Mécanismes de contrôle d'erreurs et de congestion de TCP en filaire

I. Contrôle d'erreurs

Chaque segment émis par la couche TCP est conservé en mémoire jusqu'à ce que l'acquittement correspondant soit reçu par l'émetteur. Pour détecter la perte de segment (de données ou d'acquittement), un temporisateur est armé au moment de la transmission du segment. Si le temporisateur se déclenche, l'émetteur retransmet le segment concerné.

Les segments TCP sont numérotés en séquence à l'émission et il en est de même pour les acquittements. Ainsi, si l'émetteur a transmis plusieurs segments avant de recevoir un acquittement, à partir du numéro contenu dans le dernier acquittement reçu, il sait quel segment est concerné.

***Remarque** : les numéros de séquence (placés par l'émetteur) et les numéros d'acquittement (placés par le récepteur) donnent une position en octets par rapport au début du flux de l'émetteur. Par exemple, un acquittement contenant le nombre 1021 signifie que le récepteur est prêt à recevoir le segment commençant à partir de l'octet 1021. Il ne faut donc pas confondre cette numération avec celle utilisée par la couche liaison de données qui compte en nombre de trames émises ou reçues.*

Comme la couche liaison de données, la couche TCP utilise le principe de la fenêtre coulissante pour émettre une suite de segments avant d'attendre un acquittement. Lorsqu'un acquittement est reçu, la fenêtre d'émission glisse vers la droite d'un nombre d'octets en fonction du numéro d'accusé de réception reçu.

Dans les réseaux filaires actuels (attention ce qui suit n'est absolument pas le cas des réseaux non filaires), le taux d'erreurs est très faible (surtout sur les fibres optiques). On considère alors (à juste titre) que la perte d'un segment est due à une congestion de réseau et non à une erreur de transmission. Les algorithmes de retransmission et de contrôle de congestion dans Internet sont fondés sur cette hypothèse.

Algorithme de retransmission adaptatif

TCP est un protocole destiné à fonctionner au-dessus d'un protocole de réseau éventuellement non fiable (généralement ce protocole c'est IP). Le nombre de réseaux traversés par un segment est généralement inconnu par la source. Par conséquent, le temps de transfert d'un segment peut varier considérablement d'un segment à l'autre. TCP doit donc tenir compte de ces variations pour déterminer les valeurs de temporisateurs liés à l'attente des acquittements. Il faut rappeler que cette notion de temporisateur est utilisée aussi par la couche liaison de données, mais la valeur d'armement du temporisateur est fixe, car on peut facilement estimer le délai de traverser d'un réseau local.

Pour prendre en compte la variation des délais d'acheminement de paquets par le réseau sous-jacent, TCP utilise un algorithme de retransmission adaptatif qui consiste à surveiller le comportement du réseau et en déduire les valeurs de temporisation. L'algorithme le plus répandu pour calculer les valeurs de temporisation de retransmission est celui de Jacobson (1988) dont le principe est résumé ci-dessous.

Pour obtenir les informations nécessaires à l'algorithme de retransmission, TCP enregistre l'instant auquel chaque segment de données est émis et l'instant de réception de l'acquittement correspondant. Ces deux instants permettent de déterminer une valeur (un échantillon) de délai d'aller-retour (ou *RTT : Round Trip Time*). Pour estimer avec précision le délai d'aller-retour, plusieurs échantillons sont nécessaires. Ainsi, TCP calcule et conserve les deux dernières valeurs (la dernière valeur notée *Nouveau_RTT* et celle qui la précède

notée *RTTestimé*) de délai d'aller-retour pour les différents segments transmis. Le temps de RTT moyen est calculé à tout instant par la formule suivante :

$$RTTestimé = (\alpha * RTTestimé) + (1- \alpha) * Nouveau\_RTT)$$

Le coefficient α ($0 \leq \alpha < 1$) est choisi de manière protéger l'ancienne valeur moyenne contre une influence trop forte de la nouvelle, c'est-à-dire ne pas réagir brutalement aux pics de trafic. Le choix d'une valeur proche de 1 rend la moyenne insensible aux variations brutales de trafic.

