TD Révisions R&P

Partie I

On considère une connexion TCP entre 2 machines MA et MB. Le RTT, supposé constant, vaut 100 ms et la fenêtre W annoncée par MB vaut 64Ko. On note CW la taille de la fenêtre de congestion de MA. On suppose que tous les segments font 1500 octets. On considère 3 variantes de TCP. Variante A : pas de fast retransmit ni de SACK (TCP basique), variante B : fast retransmit (triple ACK) mais pas SACK, variante C : SACK (Selective Acknowledgement). MA a envoyé un segment de numéro de séquence X, qui est bien parvenu à MB, puis un segment de numéro de séquence Y qui s ?est perdu.

1. Rappeler le principe du calcul du délai de retransmission de TCP (RTO).
2. Décrire le comportement de MA et MB et les segments échangés, avec ces 3 variantes de TCP dans les 4 cas suivants :
   – Cas 1 : MA n'a plus de données à transmettre
   – Cas 2 : MA a encore un segment de données à transmettre
   – Cas 3 : MA a encore 10 segments à transmettre et CW contient au plus 2 segments
   – Cas 4 : MA a encore 10 segments à transmettre et CW contient plus de 3 segments
3. Après quel délai le segment Y est-il reçu pour chaque combinaison des 4 cas et des 3 variantes de TCP ?
4. Le cas 3 (respectivement 4) est-il plus ou moins fréquent pendant la phase de démarrage lent (slow start) ou pendant la phase d ?évitement de congestion (congestion avoidance) ?
5. Si le segment Y ne se perd pas, mais si son acquittement se perd, cela change-t-il vos réponses à la question 2 ?

Corrections :

1. Pour chaque segment acquitté (et non retransmis), l'émetteur calcule son RTT, et s'en sert pour ajuster un RTT moyen (moyenne pondérée des RTT mesurés). Le RTO est ensuite calculé comme un petit multiple du RTT estimé (par exemple RTO = 2 RTT).
2. Dans le cas 1, les 3 variantes ont le même comportement : l'émetteur doit attendre le RTO et retransmettre le segment (et repartir en slow start). Dans le cas 2 : Variante A : l'émetteur attend RTO et retransmet le segment. Variante B : même chose car 2 il y a un seul Ack dupliqué, donc le fast retransmit ne se déclenche pas. Variante C : après un RTT, MA reçoit un SACK de MB indiquant que MB attend le segment Y et a reçu le segment Y+1500. MA retransmet donc le segment Y sans attendre le timeout. Dans le cas 3 : Comme dans le cas 2, MA peut envoyer au plus un segment après le segment Y, donc pour les variantes A et B, MA doit attendre le timeout. Pour la variante 3, le segment envoyé après Y permet de déclencher le SACK. Dans le cas 4 : Pas de changement pour la variante A. Par contre pour la variante B, comme MA envoie plus de 3 segments après le segment Y, le tripe ACK permet de déclencher le fast retransmit : le segment Y est ré-émis dès réception du triple ACK. Pour la variante C, même situation que dans les cas 2 et 3.
3. Dans la variante A, cas 1,2,3 et 4, dans la variante B cas 1,2 et 3 et variante C cas 1, il faut attendre le timeout, donc Y est retardé d'environ RTO = 2 RTT. Dans la variante B cas 4, il faut attendre le triple ACK, donc le délai est un RTT plus l'émission de 3 segments. Pour la variante C cas 2,3,4, il suffit d'attendre un RTT plus l'émission d'un segment.
4. Si CW vaut 2, c'est qu'on est en début de phase de slow start (soit démarrage, soit après un timeout). Plus CW est grand, plus il est plausible que MA soit en évitement de congestion. En particulier, le cas 4 devrait être beaucoup plus fréquent que le cas 3.

5. Comme les acquittements de TCP sont cumulatifs, dans les cas 2,3 et 4 l'acquittement du segment Y+1500 acquittera aussi le segment Y, donc il n'y aura pas retransmission. Dans le cas 1, MA ne recevant rien se comporte comme dans la question 2, retransmet Y sur timeout, et MB retransmet un acquittement pour Y.

Partie II

On considère une connexion TCP entre deux hôtes E et R. E a toujours des données à transmettre mais pas R. On supposera qu'un récepteur acquitte immédiatement tous les segments reçus. La version de TCP utilisée est la version la plus basique (Tahoe). On supposera également que R a une fenêtre de réception de taille W. On notera RTT le délai d'aller-retour entre E et R.

