Le modèle TCP/IP

Protocoles avec/sans connexions

- **Protocole avec connexion (logique):**

  Une connexion logique de *bout en bout* (échange d'informations de contrôle) est établie entre deux hôtes qui désirent communiquer. Cette étape est nécessaire avant l'acheminement des données.

Protocoles avec connexions

- L’hôte A signal à l’hôte B qu’il désire établir une connexion (SYN)
- L’hôte B acquitte (ACK, Acknowledge) la demande pour accepter la connexion.
- L’hôte A acquitte l’acquittement… et la transmission des informations peut commencer

Une raison d’établir une connexion dans un réseau à commutation par paquets est de permettre aux hôtes d’échanger des informations utiles par exemple pour re-séquencer les paquets.

Modèle TCP/IP

ISO

Application
Présentation
Session
Transport
Réseau
Liaison
Physique

TCP/IP

Application
Transport (TCP,UDP)
Internet (IP)
Hôte-réseau

TCP

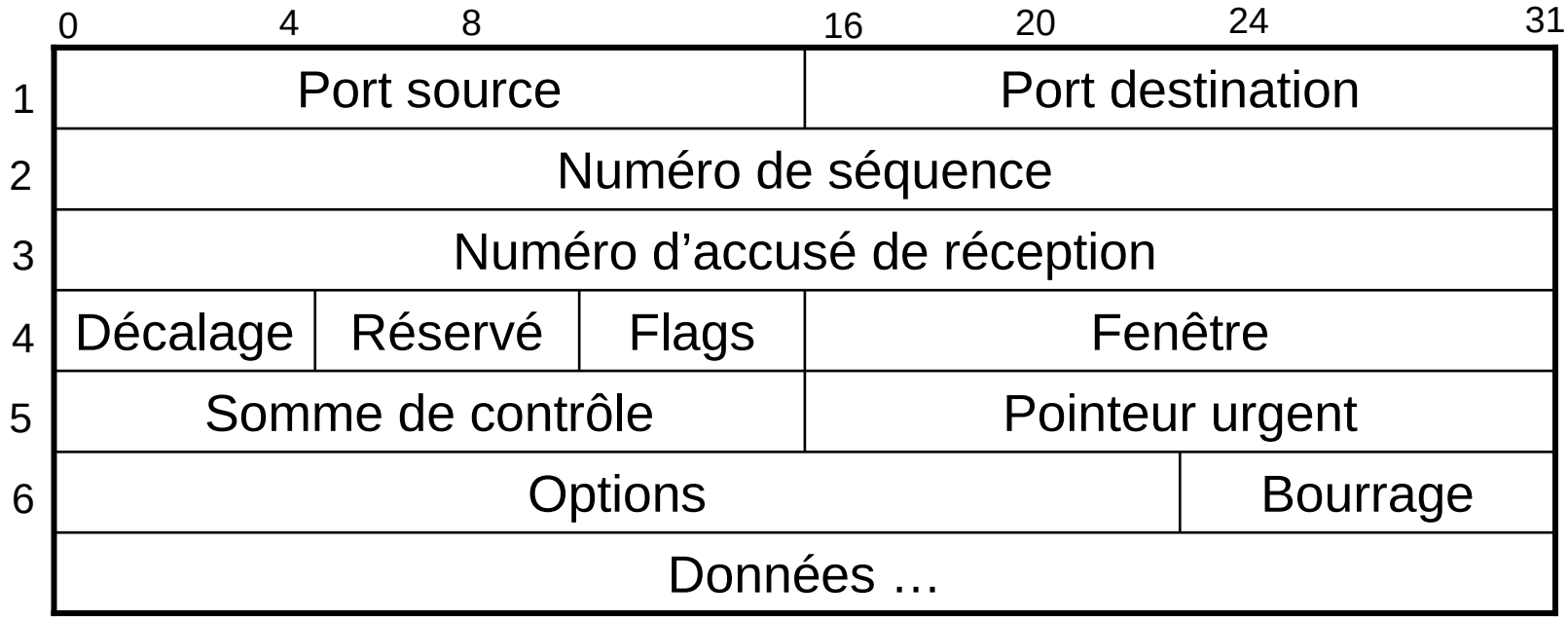
- **La couche transport (Hôte à Hôte):**
  Gère les communications entre entités pairs. Définit deux protocoles.

  – TCP (Transmission Control Protocol) :
  Protocole fiable orienté connexion
    - Contrôle que les données soient acheminées dans le réseau correctement et dans le bon ordre

  Fragmente les messages avant de les transmettre à la couche internet
  Contrôle de flux de données
  – UDP (User datagram Protocol)
  Protocole non fiable, sans connexion ni contrôle de flux
  (utile pour la transmission de la parole)

TCP – couche transport

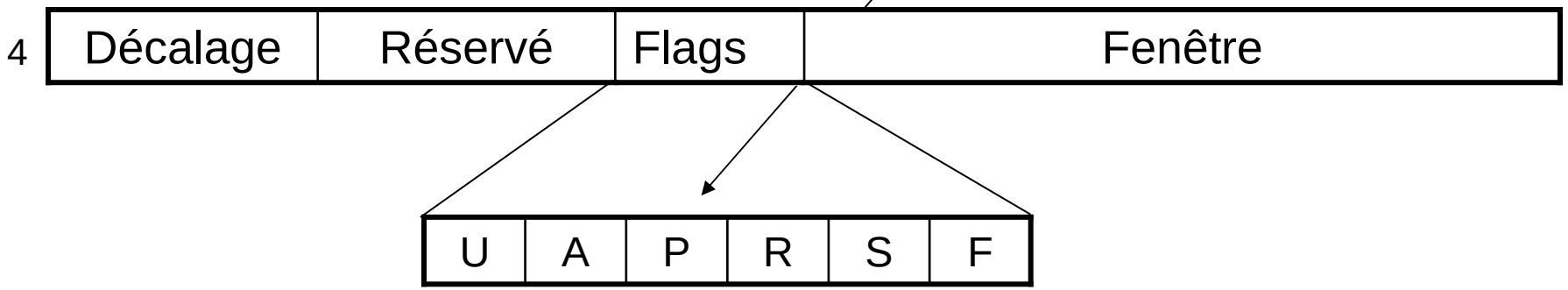
Protocole TCP: format d'un segment TCP



TCP – couche transport

Etablissement de la connexion

Push: il faut remettre les infos à l'application sans utiliser de tampon.



Acknowledge: acquittement

Reset: Réinitialise La connexion

Synchronize: Fragment de synchronisation

Fin: termine la connexion

Urgent:
par exemple en mode interactif on presse [DEL] ou [CTRL-C] oblige le transfert des informations par TCP (pas de tampons)

TCP – couche transport

Etablissement de la connexion:

- Pour synchroniser la connexion l'hôte A active le bit *S* du champ *Flags* du segment TCP transféré. Le champ séquence sert à initialiser le compteur de séquence qui permet de remettre les paquets dans le bon ordre.
- L'hôte B acquitte la tentative de connexion en répondant par un segment TCP dans lequel les bits *S* et *A* du champ *Flags* sont positionnés. Le serveur choisis aussi un numéro de séquence.
- L'hôte A envoie un segment avec le bit *A* du champ *Flags* positionné pour acquitter le segment de B et la transmission des données peux commencer.

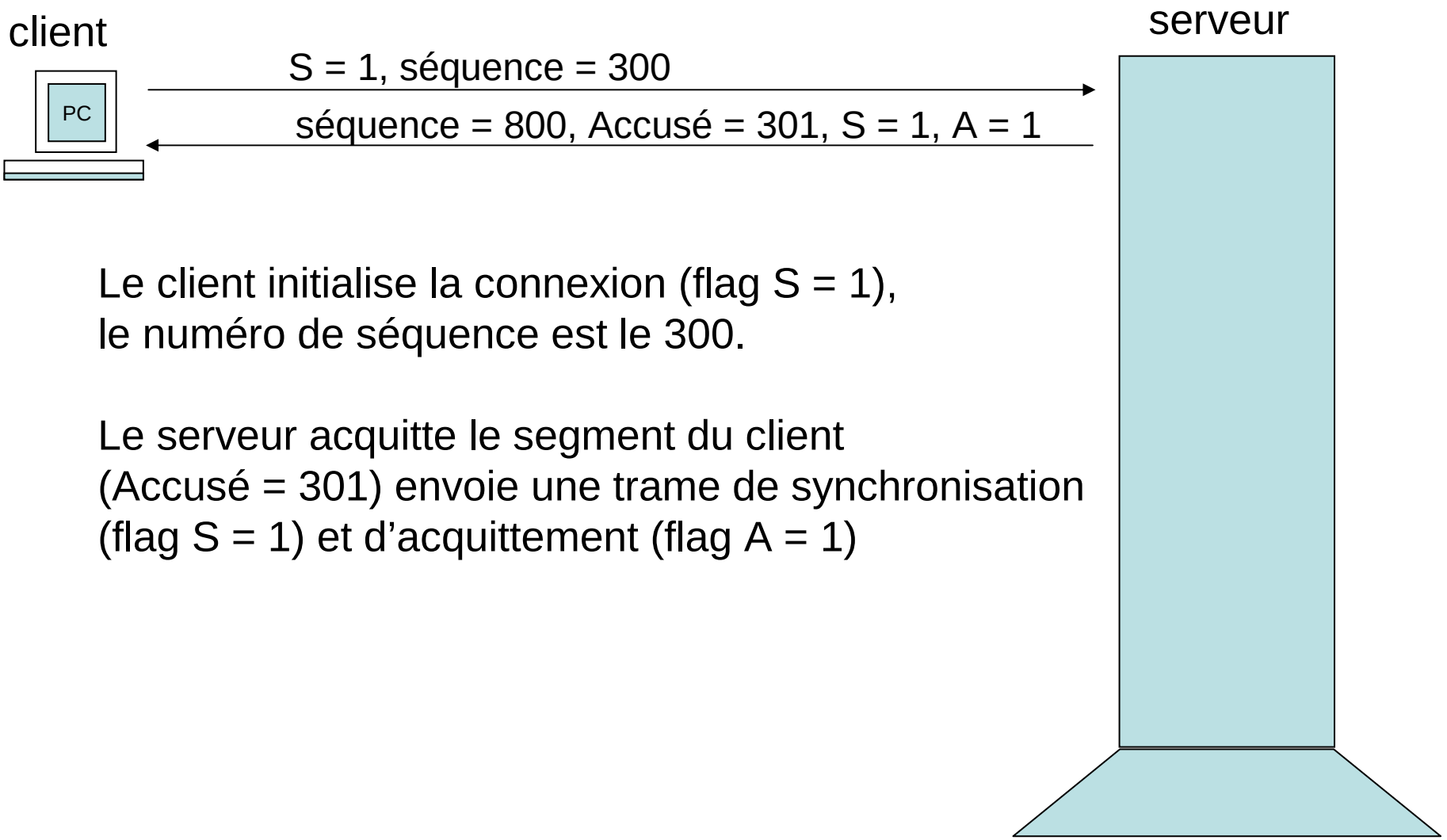
Les deux points de connexion s'appelle ***sockets***.

TCP – couche transport

Déconnexion:

- La déconnexion est identique à la connexion excepté que le bit *F* (FIN, No more data from sender) du champ *Flags* est positionné (à la place de *S*)

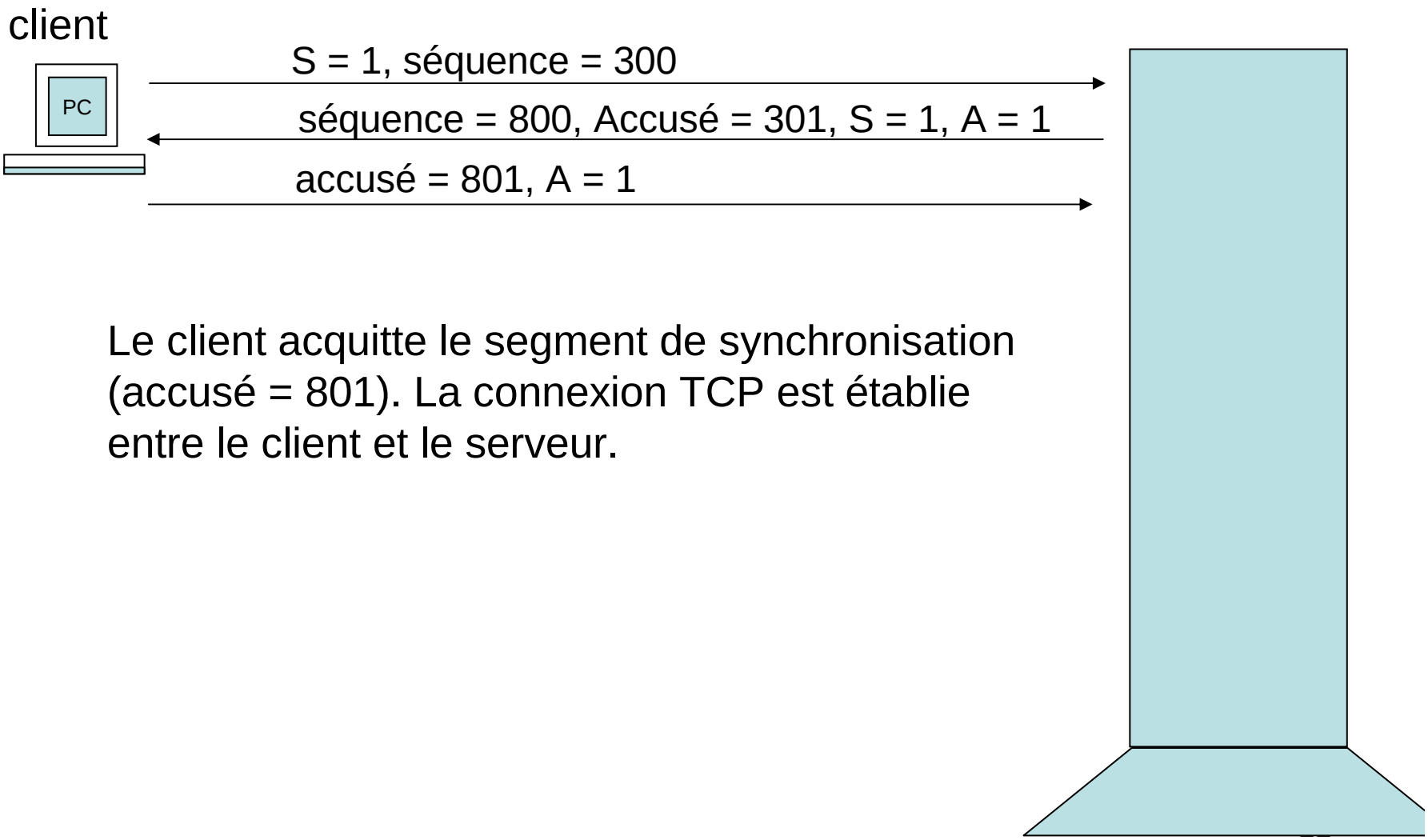
Exemple



Le client initialise la connexion (flag S = 1),
le numéro de séquence est le 300.

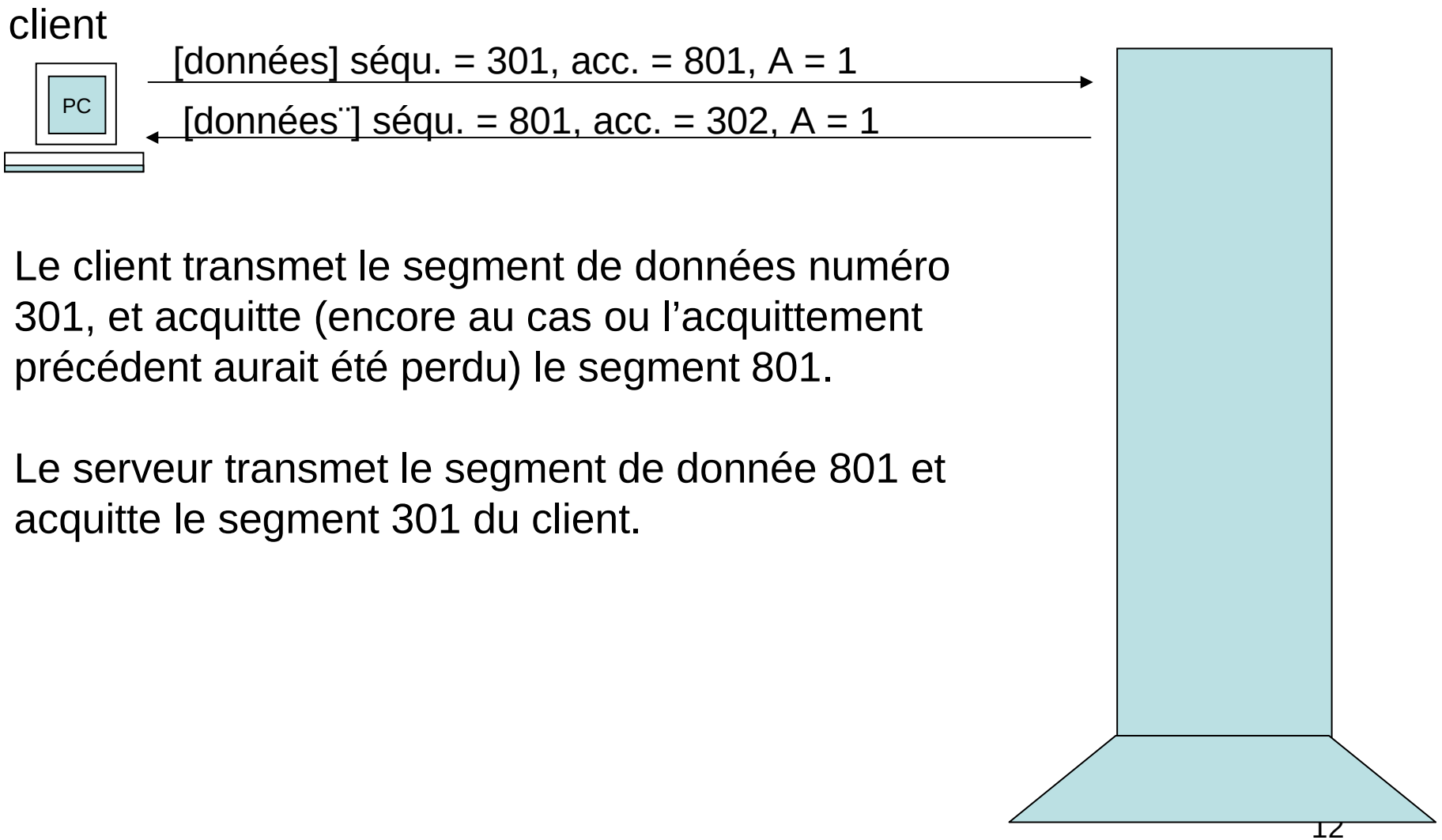
Le serveur acquitte le segment du client
(Accusé = 301) envoie une trame de synchronisation
(flag S = 1) et d'acquittement (flag A = 1)

Exemple



Le client acquitte le segment de synchronisation (accusé = 801). La connexion TCP est établie entre le client et le serveur.

