Équipe d'ingénierie de l'Internet (IETF)
Request for Comments : 6675
RFC rendue obsolète : 3517
Catégorie : En cours de normalisation
ISSN : 2070-1721
Traduction Claude Brière de L'Isle

E. Blanton, Purdue University
M. Allman, ICSI
L. Wang, Juniper Networks
I. Jarvinen & M. Kojo, University of Helsinki
Y. Nishida, WIDE Project
août 2012

Algorithme prudent de récupération de pertes fondé sur un accusé de réception sélectif (SACK) pour TCP

Résumé
Le présent document présente un algorithme prudent de récupération de pertes pour TCP qui se fonde sur l'utilisation de l'option TCP d'accusé de réception sélectif (SACK). L'algorithme présenté dans ce document se conforme à l'esprit de la spécification actuelle de contrôle d'encombrement (RFC 5681), mais permet aux envoyeurs TCP de récupérer plus efficacement lorsque plusieurs segments sont perdus à partir d'un seul envoi de données. Le présent document rend obsolète la RFC 3517 et décrit les changements par rapport à elle.

1. Introduction

Le présent document présente un algorithme prudent de récupération de perte pour TCP qui est fondé sur l'utilisation de l'option TCP d'accusé de réception sélectif (SACK, *Selective Acknowledgment*). Bien que l'option TCP SACK [RFC2018] soit fortement déployée dans l'Internet [All00], il a été prouvé que les hôtes n'utilisent pas les informations de SACK lorsque ils prennent des décisions de retransmission et de contrôle d'encombrement [PF01]. Le but de ce document est d'exposer une méthode directe pour que les mises en œuvre de TCP utilisent les informations de SACK pour augmenter les performances.

La [RFC5681] permet à TCP [RFC0793] d'utiliser des algorithmes évolués de récupération de perte pourvu qu'ils suivent l'esprit des algorithmes de contrôle d'encombrement de TCP [RFC2914], [RFC5681]. La [RFC6582] expose un de ces algorithmes évolués de récupération appelé NewReno. Le présent document expose un algorithme de récupération de perte qui utilise l'option SACK de TCP [RFC2018] pour améliorer la récupération de perte de TCP. L'algorithme exposé dans ce document, largement fondé sur l'algorithme décrit dans [FF96], est un remplaçant prudent de l'algorithme de récupération rapide [Jac90], [RFC5681]. L'algorithme spécifié dans le présent document est une stratégie directe de récupération de perte fondée sur SACK qui suit les lignes directrices établies dans la [RFC5681] et peut être utilisé en toute sécurité dans les mises en œuvre de TCP. D'autres méthodes de récupération de perte fondées sur SACK peuvent être utilisées dans TCP selon que les mises en œuvre le jugent approprié (pour autant que l'autre algorithme suive les lignes directrices fournies dans la [RFC5681]). Prière de noter cependant que dans le présent document, les décisions fondées sur SACK (comme de décider quel segment envoyer à quel instant) sont largement découplées de l'algorithme de contrôle d'encombrement, et à ce titre peuvent être traitées comme des questions distinctes si on le souhaite.

Le présent document représente une révision de la [RFC3517] pour traiter de plusieurs situations qui n'y sont pas traitées explicitement. Un résumé des changements entre le présent document et la [RFC3517] se trouve à la Section 9.

2. Définitions

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

Le lecteur est supposé familier des définitions données dans la [RFC5681], ainsi qu'avec les accusés de réception sélectifs comme spécifié dans la [RFC2018].

Pour les besoins de l'explication de l'algorithme de récupération de perte fondé sur le SACK, on définit six variables que mémorise un envoyeur TCP :

- "HighACK" est le numéro de séquence du plus fort octet de données acquitté cumulativement à un point donné.

- "HighData" est le plus fort numéro de séquence transmis à un point donné.

- "HighRxt" est le plus fort numéro de séquence qui a été retransmis durant la phase de récupération de perte en cours.

