 0x02 (MSS), MSS : 0x05A0 (1440)
Option 2 : 0x01 (NOP),
Option 3 : Type : 0x03 (WScale), Décalage : 0x00
Option 4 : Type : 0x01 (NOP)
Option 5 : Type : 0x01 (NOP)
Option 6 : Type : 0x08 (Time Stamp), TSV : 0x9E7BC4, TERV : 0x000000

0xFFAD		0x1389	
0x5C3E066A			
0x00000000			
0xA002		0x4000	
0x9C2E		0x0000	
0x02	0x04	0x05A0	
0x01	0x03	0x03	0x00
0x01	0x01	0x08	0x0A
0x009E		0x7BC4	
0x0000		0x0000	

Efficacité des transmissions (1)

Génération des segments de données

- Objectif : émission de trames pleines (MTU octets de données)
  - Mise en mémoire des écritures (retard à l'émission)
- Si MSS (Maximum Segment Size) octets de données à envoyer
  - envoi de segments pleins
- Sinon :
  - demande explicite d'émission de l'application
    - fonction push
  - sur expiration d'un temporisateur
    - pour éviter d'attendre trop longtemps MSS octets
- Découplage écriture - émission des octets de données
  - une écriture d'octets : plusieurs envois de segments de données
  - plusieurs écritures d'octets : envoi d'un segment de données

Efficacité des transmissions (2)

Génération des segments ACK

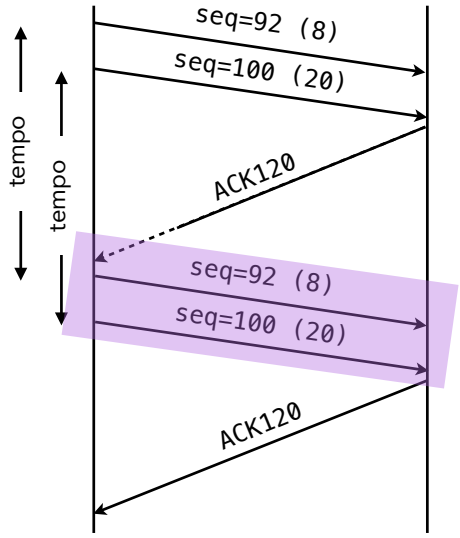
- "Dumb receiver"
  - un récepteur accuse le dernier des octets de données réçus en séquence
    - en indiquant le numéro du prochain octet attendu
  - la détection des pertes et leur réparation est à la charge de l'émetteur
    - pas de NAK dans TCP
- Acquittements retardés ou émis immédiatement :
  - ACK retardés : après un délai maximum d'attente
  - ACK cumulatifs : après réception de 2 MSS
  - ACK immédiat : sur réception d'un segment hors séquence ou corrigeant une perte

Génération des segments ACK

Evènement	Action du récepteur
Réception d'un segment en séquence : • le numéro de séquence du segment correspond au numéro de séquence attendu • les octets précédents ont déjà été acquittés	Attendre 500ms : • Si réception d'un second segment, alors envoi immédiat d'un **ACK cumulatif** • Sinon envoi d'un **ACK retardé** après 500ms
Réception d'un segment hors séquence : • un trou de séquence est détecté • le numéro de séquence du segment est supérieur au numéro de séquence attendu	Envoi immédiat d'un **ACK dupliqué** : • **l**e numéro d'acquittement indique à nouveau le numéro de séquence de l'octet attendu
Réception d'un segment qui comble le début d'un trou de séquence • partiellement ou complètement	Envoi immédiat d'un ACK : • qui accuse les octets reçus en séquence après le trou de séquence si le segment comble **complètement** le trou de séquence • qui accuse les octets de ce segment **sinon**

