Groupe de travail Réseau
Request for Comments : 4015
Catégorie : En cours de normalisation
Traduction Claude Brière de L'Isle

R. Ludwig, Ericsson Research
A. Gurtov, HIIT

février 2005

Algorithme de réponse Eifel pour TCP

Résumé
Sur la base d'un algorithme de détection approprié, l'algorithme de réponse Eifel donne le moyen à un envoyeur TCP de répondre à une fin de temporisation parasite détectée. Il adapte le temporisateur de retransmission pour éviter d'autres fins de temporisations parasites et (selon l'algorithme de détection) peut éviter les retransmissions souvent inutiles avec retour en arrière à N qu'il serait autrement nécessaire d'envoyer. De plus, l'algorithme de réponse Eifel restaure l'état de contrôle d'encombrement d'une façon telle que les salves de paquets sont évitées.

1. Introduction

L'algorithme de réponse Eifel s'appuie sur un algorithme de détection tel que l'algorithme de détection Eifel, défini dans la [RFC3522]. Le présent document contient des informatives sur les fondements et les motivations du contexte qui peuvent être utiles pour les mises en œuvre de l'algorithme de réponse Eifel, mais il n'est pas nécessaire de lire la [RFC3522] afin de mettre en œuvre l'algorithme de réponse Eifel. Noter que d'autres algorithmes de réponse ont été proposés [BA02] qui pourraient aussi s'appuyer sur l'algorithme de détection Eifel, et d'autres algorithmes de détection ont été proposés [RFC3708], [RFC4138] qui pourraient fonctionner avec l'algorithme de réponse Eifel.

Fondé sur un algorithme de détection approprié, l'algorithme de réponse Eifel donne un moyen pour que l'envoyeur TCP réponde à une fin de temporisation parasite détectée. Il adapte le temporisateur de retransmission pour éviter d'autres fins de temporisations parasites et (selon l'algorithme de détection) peut éviter les retransmissions retour à N souvent inutiles qui seraient envoyées autrement. De plus, l'algorithme de réponse Eifel restaure l'état de contrôle d'encombrement d'une façon telle que les salves de paquets peuvent être évitées.

Note : une version précédente de l'algorithme de réponse Eifel comportait aussi une réponse à une retransmission rapide parasite détectée. Cependant, comme un consensus n'a pas pu être réalisé sur la façon d'adapter le seuil d'accusés de réception dupliqués dans ce cas, cette partie de l'algorithme a été retirée pour l'instant.

1.1 Terminologie

Les mots clés "DOIT", "NE DOIT PAS", "EXIGE", "DEVRA", "NE DEVRA PAS", "DEVRAIT", "NE DEVRAIT PAS", "RECOMMANDE", "PEUT", et "FACULTATIF" en majuscules dans ce document sont à interpréter comme décrit dans le BCP 14, [RFC2119].

On se réfère à la première transmission d'un octet comme à la "transmission originale". Une transmission ultérieure du même octet est appelée une "retransmission". Dans la plupart des cas, cette terminologie peut aussi s'appliquer aux segments de données. Cependant, lorsque une remise en paquets survient, un segment peut contenir à la fois des transmissions originales et des retransmissions d'octets. Dans ce cas, cette terminologie n'est cohérente qu'appliquée aux octets. Pour les algorithmes de détection et de réponse Eifel, il n'y a pas de différence, car ils fonctionnent aussi correctement lorsque survient une remise en paquets.

On utilise le terme "ACK acceptable" comme défini dans la [RFC0793]. C'est un ACK qui accuse réception de données non acquittées précédemment. On utilise le terme "bytes_acked" pour se référer à la quantité (en termes d'octets) de données non acquittées précédemment qui sont acquittées par l'ACK acceptable le plus récemment reçu. On utilise les variables d'état d'envoyeur TCP 'SND.UNA' et 'SND.NXT' comme défini dans la [RFC0793]. SND.UNA contient le numéro de

séquence de segment du plus vieux segment en instance. SND.NXT contient le numéro de séquence de segment du prochain segment que l'envoyeur TCP va (re-)transmettre. De plus, on définit comme 'SND.MAX' le numéro de séquence de segment de la prochaine transmission originale à envoyer. La définition de SND.MAX est équivalente à la définition de "snd_max" dans [WS95].

