ce document propose des éléments de correction d'exercices non-traités en cours

1 On considère la trace suivante, obtenue par l'analyseur de protocoles Ethereal installé sur la machine émettrice de la première trame Ethernet (les trames sont données sans préambule, ni CRC) :

```
00 :  000a  b7a3  4a00  0001  026f  5e9b  0800  4500
16 :  0028  0000  4000  4001  82ae  84e3  3d17  c2c7
32 :  490a  0800  75da  9c7a  0000  d445  a63a  622a
48 :  0900  ffff  ffff  0000  0000  0000

00 :  0001  026f  5e9b  000a  b7a3  4a00  0800  4500
16 :  0028  d092  0000  3a01  5adb  c2c7  490a  84e3
32 :  3d17  0000  7dda  9c7a  0000  d445  a63a  622a
48 :  0900  ffff  ffff  0000  0000  0000
```

1. L'adresse IP de la machine qui a initié l'échange est 0x84e33d17=132.227.61.23, qui est de classe B.
2. Son adresse physique est 00:01:02:6f:5e:9b.
3. L'adresse IP de la machine qui a répondu est 0xc2c7490a=194.199.73.10, qui est de classe C.
4. Son adresse physique est 00:0a:b7:a3:4a:00.
5. Dans la première trame, on a surligné les octets composant le paquet. Les six derniers octets sont des octets de remplissage de sorte que la trame Ethernet atteigne le minimum requis de 64 octets (CRC de 32 bits compris).
6. Le champ *Protocole* (octet numéroté 23 ici) vaut 0x01=1 indiquant que le protocole encapsulé par IP est ICMP. On peut alors imaginer qu'une commande comme *ping* par exemple soit à l'origine de cet échange sur le réseau (il s'agit en effet d'une requête d'écho (type ICMP **08** *Echo request*) suivie d'une réponse d'écho (type ICMP **00** *Echo response*)).

4 On analyse un échange constitué de quatre trames.

Couche liaison. Les quatre trames données s'inscrivent dans un échange entre deux entités—appelons-les A et B—dont les adresses Ethernet sont respectivement 08:00:20:73:5e:c6 et 00:e0:1e:de:72:d6. On comprend alors que les deux premières trames sont émises par A et les deux dernières par B.

Chaque en-tête Ethernet a un champ *protocole* égal à 0x0800 = 2048 indiquant l'encapsulation d'un paquet IP.

Couche réseau. Pour chaque paquet IP, les deux premiers octets de l'en-tête valent 0x4500 précisant qu'il s'agit de la *version* 4 d'IP, que la *longueur de l'en-tête* est toujours de 5 mots de 32 bits—autrement dit qu'il n'y a pas d'options IP—et qu'il n'y a pas de *type de service*.

Le premier paquet IP a une *longueur totale* de 0x0036 = 54 octets contre 0x0028 = 40 octets pour les trois autres.

Les quatre octets suivants donnent les informations de fragmentation. Les numéros d'identification sont respectivement 0x0a61 = 2657, 0x0a62 = 2658, 0xcc1e = 52254 et 0xce1e = 52766. Les deux octets 0x4000 communs aux quatre paquets indiquent qu'il n'y a pas eu de fragmentation et que le drapeau *Don't Fragment* est positionné.

L'octet suivant indique la *durée de vie* du paquet IP : elle est de 0xff = 256 secondes pour les deux premiers paquets contre 0x7f = 127 secondes pour les deux derniers.

L'octet *protocole* 0x06 = 6 indique que chacun des quatre paquets IP encapsule un segment TCP.

Les deux octets suivants correspondent au *total de contrôle de l'en-tête* du paquet IP.

Les huit derniers octets de l'en-tête IP fixe contiennent les *adresses source* et *destination* 0xa8b00319 et 0xa8b0036c, dont les écritures décimales pointées sont 168.176.3.25 et 168.176.3.108.

Couche transport. L'en-tête TCP commence par quatre octets précisant les *ports source* et *destination*, à savoir 0x0015 = 21 (port réservé du protocole FTP) et 0x0593 = 1427 (port de connexion).

Suivent les quatre octets du numéro de séquence (0x2950145f = 693113951 dans le premier segment, 0x2950146d = 693113965 dans le deuxième et 0x008acafd = 9095933 dans les deux derniers) puis ceux du numéro d'acquittement (0x008acafd = 9095933 puis 0x2950146e = 693113966).