Lorsqu'un segment est émis, TCP calcule une valeur de temporisation *RTO* ("Retransmission TimeOut"), pour armer le temporisateur, à partir du *RTTestimé* de la manière suivante :

$$RTO = Min\{BorneSuppRTO, Max\{BorneInfRTO, RTTestimé * \beta\}\}$$

BorneSuppRTO désigne la plus grande valeur que peut prendre la temporisation (par exemple 1 minute). *BorneInfRTO* c'est la plus petite valeur que peut prendre la temporisation (par exemple 10 ms). Le coefficient β ($\beta > 1$) est utilisé pour mieux tenir compte du délai d'aller-retour. Si β est proche de 1, l'attente de l'acquittement dure l'équivalent du dernier RTT et dans ce cas on détecte rapidement la perte du segment données ou de son acquittement. Si β est nettement plus grand que 1 (par exemple $\beta = 2$), cela permet éventuellement d'être un peu plus 'patient' pour tenir compte d'une éventuelle hausse du délai d'aller-retour, mais avec cette solution on peut perdre du temps avant retransmettre. Le choix d'une valeur optimale de β est difficile ; il dépend de beaucoup de facteurs et a fait et fait encore l'objet de nombreux travaux de recherche et d'expérimentation.

Problème de calcul du temps de boucle (aller-retour)

Le calcul décrit précédemment serait simple si TCP utilise un acquittement par segment de données. Rappelez-vous que TCP utilise une technique d'acquittement cumulative : un acquittement concerne un nombre d'octets bien reçus et non un segment de données particulier. Ainsi, lorsque TCP retransmet un segment de données et puis il reçoit un acquittement, se pose alors la question : est-ce que cet acquittement (éventuellement tardif) correspond au segment de données initial ou bien est-ce qu'il correspond au segment de données retransmis ? TCP n'a aucun moins de distinguer les deux cas. Associer l'acquittement au segment de données le plus ancien peut provoquer une augmentation significative du délai d'aller-retour surtout si plusieurs tentatives de retransmissions ont été effectuées pour le même segment initial. Associer l'accusé de réception au segment de données (retransmis) le plus récent peut aussi être incorrect, car on peut recevoir l'acquittement du segment initial juste après l'avoir retransmis (dans ce cas, on attribue l'acquittement au segment retransmis et non au segment initial) et trouver un délai d'aller-retour très petit. Par conséquent, ni l'association de l'acquittement à la transmission du segment de données initial, ni l'association de l'acquittement à la transmission du dernier segment de données retransmis ne peuvent fournir une bonne estimation du délai d'aller-retour. Par conséquent, TCP ne doit pas mettre à jour la valeur du *RTTestimé* quand il s'agit de segments retransmis ; cette idée est connue sous le nom d'algorithme de Karn. Pour mieux prendre en compte les situations générées par les pertes de segments de données et des acquittements, une des solutions c'est de combiner les temporisations de retransmission avec une stratégie d'augmentation des temporisations. La *stratégie d'augmentation des temporisations* calcule une valeur initiale de temporisation *RTO*, comme vu précédemment, et en cas de retransmission, on augmente la valeur de *RTO*. Les implantations de TCP utilisent différentes manières d'augmenter la valeur de *RTO*. La plupart des implantations choisissent un facteur multiplicatif λ qui vaut généralement 2 :

$$Valeur\ de\ Tempo\ augmentée = RTO * \lambda$$

Il existe d'autres techniques plus récentes, mais plus complexes, car elles exigent plus de paramètres et de calcul, pour changer dynamiquement la valeur de temporisation.

II. Contrôle de congestion

1. Principe général d'utilisation de fenêtre

Dans TCP, les numéros de segments sont codés sur 32 bits. Si aucune précaution n'est prise, un émetteur peut envoyer en rafale jusqu'à 2^{32} octets (il envoie par exemple un fichier très volumineux) avant de se bloquer. Un tel comportement présente plusieurs inconvénients :

1) S'il y a des pertes ou des erreurs, l'émetteur risque d'effectuer un nombre élevé de retransmissions (d'où une perte de temps et une mauvaise utilisation du réseau) ;
2) Le récepteur risque d'être saturé, ce qui le conduit à rejeter des segments (car il n'a plus d'espace mémoire pour les stocker) ;
3) Le réseau traversé risque d'être saturé par le comportement en rafale de l'émetteur.