On utilise les variables d'état d'envoyeur TCP "cwnd" (fenêtre d'encombrement), et "ssthresh" (seuil de démarrage lent), et le terme "FlightSize" comme défini dans la [RFC2581]. FlightSize est la quantité (en octets) de données en instance à un instant donné. On utilise le terme "fenêtre initiale" (IW, *Initial Window*) comme défini dans la [RFC3390]. La IW est la taille de la fenêtre d'encombrement de l'envoyeur après l'achèvement de la prise de contact en trois phases. On utilise les variables d'état d'envoyeur TCP "SRTT" et "RTTVAR", et les termes "RTO" et "G" comme défini dans la [RFC2988]. G est la granularité d'horloge du temporisateur de retransmission. De plus, on suppose que l'envoyeur TCP conserve la valeur de la dernière mesure de temps d'aller retour (RTT, *round-trip time*) dans la variable (locale) "RTT-SAMPLE".

On utilise la variable d'état d'envoyeur TCP "T_last", et le terme "tcpnow" comme ils sont utilisés dans la [RFC2861]. T_last contient l'heure système lorsque l'envoyeur TCP envoie le dernier segment de données, tandis que "tcpnow" est l'heure système courante de l'envoyeur TCP.

2. Algorithmes de détection appropriés

Si l'algorithme de réponse Eifel est mis en œuvre chez l'envoyeur TCP, il DOIT être mis en œuvre avec un algorithme de détection qui est spécifié dans une RFC sur la voie de la normalisation ou expérimentale.

Les concepteurs d'algorithmes de détection qui veulent que leurs algorithmes fonctionnent avec l'algorithme de réponse Eifel devraient réutiliser la variable "SpuriousRecovery" avec la sémantique et les valeurs définies spécifiée dans la [RFC3522]. De plus, on définit la constante LATE_SPUR_TO (réglée égale à -1) comme une autre valeur possible de la variable SpuriousRecovery. Les algorithmes de détection devraient régler la valeur de SpuriousRecovery à LATE_SPUR_TO si la détection d'une retransmission parasite se fonde sur le ACK pour la retransmission (par opposition à un ACK pour une transmission originale). Par exemple, ceci s'applique aux algorithmes de détection qui se fondent sur l'option DSACK [RFC3708].

3. Algorithme de réponse Eifel

L'algorithme complet est spécifié au paragraphe 3.1. Dans les paragraphes 3.2 à 3.6, on expose les différentes étapes de l'algorithme.

3.1 L'algorithme

Soit un envoyeur TCP qui a activé un algorithme de détection conforme aux exigences de la Section 2, un envoyeur TCP PEUT utiliser l'algorithme de réponse Eifel comme défini dans ce paragraphe.

Si l'algorithme de réponse Eifel est utilisé, les étapes suivantes DOIVENT être suivies par l'envoyeur TCP, mais seulement après l'initialisation d'une récupération de perte fondée sur une temporisation. C'est-à-dire lorsque la première retransmission fondée sur la temporisation est envoyée. L'algorithme NE DOIT PAS être réinitialisé après qu'une récupération de perte fondée sur la temporisation a déjà été lancée mais pas achevée. En particulier, il ne doit pas être réinitialisé suite à des fins de temporisation ultérieures pour le même segment, ou à la retransmission de segments autres que le plus ancien segment en instance.

(0) Avant que les variables cwnd et ssthresh soient mises à jour lors de l'initialisation de la récupération de pertes, régler une variable "pipe_prev" comme suit :
pipe_prev <- max (FlightSize, ssthresh)

Régler une variable "SRTT_prev" et une variable "RTTVAR_prev" comme suit :
SRTT_prev <- SRTT + (2 * G)
RTTVAR_prev <- RTTVAR

(DET) C'est un fourre-tout pour un algorithme de détection qui doit être exécuté à ce moment, et qui règle la variable SpuriousRecovery comme indiqué à la Section 2. Si la [RFC3522] est utilisée comme algorithme de détection, les étapes (1) à (6) de cet algorithme viennent ici.

(7) Si SpuriousRecovery est égal à SPUR_TO, passer à l'étape (8) ;
autrement, si SpuriousRecovery est égal à LATE_SPUR_TO, passer à l'étape (9) ;
autrement, passer à l'étape (FIN).

