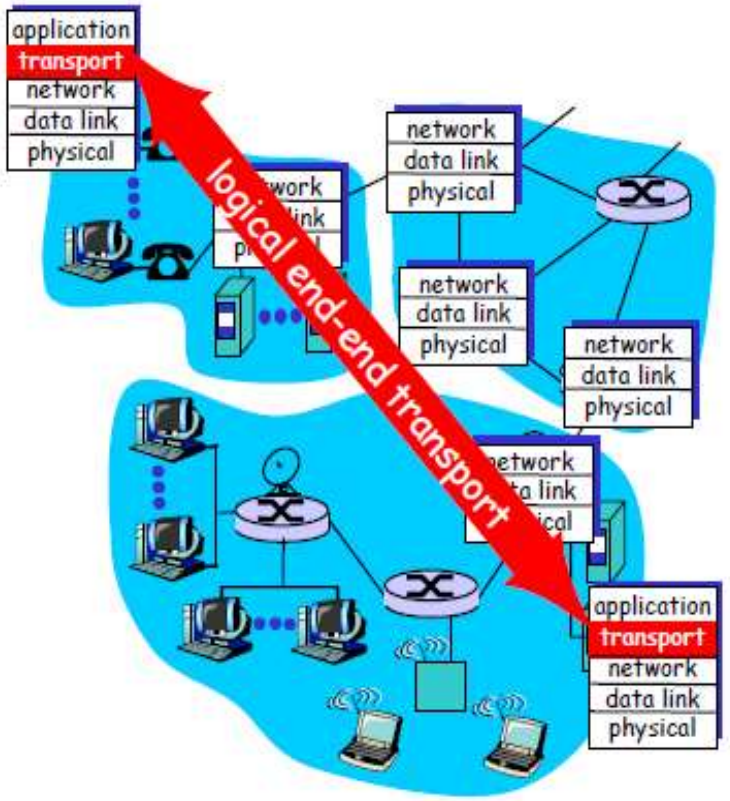
Mr A.Dekhinet
Université BATNA 2 / Département Informatique
a.dekhinet@univ-batna2.dz
http://staff.univ-batna2.dz/dekhinet-abdelhamid

Chapitre 3 : Couche Transport

Introduction

- Assure le transport de bout en bout (End-To-End) des TPDU entre entités paires.
- Abstraction de la topologie et les technologies sous-jacentes : Logical End-End Transport
- Assure la communication logique entre processus, alors que la couche réseau assure la communication logique entre machine (Device).
- Les protocoles de transports s'exécutent dans les systèmes finaux (End system), et non pas dans les systèmes intermédiaires (Routeurs, Switchs, …)
- Plusieurs protocole de transport sont offerts aux applications. Internet (TCP/IP) offre deux services de transport de base :
  - Service de transport fiable (Reliable data transport) : **TCP**
  - Service de transport non fiable (unReliable data transport) : **UDP**
- Autres protocoles de transport : SCTP (Stream Control Transmission Protocol), DCCP (Datagram Congestion Control Protocol)



Principes fondamentales de la couche transport : Services et Fonctions

- Les principaux services et fonctions qui constituent la base sur laquelle se fondent la conception de la couche transport sont :
  - *Multiplexage/Démultiplexage (Applications adressing)*
  - *Transfert de données fiable (RDT : Reliable data transfer)*
  - *Contrôle de flux (Flow control)*
  - *Contrôle de congestion (Congestion control)*
- Le choix des services, qu'offrent une couche de transport, dépend des besoins et des impléméntations.

TCP : *Envoyer et Recevoir*

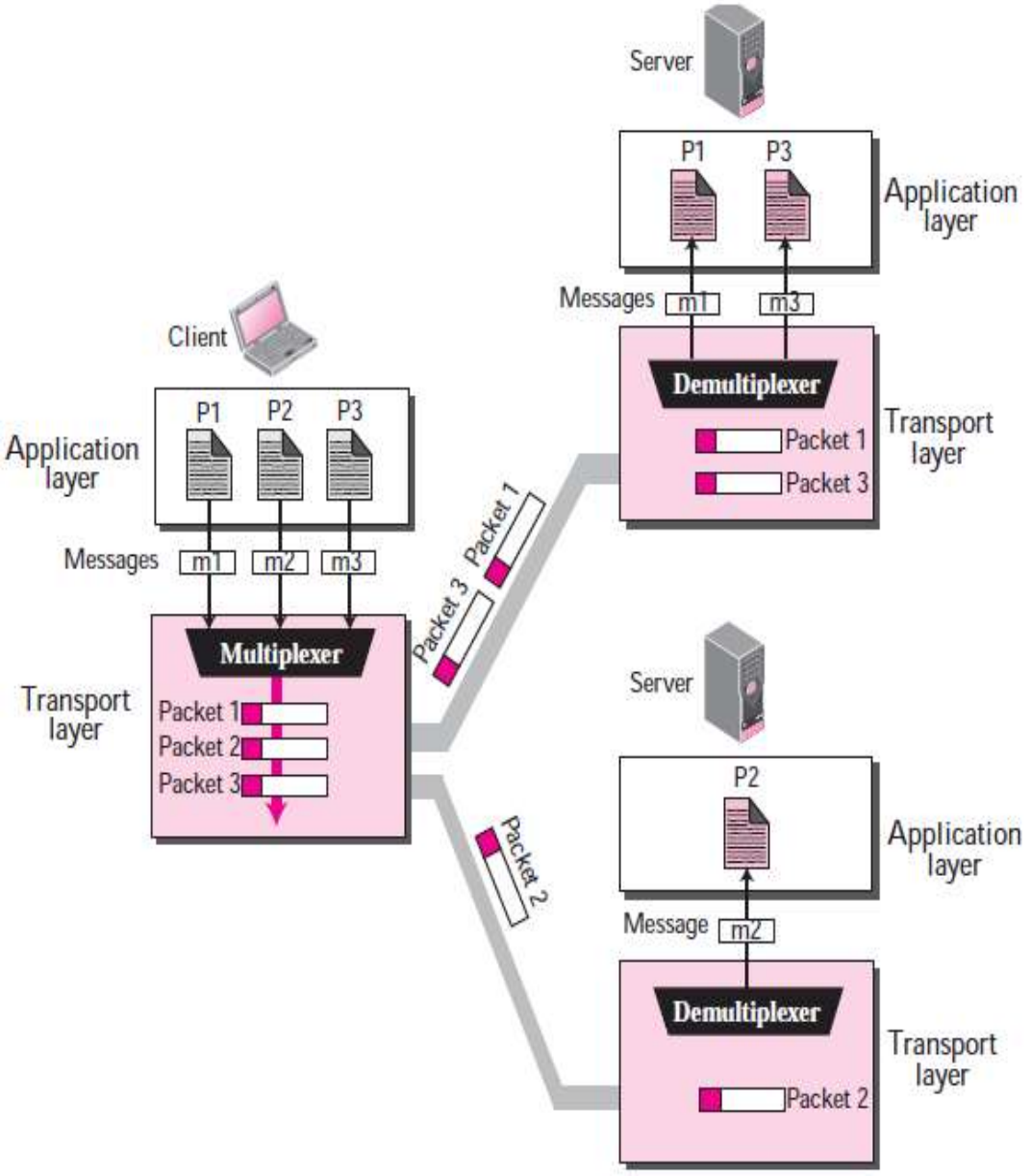
- *Multiplexage/Démultiplexage*
  Adressage et identification des applications(Ports)
- *Transfert de données fiable*
  Contrôle d'érreurs (Acquitements, Retransmission)
  Livraisoon des données en ordre (Numéro de séquence)
  Detection d'érreurs (Bit cheking checksum)
- *Contrôle de flux (Host)*
- *Contrôle de congestion (Network)*
- *Gestion de la connexion*
- *Segmentation*

UDP : *Envoyer et ???*

- *Multiplexage/Démultiplexage*
  Adressage et identification des applications(Ports)
- *Transfert de données non fiable*
  Detection d'érreurs (Bit cheking checksum)

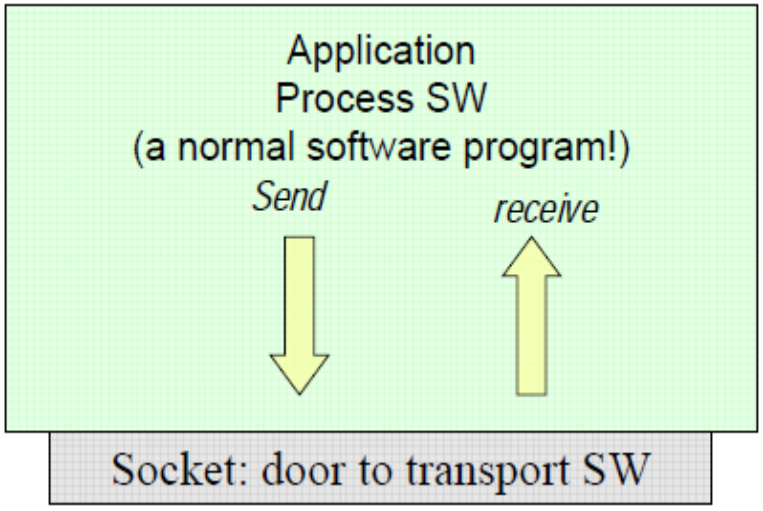
Multiplexage/Démultiplexage

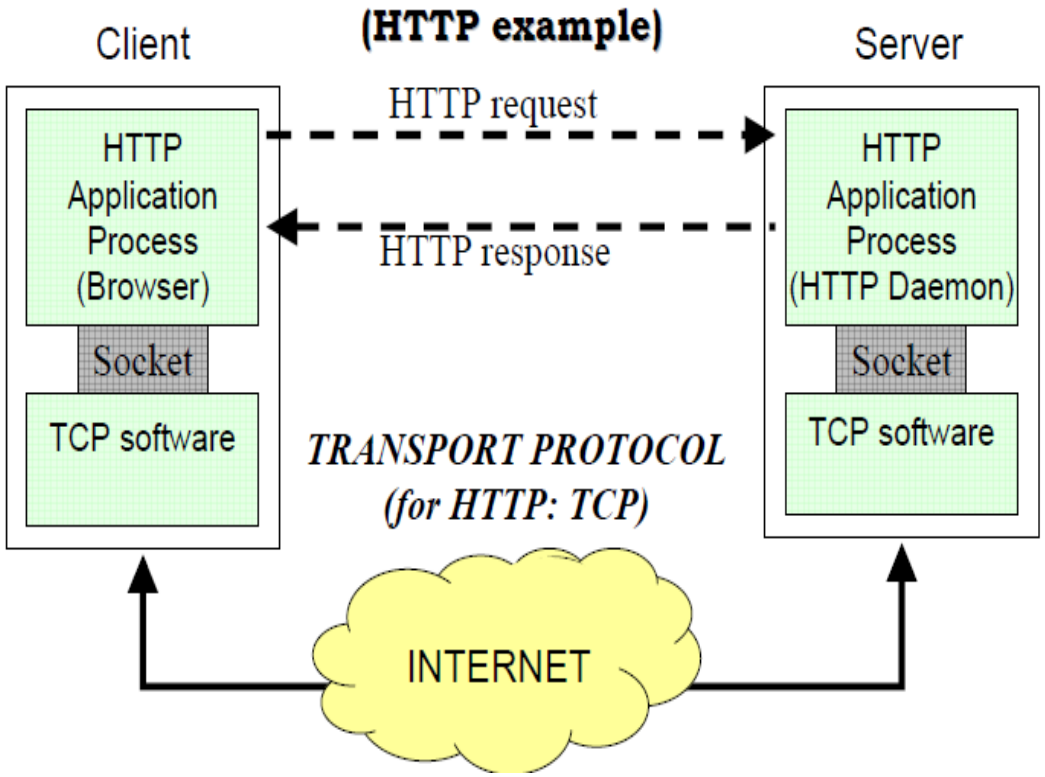
- Plusieurs procesus d'application utilisent le même protocole de transport et se distinguent par le SAP.
- Le SAP (Service Acces Point) dans TCP/IP est le port.
- La couche de transport coté client accepte trois messages des processus P1, P2 et P3 et crée un flux de trois paquets. Elle agit comme un multiplexeur.
- Les paquets 1 et 3 utilisent le même canal logique pour atteindre la couche transport du serveur.
- A la reception, la couche transport du serveur réalise le travail d'un démultiplexeur et dispatche les messages à deux processus différents P1 et P3.



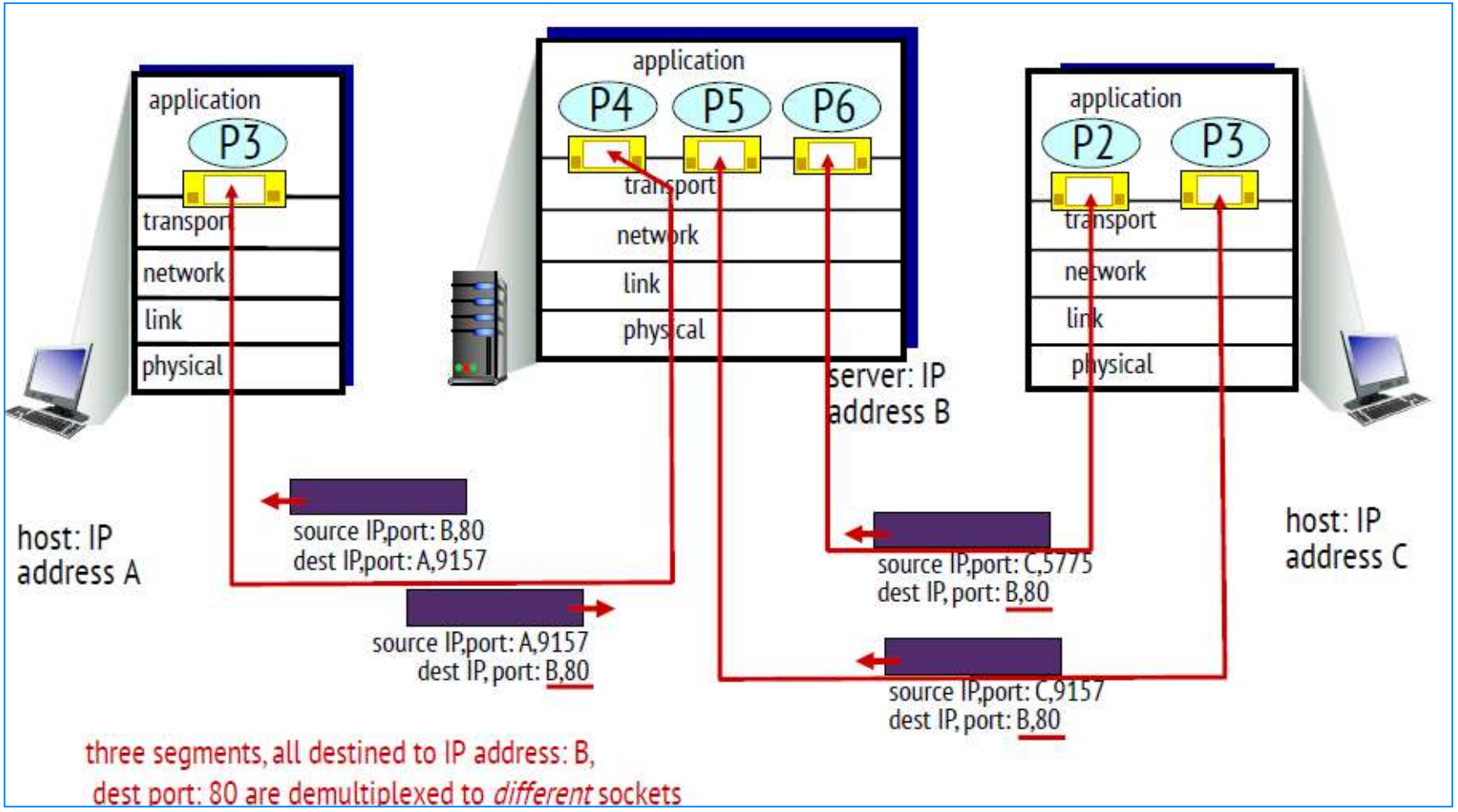
Multiplexage/Démultiplexage : TCP/IP

- D'un point de vue développeur d'application, l'envoi et la reception de donnée, se fait via une API : Socket
- L'API socket permet au developpeur d'abstraire l'infrastructure de transport.
- Multiplexage : Collecte de données à partir de plusieurs sockets, et les envelopper avec un entête.
- Démultiplexage : Remise des données au socket adéquate, en utilisant l'entête.





- Une socket est identifiée par un quadruple :
  (IPSource, PortSource, IPDestination, PortDestination)



UDP (User Datagram Protocol) : Protocole non fiable

- PDU = Datagramme
- Protocole leger : Mux/Démux, Détection des érreurs, Taille entête réduite, …
- Service non connecté : Pas de gestion de connexion
- Service non fiable (Best effort service) : Pas de garantie de réception, de delai et de l'ordre des datagrammes
  - *Les datagrammes peuvent être perdues*
  - *Les datagramme speuvent être dupliqués*
  - *Les datagramme speuvent arriver en désordre*
- Pas de contrôle de flux et de congestion

b. Header format

- *Source Port : Entier sur 16 bit (0 - 65535) identifiant l'application source*
- *Destination Port : Entier sur 16 bit identifiant l'application destination*
- *Length : Taille en octet du datagramme, entête incluse*
- *Checksum : Champ optionnel il sert à la detection des érreurs (Bit error)*

Les Numéros de port

- IANA (Internet Assigned Numbers Authority) : Organisation de standardisation qui supervise l'allocation globale des adresses IP, les numéros de ports, la gestion de la zone racine dans DNS, …
- Port 0 à 1023 : Appelés ports connus (Well-known ports), sont des ports réservés et sont attribués par l'IANA (Internet Assigned Numbers Authority). Par exemple, le port 80 est réservé pour l'application serveur Web.
- Port 1024 à 49151 : il s'agit de ports qu'une organisation, telle que des développeurs d'applications, peut enregistrer auprès de l'IANA pour être utilisés pour un service particulier. Ceux-ci doivent être traités comme semi-réservés.
- Port 49152 à 65535 : Il s'agit de ports utilisés par les programmes clients. Sont des ports éphémères (Temporaire ou de courte durée de vie)

Well-known Ports used by TCP

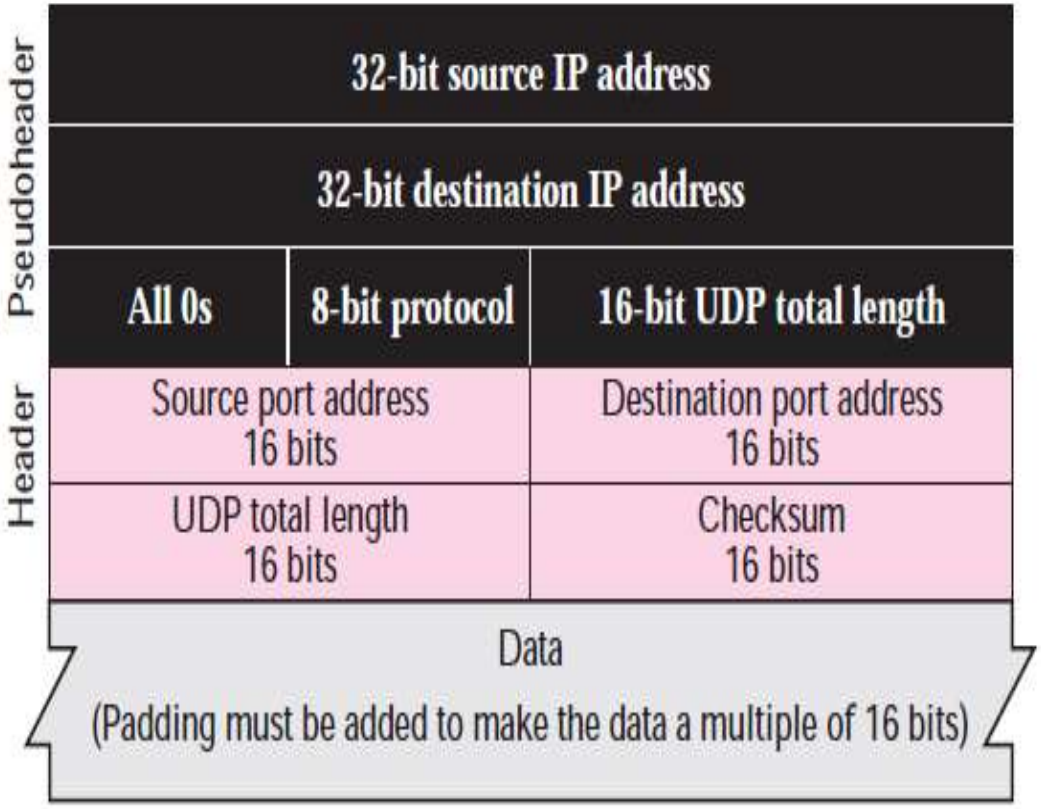
Port	*Protocol*	*Description*
7	Echo	Echoes a received datagram back to the sender
9	Discard	Discards any datagram that is received
11	Users	Active users
13	Daytime	Returns the date and the time
17	Quote	Returns a quote of the day
19	Chargen	Returns a string of characters
20 and 21	FTP	File Transfer Protocol (Data and Control)
23	TELNET	Terminal Network
25	SMTP	Simple Mail Transfer Protocol
53	DNS	Domain Name Server
67	BOOTP	Bootstrap Protocol
79	Finger	Finger
80	HTTP	Hypertext Transfer Protocol