Pour éviter ces problèmes, le protocole TCP assure une fonction de contrôle de congestion. Ainsi, on "responsabilise" la source de données afin d'éviter la congestion de réseau.

Le récepteur annonce son crédit dans le champ *Fenêtre-crédit* des segments qu'il envoie pour indiquer à son correspondant le nombre d'octets qu'il est prêt à recevoir. On dit *Fenêtre* ou *Crédit* ou encore *Fenêtre de réception*. Quand la *Fenêtre* est nulle, l'émetteur ne doit pas plus envoyer de segments sauf dans deux cas particuliers :

1) Il peut envoyer des données urgentes, pour permettre l'exécution de certaines opérations importantes sur le site destinataire (par exemple, arrêter une application distante).
2) Il peut envoyer un segment contenant un seul octet pour obliger le récepteur à ré-annoncer le prochain octet attendu ainsi que la taille de la fenêtre. Ce mécanisme permet d'éviter les situations d'interblocage en cas de perte d'annonce de fenêtre.

La taille maximum de segments que la source peut émettre avant d'attendre un acquittement est appelé *Fenêtre de congestion* (*cwd* : congestion window). Au lieu d'avoir une valeur fixe pour *cwd* (comme on le fait dans le cas de la couche liaison de données), le protocole TCP gère de manière sophistiquée la taille de la fenêtre de congestion. De manière simplifiée, cette gestion consiste à émettre beaucoup de segments de données quand la source juge que le réseau est sous-chargé et, au contraire, à réduire son rythme de transmission quand elle juge que le réseau est surchargé (voire saturé). La manière dont on ajuste dynamiquement la taille de la fenêtre de congestion a donné lieu à différentes implantations de TCP (Reno, New Reno, Vegas, Tahoe…).

Pour réagir dynamiquement à la charge du réseau, la couche TCP utilise des indicateurs de mesure.

Il faut bien noter que dans beaucoup de situations, les retransmissions aggravent la congestion au lieu de la résorber. En effet, la congestion de certains routeurs conduit à la perte de segments (car ils sont rejetés par des routeurs saturés). Ensuite, les nœuds d'extrémité qui ont perdu leurs segments retransmettent, ce qui augmente la charge du réseau et donc à plus de pertes et ainsi de suite jusqu'à ce que le réseau se bloque complètement (congestion totale). Pour éviter de tomber dans de telles situations, TCP doit réduire le débit de transmission en cas de congestion constatée. Il faut rappeler que les routeurs utilisent les paquets ICMP pour avertir certains ordinateurs afin qu'ils réduisent leur débit. Donc l'initiative de réduction de débit peut être entreprise sur la base des RTT observés par la source ou suite à la réception de paquet ICMP.

La figure 1 montre les informations mémorisées par chaque site pour suivre l'évolution de l'utilisation des fenêtres chez l'émetteur (qui envoie les données) et chez le récepteur (qui envoie les acquittements).