(8) Reprendre la transmission avec les données non envoyées précédemment :
Régler SND.NXT <- SND.MAX

(9) Inverser l'état de contrôle d'encombrement :
Si le ACK acceptable a le fanion ECN-Echo [RFC3168] établi,
alors passer à l'étape (FIN) ;
autrement, régler :
cwnd <- FlightSize + min (bytes_acked, IW)
ssthresh <- pipe_prev
Passer à l'étape (FIN).

(10) Interfonctionnement avec la validation de fenêtre d'encombrement :
Si la validation de fenêtre d'encombrement est mise en œuvre selon la [RFC2861], alors régler T_last <- tcpnow

(11) Adapter la durée du temporisateur de retransmission :
Au premier RTT-SAMPLE pris à partir des nouvelles données, c'est-à-dire, le premier RTT-SAMPLE qui peut être déduit d'un ACK acceptable pour les données qui n'étaient pas envoyées auparavant lorsque la fin de temporisation parasite s'est produite,
si le temporisateur de retransmission est mis en œuvre conformément à la [RFC2988], alors régler :
SRTT <- max (SRTT_prev, RTT-SAMPLE)
RTTVAR <- max (RTTVAR_prev, RTT-SAMPLE/2)
RTO <- SRTT + max (G, 4*RTTVAR)
Effectuer la vérification des limites sur le RTO (règles (2.4) et (2.5) dans la [RFC2988]), et redémarrer le temporisateur de retransmission;
autrement, adapter de façon appropriée la durée du temporisateur de retransmission qui est mis en œuvre.

(FIN) Pas d'autre traitement.

3.2 Mémorisation de l'état de contrôle d'encombrement actuel (étape 0)

L'envoyeur TCP mémorise dans pipe_prev ce qui est considéré comme un seuil sûr de démarrage lent (ssthresh) avant d'initialiser la récupération de pertes ; c'est-à-dire, avant que l'indication de perte ne soit prise en compte. C'est soit la FlightSize en cours, si l'envoyeur TCP est en évitement d'encombrement, soit le ssthresh actuel, si l'envoyeur TCP est en démarrage lent. Si l'envoyeur TCP détecte plus tard qu'il est entré inutilement dans la récupération de pertes, pipe_prev est alors utilisé dans l'étape (9) pour inverser l'état de contrôle d'encombrement. Donc, jusqu'à ce que la phase de récupération de pertes soit terminée, pipe_prev conserve en mémoire l'état de contrôle d'encombrement de l'instant juste avant que la phase de récupération de pertes soit initialisée. Une approche similaire est proposée dans la [RFC2861], où cet état est mémorisé directement dans ssthresh après qu'un envoyeur TCP est devenu inactif ou d'application limitée.

Il y a eu des débats autour de la question de savoir si la valeur de pipe_prev devrait être diminuée au fil du temps ; par exemple, lors de fins de temporisations successives pour le même segment en instance. On n'exige pas de diminution de pipe_prev pour l'algorithme de réponse Eifel et on ne pense pas qu'une approche aussi prudente devrait être suivie. On suit plutôt l'idée de revalidation de la fenêtre d'encombrement au travers du démarrage lent, comme suggéré dans la [RFC2861]. C'est-à-dire que, dans l'étape (9), le cwnd est remis à une valeur qui évite de grosses salves de paquets, et ssthresh est remis à la valeur de pipe_prev. Noter que les [RFC2581] et [RFC2861] n'exigent pas non plus de diminution de ssthresh après qu'il a été rétabli en réponse à une indication de perte, ou après qu'un envoyeur TCP est devenu inactif ou d'application limitée.

3.3 Suppression des retransmissions retour à N inutiles (étape 8)

Sans l'utilisation de l'option d'horodatage TCP [RFC1323], l'envoyeur TCP souffre du problème de l'ambiguïté de la retransmission [Zh86], [KP87]. Donc, lorsque arrive le premier ACK acceptable après une fin de temporisation parasite, l'envoyeur TCP doit supposer que cet ACK a été envoyé en réponse à la retransmission alors qu'en fait il a été envoyé en réponse à une transmission originale. De plus, l'envoyeur TCP doit encore supposer que tous les autres segments qui étaient en instance à ce moment ont été perdus.

Note : sauf dans certains cas où les ACK d'origine ont été perdus, le premier ACK acceptable ne peut pas porter l'option DSACK [RFC2883].