- "RescueRxt" est le plus fort numéro de séquence qui a été optimistiquement retransmis pour empêcher le calage de l'horloge de ACK lorsque il y a de la perte à la fin de la fenêtre et qu'aucune nouvelle donnée n'est disponible à la transmission.

- "Pipe" est une estimation par l'envoyeur du nombre d'octets en cours dans le réseau. C'est utilisé durant la récupération pour limiter le taux d'envoi de l'envoyeur. La variable pipe permet à TCP d'utiliser un contrôle d'encombrement fondamentalement différent de celui de l'algorithme spécifié dans la [RFC5681]. L'algorithme de contrôle d'encombrement qui utilise l'estimation de pipe est souvent désigné comme "algorithme pipe".

- "DupAcks" est le nombre d'accusés de réception dupliqués reçus depuis le dernier accusé de réception cumulatif.

Pour les besoins de la présente spécification, on définit un "accusé de réception dupliqué" comme un segment qui arrive en portant un bloc SACK qui identifie les octets non acquittés précédemment et non objets de SACK entre HighACK et HighData. Noter qu'un ACK qui porte de nouvelles données SACK est compté comme un accusé de réception dupliqué selon cette définition même si il porte de nouvelles données, change la fenêtre annoncée, ou déplace le point d'accusé de réception cumulatif, ce qui est différent de la définition de l'accusé de réception dupliqué dans la [RFC5681].

On définit une variable "DupThresh" qui contient le nombre d'accusé de réception dupliqués requis pour déclencher une retransmission. Selon la [RFC5681], ce seuil est définit à trois accusés de réception dupliqués. Cependant, les mises en œuvre devraient consulter toutes les mises à jour de la [RFC5681] pour déterminer la valeur actuelle de DupThresh (ou méthode pour déterminer sa valeur).

Finalement, une gamme de numéros de séquence [A,B] est dite "couvrir" le numéro de séquence S si $A \leq S \leq B$.

3. Conservation des informations SACK

Pour qu'un envoyeur TCP mette en œuvre l'algorithme défini à la section suivante, il doit garder une structure de données pour mémoriser les informations entrantes d'accusé de réception sélectif connexion par connexion. Une telle structure de données est couramment appelée un "tableau de résultats". Les spécificités de la structure de données dite tableau de résultats sortent du domaine d'application du présent document (pour autant que la mise en œuvre puisse effectuer toutes les fonctions requises par la présente spécification).

Noter que ce document se réfère à la prise en compte (marquage) des octets individuels de données transférés à travers une connexion TCP. Une mise en œuvre physique du tableau de résultats préférerait vraisemblablement gérer ces données comme des gammes de numéros de séquence. Les algorithmes présentés ici le permettent, mais exigent la capacité à marquer des gammes de numéro de séquence arbitraires comme ayant été acquittées sélectivement.

Finalement, noter que l'algorithme du présent document suppose un envoyeur qui ne garde pas trace des frontières de segment après la transmission du segment. Il est possible qu'il y ait des algorithmes plus raffinés et plus précis que

l'algorithme présenté ici, qui conservent cet état supplémentaire, à la disposition d'un envoyeur ; cependant, cela fera l'objet d'un travail futur.

4. Traitement des informations SACK et actions à entreprendre

La présente section décrit une structure spécifique et le flux de commandes pour la mise en œuvre du comportement TCP décrit dans la présente norme. Leur comportement est ce qui est normalisé, et cette collection particulière de fonctions est le moyen fortement recommandé de mettre en œuvre ce comportement, bien que d'autres approches soient possibles pour réaliser ce comportement.

La définition de la taille maximum de segment d'envoyeur (SMSS, *Sender Maximum Segment Size*) utilisée dans cette section est fournie dans la [RFC5681].

Pour les besoins de l'algorithme défini dans le présent document, le tableau de résultats DEVRAIT mettre en œuvre les fonctions suivantes :

Update () :
Étant données les informations fournies dans un ACK, chaque octet qui est cumulativement acquitté ou sélectivement acquitté devrait être marqué en conséquence dans le tableau de résultats, et le nombre total d'octets acquittés sélectivement devrait être enregistré.