Le premier chiffre hexadécimal de l'octet 0x50 commun aux quatre segments indique la *longueur de l'en-tête* TCP en mots de 32 bits : il n'y a donc pas d'options TCP.

L'octet suivant contient les *drapeaux* Urg, Ack, Psh, Rst, Syn et Fin. Seuls les bits Ack et Push sont positionnés pour le premier segment (0x18). Seuls les bits Ack et Fin sont positionnés pour le deuxième et le quatrième segment (0x11). Seul le bit Ack est positionné pour le troisième segment (0x10).

Les deux octets suivants indiquent la *taille de la fenêtre*, à savoir 0x2398 = 9112 octets dans les deux premiers segments et 0x203d = 8253 octets pour les deux derniers.

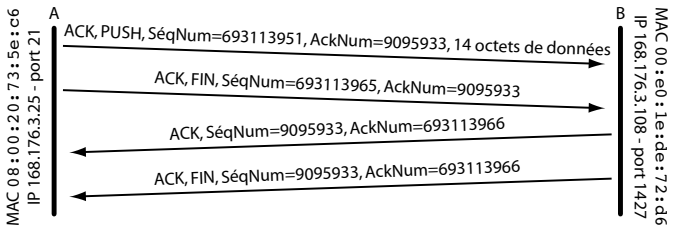
Les deux octets suivants correspondent au *total de contrôle* et les deux derniers au *pointeur d'urgence* de valeur nulle ici (le drapeau Urg n'étant pas positionné).

Couche application. Seul le premier segment contient des données (54 − 20 − 20 = 14 octets) :

```
32 : 32 : 31 : 20 : 47 : 6f : 6f : 64 : 62 : 79 : 65 : 2e : 0d : 0a
```

codant « 221 Goodbye.\CR\LF ».

Cet échange qui constitue la fin d'une connexion peut être représenté sous forme de chronogramme :



5 Trois extraits d'échanges ont été récupérés sur un réseau. Les trames sont complètes. Chaque échange comporte deux trames, la seconde répondant à la première. Sur les trois échanges, deux ont été modifiés à la main (par pur sadisme). La modification porte sur *un seul bit*.

1. Les protocoles sont :
   - premier échange : Ethernet, ARP ;
   - second échange : Ethernet, IP, UDP ;
   - troisième échange : Ethernet, IP, TCP.
2. Le premier et le troisième échanges ont été modifiés.

   Le premier échange est constitué d'une requête ARP et d'une réponse ARP. La requête demande l'adresse physique de la machine d'adresse IP 193.48.64.46 (0xc130402e) et la réponse donne l'adresse physique de la machine d'adresse IP 193.48.64.47 (0xc130402f). Un des deux champs adresse IP a donc été modifié d'un bit.

   Le premier segment du troisième échange est un segment TCP avec le bit *synchronisation* positionné (SYN=1) et le bit *accusé de réception* non positionné (ACK=0). Il s'agit donc du premier segment d'un établissement de connexion TCP. Le second segment semble être le second segment de l'établissement de connexion. En particulier, le *numéro d'accusé de réception* du second segment (0x36064724) est bien le *numéro de séquence* du premier (0x36064723) plus un. L'erreur est au niveau du champ drapeau : le bit *synchronisation* n'est pas positionné (SYN=0), ce qui est nécessaire pour le second segment d'un établissement de connexion.

3. L'échange non modifié est le second. Voici le détail de la première trame de cet échange.

Nom du champ	Valeur	Commentaire
Adresse Ethernet destination	00 d0 bc bf 6c 38	
Adresse Ethernet source	00 10 b5 03 88 44	
Type des données Ethernet	08 00	Code pour le protocole IP
Version du protocole IP	4	Version IPv4
Taille de l'entête IP	5	Pas d'options dans l'entête IP
Type de service	00	Pas de type de service requis
Taille totale du paquet IP	00 1d	Le paquet IP comporte 29 octets
Identificateur du paquet IP	83 ab	Pour la défragmentation
Informations sur la fragmentation	40 00	Fragmentation interdite
Durée de vie	40	Le paquet peut franchir 63 routeurs
Protocole IP encapsulé	11	Il s'agit du protocole UDP
Somme de contrôle de l'entête IP	bb 52	
Adresse IP source	c1 30 40 4f	193.48.64.79
Adresse IP destination	c1 30 39 22	193.48.63.34
Port UDP source	80 20	Un port utilisateur
Port UDP destination	00 0d	Le port du service d'envoi de la date
Taille du segment UDP	00 09	Le segment UDP fait 9 octets
Somme de contrôle du segment UDP	79 dc	
Données UDP	0a	Envoi d'une simple ligne vide

