

Cours 3: TCP/IP

Couche transport



1

Cours 3: Plan

3.1 Services de la couche Transport

3.2 Multiplexage et démultiplexage

3.3 Transport sans connexion: UDP

3.4 Principes de transfert de données fiable

3.5 Transport orienté connexion: TCP

- structure de segment
- transfert de données fiable
- contrôle de flux
- gestion de connections TCP

3.6 Principes de contrôle de congestion

3.7 contrôle de congestion de TCP

2

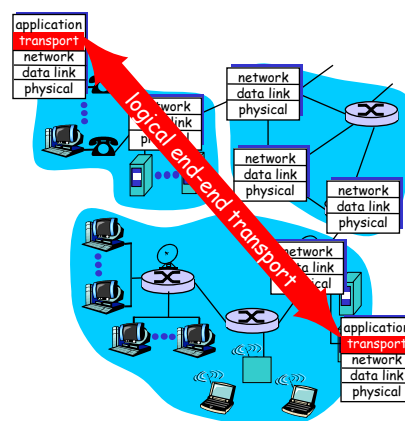
Les couches de protocoles : TCP/IP et le modèle OSI

Protocol Implementation						OSI
File Transfer	Electronic Mail	Terminal Emulation	File Transfer	Client Server	Network Mgmt	Application
File Transfer Protocol (FTP) RFC 559	Simple Mail Transfer Protocol (SMTP) RFC 821	TELNET Protocol RFC 854	Trivial File Transfer Protocol (TFTP) RFC 783	Network File System Protocol (NFS) RFC 1024, 1057 and 1094	Simple Network Management Protocol (SNMP) RFC 1157	Presentation
						Session
Transmission Control Protocol (TCP) RFC 793			User Datagram Protocol (UDP) RFC 768			Transport
Address Resolution Protocols ARP: RFC 826 RARP: RFC 903	Internet Protocol (IP) RFC 791	Internet Group Management Protocol (IGMP) RFC 2236		Internet Control Message Protocol (ICMP) RFC 792		Network
Network Interface Cards						Data Link
Ethernet	Token Ring	Starlan	Arcnet	FDDI	SMDS	
Transmission Mode						Physical
TP STP FO Satellite Microwave, etc						

3

services de transport et protocoles

- ❑ fournit *une communication logique* sur des processus d'application exploités sur des serveurs différents
- ❑ les protocoles de transport s'exécutent aux extrémités
 - les messages sont découpés en *segments*, par l'émetteur
 - Le récepteur: reassemble ces segments en messages
- ❑ Plusieurs protocoles de transport disponibles pour les applications
 - Internet: TCP et UDP



4

Relation entre couches Transport et réseau

- ❑ *Couche réseau:*
communication logique entre machines
- ❑ *Couche transport:*
communication logique entre processus

analogie:

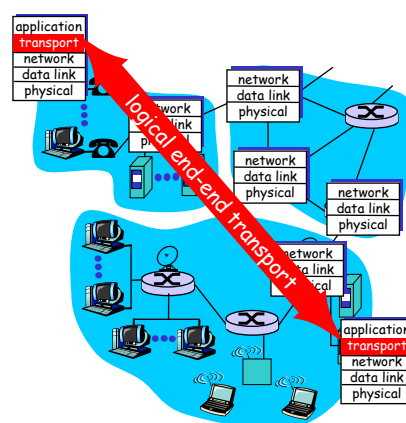
12 enfants envoient des lettres à 12 autres enfants

- ❑ processes = enfants
- ❑ app messages = lettres dans les enveloppes
- ❑ hosts = maisons
- ❑ transport protocol = Anne et Bill
- ❑ network-layer protocol = service postal

5

Les protocoles de la couche transport

- ❑ Délivrance fiable, dans l'ordre (TCP)
 - Contrôle de congestion
 - Contrôle de flux
 - Initialisation de la connection
- ❑ Délivrance Non fiable, en désordre (UDP)
- ❑ services non disponibles:
 - délai garanti
 - bandwidth garantie



6

Cours 3: Plan

3.1 Services de la couche Transport

3.2 Multiplexage et démultiplexage

3.3 Transport sans connexion: UDP

3.4 Principes de transfert de données fiable

3.5 Transport orienté connexion: TCP

- structure de segment
- transfert de données fiable
- contrôle de flux
- gestion de connections TCP

3.6 Principes de contrôle de congestion

3.7 contrôle de congestion de TCP

7


Multiplexage/Démultiplexage


Démultiplexage à rcv host:

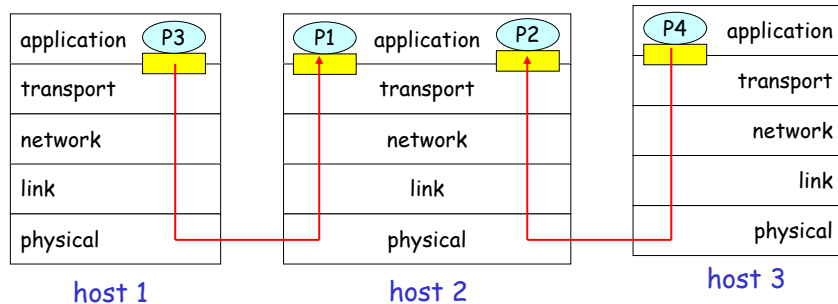
Orientation des segments reçus vers la bonne interface "socket"

Multiplexage à send host:

Rassemblement des données
Création des segments en les associant des en têtes

 = socket

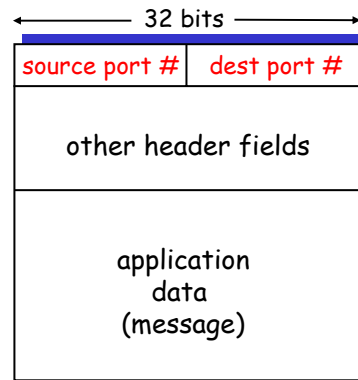
 = process



8

Comment le démultiplexage fonctionne?

- **host reçoit IP datagrams**
 - chaque datagram a une source IP adresse, une destination IP adresse
 - chaque datagram contient un segment transport
 - chaque segment a des numéros de ports source, destination
 - La machine utilise l'adresse IP et le numéro de port pour diriger le segment au socket appropriée



TCP/UDP segment format

9

Démultiplexage sans connexion

- **Creation des sockets avec les numéros de port:**

```
DatagramSocket mySocket1 = new  
    DatagramSocket(09111);
```

```
DatagramSocket mySocket2 = new  
    DatagramSocket(09222);
```

- **UDP socket identifiée par le tuple :**

(dest IP address, dest port number)

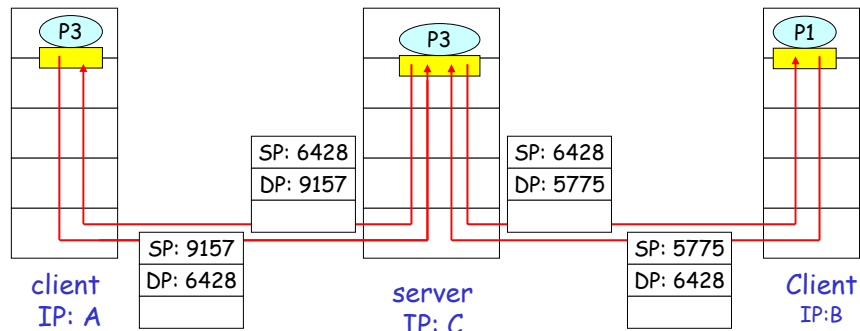
- **Quand host reçoit UDP segment:**

- contrôle le numéro de port destination dans le segment
- dirige UDP segment vers la socket avec un numéro de port

10

Démultiplexage sans connection

```
DatagramSocket serverSocket = new DatagramSocket(6428);
```



SP provides "return address"

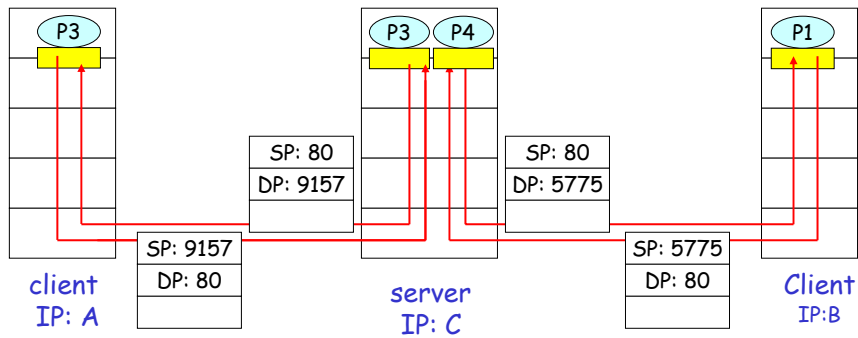
11

Démultiplexage avec Connection

- ❑ TCP socket identifiée par un tuple suivant:
 - source IP address
 - source port number
 - dest IP address
 - dest port number
- ❑ recv host utilise les 4 valeurs pour diriger le segment vers la socket appropriée
- ❑ Server host peut supporter plusieurs TCP sockets simultanées:
 - chaque socket identifiée par un tuple de 4 valeurs
- ❑ Web servers ont des différents sockets pour chaque connection client
 - HTTP ouvre différentes socket pour chaque demande

12

Démultiplexage avec Connection



13

Cours 3: Plan

3.1 Services de la couche Transport

3.2 Multiplexage et démultiplexage

3.3 Transport sans connexion: UDP

3.4 Principes de transfert de données fiable

3.5 Transport orienté connexion: TCP

- structure de segment
- transfert de données fiable
- contrôle de flux
- gestion de connections TCP

3.6 Principes de contrôle de congestion

3.7 contrôle de congestion de TCP

14

UDP: User Datagram Protocol [RFC 768]

- ❑ UDP est un protocole "best effort", UDP segments peuvent être:

- perdus
- Délivrés en désordre à la couche application

- ❑ *Sans connection:*

- pas d'accord en avance entre UDP sender, receiver
- chaque UDP segment est manipulé indépendamment des autres

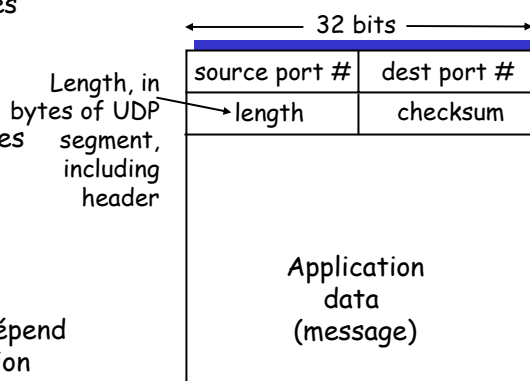
Pourquoi UDP?