Note : Les informations de SACK sont facultatives et donc les données acquittées sélectivement NE DOIVENT PAS être retirées de la mémoire tampon de retransmission de TCP jusqu'à ce que les données soient acquittées cumulativement [RFC2018].

IsLost (SeqNum) :
Ce sous-programme indique si le numéro de séquence en cause est considéré comme perdu. Le sous-programme retourne "vrai" quand DupThresh séquences acquittées sélectivement discontiguës sont arrivées au dessus de 'SeqNum' ou que plus de (DupThresh - 1) * SMSS octets avec des numéros de séquence supérieurs à 'SeqNum' ont été acquittés sélectivement. Autrement, le sous-programme retourne "faux".

SetPipe ():
Ce sous-programme traverse l'espace de numéros de séquence de HighACK à HighData et DOIT régler la variable "pipe" à une estimation du nombre d'octets qui sont actuellement en transit entre l'envoyeur TCP et le receveur TCP. Après l'initialisation de pipe à zéro, les étapes suivantes sont suivies pour chaque octet 'S1' dans l'espace de numéros de séquence entre HighACK et HighData qui n'a pas été acquitté sélectivement :
(a) Si IsLost (S1) retourne faux :
    Pipe est incrémenté de 1 octet.
    L'effet de cette condition est que pipe est incrémenté pour les paquets qui n'ont pas été acquittés sélectivement et n'ont pas été déterminés comme perdus (c'est-à-dire, les segments qui sont supposés être encore dans le réseau).
(b) Si S1 ≤ HighRxt :
    Pipe est incrémenté de 1 octet.
    L'effet de cette condition est que pipe est incrémenté pour la retransmission de l'octet.

Noter que les octets retransmis sans être considérés perdus sont comptés deux fois par le mécanisme ci-dessus.

NextSeg () :
Ce sous-programme utilise la structure de données de tableau de résultats entretenu par la fonction Update () pour déterminer quoi transmettre sur la base des informations de SACK qui sont arrivées du receveur des données (et ont donc été marquées dans le tableau de résultats). NextSeg () DOIT retourner la gamme de numéros de séquence du prochain segment qui est à transmettre, selon les règles suivantes :
(1) Si il existe un plus petit numéro de séquence 'S2' non acquitté sélectivement qui satisfait aux trois critères suivants pour déterminer une perte, la gamme de numéros de séquence d'un segment de jusqu'à SMSS octets commençant par S2 DOIT être retournée.
    (1.a) S2 est supérieur à HighRxt.
    (1.b) S2 est inférieur au plus fort octet couvert par un SACK reçu.
    (1.c) IsLost (S2) retourne vrai.
(2) Si aucun numéro de séquence 'S2' n'existe selon la règle (1) mais s'il existe des données non envoyées disponibles et si la fenêtre annoncée du receveur le permet, la gamme de numéros de séquence d'un segment de jusqu'à SMSS octets de données non envoyées précédemment commençant par le numéro de séquence HighData+1 DOIT être retourné.
(3) Si les conditions des règles (1) et (2) échouent, mais si il existe un numéro de séquence non acquitté sélectivement 'S3'

qui satisfait aux critères de détection de perte donnés aux étapes (1.a) et (1.b) ci-dessus (excluant spécifiquement l'étape (1.c)) alors un segment de jusqu'à SMSS octets commençant par S3 DEVRAIT être retourné.

(4) Si les conditions pour (1), (2), et (3) échouent, mais si il existe des données non acquittées sélectivement en cours, on assure l'opportunité d'une seule retransmission "de secours" par l'entrée dans la récupération de perte. Si HighACK est supérieur à RescueRxt (ou si RescueRxt est indéfini) alors un segment de jusqu'à SMSS octets qui DOIT inclure le plus haut numéro de séquence en cours non acquitté sélectivement DEVRAIT être retourné, et RescueRxt être réglé à RecoveryPoint. HighRxt NE DOIT PAS être mis à jour.