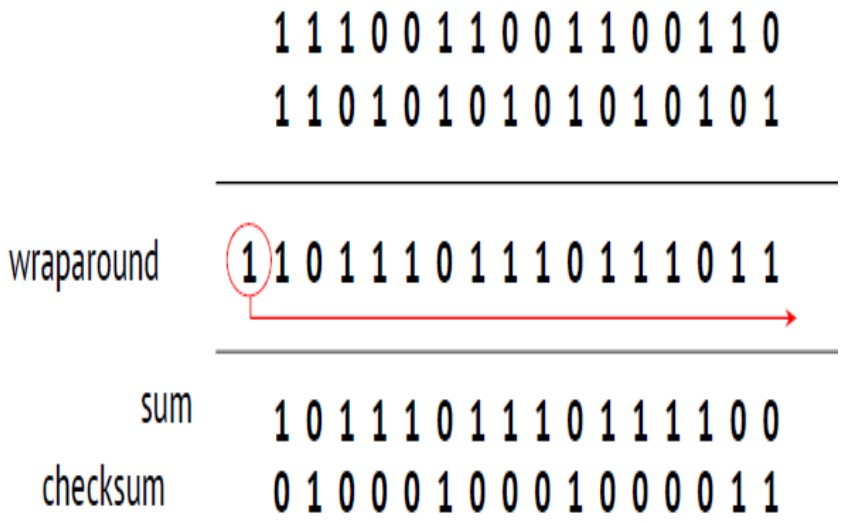
Well-known Ports used with UDP

Port	*Protocol*	*Description*
7	Echo	Echoes a received datagram back to the sender
9	Discard	Discards any datagram that is received
11	Users	Active users
13	Daytime	Returns the date and the time
17	Quote	Returns a quote of the day
19	Chargen	Returns a string of characters
53	Domain	Domain Name Service (DNS)
67	Bootps	Server port to download bootstrap information
68	Bootpc	Client port to download bootstrap information
69	TFTP	Trivial File Transfer Protocol
111	RPC	Remote Procedure Call
123	NTP	Network Time Protocol
161	SNMP	Simple Network Management Protocol
162	SNMP	Simple Network Management Protocol (trap)

UDP Checksum : Détection d'érreurs

- Le checksum inclut trois section : entête UDP, Pseudoentête, Donnée de la couche application.
- Calcul du checksum : Le complément à 1 sur 16 bits de la somme des compléments à 1 des octets des trois sections pris deux par deux (mots de 16 bits)





UDP Checksum : Détection d'érreurs

- En émission : Calcul Chesksum = Complément-a-1 (∑ 16-Bit-Word)
- En réception : Calcul Somme = ∑ 16-Bit-Word
  Si Complément-a-1(Somme) = 0 Alors Pas d'érreurs
  Sinon Il y'à des érreurs
- Exemple simple : Données à trasmettre 0110 et 1000
  Somme= 0110 + 1000 =1110
  Checksum = Complément-a-1(Somme)=0001

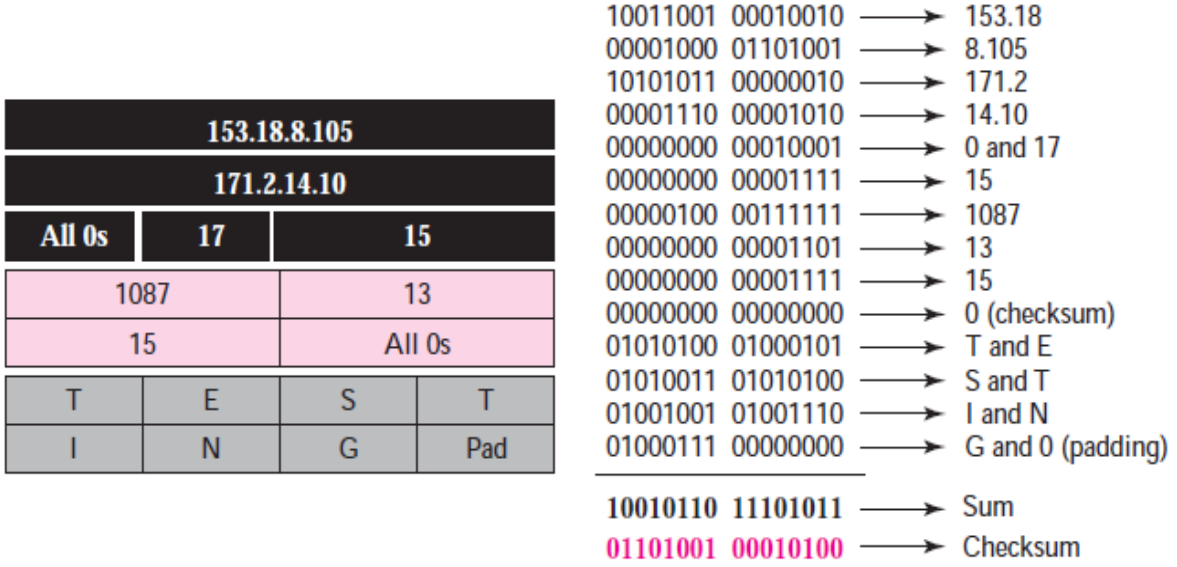
Emission : 0110 1000 0001

Réception (Sans érreur) : 0110 1000 0001

Calcul Somme = 0110+1000+0001 = 1111

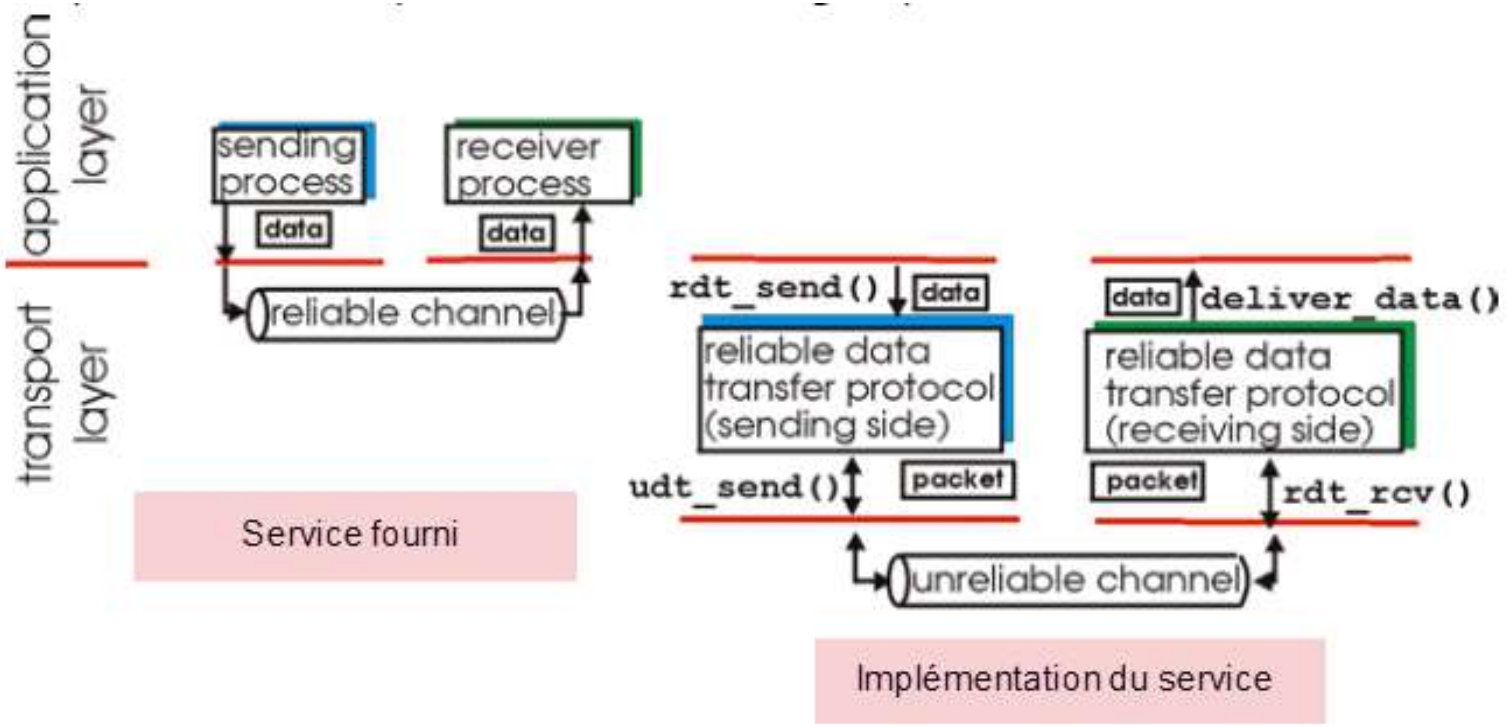
Complément-a-1(Somme) = 0000 Alors Pas d'érreurs

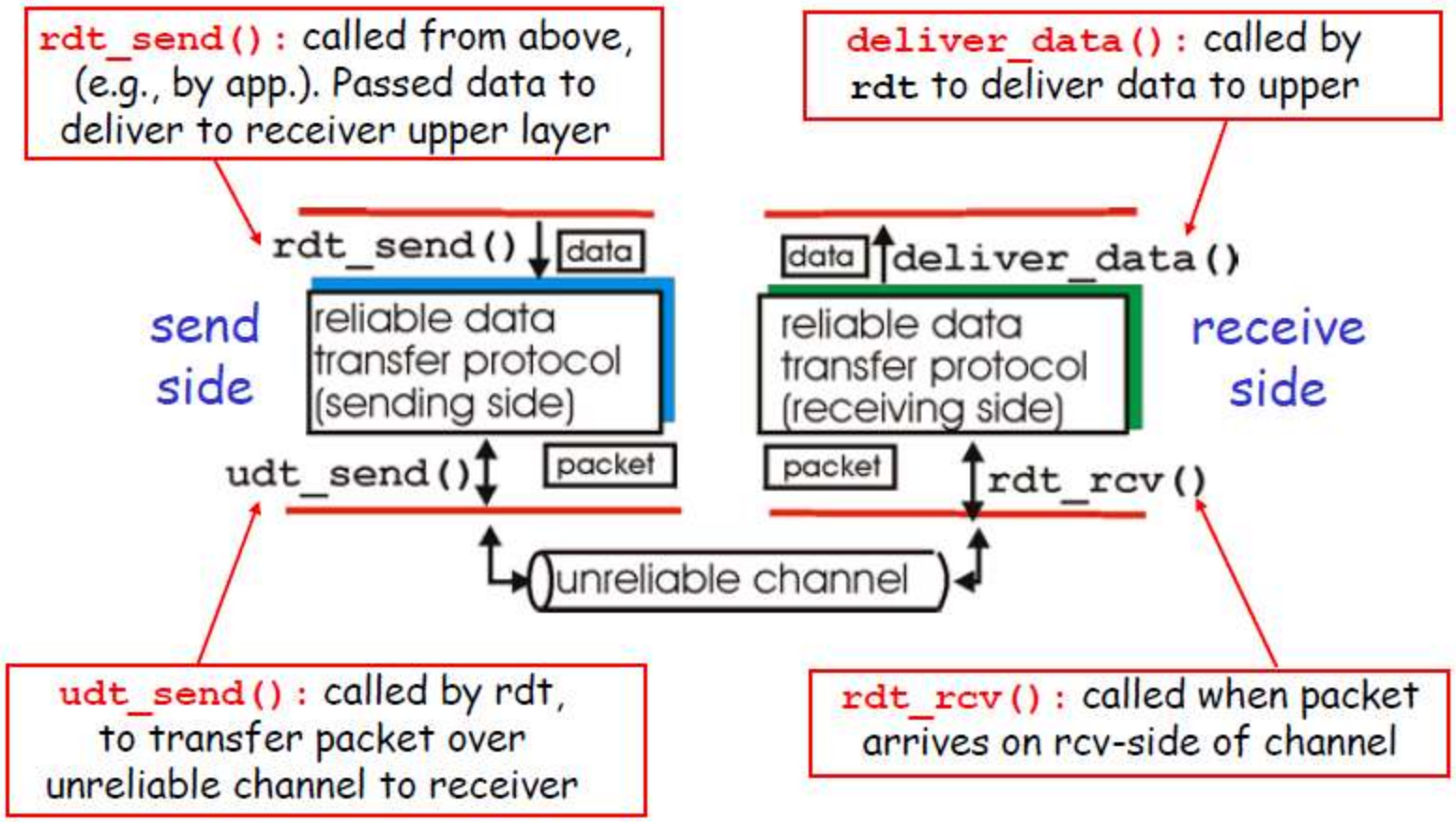
Checksum calculation of a simple UDP user datagram



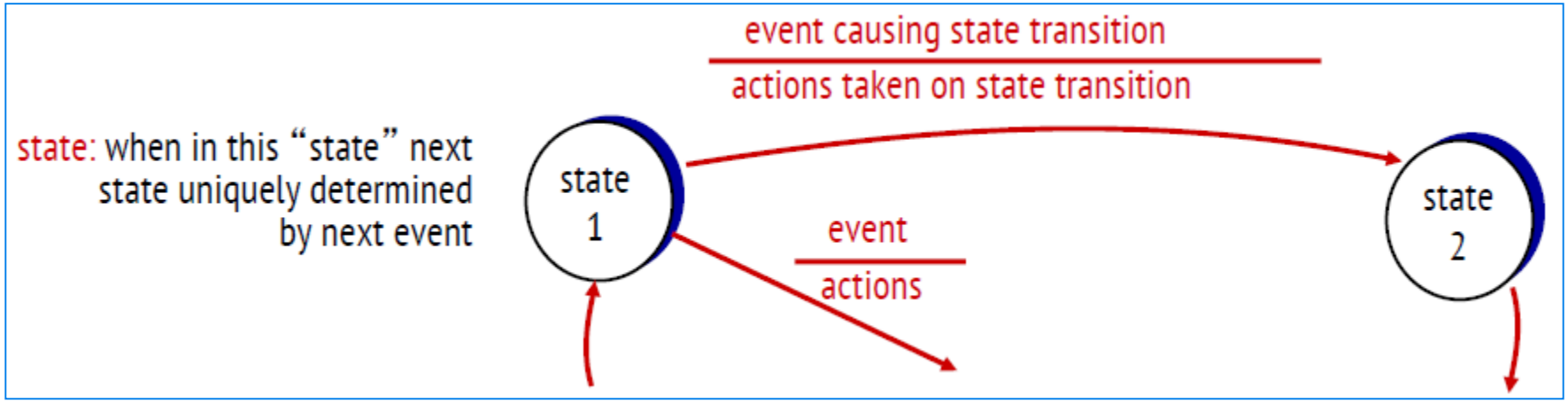
Transfert de données fiable (RDT)

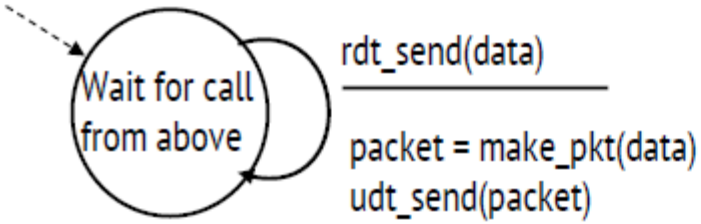
- Problème multicouche qui touche : Application, Transport, Liaison
- Top-10 des thèmes les plus importants dans le domaine des réseaux.
- Le canal de communication est généralement non fiable (Noisy : Bruité).
- La responsabilité du RDT est de mettre en œuvre l'abstraction du service.
- Les caractéristiques d'un canal non fiable détermineront la complexité du RDT



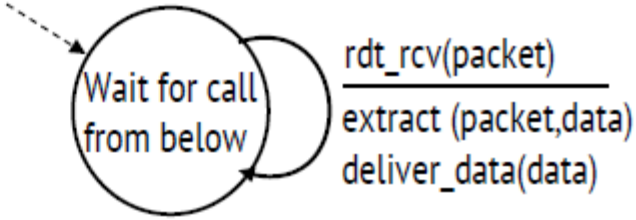

rdt_send(): called from above, (e.g., by app.). Passed data to deliver to receiver upper layer
deliver_data(): called by rdt to deliver data to upper
rdt_send()
data
data
deliver_data()
send side
reliable data transfer protocol (sending side)
reliable data transfer protocol (receiving side)
receive side
udt_send()
packet
packet
rdt_rcv()
unreliable channel
udt_send(): called by rdt, to transfer packet over unreliable channel to receiver
rdt_rcv(): called when packet arrives on rcv-side of channel

- Le comportement de l'émeteur et du récepteur peut être spécifié à l'aide des machines ou automates d'états finis (FSM : Finite State Machine)
- Exemple de canal parfaitement fiable (Pas de perte et pas d'érreurs de bit)





sender



receiver

Glossaire

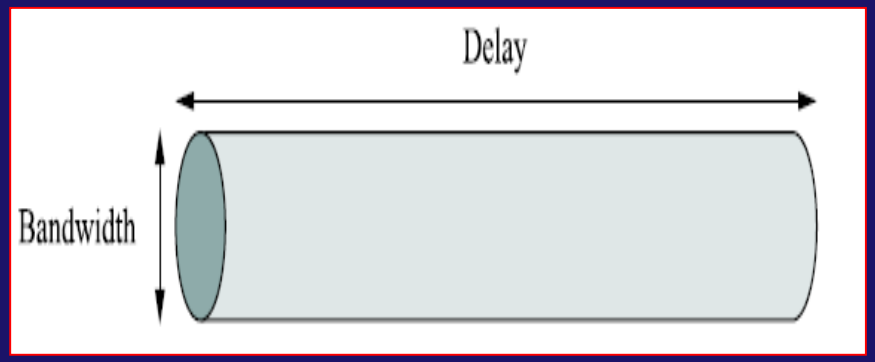
Évènements	rdt_send(data)	Envoi de **data** par l'application
	rdt_rcv(rcvpkt)	Réception de **rcvpkt** par la couche réseau
	corrupt(rcvpkt)	Vrai si **rcvpkt** est corrompu
	notcorrupt(rcvpkt)	Vrai si **rcvpkt** n'est pas corrompu
	isACK(rcvpkt)	Vrai si **rcvpkt** est un **ACK**
	isNAK(rcvpkt)	Vrai si **rcvpkt** est un **NAK**
	has_seq0(rcvpkt)	Vrai si **rcvpkt** a le numéro de séquence **0**
	has_seq1(rcvpkt)	Vrai si **rcvpkt** a le numéro de séquence **1**
	timeout	Fin du temporisateur
	udt_send(sndpkt)	Envoi de **sndpkt** à la couche réseau
	pkt=(data, checksum)	Construire un segment **pkt** avec **data** et calculer le **checksum**
Actions	extract(rcvpkt, data)	Extraire **data** du segment **rcvpkt**
	deliver_data(data)	Envoi de **data** à l'application
	start_timer	Démarrer le temporisateur
	stop_timer	Arrêter le temporisateur
	Λ	Ne rien faire

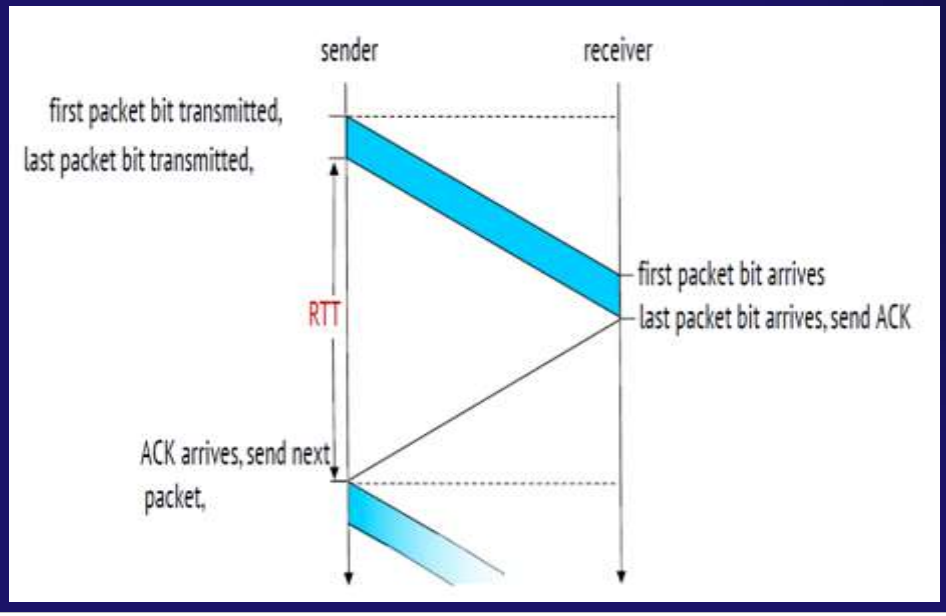
RDT : Automatic Repeat re-Quest (ARQ)

- ARQ est un protocole ou méthode de contrôle d'éreurs
- ARQ utilise des acquitement (ACK/NACK) et des timeouts pour parvenir à un transfert de données fiable.
- Parmi les protocoles ARQ on peut compter :
  - *Stop-and-Wait (Envoyer et Attendre)*
  - *Go-Back-N*
  - *Selective Repeat*
- Plusieurs couches du modèle OSI sont fondées sur ces protocoles (Procédures), et principalement la couche transport et liaison.
- L'éfficacité d'un protocole ARQ dépend de sa performance.
- Les performances dépendent des différents délais impliqués dans le transfert de PDUs (Paquet ou Message).
- Go-Back-N et Selective Repeat sont des protocoles génériques Pipelined.
- TCP utilise principalement Go-Back-N, conjointement avec le controle de flux.
- Le controle de flux dans TCP est basé sur une tehnique de fenêtre d'anticipation variable (Glissante).