- ❑ pas d'établissement de connection (qui peut ajouter un délai supplémentaire)
- ❑ simple: pas d'état de connection à l'émetteur, récepteur
- ❑ un en-tête plus court de segment (8 octets)
- ❑ pas de contrôle de congestion

15

UDP: structure de segment

- ❑ souvent utilisé pour des applications comme streaming multimédia (audio)
 - tolérance aux pertes
 - sensible au débit
- ❑ autres applications
 - DNS
 - SNMP
- ❑ le transfert sur UDP dépend de la couches application



UDP segment format

16

UDP: checksum

but: permet la détection d'erreur sur le segment par le récepteur

Sender:

- ❑ fait la somme de tous les mots de 16 bits du segment en éliminant tout dépassement de capacité
- ❑ checksum: puis on fera le complément à 1 du résultat de la somme
- ❑ sender met cette valeur dans le champ checksum du segment UDP

Receiver:

- ❑ fait la somme de tous les mots de 16 bits y compris le checksum
- ❑ si cette somme = "1111111111111111"
 - Non → error detected
 - Oui → no error detected.

17

Cours 3: Plan

3.1 Services de la couche Transport

3.2 Multiplexage et démultiplexage

3.3 Transport sans connexion: UDP

3.4 Principes de transfert de données fiable

3.5 Transport orienté connexion: TCP

- structure de segment
- transfert de données fiable
- contrôle de flux
- gestion de connections TCP

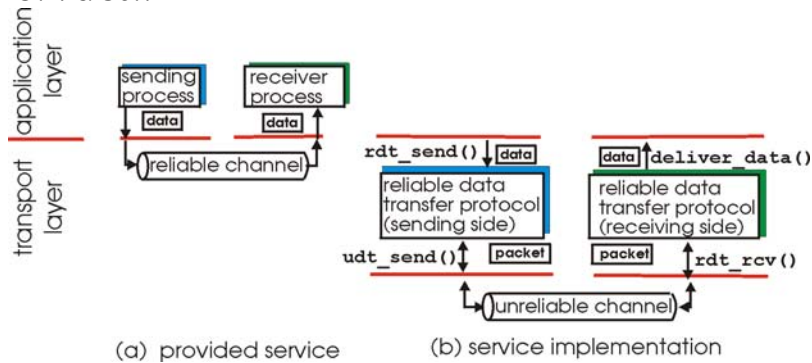
3.6 Principes de contrôle de congestion

3.7 contrôle de congestion de TCP

18

Principes du transfert de données fiable

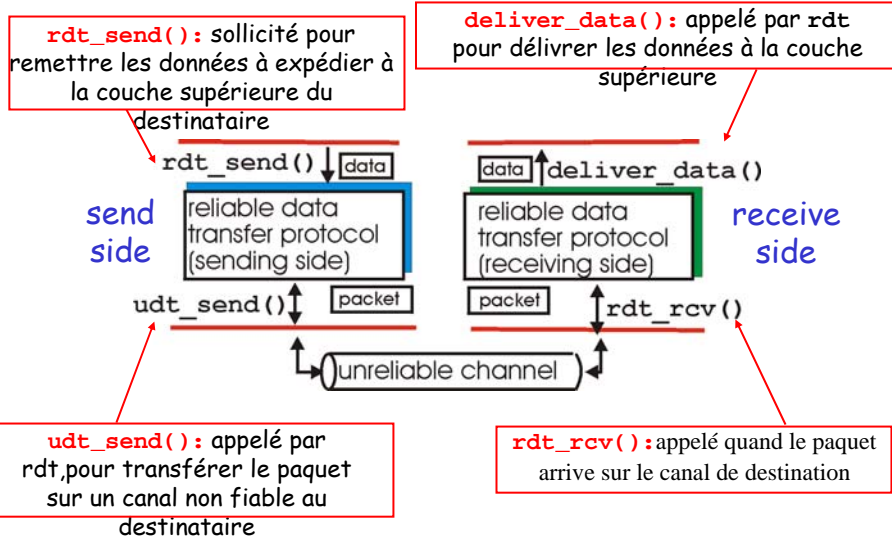
- important dans les couches application, transport et liaison



- Caractéristiques de canal non fiable détermineront la complexité du transfert de données fiable "reliable data transfer protocol (rdt)/unreliable dt (non fiable)"

19

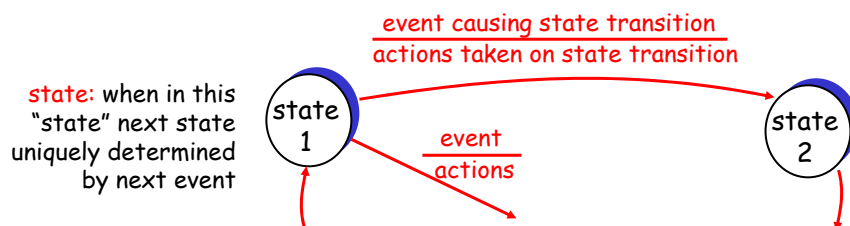
Reliable data transfer: initial



20

Reliable data transfer: initial

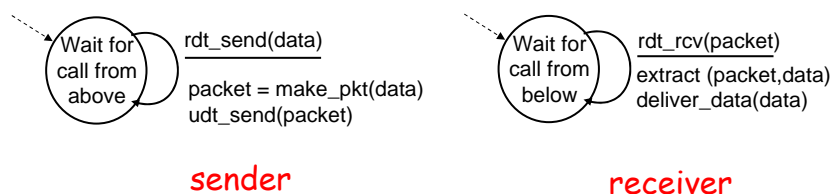
- ❑ développement de façon indépendante les rdt de l'émetteur et de récepteur
- ❑ on considère seulement unidirectionnel data transfer
 - mais le contrôle de flux est bidirectionnel
- ❑ utilise finite state machines (FSM) "automate à nombre d'états fini" pour spécifier sender, receiver



21

Transfert fiable sur un canal fiable

- ❑ au dessous un canal parfaitement fiable
 - pas d'erreurs bit
 - pas de pertes de paquets
- ❑ séparation des FSMs de sender et receiver:
 - sender envoie des data sur le canal de dessous
 - receiver lit les data de canal de dessous



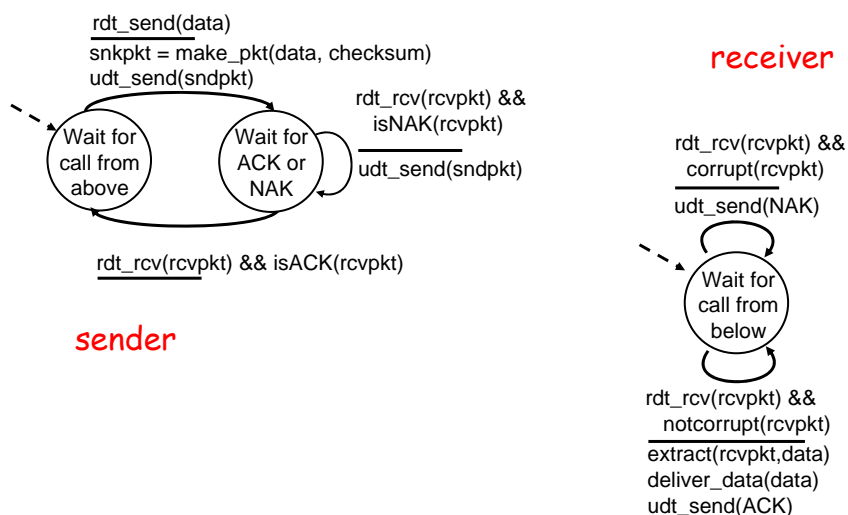
22

Transfert sur un canal avec erreurs bit

- ❑ au dessous d'un canal qui laisse introduire des erreurs
- ❑ /a question: comment détecter les erreurs:
 - *acknowledgements (ACKs)*: le destinataire notifie explicitement l'émetteur que le paquet est reçu
 - *negative acknowledgements (NAKs)*: récepteur envoie une notification explicitement à l'émetteur qu'il y a une erreur dans le paquet
 - sender retransmet les paquets sur réception d'un NAK