Par conséquent, une fois que l'état de l'envoyeur TCP a été mis à jour après l'arrivée du premier ACK acceptable, SND.NXT est égal à SND.UNA. C'est ce qui cause les retransmissions retour à N souvent inutiles. À partir de ce moment chaque ACK acceptable arrivant qui a été envoyé en réponse à une transmission originale va avancer SND.NXT. Mais tant que SND.NXT est inférieur à la valeur que SND.MAX avait lorsque s'est produite la fin de temporisation, ces ACK vont concerner les retransmissions, que les transmissions originales correspondantes aient été perdues ou non.
En fait, durant cette phase, l'envoyeur TCP casse la "conservation de paquet" [Jac88]. C'est parce que les retransmissions retour à N sont envoyées durant le démarrage lent. Pour chaque transmission originale qui quitte le réseau, deux retransmissions sont envoyées dans le réseau tant que SND.NXT n'est pas égal à SND.MAX (voir les détails dans [LK00]).

Une fois qu'une fin de temporisation parasite a été détectée (à réception d'un ACK pour une transmission originale) il est plus sûr de laisser l'envoyeur TCP reprendre la transmission avec des données non envoyées précédemment. Donc, l'algorithme de réponse Eifel change l'état de l'envoyeur TCP en réglant SND.NXT à SND.MAX. Noter que cette étape n'est exécutée que si la variable SpuriousRecovery est égale à SPUR_TO, qui à son tour exige qu'un algorithme de détection tel que l'algorithme de détection Eifel [RFC3522] ou l'algorithme F-RTO [RFC4138] détecte une retransmission parasite sur la base de la réception d'un ACK pour une transmission originale (par opposition à l'ACK pour la retransmission [RFC3708]).

3.4 Inversion de l'état du contrôle d'encombrement (étape 9)

Lorsque un envoyeur TCP entre dans la récupération de pertes, il réduit cwnd et ssthresh. Cependant, une fois que l'envoyeur TCP détecte que la récupération de pertes a été déclanchée à tort, cette réduction se révèle inutile. On estime donc qu'il est sûr de revenir à l'état de contrôle d'encombrement précédent, suivant l'approche de revalidation de la fenêtre d'encombrement comme exposé ci-dessous. Ceci sauf si l'ACK acceptable signale l'encombrement par le fanion ECN-Echo [RFC3168]. Dans ce cas, l'envoyeur TCP DOIT s'interdire d'inverser l'état de contrôle d'encombrement.

Si le fanion ECN-Echo n'est pas établi, cwnd est remis à la somme de la FlightSize courante et du minimum de bytes_acked et IW. Dans certains cas, cela peut signifier que les premiers ACK acceptables qui arrivent ne vont pas déclancher de segments de données. On se rappelle que bytes_acked est le nombre d'octets qui ont été acquittés par l'ACK acceptable. Noter que la valeur de cwnd ne doit plus être changée pour cet ACK, et que la valeur de FlightSize à ce moment peut être différente de la valeur de FlightSize à l'étape (0). La valeur de IW met une limite à la taille de la salve de paquets que l'envoyeur TCP peut envoyer dans le réseau après la fin de l'algorithme de réponse Eifel. La valeur de IW est considérée comme une taille de salve acceptable. C'est la quantité de données qu'un envoyeur TCP peut envoyer dans un réseau non encore "sondé" au début d'une connexion.

Alors ssthresh est remis à la valeur de pipe_prev. Par suite, l'envoyeur TCP soit reprend immédiatement le sondage du réseau pour plus de bande passante dans l'évitement d'encombrement, soit il fait un démarrage lent de ce qui est considéré comme un point de fonctionnement sûr pour la fenêtre d'encombrement.

3.5 Interfonctionnement avec l'algorithme CWV (étape 10)

Une mise en œuvre de l'algorithme de validation de fenêtre d'encombrement (CWV, *Congestion Window Validation*) [RFC2861] pourrait éventuellement mal interpréter une pointe de retard qui a causé une fin de temporisation parasite comme une phase où l'envoyeur TCP a été inactif. Donc, T_last est rétabli pour empêcher le déclanchement de l'algorithme CWV dans ce cas.

Note : le terme "inactif" implique que l'envoyeur TCP n'a pas de données en instance; c'est-à-dire, toutes les données envoyées ont été acquittées [Jac88]. Selon cette définition, un envoyeur TCP n'est pas inactif lorsque il attend un ACK acceptable après une fin de temporisation. Malheureusement, le pseudo code de la [RFC2861] ne comporte pas de vérification de la condition "inactif" (SND.UNA == SND.MAX). On doit donc ajouter l'étape (10) à l'algorithme de réponse Eifel.