Note : Les règles (3) et (4) sont une sorte de retransmission "de dernière instance". Elles permettent la retransmission des numéros de séquence même lorsque l'envoyeur a moins de certitude de la perte d'un segment qu'avec la règle (1). La retransmission de segments via les règles (3) et (4) va aider à entretenir l'horloge d'accusés de réception de TCP et peut donc aider à éviter des temporisations de retransmission. Cependant, en envoyant ces segments, l'envoyeur a deux copies des mêmes données considérées comme étant dans le réseau (et aussi dans l'estimation du tuyau, dans le cas de (3)). Lorsque un ACK ou SACK arrive qui couvre ce segment retransmis, l'envoyeur ne peut pas être exactement sûr de la quantité de données qui restent dans le réseau (une des deux, ou les deux, transmissions du paquet). Donc, l'envoyeur peut sous-estimer le tuyau en considérant que les deux segments ont quitté le réseau alors qu'il est possible que seul un des deux l'ai fait.

(5) Si les conditions des règles (1), (2), (3), et (4) ne sont pas satisfaites, NextSeg () DOIT alors indiquer l'échec, et aucun segment n'est retourné.

Note : L'algorithme de récupération de perte fondé sur SACK présenté dans le présent document exige plus de ressources de calcul que les précédentes stratégies TCP de récupération de perte. Cependant, on pense que la structure des données de tableau de résultats peut être mise en œuvre d'une manière raisonnablement efficace (à la fois en termes de complexité de calcul et d'utilisation de mémoire) dans la plupart des mises en œuvre TCP.

5. Détails de l'algorithme

À réception de tout ACK contenant des informations de SACK, le tableau de résultats DOIT être mis à jour via le sous programme Update ().

Si l'ACK entrant est un accusé de réception cumulatif, le TCP DOIT remettre DupAcks à zéro.

Si l'ACK entrant est un accusé de réception dupliqué selon la définition de la Section 2 (sans considération de son état comme accusé de réception cumulatif) et si TCP n'est pas actuellement en récupération de perte, TCP DOIT augmenter DupAcks de un et suivre les étapes suivantes :

(1) Si DupAcks ≤ DupThresh, passer à l'étape (4).

Note : Cette vérification couvre le cas où TCP reçoit des informations de SACK pour plusieurs segments plus petits que SMSS, ce qui peut empêcher IsLost() (étape suivante) de déclarer un segment perdu.

(2) Si DupAcks < DupThresh mais si IsLost (HighACK + 1) retourne vrai – ce qui indique qu'au moins trois segments sont arrivés au dessus du point d'accusé de réception cumulatif actuel, ce qui est pris pour l'indication de perte – passer à l'étape (4).

(3) TCP PEUT transmettre des segments de données non envoyés précédemment comme dans la transmission limitée de la [RFC5681], sauf que le nombre d'octets qui peut être envoyé est gouverné par les paramètres pipe et cwnd comme suit :

(3.1) Régler HighRxt à HighACK.

(3.2) Lancer SetPipe ().

(3.3) Si (cwnd - pipe) ≥ 1 SMSS, il existe des données non envoyées précédemment, et si la fenêtre annoncée du receveur le permet, transmettre jusqu'à 1 SMSS de données commençant par l'octet HighData+1 et mettre à jour HighData pour refléter cette transmission, puis retourner à (3.2).

(3.4) Terminer le traitement de cet ACK.

(4) Invoquer la retransmission rapide et entrer dans la récupération de perte comme suit :

(4.1) RecoveryPoint = HighData. Lorsque l'envoyeur TCP reçoit un ACK cumulatif pour cet octet de données, la phase de récupération de perte est terminée.

(4.2) ssthresh = cwnd = (FlightSize / 2)
La fenêtre d'encombrement (cwnd) et le seuil de démarrage lent (ssthresh) sont réduits à la moitié de FlightSize selon la [RFC5681]. Noter de plus que la [RFC5681] exige que tout segment envoyé au titre du mécanisme de

transmission limitée ne soit pas compté dans FlightSize pour les besoin de l'équation ci-dessus.