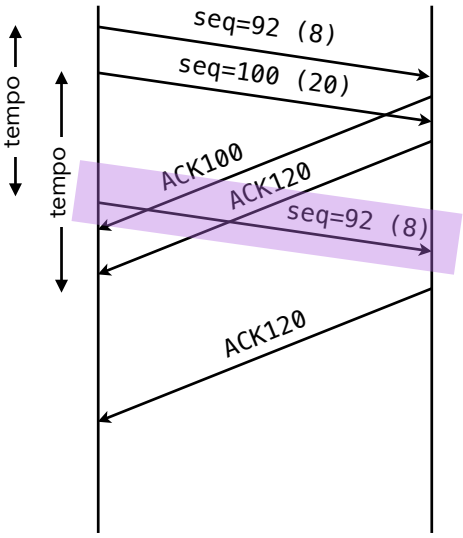
Quand retransmettre ? (2)

Perte de ACK



paquets TCP dupliqués

Expiration de temporisateur prématurée



paquet TCP et ACK dupliqués

Durée du temporisateur de retransmission

- La source arme un temporisateur dans l'attente d'un ACK
  - trop court : retransmissions inutiles
  - trop long : attente trop longue avant réparation d'une perte
- TCP calcule la durée du temporisateur en fonction du délai aller-retour (RTT)
  - les émetteurs mesurent la durée qui s'écoule entre l'envoi d'un segment de données
  - et la réception de l'ACK correspondant
    - Valeur moyennée du RTT
  - la source mesure RTT : le temps qu'a mis le dernier ACK à lui revenir
  - la source calcule SRTT : la valeur moyenne pondérée des RTT mesurés

$$SRTT = a * SRTT + (1 - a) * RTT$$

où a est un facteur de lissage

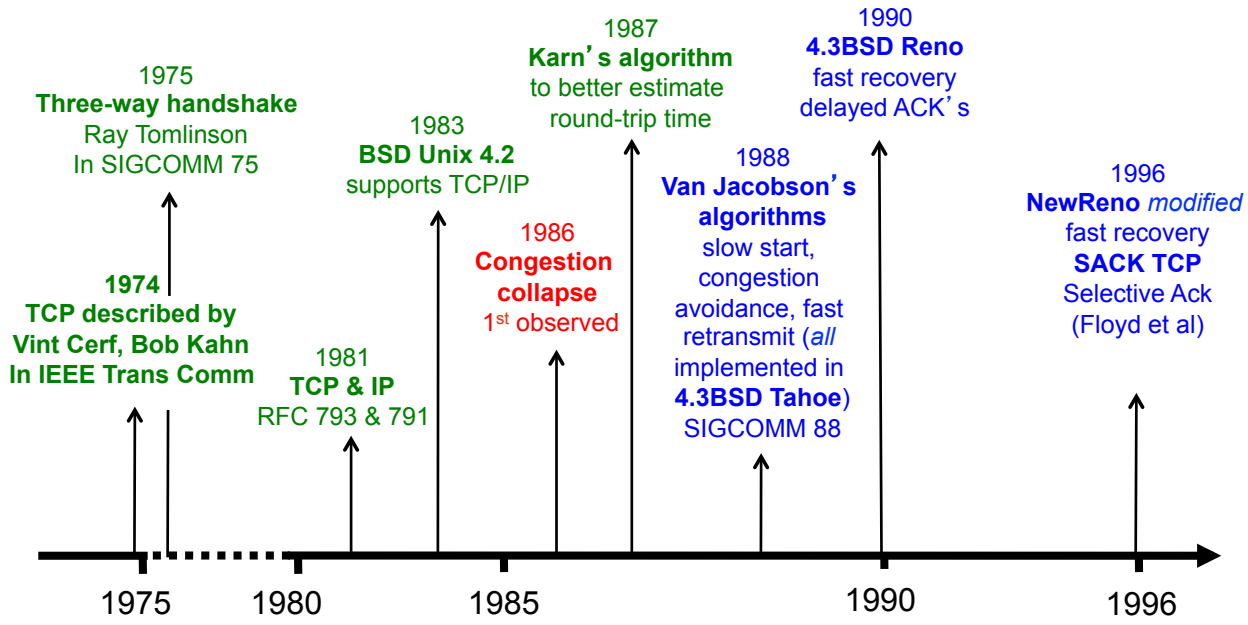
- Valeur du temporisateur de retransmission :

