Groupe de travail Réseau
Request for Comments : 2861
Catégorie: Expérimentale
Traduction Claude Brière de L'Isle

M. Handley, J. Padhye & S. Floyd
ACIRI
juin 2000

Validation de la fenêtre d'encombrement TCP

Résumé
Le fenêtre d'encombrement de TCP contrôle le nombre de paquets qu'un flux TCP peut avoir dans le réseau à tout moment. Cependant, de longues périodes lorsque l'envoyeur est au repos ou limité par l'application peuvent conduire à l'invalidation de la fenêtre d'encombrement, en ce que la fenêtre d'encombrement ne reflète plus les informations actuelles sur l'état du réseau. Le présent document décrit une simple modification des algorithmes de contrôle d'encombrement TCP pour périmer la fenêtre d'encombrement (cwnd) après la transition d'une période suffisamment longue limitée par l'application, tout en utilisant le seuil de démarrage lent (ssthresh) pour sauvegarder les informations sur la valeur précédente de la fenêtre d'encombrement.

Une fenêtre d'encombrement invalide peut aussi résulter de l'augmentation de la fenêtre d'encombrement (c'est-à-dire, dans les phases de démarrage lent ou d'évitement d'encombrement de TCP) durant les périodes limitées par l'application, lorsque la valeur précédente de la fenêtre d'encombrement peut n'avoir jamais été pleinement utilisée. On propose que l'envoyeur TCP n'augmente pas la fenêtre d'encombrement lorsque l'envoyeur TCP a été limité par l'application (et n'a donc pas pleinement utilisé la fenêtre d'encombrement actuelle). On a exploré ces algorithmes avec des simulations et des expériences d'une mise en œuvre dans FreeBSD.

1. Conventions et acronymes

Les mots clés DOIT, NE DOIT PAS, EXIGE, DEVRA, NE DEVRA PAS, DEVRAIT, NE DEVRAIT PAS, RECOMMANDE, PEUT, et FACULTATIF, lorsque ils apparaissent dans le présent document, sont à interpréter comme décrit dans la [RFC2119].

2. Introduction

La fenêtre d'encombrement TCP contrôle le nombre de paquets qu'un flux TCP peut avoir dans le réseau à tout moment. La fenêtre d'encombrement est réglée en utilisant un mécanisme d'augmentation additive, diminution multiplicative (AIMD, *Additive-Increase, Multiplicative-Decrease*) qui vérifie la bande passante disponible, s'adaptant de façon dynamique aux conditions changeantes du réseau. Ce mécanisme AIMD fonctionne bien lorsque l'envoyeur a continuellement des données à envoyer, comme c'est normalement le cas pour TCP utilisé pour un transfert de données en vrac. À l'opposé, pour TCP utilisé avec des applications telnet, l'envoyeur des données a souvent peu ou pas de données à envoyer, et le taux d'envoi est souvent déterminé par le taux auquel les données sont générées par l'usager. Avec l'arrivée de la Toile, y compris des développements tels que des envoyeurs TCP avec des données créées de façon dynamique et HTTP 1.1 avec une connexion TCP persistante, l'interaction entre des périodes limitées par l'application (lorsque l'envoyeur envoie moins que ce qui est permis par les fenêtres d'encombrement ou de receveur) et des périodes limitées par le réseau (lorsque l'envoyeur est limité par la fenêtre TCP) devient de plus en plus importante. Plus précisément, on définit une période limitée par le réseau comme toute période où l'envoyeur envoie une pleine fenêtre de données.

De longues périodes où l'envoyeur est limité par l'application peuvent conduire à l'invalidation de la fenêtre d'encombrement. Durant les périodes où l'envoyeur TCP est limité par le réseau, la valeur de la fenêtre d'encombrement est répétitivement "revalidée" par la transmission réussie d'une fenêtre de données sans perte. Lorsque l'envoyeur TCP est limité par le réseau, il y a un flux entrant d'accusés de réception qui "rythment" les nouvelles données, donnant une preuve concrète de la bande passante disponible récente dans le réseau. À l'opposé, durant les périodes où l'envoyeur TCP est limité par l'application,

l'estimation de la capacité disponible représentée par la fenêtre d'encombrement peut devenir beaucoup moins précise au fil du temps. En particulier, la capacité qui a été utilisée par la connexion limitée par le réseau peut maintenant être utilisée par d'autres trafics.

Les mises en œuvre TCP courantes ont une gamme de comportements pour redémarrer après une période d'inactivité. Certaines mises en œuvre TCP courantes font un démarrage lent après une période d'inactivité plus longue que l'estimation du temporisateur de retransmission (RTO, *Retransmission TimeOut*) comme suggéré dans la [RFC2581] et dans l'appendice de [VJ88], tandis que d'autres mises en œuvre ne réduisent pas leur fenêtre d'encombrement après une période d'inactivité. La [RFC2581] recommande ce qui suit : "un TCP DEVRAIT régler cwnd à pas plus que RW (la fenêtre initiale) avant de commencer la transmission si le TCP n'a pas envoyé de données dans un intervalle excédant la temporisation de retransmission". Une proposition que TCP fasse un démarrage lent après une période d'inactivité a aussi été discutée dans [HTH98]. La question de la validation des informations d'encombrement durant les périodes d'inactivité a aussi été traitée dans des contextes autres que TCP et IP, par exemple dans les mécanismes "Utilisé ou perdu" pour les réseaux ATM [J96,J95].