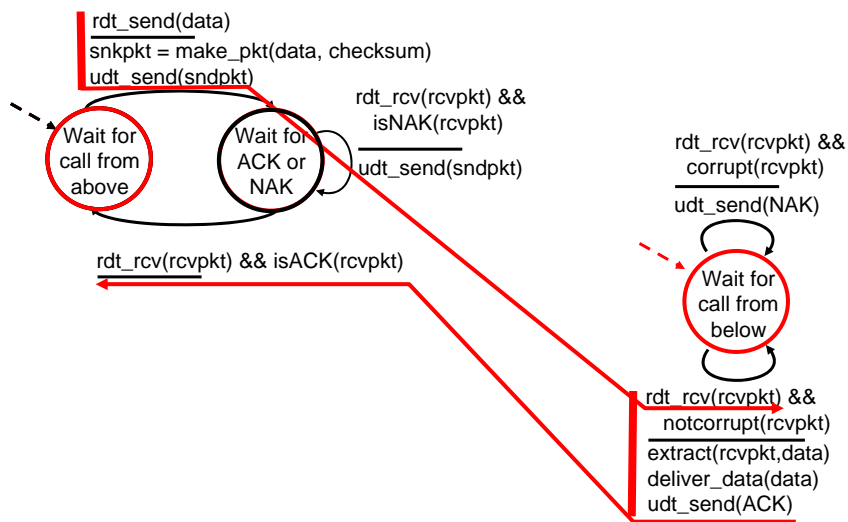
23

FSM specification



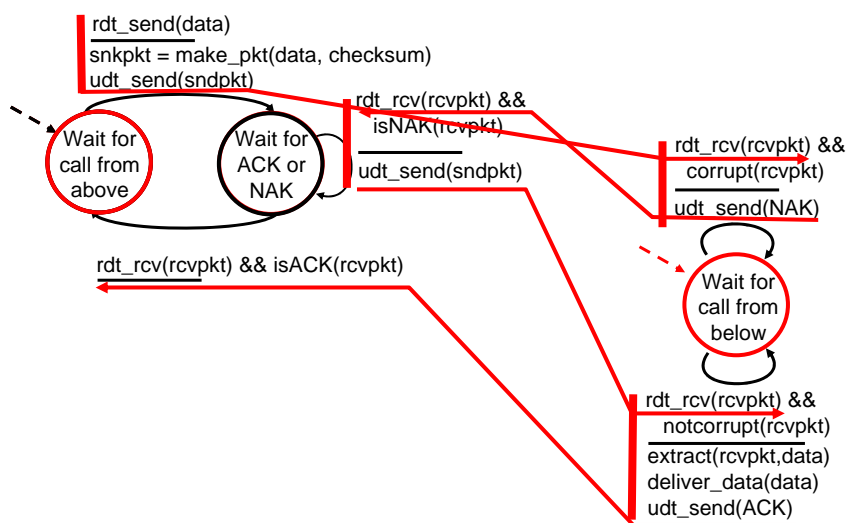
24

Opération sans erreurs



25

Scenario d'erreur



26

Erreur fatale !

Qu'est ce que se passe si ACK/NAK erronés?

- ❑ l'émetteur ne sait pas qu'est ce que se passe chez le destinataire!
- ❑ ne peut pas juste retransmettre: duplication possible

Qu'est ce qu'il fait?

- ❑ le sender si ACK/NAK se perdent?

Eviter la duplication:

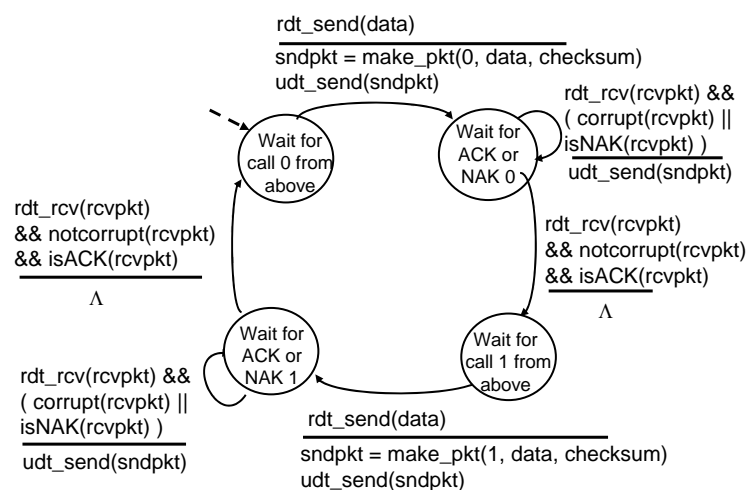
- ❑ sender ajoute *sequence number* pour chaque paquet
- ❑ sender retransmet le paquet courant si ACK/NAK corrompus
- ❑ le destinataire écarte le paquet dupliqué

stop and wait

Sender envoie un paquet alors attends la réponse de récepteur

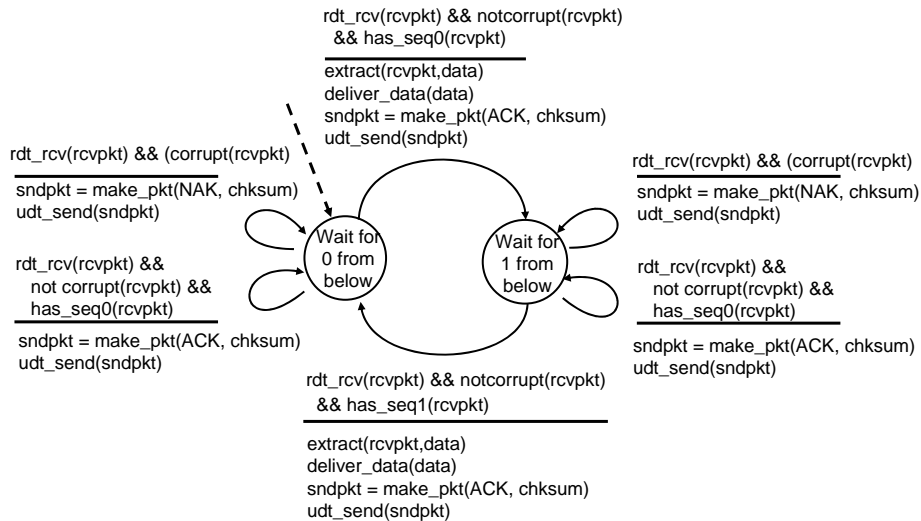
27

sender, manipule les ACK/NAKs corrompus



28

receiver, manipule les ACK/NAKs corrompus



29

Discussion

Sender:

- ❑ Seq.number est ajouté au paquet
- ❑ 2 seq.numbers suffisent (0,1)
- ❑ Doit contrôler si des ACK/NAK corrompus reçus
- ❑ Une paire d'états
 - État doit rappeler que le paquet courant a un seq. number = 1 ou 0 "

Receiver:

- ❑ Doit vérifier si le paquet reçu est dupliqué
 - État indique 0 or 1 dans le seq. number du paquet expédié
- ❑ note: receiver ne peut pas savoir si le dernier ACK/NAK reçu est bon chez le sender

30

Canaux avec perte et erreur

On suppose que: canal soit soumis à des erreurs binaires dans les paquets et certains paquets soient perdus (data or ACKs)

- L'utilisation de : checksum, seq. number, ACKs et les retransmissions peuvent aider (mesures à prendre en cas de perte) mais ne sont pas suffisantes (pour détecter les paquets perdus)

Q: comment échanger avec la perte?

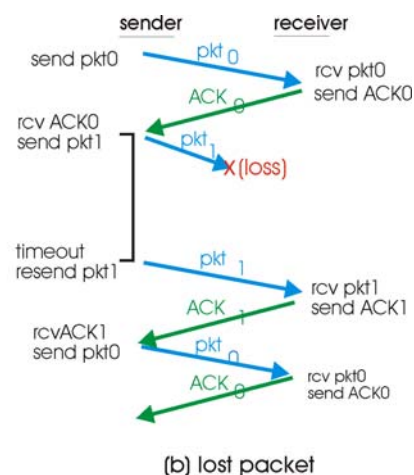
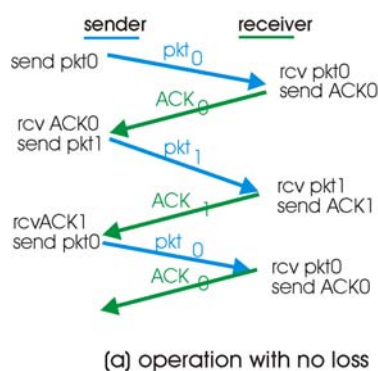
- Sender attend jusqu'à certains data ou ACK soient perdus, alors il retransmet

Approche: sender attend un temps moyen d'un ACK

- Retransmet s'il n'y a pas un ACK reçu dans ce temps
- Si le paquet (or ACK) juste retardé (pas perdu):
 - retransmission sera dupliquée mais utilise le seq. number déjà utilisé
 - receiver doit spécifier le seq. number de paquet étant acquitté
- Demande un compteur "temporisateur:compte à rebours"

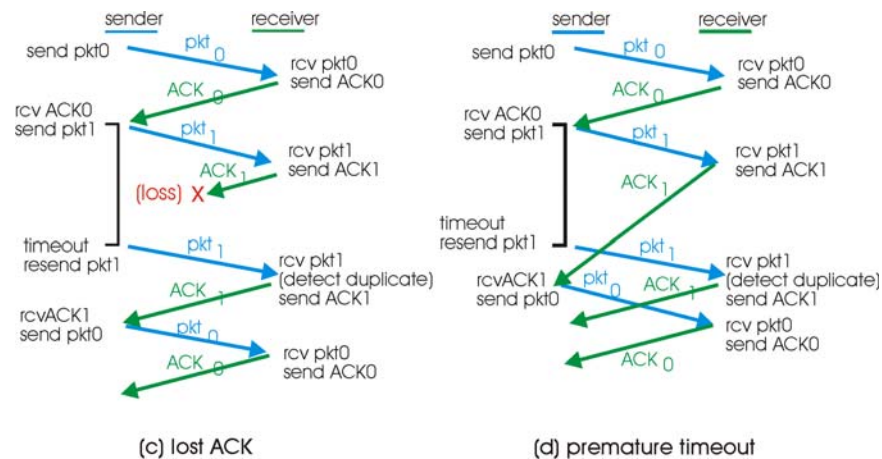
31

Action: échanges des paquets



32

Action: perte d'un ACK



33

Performance

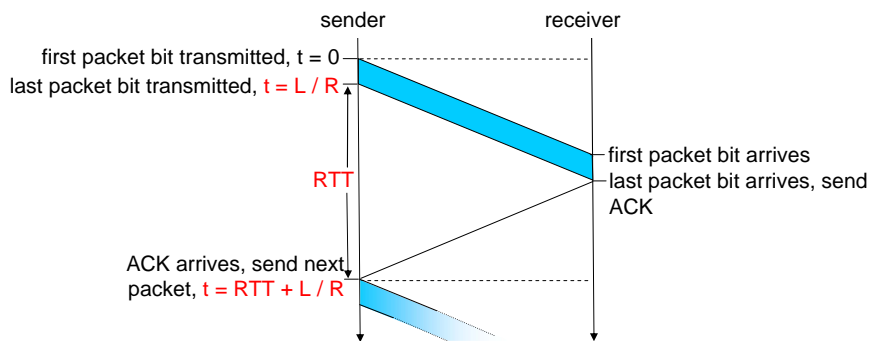
- exemple: lien d' 1 Gbps, 15 ms propagation, 1KB packet:
- U (taux d'utilisation)

$$T_{\text{transmit}} = \frac{L \text{ (packet length in bits)}}{R \text{ (transmission rate, bps)}} = \frac{8\text{kb/pkt}}{10^9 \text{ b/sec}} = 8 \text{ microsec}$$

