Groupe de travail Réseau
Request for Comments : 5681
RFC rendue obsolète : 2581
Catégorie : En cours de normalisation

M. Allman & V. Paxson, ICSI
E. Blanton, Purdue University
septembre 2009
Traduction Claude Brière de L’Isle

Contrôle d’encombrement pour TCP

Résumé
Le présent document définit les quatre algorithmes de contrôle d’encombrement imbriqués de TCP : démarrage lent, évitement d’encombrement, retransmission rapide, et récupération rapide. De plus, le document spécifie comment TCP devrait commencer la transmission après une période d’inactivité relativement longue, et discute des diverses méthodes de génération d’accusé de réception. Le présent document rend obsolète la RFC 2581.

Statut de ce mémoire
Le présent document spécifie un protocole Internet en cours de normalisation pour la communauté de l’Internet, et appelle à des discussions et des suggestions pour son amélioration. Prière de se reporter à l’édition actuelle du STD 1 "Normes des protocoles officiels de l’Internet" pour connaître l’état de normalisation et le statut de ce protocole. La distribution du présent mémoire n’est soumise à aucune restriction.

Table des Matières

1. Introduction

Le présent document spécifie quatre algorithmes de contrôle d'encombrement pour TCP [RFC0793] : démarrage lent, évitement d'encombrement, retransmission rapide et récupération rapide. Ces algorithmes ont été imaginés dans [Jac88] et [Jac90]. Leur utilisation avec TCP est normalisée dans la [RFC1122]. Les premiers travaux supplémentaires sur le contrôle d'encombrement par augmentation additive, diminution multiplicative figurent dans [CJ89].

Noter que [Ste94] donne des exemples de ces algorithmes en action et [WS95] fournit une explication du code source pour la mise en œuvre BSD de ces algorithmes.

En plus de spécifier ces algorithmes de contrôle d'encombrement, le présent document spécifie ce que les connexions TCP devraient faire après une période d'inactivité relativement longue, et spécifie et précise certaines des questions relevant de la génération de l'accusé de réception TCP.

Le présent document rend obsolète la [RFC2581], qui rendait obsolète la [RFC2001].

Le présent document est organisé comme suit. La Section 2 donne diverses définitions qui seront utilisées dans le document. La Section 3 donne la spécification des algorithmes de contrôle d'encombrement. La Section 4 évoque les problèmes qui se rapportent aux algorithmes de contrôle d'encombrement et finalement, la Section 5 évoque les considérations pour la sécurité.

Dans ce document, les mots clés DOIT, NE DOIT PAS, EXIGE, DEVRAIT, NE DEVRAIT PAS, RECOMMANDE, PEUT et FACULTATIF sont à interpréter comme décrit dans la [RFC2119].

2. Définitions

Cette section donne la définition de plusieurs termes qui seront utilisés dans le reste du présent document.

Segment : Un segment est TOUT paquet de données ou d'accusé de réception TCP/IP (ou les deux).

Taille maximum de segment d'envoyeur (SMSS, *Sender Maximum Segment Size*) : C'est la taille du plus grand segment que l'envoyeur peut émettre. Cette valeur peut se fonder sur l'unité maximum de transmission du réseau, sur l'algorithme de découverte de la MTU du chemin [RFC1191], [RFC4821], sur RMSS (voir la définition suivante) ou sur d'autres facteurs. La taille n'inclut pas les en-têtes et options TCP/IP.

Taille maximum de segment de receveur (RMSS, *Receiver Maximum Segment Size*) : C'est la taille du plus grand segment que le receveur veut accepter. C'est la valeur spécifiée dans l'option MSS envoyée par le receveur durant le démarrage de la connexion. Ou, si l'option MSS n'est pas utilisée, c'est 536 octets [RFC1122]. La taille n'inclut pas l'en-tête TCP/IP ni les options.

Segment de pleine taille : C'est un segment qui contient le nombre maximum d'octets de données permis (c'est-à-dire, un segment contenant SMSS octets de données).

Fenêtre de receveur (rwnd, *receiver window*) : C'est la fenêtre de receveur le plus récemment annoncée.

Fenêtre d'encombrement (cwnd, *congestion window*) : Variable d'état TCP qui limite la quantité de données qu'un TCP peut envoyer. À tout instant donné, un TCP NE DOIT PAS envoyer de données avec un numéro de séquence supérieur à la somme du plus fort numéro de séquence acquitté et du minimum de cwnd et rwnd.

Fenêtre initiale (IW, *Initial Window*) : C'est la taille de la fenêtre d'encombrement de l'envoyeur après l'achèvement de la prise de contact à trois phases.

Fenêtre de perte (LW, *Loss Window*) : C'est la taille de la fenêtre d'encombrement après qu'un envoyeur TCP détecte la perte en utilisant son temporisateur de retransmission.

Fenêtre de redémarrage (RW, *Restart Window*) : C'est la taille de la fenêtre d'encombrement après qu'un TCP redémarre la transmission après une période d'inactivité (si l'algorithme de démarrage lent est utilisé ; voir les détails au paragraphe 4.1).

Taille en cours *(flight size)* : C'est la quantité de données qui ont été envoyées mais non encore acquittées cumulativement.