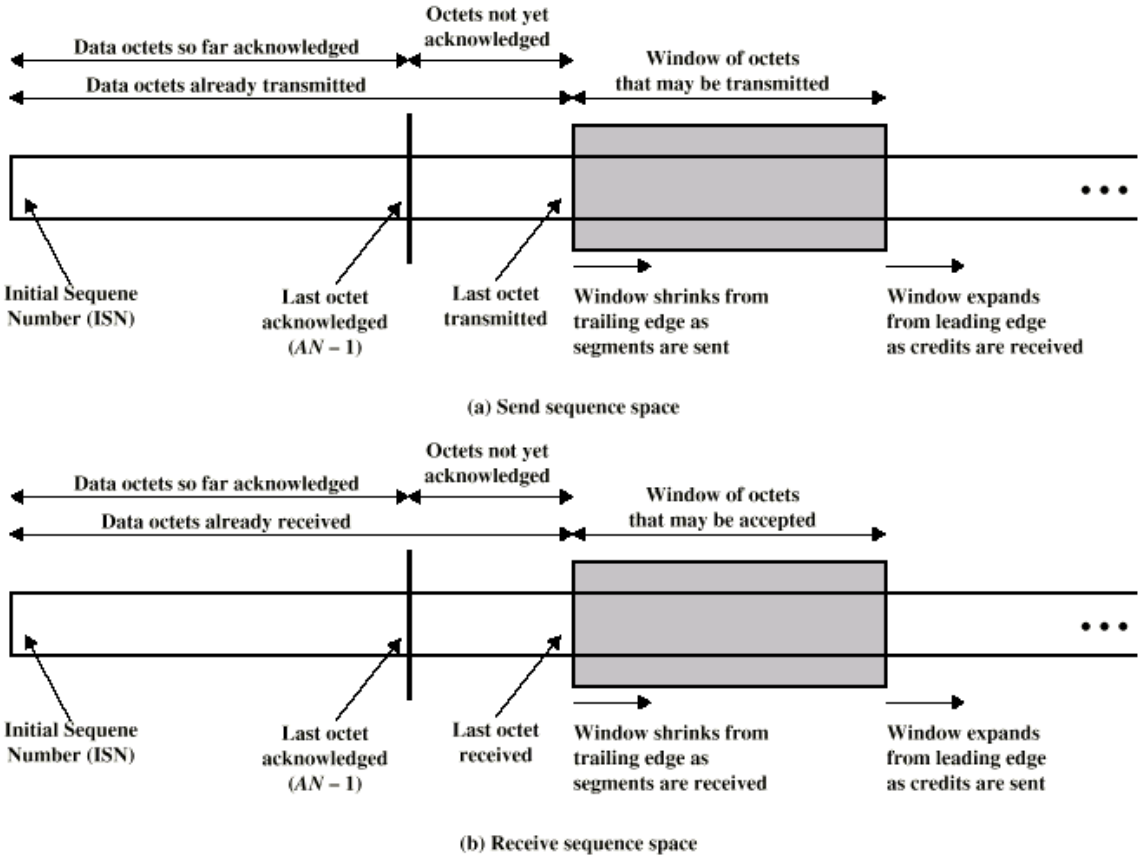
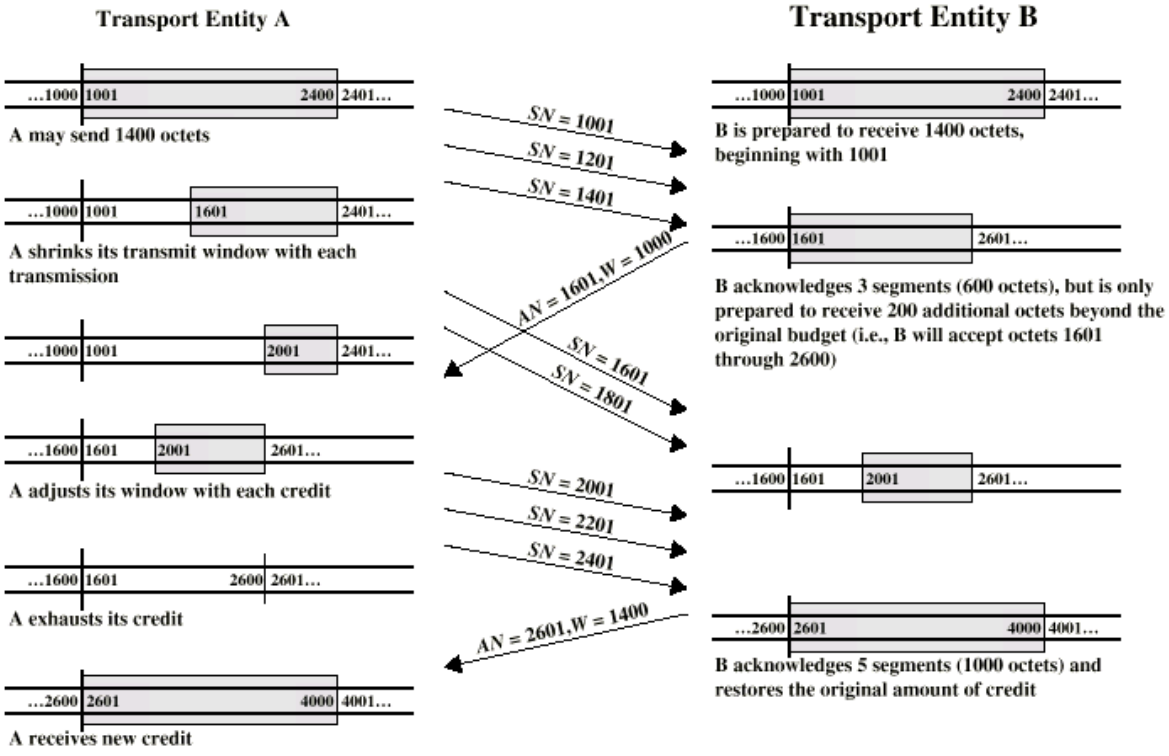