1. Rappelez l'intérêt et détaillez le déroulement de la phase de connexion de TCP.
2. Après l'initialisation de la connexion, E commence à émettre des segments TCP suivant la phase de slow-start. Durant cette phase, comment évolue la taille de la fenêtre d'émission de E ? Quel est l'incidence sur le débit ?
3. Calculez le temps mis par E pour transmettre les 15 premiers segments en fonction du RTT.
4. Calculez le débit maximum théorique que E puisse atteindre en fonction du RTT et de W.
5. Supposons qu'au temps t_1 une perte se produise. Sans optimisation particulière, comment est détectée cette perte ? Au bout de combient de temps ?
6. Suite à la détection de cette perte, détaillez le comportement de E (retransmission, taille de fenêtre, débit, etc.).
7. On suppose que E est au courant de la perte au temps t_2. Juste avant cette perte, E avait atteint le débit maximum théorique. Calculez le temps mis à partir de t_2 pour que E atteigne à nouveau le débit maximum théorique en l'absence de perte.
8. En l'absence de perte, E et R mettent très longtemps à atteindre le débit maximum théorique de la connexion. Détaillez la (les) cause(s) responsable(s) de cette mauvaise performance. Proposez des mécanismes pour les éviter.

Corrections :

1. La phase de connexion de TCP permet de créer un tunnel logique entre les deux couches transport des deux extrémités. En outre, elle permet d'échanger des informations telles que les options supportées par chaque partie. Elle se déroule en 3 étapes : émission d'un message SYN par le client, émission d'un message SYN/ACK par le serveur suite à la réception d'un message SYN et émission d'un message ACK par le client suite à la réception du message SYN/ACK.
2. La fenêtre d'émission de E augmente de façon exponentielle lors de la phase de slow-start. Par conséquent le débit augmente également de façon exponentielle.
3. E est en phase de slow-start. Par conséquent :
   – 1 segment aura été transmis après 1 RTT
   – 3 segments auront été transmis après 2 RTT
   – 7 segments auront été transmis après 3 RTT
   – 15 segments auront été transmis après 4 RTT
   La formule est : nb segments = $2^n - 1$ où n est le nombre de RTT.
4. Au maximum l'émetteur peut envoyer une fenêtre complète par RTT. Par conséquent, le débit maximum théorique est : $Dmax = \frac{W}{RTT}$
5. Sans optimisation, cette perte sera détectée suite à l'expiration d'un temporisateur (timeout) sur la réception de l'acquittement. Les temporisateurs sont armés à RTO (Retransmission TimeOut) où généralement RTO = 2 RTT. Donc elle sera détectée environ 2RTT après t1.
6. Suite à l'expiration d'un temporisateur, E :

- positionne le seuil ssthresh (seuil de démarrage lent) à la moitié de la taille de sa fenêtre d'émission lors de la perte ;
- réduit sa taille de fenêtre d'émission à 1 ;
- recommence la transmission à partir de la première trame non acquittée en phase de slow-start jusqu ?à atteindre le seuil ssthresh puis passe en phase d'évitement de congestion.

7. Au temps t2, la taille de la fenêtre d'émission de E était égale à W vu qu ?il était au débit maximum théorique. Suite à la détection de la perte, il applique les operations de la question 6 (avec `ssthresh` = $W/2$). Pour atteindre à nouveau le débit maximal, il faut : ? pendant la phase de slow-start, la fenêtre augmente exponetiellement jusqu ?à atteindre $W/2$ (le seuil `ssthresh`) : $W/2 = 2^n - 1$ => il faut donc nRTT avec n = $log_2(\frac{W}{2}+1)$? pendant la phase d'évitement de congestion (mode dans lequel passe l'émetteur dès que sa fenêtre d'émission a atteint `ssthresh`), la fenêtre augmente linéairement de $W/2$ jusqu'à atteindre W : il faut donc $W/2$ RTT. En conclusion, il faut $(log_2(W/2+1) + W/2)\times$ RTT à E pour atteindre à nouveau le débit maximum théorique.
8. Même lors de la phase de slow-start, l'augmentation de la fenêtre d'émission (et par conséquent du débit) est dépendante du RTT. Si ce dernier est très important (long fat networks) alors le temps nécessaire pour aller d'une taille de fenêtre à la suivante sera également long. Pour résoudre ce type de problème, on peut par exemple introduire une nouvelle phase d'augmentation du débit plus aggressive que la phase de slow-start. Dès lors un émetteur pourrait passer par 3 phases : phase très aggressive, puis slow-start puis évitement de congestion.