Accusé de réception dupliqué : Un accusé de réception est considéré comme "dupliqué" dans les algorithmes suivants

lorsque (a) le receveur de l'ACK a des données en suspens, (b) l'accusé de réception entrant ne porte pas de données, (c) les bits SYN et FIN sont tous deux à zéro, (d) le numéro d'accusé de réception est égal au plus grand accusé de réception reçu sur la connexion considérée (TCP.UNA de la [RFC0793]) et (e) la fenêtre annoncée dans l'accusé de réception entrant est égale à la fenêtre annoncée dans le dernier accusé de réception entrant.

Autrement, un TCP qui utilise les accusés de réception sélectifs (SACK) des [RFC2018], [RFC2883] peut se servir des informations de SACK pour déterminer quand un ACK entrant est un "dupliqué" (par exemple, si l'ACK contient des informations SACK précédemment inconnues).

3. Algorithmes de contrôle d'encombrement

Cette section définit les quatre algorithmes de contrôle d'encombrement : démarrage lent, évitement d'encombrement, retransmission rapide, et récupération rapide, développés dans [Jac88] et [Jac90]. Dans certaines situations, il peut être avantageux pour un envoyeur TCP d'être plus prudent que ce que les algorithmes permettent ; cependant, un TCP NE DOIT PAS être plus agressif que ce que les algorithmes suivants permettent (c'est-à-dire, il NE DOIT PAS envoyer de données lorsque la valeur de cwnd calculée par les algorithmes suivants ne permettrait pas que les données soient envoyées).

Noter aussi que les algorithmes spécifiés dans le présent document fonctionnent en utilisant la perte comme signal d'encombrement. La notification explicite d'encombrement (ECN, *Explicit Congestion Notification*) pourrait aussi être utilisée comme spécifié dans la [RFC3168].

3.1 Démarrage lent et évitement d'encombrement

Les algorithmes de démarrage lent et d'évitement d'encombrement DOIVENT être utilisés par un envoyeur TCP pour contrôler la quantité de données en suspens qui sont injectées dans le réseau. Pour mettre en œuvre ces algorithmes, deux variables sont ajoutées à l'état TCP par connexion. La fenêtre d'encombrement (cwnd) est une limite du côté de l'envoyeur de la quantité de données que l'envoyeur peut transmettre dans le réseau avant de recevoir un accusé de réception (ACK), tandis que la fenêtre annoncée du receveur (rwnd) est une limite du côté du receveur à la quantité de données en suspens. Le minimum de cwnd et rwnd gouverne la transmission des données.

Une autre variable d'état, le seuil de démarrage lent (ssthresh, *slow start threshold*) est utilisée pour déterminer si les algorithmes démarrage lent ou évitement d'encombrement sont utilisés pour contrôler la transmission des données, comme exposé ci-dessous.

Commencer la transmission dans un réseau dont les conditions sont inconnues exige de TCP qu'il sonde lentement le réseau pour déterminer la capacité disponible, afin d'éviter d'encombrer le réseau avec une salve de données excessive. L'algorithme démarrage lent est utilisé à cette fin au début d'un transfert, ou après la réparation d'une perte détectée par le temporisateur de retransmission. Le démarrage lent sert de plus à démarrer "l'horloge d'ACK" utilisée par l'envoyeur TCP pour libérer les données dans le réseau dans les algorithmes démarrage lent, évitement d'encombrement, et récupération de perte.

IW, la valeur initiale de cwnd, DOIT être réglée en utilisant les lignes directrices suivantes comme limite supérieure.

Si SMSS > 2190 octets : IW = 2 * SMSS octets et NE DOIT PAS faire plus de 2 segments
Si (SMSS > 1095 octets) et (SMSS ≤ 2190 octets) : IW = 3 * SMSS octets et NE DOIT PAS faire plus de 3 segments
Si SMSS ≤ 1095 octets : IW = 4 * SMSS octets et NE DOIT PAS faire plus de 4 segments

Comme spécifié dans la [RFC3390], le SYN/ACK et l'accusé de réception du SYN/ACK NE DOIVENT PAS augmenter la taille de la fenêtre d'encombrement. De plus, si le SYN ou SYN/ACK est perdu, la fenêtre initiale utilisée par un envoyeur après un SYN correctement transmis DOIT être d'un segment consistant en au plus SMSS octets.

Les raisons détaillée et une discussion du réglage de IW sont fournies dans la [RFC3390].

Lorsque des fenêtres d'encombrement initiales de plus d'un segment sont mises en œuvre avec la découverte de la MTU du chemin de la [RFC1191], et que la MSS utilisée est trouvée trop grande, la fenêtre d'encombrement cwnd DEVRAIT être réduite pour empêcher les grosses salves de plus petits segments. Précisément, cwnd DEVRAIT être réduit du ratio de la vieille taille de segment sur la nouvelle taille de segment.

La valeur initiale de ssthresh DEVRAIT être réglée arbitrairement élevée (par exemple, à la taille de la fenêtre annoncée la plus grande possible) mais ssthresh DOIT être réduit en réponse à l'encombrement. Régler ssthresh aussi haut que possible permet que les conditions du réseau, plutôt que des limites d'hôte arbitraire, dictent le taux d'envoi. Dans les cas où les systèmes d'extrémité ont une bonne compréhension du chemin du réseau, un réglage plus soigneux de la valeur initiale de ssthresh peut avoir des mérites (par exemple, que l'hôte d'extrémité ne crée pas d'encombrement le long du chemin).