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

34

Opération: stop-and-wait



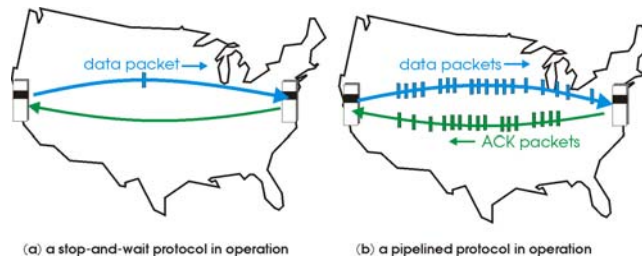
$$U_{\text{sender}} = \frac{L / R}{RTT + L / R} = \frac{.008}{30.008} = 0.00027$$

35

Protocole Pipeliné

Pipelining: envoie de plusieurs paquets avant de se mettre en attente

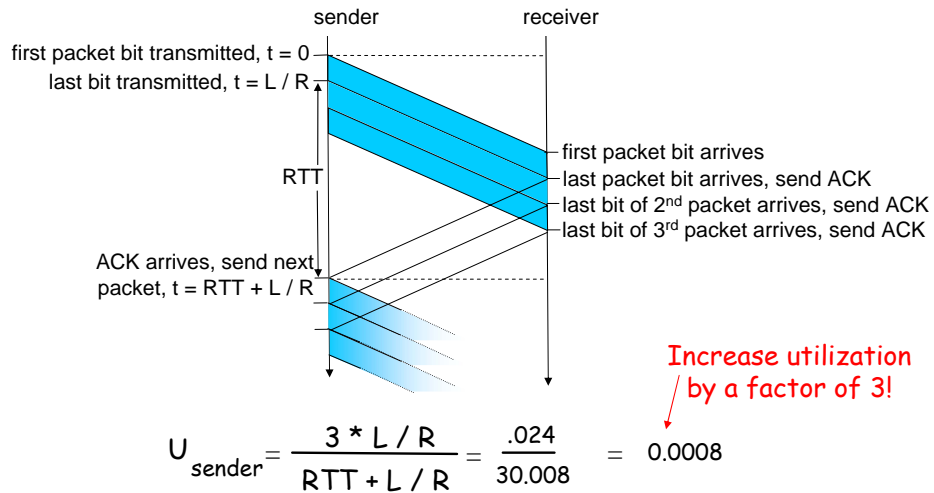
- L'intervalle de numéros de séquence doit être augmenté
- buffering à l'émetteur et/ou récepteur



- Deux méthodes de correction d'erreurs en mode pipeline:
go-Back-N (aller à la trame N), répétition sélective

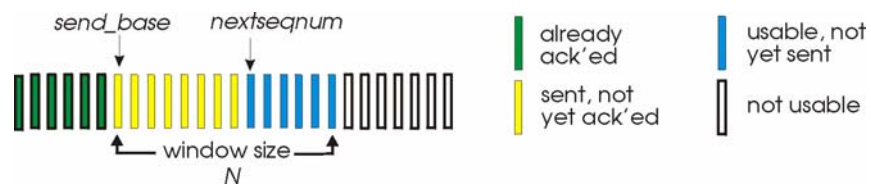
36

Pipelining: augmente l'utilisation



37

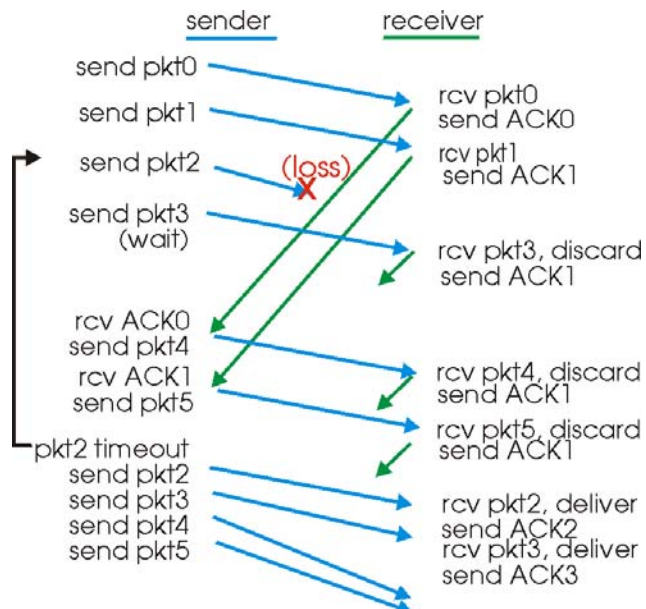
Go-Back-N



- C'est un mécanisme à fenêtre glissante

38

GBN en action



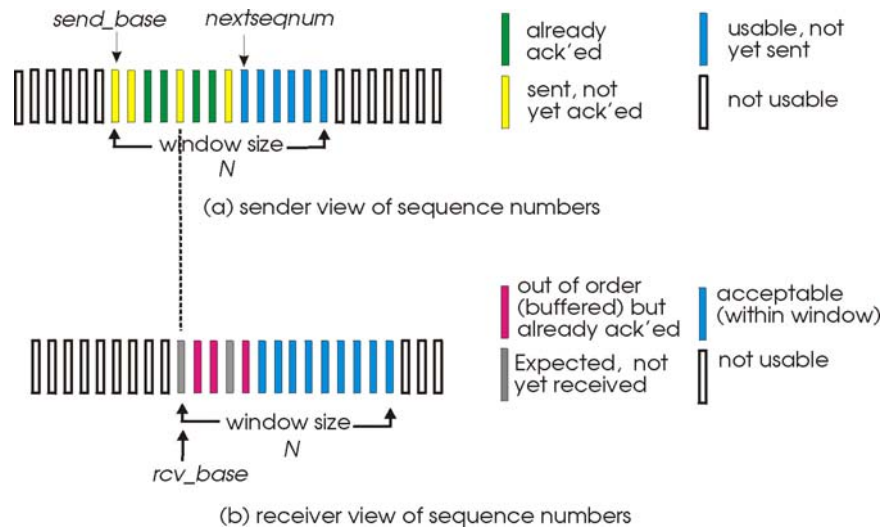
39

Répétition sélective

- ❑ Le GBN souffre aussi des problèmes de performances, notamment le débit lorsqu'il y a un paquet erroné.
- ❑ La répétition sélective, retransmet que les paquets erronés

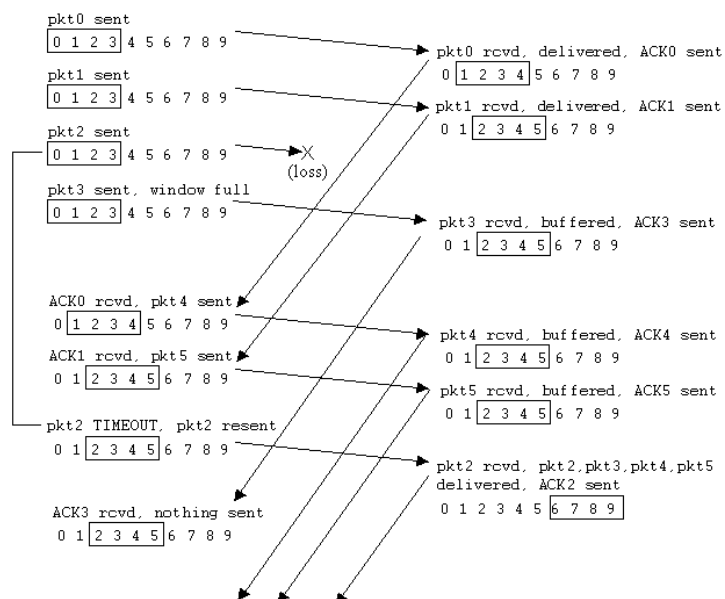
40

Selective repeat: sender, receiver windows



41

Selective repeat in action



42

Cours 3: Plan

3.1 Services de la couche Transport

3.2 Multiplexage et démultiplexage

3.3 Transport sans connexion: UDP

3.4 Principes de transfert de données fiable

3.5 Transport orienté connexion: TCP

- structure de segment
- transfert de données fiable
- contrôle de flux
- gestion de connexions TCP

3.6 Principes de contrôle de congestion

3.7 contrôle de congestion de TCP

43

TCP: Overview

RFCs: 793, 1122, 1323, 2018, 2581

□ point-to-point:

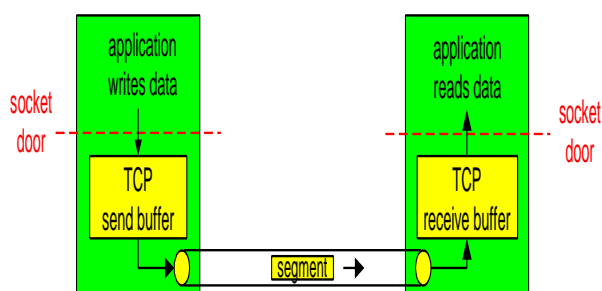
- un sender, un receiver

□ full duplex data:

□ connection-oriented

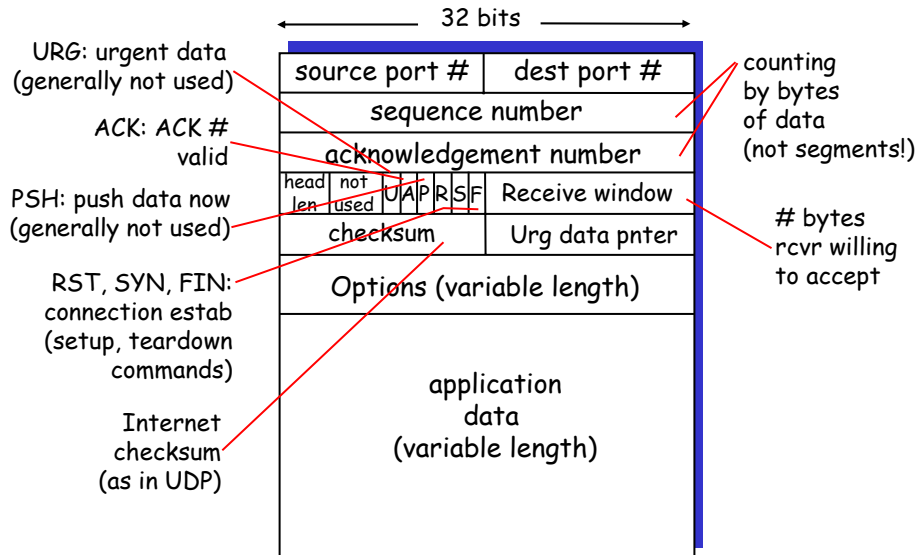
□ pipelined

□ *send & receive buffers*



44

TCP segment structure



45

TCP seq. number et ACKs

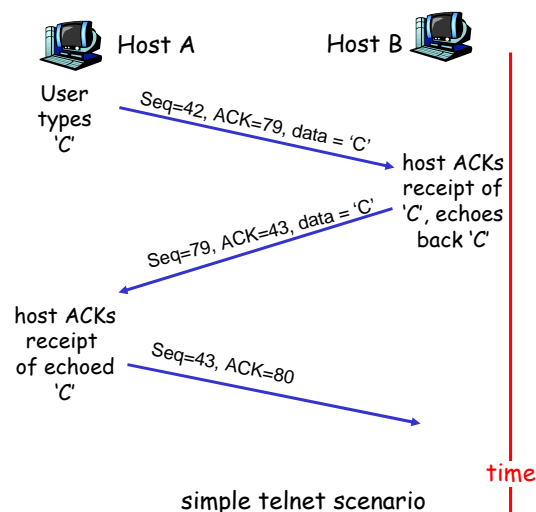
Numéro de Séquence:

- Le numéro dans le flux d'octets du premier octet de chaque segment

ACKs:

- seq. number du prochain octet expédié de l'autre côté
- en cas de perte d'un segment un cumulative ACK cumulatif est envoyé

- Q:** comment TCP résolve le problème de perte de segment hors séquence
- Le choix au programmeur de le détruire ou de le conserver



46

TCP: Round Trip Time et Timeout

Q: comment fixer la valeur timeout?

- plus supérieur que le RTT
 - mais le RTT varie
- court: timeout prématuré
 - Retransmissions non nécessaire
- long: TCP tardera à retransmettre le segments

Q: comment estimer le RTT?

- **SampleRTT**: temps écoulé entre son envoi (c'est à dire son passage à l'IP) et l'ACK renvoyé par le destinataire
 - Ignore les segments retransmis
- **SampleRTT** varié d'un segment à un autre
 - Une mesure moyenne s'avère nécessaire appelée **EstimatedRTT**

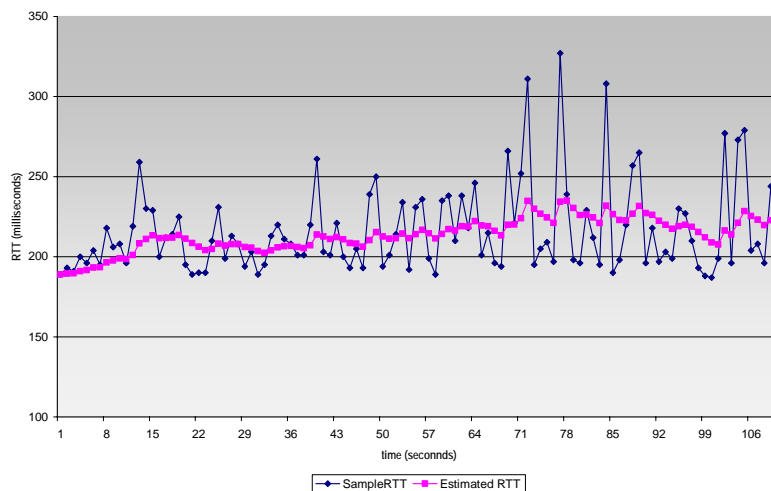
$$\text{EstimatedRTT}(\text{new}) = (1 - \alpha) * \text{EstimatedRTT}(\text{old}) + \alpha * \text{SampleRTT}(\text{new})$$

Avec $\alpha = 0,125$

47

Exemple d'estimation RTT:

RTT: gala.cs.umass.edu to fantasia.eurecom.fr



48

TCP Round Trip Time et Timeout

Timeout (temporisation de retransmission)

- Coefficient de variation du RTT (**DevRTT**):

$$\text{DevRTT} = (1-\beta) * \text{DevRTT} + \beta * |\text{SampleRTT} - \text{EstimatedRTT}|$$

(typically, $\beta = 0.25$)

$$\text{TimeoutInterval} = \text{EstimatedRTT} + 4 * \text{DevRTT}$$

49

Cours 3: Plan

3.1 Services de la couche Transport

3.2 Multiplexage et démultiplexage

3.3 Transport sans connexion: UDP

3.4 Principes de transfert de données fiable

3.5 Transport orienté connexion: TCP

- structure de segment
- transfert de données fiable
- contrôle de flux
- gestion de connections TCP

3.6 Principes de contrôle de congestion

3.7 contrôle de congestion de TCP

50

Transfert de données fiable

- ❑ TCP créé un service rdt sur la couche IP non fiable
- ❑ Segments pipelinés
- ❑ Acks cumulatifs
- ❑ TCP utilise un seul temporisateur de retransmission
- ❑ Retransmissions sont conditionnées par:
 - Événements de timeout
 - Duplication des acks
- ❑ Initialement on considère un TCP sender simplifié:
 - ignore la duplication des acks
 - ignore le contrôle de flux et de congestion

51

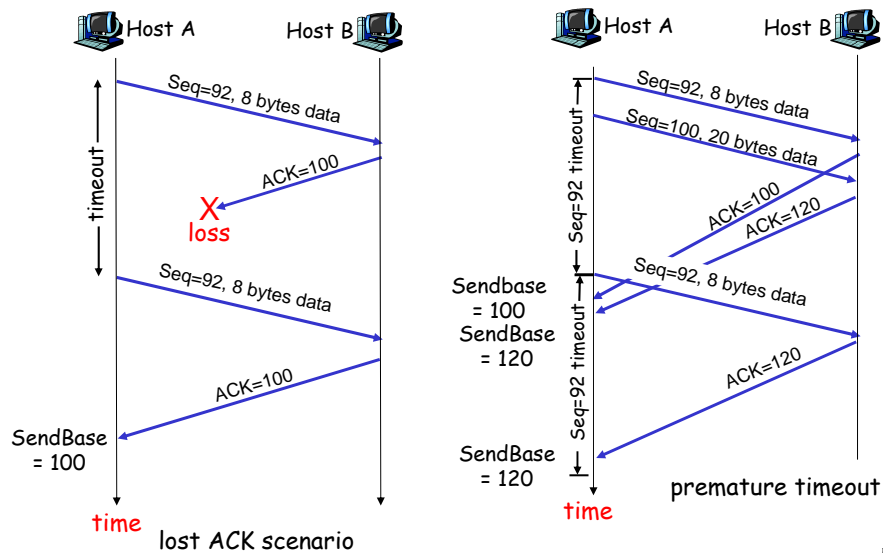
TCP sender: événements:

Trois événements principaux ayant effet sur la transmission et la retransmission de données

- ❑ Données reçues de la couche supérieure
- ❑ Timeout
- ❑ Réception de l'Ack

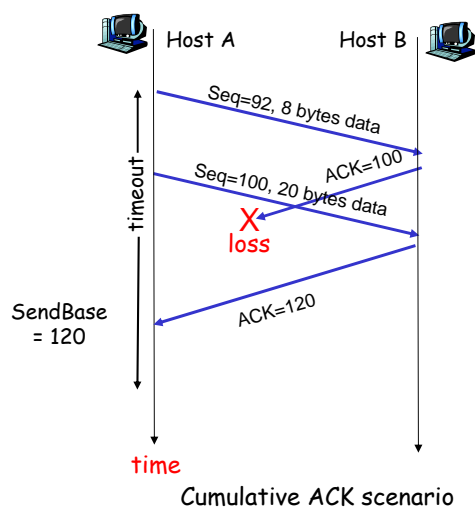
52

TCP: retransmission scenarios



53

TCP retransmission scenarios



54

TCP ACK generation [RFC 1122, RFC 2581]

Event at Receiver	TCP Receiver action
Arrivée d'un segment dans l'ordre Avec un seq. attendu. Toute les données sont acquittées jusqu'au seq.	Génération d'un ACK retardé pendant 500 ms, en attente d'un autre segment, si Rien n'arrive dans cet intervalle, envoi d'ACK
Arrivée d'un segment dans l'ordre Avec un seq. attendu. Un segment précédent est en attente de l'émission De son ACK.	Envoi immédiatement d'un ACK Cumulatif simple, ACK des deux segments (dans l'ordre)
Arrivée d'un segment hors séquence Avec un seq. supérieur au numéro attendu. Lacune détectée	Envoi immédiatement d'un ACK dupliqué Indiquant le next seq. attendu (représentant la limite inférieure de la lacune)
Arrivée d'un segment remplissant Complètement ou partiellement la lacune Dans les données reçues	Envoi immédiatement d'un ACK sous réserve que le segment coïncide avec la limite inférieure de la lacune