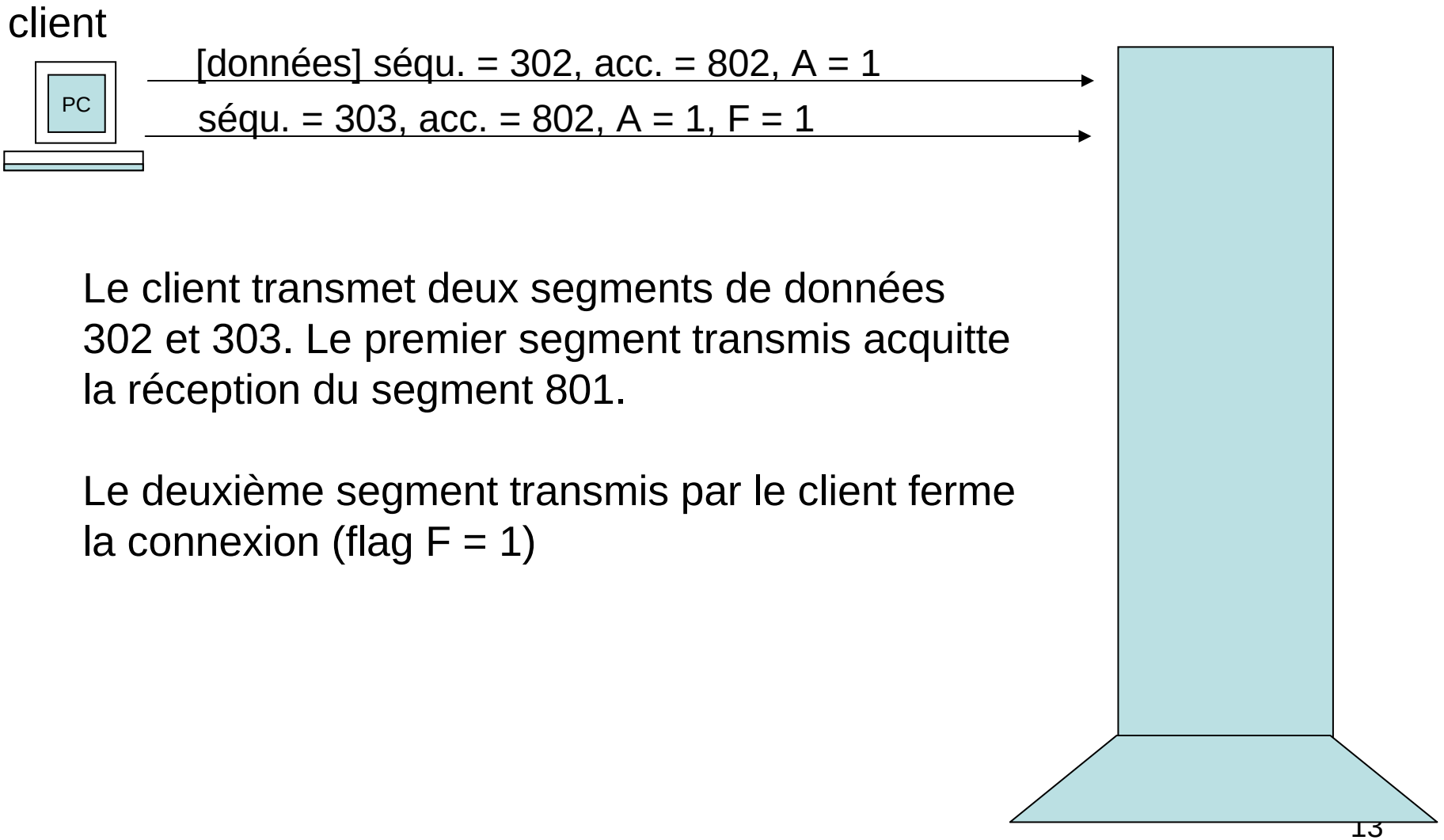
Exemple



Le client transmet le segment de données numéro 301, et acquitte (encore au cas ou l'acquittement précédent aurait été perdu) le segment 801.

Le serveur transmet le segment de donnée 801 et acquitte le segment 301 du client.

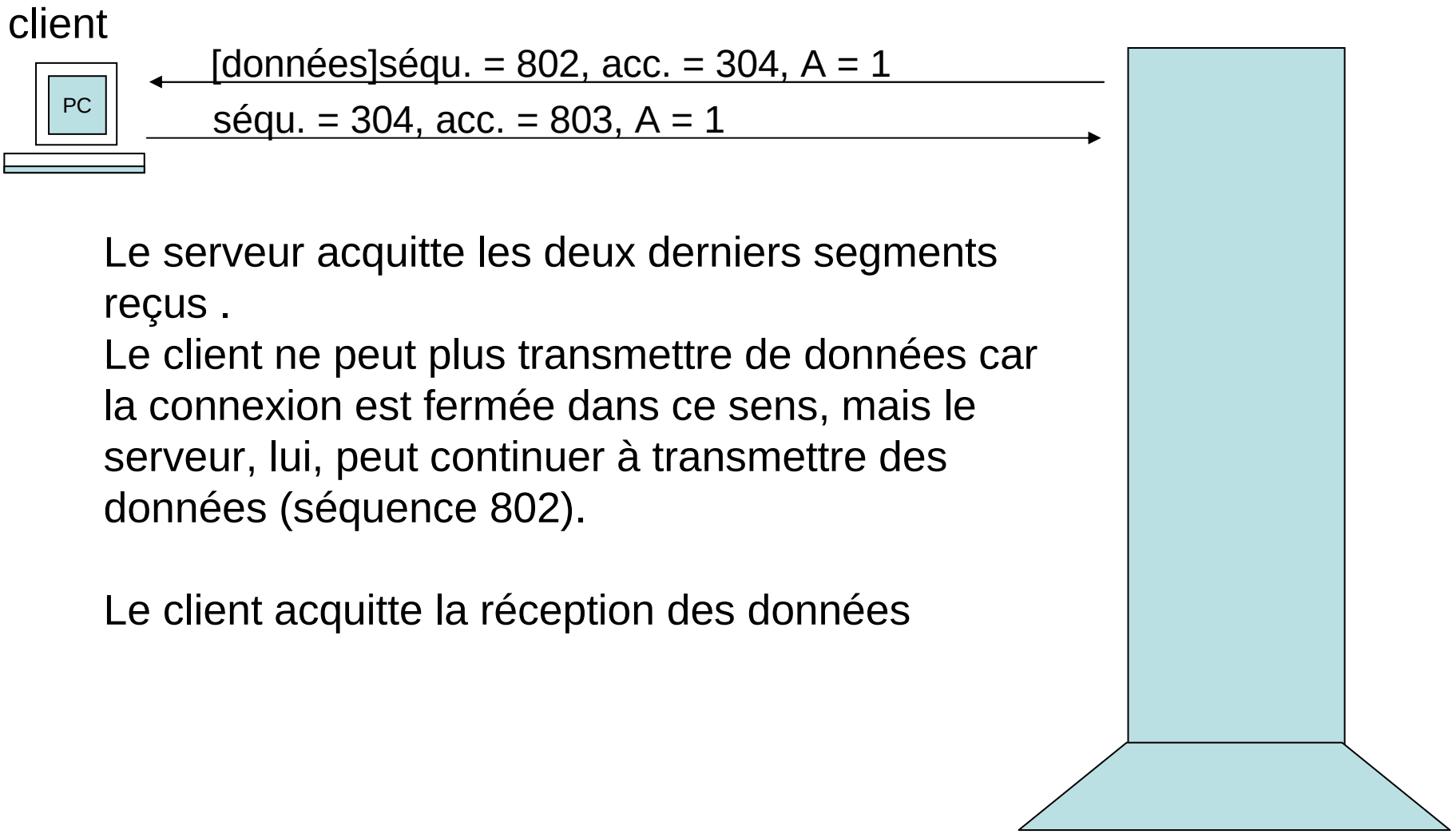
Exemple



Le client transmet deux segments de données 302 et 303. Le premier segment transmis acquitte la réception du segment 801.

Le deuxième segment transmis par le client ferme la connexion (flag F = 1)

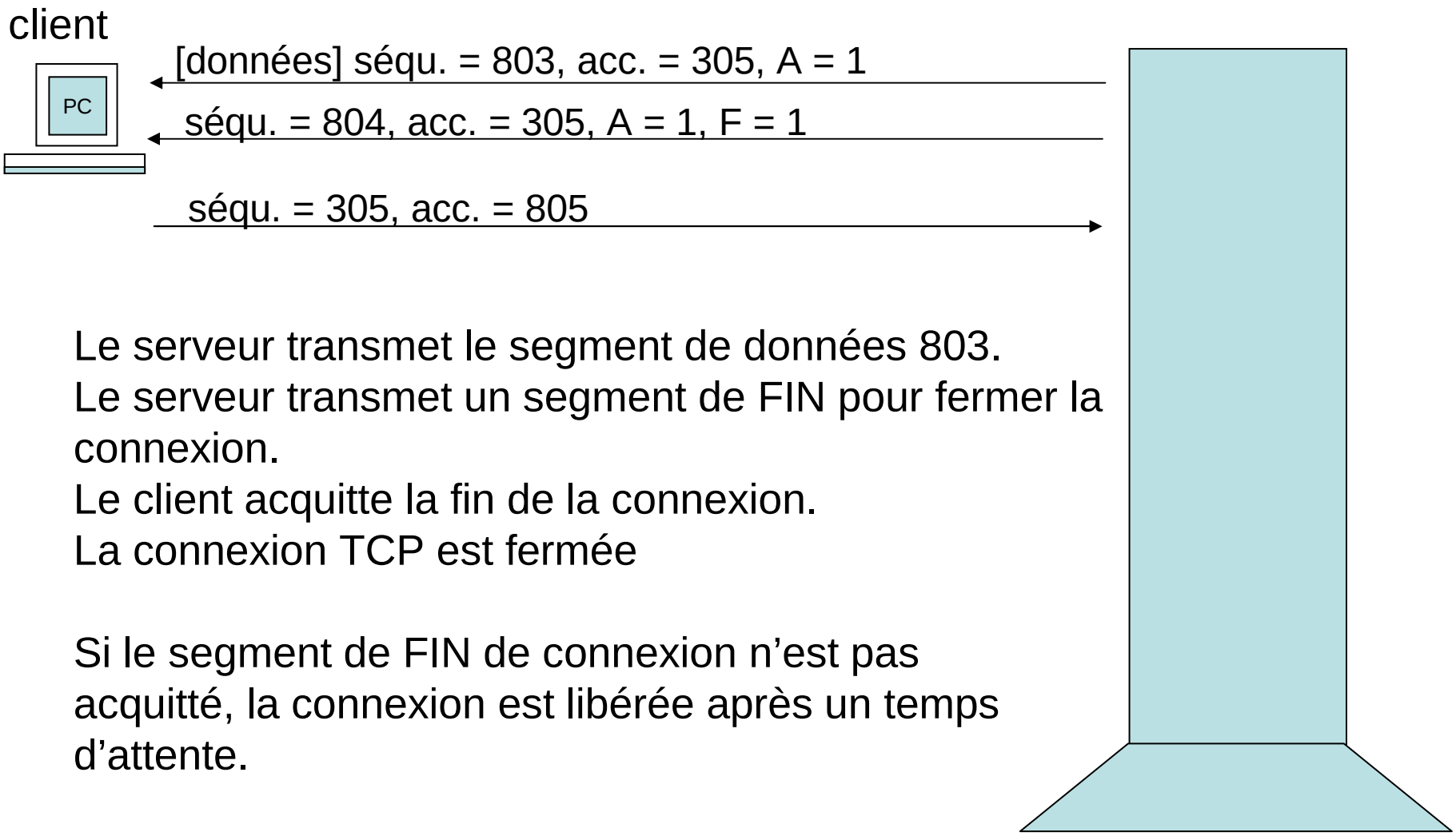
Exemple



Le serveur acquitte les deux derniers segments reçus .
Le client ne peut plus transmettre de données car la connexion est fermée dans ce sens, mais le serveur, lui, peut continuer à transmettre des données (séquence 802).

Le client acquitte la réception des données

Exemple



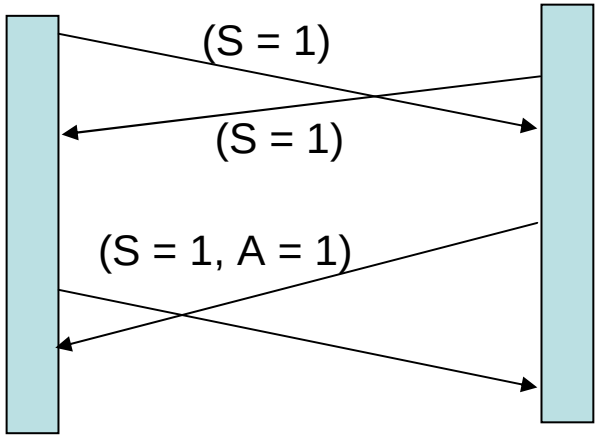
Le serveur transmet le segment de données 803.
Le serveur transmet un segment de FIN pour fermer la connexion.
Le client acquitte la fin de la connexion.
La connexion TCP est fermée

Si le segment de FIN de connexion n'est pas acquitté, la connexion est libérée après un temps d'attente.

Sockets

Pour établir une connexion TCP il faut créer deux points de connexion qui s'appellent ***sockets***.

Chaque socket possède un numéro (adresse) constituée de l'adresse IP de l'ordinateur hôte et du **numéro de port**, sur 16 bits. En particulier, on ne peux jamais établir deux connexions au lieu d'une même si les deux stations établisse une connexion simultanément.



Le numéro de port est local et on peut utiliser un même socket pour établir plusieurs connexions.

Sockets et numéros de ports

Les numéros de port inférieurs à 256 sont appelés **ports réservés** (well-known ports) et sont utilisés par les applications les plus courantes (RFC 1700).
Quelques ports réservés.

7	ECHO	Produit l'écho de ce qui est reçu
21	FTP	File Transfert Protocol, partage et transfert de fichiers
22	SSH	SSH Remote Login Protocol,
23	TELNET	Transfert de données au format ASCII
25	SMTP	Simple Mail transfert Protocol, envoie de messages en utilisant des commandes au format ASCII
67	BOOTPC	Bootstrap Protocol, configuration dynamique au boot, adresse IP hôte ou machine de service, nom d'un fichier à exécuter
80	HTTP	Protocole HTTP
…	…	…

TCP – couche transport

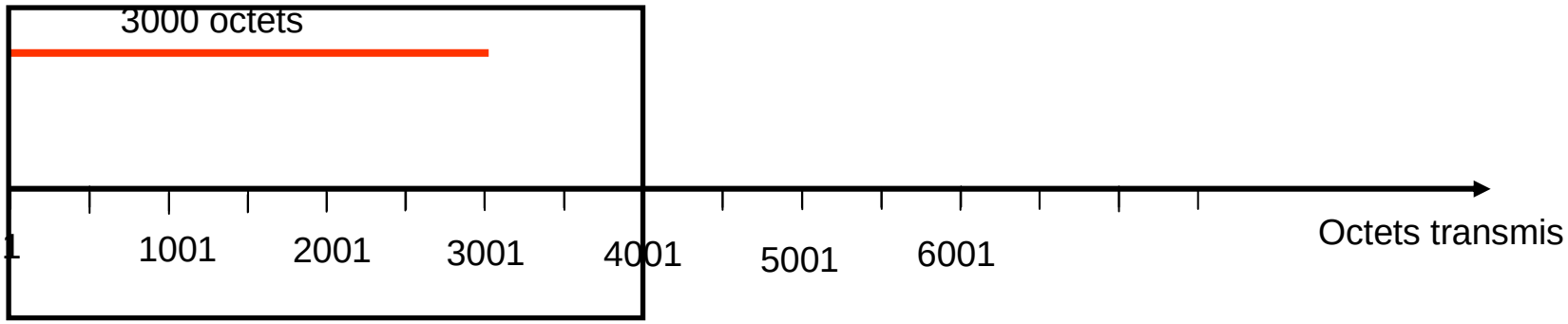
Contrôle de flux:

- Le champ *fenêtre* du segment TCP contient le nombre d'octets que le récepteur est capable d'accepter. Tant que le nombre de segments *émis et pas encore acquittés* (c'est-à-dire quelque part dans le sous-réseau) est inférieur à la taille de la fenêtre, l'émetteur peut continuer à émettre.

  1. A l'initialisation, l'émetteur choisit un numéro de séquence ISN (Initial sequence Number) en activant le bit *S* du champ *Flags*, les séquences sont numérotés séquentiellement.
  2. Le récepteur acquitte la réception des segments reçus en indiquant le numéro de séquence qu'il doit recevoir dans le champ *Numéro d'accusé de réception*.
  3. Le champ *Fenêtre* indique le nombre maximal de octets que l'émetteur peut transmettre sans recevoir d'acquittement.