L'algorithme démarrage lent est utilisé lorsque cwnd < ssthresh, tandis que l'algorithme évitement d'encombrement est utilisé lorsque cwnd > ssthresh. Lorsque cwnd et ssthresh sont égaux, l'envoyeur peut utiliser indifféremment démarrage lent ou évitement d'encombrement.

Durant le démarrage lent, un TCP incrémente cwnd d'au plus SMSS octets pour chaque ACK reçu qui accuse réception cumulativement de nouvelles données. Le démarrage lent se termine lorsque cwnd excède ssthresh (ou, facultativement, lorsque il l'atteint, comme noté ci-dessus) ou lorsque de l'encombrement est observé. Bien que traditionnellement, les mises en œuvre de TCP aient augmenté cwnd de précisément SMSS octets à réception d'un ACK couvrant de nouvelles données, on RECOMMANDE que les mises en œuvre de TCP augmentent cwnd, selon ;

cwnd + = min (N, SMSS) (2)

où N est le nombre d'octets non acquittés précédemment qui sont acquittés dans l'ACK entrant. Cet ajustement fait partie du comptage approprié des octets de la [RFC3465] et assure la robustesse contre le mauvais comportement de certains receveurs qui peuvent tenter d'induire un envoyeur à enfler artificiellement cwnd en utilisant un mécanisme appelé "division d'ACK" [SCWA99]. La division d'ACK consiste en ce qu'un receveur envoie plusieurs ACK pour un seul segment TCP, chacun n'acquittant qu'une portion des données. Un TCP qui incrémente cwnd de SMSS pour chacun de tels ACK va enfler de façon inappropriée la quantité de données injectées dans le réseau.

Durant l'évitement d'encombrement, cwnd est en gros incrémenté d'un segment de pleine taille par délai d'aller-retour (RTT). L'évitement d'encombrement continue jusqu'à ce que de l'encombrement soit détecté. Les lignes directrices de base pour incrémenter cwnd durant l'évitement d'encombrement sont :

* PEUT incrémenter cwnd de SMSS octets
* DEVRAIT incrémenter cwnd selon l'équation (2) une fois par RTT
* NE DOIT PAS incrémenter cwnd de plus de SMSS octets

On note que la [RFC3465] permet que cwnd augmente de plus de SMSS octets pour les accusés de réception entrants durant le démarrage lent sur une base expérimentale ; cependant, un tel comportement n'est pas permis au titre de la présente norme.

La façon RECOMMANDÉE d'augmenter cwnd durant l'évitement d'encombrement est de compter le nombre d'octets qui ont été acquittés par des ACK pour les nouvelles données. (Un inconvénient de cette mise en œuvre est qu'elle exige de maintenir une variable d'état supplémentaire.) Lorsque le nombre d'octets acquittés atteint cwnd, celui-ci peut être incrémenté de jusqu'à SMSS octets. Noter que durant l'évitement d'encombrement, cwnd NE DOIT PAS être augmenté de plus de SMSS octets par RTT. Cette méthode permet à la fois aux TCP d'augmenter cwnd d'un segment par RTT en présence d'ACK retardés et fournit de la robustesse contre les attaques par division d'ACK.

Une autre formule courante que PEUT utiliser un TCP pour mettre à jour cwnd durant l'évitement d'encombrement est données dans l'équation (3) :

cwnd + = SMSS*SMSS/cwnd (3)

Cet ajustement est exécuté sur chaque ACK entrant qui acquitte des nouvelles données. L'équation (3) donne une approximation acceptable du principe sous-jacent d'augmentation de cwnd d'un segment de pleine taille par RTT. (Noter que pour une connexion dans laquelle le receveur accuse réception d'un paquet sur deux, (3) est moins agressif que ce qui est permis – augmentant en gros cwnd tous les seconds RTT.)

Note de mise en œuvre : Comme une arithmétique d'entiers est généralement utilisée dans les mises en œuvre de TCP, la formule données dans l'équation (3) peut échouer à augmenter cwnd lorsque la fenêtre d'encombrement est supérieure à SMSS*SMSS. Si la formule ci-dessus donne 0, le résultat DEVRAIT être arrondi à un octet.

Note de mise en œuvre : Les anciennes mises en œuvre ont une constante additive supplémentaire sur le côté droit de l'équation (3). C'est incorrect et peut en fait conduire à diminuer les performances [RFC2525].

Note de mise en œuvre : Certaines mises en œuvre conservent cwnd en unités d'octets, alors que d'autres le font en unités de segments de pleine taille. Ces dernières pourraient trouver l'équation (3) difficile à utiliser, et pourront préférer utiliser l'approche du comptage exposée au paragraphe précédent.

Lorsque un envoyeur TCP détecte une perte de segment en utilisant le temporisateur de retransmission et que le segment en cause n'a pas encore été renvoyé au moyen du temporisateur de retransmission, la valeur de ssthresh DOIT être réglée à pas plus que la valeur donnée par l'équation (4) :

ssthresh = max (Taille en cours / 2, 2*SMSS) (4)

où, comme exposé ci-dessus, Taille en cours est la quantité de données en instance dans le réseau.

D'un autre côté, lorsque un envoyeur TCP détecte une perte de segment en utilisant le temporisateur de retransmission et que le segment en cause a déjà été retransmis au moyen du temporisateur de retransmission au moins une fois, la valeur de ssthresh est gardée constante.