55

Retransmission rapide

- ❑ Délai de Time-out relativement long:
 - long délai avant de renvoyer un paquet perdu
- ❑ Détection des segments perdus via des ACK dupliqués
 - sender envoie souvent un grand nombre de segments l'un derrière l'autre
 - la perte d'un segment génère un grand nombre d'ACK en chaîne
- ❑ Si le sender reçoit 3 ACKs pour le même paquet de données, il suppose que le paquet soit perdu dans le réseau → une retransmission rapide
 - retransmission rapide: renvoie le segment manquant avant l'expiration de temporisateur

56

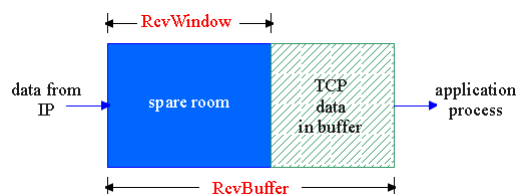
Cours 3: Plan

- 3.1 Services de la couche Transport
- 3.2 Multiplexage et démultiplexage
- 3.3 Transport sans connexion: UDP
- 3.4 Principes de transfert de données fiable
- 3.5 Transport orienté connexion: TCP
 - structure de segment
 - transfert de données fiable
 - **contrôle de flux**
 - gestion de connections TCP
- 3.6 Principes de contrôle de congestion
- 3.7 contrôle de congestion de TCP

57

TCP: Flow Control

- tampon (buffer) de réception:



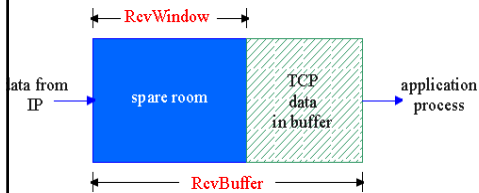
flow control

L'objectif est d'équilibrer le rythme d'envoi au sender à la vitesse de lecture de destinataire

- le processus d'application peut être lent pour lire dans le buffer

58

Flow control: fonctionnement



(Suppose que le receiver supprime tous les segments hors séquence)

- Espace mémoire disponible(dans le buffer)
= RcvWindow
= RcvBuffer - [LastByteRcvd - LastByteRead]

- Rcv informe le sender de l'espace disponible dans son tampon en insérant la valeur RcvWindow dans les segments
- Sender maintient les données non confirmées au dessous de RcvWindow
 - Données non confirmées = LastByteSent - LastByteAcked
 - garantir le non débordement de buffer de réception

59

Cours 3: Plan

3.1 Services de la couche Transport

3.2 Multiplexage et démultiplexage

3.3 Transport sans connexion: UDP

3.4 Principes de transfert de données fiable

3.5 Transport orienté connexion: TCP

- structure de segment
- transfert de données fiable
- contrôle de flux
- gestion de connections TCP

3.6 Principes de contrôle de congestion

3.7 contrôle de congestion de TCP

60

TCP: Gestion de Connection

Rappel: sender, receiver établissent une "connection" avant d'échanger les segments de données

- ❑ Initialisent les variables:
 - seq. number
 - buffers, flow control info (RcvWindow)

- ❑ *client*: informe le récepteur par

```
Socket clientSocket = new  
Socket("hostname", "port  
number");
```

- ❑ *serveur*: contacté par le client

```
Socket connectionSocket =  
welcomeSocket.accept();
```

L'établissement se fait en 3 étapes:

Step 1: client envoie un SYN segment au serveur

- spécifie initial seq. number
- pas de données

Step 2: serveur reçoit SYN, répond par un SYNACK segment

- serveur alloue les buffers
- spécifie son initial seq. number

Step 3: client reçoit SYNACK, répond avec ACK segment, qui peut contenir des données

61

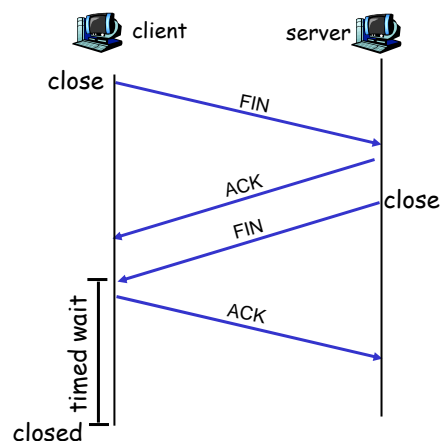
TCP:Gestion de Connection

Fermeture de connection:

client ferme la socket:
`clientSocket.close();`

Step 1: client envoie un segment FIN au serveur

Step 2: serveur reçoit le segment FIN, répond avec ACK. Ferme connection, envoie FIN.



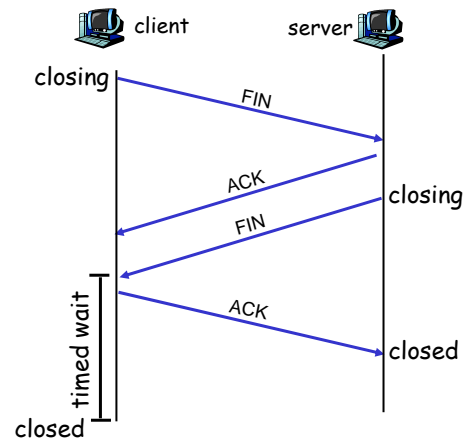
62

TCP: Gestion de Connection

Step 3: client reçoit FIN, répond avec ACK.

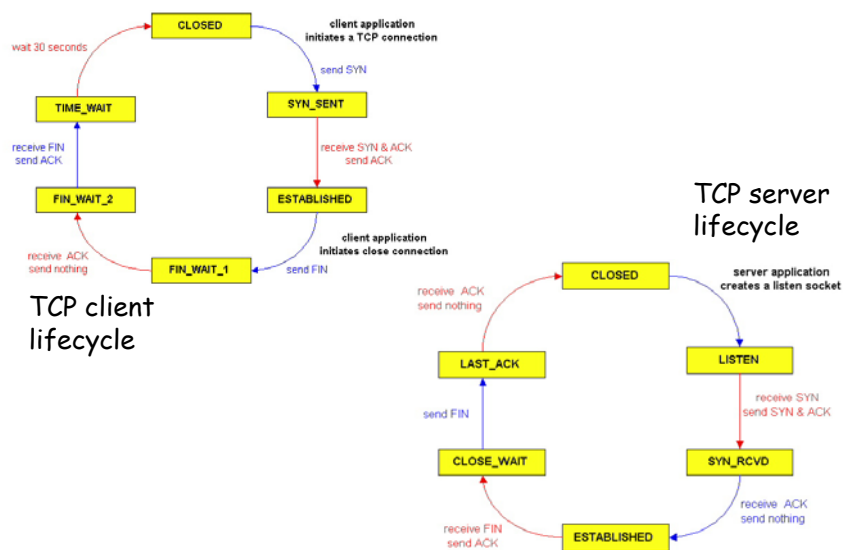
- Déclenche un "timed wait" = 30 secondes qui lui permet de renvoyer l'ACK en cas de perte

Step 4: server, reçoit un ACK. La Connection est fermée.



63

TCP: Gestion de Connection



64

Cours 3: Plan

- 3.1 Services de la couche Transport
- 3.2 Multiplexage et démultiplexage
- 3.3 Transport sans connexion: UDP
- 3.4 Principes de transfert de données fiable
- 3.5 Transport orienté connexion: TCP
 - structure de segment
 - transfert de données fiable
 - contrôle de flux
 - gestion de connexions TCP
- 3.6 Principes de contrôle de congestion
- 3.7 contrôle de congestion de TCP

65

Principes du contrôle de Congestion

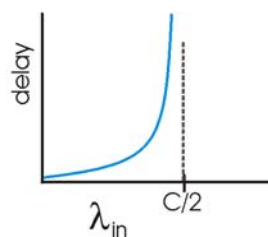
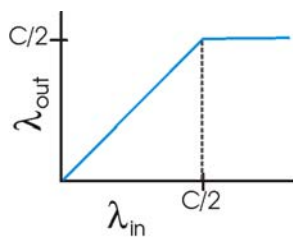
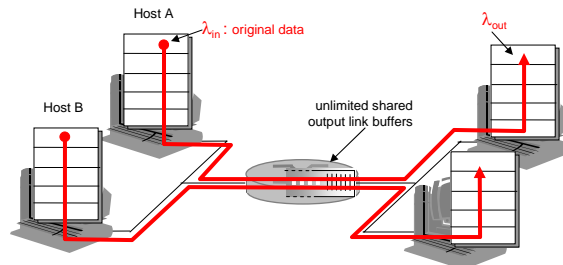
Congestion:

- informellement: "plusieurs sources, beaucoup de données plus rapide pour le réseau à manipuler "
- manifestations:
 - pertes des paquets (saturation des buffers des routeurs)
 - longs délais (en file dans les buffers de routeurs)

66

Causes/effets de congestion

- 2 senders, 2 receivers, un routeur aux buffers infinis
- Non retransmission

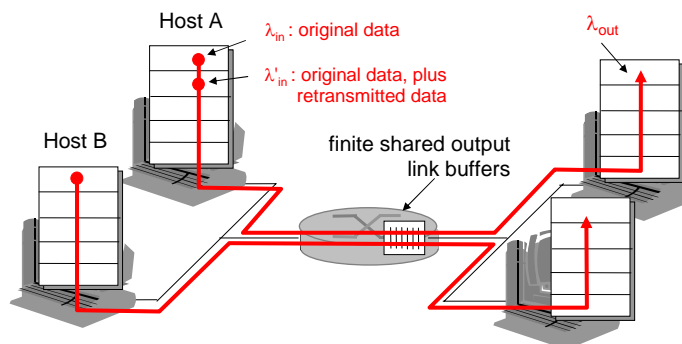


- grands délais dans les routeurs intermédiaires

67

Causes/effets de congestion

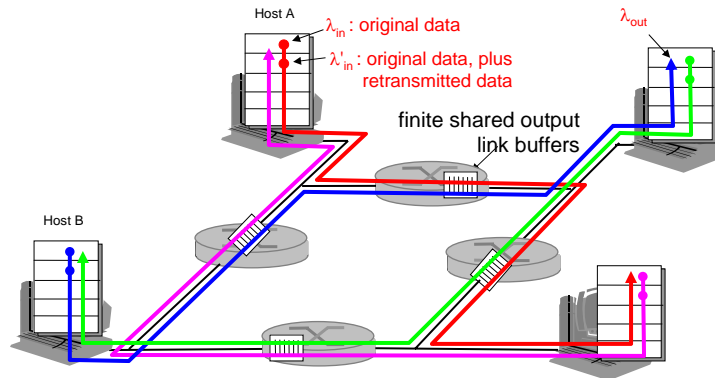
- un routeur aux buffers *finis*
- retransmission des paquets perdus