RDT : Performances

- Les performances réseaux sont influencées et dépendent de plusieurs facteurs ou métriques (metric)
  - *Délai ou Latence (Delay) : Le temps de transfert total d'un PDU (Incluant tous les délais), depuis une source vers une destination.*
  - *Bande passante (BP: Bandwidth) : Capacité théorique maximale d'un canal de communication(Lien). C'est-à-dire le nombre maximum de bits qui peuvent être transmis par seconde sur un support physique .*
  - *Débit (Throughput) : Capacité actuel ou effective d'un lien, C'est-à-dire le nombre de bits qui peuvent être transmis par seconde effectivement.*
  - *RTT : C'est le temps d'aller et de retour d'un PDU de taille réduite*
  - *Le produit BP*Delai (BPD) : C'est le nombre maximal de bits ou paquets qui pourraient être en transit dans le pipe à un instant donné ou durant un RTT. Idéalement, envoyez suffisamment de paquets pour remplir le pipe avant le premier ACK.*





RDT : Performances

- Durant son acheminement, un PDU peut traverser plusieurs noeuds (Routeurs, …) dans le réseau.
- Le PDU accumule un retard à chaque traversé d'un nœud dans son chemin.
- Le délai nodal total (Latence) de tranmission d'un PDU d'une source vers une destination est :

$$d_{nodal} = d_{trans} + d_{prop} + d_{proc} + d_{queue}$$

d_{trans} : *Temps requis pour faire passer ou mettre tous les bits du PDU dans le lien (Support)*

$d_{trans} = L/R$; L = *Taille du PDU en bits,*

R = *Bande passante ou Débit*

d_{prop} : *Temps de propagation de PDU dans le lien*

$d_{prop} = d/s$; d = *Distance en mêtre*

s = *Vitesse de propagation du lien en m/s* (2.10^8 *meter/sec* - 3.10^8 *meter/sec*)

d_{proc} : *Temps de traitement de PDU dans un nœud*

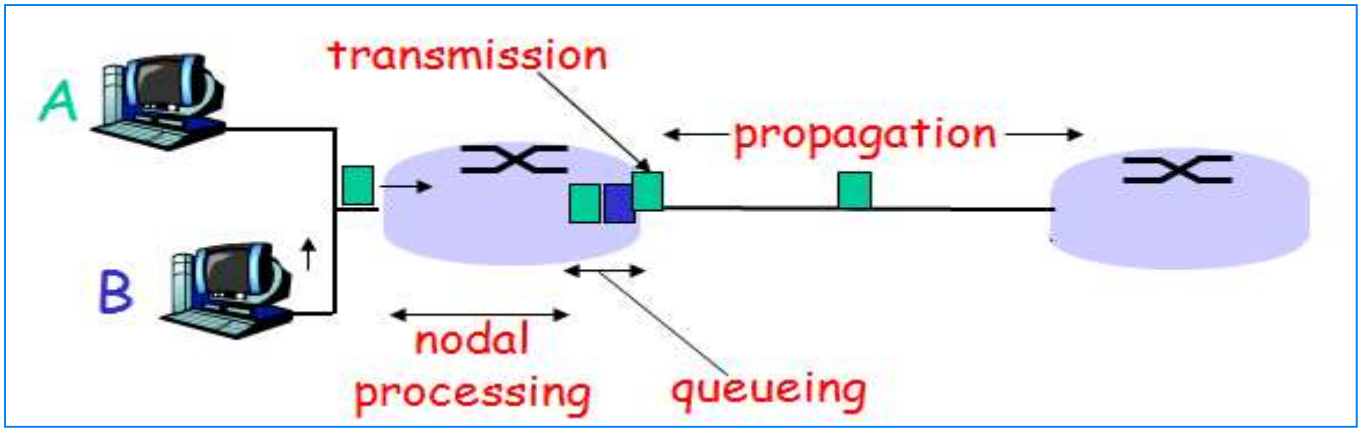
d_{proc} : *Détection érreurs, Algorithme de routage, Mise en file d'attente associé à l'interface de sortie, …*

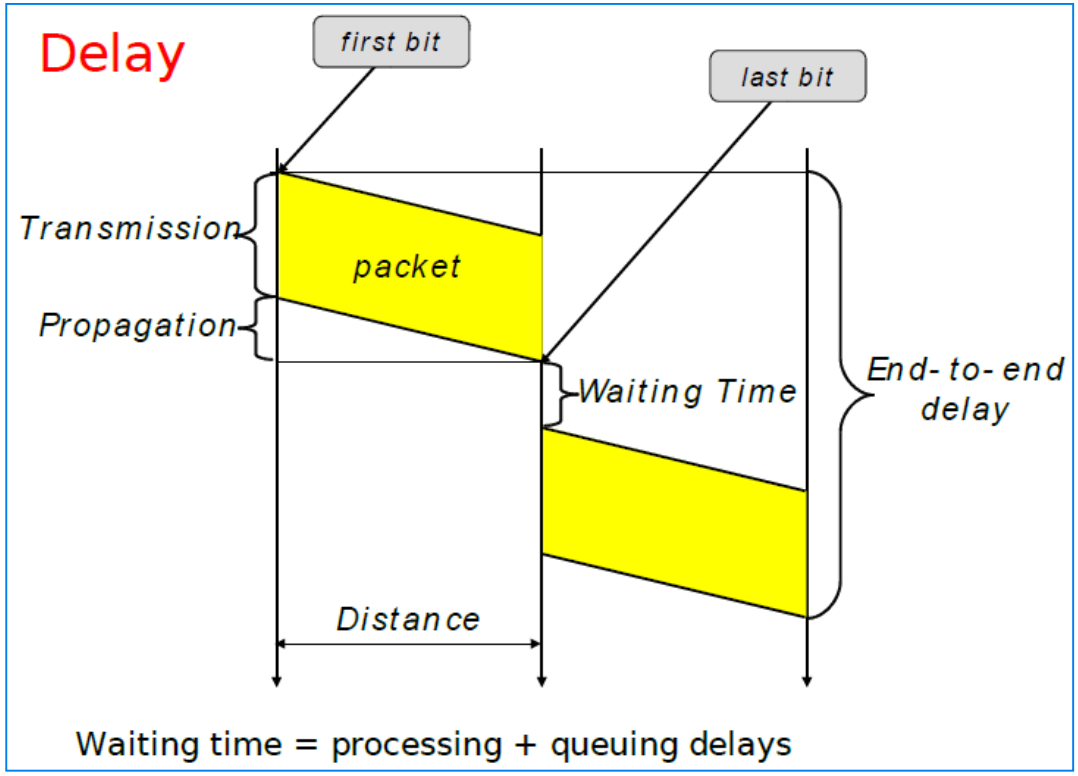
d_{queue} : *Temps d'attente du PDU dans un nœud*

d_{queue} = *Délai d'attente dans la file*

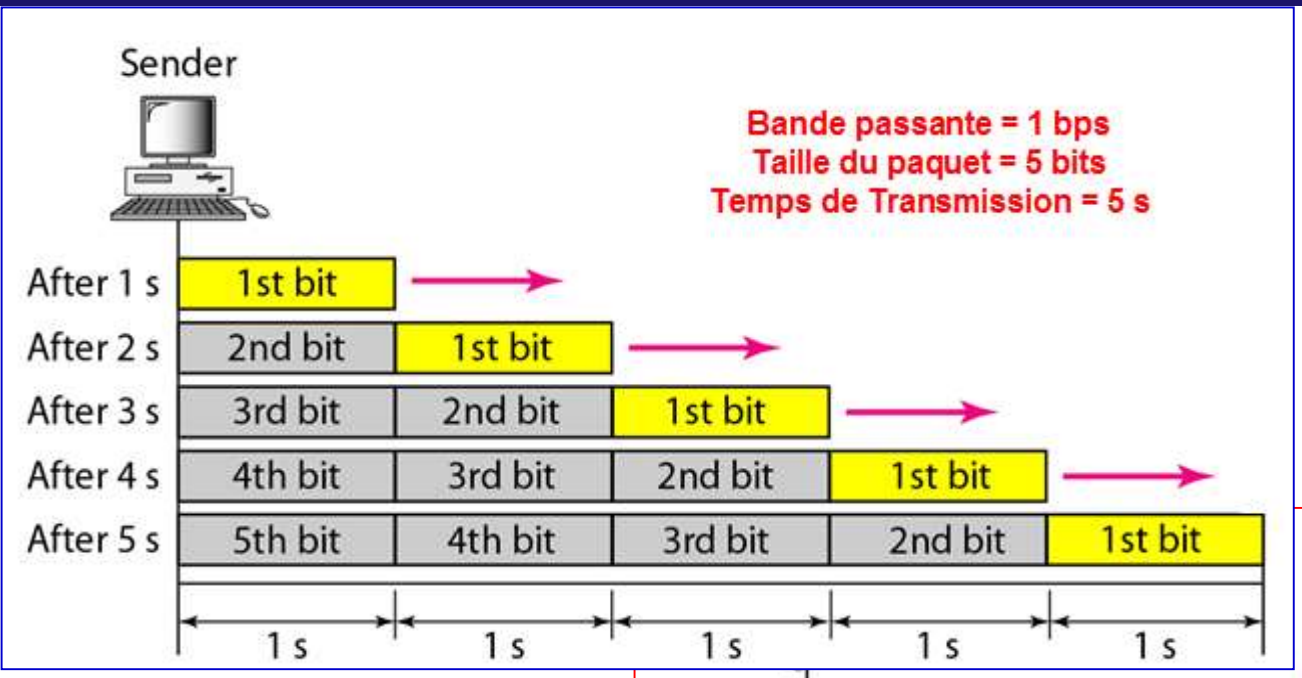
$$d_{End\text{-}To\text{-}End} = N(d_{trans} + d_{prop} + d_{proc} + d_{queue})$$

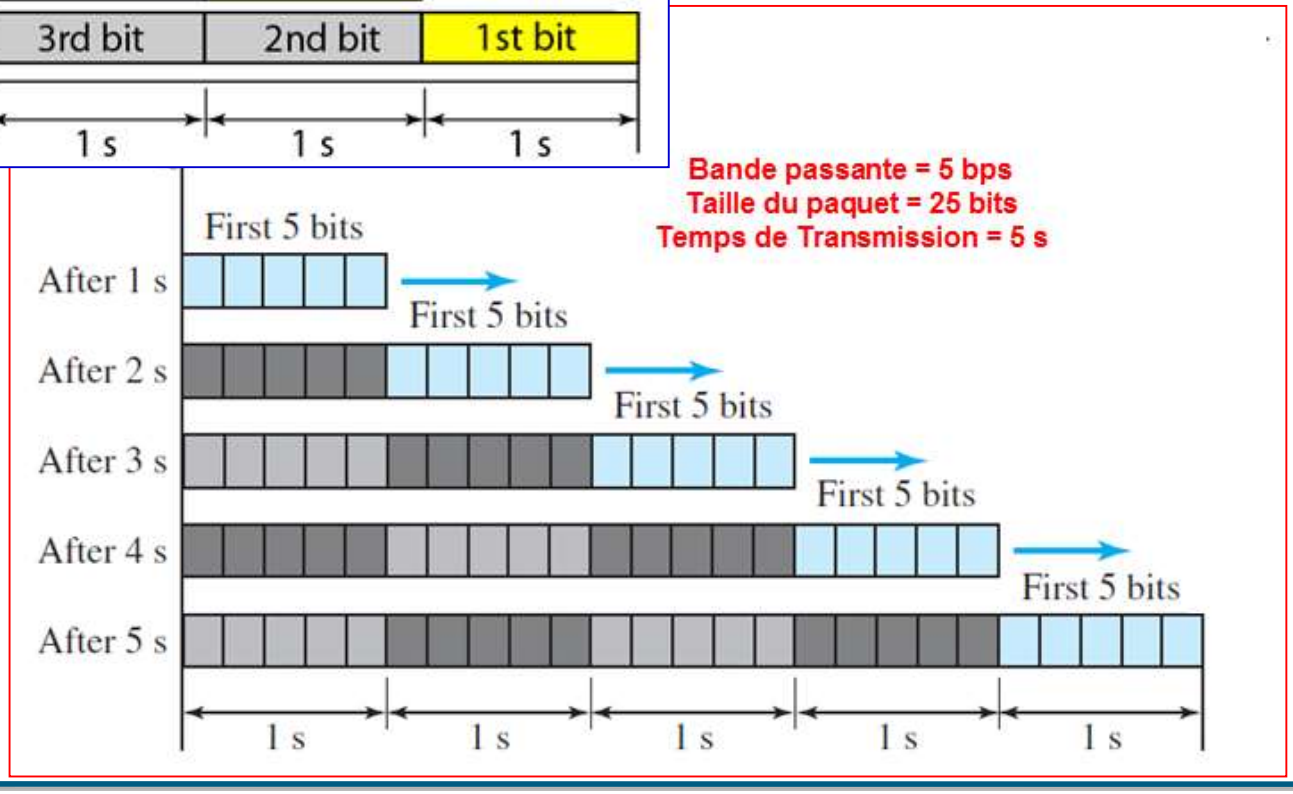
N(Nombre de lien) = Nombre de noeud (Routeur) + 1

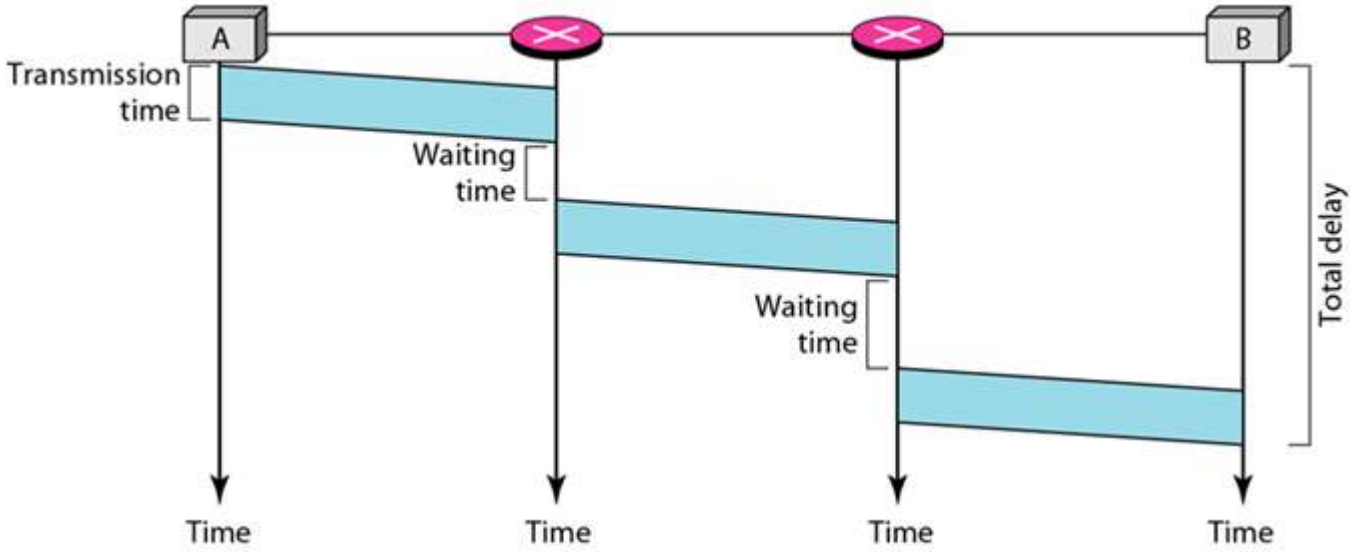

A
B
transmission
propagation
nodal
processing
queueing


Delay
first bit
last bit
Transmission
packet
Propagation
Waiting Time
End-to-end
delay
Distance
Waiting time = processing + queuing delays

RDT : Performance







Total Delay =3T + 3P +2*W

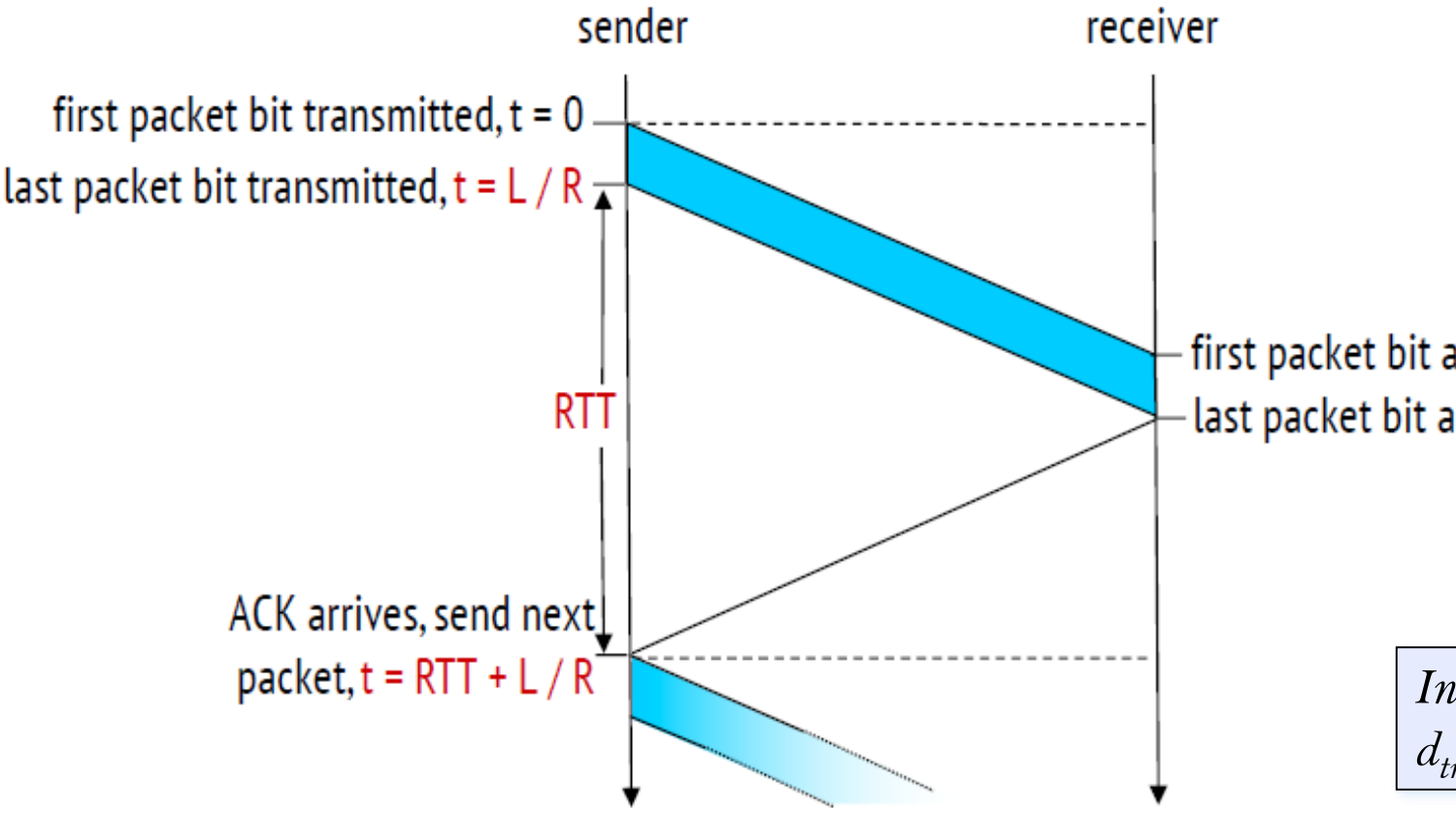
T: transmission delays

P: propagation delays

W : waiting times

RDT : Performance

- RTT peut inclure tous les délais (voir commande traceroute), Dans la plupart des cas on considère $RTT = 2*d_{prop}$
- U_{sender} ou U dénote l'efficacité (Taux ou Ratio d'activité) d'un protocole, il dépend du d_{trans} et d_{prop} .



Inefficace si $d_{trans} << RTT$

RDT : Performance

- Exemple 1 : Supposez qu'un lien point-à-point à 100 Mb/s soit mis en place entre la terre et une colonie lunaire. La distance de la lune à la terre est d'environ 390 000 km et les données voyagent sur le lien à la vitesse de la lumière, 300 000 km/s.

1) Calculer le RTT minimum pour le lien.
2) En utilisant le RTT comme délai, calculez BPD
3) Quelle est la signification de ce produit ?
4) Une caméra sur la base lunaire prend des photos de la Terre et les enregistre au format numérique. Supposons que l'équipe sur terre souhaite télécharger l'image la plus récente, qui fait 25 MB.Quel est le délai minimum qui s'écoulera entre le moment où la requete de recupération de l'image est envoyée et le moment où le transfert est terminé ?