Note de mise en œuvre : Une erreur courante est de simplement utiliser cwnd, plutôt que Taille en cours, qui dans certaines mises en œuvre peut incidemment augmenter bien au delà de rwnd.

De plus, sur une fin de temporisation(comme spécifié dans la [RFC2988]) cwnd DOIT être réglé à pas plus que la fenêtre de perte, LW, qui est égale à un segment de pleine taille (sans considération de la valeur de IW). Donc, après retransmission du segment éliminé, l'envoyeur TCP utilise l'algorithme de démarrage lent pour augmenter la fenêtre entre un segment de pleine taille et la nouvelle valeur de ssthresh, point auquel l'évitement d'encombrement reprend la main.

Comme on le montre dans [FF96] et la [RFC3782], la récupération de perte fondée sur le démarrage lent après une fin de temporisation peut causer des retransmissions parasites qui déclenchent des accusés de réception dupliqués. La réaction à l'arrivée de ces ACK dupliqués dans les mises en œuvre TCP varie largement. Le présent document ne spécifie pas comment traiter de tels accusés de réception, mais note cela comme un domaine qui pourra bénéficier d'une attention, expérimentation et spécification supplémentaires.

3.2 Retransmission/récupération rapide

Un receveur TCP DEVRAIT envoyer un ACK dupliqué immédiat lorsque un segment déclassé arrive. L'objet de cet ACK est d'informer l'envoyeur qu'un segment a été reçu déclassé et du numéro de séquence qui est attendu. Du point de vue de l'envoyeur, les ACK dupliqués peuvent être causés par un certain nombre de problèmes de réseau. D'abord, ils peuvent être causés par des segments abandonnés. Dans ce cas, tous les segments après le segment abandonné vont déclancher des ACK dupliqués jusqu'à ce que la perte soit réparée. Ensuite, les ACK dupliqués peuvent être causés par le réarrangement des segments de données par le réseau (ce qui n'est pas un événement rare le long de certains chemins du réseau [Pax97]). Finalement, les ACK dupliqués peuvent être causés par la réplication des segments d'ACK ou de données par le réseau. De plus, un receveur TCP DEVRAIT envoyer un ACK immédiat lorsque le segment entrant bouche en tout ou partie un trou dans l'espace des numéros de séquence. Cela va générer des informations plus à jour pour un envoyeur qui récupère d'une perte au moyen d'une temporisation de retransmission, d'une retransmission rapide, ou d un algorithme évolué de récupération de perte, comme évoqué au paragraphe 4.3.

L'envoyeur TCP DEVRAIT utiliser l'algorithme de "retransmission rapide" pour détecter et réparer les pertes, sur la base des ACK dupliqués entrants. L'algorithme de retransmission rapide utilise l'arrivée de 3 ACK dupliqués (comme défini à la section 2, sans aucun ACK intervenant qui déplace SND.UNA) comme indication qu'un segment a été perdu. Après avoir reçu 3 ACK dupliqués, TCP effectue une retransmission de ce qui paraît être le segment manquant, sans attendre l'arrivée à expiration du temporisateur de retransmission.

Après que l'algorithme de retransmission rapide a envoyé ce qui paraît être le segment manquant, l'algorithme de "récupération rapide" gouverne la transmission des nouvelles données jusqu'à ce qu'arrive un ACK non dupliqué. La raison pour ne pas effectuer le démarrage lent est que la réception des ACK dupliqués non seulement indique qu'un segment a été perdu, mais aussi que des segments sont très vraisemblablement en train de quitter le réseau (bien qu'une duplication massive de segments par le réseau puisse invalider cette conclusion). En d'autres termes, comme le receveur peut seulement générer un ACK dupliqué lorsque un segment est arrivé, ce segment a quitté le réseau et est dans la mémoire tampon du receveur, de sorte que nous savons qu'il ne consomme plus les ressources du réseau. De plus, comme "l'horloge" des ACK [Jac88] est préservée, l'envoyeur TCP peut continuer de transmettre de nouveaux segments (bien que la transmission doive continuer d'utiliser un cwnd réduit, car la perte est une indication d'encombrement).

Les algorithmes de retransmission rapide et de récupération rapide sont mis en œuvre ensemble comme suit.

1. Sur le premier et second ACK dupliqués reçus chez un envoyeur, un TCP DEVRAIT envoyer un segment de données non envoyées précédemment selon la [RFC3042] pourvu que la fenêtre annoncée du receveur le permette, la taille totale

de données en cours resterait inférieure ou égale à cwnd plus 2*SMSS, et que les nouvelles données soient disponibles pour transmission. De plus, l'envoyeur TCP NE DOIT PAS changer cwnd pour refléter ces deux segments [RFC3042]. Noter qu'un envoyeur qui utilise SACK [RFC2018] NE DOIT PAS envoyer de nouvelles données à moins que l'accusé de réception dupliqué entrant ne contienne de nouvelles informations SACK.

2. Lorsque le troisième ACK dupliqué est reçu, un TCP DOIT régler ssthresh à pas plus que la valeur donnée dans l'équation (4). Lorsque la [RFC3042] est utilisée, les données supplémentaires envoyées en transmission limitée NE DOIVENT PAS être incluses dans ce calcul.