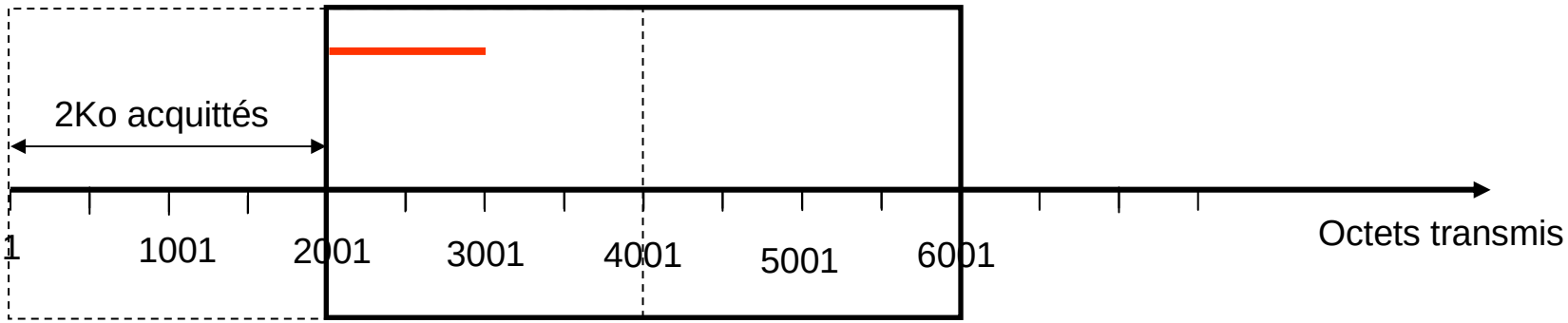
TCP – couche transport

Contrôle de flux: taille de la fenêtre de réception = 4000 octets



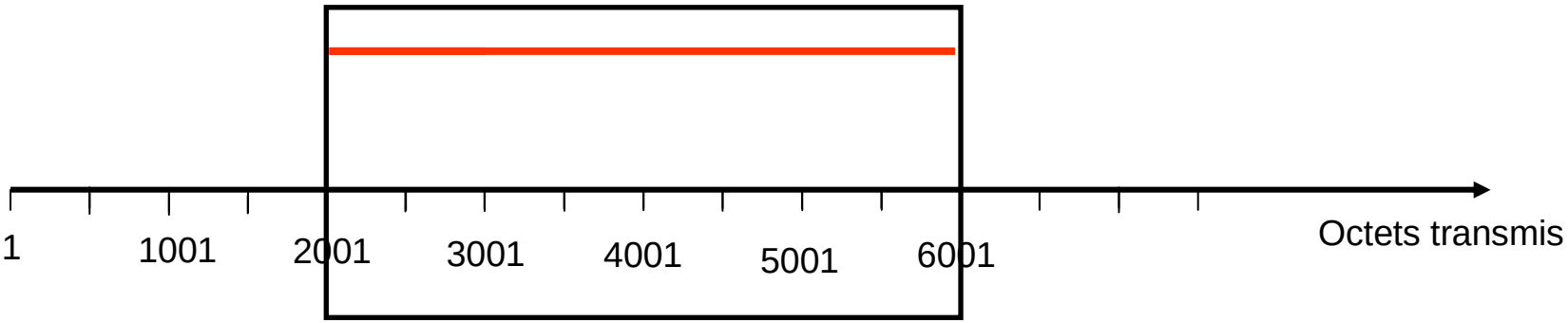
TCP – couche transport

Contrôle de flux: taille de la fenêtre de réception = 4000 octets



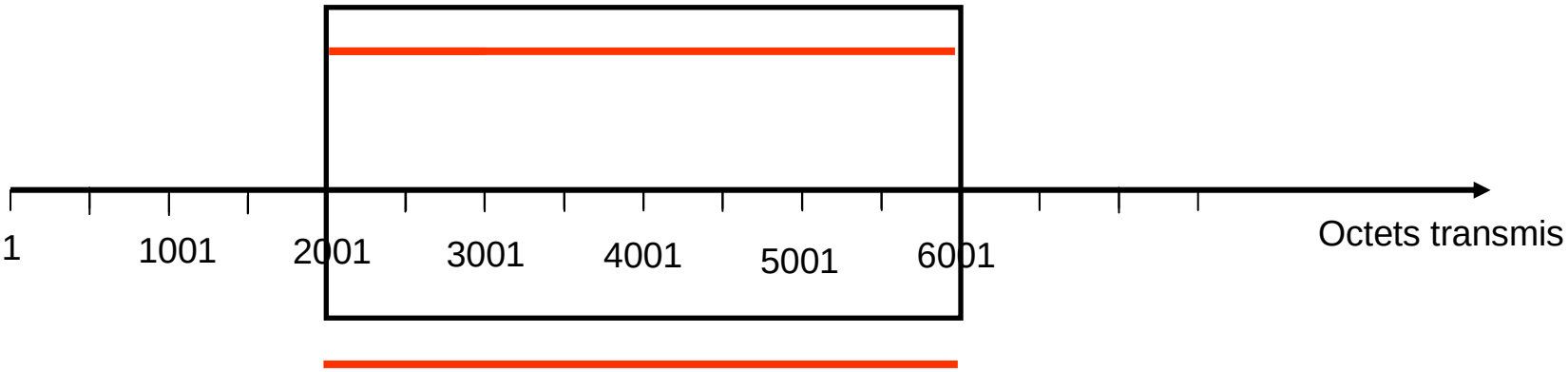
TCP – couche transport

Contrôle de flux: taille de la fenêtre de réception = 4000 octets



TCP – couche transport

Contrôle de flux: taille de la fenêtre de réception = 4000 octets



Retransmission des octets non acquittés après un délai d'attente. Possibilité de diminuer la vitesse de transmission (bits/sec).

TCP – couche de transport

Au niveau du protocole de transport, des tampons d'émission et de réception son gérés.

Le tampon d'émission contient

1. Les données à transmettre générées par l'application.
2. Les données transmise mais pas encore acquittées

Le tampon de réception contient:

1. Les données reçues dans l'ordre, ces données sont disponibles pour l'application
2. Les données reçues dans le désordre, TCP attend les données manquantes

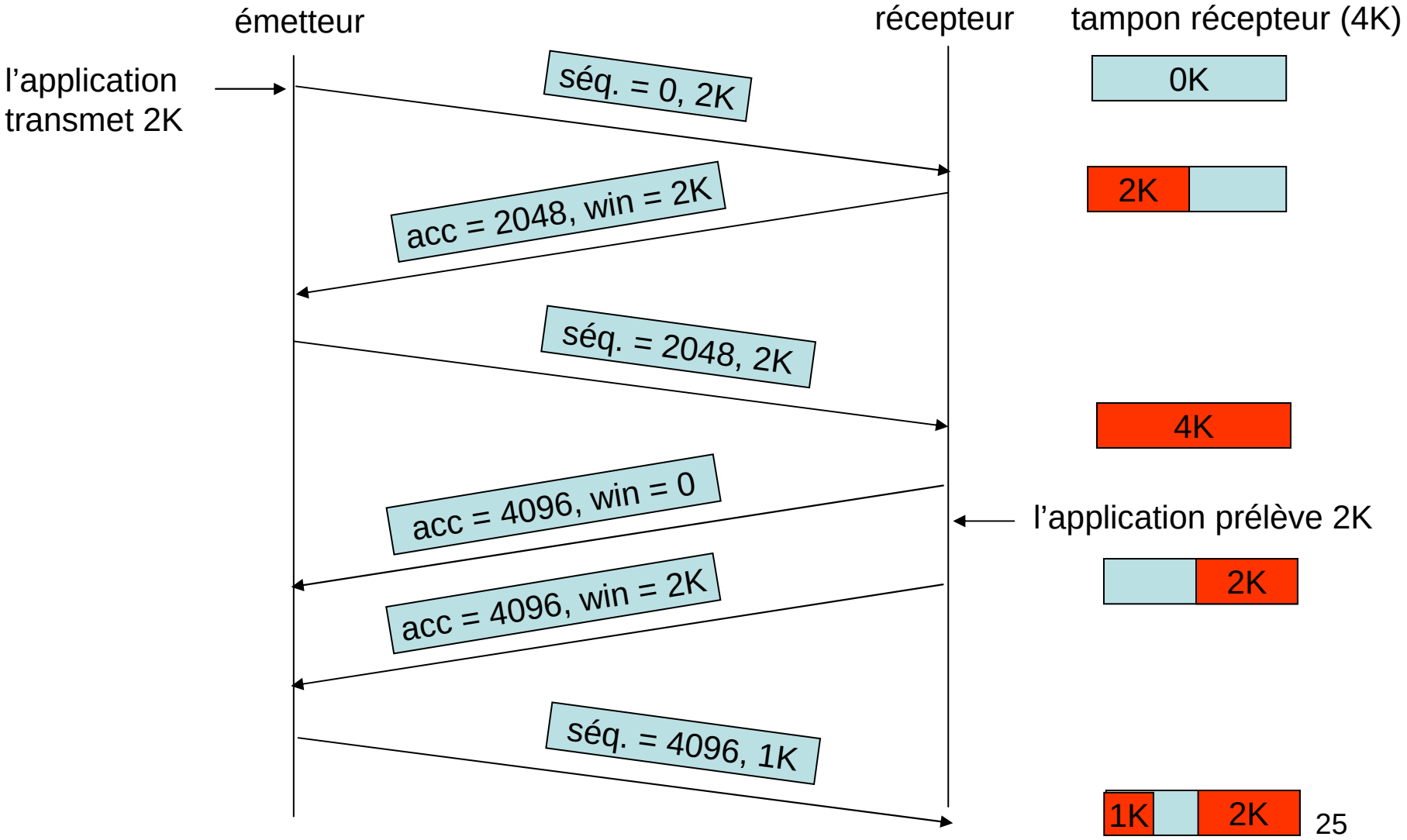
L'émetteur doit tenir compte de la taille de la fenêtre du récepteur et ne pas transmettre plus de données que possible.

TCP - transmission

Au niveau de TCP, la gestion des fenêtres n'est pas directement liée aux acquittements mais plutôt à la disponibilité du tampon de réception.

Par exemple, le récepteur annonce une fenêtre de 4K octets. Après réception de 2K octets de données, il acquitte la réception et annonce une taille de fenêtre de réception de 2K octets si l'application n'a pas prélevé les données.

Fenêtre de réception



TCP - transmissions

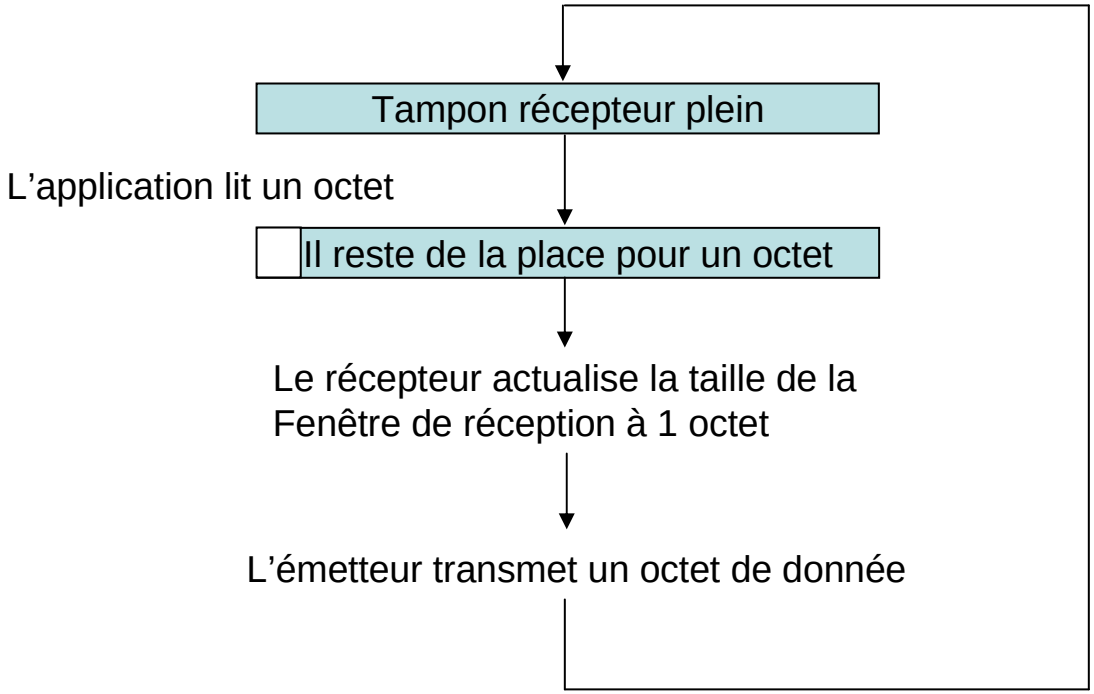
En général, les implémentations de TCP retardent les transmission des acquittements de 500ms dans l'espoir que des données vont être générées et pourront être transmises en même temps. De même pour les opérations d'actualisation des fenêtres (déplacement,…) et l'écho.

L'algorithme de Nagle est utilisé lorsque l'application génère des données octets par octets. Par exemple, lors d'une session Telnet. La stratégie consiste à transmettre le premier octet et à accumuler les autres octets jusqu'à la réception de l'acquittement. Ensuite, on transmet les octets accumulés dans une segment TCP et on continue.

Ce mécanisme est désactivé lorsque l'on utilise des applications *X windows,* par exemple, car il résulte des mouvements saccadés des fenêtres

TCP transmissions

Une situation similaire se produite lorsque l'application coté récepteur lit les données octets par octets.



TCP transmission

Dans les deux situations précédentes, on transmet 41 octets (20 octets pour l'entête TCP, 20 octets pour l'entête IP) au total par octet de données.

La **solution de Clark** consiste à empêcher le récepteur d'actualiser la fenêtre de réception pour un seul octet. En pratique, le récepteur transmet une actualisation de la taille de la fenêtre si son tampon est à moitié vide ou s'il dispose d'assez de place pour stocker un segment correspondant au plus grand segment reçu (on choisit le plus petit des deux).

TCP transmissions

Le récepteur peut aussi accumuler les données avant de les transmettre à l'application. Cela évite de vider le tampon octet par octet.

Le récepteur acquitte tous les paquets, le protocole est du type selective repeat. C'est-à-dire que si l'émetteur reçoit un paquet qui n'est pas dans l'ordre il mémorise le paquet mais transmet un acquittement avec le numéro d'accusé de réception qui correspond au paquet attendu.

L'émetteur mémorise les paquets transmis et non acquittés. A chacun de ces paquets est associé un temporisateur. Si le temporisateur expire – le paquet n'a pas été acquitté (assez vite) alors l'émetteur retransmet le paquet.

Fenêtre de congestion

TCP permet une gestion dynamique du contrôle de flux qui utilise une deuxième **fenêtre de congestion**.

Le nombre d'octets transmis sans acquittement correspond au minimum indiqué par les deux fenêtres – **de réception et de congestion.**

La fenêtre de congestion permet à l'émetteur de limiter le nombre d'octets transmis s'il observe que le récepteur ne peux pas traiter les données à la vitesse indiquée par la fenêtre de transmission.

La taille de la fenêtre de congestion double chaque fois que les données sont correctement acquittées (slow-start).

Par exemple, la fenêtre de congestion est initialisée à 1024, puis 2048, puis 4096, puis 8192. Si l'émetteur observe que les données ne sont pas correctement acquittée avec une fenêtre de congestion de 8196 octets, il l'a fixe à 4096.

La croissance de la fenêtre est exponentielle

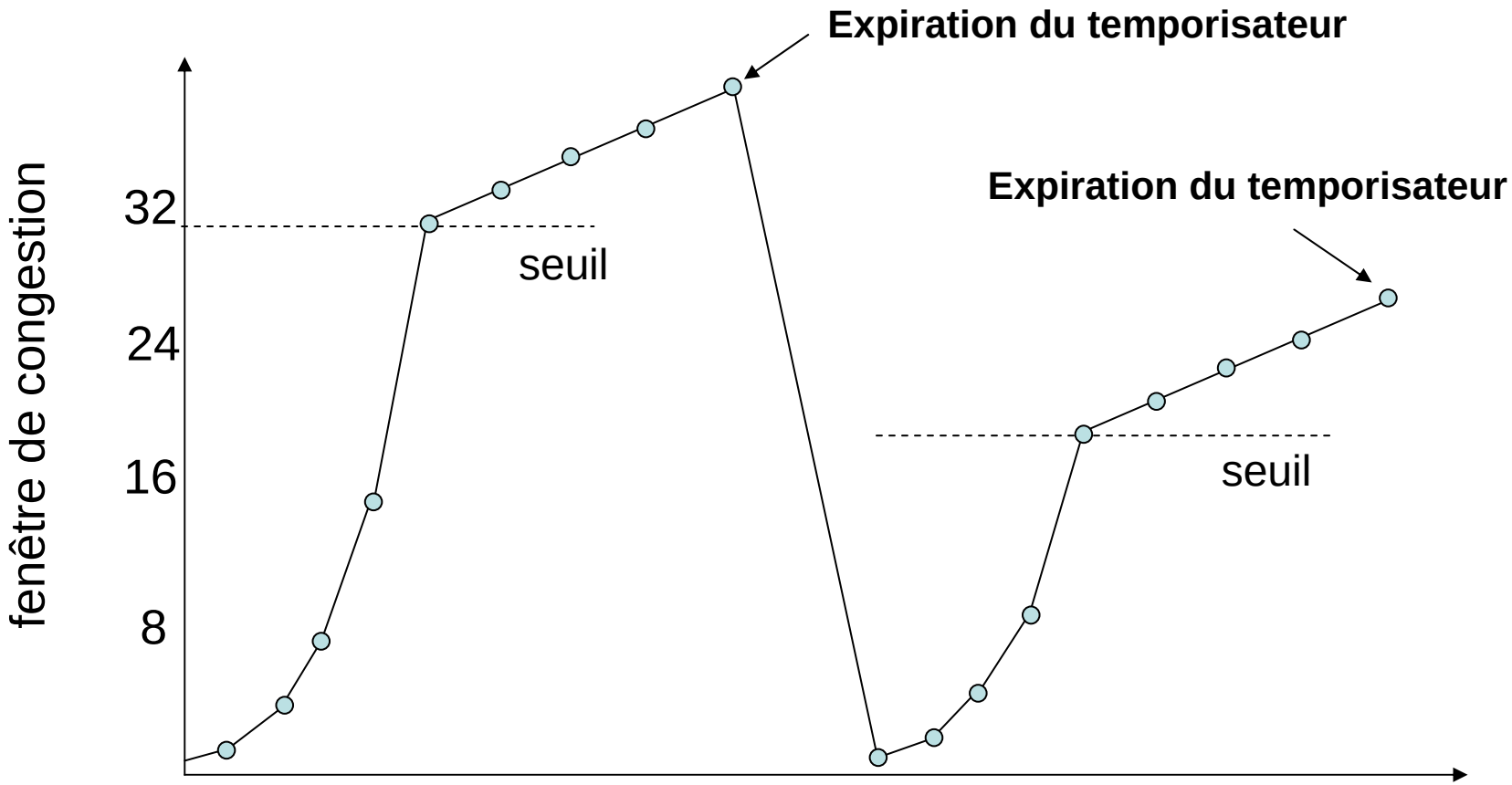
Seuil d’évitement de congestion

L’algorithme implanté pour la gestion du flot utilise en plus des fenêtres de transmission et de congestion **un seuil d’évitement des congestions** (threshold).