$$RTO = 2 * SRTT$$

  - valeur initiale du RTO = 1 seconde

TCP Congestion Control

Evolution of TCP



Notations

- `RECEIVER MAXIMUM SEGMENT SIZE (RMSS)`
  - Size of the largest segment the receiver is willing to accept
  - Value sent in the MSS option by the receiver during the 3-WHS
- `SENDER MAXIMUM SEGMENT SIZE (SMSS)`
  - Size of the largest segment that the sender can transmit
  - Based on the MTU of the network, or the path MTU discovery, or the RMSS, or …
- `RECEIVER WINDOW (rwnd)`
  - Most recently advertised receiver window
- `CONGESTION WINDOW (cwnd)`
  - TCP state variable that limits the amount of data a TCP can send
  - Data with sequence number lower than the sum of the highest acknowledged sequence number and the *minimum* of cwnd and rwnd can be sent
- `FLIGHT SIZE`
  - Amount of data that has been sent but not yet cumulatively acknowledged

TCP Algorithms

- Assumption:
  - a loss always results of network congestion
- *Slow start* and *congestion avoidance* algorithms
  - control the amount of outstanding data injected into the network
- TCP per-connection state variables
  - **`cwnd`**: sender-side limit on the amount of in-flight data the sender can transmit into the network before receiving an ACK
  - **`rwnd`**: receiver-side limit on the amount of outstanding data
    - The *minimum* of `cwnd` and `rwnd` governs data transmission
  - **Slow start threshold (`ssthresh`)**: determine whether the slow start or congestion avoidance algorithm is used to control data transmission

Slow Start

- cwnd initial value ≤ 2*SMSS (if SMSS > 2190 bytes)
- Slow start rationale
  - Transmission starts with unknown conditions: slow but sure probe of the network to determine the available capacity
- Slow start used
  - when cwnd < ssthresh or
  - after repairing loss detected by the retransmission timer
- TCP increments cwnd by at most SMSS bytes for each ACK received that cumulatively acknowledges new data:

  cwnd += SMSS
- Multiplicative increase: doubling of cwnd each RTT

Congestion Avoidance (AIMD)

- Carefully probes for congestion

- Used when:

  cwnd > ssthresh (initial value: ssthresh = 64K)

- ACK received: Additive Increase

  cwnd = cwnd + MSS x MSS/cwnd
  - implies additive increase: 1 SMSS per RTT
  - continues until congestion is detected

Retransmission Timeout (RTO)

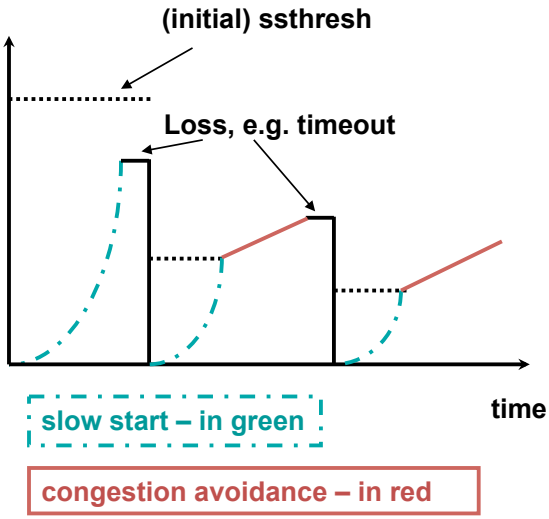
- Timeout on a segment: Multiplicative Decrease

```
ssthresh = max(FlightSize/2, 2*SMSS)
cwnd = 1SMSS
```

- TCP sender
  - retransmits the dropped segment
  - uses the slow start algorithm to increase the window size (from 1 full-sized segment to the new value of `ssthresh`)