Voici le détail de la seconde trame de l'échange :

Nom du champ	Valeur	Commentaire
Adresse Ethernet destination	00 10 b5 03 88 44	
Adresse Ethernet source	00 d0 bc bf 6c 38	
Type des données Ethernet	08 00	Code pour le protocole IP
Version du protocole IP	4	Version classique IPv4
Taille de l'entête IP	5	Pas d'options dans l'entête IP
Type de service	00	Pas de type de service requis
Taille totale du paquet IP	00 36	Le paquet IP comporte 54 octets
Identificateur du paquet IP	00 00	Pour la défragmentation
Informations sur la fragmentation	40 00	Fragmentation interdite
Durée de vie	3f	
Protocole IP encapsulé	11	Il s'agit du protocole UDP
Somme de contrôle de l'entête IP	3f e5	
Adresse IP source	c1 30 39 22	193.48.63.34
Adresse IP destination	c1 30 40 4f	193.48.64.79
Port UDP source	00 0d	Le port du service d'envoi de la date
Port UDP destination	80 20	Le port utilisateur du segment précédent
Taille du segment UDP	00 22	Le segment UDP fait 34 octets
Somme de contrôle du segment UDP	82 ea	
Données UDP	46 72 69 20 41 70 72 20 32 38 20 31 34 3a 35 37 3a 34 39 20 32 30 30 36 0d 0a	La réponse du serveur de date : "Fri Apr 28 14:57:49 2006"

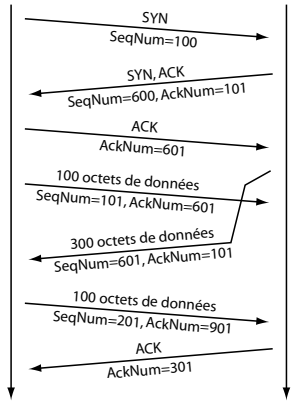
6 Lorsque TCP envoie un [SYN, SeqNum = x] ou un [FIN, SeqNum = x], le segment ACK correspondant porte AckNum = x+1. En d'autres termes, les segments SYN et FIN consomment une unité de l'espace de numérotation.

Il est nécessaire d'incrémenter le AckNum d'un **segment FIN**, de façon à ce que l'émetteur du FIN puisse déterminer si son FIN a bien été reçu, et non pas seulement les données précédentes.

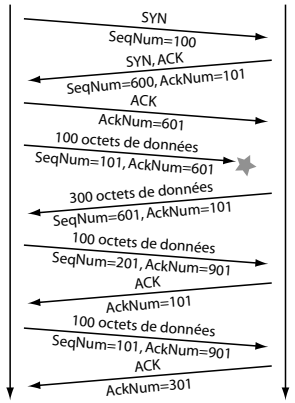
Pour un **segment SYN**, tout ACK pour des données postérieures au SYN incrémente AckNum : un tel ACK va donc acquitter implicitement le SYN (puisque des données ne peuvent pas être acquittées tant que la connexion n'est pas établie). Par conséquent, le fait d'avoir [ACK, AckNum = x+1] au lieu de [ACK, AckNum = x] relève plus d'une convention dans un souci d'homogénéité que d'une nécessité protocolaire.

7 On considère une connexion TCP entre deux applications distantes A et B. A doit envoyer à B deux segments de 100 octets de données chacun et B doit envoyer à A un segment de 300 octets de données. On sait que A est à l'origine de l'établissement et que la référence initiale de A est égale à 100 et celle de B à 600.

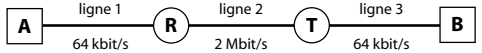
Voici le chronogramme d'un échange entre les deux hôtes :



Voici ce qui peut se passer si le premier segment de données de A se perd :



8 Considérons le réseau de datagrammes suivant où l'entité de transport de A transmet des données à celle de B (1024 octets de données par segment) en utilisant un protocole Stop & Wait.