(4.3) Retransmettre le premier segment de données présumé abandonné -- le segment qui commence par le numéro de séquence HighACK + 1. Pour empêcher la retransmission répétée des mêmes données ou une retransmission de secours prématurée, régler HighRxt et RescueRxt tous deux au plus haut numéro de séquence dans le segment retransmis.

(4.4) Lancer SetPipe ()
Régler une variable "pipe" au nombre d'octets en cours actuellement "dans le tuyau" ; ce sont les données qui ont été envoyées par l'envoyeur TCP mais pour lesquelles aucun accusé de réception cumulatif ou sélectif n'a été reçu et il n'a pas été déterminé que les données aient été abandonnées dans le réseau. On suppose que les données sont toujours en train de traverser le chemin du réseau.

(4.5) Afin de tirer parti d'une cwnd potentiellement disponible supplémentaire, passer à l'étape (C) ci-dessous.

Une fois que TCP est dans la phase de récupération de perte, la procédure suivante DOIT être utilisée pour chaque ACK arrivant :

(A) Un ACK cumulatif entrant pour un numéro de séquence supérieur à RecoveryPoint signale la fin de la récupération de perte, et la phase de récupération de perte DOIT être terminée. Toute information contenue dans le tableau des résultats pour un numéro de séquences supérieur à la nouvelle valeur de HighACK NE DEVRAIT PAS être supprimée lorsque la phase de récupération de perte est quittée.

(B) À réception d'un ACK qui ne couvre pas le RecoveryPoint, les actions suivantes DOIVENT être effectuées :
  (B.1) Utiliser Update () pour enregistrer les nouvelles informations de SACK convoyées dans l'ACK entrant.
  (B.2) Utiliser SetPipe () pour recalculer le nombre d'octets encore dans le réseau.

(C) Si cwnd - pipe ≥ 1 SMSS, l'envoyeur DEVRAIT transmettre un ou plusieurs segments comme suit :
  (C.1) Le tableau des résultats DOIT être interrogé via NextSeg () sur la gamme de numéros de séquence du prochain segment à transmettre (si il en est) et ce segment doit être envoyé. Si NextSeg () retourne un échec (pas de données à envoyer à retourner sans rien envoyer (c'est-à-dire, terminer les étapes C.1 à C.5).
  (C.2) Si un ou des octets de données envoyés en (C.1) sont en dessous de HighData, HighRxt DOIT être réglé au plus fort numéro de séquence du segment retransmis sauf si la règle (4) NextSeg () a été invoquée pour cette retransmission.
  (C.3) Si un ou des octets de données envoyés en (C.1) sont supérieurs à HighData, HighData doit être mis à jour pour refléter la transmission de données non envoyées précédemment.
  (C.4) L'estimation de la quantité de données en instance dans le réseau doit être mise à jour en incrémentant pipe du nombre d'octets transmis en (C.1).
  (C.5) Si cwnd - pipe ≥ 1 SMSS, retourner à (C.1)

Note : Les étapes (A) et (C) peuvent envoyer une salve de segments dos à dos dans le réseau si l'accusé de réception cumulatif entrant est pour plus de SMSS octets de données, ou si les blocs SACK entrants indiquent que plus de SMSS octets de données ont été perdus dans la seconde moitié de la fenêtre.

5.1 Temporisations de retransmission

Afin d'éviter des situations impossibles pour la mémoire, il est permis au receveur TCP d'éliminer des données qui ont déjà été acquittées sélectivement. Par suite, la [RFC2018] suggère qu'un envoyeur TCP DEVRAIT purger les informations de SACK collectées d'un receveur après une fin de temporisation de retransmission (RTO) "car la fin de temporisation pourrait indiquer que le receveur des données a renoncé". De plus, un envoyeur TCP DOIT "ignorer les informations de SACK précédentes pour déterminer quelles données retransmettre". Cependant, depuis la publication de la [RFC2018], certains en sont venus à considérer cela comme trop dur. Il a été suggéré que, dans la mesure ou des vérifications robustes sont présentes pour la renonciation, une mise en œuvre peut conserver et utiliser les informations de SACK après un événement de fin de temporisation [Errata1610]. Bien que le présent document ne change pas la spécification de la [RFC2018], on note que les mises en œuvre devraient consulter toutes les mises à jour de la [RFC2018] sur ce sujet. De plus, un envoyeur de SACK TCP DEVRAIT utiliser toutes les informations de SACK rendues disponibles pendant la récupération de perte suivant un RTO.