3. Le segment perdu commençant par SND.UNA DOIT être retransmis et cwnd réglé à ssthresh plus 3*SMSS. Cela "gonfle" artificiellement la fenêtre d'encombrement du nombre de segments (trois) qui ont quitté le réseau et que le receveur a mis dans sa mémoire tampon.

4. Pour chaque ACK dupliqué supplémentaire reçu (après le troisième), cwnd DOIT être incrémenté de SMSS. Cela gonfle artificiellement la fenêtre d'encombrement afin de refléter le segment supplémentaire qui a quitté le réseau.

Note : [SCWA99] expose une attaque fondée sur le receveur par laquelle de nombreux ACK dupliqués bogués sont envoyés à l'envoyeur des données afin de gonfler artificiellement cwnd et causer l'utilisation d'un taux d'envoi plus élevé qu'approprié. Un TCP PEUT donc limiter au nombre de segments en cours (ou à un nombre approchant) le nombre de fois où cwnd est gonflé artificiellement durant la récupération de perte.

Note : Lorsque un mécanisme évolué de récupération de perte (tel qu'évoqué au paragraphe 4.3) n'est pas utilisé, cette augmentation de Taille en cours peut être cause que l'équation (4) gonfle légèrement cwnd et ssthresh, car certains des segments entre SND.UNA et SND.NXT sont supposés avoir quitté le réseau mais être encore reflétés dans Taille en cours.

5. Lorsque des données non envoyées précédemment sont disponibles et que la nouvelle valeur de cwnd et la fenêtre annoncée du receveur le permettent, un TCP DEVRAIT envoyer 1*SMSS octets de données non envoyées précédemment.

6. Lorsque arrive le prochain ACK qui accuse réception de données non acquittées précédemment, un TCP DOIT régler cwnd à ssthresh (la valeur établie à l'étape 2). On appelle cela "dégonfler" la fenêtre.

Cet ACK devrait être l'accusé de réception choisi par la retransmission de l'étape 3, un RTT après la retransmission (bien qu'il puisse arriver plus tôt en présence de livraison de segments significativement déclassés chez le receveur). De plus, cet ACK devrait accuser réception de tous les segments intermédiaires envoyés entre le segment perdu et la réception du troisième ACK dupliqué, si aucun d'eux n'a été perdu.

Note : Il est connu que cet algorithme ne récupère généralement pas efficacement de pertes multiples dans un seul envoi de paquets [FF96]. Le paragraphe 4.3 ci-dessous traite de tels cas.

4. Considérations supplémentaires

4.1 Redémarrage des connexions inactives

Un problème connu des algorithmes TCP de contrôle d'encombrement décrits ci-dessus est qu'ils permettent qu'une salve potentiellement inappropriée de trafic soit transmise après que TCP a été inactif pendant une période relativement longue. Après une période d'inactivité, TCP ne peut pas utiliser l'horloge d'ACK pour injecter de nouveaux segments dans le réseau, car tous les ACK ont quitté le réseau. Donc, comme spécifié ci-dessus, TCP peut envoyer une salve de taille cwnd au taux de ligne dans le réseau après une période d'inactivité. De plus, les conditions changeantes du réseau peuvent avoir rendue inappropriée, après une longue période d'inactivité, la notion de capacité disponible de bout en bout du réseau entre deux points d'extrémité telle qu'estimée par cwnd, de TCP.

[Jac88] recommande que TCP utilise le démarrage lent pour redémarrer la transmission après une relativement longue période d'inactivité. Le démarrage lent sert à redémarrer l'horloge d'ACK, tout comme il le fait au début d'un transfert. Ce mécanisme a été largement déployé de la façon suivante : lorsque TCP n'a pas reçu un segment pendant plus d'une temporisation de retransmission, cwnd est réduit à la valeur de la fenêtre de redémarrage (RW) avant que la transmission commence.

Pour les besoins de la présente norme, on définit RW = min(IW,cwnd).

Utiliser l'heure de réception du dernier segment pour déterminer de diminuer ou non cwnd peut échouer à dégonfler cwnd dans le cas courant de connexions HTTP persistantes [HTH98]. Dans ce cas, un serveur de la Toile reçoit une demande avant de transmettre les données au client. La réception de la demande fait échouer la vérification de l'inactivité de la connexion, et permet à TCP de commencer la transmission avec un cwnd éventuellement d'une taille inappropriée.

Donc, un TCP DEVRAIT régler cwnd à pas plus que RW avant de commencer la transmission si le TCP n'a pas envoyé de données dans un intervalle excédant la temporisation de retransmission.

4.2 Génération des accusés de réception

L'algorithme d'ACK retardé spécifié dans la [RFC1122] DEVRAIT être utilisé par un receveur TCP. Lors de l'utilisation des ACK retardés, un receveur TCP NE DOIT PAS retarder excessivement les accusés de réception. Précisément, un ACK DEVRAIT être généré au moins tous les seconds segments de pleine taille, et DOIT être généré dans les 500 ms de l'arrivée du premier paquet non acquitté.

L'exigence qu'un ACK "DEVRAIT" être généré au moins tous les seconds segments de pleine taille est citée dans la [RFC1122] comme un DEVRAIT à un endroit et comme un DOIT à un autre. On déclare ici sans ambiguïté un DEVRAIT. On souligne aussi que ce DEVRAIT signifie qu'une mise en œuvre ne devrait bien sûr dévier de cette exigence qu'après un examen attentif des ses implications. Voir la discussion de "violation d'ACK étiré" dans la [RFC2525] et les références qui y sont contenues pour une discussion des possibles problèmes de performances découlant de la génération des ACK moins fréquemment que tous les seconds segments de pleine taille.