1. $RTT = 2 * d_{prop}$

   $RTT = 2 * (390000\ km / 300000\ km/s)$

   $RTT = 2.6\ seconde$

2. $BPD = 2.6\ s * 100Mb/s = 260\ Mb$

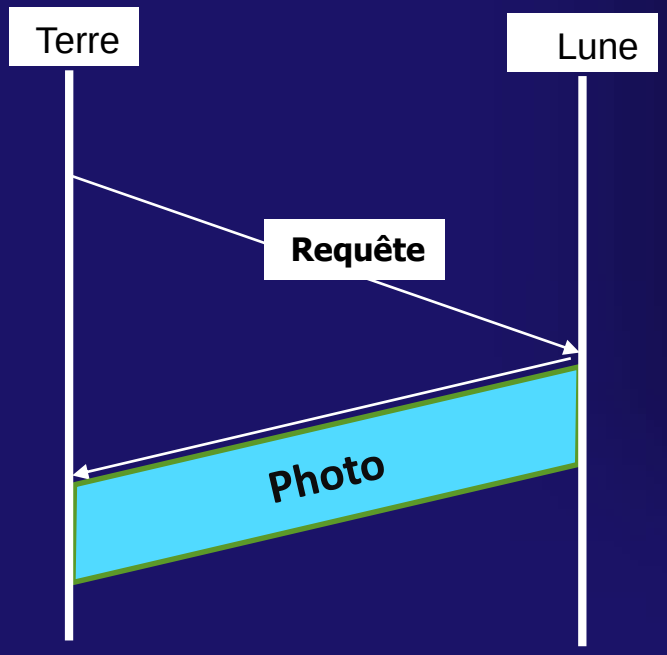
3. *Le BPD : 260 Mb seront envoyés avant que le premier bit d'acquittement ne soit reçu.*

4. $Délai\text{-}Minimum = RTT + Dtrans$

   $= 2.6 + 200\ Mb/100Mbps$

   $= 2.6 + 2$

   $= 4.6\ s$



- Exemple 2 : Calculer le produit Bande-Passante*Délai (BPD) dans chaque cas :

1) Bande passante: 1 Mbps, RTT: 20 ms, Taille du paquet: 1000 bits
2) Bande passante : 10 Mbps, RTT: 20 ms, Taille du paquet : 2000 bits
3) Bande passante : 1 Gbps, RTT: 4 ms, Taille du paquet : 10,000 bits

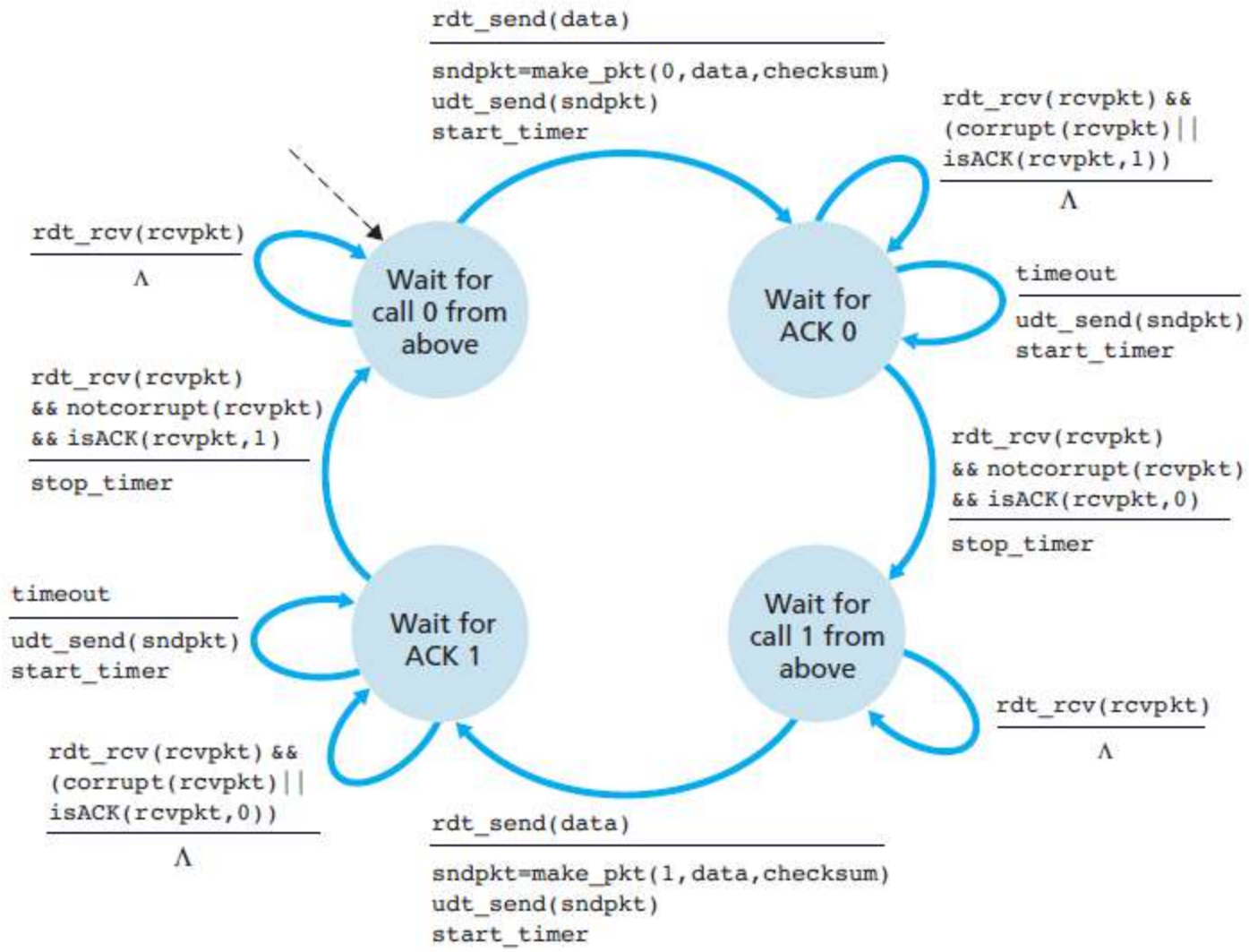
1. $BPD = 1\ Mbps \times 20\ ms = 20{,}000\ bits = 20\ paquet$
2. $BPD = 10\ Mbps \times 20\ ms = 200{,}000\ bits = 100\ paquets$
3. $BPD = 1\ Gbps \times 4\ ms = 4{,}000{,}000\ bits = 400\ paquets$

Protocole Stop-and-Wait

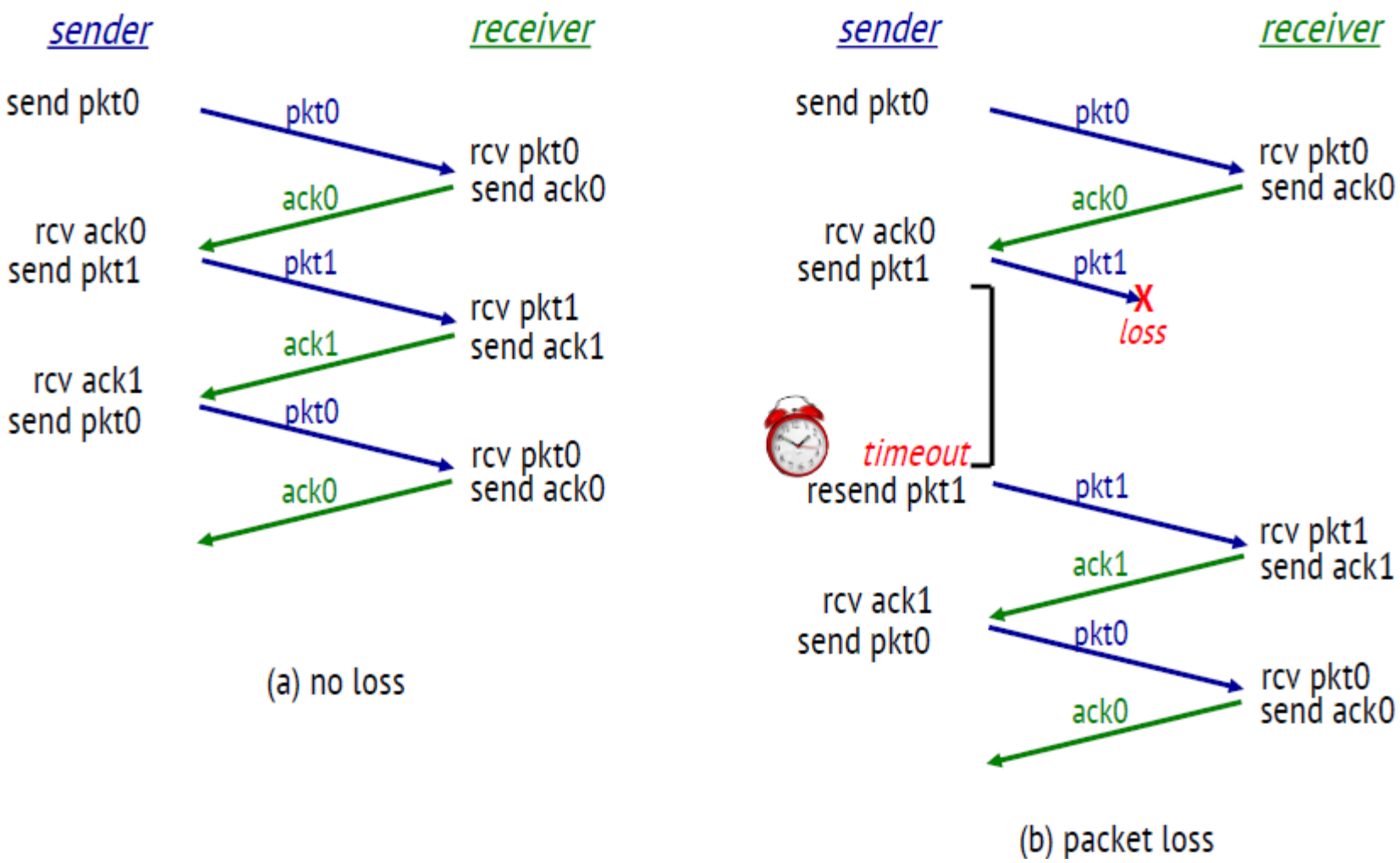
- Stop-and-Wait avec ARQ :
  - *L'émetteur envoi un paquet, puis s'arrête et attend.*
  - *Le récepteur répond par un ACK, si le paquet envoyé est reçu correctement.*
  - *En recevant le ACK, l'émetteur envoi le paquet suivant : Contrôle de flux simple*
  - *Au bout d'un certain temps, si l'émetteur ne reçoit pas le ACK, il retransmet le même paquet.*
  - *Aussi, si le récepteur détecte une erreur dans le paquet (Checksum, CRC, …), il envoi un négatif ACK (NACK), demandant à l'émetteur le renvoi du paquet*

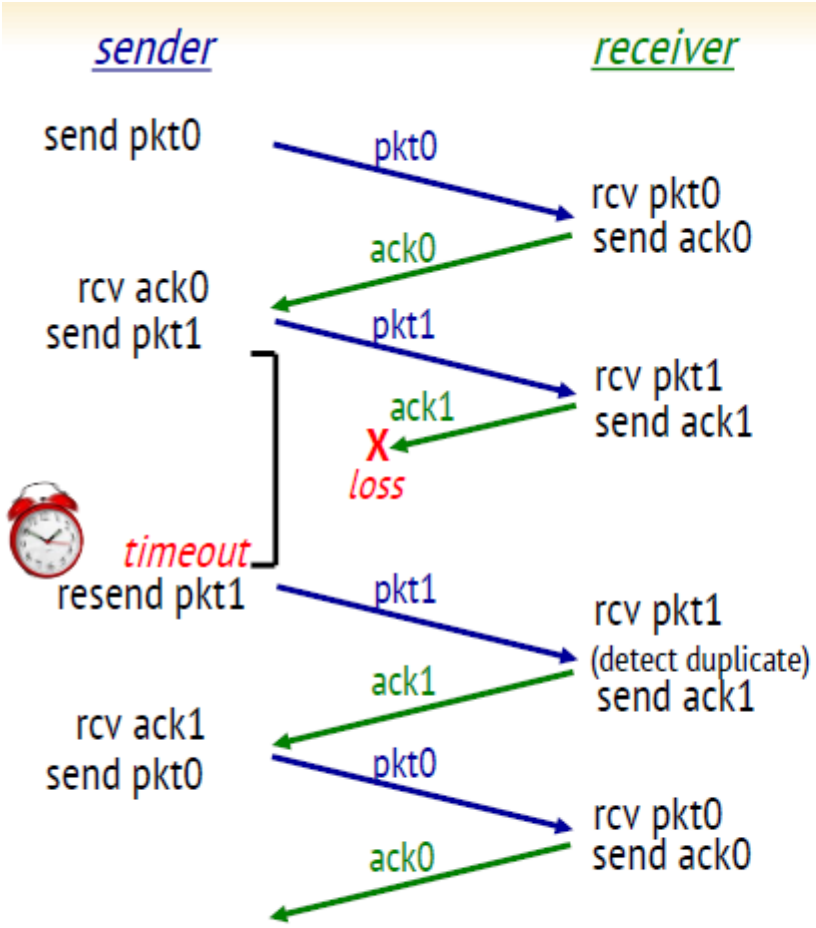
- Stop-and-Wait est un protocole simpliste et facile à implémenter.
- En matière de performance, il est très pauvre : Un paquet par RTT.
- La version de base ne numérote pas les séquences des paquets.
- La version Bit Alterné (ABP) réserve un bit pour la numérotaion (Les numéros de séquence sont Modulo 2).
- Contrôle de flux simple : Envoyer et attendre un ACK (Supposition : Le récepteur ne peut recevoir qu'un seul paquet)

Stop-and-Wait : FSM émetteur

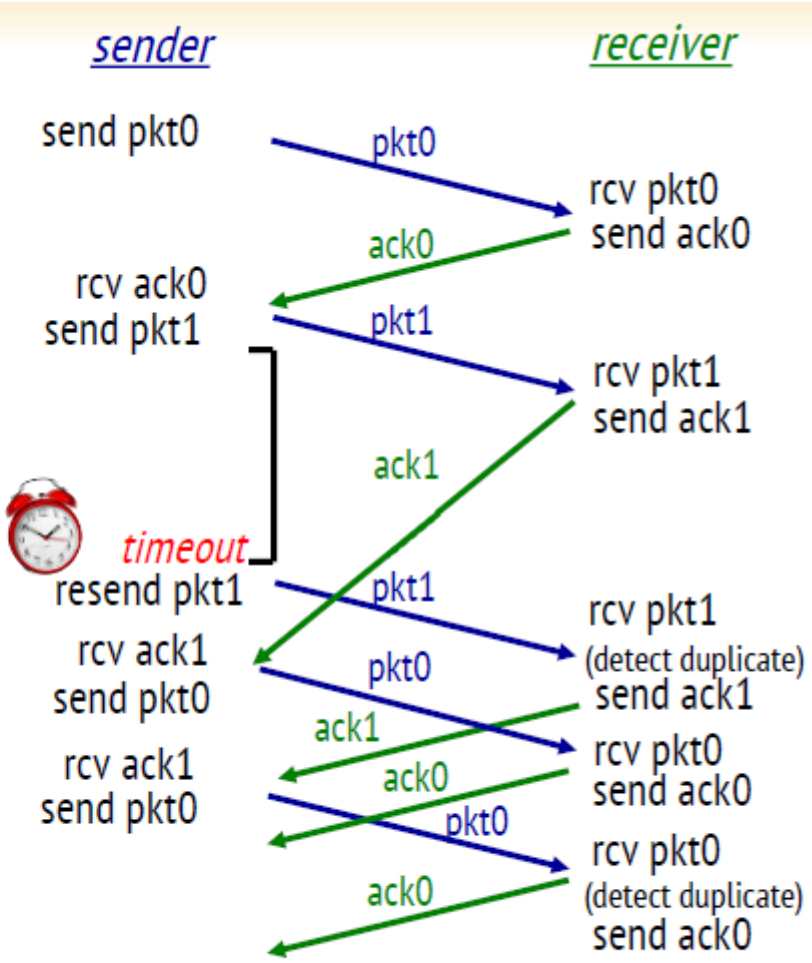


Stop-and-Wait





(c) ACK loss



(d) premature timeout/ delayed ACK

Stop-and-Wait : Performances

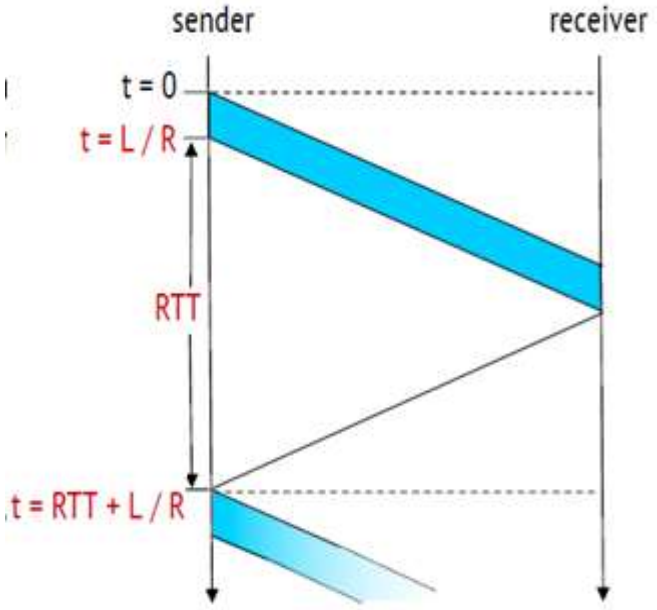
- U_{sender} du protocole Stop-and-Wait dans le cas d'un canal non bruité (sans érreur).
- $U_{sender} = d_{trans} / (2*d_{prop} + d_{trans}) = 1/(2\alpha+1)$; $\alpha = d_{prop}/d_{trans}$
- Exemple : Lien 1Gb/s , d_{prop} = 15ms , paquet = 1KB (8000 bit)

  $RTT = 2*d_{prop}$ = 30 ms

$$d_{trans} = \frac{L}{R} = \frac{8000 \text{ bits / packet}}{10^9 \text{ bits / sec}} = 8 \text{ microseconds}$$

$$U_{sender} = \frac{\frac{L}{R}}{RTT + \frac{L}{R}} = \frac{.008}{30.008} = 0.00027$$

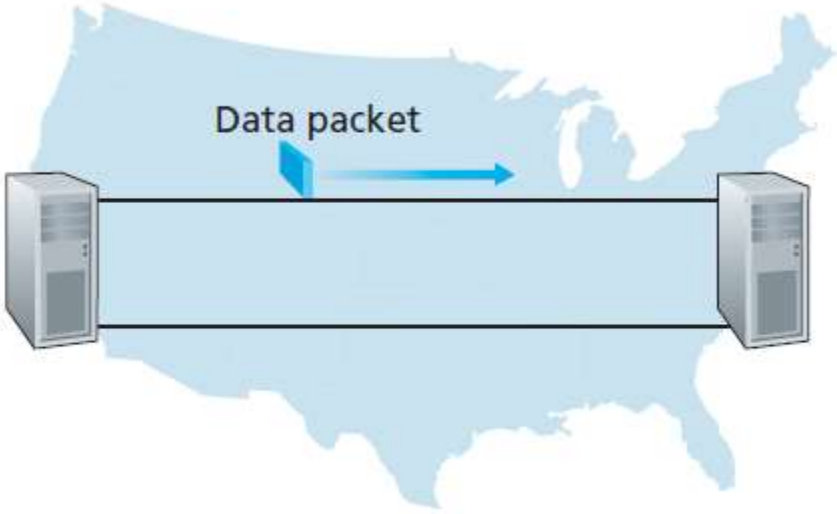
$$U_{sender} = 1/(2\alpha+1)$$



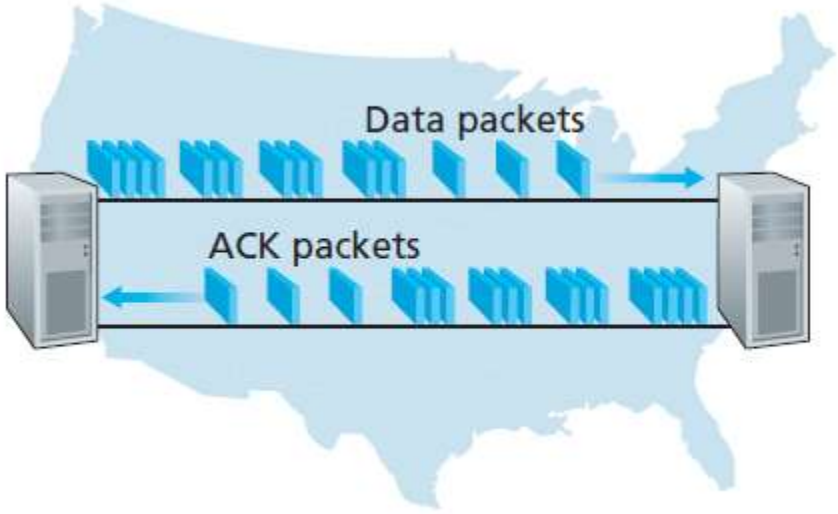
- L'émetur n'est actif que pendant 0.027% du temps ($d_{trans} << d_{prop}$)
- L'expéditeur n'a pu envoyer que 1000 Byte en 30.008 millisecondes, soit un débit effectif de 267 kbps seulement , en disposant d'une liaison de 1 Gbps !!!!!