Les routeurs R et T perdent en moyenne un paquet sur 10 (par congestion de leurs tampons). L'émetteur retransmet chaque paquet s'il ne reçoit pas un acquittement (24 octets) avant l'expiration de son temporisateur (T_1 = 600 ms). On demande de calculer pour ce type de scénario l'efficacité des différentes lignes (les lignes sont full-duplex) en considérant à la fois le trafic de données et le trafic des acquittements. On tient compte des délais introduits dans chaque équipement pour le traitement d'un datagramme (10 ms pour l'émetteur et le récepteur, 5 ms pour chaque routeur). Pour simplifier, on néglige les délais de propagation et les surcharges introduites par l'encapsulation des données.

On a

$$T_0 = T_{\text{data}} + T_{\text{ack}} + T_{\text{delay}} = 306,2\text{ms}$$

avec $T_{\text{data}} = 8 \times 1024 \times \left(\frac{1}{64.10^3} + \frac{1}{2.10^6} + \frac{1}{64.10^3}\right)$, $T_{\text{ack}} = 8 \times 24 \times \left(\frac{1}{64.10^3} + \frac{1}{2.10^6} + \frac{1}{64.10^3}\right)$ et $T_{\text{delay}} = 2 \times (10^{-2} + 5.10^{-3} + 5.10^{-3})$.

L'efficacité est le rapport entre le temps nécessaire à la transmission et le temps total du scénario. Nous savons que la probabilité de perdre un paquet dans un routeur est $p = 10^{-1}$. Un paquet émis par A a une probabilité $(1-p)^2$ d'être reçu par B et un paquet acquitté par B a une probabilité $(1-p)^2$ d'être reçu par A : un paquet émis par A a donc une probabilité $(1-p)^4$ que son acquittement soit reçu par A.

On peut poser $q = 1-(1-p)^4 = 0.3439$: q représente donc la probabilité que A envoie un paquet et qu'il ne reçoive pas son acquittement. Si A doit réexpédier n fois le paquet avant qu'il ne soit acquitté, A devra donc attendre $t(n) = T_0 + nT_1$. Le temps moyen d'un scénario est donc :

$$\begin{aligned}\bar{t} = \sum_{n\geq 0} q^n(1-q)t(n) &= (1-q)T_0\sum_{n\geq 0} q^n + q(1-q)T_1\sum_{n\geq 0} nq^{n-1} \\ &= (1-q)T_0\frac{1}{1-q} + q(1-q)T_1\frac{1}{(1-q)^2} \\ &= T_0 + \frac{q}{1-q}T_1 = 620,69 \text{ ms}.\end{aligned}$$

On peut maintenant calculer l'efficacité de chaque ligne :

ligne 1 : $\frac{1}{2}\cdot\frac{1}{\bar{t}}\cdot\frac{8\times 1024}{64.10^3} = 10,31\%$

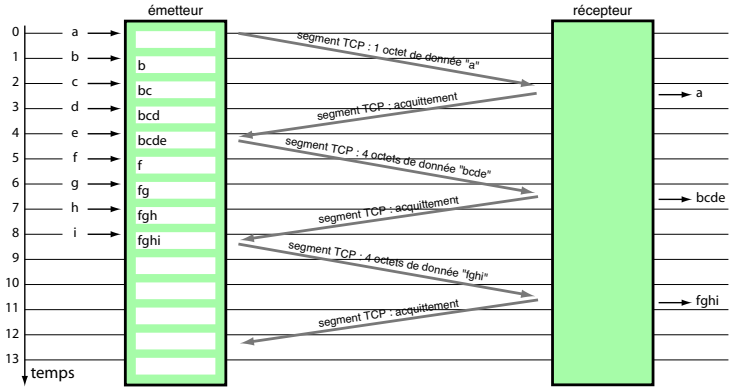
ligne 2 : $\frac{1}{2}\cdot\frac{1}{\bar{t}}\cdot\frac{8\times 1024}{2.10^6} = 0,33\%$

ligne 3 : $\frac{1}{2}\cdot\frac{1}{\bar{t}}\cdot\frac{8\times 1024}{64.10^3} = 10,31\%$

Le coefficient $\frac{1}{2}$ dans les expression d'efficacité provient du fait que les lignes full-duplex ne sont utilisées qu'en half-duplex puisque le protocole Stop & Wait est utilisé.

9 Supposons que les caractères « abcdefghi » soient envoyées, à raison d'une lettre par seconde, sur une connexion TCP dont le RTT est de 4,1 s.