Figure 1. Principe d'utilisation de fenêtres

La figure 2 montre un exemple d'échange de données et acquittement avec changement de la taille de la *Fenêtre*.



SN: Numéro de séquence à l'émission AN : Numéro d'acquittement W : Fenêtre

Figure 2. Exemple d'échange de données et acquittement.

2. Principe "Additive increase – Multiplicative decrease"

Le standard TCP recommande l'utilisation combinée de deux techniques : ***démarrage lent*** et la ***diminution dichotomique***. Ces deux techniques combinées sont souvent désignées par *Additive increase – Multiplicative decrease*.

A tout instant, la source peut au maximum un nombre d'octets déterminé par la valeur de *Fenêtre de crédit* incluse dans l'acquittement. Pour contrôler la congestion, TCP gère un seuil appelé *Fenêtre de congestion*. A tout moment, TCP fonctionne comme si :

Fenêtre autorisée = minimum (Crédit restant, Fenêtre de congestion)
Crédit restant = Fenêtre de crédit – quantité d'octets envoyés et non encore acquittée

Dans les situations où il n'y a pas de congestion sur une connexion, la valeur de *Fenêtre de congestion* est égale à celle de Crédit restant. Dans ce cas, le rythme de transmission est limité uniquement par les capacités du récepteur.

Pour réduire la *Fenêtre de congestion*, TCP applique la technique de diminution dichotomique (dite aussi 'multiplicative decrease'), en considérant que la plupart des pertes de segments sont dues à des routeurs saturés. Cette technique fonctionne de la manière suivante :

> *En cas de perte d'un segment de données, réduire la fenêtre de congestion de moitié jusqu'à atteindre le minimum, c'est-à-dire un seul segment à la fois. Pour les segments qui restent dans la fenêtre d'émission, augmenter la temporisation de manière exponentielle (en la multipliant à chaque fois par deux).*

Lorsque la situation de congestion se résorbe, TCP doit revenir à un état de fonctionnement normal et augmenter son débit. Pour éviter un changement brutal du débit de transmission, TCP utilise la technique dite de *Démarrage lent* dont le principe est le suivant :

> *Au début du trafic sur une connexion ou lorsque que le trafic reprend après une phase de congestion, commencer une fenêtre de congestion limitée à un seul segment et incrémenter la fenêtre de congestion d'un segment à la fois chaque fois qu'un acquittement est reçu.*

Avec un démarrage lent, la source peut atteindre rapidement un rythme de croisière (c'est-à-dire où elle peut transmettre un maximum de segments) si les acquittements lui parviennent rapidement.

La figure 3 montre un exemple d'évolution du débit d'une connexion en fonction des situations de non-réception d'acquittement. Le débit observé fonctionne en dents de scie.