Partie III

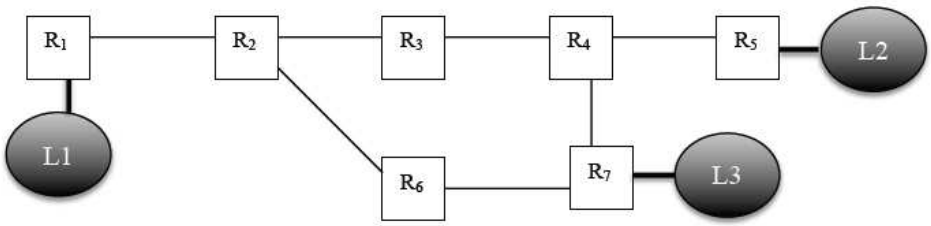

FIGURE 1 – Mon super réseau

On considère le réseau ci-dessus, où les R_i sont des routeurs IP et où L1, L2, L3 sont des réseaux locaux qui contiennent respectivement 25, 70 et 50 machines. On dispose du réseau IP 192.168.100.0/24 pour adresser l'ensemble.

1. Donner un schéma d'adressage possible pour chaque réseau local et chaque liaison entre routeurs. Pour chaque réseau Li, donner le nombre maximum de machines qui pourrait être connectées.
2. Reste-t-il assez d'adresses pour ajouter un LAN L4, et si oui combien de machines pourrait-il contenir ?
3. Une machine du réseau L1 est déplacée dans le réseau L2. Peut-elle conserver son adresse IP ? Justifiez. On suppose que chaque routeur utilise le protocole de routage RIP avec retour empoisonné, le poids de chaque lien étant 1. Initialement les liens R2 - R6 et R4 - R5 sont coupés.
4. Donnez la table de routage de R1.
5. Le lien R2 - R6 est rétabli. Donnez les vecteurs de distance qui sont échangés, et la nouvelle table de R1.
6. Même question après que le lien R4 - R5 soit aussi rétabli.
7. Le lien R4 - R5 est de nouveau coupé. Pour quelle(s) destination(s) peut-il y avoir "comptage à l'infini". Donnez un exemple d'échange de vecteurs de distances jusqu'à stabilisation des tables de routage dans un cas de comptage à l'infini. Combien de fois la table de R1 change-t-elle ?

Corrections :

1. Dans la suite on suppose que dans un sous-réseau l'adresse nulle et l'adresse tout à 1 (broadcast) sont réservées. On peut affecter 192.168.100.0/25 (126 machines, y compris le routeur) à L2, 192.168.100.128/26 (62 machines) à L3 et 192.168.192/27 (30 machines) à L1. On a ensuite 7 liens point-à-point entre routeurs qui a priori utiliseront chacun un sousréseau de 4 adresses, par exemple 192.168.100.224/30 pour le lien R1 ? R2, 192.168.100.228/30 pour le lien R2 - R3, ? (total 7 réseaux /30), il reste donc un /30 libre.
2. Le /30 libre permet de créer L4, avec 2 machines (dont le routeur), voire 3 machines si on utilise l'adresse nulle.
3. Les paquets envoyés à l'ancienne adresse de la machine seront routés de proche en proche vers le sous-réseau L1 et n'atteindrons donc pas cette machine. Il faut donc lui attribuer une nouvelle adresse dans le sous-réseau de L2. On suppose que chaque routeur utilise le protocole de routage RIP avec retour empoisonné, le poids de chaque lien étant 1. Initialement les liens R2 - R6 et R4 - R5 sont coupés.
4.