Si un RTO survient durant la récupération de perte, comme spécifié dans le présent document, RecoveryPoint DOIT être réglé à HighData. De plus, la nouvelle valeur de RecoveryPoint DOIT être préservée et l'algorithme de récupération de perte présenté dans ce document DOIT être terminé. De plus, une nouvelle phase de récupération (comme décrit à la Section 5) NE DOIT PAS être initiée tant que HighACK est supérieur ou égal à la nouvelle valeur de RecoveryPoint.

Comme décrit aux Sections 4 et 5, Update() DEVRAIT continuer d'être utilisé de façon appropriée à réception des ACK. Cela permet que la période de récupération après un RTO bénéficie de toutes les informations disponibles fournies par le receveur, même si les informations de SACK ont été purgées à cause du RTO.

Si il y a des segments qui manquent dans la mémoire tampon du receveur à la suite du traitement du segment retransmis, l'ACK correspondant va contenir des informations de SACK. Dans ce cas, un envoyeur TCP DEVRAIT utiliser ces informations de SACK pour déterminer quelles données devraient être envoyées dans chaque segment suivant un RTO. L'algorithme exact pour cette sélection n'est pas spécifié dans le présent document (précisément, NextSeg () n'est pas approprié durant la récupération de perte après un RTO). Une approche relativement directe de "remplissage" de l'espace de numéros de séquence rapportés comme manquants devrait être une approche raisonnable.

6. Gestion du temporisateur RTO

L'estimateur RTO standard TCP est défini dans la [RFC6298]. Dû au fait que l'algorithme SACK du présent document peut avoir un impact sur le comportement de l'estimateur, les mises en œuvre peuvent souhaiter examiner comment le temporisateur est géré. La [RFC6298] dit que le temporisateur RTO doit être réarmé chaque fois qu'un ACK arrive et fait avancer le point d'ACK cumulatifs. Comme l'algorithme présenté dans ce document peut garder l'horloge d'ACK en fonctionnement à travers un événement de perte très significatif (relativement plus long que l'algorithme décrit dans la [RFC5681]) sur certains réseaux l'événement de perte pourrait durer plus longtemps que le RTO. Dans ce cas, le temporisateur RTO va expirer prématurément et un segment qui n'a pas besoin d'être retransmis sera renvoyé.

Donc, on donne aux mises en œuvre la latitude d'utiliser la gestion de RTO standard du style de la [RFC6298] ou, facultativement, on PEUT utiliser une variante plus prudente qui réarme le temporisateur RTO à chaque retransmission qui est envoyée durant la récupération. Cela donne un temporisateur plus prudent que celui spécifié dans la [RFC6298], et cette solution de remplacement peut n'être pas toujours attractive. Cependant, dans certains cas, cela peut empêcher des retransmissions inutiles, une transmission N retours, et une plus grande réduction de la fenêtre d'encombrement.

7. Recherches en cours

L'algorithme spécifié dans ce document est analysé dans [FF96], qui montre que cet algorithme est efficace pour réduire le temps de transfert sur le Reno TCP standard [RFC5681] lorsque plusieurs segments sont abandonnés d'une fenêtre de données (en particulier lorsque le nombre d'abandons augmente). [AHKO97] montre que l'algorithme défini dans ce document peut améliorer notablement le débit dans les connexions qui traversent des canaux par satellite.