3. Syndrome de la fenêtre stupide

C'est une situation où les tampons d'émission ou de réception sont vidés à raison d'un octet (ou d'un petit nombre d'octets) par segment. Dans ce cas, du côté émetteur, TCP peut envoyer des segments qui ne contiennent qu'un octet de données par segment et du côté récepteur, TCP peut envoyer un acquittement pour chaque octet livré au destinataire. Comme chaque segment TCP est encapsulé dans un paquet IP, il y a un gaspillage des ressources du réseau. Les implantations de TCP doivent éviter de tomber dans de telles situations. Pour cela, TCP émetteur essaie de retarder le plus possible la transmission d'un segment tant qu'il n'a pas un nombre suffisant d'octets à émettre et le récepteur retarde l'envoi d'acquittement tant qu'une quantité suffisante d'octets n'a pas encore été délivrée au destinataire.

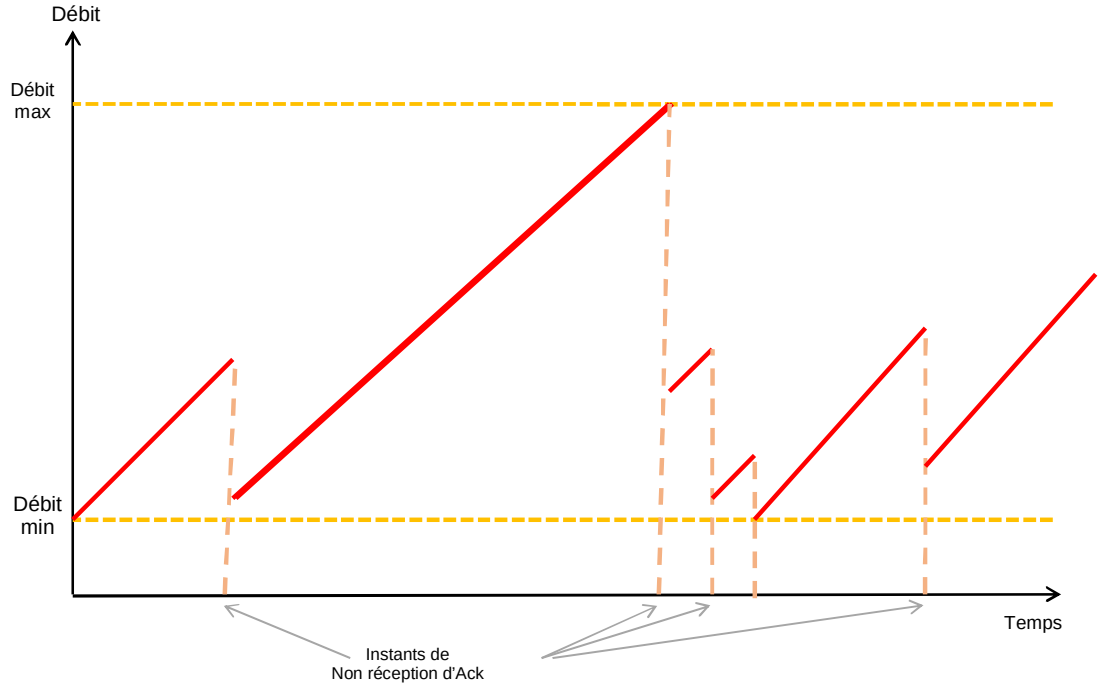


Figure 3. Evolution du débit d'une connexion en fonction à cause de la prte des acquittements.