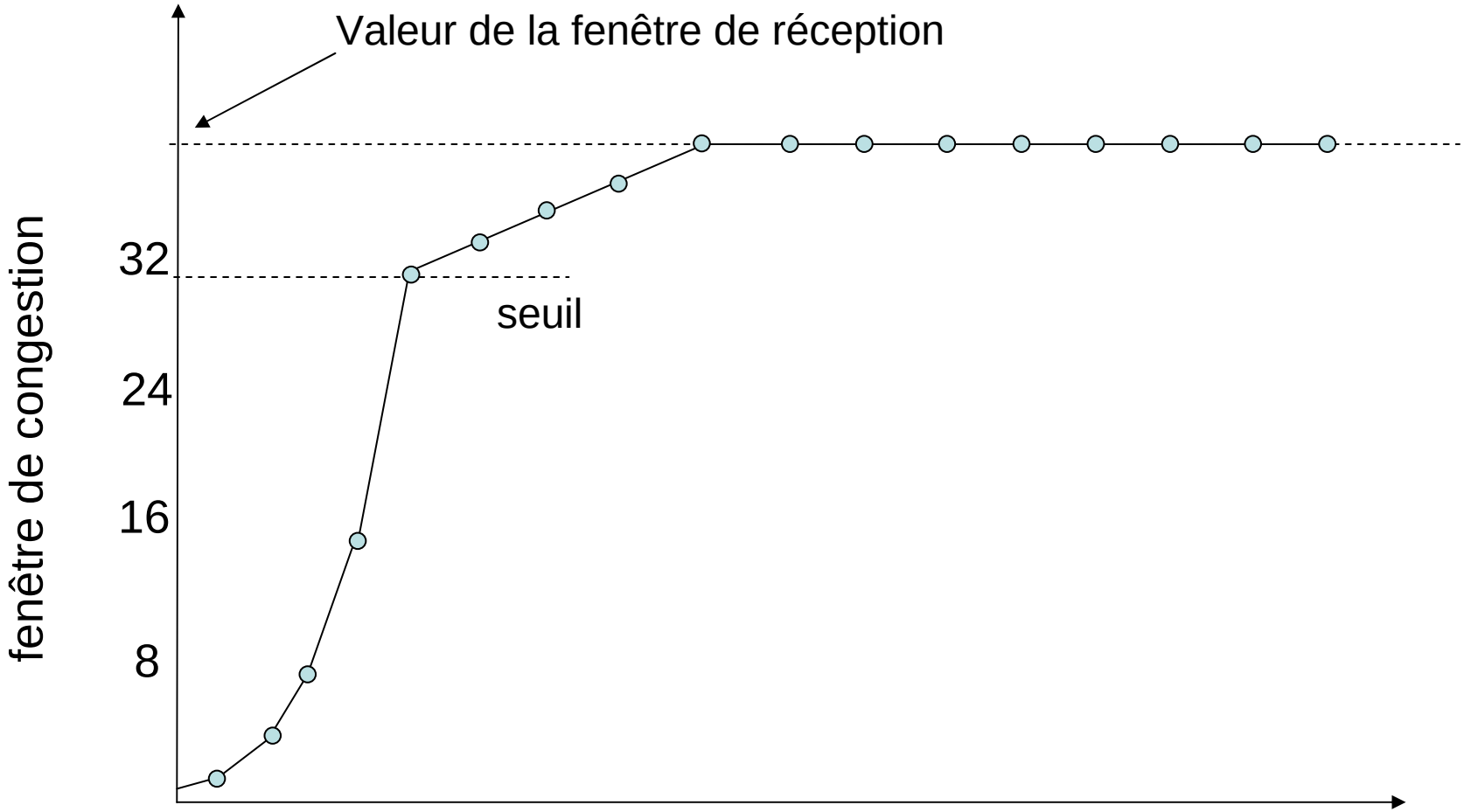
Le flot de données augmente d’abord exponentiellement en doublant la taille de la fenêtre de congestion chaque fois que les données sont correctement acquittées.

Lorsque la taille de la fenêtre de congestion atteint le seuil d’évitement de congestion, la croissance est linéaire.

Contrôle de congestion



Contrôle de congestion



S'il n'y a pas expiration du timer, la fenêtre de congestion continue de grandir jusqu'à atteindre la taille de la **fenêtre de réception**.

Fast recovery

Une situation courante est qu'un paquet x soit perdu et que les paquets suivants x+1, x+2, soient reçus.

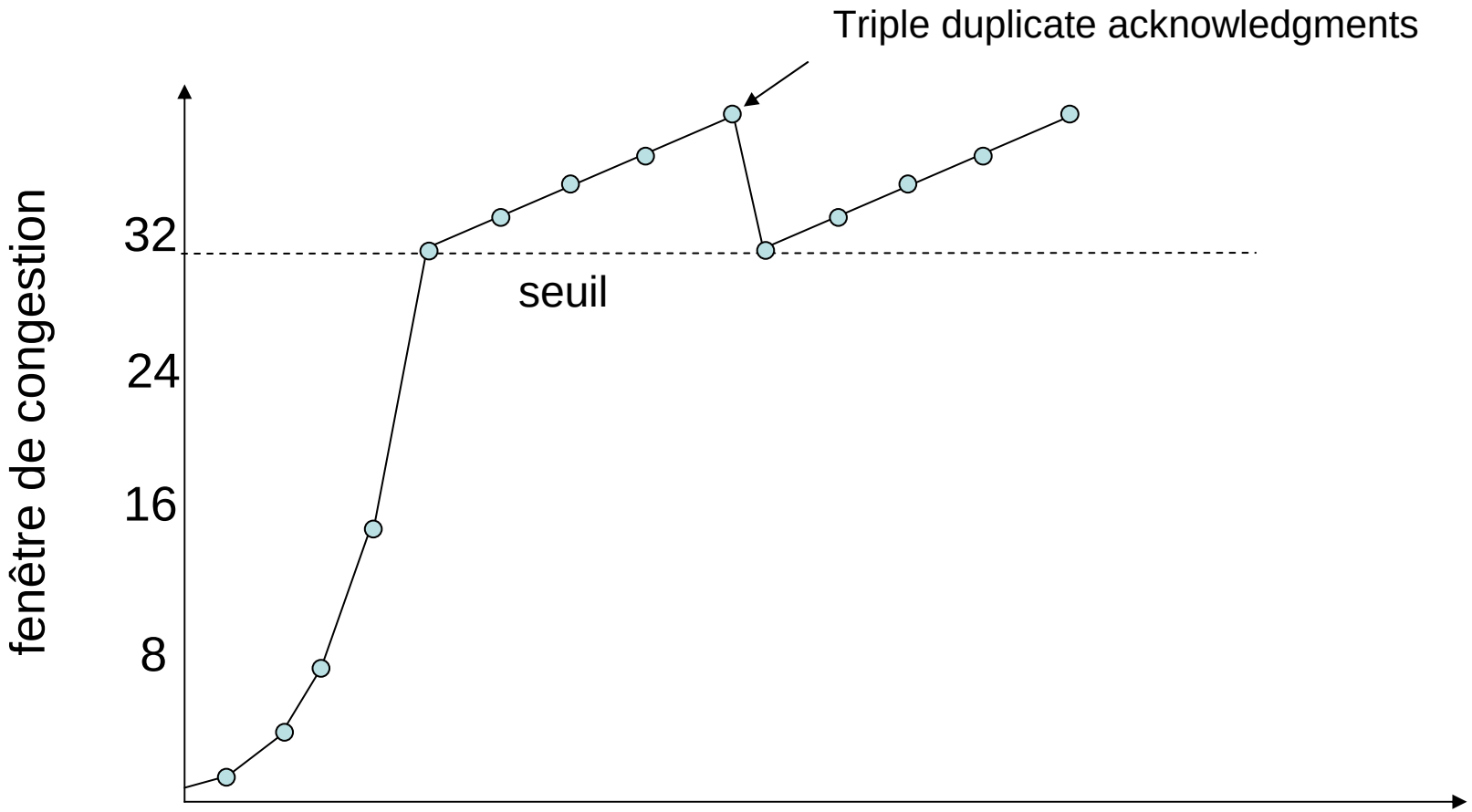
Dans ce cas à réception du paquet x+1 le récepteur mémorise le paquet et transmet un accusé de réception avec le numéro x – puisque ce paquet n'a pas été reçu.

Idem à la réception des paquets x+2, x+3, …

A la réception de 3 accusés de réception avec le même numéro x l'émetteur retransmet le paquet x. C'est le **fast recovery**. Après réception, la fenêtre de congestion est initialisée à la valeur du seuil et la croissance linéaire reprend.

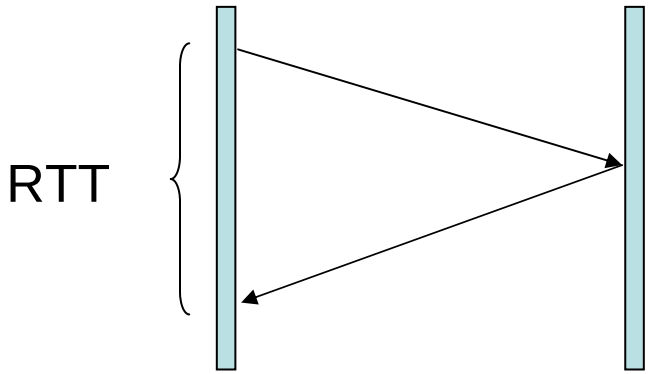
A réception du paquet x le récepteur acquitte les paquets reçus (x+4).

Fast recovery



Round Trip Time

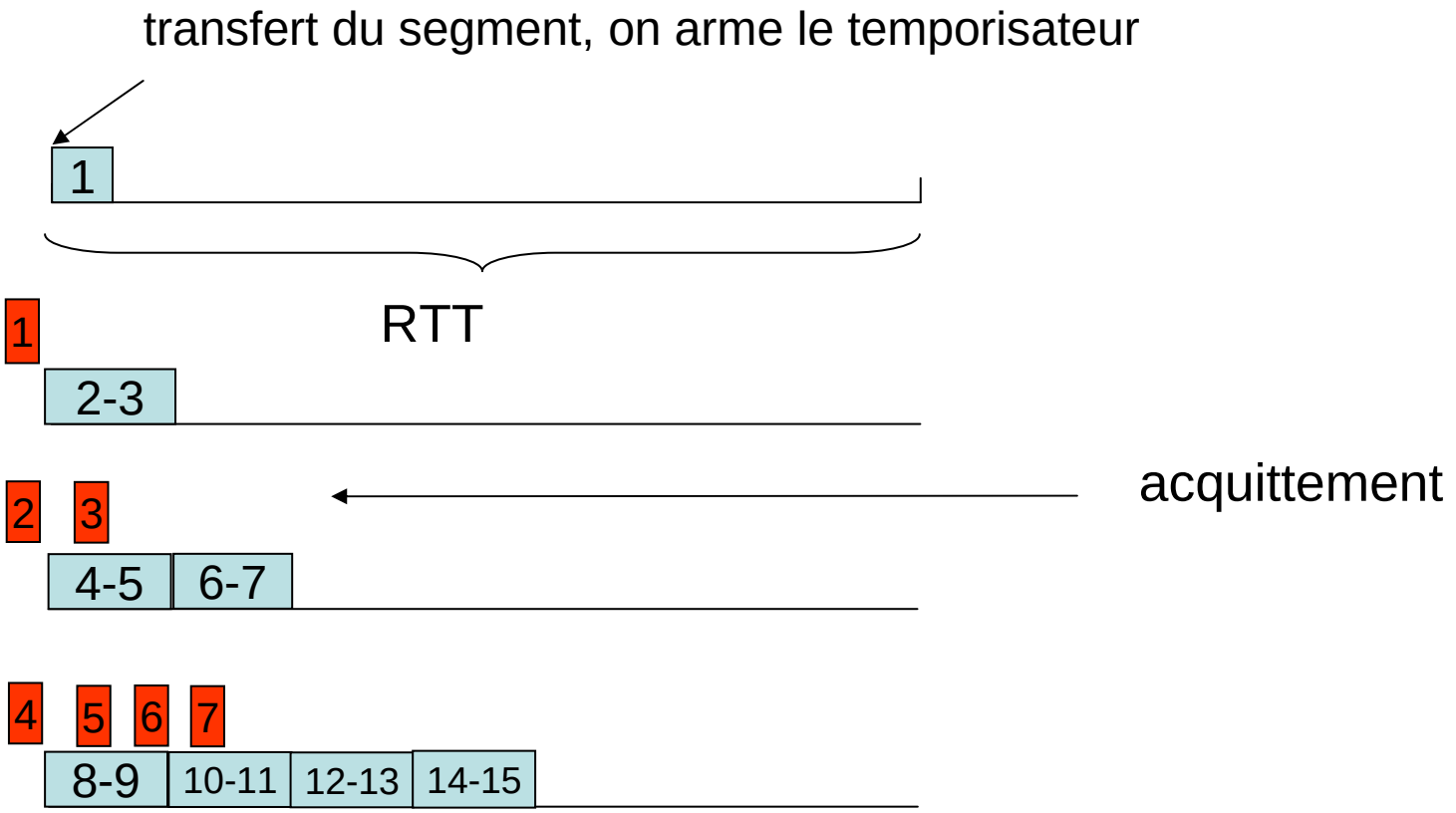
Le temporisateur de retransmission estime le temps de boucle, c'est-à-dire le temps nécessaire au retour de l'acquittement



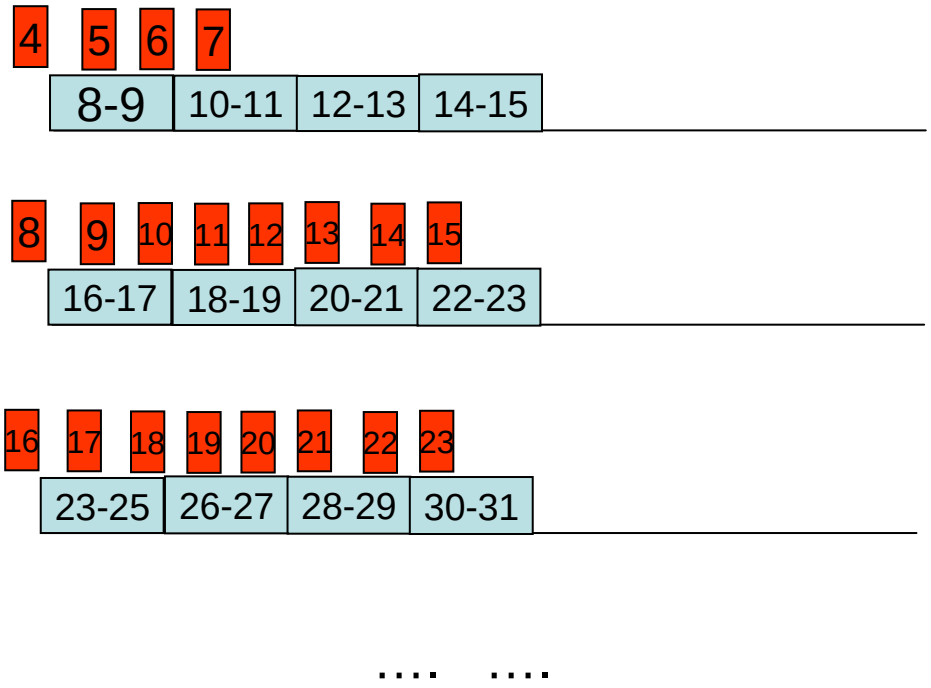
RTT: Round Trip Time

Slow-start

Les différentes étapes du protocole de contrôle de flot et évitement de congestion peut se représenter comme:

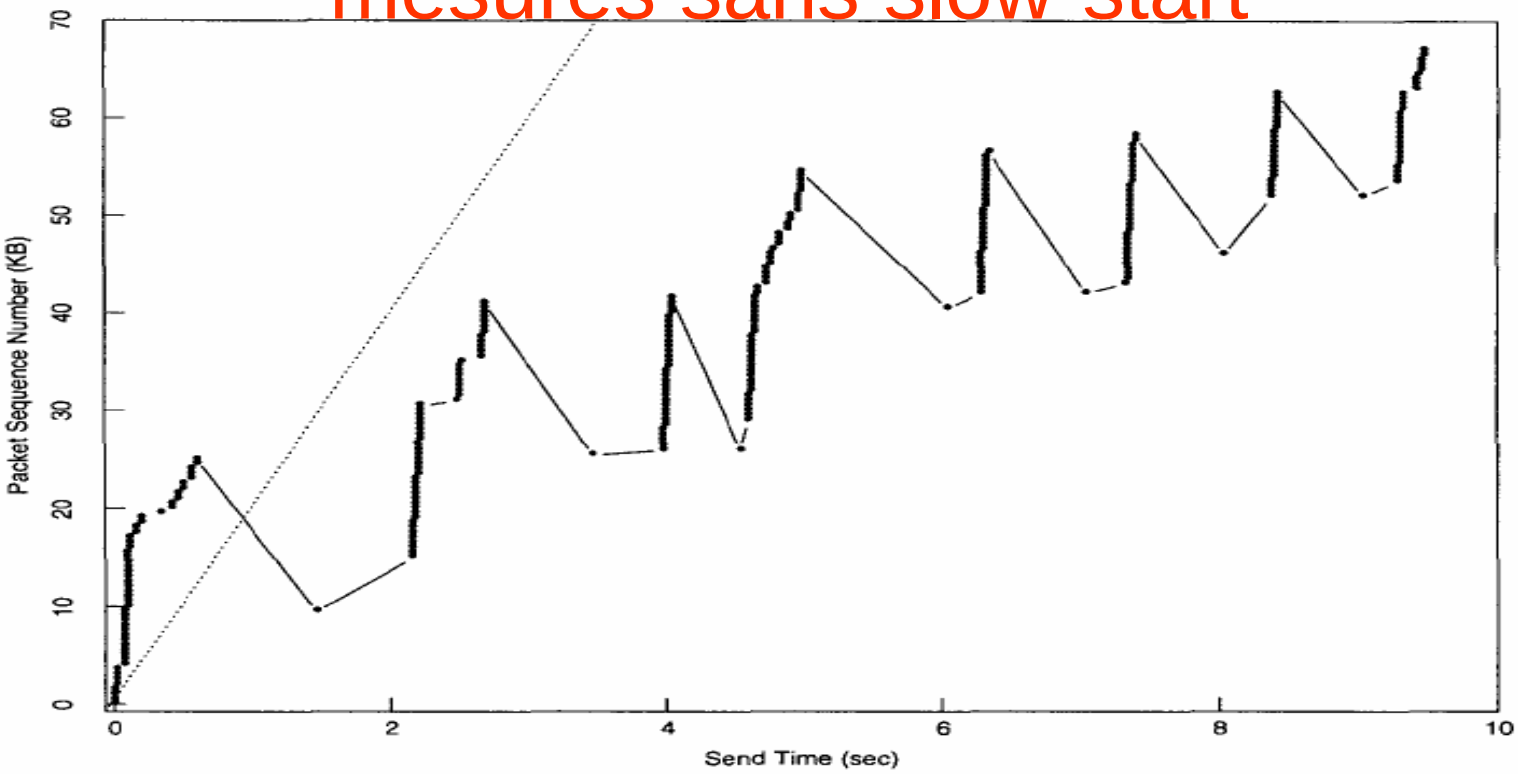


Slow-start



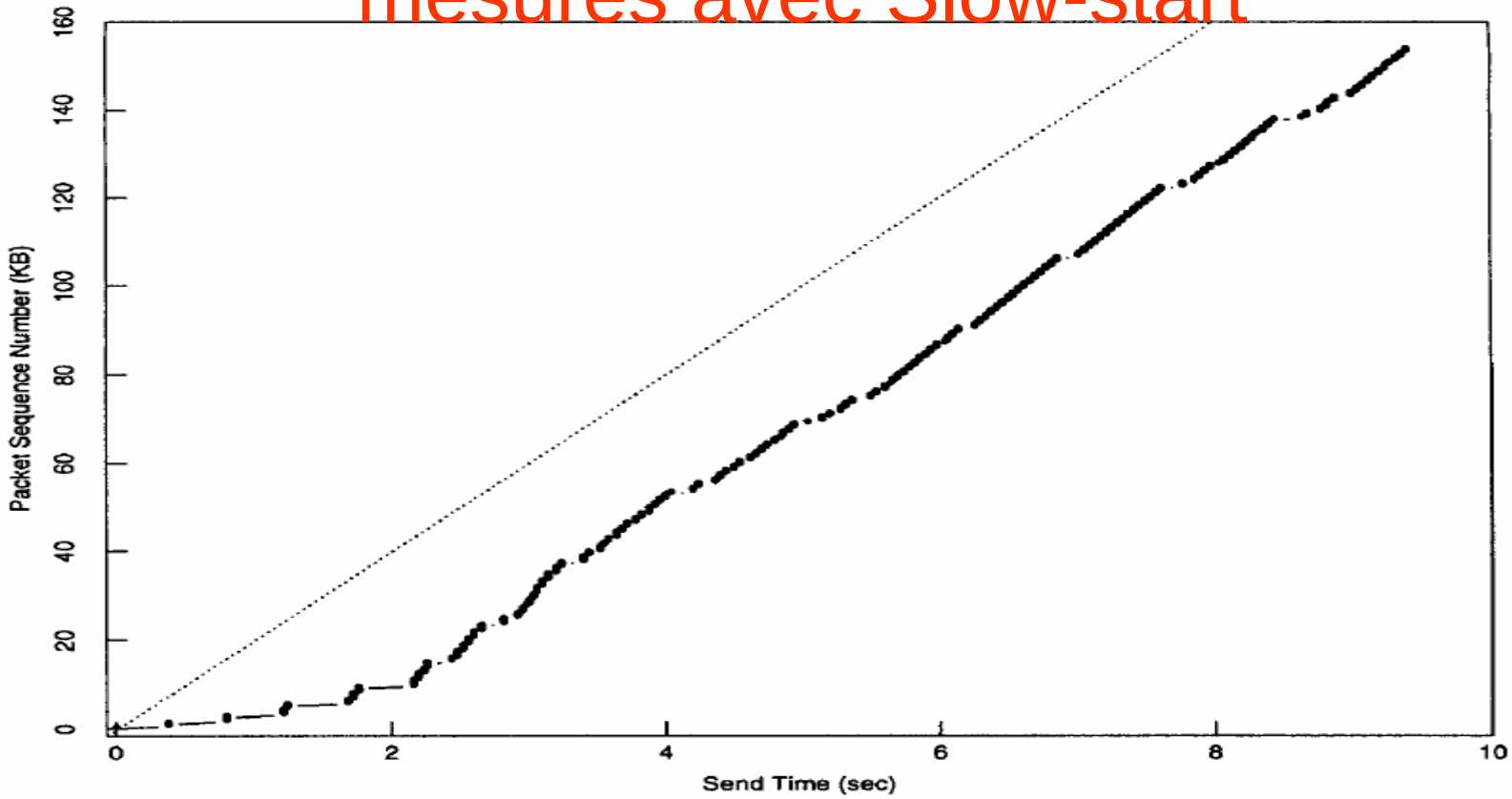
Le flow est constant limité par la fenêtre de congestion et la fenêtre de réception.

Jacobson 1986
mesures sans slow-start



Trace data of the start of a TCP conversation between two Sun 3/50s running Sun OS 3.5 (the 4.3BSD TCP). The two Suns were on different Ethernets connected by IP gateways driving a 230.4 Kbs point-to-point link (essentially the setup shown in fig. 7).
Each dot is a 512 data-byte packet. The x-axis is the time the packet was sent. The y-axis is the sequence number in the packet header. Thus a vertical array of dots indicate back-to-back packets and two dots with the same y but different x indicate a retransmit. 'Desirable' behavior on this graph would be a relatively smooth line of dots extending diagonally from the lower left to the upper right. The slope of this line would equal the available bandwidth. Nothing in this trace resembles desirable behavior.
The dashed line shows the 20 KBps bandwidth available for this connection. Only 35% of this bandwidth was used; the rest was wasted on retransmits. Almost everything is retransmitted at least once and data from 54 to 58 KB is sent five times.

Jacobson 1986
mesures avec Slow-start



Same conditions as the previous figure (same time of day, same Suns, same network path, same buffer and window sizes), except the machines were running the 4.3+ TCP with slow-start.
No bandwidth is wasted on retransmits but two seconds is spent on the slow-start so the effective bandwidth of this part of the trace is 16 KBps — two times better than figure 3. (This is slightly misleading: Unlike the previous figure, the slope of the trace is 20 KBps and the effect of the 2 second offset decreases as the trace lengthens. E.g., if this trace had run a minute, the effective bandwidth would have been 19 KBps. The effective bandwidth without slow-start stays at 7 KBps no matter how long the trace.)

Remarques

- L’algorithme slow-start n’est pas lent … le nombre de paquets augmente exponentiellement.
- L’algorithme est adaptatif, c’est-à-dire que le flot de paquets transmis est régulé en fonction des capacités du réseau et de la station réceptrice.
- Les acquittements sont utilisés comme une horloge qui est générée par la station réceptrice.
- La fenêtre de réception permet au récepteur de contrôler le flot.
- La fenêtre de congestion permet à l’émetteur de contrôler le flot en fonction de la charge du réseau

Estimation de RTT

Le temps de boucle RTT n'est pas constant, en général il fluctue autour d'une valeur moyenne (variable aléatoire)

L'estimation du temps de boucle RTT se fait de manière récursive.

On dispose d'une première approximation RTT,

On transmet un segment et on mesure le temps nécessaire à l'acquittement pour arriver, M,

On met à jour RTT selon

$$RTT_{i+1} = \alpha RTT_i + (1-\alpha)M_i$$

Où M_i est le temps de boucle mesuré. En pratique on utilise

$$\alpha = \frac{7}{8}$$

Estimation de RTT

Pour modéliser le problème de l'estimation du temps de boucle RTT, on suppose que les valeurs mesurées M_i sont des variables aléatoires satisfaisant

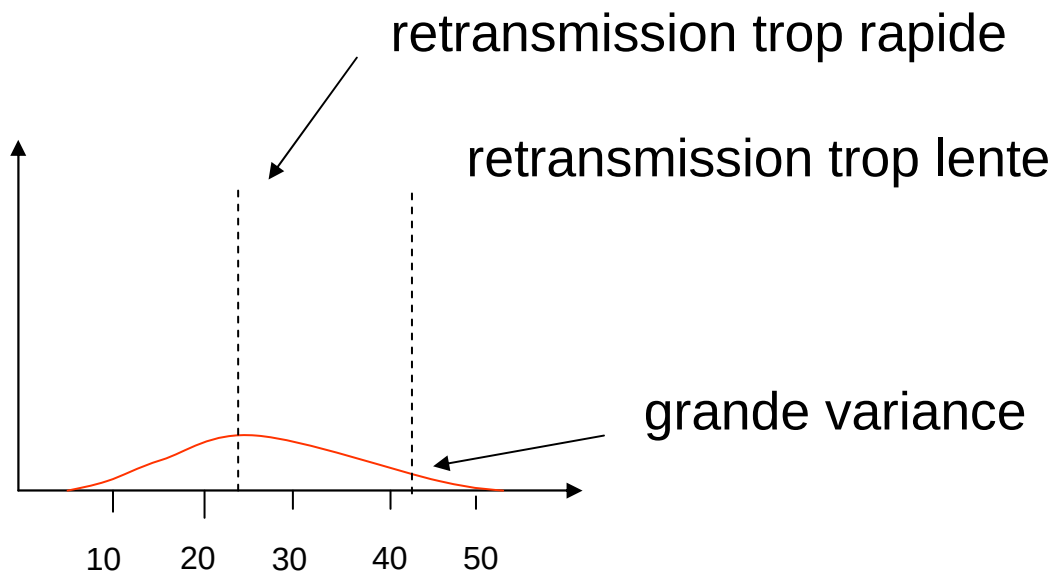
$$M_i = RTT_{reel} + \xi_i, \quad \text{avec } \xi_i \text{ i.i.d et } E(\xi_i) = 0$$

On a:

$$\begin{aligned} E(RTT_{i+1}) &= \alpha E(RTT_i) + (1-\alpha) RTT_{reel} \\ &= \alpha^2 E(RTT_{i-1}) + \alpha(1-\alpha) RTT_{reel} + (1-\alpha) RTT_{reel} \\ &= \dots \\ &= \underbrace{\alpha^{i+1}}_{\to 0} RTT_0 + \underbrace{\sum_{j=0}^{i} \alpha^j}_{\to \frac{1}{1-\alpha}} (1-\alpha) RTT_{reel} \to RTT_{reel} \end{aligned}$$

RTT - variance

Un problème fondamental est que le temps RTT varie considérablement. la distribution de ce temps est du type:



On doit donc estimer aussi la variance du temps RTT, plus la variance est petite moins RTT varie avec le temps et vice-versa.

RTT - variance

Pour estimer la valeur du temporisateur on doit tenir compte de la variance D du temps de boucle RTT.

Un estimateur simple et récursif de la variance est basé sur la différence entre le temps de boucle estimé RTT (estimation courante) et le temps de boucle mesuré M, on utilise

$$\| RTT - M \|$$

La variance D est mis-à-jour suivant la formule

$$D = \alpha D + (1 - \alpha) \| RTT - M \|$$

Le coefficient de pondération α peut-être différent que pour la mise à jour de RTT

Valeur du temporisateur

La valeur du temporisateur est déterminée par:

$$temporisateur = RTT + 4D$$

Le choix du facteur 4 est motivé par:

1. facile à calculer avec des décalages
2. moins de 1% des segments arrive avec un délai supérieur à 4 fois l'écart type

La valeur du facteur égale à 4 est validée expérimentalement

Estimation de RTT et retransmission

Lorsqu'un segment est retransmis et que l'on reçoit un acquittement, on ne peux savoir si le récepteur acquitte le premier segment transmis ou le second. Ce problème fausse l'estimation de RTT puisqu'on ne peux pas calculer M.

Une stratégie couramment implémentée dans TCP est la suivante:

Lorsqu'un segment est retransmis on ne met pas à jour RTT

On double le temps de temporisation jusqu'à ce que le segment soit correctement acquitté.

On parle d'algorithme de Karn.

Persistance

Un autre temporisateur utilisé par TCP est le **temporisateur de persistance**. Ce temporisateur permet de résoudre le problème suivant:

Le récepteur peut envoyer un segment d'acquittement avec une fenêtre d'émission nulle qui à pour effet de demander à l'émetteur d'attendre.

Le récepteur envoie un autre segment avec une fenêtre d'émission non nulle pour indiquer à l'émetteur de recommencer à transmettre.

Si ce segment est perdu, l'émetteur et le récepteur attendent l'un sur l'autre, on a un **inter-blocage**.

Pour éviter que cette situation se produise, l'émetteur utilise un temporisateur de persistance. Après expiration, il transmet un segment et reçoit en réponse un acquittement avec la taille correcte de la fenêtre (qui peut être nulle).

TCP – couche de transport

Pour désigner le mode de transmission et acquittement de TCP, on parle **d'acquittement positif avec retransmission** (PAR).

Il existe d'autres modes de fonctionnement, par exemple certains protocoles peuvent désigner les séquences incorrectes ce qui permet de réduire les retransmissions.

Fenêtre

Dans le segment TCP, le champ fenêtre occupe 16 bits, se qui limite à 65’535 octets transmis sans acquittement.

Cette limitation est problématique dans les réseaux à large échelle (de type WAN) où le temps de latence est grand et le débit très élevé. En effet, le temps d’attente de l’acquittement devient trop important par rapport au temps nécessaire à la transmission des 65535 octets.

Le champ ***option*** du segment TCP est utilisé pour augmenter la taille possible de la fenêtre TCP. En effet, le récepteur peut écrire une valeur sur 14 bits dans ce champ qui correspond au nombre de décalage que doit subir le champ *fenêtre.* On peut donc prescrire une fenêtre sur 30 bits (1 gigaoctet).

RFC

Les nouveaux standards sont définis par une association appelée *Internet Engineering Task Force (IETF).*

Cette association est composée de personnes qui proposent des solutions à des problèmes ou des évolutions du réseaux Internet.

Ces personnes écrivent des rapports qui s'appelle RFC (Request For Comments) qui sont disponibles à **http://www.ietf.org/rfc.html.**

Par exemple, la RFC 1323 décrit comment déterminer les temps d'attente ainsi que l'utilisation du champ option des segments TCP pour augmenter la taille de la fenêtre TCP (entre autres).

Il y a deux type de RFC, les FYI (For Your Information) qui sont moins techniques et proposent des descriptions générales

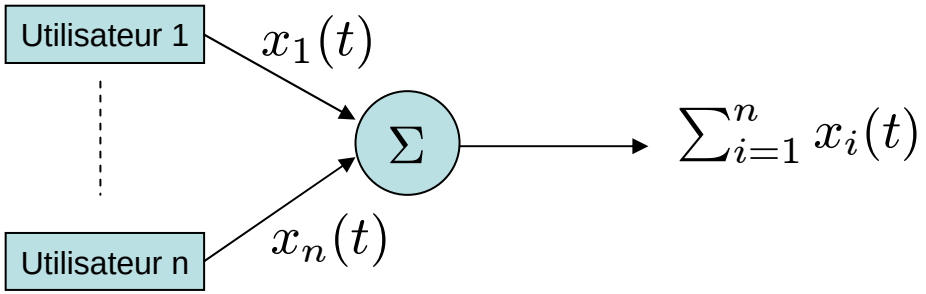
Les STD qui référencent les RFC qui sont devenus des standards (voir la RFC 2026, The Internet Standard Process)

Analyse des strategies d'évitement de congestion

Dans cette partie on cherche a formuler plus généralement et plus précisément les stratégie d'évitement (congestion avoidance) et contrôle de congestion.

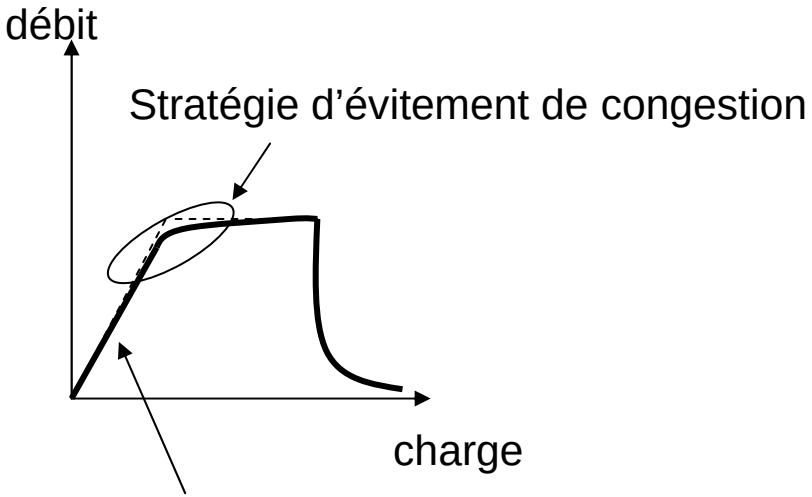
En considérant une sous-classe (linéaire) de contrôle admissible, on pourra déterminer les stratégies optimales. Le mécanisme implémenté dans TCP par la fenêtre de congestion en fait partie.