Dans certains cas, l'envoyeur et le receveur peuvent ne pas s'accorder sur ce qui constitue un segment de pleine taille. Une mise en œuvre est réputée se conformer à cette exigence si elle envoie au moins un accusé de réception chaque fois qu'elle reçoit 2*RMSS octets de nouvelles données de la part de l'envoyeur, où RMSS est la taille maximum de segment spécifiée par le receveur à l'envoyeur (ou la valeur par défaut de 536 octets, selon la [RFC1122], si le receveur ne spécifie pas une option MSS durant l'établissement de la connexion). L'envoyeur peut être forcé d'utiliser une taille de segment inférieure à RMSS à cause de l'unité de transmission maximum (MTU), de l'algorithme de découverte de la MTU de chemin, ou d'autres facteurs. Par exemple, considérons le cas où le receveur annonce une RMSS de X octets mais où l'envoyeur finit par utiliser une taille de segment de Y octets (Y < X) due à la découverte de la MTU de chemin (ou de la taille de la MTU de l'envoyeur). Le receveur va générer des ACK étirés si il attend qu'arrivent 2*X octets avant qu'un ACK soit envoyé. Cela va clairement prendre plus de deux segments de taille Y octets. Donc, bien qu'un algorithme spécifique ne soit pas défini, il est souhaitable que les receveurs tentent de prévenir cette situation, par exemple, en accusant réception au moins tous les seconds segments, sans considération de taille. Finalement, on répète qu'un ACK NE DOIT PAS être retardé de plus de 500 ms en attendant l'arrivée d'un second segment de pleine taille.

Les segments de données déclassés DEVRAIENT être acquittés immédiatement, afin d'accélérer la récupération de perte. Pour déclencher l'algorithme de retransmission rapide, le receveur DEVRAIT envoyer un ACK dupliqué immédiat lorsque il reçoit un segment de données sur un trou dans l'espace des numéros de séquence. Pour fournir un retour aux envoyeurs qui récupèrent de pertes, le receveur DEVRAIT envoyer un ACK immédiat lorsque il reçoit un segment de données remplissant tout ou partie d'un trou dans l'espace des numéros de séquence.

Un receveur TCP NE DOIT PAS générer plus d'un ACK pour chaque segment entrant, autre que pour la mise à jour de la fenêtre offerte lorsque l'application receveuse consomme de nouvelles données (voir la [RFC0813] et l'entrée "Gestion de la fenêtre" du paragraphe 3.7 de la [RFC0793]).

4.3 Mécanismes de récupération de perte

Un certain nombre d'algorithmes de récupération de perte qui augmentent la retransmission rapide et la récupération rapide ont été suggérés par les chercheurs sur TCP et spécifiés dans la série des RFC. Alors que certains de ces algorithmes se fondent sur l'option TCP d'accusé de réception sélectif (SACK) [RFC2018], tels que [FF96], [MM96a], [MM96b], et la [RFC3517], d'autres n'exigent pas de SACK, tels que [Hoe96], [FF96], et la [RFC3782]. Les algorithmes non SACK utilisent les "accusés de réception partiels" (des ACK qui couvrent les données non acquittées précédemment, mais pas toutes les données en cours lorsque la perte a été détectée) pour déclencher les retransmissions. Bien que le présent document ne normalise aucun des algorithmes spécifiques qui peuvent améliorer la retransmission rapide/récupération rapide, ces algorithmes améliorés sont implicitement permis, tant qu'ils suivent les principes généraux des quatre algorithmes de base évoqués plus haut.

C'est-à-dire que lorsque est détectée la première perte dans une fenêtre de données, ssthresh DOIT être réglé à pas plus que la valeur donnée par l'équation (4). Ensuite, jusqu'à ce que tous les segments perdus dans la fenêtre de données en question

soient réparés, le nombre de segments transmis dans chaque RTT DOIT être pas plus de la moitié du nombre de segments en cours lorsque la perte a été détectée. Enfin, après que toutes les pertes dans la fenêtre de segments concernée ont réussi à être retransmises, cwnd DOIT être réglé à pas plus que ssthresh et l'évitement d'encombrement DOIT être utilisé pour augmenter encore cwnd. La perte dans deux fenêtres de données successives, ou la perte d'une retransmission, devrait être considérée comme deux indications d'encombrement et donc, cwnd (et ssthresh) DOIT être diminué deux fois dans ce cas.

On RECOMMANDE que les mises en œuvre de TCP emploient une forme évoluée de récupération de perte qui puisse s'accommoder de plusieurs pertes dans une fenêtre de données. Les algorithmes décrits dans les [RFC3782] et [RFC3517] se conforment aux principes généraux évoqués ci-dessus. On note qu'alors que ceux-ci ne sont pas les deux seuls algorithmes qui se conforment aux principes généraux ci-dessus, ces deux algorithmes ont été adoptés par la communauté et sont actuellement en cours de normalisation.