1. L'algorithme de Nagle permet une amélioration de l'efficacité proposée pour les applications produisant des données octet par octet telles que sont les applications interactives. Il est intégrée dans la plupart des implémentations TCP et impose à l'émetteur de stocker les octets de données passées par l'application source (même s'il y a des push) avant de constituer un segment, tant qu'il n'a pas MSS (Maximum Segment Size) octets ou tant qu'il n'a pas reçu le ACK du segment précédent.
2. Voici le chronogramme des émissions de segments :



3. L'écho des caractères ne se fait que toutes les 4 secondes, et par bloc de 4 caractères. Un tel comportement est courant sur les connexions Telnet, même pour celles ayant des RTT plus modestes.

10 L'algorithme *Congestion Avoidance* est utilisé pour fc≥seuil et ce jusqu'à ce qu'il y ait détection de congestion. Posons fc = $n\times$MSS. La source attend n ACK acquittant des données non encore acquittées (*i.e.* non dupliqués). À la réception de chacun d'eux, elle augmente sa fenêtre de MSS^2/fc. Ainsi, on obtient :

$$\text{fc} \longleftarrow \text{fc} + n\times\text{MSS}^2/\text{fc} = \text{fc} + n\times\text{MSS}^2/(n\times\text{MSS}) = \text{fc} + \text{MSS}.$$

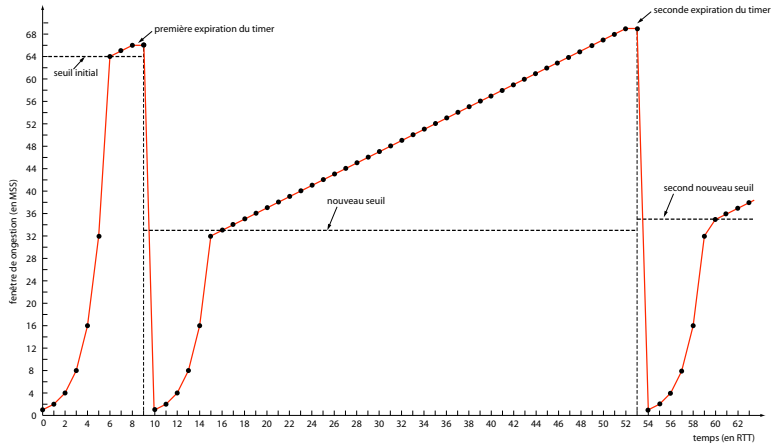
On parle alors d'accroissement additif (*Additive Increase*).

11 Un récepteur émet immédiatement un acquittement (dupliqué) lorsqu'un segment déséquencé est reçu. Cet ack sert à informer l'émetteur de la réception du paquet déséquencé et du numéro de séquence attendu. Sur réception d'un ack dupliqué, l'émetteur n'est toutefois pas capable de déterminer de manière sûre si cette duplication est due à la perte d'un paquet, à un déséquencement ou à une duplication dans le réseau.

Si l'émetteur reçoit plus de trois acquittements, il est extrêmement probable que la cause est une perte de paquet et la retransmission devient nécessaire. Dans les autres cas, l'émetteur considère qu'un simple déséquencement est à l'origine de l'ack dupliqué. La réception d'au moins trois acks dupliqués indique qu'un seul paquet s'est perdu. L'émetteur considère une perte due à une erreur sur les données, il retransmet le segment manquant. Il est donc inutile d'appliquer l'algorithme de contrôle de congestion par une diminution brutale de la taille de la fenêtre de l'émetteur, d'autant plus qu'un certain nombre de paquets sont encore en transit sur le réseau. Ces deux principes correspondent aux algorithmes de *retransmission rapide* et de *récupération rapide*. À la réception du troisième ack dupliqué, l'émetteur part du principe qu'un (seul) paquet s'est perdu. Il applique alors le *Fast Recovery* (et non le *Slow Start* comme dans le cas d'une expiration du timer de retransmission) qui régit la transmission de nouvelles données jusqu'à ce qu'un ack non dupliqué arrive. Il y a deux raisons à ce choix :

- le récepteur ne peut générer un acquittement dupliqué à la réception d'un segment uniquement si ce dernier a quitté le réseau et se trouve dans les tampons de réception du récepteur (et ne consomme donc pas de ressources réseau).
- il y a préservation du *ack clocking* (principe de conservation des paquets) ce qui permet à l'émetteur de continuer à transmettre de nouveaux segments.