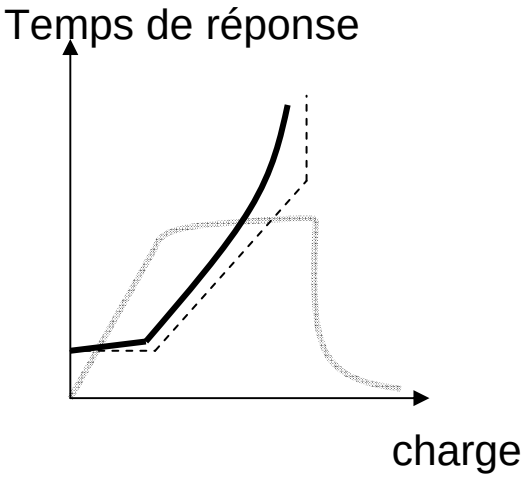
On considère la situation ou n utilisateurs se partagent un ressource.



Congestion

Lorsque la ressource est un réseau informatique, on observe les courbes de réponses suivantes





Stratégie de contrôle de congestion

Feedback

La ressource peut être soit sur-utilisée soit sous-utilisée. On suppose que les utilisateurs disposent d'un signal de feedback leur permettant d'évaluer si la charge à laquelle est soumise la ressource peut-être augmentée ou doit être diminuée. Formellement, on suppose que les utilisateurs ont accès à un signal définit comme:

$$y(t) = \begin{cases} 0 & \text{augmentation de la charge} \\ 1 & \text{diminution de la charge} \end{cases}$$

Dans notre exemple $y(t)$ est un signal binaire qui indique si

$$\sum x_i(t) > X_{\text{target}}$$

Contrôle

Selon la valeur du signal de feedback les utilisateurs adaptent la charge qu'ils soumettent à la ressource, c'est-à-dire

$$x_i(t+1) = x_i(t) + u_i(t)$$

On étudiera seulement les fonctions $u_i(t)$ qui sont additives ou multiplicatives. Remarquez qu'en toute généralité, on devrait écrire

$$u_i(t) = f(x_i(t), y(t))$$

Pour indiquer que le contrôle dépend du signal de feedback.

Contrôles distribués

On peut même considérer une situation plus générale, dans laquelle on aurait

$$u_i(t) = f(x_1(t), x_2(t), \ldots, x_n(t), y(t))$$

En pratique on ne s'intéresse pas à cette situation. En effet, l'implémentation d'un tel contrôle nécessite de connaître l'existence d'autres utilisateurs ainsi que l'utilisation qu'ils font de la ressource. On s'intéresse à des contrôles distribués qui s'adaptent à toutes les situations.

Contrôles additifs/multiplicatifs

Les contrôles additifs seront de la forme

$$x_i(t+1) = \begin{cases} a_I + x_i(t) & \text{si } y(t) = 0 \\ a_D + x_i(t) & \text{si } y(t) = 1 \end{cases}$$

A priori, pour un tel type de contrôle on s'attend à $a_I > 0, a_D < 0$

Les contrôles multiplicatifs seront de la forme

$$x_i(t+1) = \begin{cases} b_I x_i(t) & \text{si } y(t) = 0 \\ b_D x_i(t) & \text{si } y(t) = 1 \end{cases}$$

Avec $b_I > 1, b_D < 1$

Contrôles mixtes

En toute généralité on va considérer tous les contrôles mixtes possibles

$$x_i(t) = \begin{cases} a_I + b_I x_i(t) & \text{si } y(t) = 0 \\ a_D + b_D x_i(t) & \text{si } y(t) = 1 \end{cases}$$

Critère de sélection des contrôles

Pour comparer les contrôles possibles, il faut définir les critères qu'on retiendra pour les comparer. On considérera

1. **Efficacité**: Le mécanisme de contrôle doit assurer qu'on utilise au maximum la ressource considérée. Cela veut dire que la charge appliquée à la ressource doit être proche de X_{target}

2. **Equité**: (fairness) La ressource doit être utilisée de manière équitable par les différents utilisateurs. En pratique on cherche un mécanisme qui nous assure $x_i(t) = x_j(t)$.

3. **Implémentation distribuée**: Le mécanisme dispose d'un minimum d'information, la valeur $x_i(t)$ et le signal de feedback $y(t)$.

Index d'équité

Pour mesurer si un mécanisme de contrôle est équitable ou pas, on utilise l'index d'équité suivant

$$F(x) = \frac{\left(\sum x_i\right)^2}{n \sum x_i^2} \qquad x = (x_1, \ldots, x_n)$$

Cette fonction satisfait

1. $0 \leq F(x) \leq 1$, la fonction vaut 1 si $x_i = x_j, \ \forall i, j$
2. La valeur de la fonction est invariante par changement d'échelle
3. C'est une fonction continue
4. Si seulement k parmi n utilisateurs se partagent la ressource équitablement, la fonction vaut k/n

TCP

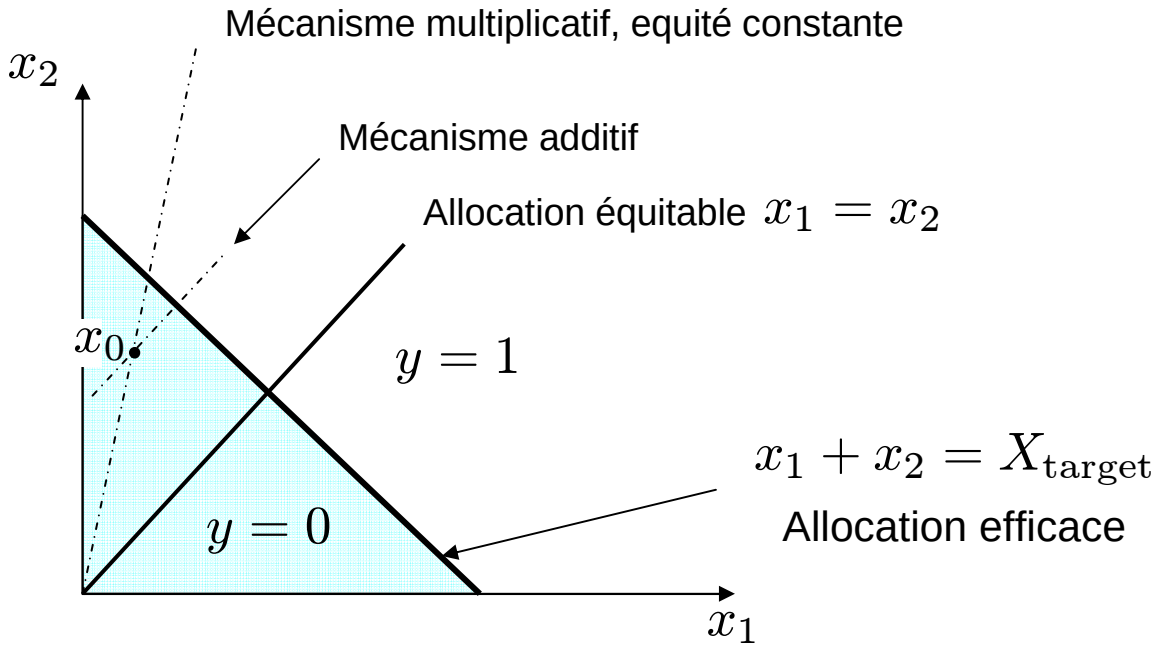
Pour appliquer le formalisme à l'analyse du mécanisme de contrôle de flot implémenté dans TCP, on pose que $x_i(t)$ est la taille de la fenêtre de congestion de l'utilisateur i.

Un mécanisme équitable doit assurer que les tailles des fenêtres de congestion sont les mêmes pour tous les utilisateurs qui partagent la ressource.

Pour faire cette analyse on suppose que les utilisateurs sont saturés, c'est-à-dire qu'ils ont tout le temps des données à transmettre et désirent utiliser les maximum de la bande passante qui leur est attribuée.

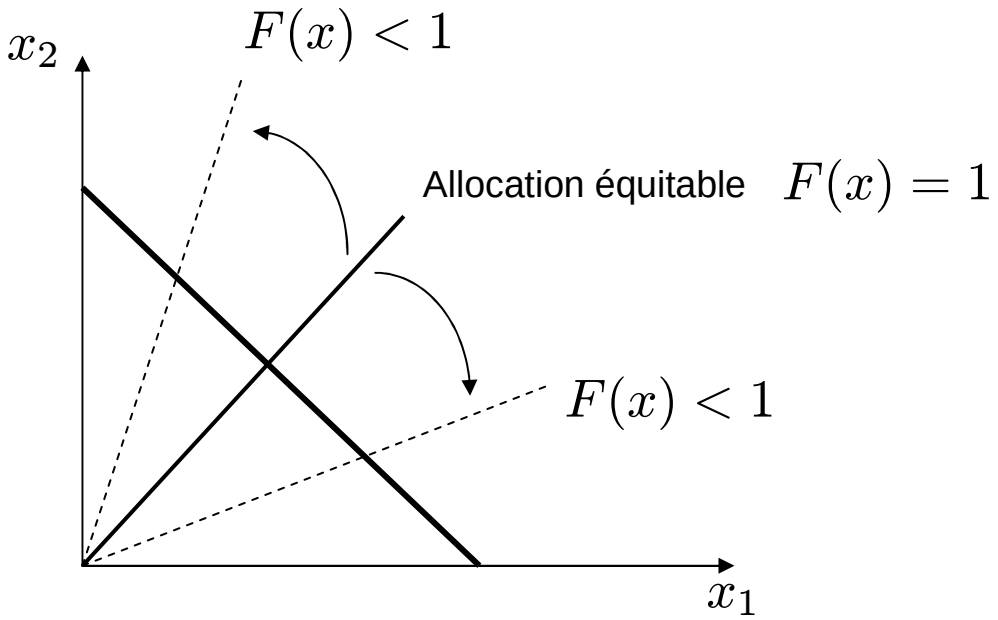
Représentation graphique

Dans l'analyse qui suit, on fait l'hypothèse que les utilisateurs procède à la mise-à-jour de leur variable $x_i(t)$ simultanément. En pratique, cette hypothèse est vérifiée 'à peu près', les modifications s'effectuent à intervalles de temps comparables. Pour deux utilisateurs, on peut représenter les concepts graphiquement $x_0 = (x_1, x_2)$



Mécanisme multiplicatif

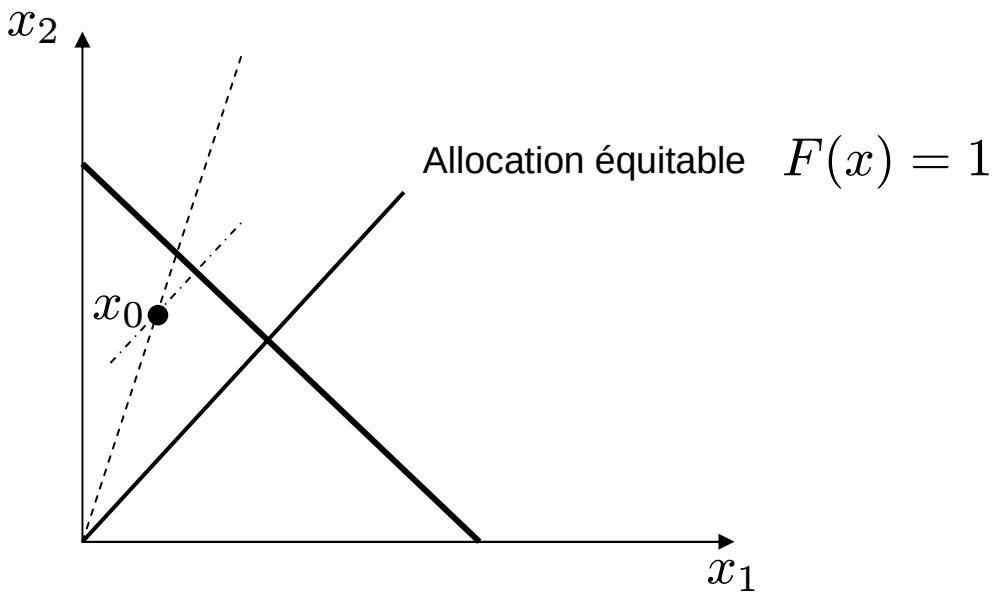
Les droites qui partent de l'origine représentent des allocations équivalentes en terme d'équité



En particulier, un mécanisme multiplicatif pur n'augmente pas l'équité

Mécanisme additif

De même, un mécanisme additif pur augmente l'équité seulement si a>0.

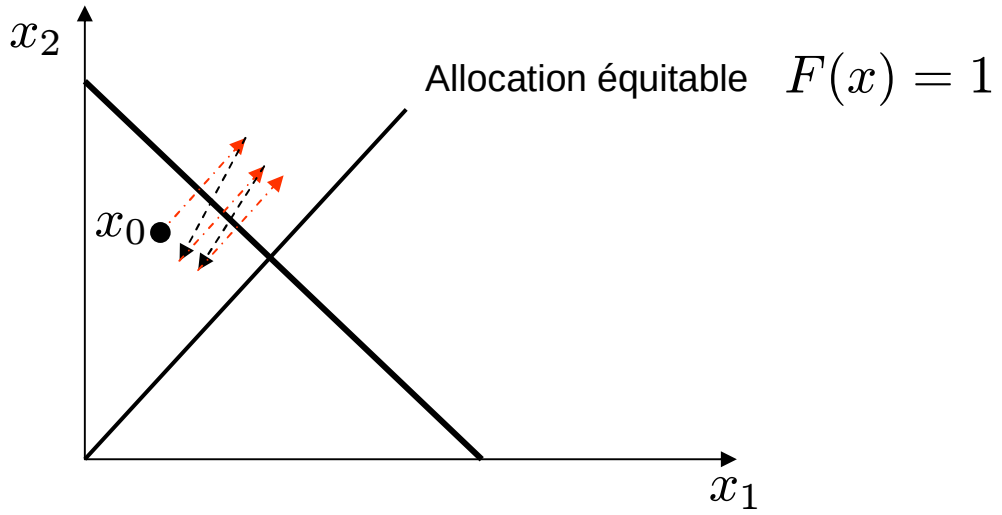


Mécanismes additif/multiplicatif et convergence

On considère le mécanisme mixte suivant $a_I > 0 \leftrightarrow a_I^i > 0$

$$x(t+1) = \begin{cases} x(t) + a_I & \text{si } y = 0 \\ b_D x(t) & \text{si } y = 1 \end{cases}$$

L'équité augmente seulement lorsque la taille des fenêtres augmentent et reste constante sinon. Le mécanisme finalement oscille autour de la valeur optimale

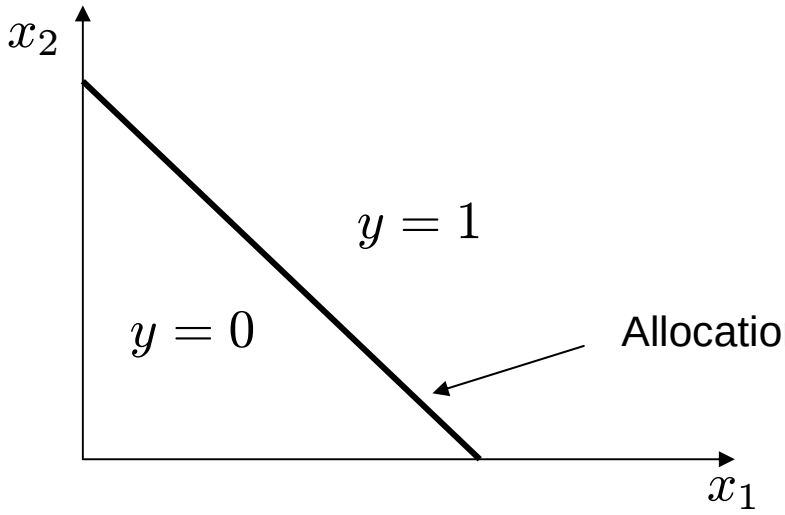


Convergence - efficacité

La convergence vers une allocation efficace de la ressource implique que

Soit $\sum_i x_i(t) \to X_{\text{target}}$

$$\begin{cases} y(t) = 0 \Rightarrow \sum_i x_i(t+1) > \sum_i x_i(t) \\ y(t) = 1 \Rightarrow \sum_i x_i(t+1) < \sum_i x_i(t) \end{cases}$$



Allocation efficace $\sum_i x_i = X_{\text{target}}$

Critère de convergence I - efficacité

On considère un mécanisme (général) mixte.

$$\begin{cases} y(t) = 0 \Rightarrow x_i(t+1) = a_I + b_I x_i(t) \\ y(t) = 1 \Rightarrow x_i(t+1) = a_D + b_D x_i(t) \end{cases}$$

D'où les conditions

$$na_I + (b_I - 1)\sum_i x_i > 0,\ \forall n, \sum x_i < X_{\text{target}}$$

$$na_D + (b_D - 1)\sum_i x_i < 0,\ \forall n, \sum x_i > X_{\text{target}}$$

Ou encore

$$b_I > 1 - \frac{na_I}{\sum x_i}, \quad b_D < 1 - \frac{na_D}{\sum x_i}$$

Critère de convergence II - efficacité

La condition

$$\begin{cases} y(t) = 0 \Rightarrow \sum_i x_i(t+1) > \sum_i x_i(t) \\ y(t) = 1 \Rightarrow \sum_i x_i(t+1) < \sum_i x_i(t) \end{cases}$$

Est difficile à assurer de manière distribuée. En effet, les conditions que l'on obtient sur les coefficients du mécanisme a_I, b_I, a_D, b_D dépendent du nombre d'utilisateurs et de la valeur $\sum_i x_i$.