8. Considérations pour la sécurité

L'algorithme présenté dans cet article partage les considérations pour la sécurité avec la [RFC5681]. Une différence clé est qu'un algorithme fondé sur les SACK est plus robuste contre les attaquants qui font de faux ACK dupliqués pour forcer l'envoyeur TCP à réduire la cwnd. Avec les SACK, les envoyeurs TCP ont en plus à vérifier si un ACK particulier est légitime ou non. Bien qu'il ne soit pas à toute épreuve, SACK fournit bien une certaine protection dans ce domaine.

De la même façon, [CPNI309] décrit une variante de l'attaque en aveugle de la [RFC5961] par laquelle un attaquant peut parodier des données hors fenêtre pour un point d'extrémité TCP, l'amenant à répondre à l'homologue légitime avec un ACK dupliqué cumulatif, selon la [RFC0793]. Ajouter une exigence fondée sur SACK de déclancher la récupération de perte atténue effectivement cette attaque, car les ACK dupliqués causés par des segments hors fenêtre ne contiendront pas d'informations de SACK indiquant la réception de données dans la fenêtre non acquittées sélectivement précédemment.

9. Changements par rapport à la RFC 3517

La variable d'état "DupAcks" a été ajoutée à la liste des variables entretenue par cet algorithme, et son usage est spécifié.

La fonction IsLost () a été modifiée pour exiger que plus de (DupThresh - 1) * SMSS octets aient été acquittés sélectivement au dessus d'un certain numéro de séquence comme indication qu'il est perdu, ce qui change de l'exigence minimum de (DupThresh * SMSS) décrite dans la [RFC3517]. Cela conserve l'exigence qu'au moins trois segments suivant le numéro de séquence en question aient été acquittés sélectivement, tout en améliorant la détection dans le cas où l'envoyeur a des segments en cours qui sont plus petits que SMSS.

La définition d'un "accusé de réception dupliqué" a été modifiée pour utiliser les informations de SACK à détecter les pertes. Les accusés de réception cumulatifs dupliqués peuvent être causés par la perte ou par un réarrangement dans le réseau. Pour distinguer sans ambiguïté perte et réarrangement, l'algorithme de retransmission rapide de TCP [RFC5681] attend que trois ACK dupliqués arrivent pour déclancher la récupération de perte. Cette notion était alors la base de

l'algorithme spécifié dans la [RFC3517]. Cependant, avec les informations de SACK, il n'est pas besoin de s'appuyer aveuglément sur le champ d'accusé de réception cumulatif. On peut s'appuyer sur les informations supplémentaires présentes dans les blocs SACK pour comprendre que trois segments se tenant au dessus d'un trou dans l'espace des numéros de séquence sont arrivés chez le receveur, et on peut utiliser cette compréhension pour déclancher la récupération de perte. Cette notion a été utilisée dans la [RFC3517] durant la récupération de perte, et le changement dans le présent document est que la notion est aussi utilisée pour entrer dans une phase de récupération de perte.

La variable d'état "RescueRxt" a été ajoutée à la liste des variables entretenues par l'algorithme, et son usage spécifié. Cette variable est utilisée pour permettre une retransmission par entrée supplémentaire en récupération de perte, afin de garder en fonctionnement l'horloge des ACK dans certaines circonstances qui impliquent une perte à la fin de la fenêtre. Ce mécanisme permet que pas plus d'un segment d'au plus 1 SMSS soit au mieux retransmis par récupération de perte.

La règle (3) de NextSeg() a été changée de PEUT en DEVRAIT, pour refléter de façon appropriée l'opinion des auteurs et du groupe de travail qu'elle devrait être laissée dedans, plutôt que sortie, si une mise en œuvre n'a pas une raison impérieuse de faire autrement.

10. Remerciements

Les auteurs souhaitent remercier Sally Floyd pour son soutien à la [RFC3517] et ses commentaires sur les premiers projets. L'algorithme décrit dans le présent document est plus ou moins fondé sur un algorithme présenté par Kevin Fall et Sally Floyd dans [FF96], bien que les auteurs de ce document assument la responsabilité de toutes les fautes du présent texte.

Kevin Fall était co-auteur de la [RFC3517] et y a fait une contribution cruciale, et donc aussi à ce travail qui en découle.