68

Causes/effets de congestion

- 4 senders
- chemins multihops
- timeout/retransmission



69

Différentes approches du contrôle de congestion

Deux approches du contrôle de congestion :

Contrôle de congestion de bout-en-bout:

- pas de support explicite à la couche transport
- la congestion est reconnue directement par les terminaux
- approche adoptée par TCP

Contrôle de congestion assisté par le réseau:

- routeurs fournissent de l'info. de congestion au terminaux
 - indication par un simple congestion (SNA, DECbit, TCP/IP ECN, ATM)
 - Taux de transmission explicite envoyé au sender par le routeur

70

Cas d'étude: contrôle de congestion ATM

ABR

ABR: available bit rate:

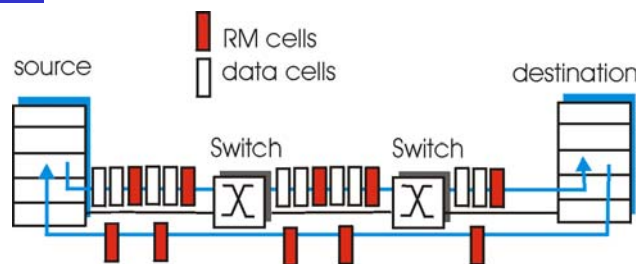
- "elastic service"
- si le trafic réseau est faible
 - sender utilise la bandwidth disponible
- si le réseau saturé:
 - sender doit limiter son débit à un taux de transmission minimal prédéfini

RM (resource management) cells:

- cellules alternées avec les cellules de données, une cellule RM toutes les 32 de données
- bits dans la cellule RM sont positionnés par les switches ("network-assisted")
 - NI bit: no increase dans le cas de faible saturation (congestion moyenne)
 - CI bit: congestion indication (congestion aigüe)
- RM cells retournées au sender par le receiver, avec les bits NI et CI intacts

71

Cas d'étude: ATM ABR congestion control



- 2 octets ER (explicit rate) field in RM cell
 - Un commutateur saturé peut diminuer la valeur du champ ER, ce champ sera réglé au débit minimum acceptable par tous les commutateurs de parcours
 - Sender envoie le débit acceptable sur le chemin
- EFCI bit in data cells: positionné à 1 par le switch congestionné
 - Si la data cell précédent RM cell a EFCI positionné, le destinataire positionne CI bit dans la cellule RM cell et la retourne

72

Cours 3: Plan

- 3.1 Services de la couche Transport
- 3.2 Multiplexage et démultiplexage
- 3.3 Transport sans connexion: UDP
- 3.4 Principes de transfert de données fiable
- 3.5 Transport orienté connexion: TCP
 - structure de segment
 - transfert de données fiable
 - contrôle de flux
 - gestion de connections TCP
- 3.6 Principes de contrôle de congestion
- 3.7 **contrôle de congestion de TCP**

73

TCP Congestion Control

- contrôle de bout en bout
- sender limite la transmission:
 $\text{LastByteSent} - \text{LastByteAcked} \leq \text{CongWin}$

$$\text{Taux d'envoi} = \frac{\text{CongWin}}{\text{RTT}} \text{ Bytes/sec}$$

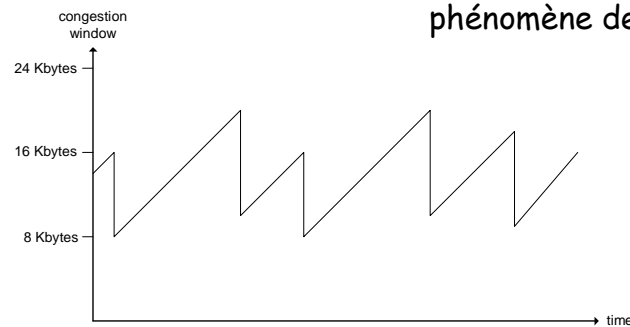
- CongWin: limite le rythme auquel l'expéditeur est autorisé à charger le réseau
- Comment un expéditeur perçoit la congestion?
- soit par un timeout *ou* par les 3 acks dupliqués
 - TCP sender réduit le taux (CongWin) si le phénomène de perte se déclare
- trois mécanismes:
- AIMD(Additive Increase, Multiplicative Decrease)= Accroissement additif et décroissance multiplicative
 - slow start (départ lent)
 - conservative after timeout events (une réaction au phénomène d'expiration)

74

TCP AIMD

Décroissance multiplicative :
réduire son taux d'envoi
CongWin à moitié dès que
le phénomène de perte se
déclare

Croissance additive:
augmente CongWin
par 1 MSS à chaque
RTT en absence de
phénomène de perte



Contrôle de congestion en mode AIMD

75

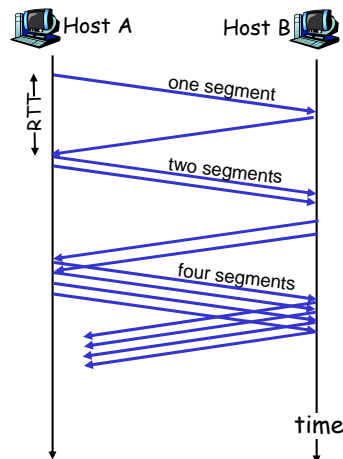
TCP: départ lent (Slow Start)

- Quand la connexion commence, CongWin = 1 MSS
 - Exemple: MSS = 500 bytes & RTT = 200 msec
 - débit initial = 20 kbps
- bandwidth disponible >> MSS/RTT

- Quand la connexion commence, augmente le débit exponentiellement jusqu'au le premier phénomène de perte se produit
 - double CongWin chaque RTT
 - fait par incrémentation de CongWin pour chaque ACK reçu

76

TCP: Slow Start



77

Réaction au temporisation

- Après 3 dup ACKs:
 - CongWin est diminuée en moitié
 - window s'accroît linéairement
 - Mais après un timeout:
 - CongWin = 1 MSS;
 - window s'accroît exponentiellement
- 3 dup ACKs indique que le réseau est capable de délivrer quelques segments
 - timeout avant 3 dup ACKs est alarmé

78

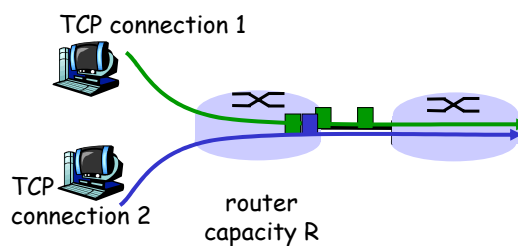
Résumé: TCP Congestion Control

- ❑ Quand le CongWin est au dessous de seuil, le sender est dans la phase **slow-start**, window s'accroît exponentiellement
- ❑ Quand le Congwin est au dessus de seuil, le sender est dans la phase **congestion-avoidance**, window s'accroît linéairement.
- ❑ Quand un **triple duplicate ACK** se produit, seuil = $\text{CongWin}/2$ et $\text{CongWin} = \text{Threshold}$.
- ❑ Quand le **timeout** produit, $\text{Threshold} = \text{CongWin}/2$ et $\text{CongWin} = 1 \text{ MSS}$.

79

Équité de TCP

Le but d'équité: si K TCP sessions partagent un même lien de bandwidth = R, chaque session aura un débit de R/K

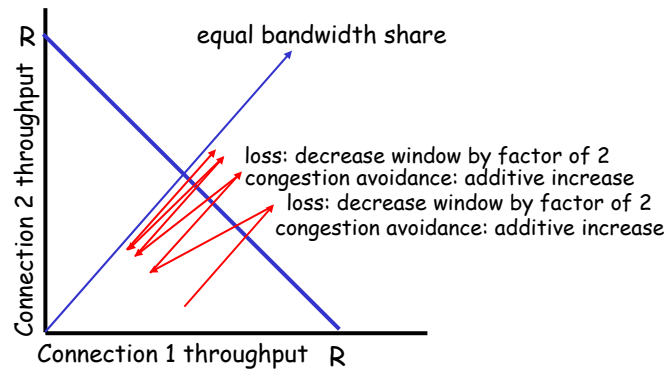


80

Pourquoi TCP est équitable?

Deux connexions compétitives:

- Additive augmente le débit
- multiplicative diminue le débit proportionnellement



81

Équité

Équité et UDP

- Multimédia apps souvent n'utilisent pas TCP
 - Parcequ'elles ne veulent pas voir leur débit bridé
 - Avec UDP peut utiliser un débit constant et une tolérance à la perte des paquets comme l'audio/vidéo

Équité et connexion TCP parallèles

- Aucun moyen d'empêcher une connexion d'ouvrir plusieurs connexions en parallèles
- Web browsers
- Exemple: lien d'un débit R emprunté par 9 applications;
 - Si une nouvelle application cherche à se joindre au groupe, son débit sera de $R/10$

82

Modélisation du délai de réponse de TCP

Q: quelle longueur prise pour recevoir un objet de serveur Web après l'envoi d'une demande ?