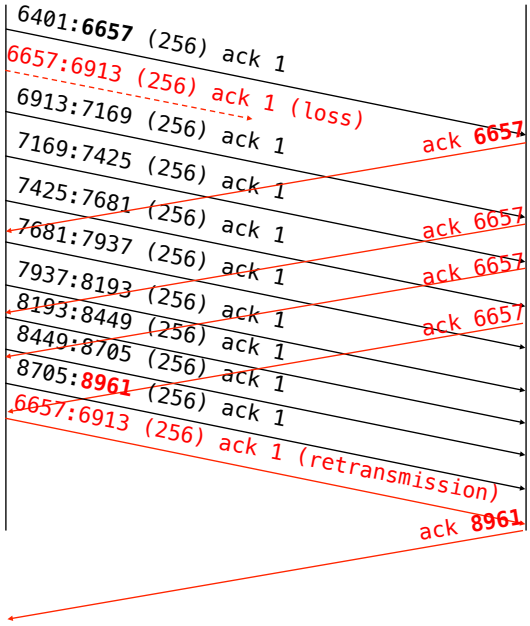
Slow Start & Congestion Avoidance, Retransmission Timeouts

- Initally:
  - cwnd ≤ 2*MSS
  - ssthresh = very high (64KB)
- If a new ACK comes:
  - if cwnd < ssthresh update cwnd according to slow start
  - if cwnd > ssthresh update cwnd according to congestion avoidance
  - If cwnd = ssthresh either
- If timeout (i.e. loss) :
  - ssthresh = max (flight size/2, 2*SMSS)
  - cwnd = 1*SMSS



Duplicate ACKs

- Duplicate ACK send immediately
  - upon arrival of out-of-order segment
  - when incoming segment fills in all or part of a gap in the sequence space
- Duplicate ACK result of various network problems:
  - Data segments re-ordering
  - ACK or data segments replication
  - Dropped segments: segments after a dropped segment trigger duplicate ACKs until loss repair
- 3 duplicate ACKs indicates a 'at-least-one' segment loss
  - TCP sender uses the "fast retransmit"algorithm



Fast Retransmit
TCP Tahoe (1988)

- Principle:
  - Speeds loss recognition: Duplicate ACKs instead of timeout
  - Duplicate ACK: loss or simple reordering?
    - TCP packets network reordering
    - ACK or data segments replication
    - Dropped segments
- Assumption:
  - Triple duplicate ACK is loss indication
- Fast Retransmit algorithm:
  - On arrival of the 3rd duplicate ACK:
    - retransmit missing segment
    - enter in fast recovery (instead of slow start)

Fast Recovery
TCP Reno (1990)

- Returning to slow start after fast retransmitting kills ACK clocking
- On arrival on the third duplicate ACK is received:

  ssthresh = max (FlightSize / 2, 2*SMSS)
- Retransmit missing segment and:

  cwnd = ssthresh + 3
- For each additional duplicate ACK (after the 3rd):

  cwnd += SMSS
- Send SMSS bytes of previously unsent data if cwnd and rwnd allow
- On arrival of ACK for previously unacknowledged data:

  cwnd = ssthresh

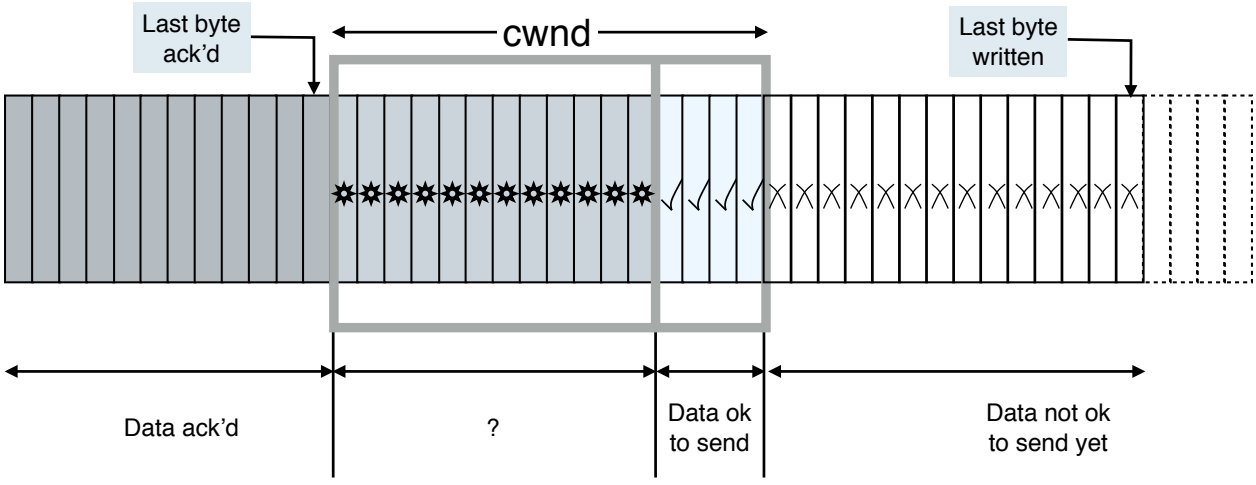
artificial "inflate" by the number of segments (i.e. 3) that have left the network

artificial "inflate" reflect the additional segment that has left the network

"deflate" the window

Artificial Inflate

- During fast recovery:
  - The right edge moves of SMSS bytes on the right for each dup ACK received
  - The left edge of the cwnd window is locked on the last byte ack'd by the dup ACKs