On obtient des conditions plus fortes et implémentables de manières distribuées en demandant

$$\begin{cases} y(t) = 0 \Rightarrow x_i(t+1) > x_i(t), \ \forall i \\ y(t) = 1 \Rightarrow x_i(t+1) < x_i(t), \ \forall i \end{cases}$$

Critère de convergence II - efficacité

Dans ce cas, on obtient les conditions

$$a_I + (b_I - 1)x_i > 0$$
$$a_D + (b_D - 1)x_i < 0$$

D'où

$$a_I > 0,\ b_I \geq 1,\ a_D = 0,\ 0 \leq b_D < 1$$

Formellement, on obtient $a_D \leq 0$ mais on doit avoir $a_D \geq 0$ pour assurer qu'on améliore l'équité (voir les représentations graphiques et ce qui suit, sinon x_i peut devenir négatif). De plus le signe de a_D et b_D doivent être les mêmes pour assurer la convergence vers l'équité.

Critère de convergence - équité

Pour que le mécanisme de contrôle converge vers une solution équitable, il faut que

$$F\big(x(t)\big) \to 1$$

On a

$$F\big(x(t+1)\big) = \frac{\big(\sum x_i(t+1)\big)^2}{n\sum\big(x_i(t+1)\big)^2} = \frac{\big(\sum a + bx_i(t)\big)^2}{n\sum\big(a+bx_i(t)\big)^2}$$

$$= \frac{\big(\sum c + x_i(t)\big)^2}{n\sum\big(c+x_i(t)\big)^2}$$

$$\text{avec } c = a/b$$

Calculs …

On cherche les valeurs de c tels que $F\big(x(t+1)\big) > F\big(x(t)\big)$

$$\frac{\left(\sum c + x_i\right)^2}{n\sum(c+x_i)^2} = \frac{n^2c^2 + 2nc\sum x_i + (\sum x_i)^2}{n\sum x_i^2}\,\frac{\sum x_i^2}{\sum(c+x_i)^2}$$

$$= \Big[F(x) + \frac{nc^2 + 2c\sum x_i}{\sum x_i^2}\Big]\Big[\underbrace{\frac{\sum x_i^2}{\sum(c+x_i)^2} - 1}_{-\frac{nc^2+2c\sum x_i}{\sum(c+x_i)^2}} + 1\Big]$$

$$= F(x) + \frac{nc^2 + 2c\sum x_i}{\sum(c+x_i)^2}\Big[-F(x) - \frac{nc^2 + 2c\sum x_i}{\sum x_i^2} + \frac{\sum(c+x_i)^2}{\sum x_i^2}\Big]$$

$$= F(x) + \Big[1 - \frac{\sum x_i^2}{\sum(c+x_i)^2}\Big] \cdot \big[-F(x) + 1\big]$$

Résumé

On obtient

$$F\big(x(t+1)\big) = \frac{\left(\sum x_i(t+1)\right)^2}{n\sum x_i(t+1)^2} = F(x) + \left[1 - \frac{\sum x_i^2}{\sum (c+x_i)^2}\right] \cdot \underbrace{\left[-F(x)+1\right]}_{\geq 0}$$

Et donc si c>0 on a convergence vers l'équité. Comme c=a/b on voit que a et b sont de même signe.

Remarquez que si c=0 l'index d'équité n'est pas modifié. En fait on doit avoir

$$\frac{a_I}{b_I} \geq 0 \text{ et } \frac{a_D}{b_D} > 0$$

ou

$$\frac{a_I}{b_I} > 0 \text{ et } \frac{a_D}{b_D} \geq 0$$

Un résultat

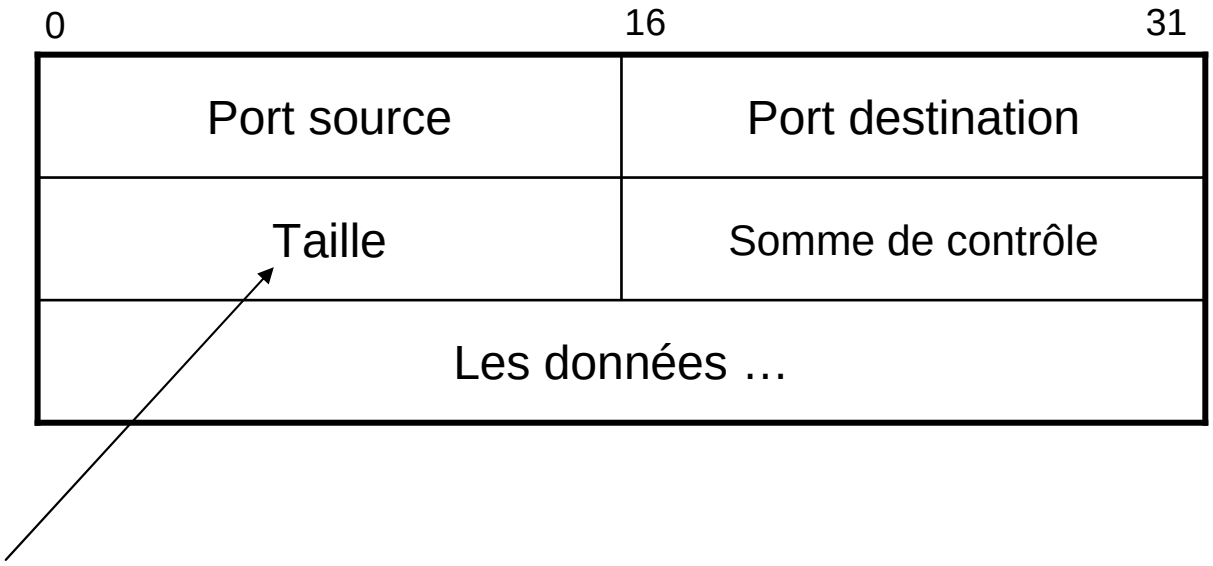
On a démontré

Pour assurer que le mécanisme de contrôle de congestion convergence vers une solution équitable et efficace, il faut:

- Les phases de décroissance doivent être multiplicative (pures)
- Les phases de croissances sont toujours additives avec un coefficients positif et il peut a avoir un facteur multiplicatif pas plus petit que 1.

UDP – couche transport

Protocole UDP: format d'un segment UDP



UDP – couche transport

Utilité: chaque fois qu'on désire transmettre des données sans avoir à établir de connexion.

1. Application requête-réponse (mot de passe)
2. Audio
3. Applications qui mettent en œuvre leurs propres mécanismes de fiabilisation de l'acheminement des données.

UDP et TCP sans fil

En théorie, dans les piles de protocole décrite pour TCP/IP et le modèle ISO chaque couche de la pile est indépendante des couches supérieures et inférieures.

En pratique, les paramètres des protocoles doivent être adaptés aux couches voisines.

Par exemple, le protocole de contrôle de congestion utilise des temporisateurs (paramètres) et une fenêtre de congestion dont la taille est un paramètre.

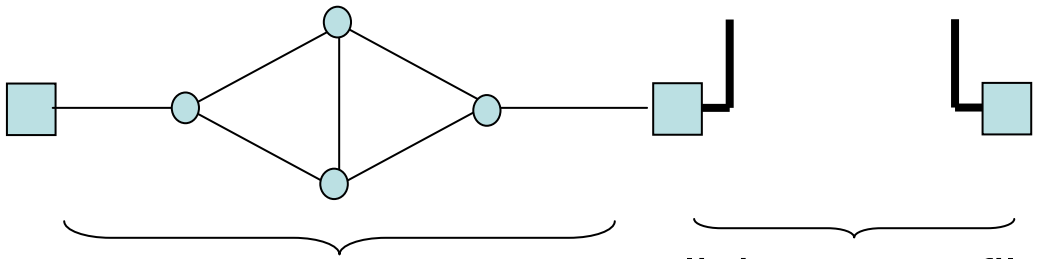
Ce protocole est conçu pour éviter les congestions dans le réseau et lorsqu'un segment n'est pas acquitté à temps, il diminue la vitesse de transmission.

UDP et TCP sans fil

Si on utilise des transmissions sans fil, les segments non acquittées sont en **général perdus** (près de 50% des segments dans certains cas), et ralentir la transmission des données n'améliore pas les performances du réseaux.

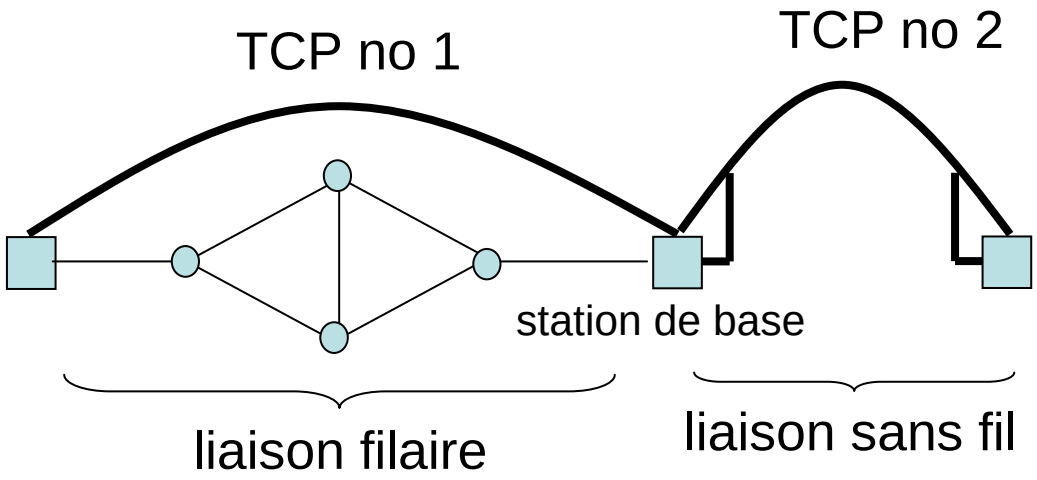
Dans cette situation, il est préférable de retransmettre les segments **aussi vite que possible.**

Le protocole doit donc connaître la nature du média de communication, certaines fois elle n'est pas homogène



UDP et TCP sans fil

Dans cette situation il est recommandé de scinder la connexion en deux connexions homogènes, on parle de **TCP indirect**.



On perd la sémantique de TCP, puisqu'avec cette solution les segments acquittées ne sont pas reçus par le destinataire mais par la station de base

UDP et TCP sans fil

Une autre solution consiste à implémenter un agent qui contrôle les segments transmis par la couche TCP et qui utilise un temporisateur de retransmission relativement court.

Si un segment est non acquitté. cet agent prend en charge la retransmission du segment avant la couche TCP. Ainsi, les segments sont retransmis rapidement sans modifier TCP.

De plus, si des segments sont perdus, la station de base demande la réémission sélective des segments perdus.

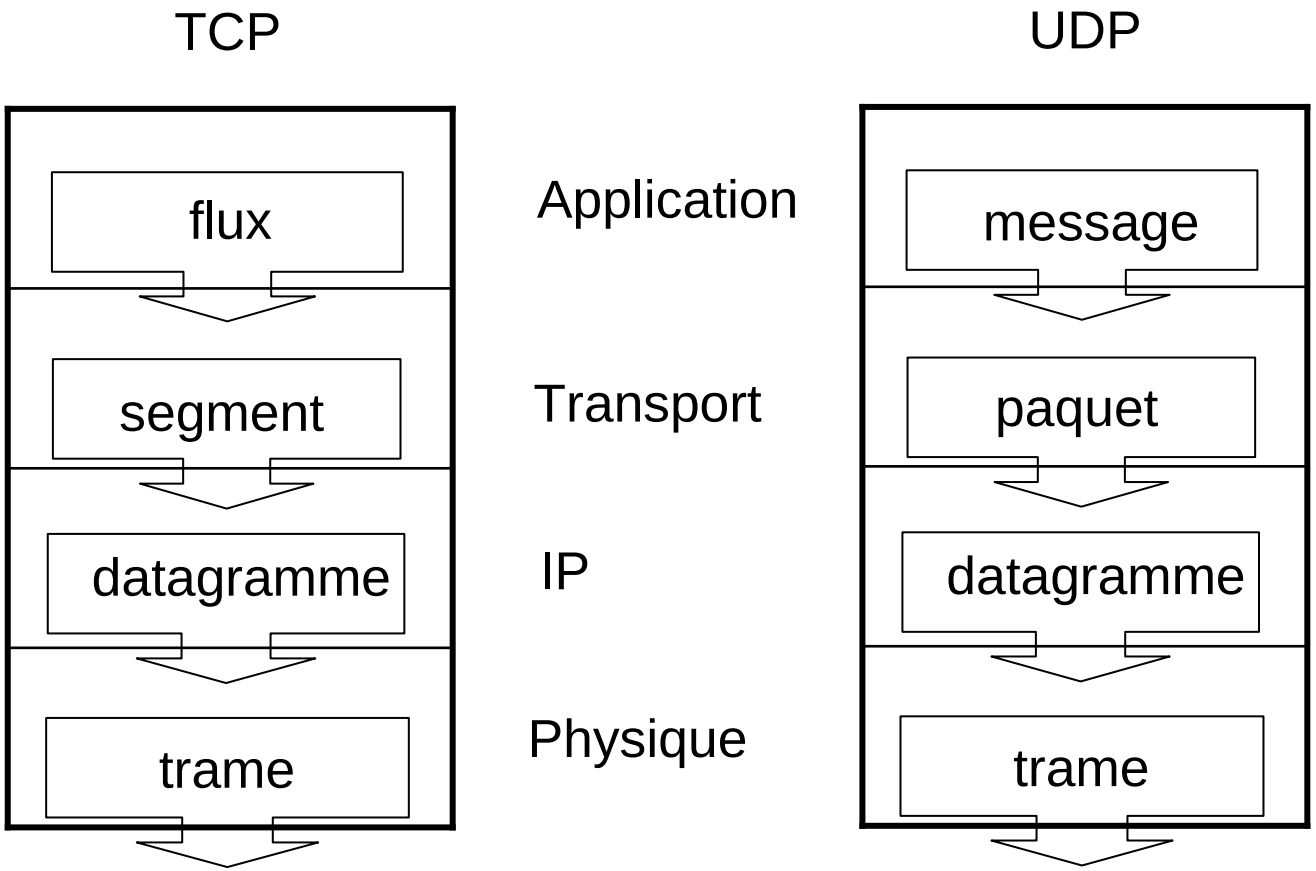
UDP et TCP sans fil

En général, on utilise le protocole UDP pour des connexions qui sont fiables.

Dans cette situation, l’impact des liaisons sans fil peut être désastreux sur les performances du système.

Modèle TCP/IP

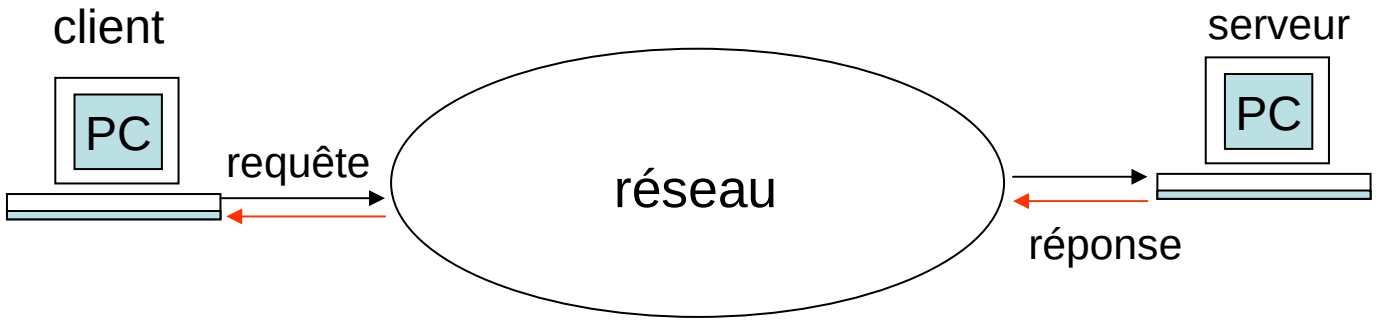
Vocabulaire:



Le modèle client-serveur

La communication entre deux ordinateurs s'effectue souvent selon le modèle client-serveur.

- Le client envoie une requête au serveur
- Le serveur effectue une certaine tâche conformément à la requête du client.



applications

- Gestion de bases de données
- Systèmes transactionnels
- Messagerie électronique
- Partage de données (serveurs de fichiers)
- Calcul scientifique
- …

Exemples

Serveur de fichier:

permet de stocker des fichiers et de partager leurs accès par le réseau. Exemple, serveur nfs

les demandes du client peuvent être *lire* ou *écrire* le fichier.

Serveur de messagerie électronique:

le serveur stocke les messages reçus et permet leurs lectures à la demande du client. Le serveur permet aussi l'envoie de messages électronique. Exemples, serveurs SMTP, POP, IMAP

Exemples

Serveur d’impression:

permet à tous les utilisateurs d’un réseau de partager une imprimante. Exemple, serveur SAMBA.

Serveur web:

processus logiciel qui peut-être accéder par un client via le réseau Internet. Le navigateur web permet de visualiser les informations transmises par le serveur web au client. Le navigateur est le programme client.

Particularités du modèle client-serveur

- Communication d'égal à égal entre les application
- Dialogue entre deux processus pairs
- Répartition des services plutôt que des applications
- Processus coopératifs
- Un service réparti est typiquement un service qui nécessite beaucoup de ressources machines (CPU, mémoire, etc.)