Ignorant la congestion, délai est influencé par:

- l'établissement de la connexion TCP
- Délai de transmission des données
- slow start

Notation, suppositions:

- Une seule liaison entre le client et le serveur de débit R
- S : MSS (bits)
- O : longueur de l'objet (bits)
- Pas de retransmissions (ni pertes, ni erreurs)

Longueur de la fenêtre :

- fenêtre de congestion statique, W segments
- fenêtre de congestion dynamique window (slow start)

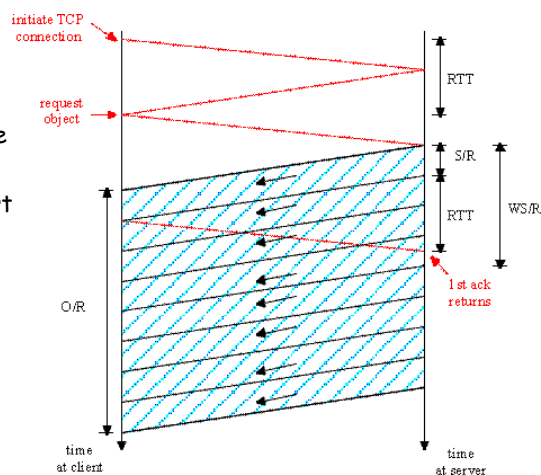
83

Fenêtre de congestion statique

Cas 1 :

$WS/R > RTT + S/R$: le serveur reçoit un ACK concernant le premier segment dans la première fenêtre avant d'avoir terminé de transférer tout son contenu

Temps de latence = $2RTT + O/R$



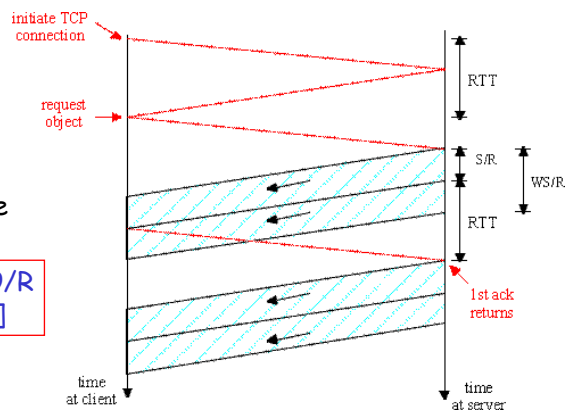
84

Fenêtre de congestion statique

Second cas:

- $WS/R < RTT + S/R$: le serveur transmet les contenus de segments dans la première fenêtre avant de recevoir l'ACK

$$\text{temps de latence} = 2RTT + O/R + (K-1)[S/R + RTT - WS/R]$$



85

TCP Délai de Modélisation: Slow Start

Maintenant, on suppose que la fenêtre augmente accordant un slow start

Nous montrons que le délai pour un objet est:

$$\text{Latence} = 2RTT + \frac{O}{R} + P \left[RTT + \frac{S}{R} \right] - (2^P - 1) \frac{S}{R}$$

P est le nombre de fois où le serveur se met réellement en attente

$$P = \min\{Q, K-1\}$$

-Où Q est le nombre de fois, le serveur se mettrait en attente si l'objet était Constitué d'un nombre de segment infini

- et K est le nombre de fenêtres que couvre l'objet

86

TCP Délai de Modélisation: Slow Start

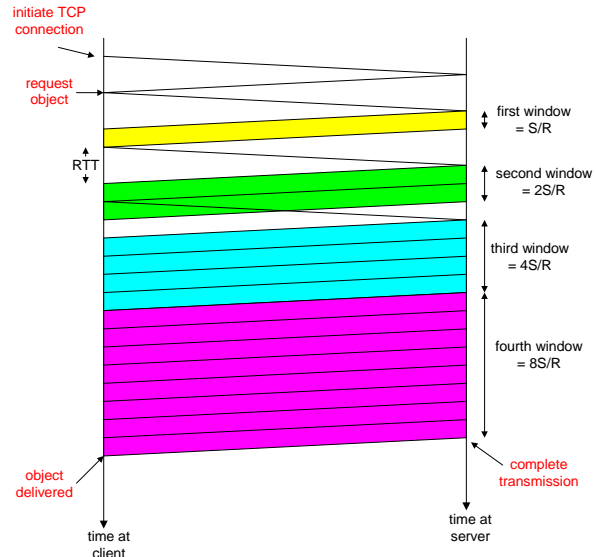
Composition de Délai:

- 2 RTT pour établir et demande de connexion
- O/R pour transmettre l'objet
- le temps de serveur en attente dû au slow start

Server en attente:
 $P = \min\{K-1, Q\}$ fois

Exemple:

- $O/S = 15$ segments
- $K = 4$ windows
- $Q = 2$
- $P = \min\{K-1, Q\} = 2$



87

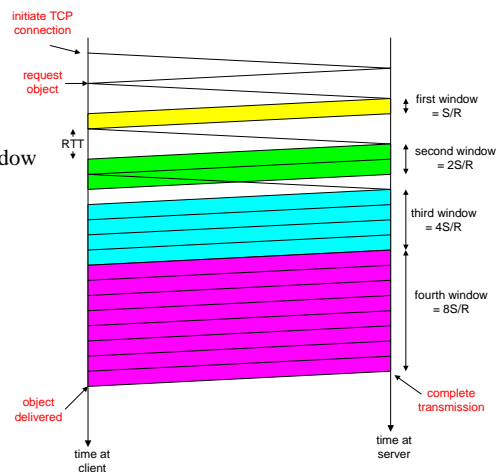
TCP Délai de Modélisation: Slow Start

$\frac{S}{R} + RTT$ = time from when server starts to send segment
 until server receives acknowledgement

$2^{k-1} \frac{S}{R}$ = time to transmit the kth window

$\left[\frac{S}{R} + RTT - 2^{k-1} \frac{S}{R} \right]^+ = \text{idle time after the } k\text{th window}$

$$\begin{aligned} \text{delay} &= \frac{O}{R} + 2RTT + \sum_{p=1}^P \text{idleTime}_p \\ &= \frac{O}{R} + 2RTT + \sum_{k=1}^P \left[\frac{S}{R} + RTT - 2^{k-1} \frac{S}{R} \right] \\ &= \frac{O}{R} + 2RTT + P \left[RTT + \frac{S}{R} \right] - (2^P - 1) \frac{S}{R} \end{aligned}$$



88

TCP Délai de Modélisation: Slow Start

Recall K = number of windows that cover object

How do we calculate K ?

$$\begin{aligned} K &= \min\{k : 2^0 S + 2^1 S + \dots + 2^{k-1} S \geq O\} \\ &= \min\{k : 2^0 + 2^1 + \dots + 2^{k-1} \geq O/S\} \\ &= \min\{k : 2^k - 1 \geq \frac{O}{S}\} \\ &= \min\{k : k \geq \log_2(\frac{O}{S} + 1)\} \\ &= \left\lceil \log_2(\frac{O}{S} + 1) \right\rceil \end{aligned}$$

Calculation of Q , number of idles for infinite-size object,

89

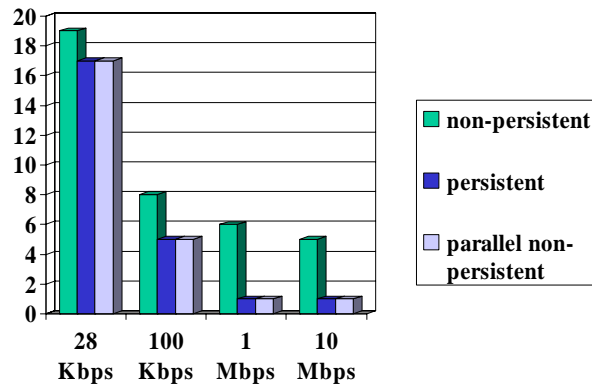
Exemple HTTP

- ❑ **Supposant la page Web composée de:**
 - 1 base HTML page (de taille O bits)
 - M images (O bits chacune)
- ❑ **Non-persistent HTTP:**
 - $M+1$ TCP connections en series
 - *Response time* = $(M+1)O/R + (M+1)2RTT$ + nombre de fois du temps d'attente
- ❑ **Persistent HTTP:**
 - $2 RTT$ to request and receive base HTML file
 - $1 RTT$ to request and receive M images
 - *Response time* = $(M+1)O/R + 3RTT$ + nombre de fois du temps d'attente
- ❑ **Non-persistent HTTP avec X connections parallèles**
 - Suppose M/X integer.
 - 1 TCP connection for base file
 - M/X sets of parallel connections for images.
 - *Response time* = $(M+1)O/R + (M/X + 1)2RTT$ + nombre de fois du temps d'attente

90

HTTP Response time (in seconds)

RTT = 100 msec, O = 5 Kbytes, M=10 and X=5



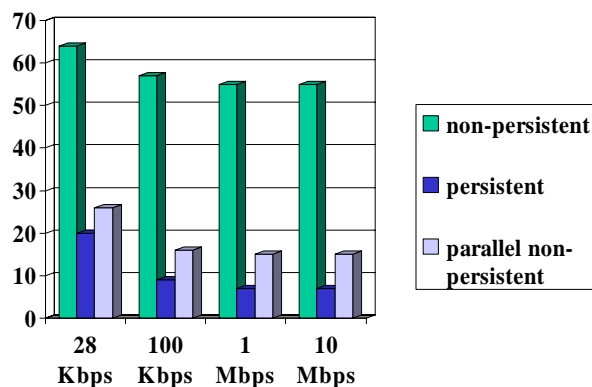
pour une bandwidth plus petite, connection & temps de réponse dominé par le temps de transmission.

Persistent connections seulement donne une amélioration mineur
Sur les connexions parallèles

91

HTTP Response time (in seconds)

RTT = 1 sec, O = 5 Kbytes, M=10 and X=5



Pour un grand RTT, temps de réponse est dominé par les délais d'établissement TCP & slow start. Connexions persistantes donnent maintenant une amélioration importante: particulièrement dans les réseaux de haut débit
« delay • bandwidth »

92