Protocoles Pipelined

- L'émetteur peut envoyer plusieurs paquets sans attendre l'acquittement.
- L'intervalle des numéros de séquence doit être élargie (Stop-and-Wait : Seq=0 ou 1)
- Mise en mémoire tampon (Buffering) coté émetteur et/ou coté récepteur.
- Deux protocoles génériques pipelined : Go-Back-N et Selective-Repeat
- Controle de flux dans les protocoles : Sliding Window

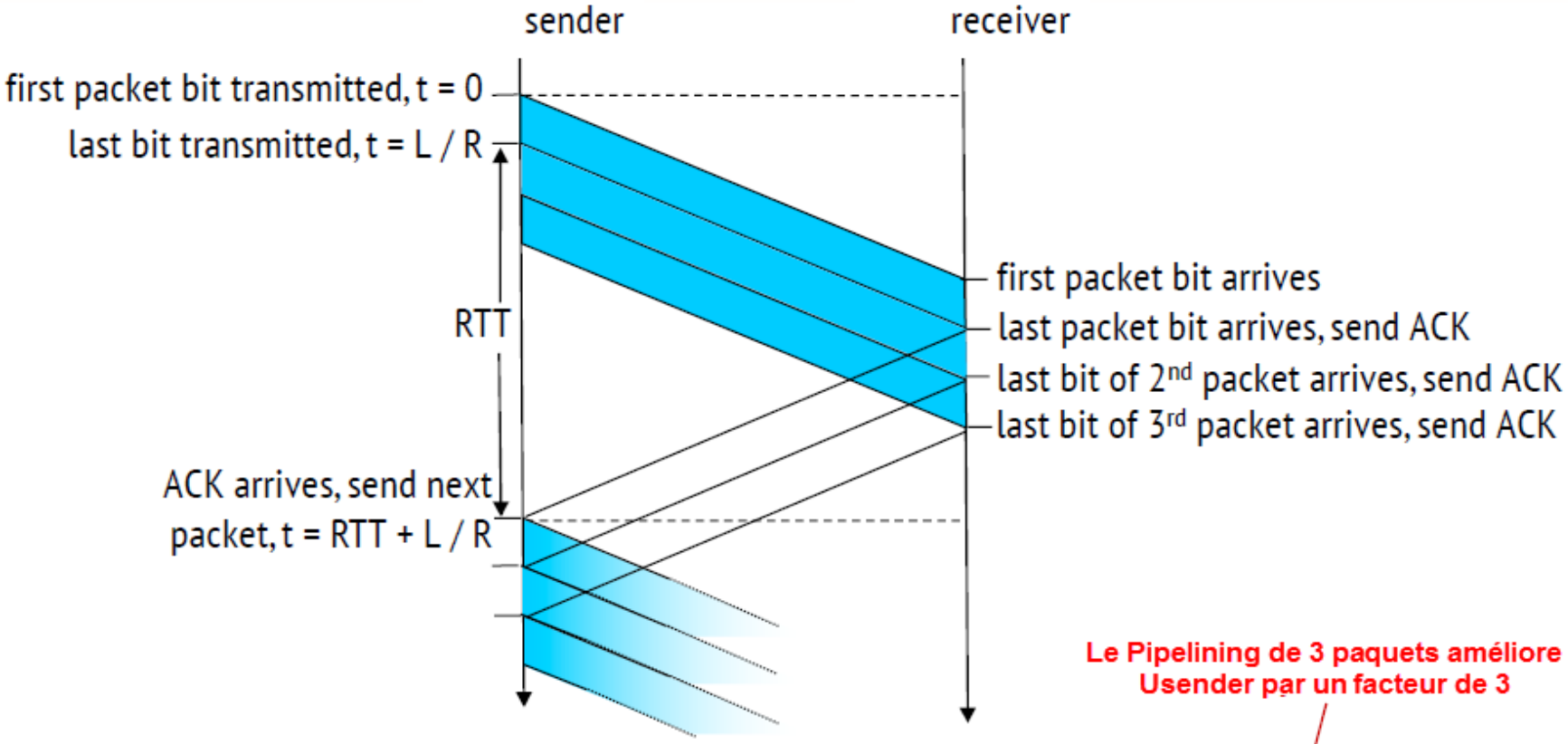


a. A stop-and-wait protocol in operation



b. A pipelined protocol in operation

Protocoles Pipelined : Amélioration de U_{sender}



Le Pipelining de 3 paquets améliore Usender par un facteur de 3

$$U_{sender} = \frac{3L/R}{RTT + L/R} = \frac{.0024}{30.008} = 0.00081$$

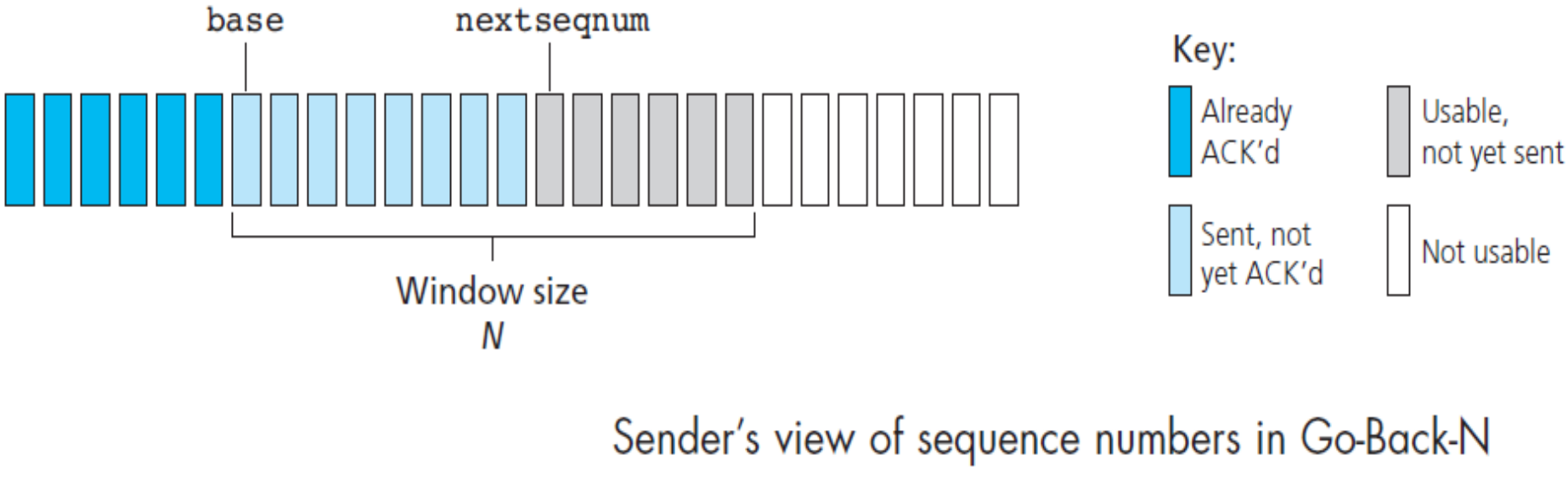
$$U_{sender} = 3/(2\alpha+1)$$

Go-Back-N (GBN)

- Go-Back-N : Protocole pipelined, Protocole a Fenêtre glissante (Sliding Window)
- Pour améliorer l'efficacité, il faut maximiser le BPD (Le nombre de paquets dans le pipe).
  - *L'émetteur envoi W paquets et attend le ACK*
  - *S'il n'y a pas d'erreurs il envoie les prochains paquets*
  - *En cas d'érreur, l'émetteur réenvoi tous les paquets à partir du numéro de ACK : Acquittement cumulatif*

    ACK3 signifie que le récepteur a reçu correctement le paquet 0, 1 et 2, et il est prêt à recevoir le paquet 3 et le reste des N paquets dans la fenêtre
  - *L'émetteur maintient un timer pour le plus ancien paquet non acquité, en cas de tmeout, l'émetteur réenvoi tous les paquets (Le plus ancien et ses successeurs)*
- W désigne la taille de la fenêtre glissante, nombre de paquet permis dans le pipe.
- Maximiser U $\Leftrightarrow W_{min}$ = BPD
- Il utilise des numéros de séquence et des numéros d'acquittement pour éviter la duplication de paquets.
- L'entête doit inclure un champs pour le numéro de séquence du paquet.
- Les numéros de séquence sont Modulo $2^m - 1$; *m désigne la taille du champ en bits.*
- Le numéro d'acquittement ACK est *cumulatif* et définit le numéro de séquence du prochain paquet attendu.
- Pourquoi imposer la limite W ?

Contrôle de flux et contrôle de congestion

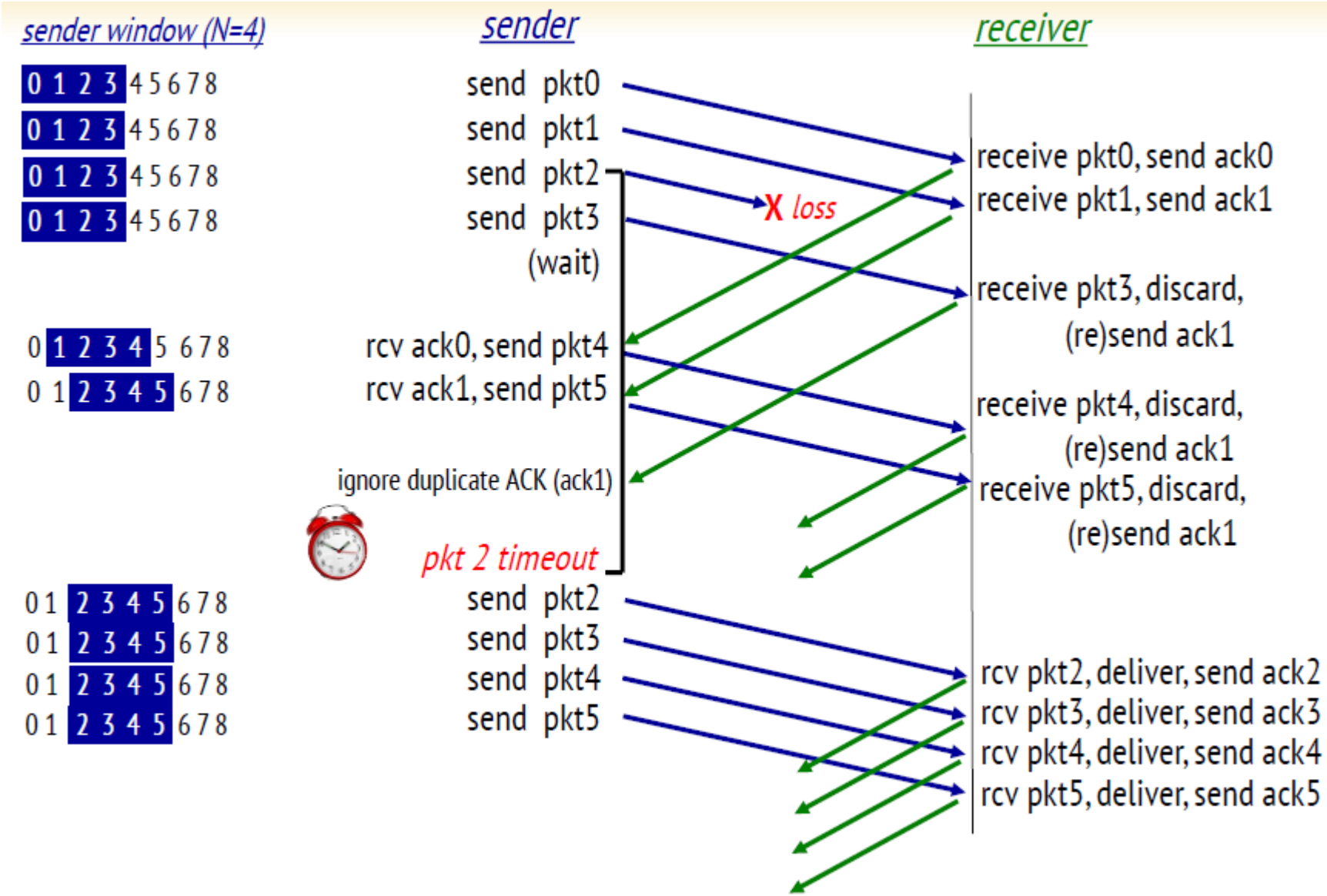


Sender's view of sequence numbers in Go-Back-N

Les intervalles de numéros de séquences

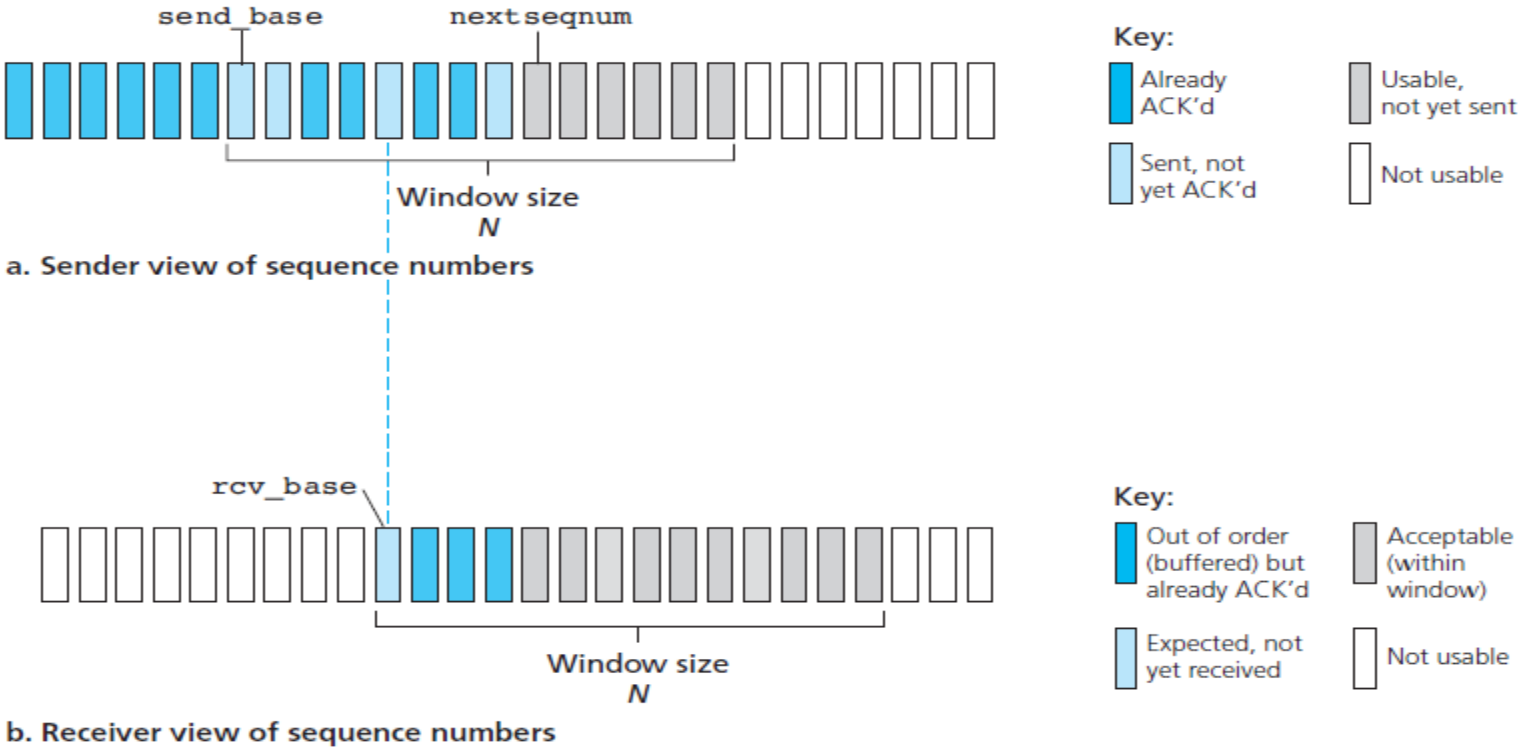
- $[0, base-1]$ correspond au paquets envoyés et acquités
- $[base, nextseqnum-1]$ paquets envoyés et non acquités
- $[nextseqnum, base+N-1]$ paquets pouvant être envoyés, dès leurs réception de la couche supérieure
- $[base+N, ..]$ paquets ne pouvant pas être envoyés, tanqu'il y'à des paquets non acquités dans le pipe.

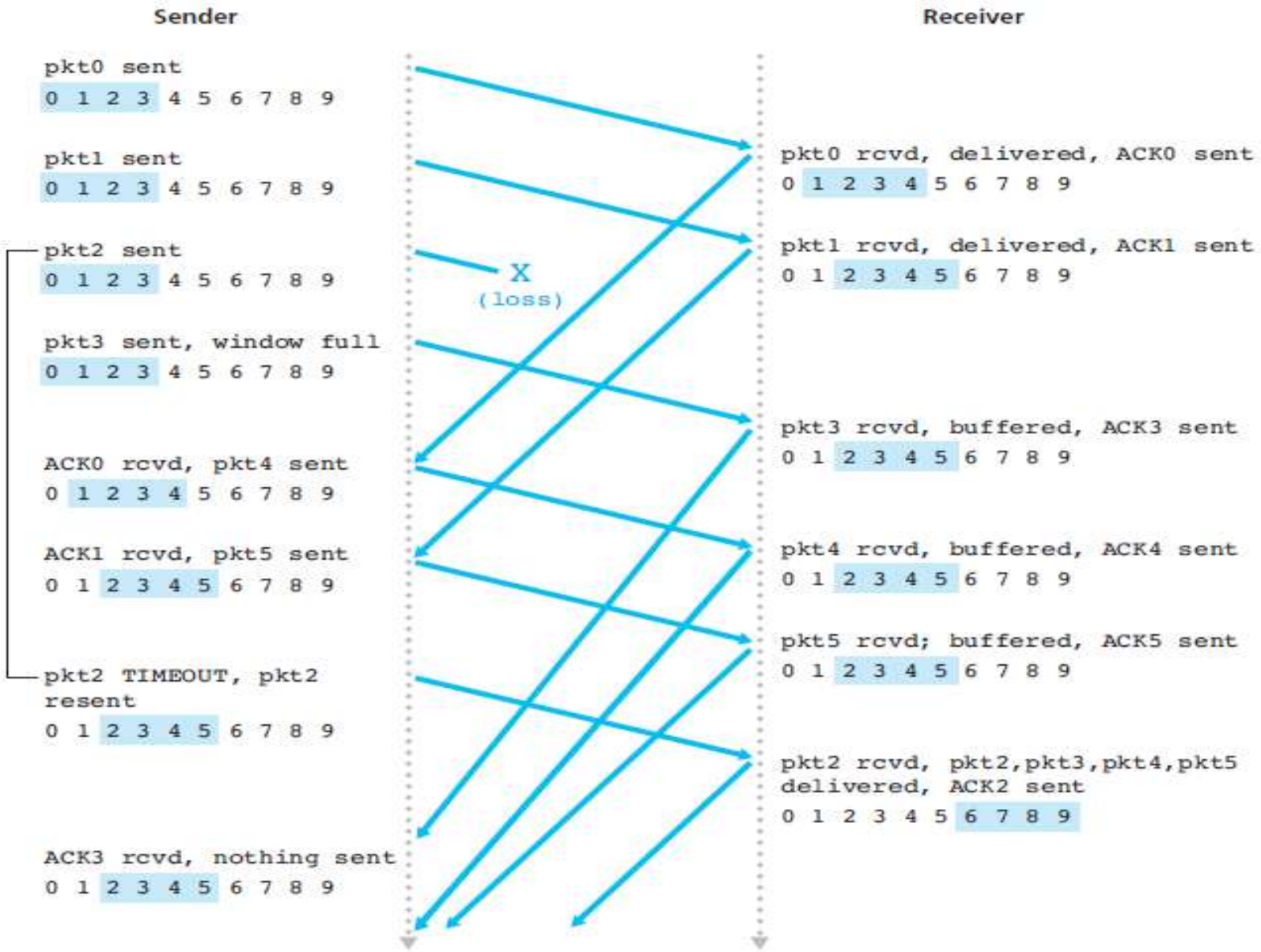
- Condition : $nextseqnum - base \leq N$


sender window (N=4)
sender
receiver
0 1 2 3 4 5 6 7 8
0 1 2 3 4 5 6 7 8
0 1 2 3 4 5 6 7 8
0 1 2 3 4 5 6 7 8
send pkt0
send pkt1
send pkt2
send pkt3
(wait)
X loss
receive pkt0, send ack0
receive pkt1, send ack1
receive pkt3, discard,
(re)send ack1
0 1 2 3 4 5 6 7 8
0 1 2 3 4 5 6 7 8
rcv ack0, send pkt4
rcv ack1, send pkt5
receive pkt4, discard,
(re)send ack1
receive pkt5, discard,
(re)send ack1
ignore duplicate ACK (ack1)
pkt 2 timeout
0 1 2 3 4 5 6 7 8
0 1 2 3 4 5 6 7 8
0 1 2 3 4 5 6 7 8
0 1 2 3 4 5 6 7 8
send pkt2
send pkt3
send pkt4
send pkt5
rcv pkt2, deliver, send ack2
rcv pkt3, deliver, send ack3
rcv pkt4, deliver, send ack4
rcv pkt5, deliver, send ack5