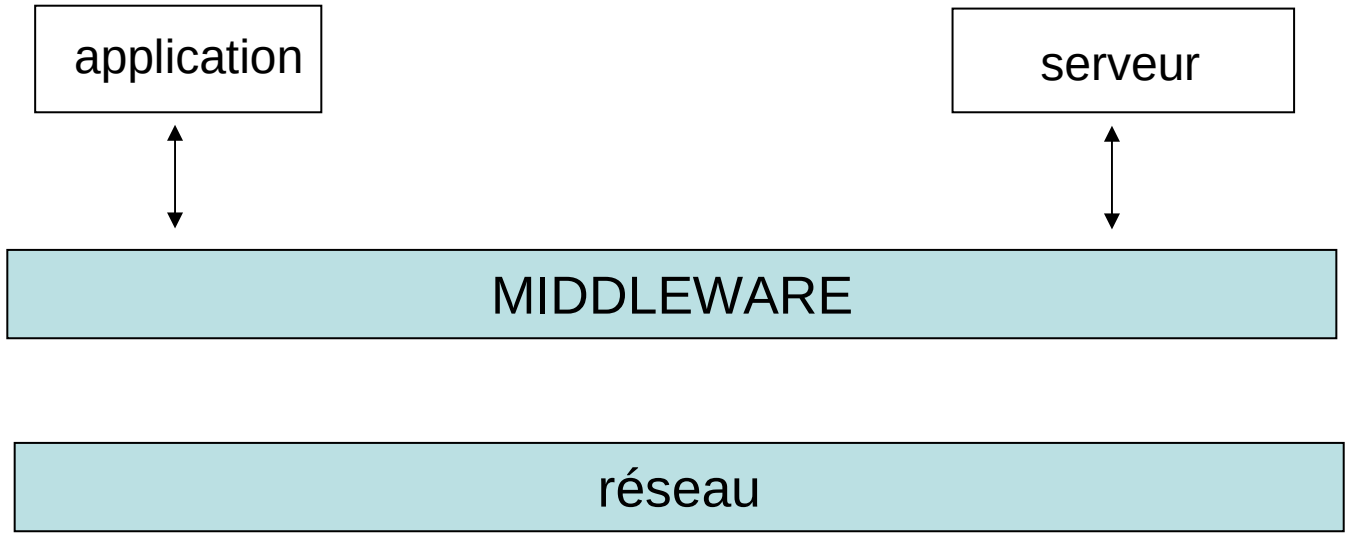
Tous les membres du réseau peuvent:

- Être client ou serveur
- Demander un service
- Réaliser un service

contre

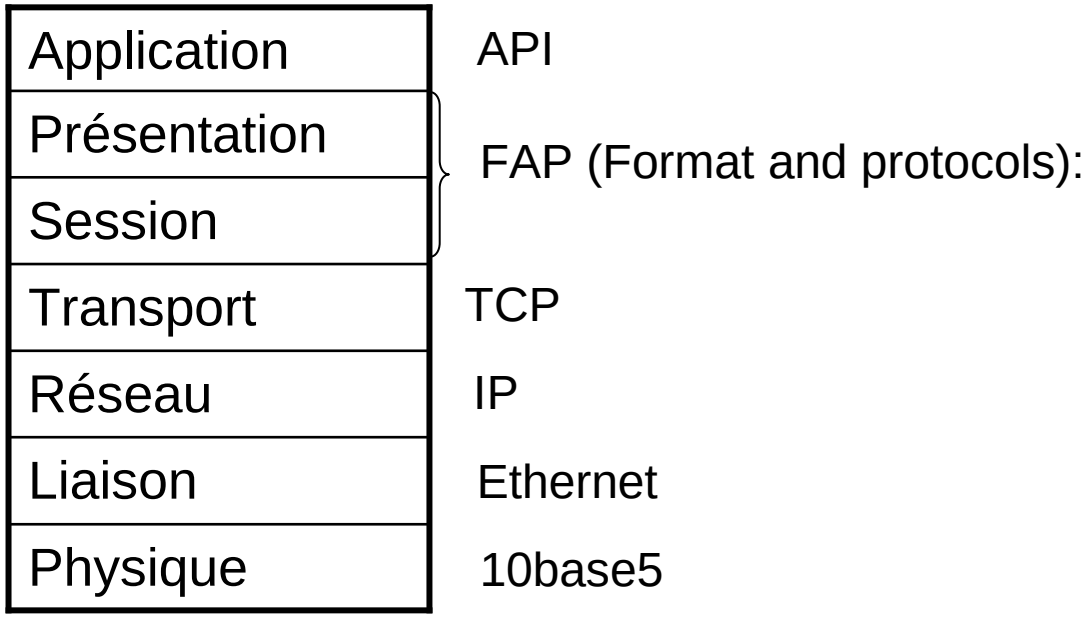
- Difficile de développer des applications distribuées
- Manque de cohérences entre les clients/serveurs
- Pas d’administration centralisées des serveurs

Middleware



L’objectif du middleware est d’offrir aux applications une interface unifiée permettant l’accès aux services disponibles sur un réseau via des API (Application Programming Interface, par exemple les Sockets)

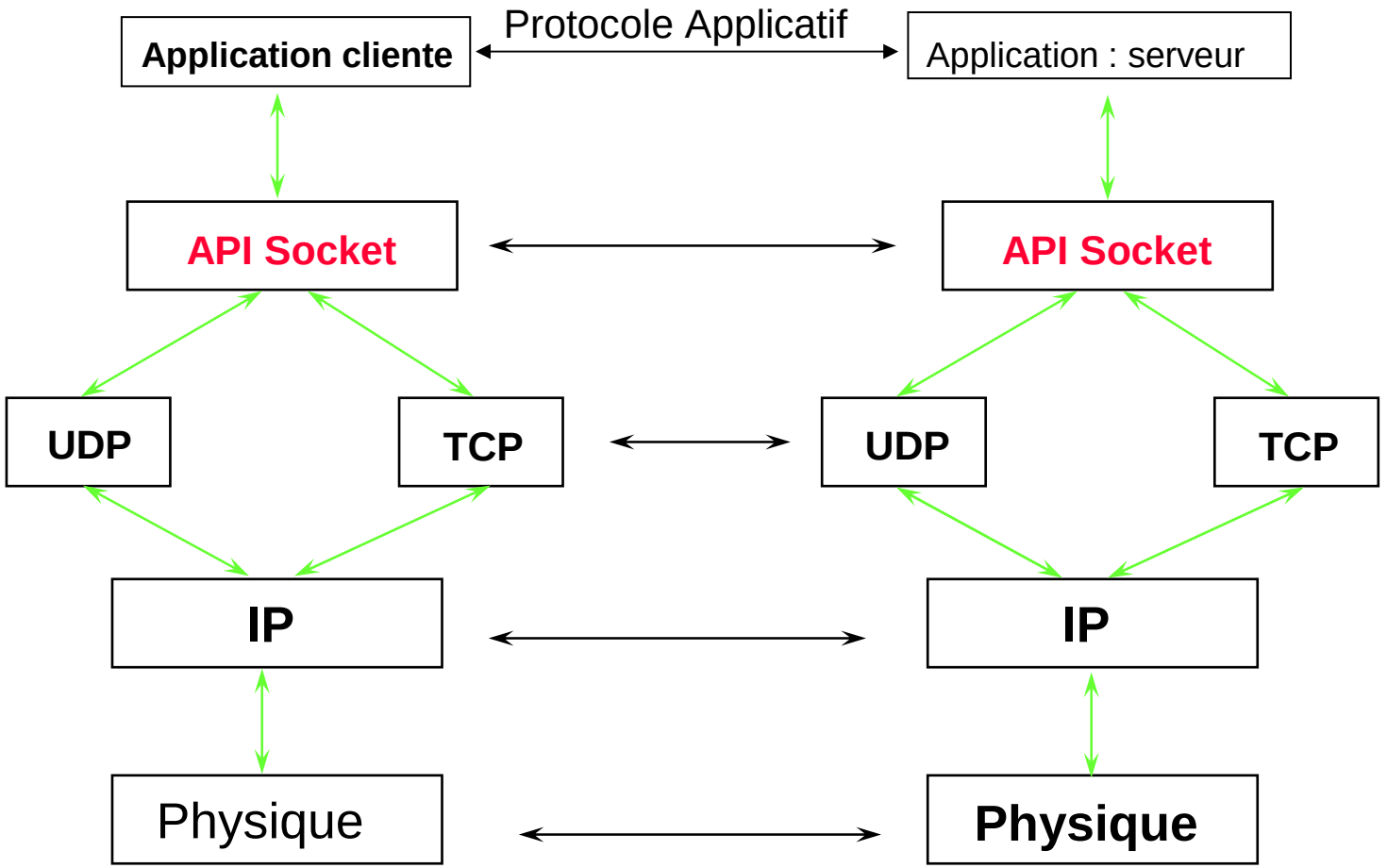
Middleware et modèle ISO



FAP: synchronisation du dialogue, défini le format des données échangées, lien avec la couche transport

API Sockets

- Les sockets : interface client/serveur utilisée à l'origine dans le monde UNIX et TCP/IP.
- fournit les primitives pour le support des communications reposant sur toute suite de protocoles; les protocoles TCP/IP sont à l'origine des développements.
- Les applications cliente et serveur ne voient les couches de communication qu'à travers l'API socket (abstraction):


Application cliente
Protocole Applicatif
Application : serveur
API Socket
API Socket
UDP
TCP
UDP
TCP
IP
IP
Physique
Physique

Conception d'application client-serveur

Une application informatique peut-être divisée selon trois couches

1. Interface Homme/Machine et gestion de l'affichage.
2. Couche de traitement (fonctionnalités de l'application), algorithmique,
3. Couche de données, stockage/récupération des données.

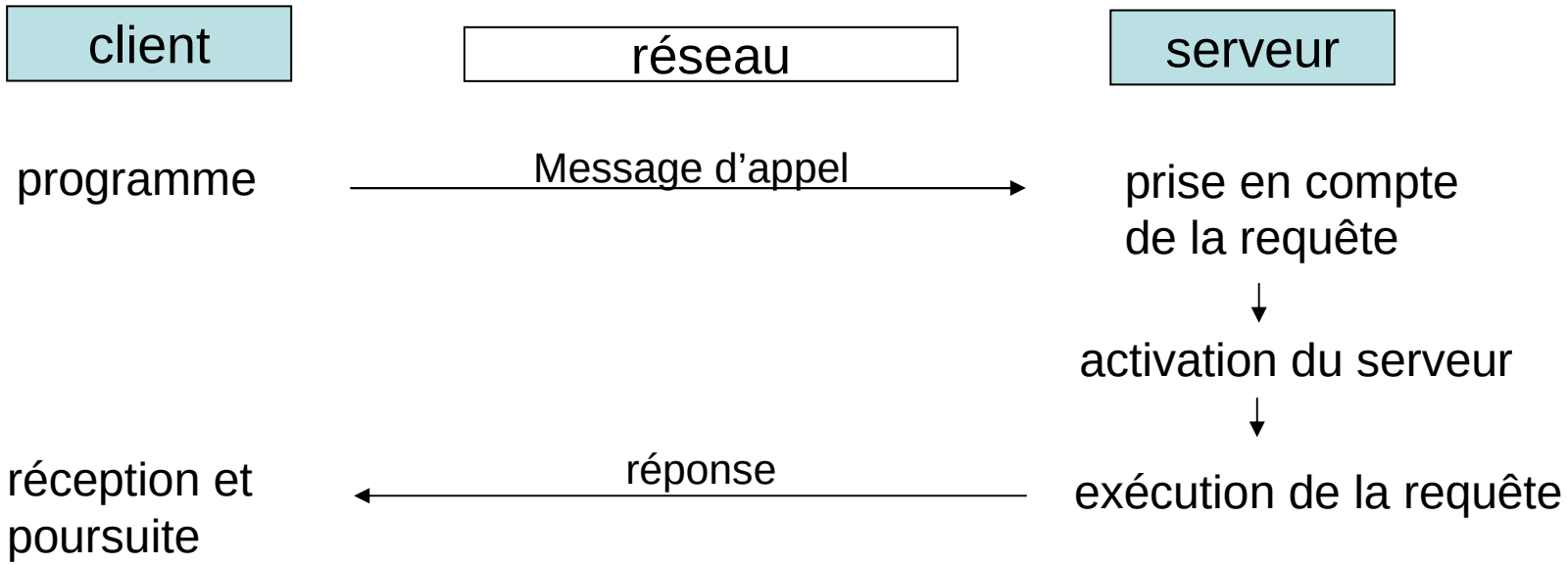
Chacune de ces couches peut-être implantée sur des serveurs distincts

Conception
modes de communication

Mode non-connecté (peu de traitement par requête):

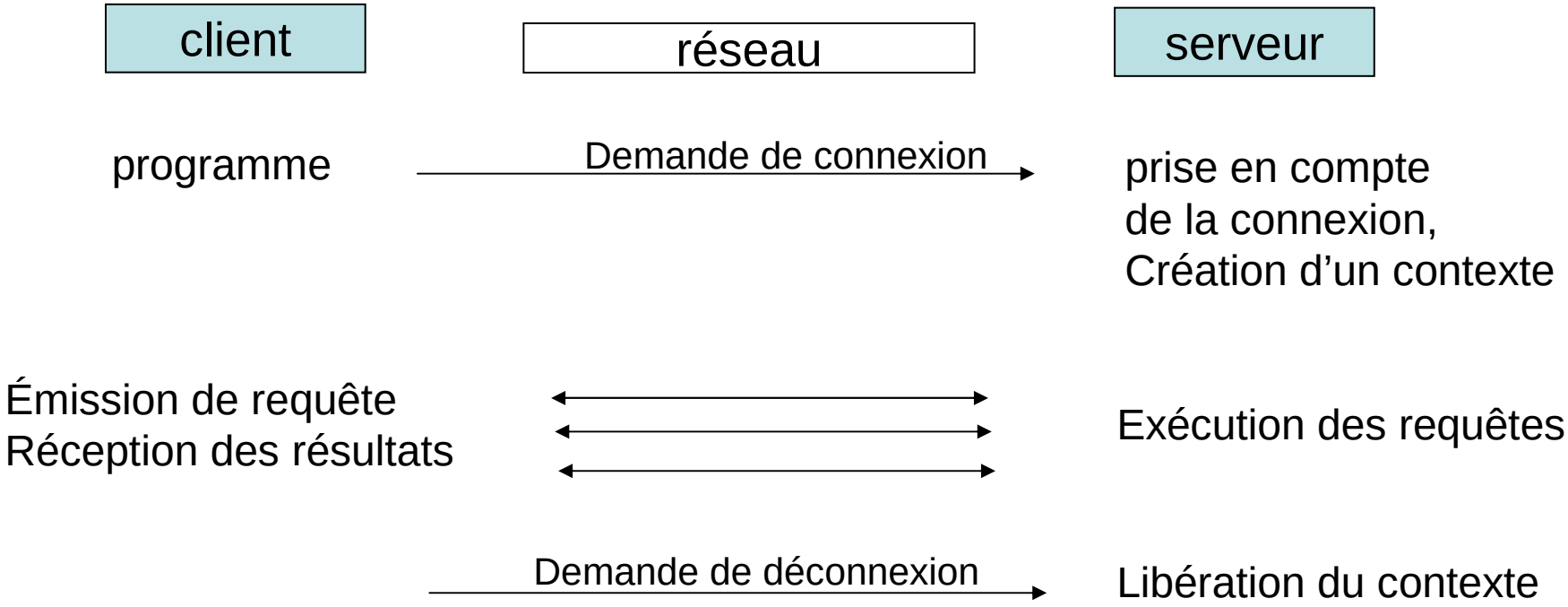
l'application doit gérer l'arrivée des données

généralement un connexion synchrone



Modes de communication

Mode connecté:



Meilleure sécurité pendant le transfert des données, contrôle de flux, négociation de la qualité du service (p.ex. taux de compression)

Types de serveurs

serveurs itératifs: acceptent un seul client à la fois

les serveurs itératifs en mode non-connecté:

- offrent une interface de communication sur le réseau en mode non-connecté,
- indéfiniment : réceptionnent une requête client, formulent une réponse, et renvoient le message réponse vers le client selon le protocole applicatif défini.

les serveurs itératifs en mode connecté:

- offrent une connexion sur le réseau en mode connecté,
- (*) réceptionnent une connexion client,
- offrent une nouvelle connexion sur le réseau,
- répétitivement : réceptionnent une requête pour cette connexion, formulent une réponse, et renvoient le message réponse vers le client,
- lorsque le traitement pour ce client est terminé -->(*).

Types de serveurs

Serveurs parallèles: acceptent plusieurs clients simultanément

les serveurs parallèles en mode non-connecté:

- offrent une interface de communication en mode non-connecté
- répétitivement : réceptionne la requête client; crée un processus secondaire (PR. S.) chargé de traiter la requête courante.
- (PR. S.) : formule une réponse à la requête client, et renvoie le message
- (PR. S.) : lorsque le traitement est terminé, libère la communication, Exit.

Types de serveurs

les serveurs concurrents en mode connecté:

• offrent une connexion sur le réseau en mode connecté

• répétitivement : réceptionne une connexion client, offre une nouvelle connexion sur le réseau, crée un PR. S. chargé de traiter la connexion courante.

• (PR. S.) : répétitivement : réceptionne une requête pour cette connexion, formule une réponse, et renvoi le message réponse vers le client selon le protocole applicatif défini,

• (PR. S.) : lorsque le traitement est terminé (propre au protocole applicatif), libère la connexion, Exit.

Remote procedure Call (RPC)

Les RPC permettent le développement d'applications selon le schéma:

1. construction d'une application conventionnelle, dans un environnement mono-machine,

2. subdivision de l'application en plusieurs modules qui pourront s'exécuter sur différentes machines.

Le modèle RPC permet d'exécuter des procédures situées sur des machines distantes.

RPC

- Les appels de procédures sont les requêtes clients

- Les retours des procédures sont les réponses serveurs

- Un appel de procédure s'effectue de manière synchrone, l'instruction située après l'appel est exécutée après la procédure.

BUT: conserver la sémantique associée aux appels de procédure dans les environnements monoprocesseur.

RPC - particularités

- Les temps d'exécution d'une procédure distante peut-être plus long à cause des délais introduits par le réseau
- Les RPC n'admettent pas d'arguments de type pointeur
- Les descripteur d'entrées/sorties sont inconnus, prévient l'utilisation des périphériques locaux (p.ex. messages d'erreurs)