4. Principe de retardement de segment

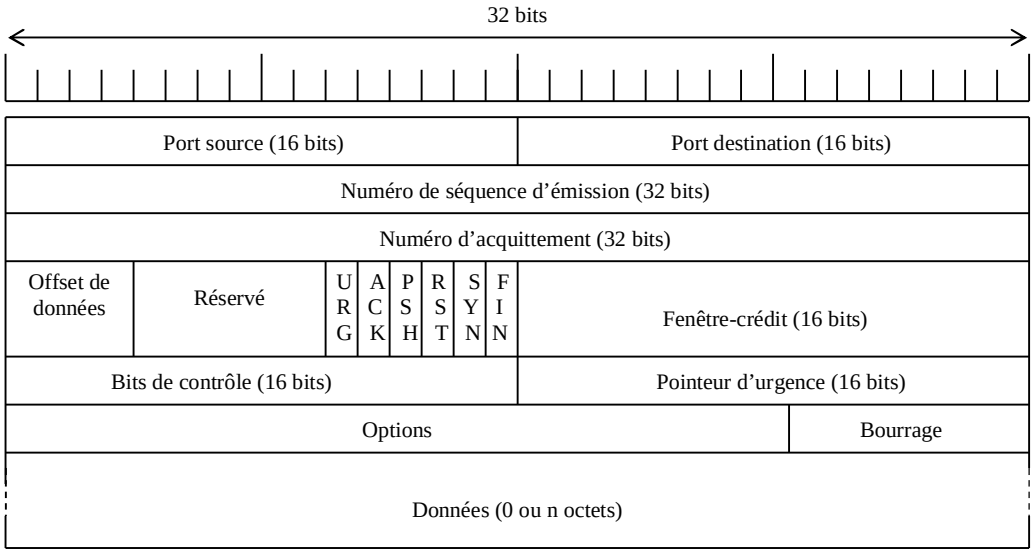
Beaucoup d'implémentations de TCP utilisent le principe de retardement des accusés de réception et les indications de *Fenêtre-crédit* pendant un certain délai (500 ms en général) dans l'espoir d'avoir plus de données à transmettre sur lesquelles on pourra greffer les accusés de réception et les informations de *Fenêtre-crédit* (qui seront alors transportés sans coût supplémentaire, c'est-à-dire sans utiliser un segment exprès pour ces informations). Ce principe permet d'améliorer l'utilisation de la bande passante, mais ce n'est pas toujours vrai.

5. Algorithme de Nagle

Cet algorithme fonctionne de la manière suivante : quand une application génère les données octet par octet (par exemple, une application où on saisit les caractères tapés par une personne), on envoie le premier octet et on accumule les autres jusqu'à ce que l'acquittement de l'octet envoyé soit reçu. Puis on envoie les octets accumulés dans un seul segment, ensuite on accumule les octets en attendant l'acquittement du segment envoyé. Et ainsi de suite... Cependant, pour plus d'efficacité si l'application génère des octets avec un rythme élevé, l'algorithme prévoit d'envoyer un segment s'il y a eu assez de données pour remplir la moitié de la fenêtre de crédit ou si les données accumulées ont atteint la taille maximale de segment.

L'algorithme de Nagle (1984) est disponible sur beaucoup d'implantations de TCP. Il peut être activé ou désactivé selon les spécificités des applications.