12 Voici la courbe illustrant les variations de la taille de la fenêtre de congestion de TCP sous les hypothèses suivantes. La taille maximum de segment est de 1024 octets et, initialement, le seuil de congestion est fixé à 64 Ko. Le temporisateur de retransmission expire une première fois après 9 RTT, puis une seconde fois après 46 nouveaux RTT.



13 Supposons que la fenêtre de congestion de TCP soit égale à 18 Ko (MSS = 1 Ko) et qu'un temporisateur expire. Pour que la taille soit maximale, il faut que les rafales successives remplissent la fenêtre de congestion, sinon la taille de la fenêtre augmentera plus lentement que par doublements successifs, ce qui ne permettra pas d'atteindre la taille maximale.

Après l'expiration, la fenêtre de congestion vaut la taille de un MSS (1 Ko) et le seuil de congestion vaut la moitié (9 MSS) de la taille de la fenêtre de congestion avant le crash.

temps	reçu	taille de la fenêtre	émis
0	–	1	1 MSS
1 RTT	1 ACK	$1+1=2$	2 MSS
2 RTT	2 ACK	$2+2=4$	4 MSS
3 RTT	4 ACK	$4+4=8$	8 MSS
4 RTT	8 ACK	$\min(8+8,9)$= 9	9 MSS

La taille maximale de la fenêtre après 4 rafales de segments vaut 9 Ko.

14 Une machine X transmet des données à une machine Y à l'aide d'une connexion TCP sur laquelle se trouvent trois routeurs R, S et T. La fenêtre de réception de Y est de 12 MSS, sachant que la taille maximale des segments a été négociée à 512 octets. Le temps de réaction des routeurs est $T_r = 10$ ms, et le temps de traitement des stations (vérification d'un segment et génération d'un acquittement, vérification d'un acquittement et préparation d'émission d'un nouveau segment) est $T_t = 15$ ms. Chaque segment reçu par Y donne lieu à l'émission immédiate d'un acquittement (segment TCP vide). La durée de temporisation avant que X ne considère un segment comme perdu est $T_{\text{temp}} = 2$ s. On prendra en compte la surcharge introduite par les encapsulations TCP et IP.

1. Calculons le RTT, c'est-à-dire le délai entre l'émission du premier bit d'un paquet et la réception du dernier bit de son acquittement. On a $T_{\text{MSS@500k}} = \dfrac{(512+40)\times 8}{500.10^3} = 8,832$ ms et $T_{\text{ACK@500k}} = \dfrac{40\times 8}{500.10^3} = 0,640$ ms d'où

$$\text{RTT} = 3,5T_{\text{MSS@500k}} + 3,5T_{\text{ACK@500k}} + 6T_r + 2T_t = 123,152 \text{ ms.}$$

On trouve alors $\dfrac{\text{fr}\times(40+\text{MSS})\times 8}{\text{RTT}} \sim 431.10^3 \leq 500.10^3$, ce qui indique que c'est la fenêtre de réception fr qui va limiter la vitesse maximale et que la taille maximale de la fenêtre de congestion sera la taille de la fenêtre de réception.

Le seuil de congestion étant initialisé à 4 Ko=8 MSS, les algorithmes *Slow Start* et *Congestion Avoidance* donnent :

temps	reçu	taille de la fenêtre	émis
0	–	1	1 MSS
1 RTT	1 ACK	1+1=2	2 MSS
2 RTT	2 ACK	2+2=4	4 MSS
3 RTT	4 ACK	4+4=8	8 MSS
4 RTT	8 ACK	8+1= 9	9 MSS
5 RTT	9 ACK	9+1=10	10 MSS
6 RTT	10 ACK	10+1=11	11 MSS
7 RTT	11 ACK	11+1=12	12 MSS