5. Considérations pour la sécurité

Le présent document exige d'une mise en œuvre de TCP qu'elle diminue son taux d'envoi en présence de fins de temporisation de retransmission et de l'arrivée d'accusés de réception dupliqués. Un attaquant peut donc dégrader les performances d'une connexion TCP soit en causant la perte de paquets de données ou de leurs accusés de réception, soit en falsifiant un nombre excessif d'accusés de réception dupliqués.

En réponse à l'attaque de division d'ACK mentionnée dans [SCWA99], le présent document RECOMMANDE d'augmenter la fenêtre d'encombrement sur la base du nombre d'octets nouveaux acquittés dans chaque ACK arrivant plutôt que d'une constante particulière sur chaque ACK arrivant (comme mentionné au paragraphe 3.1).

L'Internet s'appuie, dans une mesure considérable, sur la mise en œuvre correcte de ces algorithmes afin de préserver la stabilité du réseau et éviter l'écroulement par encombrement. Un attaquant pourrait faire que les points d'extrémité TCP répondent de façon plus agressive en présence d'encombrement en envoyant un nombre excessif de faux accusés de réception dupliqués ou de faux accusés de réception pour de nouvelles données. On peut concevoir qu'une telle attaque pourrait mettre une portion du réseau dans une situation d'écroulement par encombrement.

6. Changements entre la RFC 2001 et la RFC 2581

La [RFC2001] a été largement réécrite et il n'est pas possible de faire une liste de tous les changements survenus entre la [RFC2001] et la [RFC2581]. L'intention de la [RFC2581] était de ne pas changer les recommandations données dans la [RFC2001], mais de mieux préciser les cas qui n'étaient pas discutés en détail dans la [RFC2001]. Précisément, la [RFC2581] suggérait ce que les connexions TCP devraient faire après une période relativement longue d'inactivité, ainsi que spécifiait et clarifiait certaines des questions relatives à la génération des ACK TCP. Finalement, la limite supérieure admissible pour la fenêtre d'encombrement initiale a été relevée de un à deux segments.

7. Changements par rapport à la RFC 2581

Une définition spécifique de "accusé de réception dupliqué" a été ajoutée, sur la base de la définition utilisée par BSD TCP.

Le document note maintenant que ce qu'il faut faire des ACK dupliqués après l'arrivée à expiration du temporisateur de retransmission reste à définir et est explicitement non spécifié dans ce document.

Les exigences de fenêtre initiale ont été changées pour permettre une plus grande fenêtre initiale comme spécifié dans la [RFC3390]. De plus, sont détaillées les étapes à suivre lorsque une fenêtre initiale trop grande est découverte suite à la découverte de la MTU de chemin [RFC1191].

La valeur initiale recommandée pour ssthresh a été changée pour dire qu'elle DEVRAIT être arbitrairement élevée, alors qu'il était dit précédemment PEUT. Cela donne des lignes directrices supplémentaires pour la mise en œuvre.

Durant le démarrage lent, l'usage du comptage d'octets appropriés de la [RFC3465] avec L=1*SMSS est explicitement recommandé. La méthode d'augmentation de cwnd donnée dans la [RFC2581] est toujours explicitement permise. Le comptage d'octets durant l'évitement d'encombrement est aussi recommandé, alors que la méthode de la [RFC2581] et d'autres méthodes sûres est toujours permis.

Le traitement de ssthresh sur la fin de temporisation de retransmission a été précisé. En particulier, ssthresh doit être réglé à

la moitié de Taille en cours sur la première retransmission d'un segment particulier et ensuite est constant sur les retransmissions suivantes du même segment.

La description de la retransmission rapide et de la récupération rapide a été précisée, et l'utilisation de la transmission limitée de la [RFC3042] est maintenant recommandée.

Les mises en œuvre de TCP PEUVENT maintenant limiter le nombre d'ACK dupliqués qui gonflent artificiellement cwnd durant la récupération de perte au nombre de segments en cours pour éviter l'attaque de faux ACK dupliqués décrite dans [SCWA99].

La fenêtre de redémarrage a été changée de IW en min(IW,cwnd). Ce comportement était décrit comme "expérimental" dans la [RFC2581].

Il est maintenant recommandé que les mises en œuvre de TCP utilisent un algorithme évolué de récupération de perte se conformant aux principes évoqués dans le présent document.

Les considérations de sécurité ont été mises à jour pour discuter de la division d'ACK et recommander le comptage des octets comme moyen de contrer cette attaque.

8. Remerciements

Les algorithmes centraux décrits ont été développés par Van Jacobson [Jac88], [Jac90]. De plus, la transmission limitée [RFC3042] a été développée en conjonction avec Hari Balakrishnan et Sally Floyd. La taille de fenêtre d'encombrement initiale spécifiée dans ce document est le résultat d'un travail avec Sally Floyd et Craig Partridge [RFC2414], [RFC3390].

W. Richard ("Rich") Stevens a écrit la première version de ce document [RFC2001] et il est coauteur de la seconde version [RFC2581]. La présente version a beaucoup bénéficié de la clarté et de l'intelligence de ses descriptions, et nous remercions Rich de ses contributions pour élucider le contrôle d'encombrement de TCP, ainsi que en nous aidant à comprendre de nombreuses questions relatives au réseautage.

Nous souhaitons souligner que les insuffisances et erreurs du présent document sont de la seule responsabilité des auteurs actuels.