Selective Repeat (SR)

- Le principe de fonctionnement est similaire à GBN, sauf que l'émetteur ne renvoie que les paquets non acquités : *Acquittement sélectif*
  - *L'émetteur envoi W paquet et attend le ACK*
  - *Le récepteur acquite individuellement chaque paquet reçu*
  - *En cas d'érreur, l'émetteur réenvoi uniquement le paquet non aquité .*
  - *L'émetteur maintient un timer pour chaque paquet non acquité, en cas de tmeout, l'émetteur réenvoi tous le paquet .*




Sender
Receiver
pkt0 sent
0 1 2 3 4 5 6 7 8 9
pkt1 sent
0 1 2 3 4 5 6 7 8 9
pkt2 sent
0 1 2 3 4 5 6 7 8 9
X
(loss)
pkt3 sent, window full
0 1 2 3 4 5 6 7 8 9
ACK0 rcvd, pkt4 sent
0 1 2 3 4 5 6 7 8 9
ACK1 rcvd, pkt5 sent
0 1 2 3 4 5 6 7 8 9
pkt2 TIMEOUT, pkt2
resent
0 1 2 3 4 5 6 7 8 9
ACK3 rcvd, nothing sent
0 1 2 3 4 5 6 7 8 9
pkt0 rcvd, delivered, ACK0 sent
0 1 2 3 4 5 6 7 8 9
pkt1 rcvd, delivered, ACK1 sent
0 1 2 3 4 5 6 7 8 9
pkt3 rcvd, buffered, ACK3 sent
0 1 2 3 4 5 6 7 8 9
pkt4 rcvd, buffered, ACK4 sent
0 1 2 3 4 5 6 7 8 9
pkt5 rcvd; buffered, ACK5 sent
0 1 2 3 4 5 6 7 8 9
pkt2 rcvd, pkt2,pkt3,pkt4,pkt5
delivered, ACK2 sent
0 1 2 3 4 5 6 7 8 9

Comparaison de performances

Stop and Wait Flow Control: $U = 1/(1+2\alpha)$

Window Flow Control:

$$U = \begin{cases} 1 & W \geq 2\alpha+1 \\ W/(2\alpha+1) & W < 2\alpha+1 \end{cases}$$

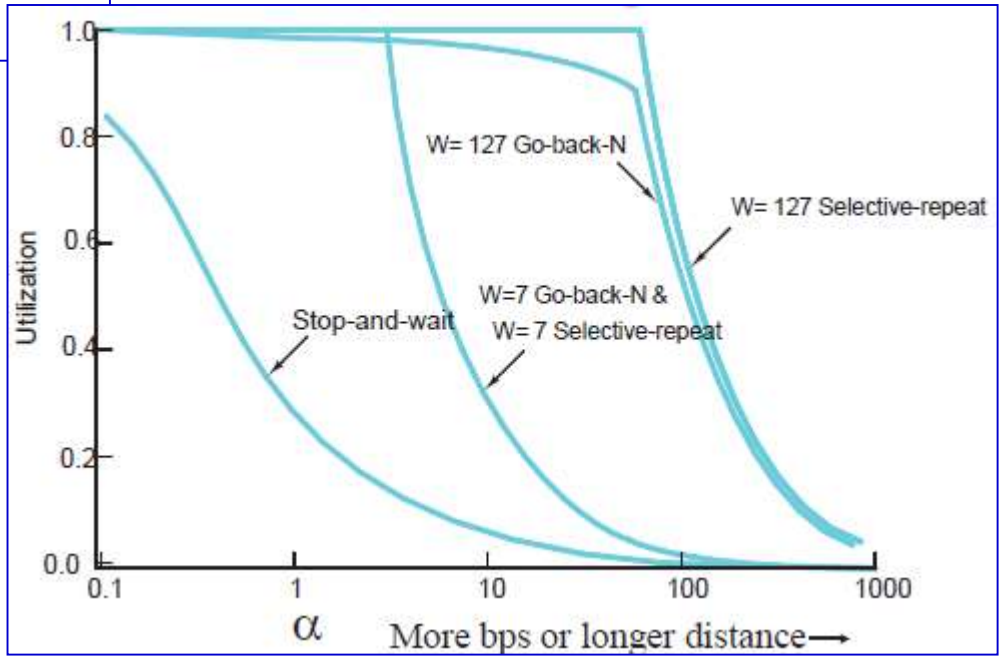
Stop and Wait ARQ: $U = (1-P)/(1+2\alpha)$

Go-back-N ARQ: P = Probability of Loss

$$U = \begin{cases} (1-P)/(1+2\alpha P) & W \geq 2\alpha+1 \\ W(1-P)/[(2\alpha+1)(1-P+WP)] & W < 2\alpha+1 \end{cases}$$

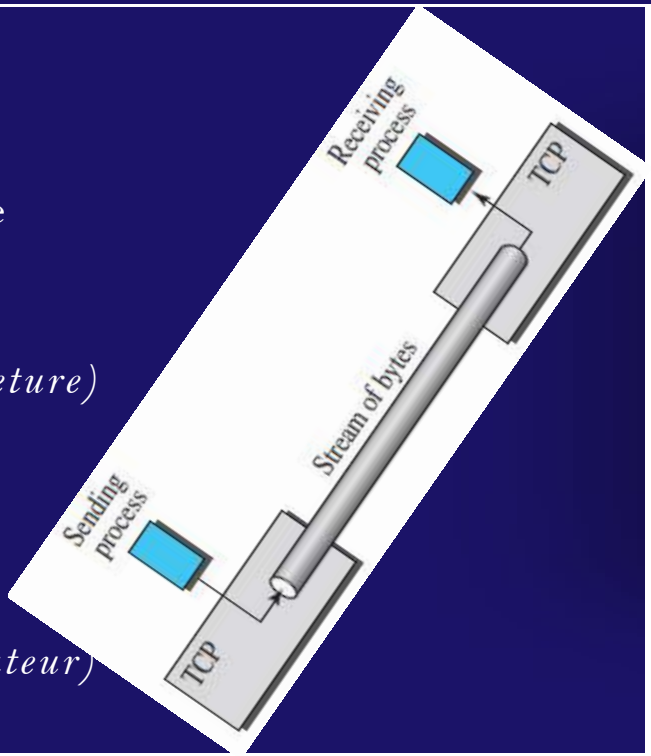
Selective Repeat ARQ:

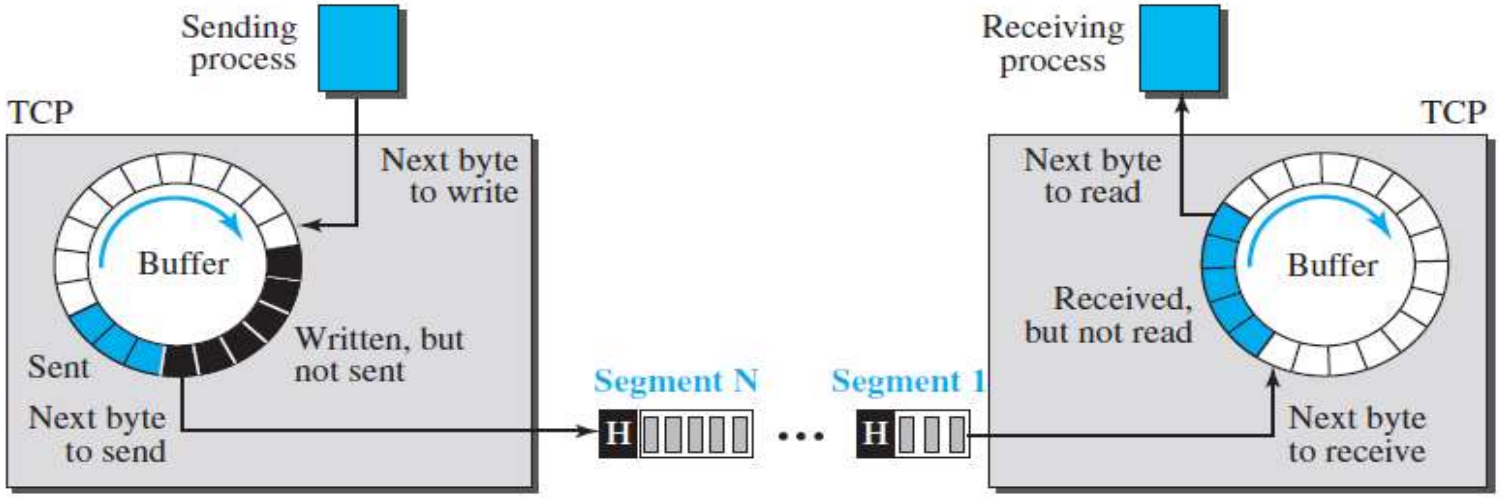
$$U = \begin{cases} (1-P) & W \geq 2\alpha+1 \\ W(1-P)/(2\alpha+1) & W < 2\alpha+1 \end{cases}$$



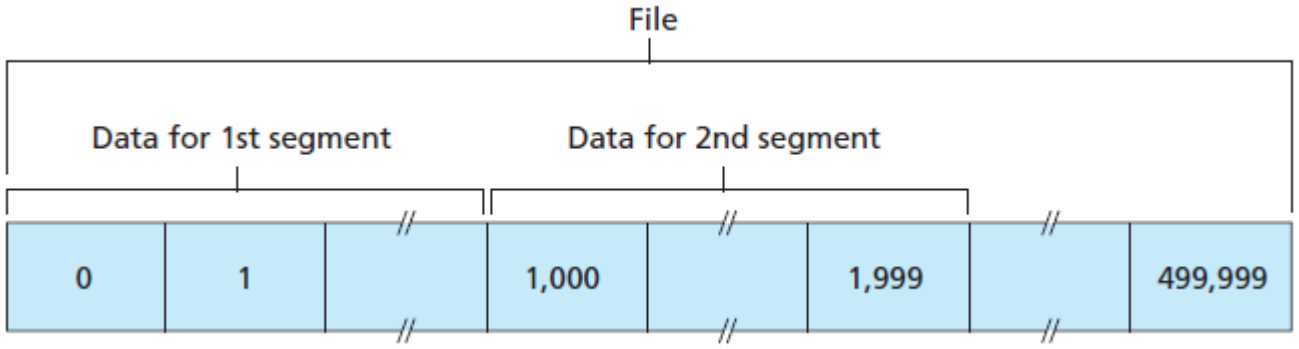
TCP (Transmission Control Protocol) : Connection-oriented transport Protocol

- PDU = Segment (Segmentation)
- Protocole ARQ basé sur Go-Back-N (Pipelining)
- Point-à-Point : Un emétteur et un récepteur
- Orienté flux d'octets (Byte Stream) : Pas de frontières dans le message
- Full duplex : Une connexion pour un flux de données bi-directionnel
  - ↳ *MSS (Maximun Segment Size) : 1460, 1448, 512, …*
- Orienté connexion : *Gestion de la connexion (Ouverture et Fermeture)*
- Multiplexage/Démultiplexage
- Transfert de données fiable
  - *Detection d'érreurs (Bit cheking checksum)*
  - *Livraison des segments en ordre (Numéro de séquence)*
  - *Contrôle d'érreurs (Acquitements, Retransmission, Temporisateur)*
- Contrôle de flux et Contrôle de congestion : Fenêtre glissante

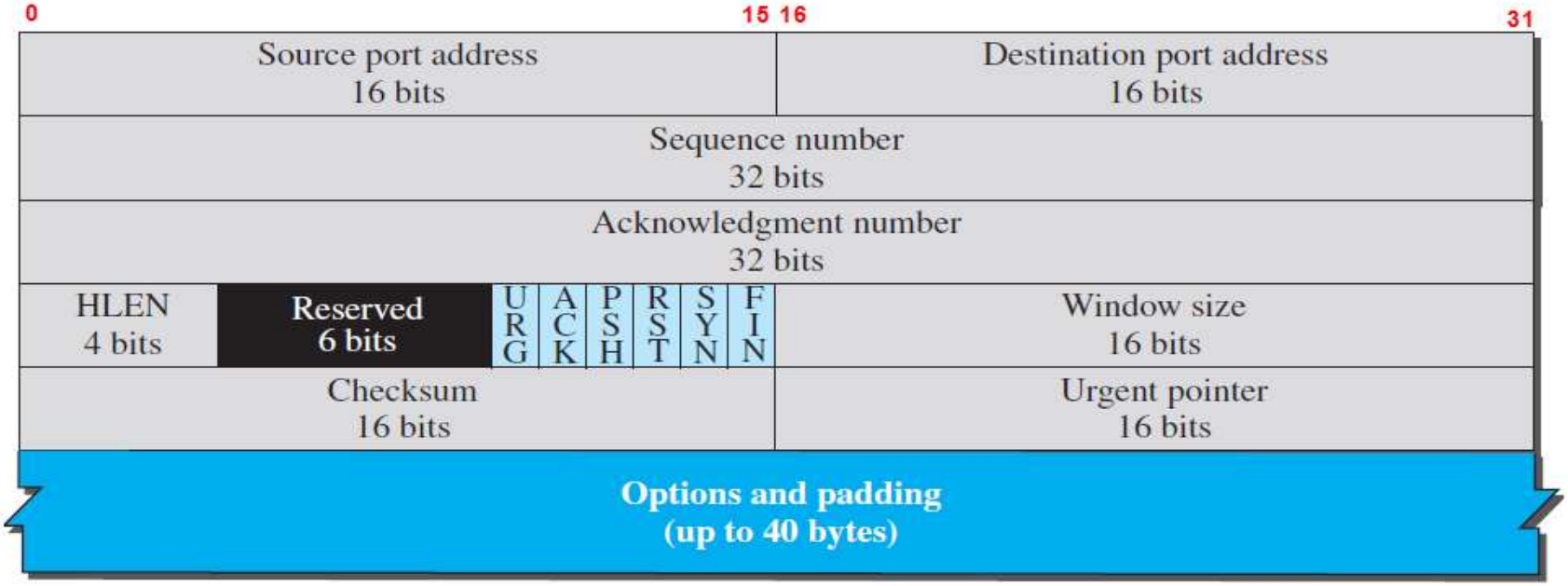




TCP : Segment



Segment



Header

TCP : Segment

- *Source Port* : Champ de 16 bits qui définit le numéro de port de l'application source (Ex. Navigateur Web Google Chrome)
- *Destination Port* : Champ de 16 bits qui définit le numéro de port de l'application destination (Ex. Seveur Web nginx).
- *Sequence Number* : Le champ numéro de séquence de 32 bits et le champ numéro d'accusé de réception de 32 bits sont utilisés pour implémenter le RDT. TCP est un protocole de transport orienté flux d'octets. Pour assurer la connectivité, chaque octet à transmettre est numéroté. Le numéro de séquence est le numéro du premier octet du segment. Lors de l'établissement de la connexion, les entités paires utilise un générateur de nombres aléatoires pour créer un numéro de séquence initial (ISN), qui est généralement différent dans chaque direction.
- *Acknowledgement Number* : Indique le prochain numéro d'octet a recevoir. Si le récepteur a reçu avec succès l'octet numéro x , il renvoie $x + 1$ comme numéro d'accusé de réception. L'accusé de réception et les données peuvent être combinés (Piggybacking).
- *Window Size* : Le champ taille de la fenêtre de 16 bits est utilisé pour le contrôle de flux. Il est utilisé pour indiquer le nombre d'octets qu'un récepteur est prêt à recevoir.
- *HLEN* : Longueur de l'entete exprimée en mot de 32 bits. En fonction du champ options, Hlen peut varier entre 5 et 15 mots.
- *Checksum* : Idem que UDP

TCP : Segment

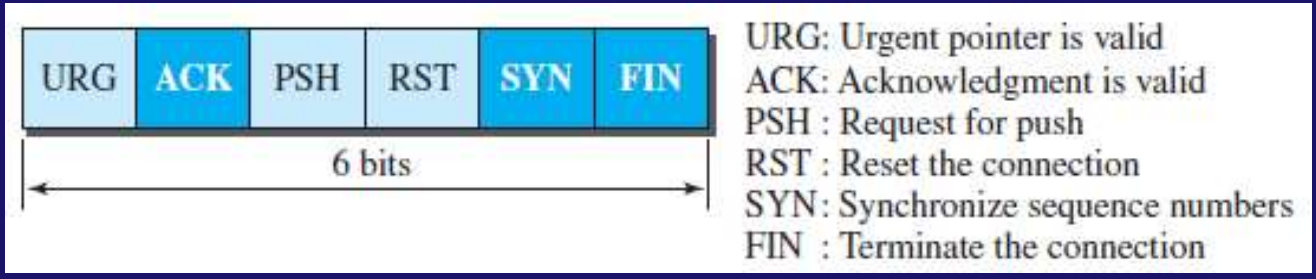
- *Flags (Indicateurs Flags) : Champ de 6 bits.*

  ACK : Accuse réception du dernier segment recu avec succès.

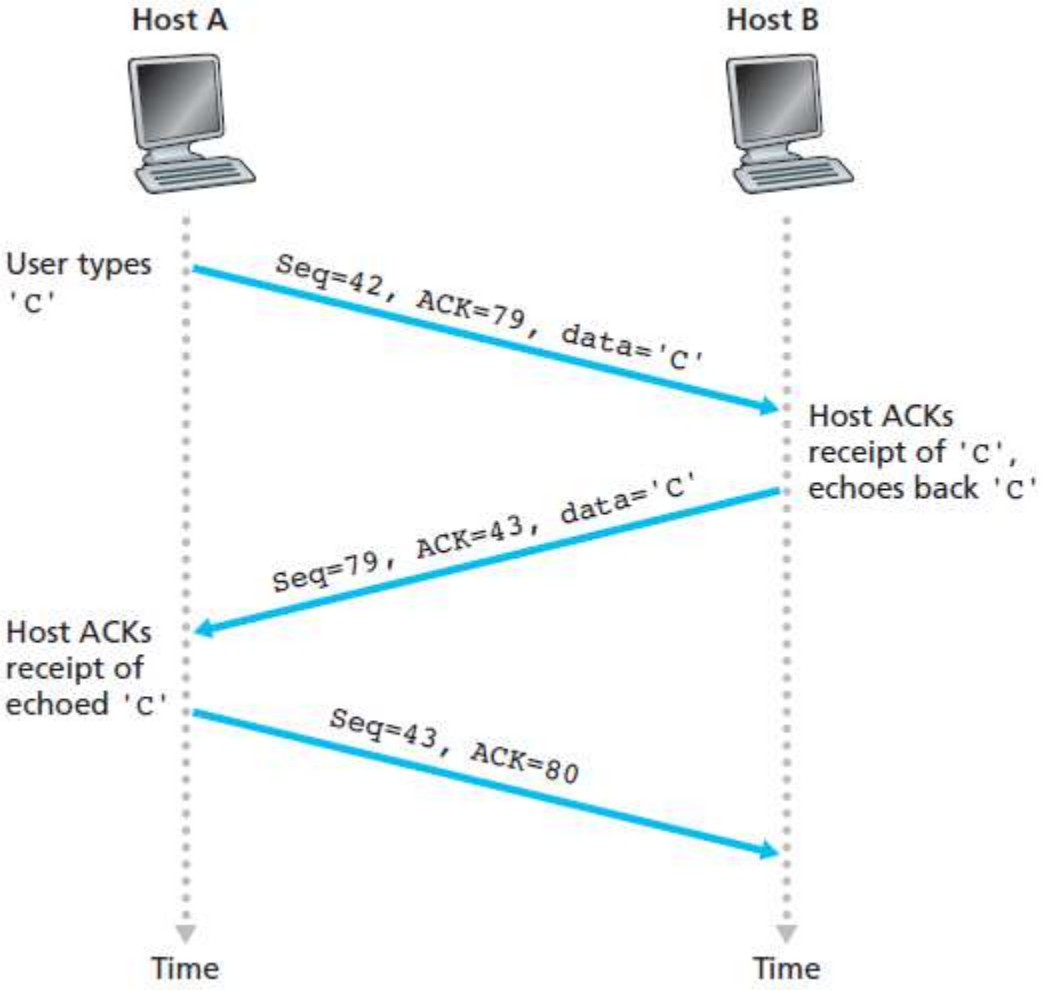
  SYN, RST, FIN : Sont utilisés pour la gestion de la connexion (Ouverture, Fermeture, Réinitialiser)

  PSH : Indique au récepteur qu'il doit remettre immédiatement les donnée à la couche supérieure.

  URG : Utilisé pour indiquer que l'entité de la couche supérieur coté émetteur que le segment contient des données urgentes. La position du dernier octet de ces données est obtenue à partir de la valeur du champ Urgent Pointer.