III. Format de segment TCP



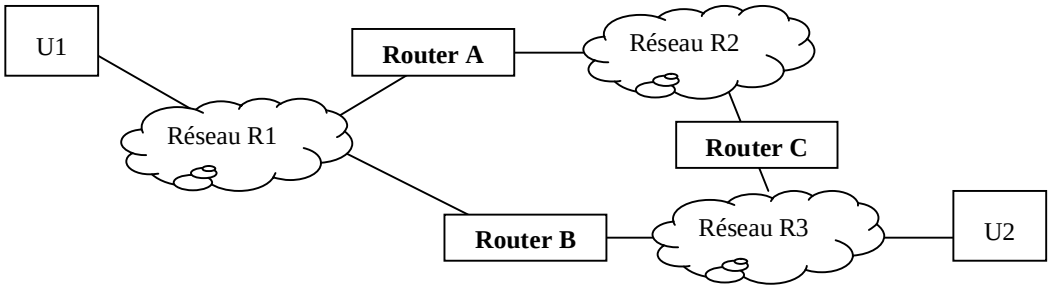
En-tête de segment TCP

- *Port source* et *Port destination* : indiquent les ports utilisés par la couche Application.
- *Numéro de séquence d'émission* : numéro du premier octet de données dans le segment (sauf pour un segment avec *SYN*=1). Si le bit *SYN* est à 1 (c.-à-d. si le segment est une demande de connexion), le numéro de séquence signale au destinataire que le prochain segment de données qui sera émis commencera à partir de l'octet *Numéro de séquence d'émission + 1*.
- *Numéro d'acquittement* : indique le numéro du prochain octet attendu par le destinataire. Si dans le segment reçu *FIN*=1 et *Numéro de séquence d'émission* = x, alors le *Numéro d'acquittement* renvoyé est x+1 (x est interprété comme étant le numéro de l'octet *FIN* et que cet octet est acquitté).
- *Offset de données* : des options (de taille variable) peuvent être intégrées à l'entête et des données de bourrage peuvent être rajoutées pour rendre la longueur de l'entête multiple de 4 octets. L'Offset indique la position relative où commencent les données.
- *Fenêtre-crédit* : indique le nombre d'octets que le destinataire peut recevoir. La notion de crédit est utilisée pour gérer le contrôle de flux entre émetteur et récepteur. Si *Fenêtre-crédit* = F et que le segment contient un *Numéro d'acquittement* = V, alors le récepteur accepte de recevoir les octets numérotés de V à V + F -1.
- *Bits de contrôle* : séquence de contrôle (CRC) portant sur l'entête de segment.
- Flags (*URG*, *ACK*, *PSH*, *RST*, *SYN*, *FIN*)
    - *URG* = 1 si le segment contient des données urgentes et *URG* = 0 sinon.
    - *ACK* = 1 indique que le numéro d'acquittement est valide et il peut être pris en compte par le récepteur. *ACK* = 0 si l'accusé de réception est non valide.
    - *PSH* = 1 indique que les données doivent être remises à l'application dès leur arrivée et de ne pas les stocker dans une file d'attente.
    - *RST* = 1 pour demander la réinitialisation d'une connexion TCP.
    - *SYN* = 1 et *ACK* = 0 servent à demander l'établissement de connexion TCP.
    - *SYN* = 1 et *ACK* = 1 servent à accepter une demande de connexion TCP.
    - *FIN* = 1 indique que l'émetteur n'a plus de données à émettre et demande de rompre la connexion de son côté.
- Pointeur d'urgence : indique l'emplacement (numéro d'octet) des données urgentes dans un segment. Il es valable uniquement si le bit URG=1.
- *Options* (taille variable) : options nécessitant des traitements particuliers.
- *Données (*de taille variable) : données provenant de la couche Application.

Exercices

Exercice 1

On s'intéresse à un utilisateur U1 de la couche TCP qui émet un segment de données de 3 K octets vers son correspondant U2. Ce segment traverse des routeurs interconnectés via 3 réseaux Ethernet R1, R2 et R3. Les MTU (Maximum Transfer Unit) des trois réseaux sont respectivement 1 k, 1.5 k et 0.5 k octets. Donner le nombre de trames émises et expliquer le principe de fragmentation et réassemblage du segment émis par U1.



Exercice 2

TCP assure le contrôle de congestion selon le principe "Additive Increase – Multiplicative decrease".

- en partant d'un réseau en sous-charge, au bout de combien de transmissions, une source peut-elle atteindre son rythme maximal ? et au bout de combien de temps ?
- en partant d'un réseau surchargé, au bout de combien de transmissions, une source se retrouve t-elle à son rythme de transmission le plus bas ?

On suppose que les segments ont une taille fixe Z.

Exercice 3

Q1. On suppose qu'à un instant *t* le RTT estimé est égal à 12 ms. Ensuite, trois accusés de réception sont reçus aux instants *t+10, t+80* et *t+100*.
Quelle est la nouvelle valeur du RTT estimée après le dernier Ack si on prend $\alpha = 0.9$?
Quelle est la nouvelle valeur du RTT estimée après le dernier Ack si on prend $\alpha = 0.1$?
Quelle conclusion peut-on tirer de la variation de α ?

Q2. Dans un réseau où un segment de données ne peut pas transporter plus de 128 octets, le temps minimum de traversée du réseau est de 25 ms et le numéro de séquence est sur 8 bits, quel est le débit maximal par connexion ?