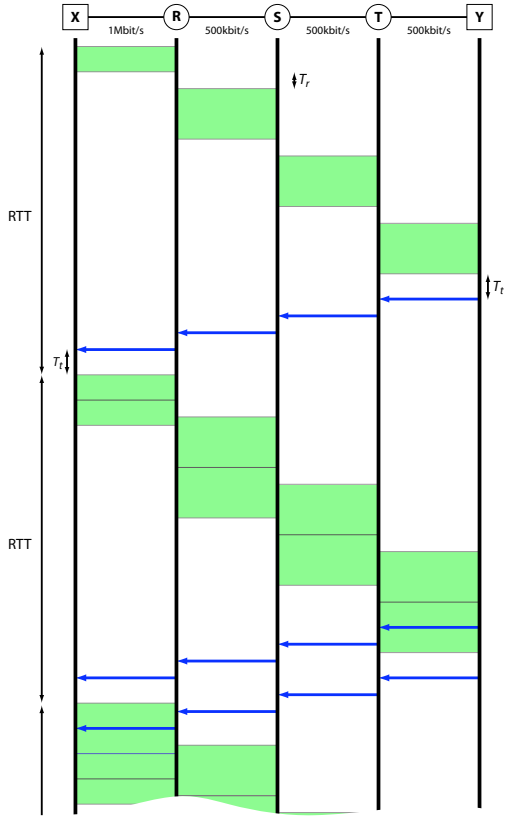
Il faut donc 7 RTT, soit environ 862 ms, pour atteindre le débit maximal (correspondant à la fenêtre de réception de Y annoncée à 12 MSS).

2. À la vitesse maximale, le débit est

$$D = \frac{\text{fc}\times(40+\text{MSS})\times 8}{\text{RTT}} = \frac{12\times(40+512)\times 8}{0,123152} \sim 431 \text{ kbit/s.}$$

Sur les lignes à 500 kbit/s, cela correspond à une efficacité de 86%, et à une efficacité de 43% sur la ligne à 1Mbit/s.

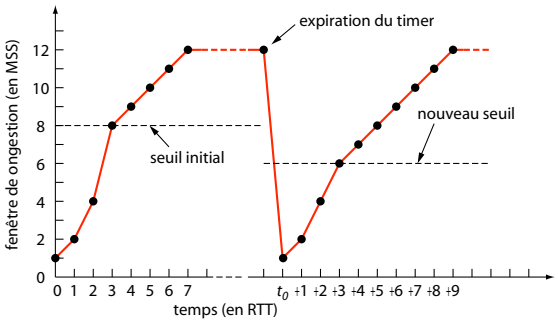
3. Puisque tous les acquittements d'une rafale viennent d'être perdus, il faudra attendre l'expiration du timer de retransmission pour que la perte soit détectée, après quoi, disons à $t = t_0$, un nouveau *Slow Start* sera nécessaire. Le seuil de congestion pour ce nouveau *Slow Start* est fixé à la moitié de la fenêtre de congestion actuelle, soit seuil $= \dfrac{\text{fc}}{2} = 6$ MSS.



On obtient :

temps	reçu	taille de la fenêtre	émis
t_0	–	1	1 MSS
t_0+1 RTT	1 ACK	1+1=2	2 MSS
t_0+2 RTT	2 ACK	2+2=4	4 MSS
t_0+3 RTT	4 ACK	min(4+4,seuil)=6	6 MSS
t_0+4 RTT	6 ACK	6+1= 7	7 MSS
t_0+5 RTT	7 ACK	7+1= 8	8 MSS
t_0+6 RTT	8 ACK	8+1= 9	9 MSS
t_0+7 RTT	9 ACK	9+1=10	10 MSS
t_0+8 RTT	10 ACK	10+1=11	11 MSS
t_0+9 RTT	11 ACK	11+1=12	12 MSS

La graphique suivant montre l'évolution de la fenêtre de congestion :



Cette fois, 9 RTT seront nécessaires pour atteindre le débit maximal, ce qui donne donc un délai global $T_{\text{récup}} = T_{\text{temp}} + 9\ \text{RTT} \sim 3,108$ s.

15 Toutes les lignes sont full duplex et A a une infinité de données à transmettre à B à l'aide d'une connexion TCP. La fenêtre de réception de B est fr=64 MSS et la taille maximale des segments est MSS=1460 octets. Le temps de traitement de R4 est de $T_4 = 300$ ms (on suppose que les autres équipements ont un temps de traitement nul). Chaque segment reçu par B donne lieu à l'émission immédiate d'un acquittement (segment TCP vide). Si A ne reçoit pas d'acquittement dans les deux secondes qui suivent l'envoi d'un paquet, il considère celui-ci comme perdu. On prendra en compte la surcharge introduite par les encapsulations TCP et IP.