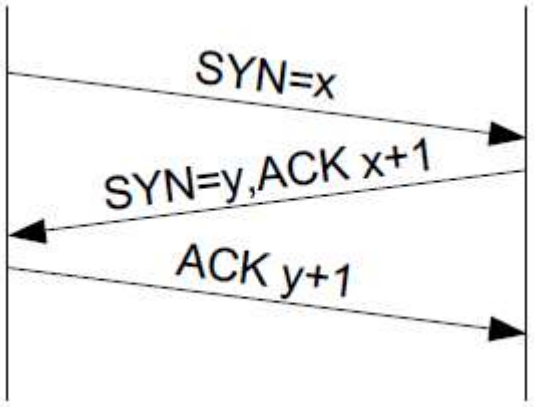
- *Urgent Pointer : Il définit une valeur qui doit être ajoutée au numéro de séquence pour obtenir le numéro du dernier octet urgent .*
- *Options : Champ utilisé pour la négotiation de la taille de la fenêtre (MSS), Timestamp pour le calcul du RTT, …*



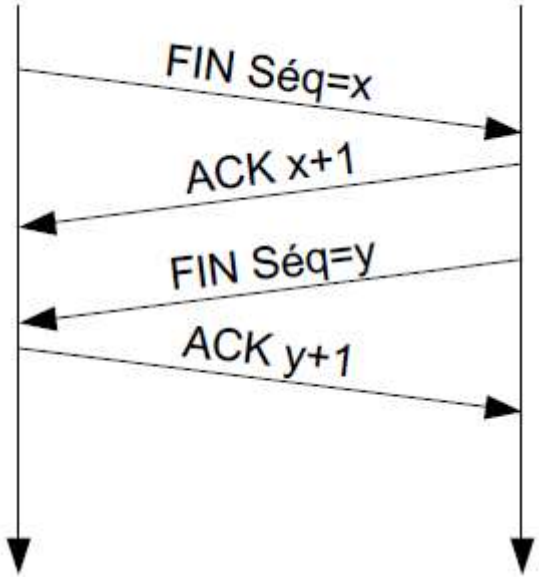
Exemple : Simple application Telnet

TCP : Gestion de la connexion

- Connexion : Etablissement de la connexion en trois phases ou étapes (Three-Way-Handshake)
- Libération : Libération de la connexion dans les deux sens par les deux cotés.

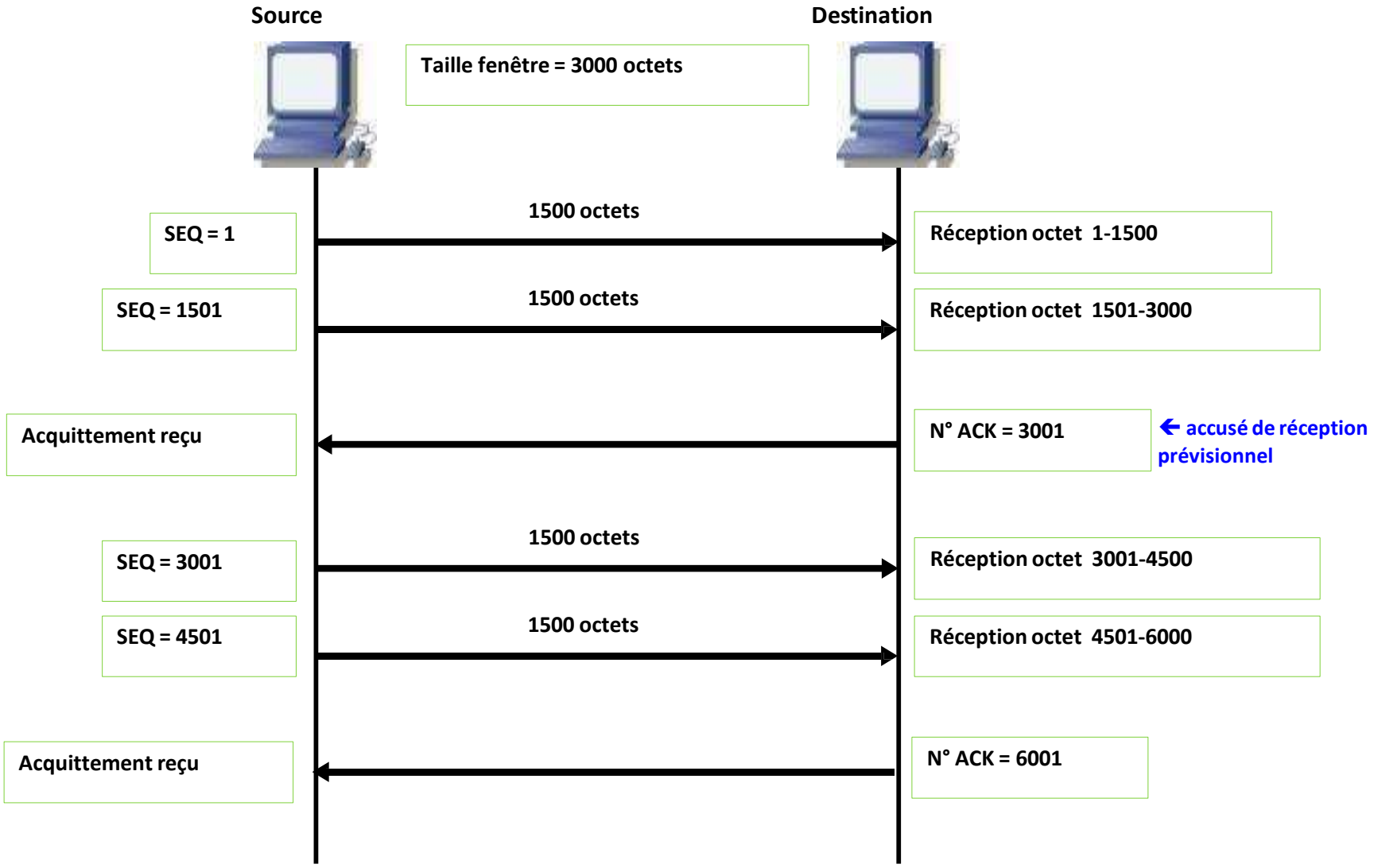


Connexion

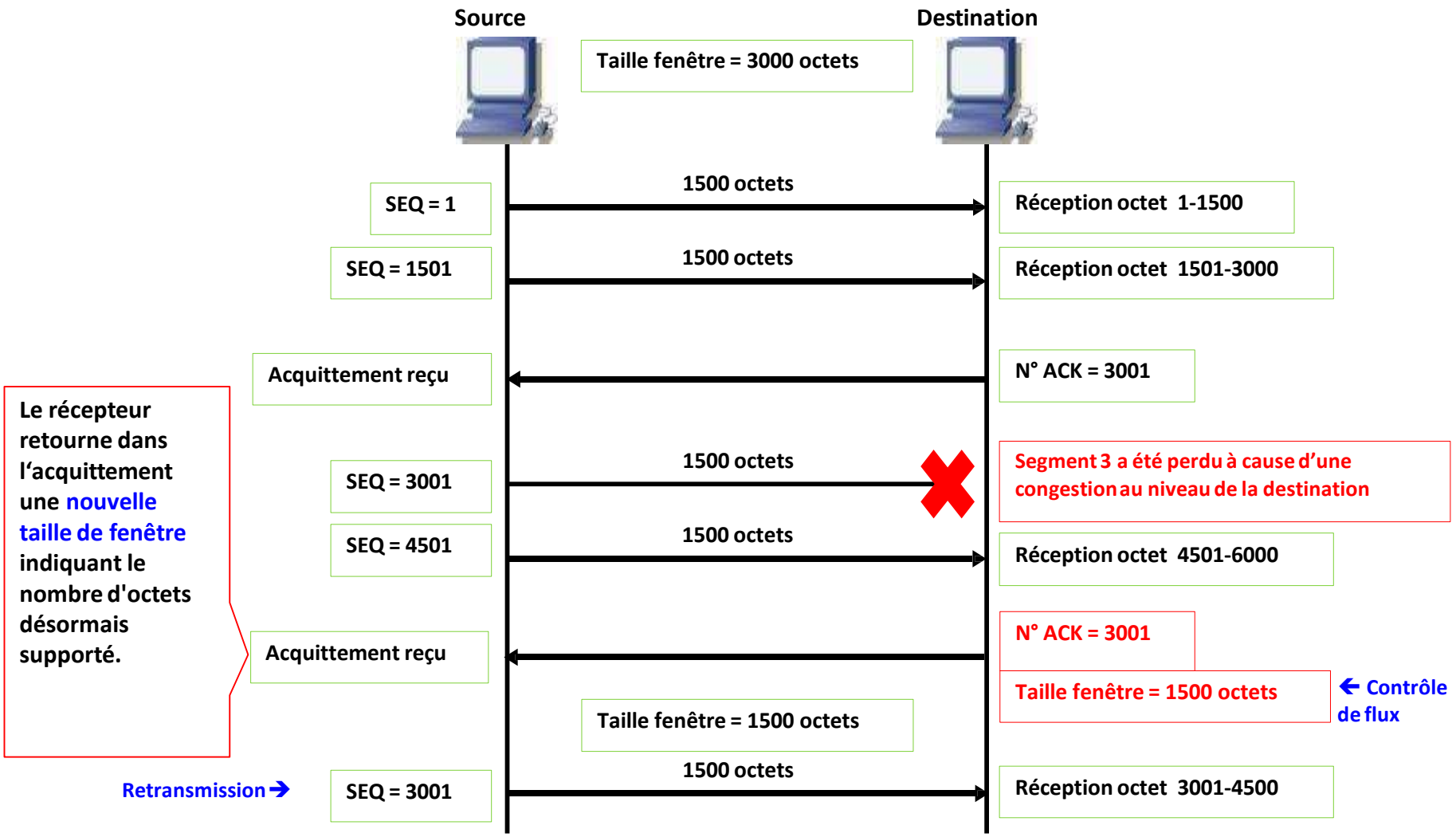


Libération

TCP : Contrôle de flux et Retransmission

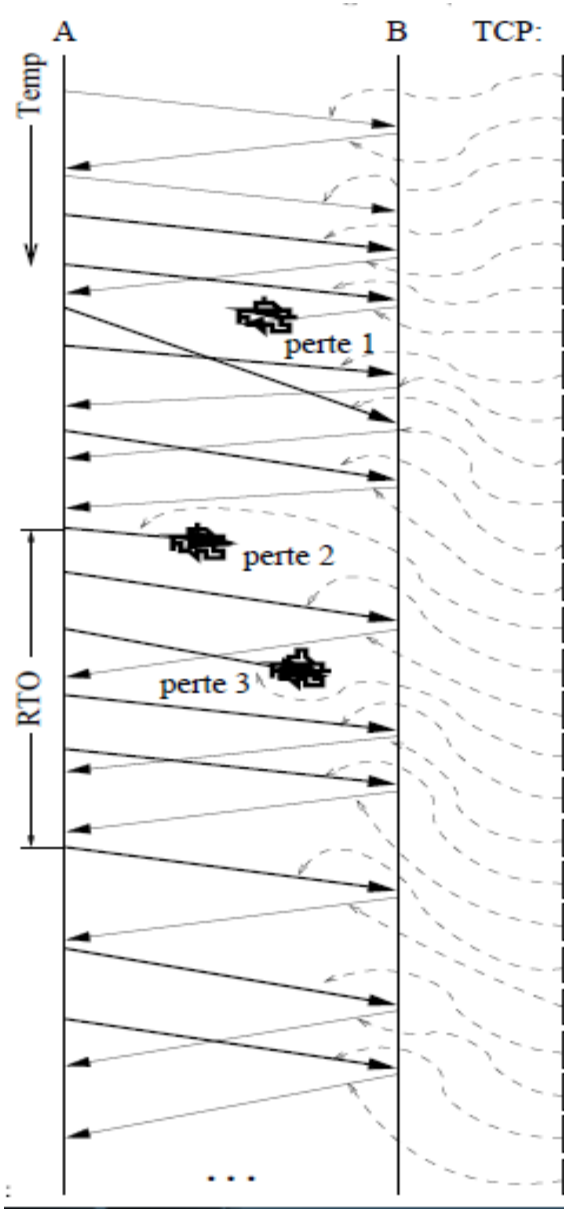


TCP : Contrôle de flux et Retransmission



Exemple : Considérons le transfert d'un fichier de taille illimittée sur une connexion TCP. La taille maximum des données d'un segment (MSS) est 200 Octets. La transmission démarre, complétez les paramètres associés à chaque segment (Numéro de séquence, Numéro d'acquitement, flags, Taille des données) dans le diagramme d'échange.

TCP : Contrôle de flux et Retransmission



TCP: No Sequence	No Acquit.	U	A	P	R	S	F	Taille données
3999	–						x	0
122	4000		x				x	0
4000	123		x					0
4000	123		x					200
123	4200		x					0
4200	123		x					200
123	4400		x					0
4600	123		x					200
123	4400		x					0
4400	123		x					200
123	4800		x					0
4800	123		x					200
123	5000		x					0
5000	123		x					200
5200	123		x					200
123	5000		x					0
5400	123		x					200
5600	123		x					200
123	5000		x					0
5800	123		x					200
123	5000		x					0
5000	123		x					200
123	5400		x					0
5400	123		x					200
123	6000		x					0
5600	123		x					200
123	6000		x					0

TCP : Estimation du RTO (Retransmission Time-Out)

- Comment estimer et choisir le RTO :

  RTO petit ⇒ Transmission inutiles

  RTO grand ⇒ Délai d'attente, avant retransmission des paquets perdus, élevé

$$EstimatedRTT = (1-\alpha)\, EstimatedRTT + \alpha\, .SampleRTT$$
$$DevRTT = (1-\beta)\, DevRTT + \beta\, .|SampleRTT - EstmatedRTT|$$
$$RTO = EstimatedRTT + 4\, .DevRTT$$

Si Expiration du RTO avant acquitement RTO = 2 * RTO

- EstimatedRTT est la moyenne pondérée des RTT mesurés (Estimated) et ses valeurs précédentes.
- DevRTT est la déviation de SampleRTT du EstimatedRTT
- Les valeurs initiales de SampleRTT et DevRTT peuvent être fixés ou prises aléatoirement
- Les valeurs de α et β dépendent des implémentations.
- Généralement $\alpha = 1/8$ et $\beta = 1/4$

Exemple : RTT_S = EstimatedRTT

RTT_M = SampleRTT

RTT_D = DevRTT

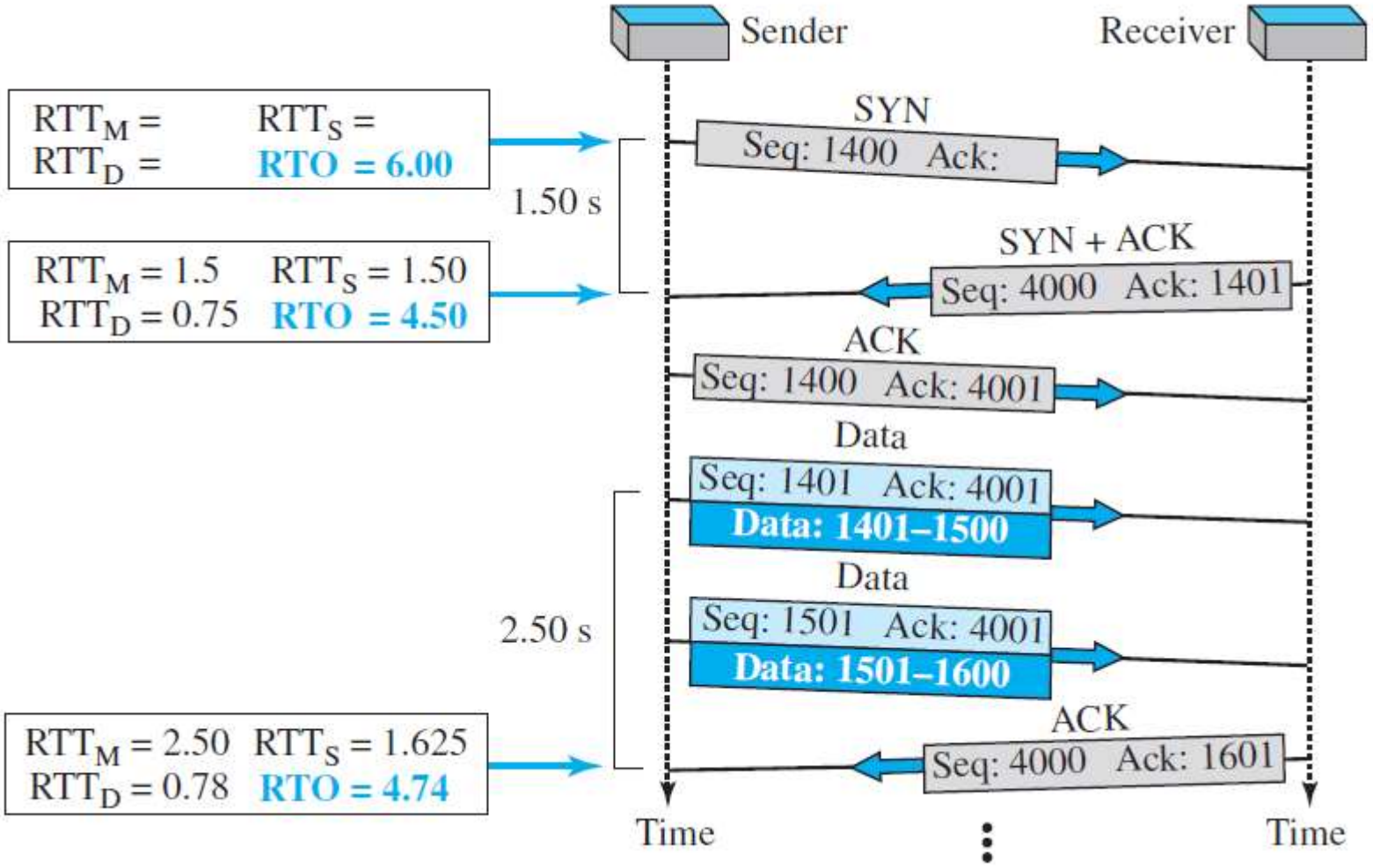
Après l'envoi du premier SYN les valeurs de ces variables sont inconnues, et RTO prend la valeur de 6s.

Une fois le RTT_M est mesuré (1.5s sur la figure), on prend :

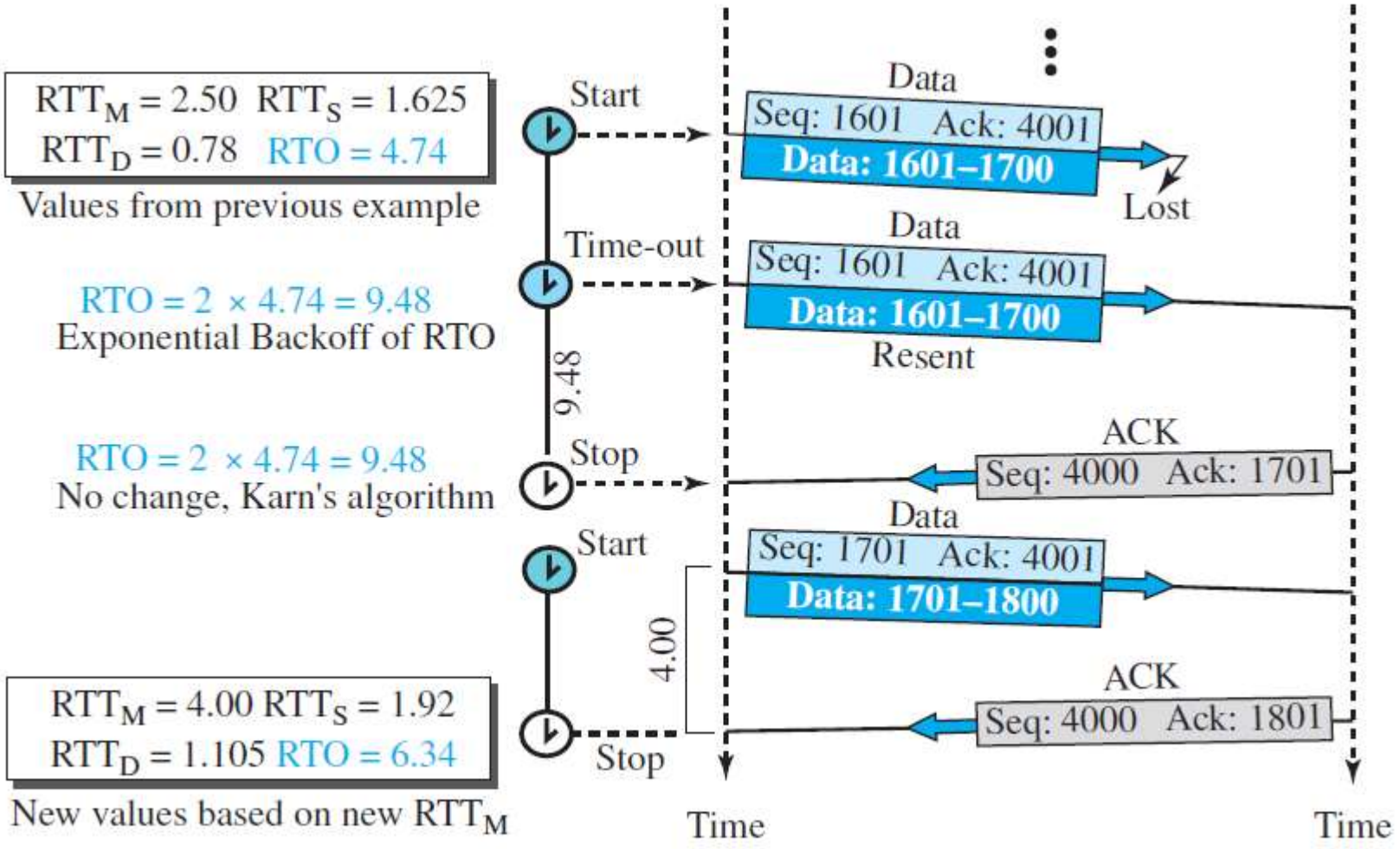
$RTT_S = RTT_M$

$RTT_D = 1/2\ RTT_M$

TCP : Estimation du RTO (Retransmission Time-Out)