Destination	Prochain saut	distance
R1	Local	0
R2	R2	1
R3	R2	2
R4	R2	3
R6	R2	5
R7	R2	4
L1	Local	0
L3	R2	5

   R5 et L2 ne figurent pas dans la table initiale car ils sont non accessibles.
5. R2 et R6 s'envoient mutuellement leurs vecteurs complets (toutes les destinations connues). R2 met à jour sa distance à R6, R7 et L3 et envoie la mise à jour ((R6,1), (R7,2) ,(L3,3)) à R1 et R3 et ((R6,∞), (R7,∞) ,(L3,∞)) à R6 : retour empoisonné. Symétriquement, R6 met à jour sa distance à L1, R1, R2 et R3, et envoie ((L1,3), (R1,2), (R2,1),(R3,2)) à R7 et ((L1,∞), (R1,∞), (R2,∞),(R3,∞)) à R2. A réception du vecteur de R2, R1 met à jour sa distance à R6, R7 et L3 et envoie ((R6,∞), (R7,∞), (L3,∞)) à R2. A réception du vecteur de R2, R3 ne change que sa distance à R6 (celles pour R7 et L3 n'ont pas diminué), et envoie ((R6,2)) à R4 et ((R6,∞)) à R2, ce qui ne changera rien. De son côté, R7 met à jour sa distance L1, R1, R2 (pas de changement pour R3) et envoie ((L1,4), (R1,3),(R2,2)) à R4 et ((L1,∞), (R1,∞),(R2,∞)) à R6. Ces deux vecteurs n'ont aucun effet, et le routage a maintenant convergé. La nouvelle table de R1 est :

Destination	Prochain saut	distance
R1	Local	0
R2	R2	1
R3	R2	2
R4	R2	3
R6	R2	2
R7	R2	3
L1	Local	0
L3	R2	4

6. R5 reçoit un vecteur de R4 qui lui permet d'insérer sa distance à toutes les autres destinations, et donc R5 est à jour. Symétriquement, R4 ajoute les destinations R5 et L2 dans sa table, envoie ((R5,1),(L2,2)) à R3 et R7 et ((R5, ∞),(L2, ∞)) à R5. De même, R3 et R7 ajoutent les destinations R5 et L2, et envoient leur distance respectivement à R2 et R6. Si le vecteur de R3 à R2 arrive avant le vecteur de R6 à R2, R2 met directement la bonne distance pour R5 et L2 et envoie le vecteur ((R5,3),(L2,4)) à R1 et R6 et ((R5,∞),(L2,∞)) à R3. Dans le cas contraire, R2 met une première fois à jour sa table et envoie ((R5,4),(L2,5)) à R1 et R3 et ((R5,∞),(L2,∞)) à R6, puis à réception du vecteur de R3 met définitivement à jour sa table. La table de R1 est la même que précédemment avec 2 lignes supplémentaires :

Destination	Prochain saut	distance
R5	R2	4
L2	R2	5

7. Pour simplifier, on ne donnera que le couple concernant la destination R5 (le cas de L2 étant identique). Quand R4 détecte la coupure, il met sa distance à R5 à ∞ et envoie (R5,∞) à R3 et R7, qui propagent vers R2 et R6. Il y a comptage à l'infini par exemple si R2 reçoit (R5,∞) de R3 puis l'ancien vecteur de R6, (R5,3). Dans ce cas R2 met à jour sa distance à R5 (4) et l'envoie à R3 qui l'envoie à R4 => R7 => R6 => R2. Au bout d'un tour, la distance à R5 pour R2 est passée à 9. Après un deuxième tour, la distance vaut 14. Au troisième tour, la distance atteint 16 = ∞ et donc R5 est reconnu comme inaccessible. A chaque tour, R2 envoie une mise à jour à R1 qui recevra successivement (R5,4), puis (R5,9), puis (R5,14), puis (R5,∞). La table de R1 change donc 3 fois.

Partie IV

On souhaite construire un code envoyant des octets de données et pouvant les protéger pour au moins une erreur.

1. Première solution : on construit le code C1 (14,8) en mettant bout à bout deux mots m1, m2 où m1 est calculé avec le code de Hamming (7,4), à partir des 4 premiers bits d'information, et m2 est calculé avec le même code à partir des 4 bits d'information suivants.
   – ce code est-il linéaire ?
   – quelle est sa distance minimale ?
   – ce code peut-il corriger une erreur ? Toute combinaison de 2 erreurs ? Plus que 2 erreurs ?
2. Deuxième solution : on construit le code C2 (12,8) en raccourcissant le code de Hamming (15,11) : on ajoute 3 bits nuls aux 8 bits d'information à envoyer, uniquement pour calculer les 4 bits de contrôle (bien entendu, on n'envoie pas les 3 bits nuls).
   – ce code est-il linéaire ?
   – quelle est la distance minimale de ce code ?
   – ce code peut-il corriger une erreur ? Deux erreurs ? Plus que 2 erreurs ?
3. Quels sont les avantages et inconvénients de ces 2 codes C1 et C2 ?
4. Peut-on construire un code (n,8) correcteur d'une erreur avec n < 12 ? (Indication : compter le nombre de mots de longueur n qui sont à distance 0 ou 1 d'un mot du code).