Murali Bashyam, Ken Calvert, Tom Henderson, Reiner Ludwig, Jamshid Mahdavi, Matt Mathis, Shawn Ostermann, Vern Paxson, et Venkat Venkatsubra ont fourni di réactions précieuses sur les premières versions de ce document.

Merci à Matt Mathis et Jamshid Mahdavi pour la mise en œuvre du tableau de résultats dans NS et d'avoir guider nos pensées sur le suivi de l'état de SACK.

Le premier auteur tient à remercier l'Université de l'Ohio et le groupe de recherches Internetworking de l'Université de l'Ohio pour la prise en charge matérielle de ce travail sur la RFC 3517, d'où est dérivé le présent document.

11. Références

11.1 Références normatives

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

[RFC2018] M. Mathis et autres, "Options d'accusé de réception sélectif sur TCP", octobre 1996. *(P.S.)*

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

[RFC5681] M. Allman, V. Paxson, E. Blanton, "Contrôle d'encombrement de TCP", septembre 2009. *(D. S.)*

11.2 Références pour information

[AHKO97] ark Allman, Chris Hayes, Hans Kruse, Shawn Ostermann, "TCP Performance Over Satellite Links", Proceedings of the 5th International Conference on Telecommunications Systems, Nashville, TN, mars 1997.

[All00] Mark Allman, "A Web Server's View of the Transport Layer", ACM Computer Communication Review, 30(5), octobre 2000.

[CPNI309] Fernando Gont, "Security Assessment of the Transmission Control Protocol (TCP)", CPNI Technical Note 3/2009, <http://www.gont.com.ar/papers/tn-03-09-security-assessment-TCP.pdf>, février 2009.

[Errata1610] RFC Errata, Errata ID 1610, RFC 2018, <http://www.rfc-editor.org>. *(Intégré dans la VF de la RFC2018)*

[FF96] Kevin Fall et Sally Floyd, "Simulation-based Comparisons of Tahoe, Reno et SACK TCP", Computer Communication Review, juillet 1996.

[Jac90] Van Jacobson, "Modified TCP Congestion Avoidance Algorithm", Rapport technique, LBL, avril 1990.

[PF01] Jitendra Padhye, Sally Floyd "Identifying the TCP Behavior of Web Servers", ACM SIGCOMM, août 2001.

[RFC2914] S. Floyd, "Principes du contrôle d'encombrement", BCP 41, septembre 2000.

[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 la présente RFC)*

[RFC5961] A. Ramaiah, R. Stewart, M. Dalal, "Amélioration de la robustesse de TCP aux attaques sur fenêtre aveugle" août 2010. (*P.S.)*

[RFC6298] V. Paxson, M. Allman, J. Chu, M. Sargent, "Calcul du temporisateur de retransmission de TCP", juin 2011. *(P.S.)*

[RFC6582] T. Henderson, S. Floyd, A. Gurtov, Y. Nishida, "Modification NewReno à l'algorithme de récupération rapide de TCP", avril 2012. (*Remplace la* RFC3782) *(P.S.)*

Adresse des auteurs

Ethan Blanton
Purdue University Computer Sciences
305 N. University St.
West Lafayette, IN 47907
United States
mél : elb@psg.com

Mark Allman
International Computer Science Institute
1947 Center St. Suite 600
Berkeley, CA 94704
United States
mél : mallman@icir.org
http://www.icir.org/mallman

Lili Wang
Juniper Networks
10 Technology Park Drive
Westford, MA 01886
United States
mél : liliw@juniper.net

Ilpo Jarvinen
University of Helsinki
P.O. Box 68
FI-00014 University of Helsinki
Finland
mél : ilpo.jarvinen@helsinki.fi

Markku Kojo
University of Helsinki
P.O. Box 68
FI-00014 University of Helsinki
Finland
mél : kojo@cs.helsinki.fi

Yoshifumi Nishida
WIDE Project
Endo 5322
Fujisawa, Kanagawa 252-8520
Japan
mél : nishida@wide.ad.jp