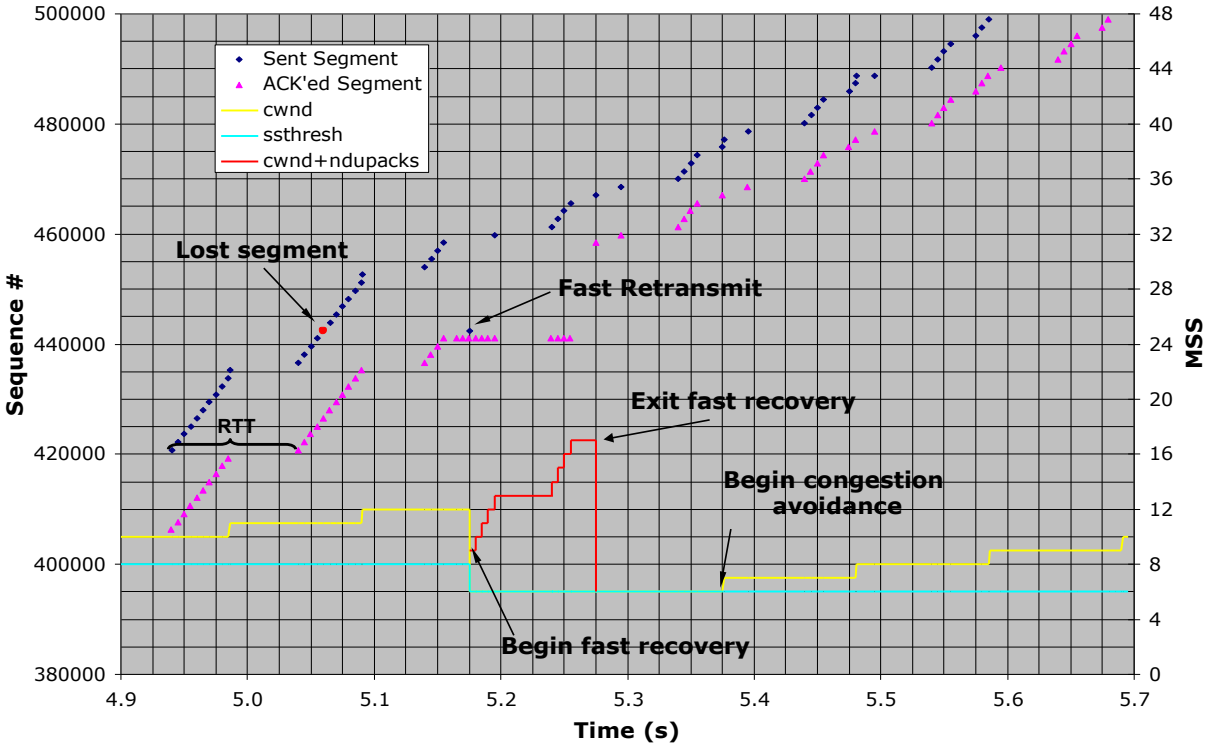


TCP NewReno
Modified Fast Recovery (1996)

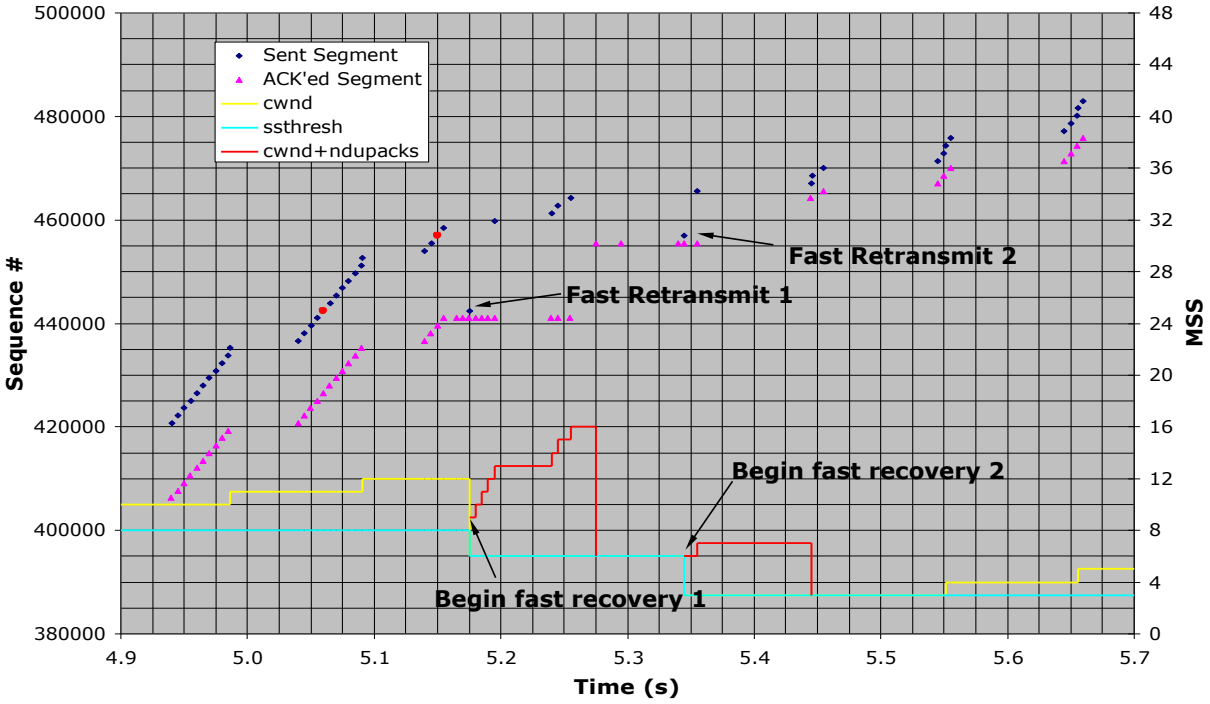
- Partial ACKs:
  - An ACK that acknowledges some but not all the segments that were outstanding at the start of fast recovery
  - NewReno interprets this as an indication of multiple loss
- If partial ACK received, re-transmit the next missing segment immediately and deflate the congestion window:
  - cwnd = ssthresh
  - ssthresh = max (FlightSize / 2, 2*SMSS)
- Sender remains in fast recovery until all data outstanding when fast recovery was initiated is ACK'ed.
  - Additional dupACKs artificially increase the window