Une partie du texte de ce document est tirée de "TCP/IP Illustrated, Volume 1: The Protocols" de W. Richard Stevens (Addison-Wesley, 1994) et "TCP/IP Illustrated, Volume 2: The Implementation" de Gary R. Wright et W. Richard Stevens (Addison-Wesley, 1995). Ce matériel est utilisé avec la permission de Addison-Wesley.

Anil Agarwal, Steve Arden, Neal Cardwell, Noritoshi Demizu, Gorry Fairhurst, Kevin Fall, John Heffner, Alfred Hoenes, Sally Floyd, Reiner Ludwig, Matt Mathis, Craig Partridge, et Joe Touch ont contribué par nombre de suggestions utiles.

9. Références

9.1 Références normatives

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

[RFC1122] R. Braden, "Exigences pour les hôtes Internet – couches de communication", STD 3, octobre 1989. *(MàJ par la* RFC6633)

[RFC1191] J. Mogul et S. Deering, "Découverte de la MTU de chemin", novembre 1990.

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

9.2 Références pour information

[CJ89] Chiu, D. et R. Jain, "Analysis of the Increase/Decrease Algorithms for Congestion Avoidance in Computer Networks", Journal of Computer Networks et ISDN Systems, vol. 17, n°1, pp. 1-14, juin 1989.

[FF96] Fall, K. et S. Floyd, "Simulation-based Comparisons of Tahoe, Reno et SACK TCP", Computer Communication Review, juillet 1996, ftp://ftp.ee.lbl.gov/papers/sacks.ps.Z .

[Hoe96] Hoe, J., "Improving the Start-up Behavior of a Congestion Control Scheme for TCP", In ACM SIGCOMM, août 1996.

[HTH98] Hughes, A., Touch, J., et J. Heidemann, "Issues in TCP Slow-Start Restart After Idle", Non publié, mars 1998.

[Jac88] Jacobson, V., "Congestion Avoidance et Control", Computer Communication Review, vol. 18, no. 4, pp. 314-329, août 1988. ftp://ftp.ee.lbl.gov/papers/congavoid.ps.Z .

[Jac90] Jacobson, V., "Modified TCP Congestion Avoidance Algorithm", Liste de diffusion end2end-interest, 30 avril 1990. ftp://ftp.isi.edu/end2end/end2end-interest-1990.mail .

[MM96a] Mathis, M. et J. Mahdavi, "Forward Acknowledgment: Refining TCP Congestion Control", Proceedings of SIGCOMM'96, août 1996, Stanford, CA. Disponible à http://www.psc.edu/networking/papers/papers.html

[MM96b] Mathis, M. et J. Mahdavi, "TCP Rate-Halving with Bounding Parameters", Rapport technique. Disponible à http://www.psc.edu/networking/papers/FACKnotes/current .

[Pax97] Paxson, V., "End-to-End Internet Packet Dynamics", Proceedings of SIGCOMM '97, Cannes, France, septembre 1997.

[RFC0813] D. Clark, "Fenêtre et stratégie d'accusé de réception dans TCP", juillet 1982.

[RFC2001] W. Stevens, "Algorithmes de démarrage lent, d'évitement d'encombrement, de retransmission rapide et de récupération rapide dans TCP", janvier 1997. (*Obsolète, voir* RFC2581) *(P.S.)*

[RFC2018] M. Mathis et autres, "Options d'accusé de réception sélectif sur TCP", octobre 1996. (*Remplace* RFC1072) *(P.S.)*

[RFC2414] M. Allman, S. Floyd, C. Partridge, "Accroissement de la fenêtre initiale de TCP", septembre 1998. *(Expérimentale)* (*Obsolète, voir* RFC3390)

[RFC2525] V. Paxson et autres, "Problèmes connus de mise en œuvre de TCP", mars 1999. *(Information)*

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

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

[RFC2988] V. Paxson, M. Allman, "Calcul du temporisateur de retransmission de TCP", novembre 2000. *(voir 6298)*

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

[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.)*

[RFC3465] M. Allman, "Contrôle d'encombrement sur TCP avec compte d'octets approprié (ABC)", février 2003. *(Expérimentale)*

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

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

[RFC4821] M. Mathis, J. Heffner, "Découverte de la MTU de chemin de couche de mise en paquet", mars 2007. *(P.S.)*

[SCWA99] Savage, S., Cardwell, N., Wetherall, D., et T. Anderson, "TCP Congestion Control With a Misbehaving Receiver", ACM Computer Communication Review, 29(5), octobre 1999.

[Ste94] Stevens, W., "TCP/IP Illustrated, Volume 1: The Protocols", Addison-Wesley, 1994.

[WS95] Wright, G. et W. Stevens, "TCP/IP Illustrated, Volume 2: The Implementation", Addison-Wesley, 1995.

Adresse des auteurs

Mark Allman	Vern Paxson
International Computer Science Institute (ICSI)	International Computer Science Institute (ICSI)
1947 Center Street	1947 Center Street
Suite 600	Suite 600
Berkeley, CA 94704-1198	Berkeley, CA 94704-1198
téléphone : +1 440 235 1792	téléphone : +1 510/642-4274 x302
mél : mallman@icir.org	mél : vern@icir.org
http://www.icir.org/mallman/	http://www.icir.org/vern/

Ethan Blanton
Purdue University Computer Sciences
305 North University Street
West Lafayette, IN 47907
mél: eblanton@cs.purdue.edu
http://www.cs.purdue.edu/homes/eblanton/