3.6 Adapter le temporisateur de retransmission (étape 11)

Il n'y a actuellement qu'un seul temporisateur de retransmission normalisé pour TCP [RFC2988]. On ne vise donc explicitement que ce temporisateur. De futures normes qui pourraient définir des solutions de remplacement pour la [RFC2988] devraient proposer des mesures similaires pour adapter la durée du temporisateur de retransmission.

Une fin de temporisation parasite résulte souvent d'une pointe de retard, qui est une soudaine augmentation du RTT qui ne

peut généralement pas être prédite. Après une pointe de retard, le RTT peut avoir changé de façon permanente ; par exemple, du à un changement de chemin, ou parce que la bande passante disponible a diminué sur un chemin dominé par la bande passante. Ceci peut souvent se produire sur des liaisons d'accès sans fil de large zone. Dans ce cas, les estimateurs du RTT (SRTT et RTTVAR) devraient être réinitialisés à partir du premier RTT-SAMPLE tiré des nouvelles données conformément à la règle (2.2) de la [RFC2988]. C'est-à-dire, depuis le premier RTT-SAMPLE qui peut être déduit comme un ACK acceptable pour les données qui n'ont pas été envoyées antérieurement lorsque la fin de temporisation parasite s'est produite.

Cependant, une pointe de retard peut n'indiquer qu'une phase transitoire, après laquelle le RTT revient à sa gamme de valeurs antérieure, ou même à de plus petites valeurs. Aussi, une fin de temporisation parasite peut se produire parce que l'estimateur de RTT de l'envoyeur TCP était assez imprécis pour que le temporisateur de retransmission arrive à expiration "un petit peu trop tôt". On estime que deux fois la granularité d'horloge du temporisateur de retransmission (2 * G) est une limite supérieure raisonnable du "un petit peu trop tôt". Donc, lorsque le nouveau RTO est calculé dans l'étape (11), on s'assure qu'il est d'au moins (2 * G) supérieur (voir aussi l'étape (0)) à ce que le RTO était avant qu'arrive la fin de temporisation parasite.

Noter qu'un autre traitement d'envoyeur TCP va normalement avoir lieu avant les étapes (10) et (11). Durant cette phase (c'est-à-dire, avant que l'étape (11) ait été atteinte) le RTO est géré selon les règles de la [RFC2988]. On estime que ceci est suffisamment prudent pour les raisons suivantes. D'abord, le temporisateur de retransmission est relancé sur l'ACK acceptable qui a été utilisé pour détecter la fin de temporisation parasite. Par suite, la pointe de retard est déjà implicitement mise en facteurs pour les segments en instance à ce moment. Ceci est exposé plus en détails dans [EL04], où cet effet est appelé le "décalage de RTO". De plus, si les horodatages sont activés, un nouveau RTT-SAMPLE valide peut être déduit de cet ACK acceptable. Ce RTT-SAMPLE doit être relativement grand, car il inclut la pointe de retard qui a causé la fin de temporisation parasite. Par conséquent, les estimateurs de RTO vont être mis à jour de façon assez prudente. Sans horodatage, le RTO va être retardé de façon prudente du fait de l'algorithme de Karn [RFC2988] jusqu'à ce que le premier RTT-SAMPLE puisse être déduit d'un ACK acceptable pour les données qui n'ont pas été envoyées précédemment lorsque la fin de temporisation parasite s'est produite.

Pour que le nouveau RTO devienne effectif, le temporisateur de retransmission doit être relancé. Ceci est cohérent avec la [RFC2988], qui recommande de relancer le temporisateur de retransmission à l'arrivée d'un ACK acceptable.

4. La récupération de pertes évoluée est cruciale pour l'algorithme de réponse Eifel

On a étudié des environnements où les fins de temporisations parasites et de multiples pertes provenant de la même portée de paquets coïncident souvent [GL02], [GL03]. Dans un tel cas, le plus vieux segment en instance arrive au receveur TCP, mais un ou plusieurs paquets de ce qui reste de la portée en instance sont perdus. Dans ces environnements, les performances de bout en bout souffrent si l'algorithme de réponse Eifel est utilisé sans un schéma évolué de récupération de pertes comme un schéma fondé sur le SACK [RFC3517] ou NewReno [RFC3782]. La raison en est l'agressivité du Reno TCP après une fin de temporisation parasite. Même si Reno TCP casse la "conservation des paquets" (voir au paragraphe 3.3) lorsque il retransmet aveuglément tous les segments en instance, il récupère généralement tous les paquets perdus d'une portée dans un seul délai d'aller retour. Au contraire, le plus prudent Reno TCP avec Eifel est souvent forcé de subir une autre fin de temporisation. Donc, on recommande que l'algorithme de réponse Eifel fonctionne toujours en combinaison avec la [RFC3517] ou la [RFC3782]. Une robustesse supplémentaire est réalisée avec les algorithmes de transmission limitée et de retransmission précoce [RFC3042], [AAAB04].