Corrections :

1. Première solution :
   – Un mot de C1 est de la forme (m1,m2) où m1 et m2 appartiennent au code de Hamming (7,4) donc en particulier (0,0) $\in$ C1, et si (m'1, m'2) $\in$ C1 alors (m1+m'1, m2+m'2) $\in$ C1, et le produit par 0 ou 1 d'un mot de C1 appartient à C1, donc C1 est linéaire. On peut aussi voir que ce code est engendré par la matrice $G1 =$

I_4	A	0	0
0	0	I_4	A

   si $G = (I_4, A)$ est la matrice génératrice de Hamming (7,4).
   – Comme (m1,0) $\in$ C1, le poids minimum de C1 est le même que celui du code de Hamming (7,4) donc 3, et la distance minimale est identique le code étant linéaire.
   – La distance minimale étant 3, ce code peut corriger 1 erreur et ne peut pas corriger toute combinaison de 2 erreurs. Par exemple si 2 erreurs surviennent dans les 7 premiers bits, le code ne peut pas les corriger puisque le code de Hamming (7,4) ne le peut pas. Par contre s'il y a une erreur dans les 7 premiers bits et une autre dans les 7 derniers, ces 2 erreurs peuvent être corrigées. S'il y a au moins 3 erreurs, il y en a donc au moins 2 dans une des deux moitiés, et elles ne peuvent donc pas être corrigées.
2. Deuxième solution :
   – Les mots de C2 sont obtenus en prenant un mot de Hamming (15,11) dont (par exemple) les 3 premiers bits d'informations sont nuls et en supprimant ces 3 bits. Si G = (I11, A) est une matrice génératrice de Hamming (15,11), et si i est un vecteur de 8 bits d'information, alors (0,0,0,i).G = (0,0,0,i, (0,0,0,i).A) et le mot correspondant de C2 est $(i, (0,0,0,i).A) = (i, i.A')$ où A' est obtenu en supprimant les 3 premières lignes de A. En particulier C2 est le code linéaire engendré par $G2 = (I_8, A')$, où G2 est obtenue en supprimant les 3 premières lignes de G, et les 3 premières colonnes. Le code C2 est donc linéaire.

- Tout mot de C2 a le même poids qu'un mot de Hamming (15,11), donc le poids minimal de C2 est au moins celui de Hamming, 3. Comme les lignes de A sont constituées de toutes les combinaisons de 4 bits de poids 2, 3, 4, et qu'il y a 6 lignes de A de poids 2, en enlevant 3 lignes de G, il y a donc au moins 6 ? 3 = 3 lignes de G2 de poids 3, donc le poids minimal du code est au plus 3. Au final le poids minimal, et la distance minimale, sont donc égaux à 3.
- Le code ayant une distance minimale égale à 3 peut corriger 1 erreur, et ne peut pas corriger toute combinaison de 2 erreurs ou plus.

3. Le code C1 a un rendement moins bon que C2 puisque il nécessite 14 bits au lieu de 12. Dans les 2 cas on ne peut pas corriger toute combinaison de 2 erreurs. Par contre C1 permet de corriger 2 erreurs survenant sur 2 quartets différents.
4. Pour pouvoir corriger une erreur, tout mot situé à distance 0 ou 1 d'un mot du code doit être à distance > 1 de tout autre mot du code, les "sphères" de rayon 1 centrées sur les 28 mots du code doivent être toutes disjointes. Chaque sphère contient n+ 1 mots, d'où (n+1) $2^8 \leq 2n$, ou $(n+1) \leq 2^{n-8}$. Cette inégalité n'est pas vraie pour $n < 12$. Donc il n'y a pas de code (n, 8) 1- correcteur de meilleur rendement que C2. Autre raisonnement : dans un code 1-correcteur $(n, 8)$ la matrice de contrôle $(n, n-8)$ doit avoir toutes ses n colonnes distinctes ce qui est impossible pour $n < 12$.