- With TCP Reno, sender will go to congestion avoidance after the first partial ACK and eventually timeout

TCP Reno



TCP NewReno



More TCP Variants

- TCP-Tahoe:
  - implements the slow start, congestion avoidance, and fast retransmit
- TCP-Reno:
  - implements the slow start, congestion avoidance, fast retransmit, and fast recovery
- Among other implementations are:
  - TCP NewReno, the most commonly implemented on webservers today, according to surveys
  - TCP Vegas, SACK TCP

Wrapup

- Slow Start
  - Every ack increases the sender's window (cwnd) size by 1
- Congestion Avoidance
  - Reducing sender's window size by half at experience of loss, and increase the sender's window at the rate of about one packet per RTT
- Fast Retransmit
  - Don't wait for retransmit timer to go off, retransmit packet if 3 duplicate ACKs received
- Fast Recovery
  - Since duplicate ack came through, one packet has left the wire
  - Perform congestion avoidance, don't jump down to slow start
- Modified Fast Recovery
  - Sender remains in fast recovery until all data outstanding when fast recovery was initiated is ACK'd

Summary of TCP Behavior

TCP Variation	Response to 3 dupACK's	Response to Partial ACK of Fast Retransmission	Response to "full" ACK of Fast Retransmission
Tahoe	Do fast retransmit, enter slow start	++cwnd	++cwnd
Reno	Do fast retransmit, enter fast recovery	Exit fast recovery, deflate window, enter congestion avoidance	Exit fast recovery, deflate window, enter congestion avoidance
NewReno	Do fast retransmit, enter modified fast recovery	Fast retransmit and deflate window – remain in modified fast recovery	Exit modified fast recovery, deflate window, enter congestion avoidance

- When entering slow start, if connection is new,
  ssthresh = arbitrarily large value
  cwnd = 1.
  else,
  ssthresh = max(flight size/2, 2*MSS)
  cwnd = 1.
- In slow start ++cwnd on new ACK

- When entering either fast recovery or modified fast recovery,
  ssthresh = max(flight size/2, 2*MSS)
  cwnd = ssthresh
- In congestion avoidance
  cwnd += 1*MSS per RTT

Is There a Better Way?

- Tahoe, Reno and NewReno can detect congestion:
  - … by creating congestion!
- They carefully probe for congestion
  - by slowly increasing their sending rate
- When they find (create), congestion,
  - they cut sending rate at least in half!
- Saw-toothed sending rate:
  - highly erratic throughput
- What if TCP could detect congestion without causing congestion?

Active Queue Management

- Random Early Detection (RED)
  - Randomly drops packets before queue is full
  - In the hope of reducing the rates of some flows
  - Packet drop probability as a function of queue length
  - *Hard to tune*
- Explicit Congestion Notification (ECN)
  - routers mark packets instead of dropping them in order to signal impending congestion.
  - Receivers echo the congestion indication to the senders, which reduces its transmission rate
  - *Must be supported by the end hosts and the routers*

Conclusions

- Congestion is inevitable
  - Internet does not reserve resources in advance
  - TCP actively tries to push the envelope
- Congestion can be handled
  - Additive increase, multiplicative decrease
  - Slow start, and slow-start restart
- Active Queue Management can help
  - Random Early Detection (RED)
  - Explicit Congestion Notification (ECN)

Conclusions

- TCP et UDP
  - Multiplexage et démultiplexage
  - Détection d'erreur
    - somme de contrôle
- TCP
  - Connexion et états
  - Réparation des pertes et correction d'erreur
    - numéros de séquence
    - estimation du RTT
    - expiration de temporisateur et retransmission
  - Contrôle de flux
    - fenêtre glissante
  - Contrôle de congestion