1. Le seuil de congestion est initialisé à 32 MSS et le chemin choisi est celui passant par R4. Calculons le *Round Trip Time* :

$$\text{RTT} = T_{\text{MSS}} + T_{\text{ACK}} + T_{\text{pr}} + T_{\text{tr}}$$

avec $T_{\text{MSS}} = 2\times 8(1460+40)\left(\frac{1}{4.10^6} + \frac{1}{1.10^6}\right) = 0,03$ s, $T_{\text{ACK}} = 2\times 8\times 40\left(\frac{1}{4.10^6} + \frac{1}{1.10^6}\right) = 0,0008$ s, $T_{\text{pr}} = 2(2\times 20.10^{-3} + 2\times 72,3.10^{-3}) = 0,3692$ s et $T_{\text{tr}} = 2T_4 = 0,6$ s. On obtient $\text{RTT} = 0,03 + 0,0008 + 0,3692 + 0,6 = 1$ s.

On trouve alors $\dfrac{\text{fr}\times 8(1460+40)}{\text{RTT}} \leq 1.10^6$, ce qui indique que c'est la fenêtre de réception qui va limiter la vitesse maximale et que la taille maximale de la fenêtre de congestion sera la taille de la fenêtre de réception.

Les algorithmes *Slow Start* et *Congestion Avoidance* donnent :

temps	reçu	taille de la fenêtre	émis
0	–	1	1 MSS
1 RTT	1 ACK	1+1=2	2 MSS
2 RTT	2 ACK	2+2=4	4 MSS
3 RTT	4 ACK	4+4=8	8 MSS
4 RTT	8 ACK	8+8=16	16 MSS
5 RTT	16 ACK	16+16=32	32 MSS
6 RTT	32 ACK	32+1=33	33 MSS
7 RTT	33 ACK	33+1=34	34 MSS
⋮	⋮	⋮	⋮
37 RTT	63 ACK	63+1=64	64 MSS

Il faudra donc 37 RTT soit 37 s pour parvenir au débit maximal. Le débit maximal est égal à $D = \frac{64\times 8(1460+40)}{1} = 768$ kbit/s. On peut alors calculer l'efficacité des différentes lignes : 9,6% pour celles à 4 Mbit/s et 38,4% pour celles à 1 Mbit/s.

2. Après 100 s, R1 envoie le premier paquet de la rafale sur R4 avant de décider de router tous les paquets suivants par R3. Puisque 100 s se sont écoulées, on est au débit maximal D. Le temps pour que le premier paquet arrive en B (à partir de R1) est égal à :

$$t_1 = 2\frac{1500\times 8}{1.10^6} + \frac{1500\times 8}{4.10^6} + \frac{2\times 72,3 + 20 + 300}{1000} = 0,4916 \text{ s.}$$

Pour chacun des paquets suivants, le temps nécessaire est :

$$t_2 = 3\frac{1500\times 8}{4.10^6} + 3.20.10^{-3} = 0,069 \text{ s.}$$

Il faut donc $0,207$ s pour que les trois paquets suivants qui arriveront avant le premier paquet de la rafale. Dès lors, A va recevoir trois doublons d'acquittements pour le paquet envoyé avant le premier paquet de la rafale. On a alors les mises à jour suivantes : seuil $\longleftarrow \dfrac{\text{fc}}{2} = 32$ et fc $\longleftarrow$ seuil $= 32$.

Le nouveau *Round Trip Time* vaut

$$\text{RTT} = 4\frac{1500\times 8}{4.10^6} + 8.20.10^{-3} + 4\frac{40\times 8}{4.10^6} = 172,32 \text{ ms.}$$

On trouve alors $\dfrac{\text{fr}\times 8(1460+40)}{\text{RTT}} (\sim 4457.10^3) \geq 4.10^6$, ce qui indique que ce n'est pas la fenêtre de réception mais la capacité du sous-réseau qui va limiter la vitesse maximale. La taille maximale de la fenêtre de congestion doit vérifier $\dfrac{\text{fc}\times 8(1460+40)}{\text{RTT}} \leq 4.10^6$, soit $\text{fc} \leq 57,44$: la taille maximale de la fenêtre de congestion est donc de 57 MSS. Comme la taille courante de la fenêtre de congestion est fixée 32 MSS par l'algorithme *Fast Retransmit Fast Recovery*, il faudra 25 RTT $= 4,308$ s pour atteindre le débit maximal.