Note: Le schéma fondé sur le SACK qu'on a utilisé pour nos simulations dans [GL02] et [GL03] est différent du schéma fondé sur SACK qui a ensuite été normalisé dans la [RFC3517]. La différence clé est que la [RFC3517] est plus robuste à des pertes multiples sur la même portée. Il est moins prudent en déclarant qu'un paquet a quitté le réseau, et dépend donc moins des fins de temporisations pour récupérer d'authentiques pertes de paquets.

Si l'algorithme NewReno [RFC3782] est utilisé en combinaison avec l'algorithme de réponse Eifel, l'étape (1) de l'algorithme NewReno DEVRAIT être modifiée comme suit, mais seulement si SpuriousRecovery égale SPUR_TO :

(1) Trois ACK dupliqués :
Lorsque le troisième ACK dupliqué est reçu et que l'envoyeur n'est pas déjà dans la procédure de récupération rapide, passer à l'étape 1A.

C'est-à-dire que l'étape 1B entière de l'algorithme NewReno est obsolète parce que l'étape (8) de l'algorithme de réponse Eifel évite le cas où trois ACK dupliqués résultent de retransmissions retour à N inutiles après une fin de temporisation. L'étape (8) de l'algorithme de réponse Eifel évite tout d'abord de telles retransmissions retour à N inutiles. Cependant, on se rappelle que l'étape (8) n'est exécutée que si la variable SpuriousRecovery est égale à SPUR_TO, ce qui à son tour exige un

algorithme de détection, comme l'algorithme de détection Eifel [RFC3522] ou l'algorithme F-RTO [RFC4138], qui détecte une retransmission parasite sur la base de la réception d'un ACK pour une transmission d'origine (par opposition à l'ACK pour la retransmission [RFC3708]).

5. Considérations sur la sécurité

Il existe un risque qu'un algorithme de détection soit perturbé par des ACK usurpés qui feraient apparaître d'authentiques retransmissions à l'envoyeur TCP comme des retransmissions parasites. Lorsque un tel algorithme de détection fonctionne conjointement avec l'algorithme de réponse Eifel, cela pourrait effectivement désactiver le contrôle d'encombrement chez l'envoyeur TCP. Si cela devait devenir un problème, l'algorithme de réponse Eifel DEVRAIT ne fonctionner qu'avec des algorithmes de détection qui sont connus pour être sûrs contre de telles "attaques par usurpation d'ACK".

Par exemple, la variante sûre de l'algorithme de détection Eifel [RFC3522], est une méthode fiable de protection contre ce risque.

6. Remerciements

Tous nos remerciements à Keith Sklower, Randy Katz, Michael Meyer, Stephan Baucke, Sally Floyd, Vern Paxson, Mark Allman, Ethan Blanton, Pasi Sarolahti, Alexey Kuznetsov, et Yogesh Swami pour les nombreuses discussions qui ont contribué au présent travail.

7. Références

7.1 Références normatives

[RFC0793] J. Postel (éd.), "Protocole de commande de transmission – Spécification du protocole du programme Internet DARPA", STD 7, septembre 1981.

[RFC2119] S. Bradner, "Mots clés à utiliser dans les RFC pour indiquer les niveaux d'exigence", BCP 14, mars 1997.