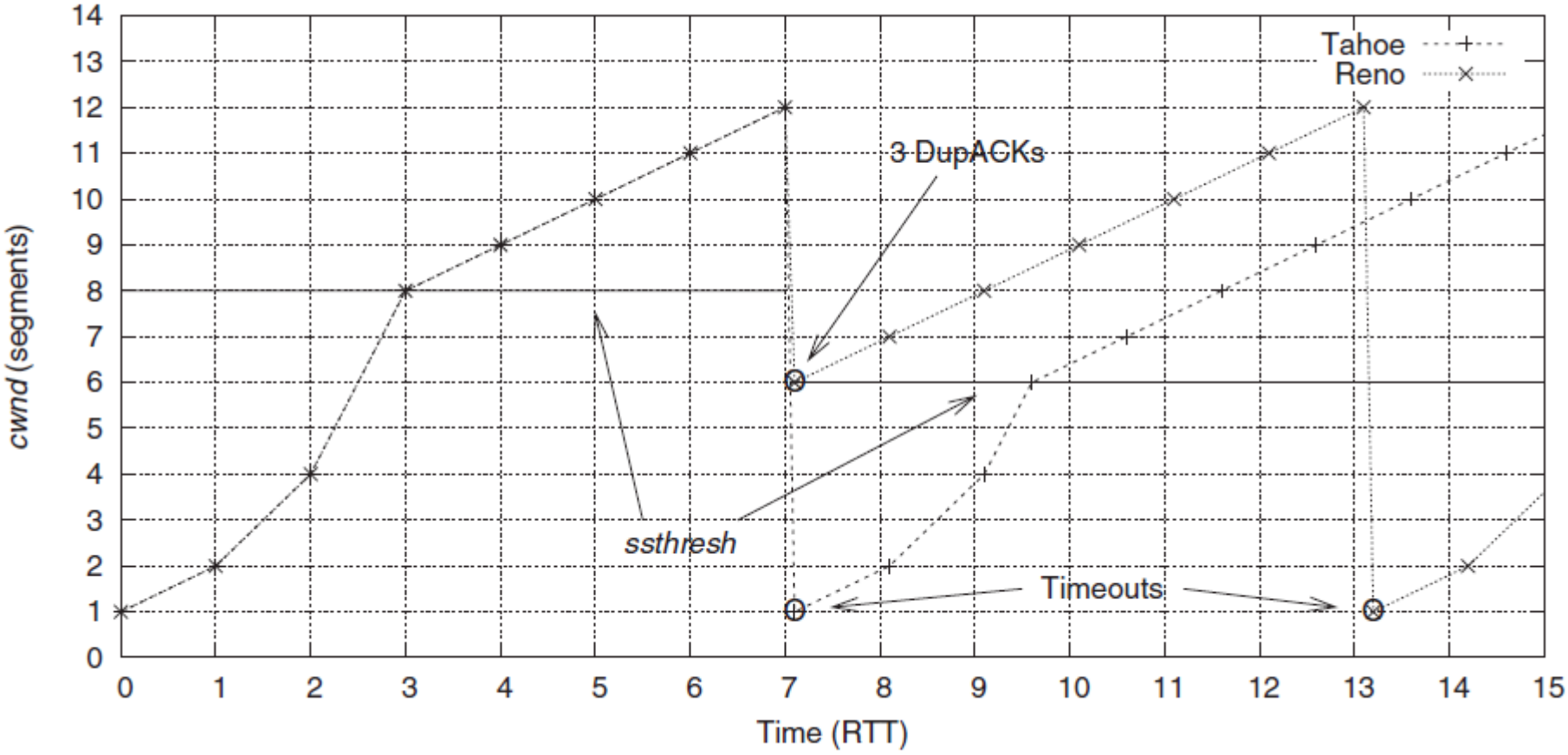


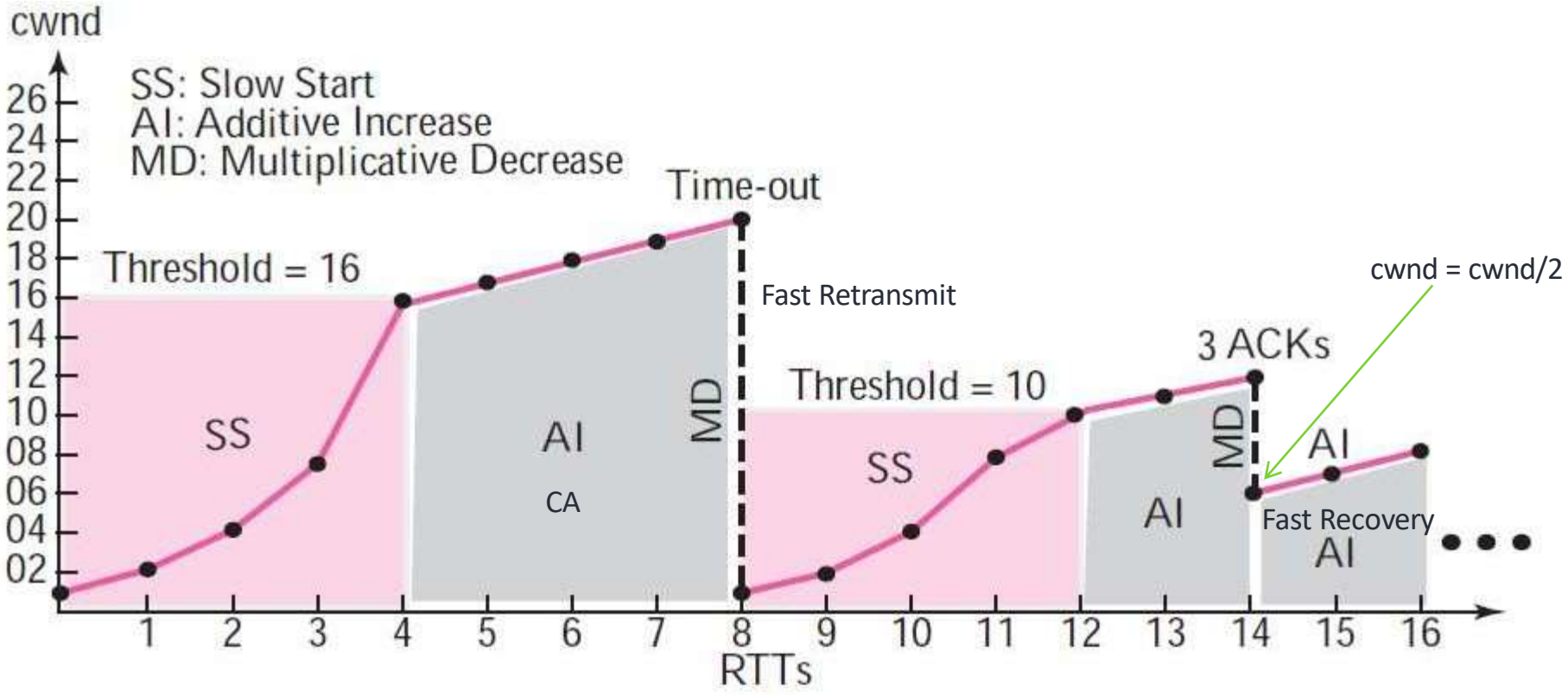
TCP : Estimation du RTO (Retransmission Time-Out)



TCP : Contrôle de Congestion

- Le contrôle de congestion dans TCP est basé principalement sur quatre procédures (Algorithmes).
- Slow Start (SS), Congestion Avoidance (CA), Fast Reovery, Fast Retransmit.
- La taille de la fenêtre de l'émetteur dans le contrôle de flux est controlé par la fenêtre du récepteur (*rwind*)
- Dans le contrôle de congestion, une autre fenêtre est utilisée : Fenêtre de congestion (*cwind*).
- Taille de la fenêtre TCP actuelle = Min(rwind , cwind)

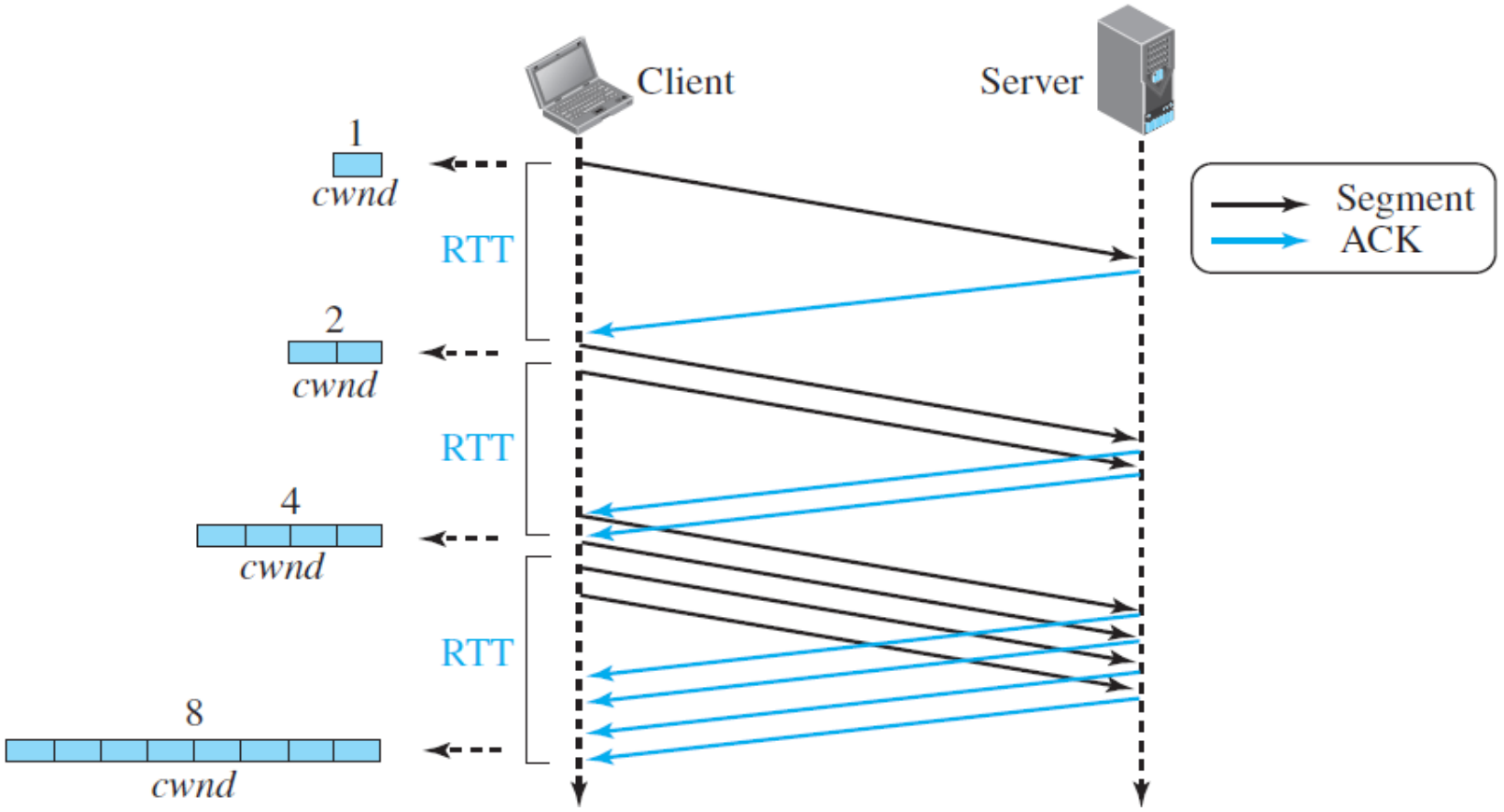

Tahoe
Reno
3 DupACKs
ssthresh
Timeouts
cwnd (segments)
Time (RTT)
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15


cwnd
SS: Slow Start
AI: Additive Increase
MD: Multiplicative Decrease
Time-out
Threshold = 16
Fast Retransmit
cwnd = cwnd/2
3 ACKs
Threshold = 10
SS
AI
CA
MD
SS
AI
MD
AI
Fast Recovery
AI
26
24
22
20
18
16
14
12
10
08
06
04
02
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
RTTs

TCP : Contrôle de Congestion

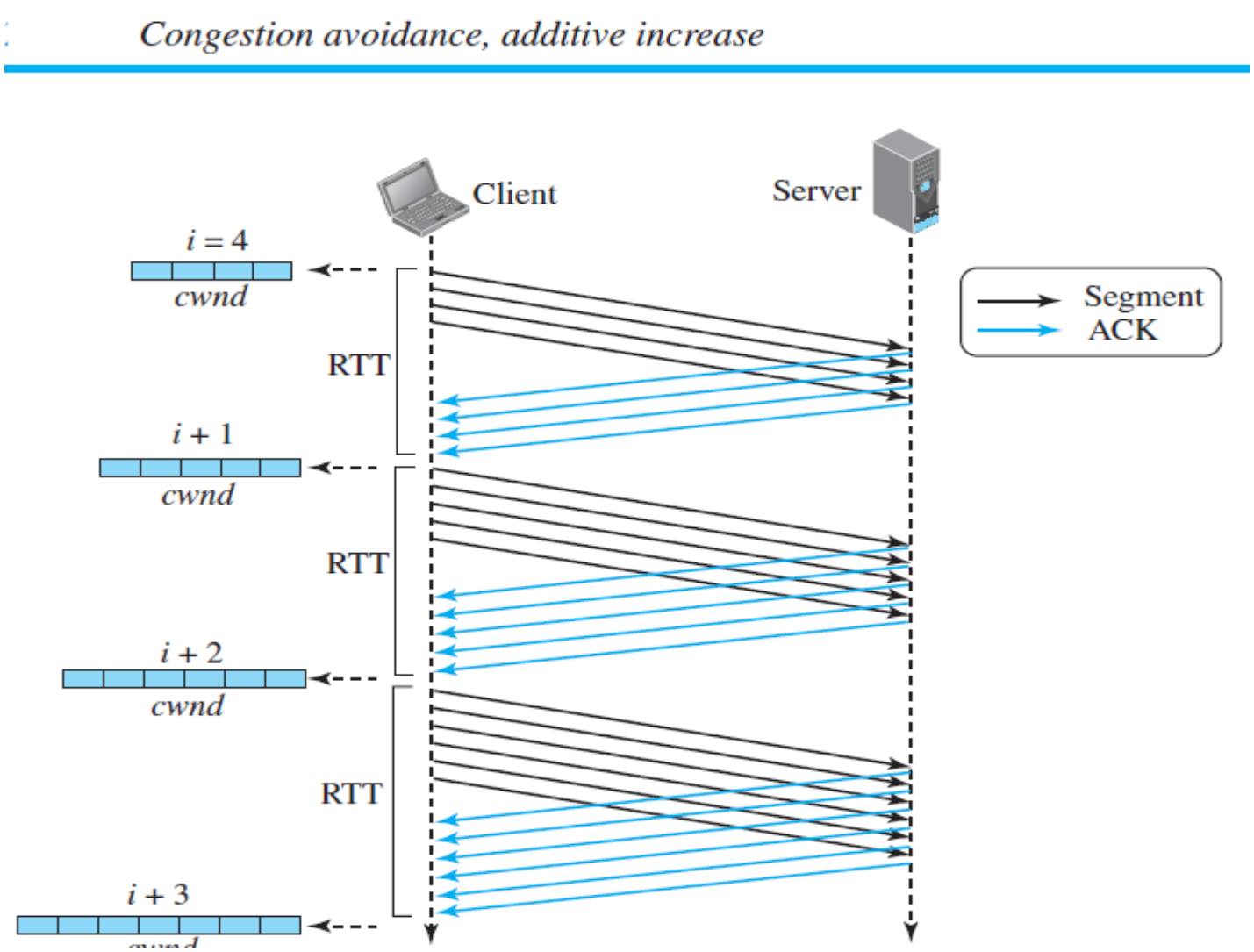
- Slow Start : Exponential Increase (Accroissement exponentielle)
  Si ACK Alors cwnd = cwnd + 1
  En terme de RTT cwnd croit exponentiellement : 2^n
- L'sccroissement ne peut être illimité, il est bornée par une variable nommée *ssthres* (slow-start threshold)

Slow start, exponential increase



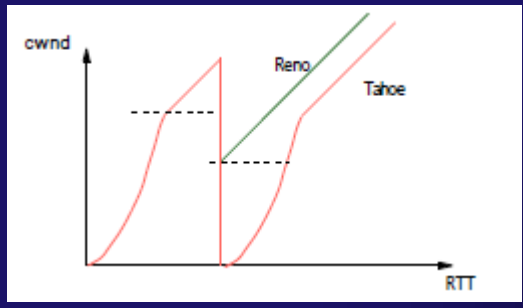
TCP : Contrôle de Congestion

- Lorsque cwnd attent threshold le SS s'arrete et commence la phase CA avec un accroissement incrémentale (AI : Additive Increase).
- En terme de RTT : cwnd = cwnd + 1



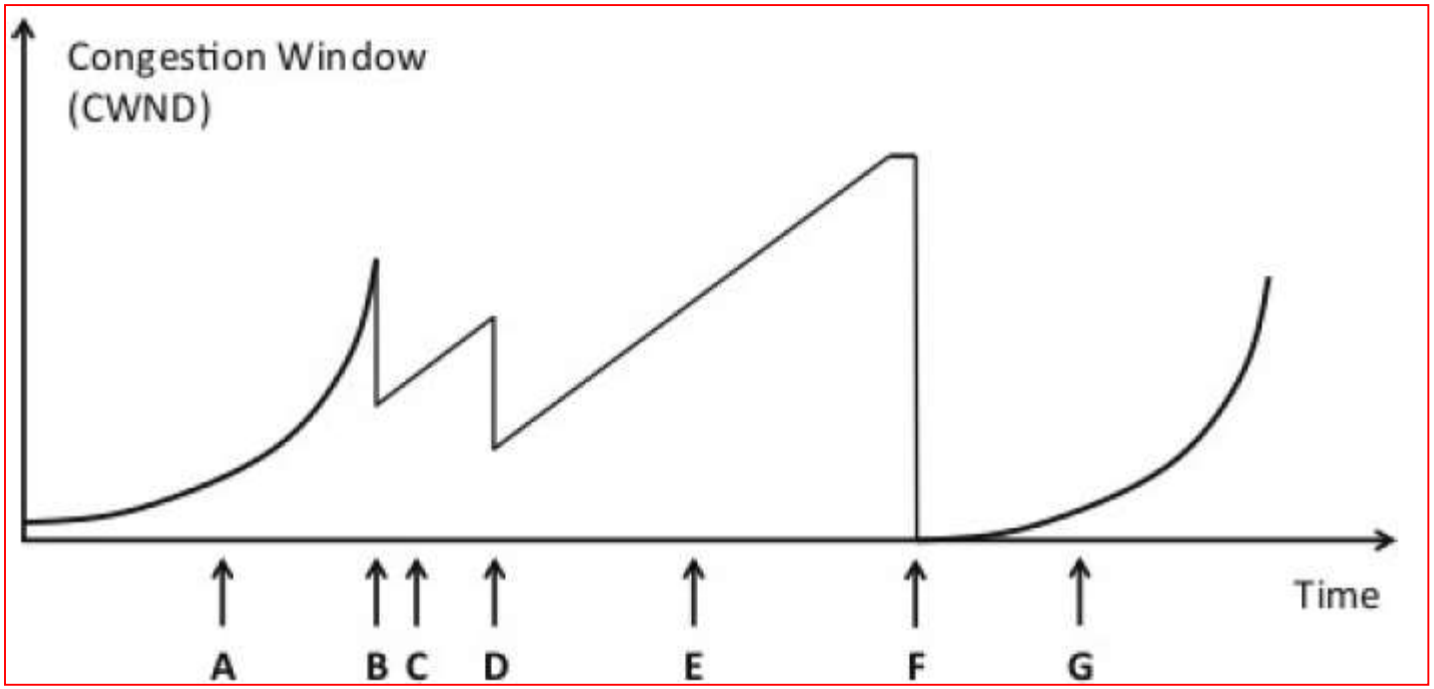
TCP : Contrôle de Congestion

- Parmi les différentes implémentation de TCP, citons :



- *TCP Tahoe 1988*
  - *slow start + congestion avoidance + fast Retransmit*
- *TCP Reno 1990 (RFC 2581)*
  - *Tahoe + Fast recovery*
  - *fast recovery (pas de slow start aprés un fast retransmit)*
- *TCP newReno 1996 (RFC 2582)*
  - *idem TCP Reno*
  - *pas de slow start à la première congestion et ajustement de cwin*
  - *SACK (RFC 2018)*
- *TCP Vegas*
  - *Evite la congestion en anticipant les pertes*
  - *Réduction du débit en fonction des variations du RTT*

- Exemple : Décrire, sur la figure suivante, les différents états (A, B, C, D, E, F et G)



A : SS
B : 3 ACK, CW/2
C: CA
D: 3 ACK, CW/2
E: CA
F: Timeout
G: SS