[RFC2581] M. Alman, V. Paxson et W. Stevens, "Contrôle d'encombrement avec TCP", avril 1999. *(Obsolète, voir* RFC5681)

[RFC2861] M. Handley, J. Padhye, S. Floyd, "Validation de fenêtre d'encombrement TCP", juin 2000. *(Expérimentale)*

[RFC2988] V. Paxson, M. Allman, "Calcul du temporisateur de retransmission de TCP", novembre 2000. *(P.S.)(Obs., voir* RFC6298)

[RFC3168] K. Ramakrishnan et autres, "Ajout de la notification explicite d'encombrement (ECN) à IP", septembre 2001. *(P.S.)*

[RFC3390] M. Allman, S. Floyd, C. Partridge, "Augmentation de la fenêtre initiale de TCP", octobre 2002. *(P.S.)*

[RFC3522] R. Ludwig, M. Meyer, "Algorithme Eifel de détection pour TCP", avril 2003. *(Expérimentale)*

[RFC3782] S. Floyd, T. Henderson, A. Gurtov, "Modification NewReno à l'algorithme de récupération rapide de TCP", avril 2004. *(P.S.) (Obs., voir* RFC6582)

7.2 Références pour information

[AAAB04] Allman, M., Avrachenkov, K., Ayesta, U., and J. Blanton, "Early Retransmit for TCP and SCTP", Travail en cours, juillet 2004.

[BA02] Blanton, E. and M. Allman, "On Making TCP More Robust to Packet Reordering", ACM Computer Communication Review, Vol. 32, n° 1, janvier 2002.

[EL04] Ekstrom, H. and R. Ludwig, The Peak-Hopper: A New End-to-End Retransmission Timer for Reliable Unicast Transport", dans Proceedings of IEEE INFOCOM 04, mars 2004.

[GL02] Gurtov, A. and R. Ludwig, "Evaluating the Eifel Algorithm for TCP in a GPRS Network, In Proceedings of the European Wireless Conference, février 2002.

[GL03] Gurtov, A. and R. Ludwig, "Responding to Spurious Timeouts in TCP", dans Proceedings of IEEE INFOCOM 03, avril 2003.

[Jac88] Jacobson, V., Congestion Avoidance and Control, In Proceedings of ACM SIGCOMM 88.

[KP87] Karn, P. and C. Partridge, "Improving Round-Trip Time Estimates in Reliable Transport Protocols", dans Proceedings of ACM SIGCOMM 87.

[LK00] Ludwig, R. and R. H. Katz, "The Eifel Algorithm: Making TCP Robust Against Spurious Retransmissions", ACM Computer Communication Review, Vol. 30, n° 1, janvier 2000.

[RFC1323] V. Jacobson, R. Braden et D. Borman, "Extensions TCP pour de bonnes performances", mai 1992.

[RFC2883] S. Floyd et autres, "Extension à l'option d'accusé de réception sélectif (SACK) pour TCP", juillet 2000. *(P.S.)*

[RFC3042] M. Allman, H. Balakrishnan, S. Floyd, "Amélioration de la récupération de perte dans TCP avec la transmission limitée", janvier 2001. *(P.S.)*

[RFC3517] E. Blanton et autres, "Algorithme de récupération de perte fondé sur l'accusé de réception sélectif prudent (SACK) pour TCP", avril 2003. (*Remplacée par* RFC6675) *(P.S.)*

[RFC3708] E. Blanton, M. Allman, "Utilisation d'accusés de réception sélectifs dupliqués (DSACK) de TCP et de numéros de séquence de transmission dupliqués (TSN) du protocole de transmission de contrôle de flux (SCTP) pour détecter les retransmissions parasites", février 2004. *(Expérimentale)*

[RFC4138] P. Sarolahti, M. Kojo, "Récupération directe de RTO (F-RTO) : un algorithme pour la détection des fins de temporisation de retransmission parasites avec TCP et le protocole de transmission de contrôle de flux (SCTP)", août 2005. *(Expérimentale)*

[WS95] Wright, G. R. and W. R. Stevens, TCP/IP Illustrated, Volume 2 (The Implementation), Addison Wesley, janvier 1995.

[Zh86] Zhang, L., "Why TCP Timers Don't Work Well", dans Proceedings of ACM SIGCOMM 88.

Adresse des auteurs

Reiner Ludwig
Ericsson Research (EDD)
Ericsson Allee 1
52134 Herzogenrath, Germany
mél : Reiner.Ludwig@ericsson.com

Andrei Gurtov
Helsinki Institute for Information Technology (HIIT)
P.O. Box 9800, FIN-02015
HUT, Finland
mél : andrei.gurtov@cs.helsinki.fi
http://www.cs.helsinki.fi/u/gurtov

Déclaration complète de droits de reproduction

Remerciement
Le financement de la fonction d'édition des RFC est actuellement fourni par l'Internet Society.