Pour traiter de la revalidation de la fenêtre d'encombrement après une période limitée par l'application, on propose une simple modification aux algorithmes TCP de contrôle d'encombrement pour réduire la fenêtre d'encombrement cwnd après la transition d'une période suffisamment longue limitée par l'application (c'est--dire, au moins un délai d'aller-retour) à une période limitée par le réseau. En particulier, on propose qu'après la période d'inactivité, l'envoyeur TCP devrait réduire sa fenêtre d'encombrement de moitié à chaque RTT où le flux est resté inactif.

Lorsque la fenêtre d'encombrement est réduite, le seuil de démarrage lent ssthresh reste comme "souvenir" de la récente fenêtre d'encombrement. Précisément, ssthresh n'est jamais diminué lorsque cwnd est réduit après une période limitée par l'application ; avant que cwnd soit réduit, ssthresh est réglé au maximum de sa valeur courante, et à mi chemin entre l'ancienne et la nouvelle valeur de cwnd. Cette utilisation de ssthresh permet à un envoyeur TCP qui augmente son taux d'envoi après une période limitée par l'application de faire rapidement un démarrage lent pour récupérer la plupart des valeurs précédentes de la fenêtre d'encombrement. Pour être plus précis, si ssthresh est de moins des 3/4 de cwnd lorsque la fenêtre d'encombrement est réduite après une période limitée par l'application, alors ssthresh est augmenté des 3/4 de la cwnd d'avant la réduction de la fenêtre d'encombrement.

Une fenêtre d'encombrement invalide apparaît aussi lorsque la fenêtre d'encombrement est augmentée (c'est-à-dire, dans les phases de démarrage lent ou d'évitement d'encombrement de TCP) durant les périodes limitées par l'application, lorsque la valeur précédente de la fenêtre d'encombrement pourrait n'avoir jamais été pleinement utilisée. Pour autant qu'on le sache, toutes les mises en œuvre TCP actuelles augmentent la fenêtre d'encombrement lorsque arrive un accusé de réception, si c'est permis par la fenêtre annoncée du receveur et par l'algorithme de démarrage lent ou d'augmentation de fenêtre d'évitement d'encombrement, sans vérifier pour voir si la valeur précédente de la fenêtre d'encombrement a en fait été utilisée. Le présent document propose que l'algorithme d'augmentation de fenêtre ne soit pas invoqué durant les périodes limitées par l'application [MSML99]. En particulier, l'envoyeur TCP ne devrait pas augmenter la fenêtre d'encombrement lorsque l'envoyeur TCP a été limité par l'application (et n'a donc pas pleinement utilisé la fenêtre d'encombrement actuelle). Cette restriction empêche que la fenêtre d'encombrement devienne arbitrairement grande, en l'absence de preuve que la fenêtre d'encombrement puisse être acceptée par le réseau. D'après le paragraphe 5.2 de [MSML99] : "Cette restriction assure que [cwnd] ne croît qu'autant que TCP réussit en fait à injecter assez de données dans le réseau pour tester le chemin."

Un problème quelque peu orthogonal associé à la maintenance d'une grande fenêtre d'encombrement après une période limitée par l'application est que l'envoyeur, avec une soudaine grande quantité de données à envoyer après une période de repos, peut immédiatement envoyer une pleine fenêtre d'encombrement de paquets dos à dos. Ce problème de l'envoi de grosses salves de paquets dos à dos peut effectivement être traité en utilisant le ralentissement fondé sur le taux d'envoi (RBP, *rate-based pacing*) [VH97]), ou en utilisant un contrôle de taille maximum de salve [FF96]. On soutiendra que même avec des mécanismes de limitation de l'envoi de paquets dos à dos ou de ralentissement de l'envoi de paquets sur la période d'un aller-retour, une vieille fenêtre d'encombrement qui n'a pas été pleinement utilisée pendant un certain temps ne peut pas être une indication fiable de la bande passante actuellement disponible pour ce flux. On soutiendra que les mécanismes pour ralentir la sortie des paquets permise par la fenêtre d'encombrement sont largement orthogonaux aux algorithmes utilisés pour déterminer la taille appropriée de la fenêtre d'encombrement.

3. Description

Lorsque un envoyeur TCP a suffisamment de données disponibles pour remplir la capacité réseau disponible pour ce flux, cwnd et ssthresh sont réglés aux valeurs appropriées pour les conditions du réseau. Lorsque un envoyeur TCP arrête d'envoyer, le flux arrête d'échantillonner les conditions du réseau, de sorte que la valeur de la fenêtre d'encombrement peut devenir inappropriée. On pense que le comportement prudent correct dans ces circonstances est de diminuer la fenêtre d'encombrement

de moitié pour chaque RTT où le flux reste inactif. La valeur de moitié est un chiffre très prudent fondé sur la façon dont la diminution multiplicative aurait rapidement diminué la fenêtre en présence de pertes.

Une autre possibilité est que l'envoyeur ne puisse pas arrêter d'envoyer, mais puisse devenir limité par l'application plutôt que limité par le réseau, et offre moins de données au réseau que ce que la fenêtre d'encombrement permet d'envoyer. Dans ce cas, le flux TCP échantillonne toujours les conditions du réseau, mais n'offre pas suffisamment de trafic pour être sûr qu'il y a encore une capacité suffisante dans le réseau pour ce flux pour envoyer une pleine fenêtre d'encombrement. Dans ces circonstances, on pense que le comportement prudent correct est que l'envoyeur garde trace de la quantité maximum de fenêtre d'encombrement utilisée durant chaque RTT, et de diminuer la fenêtre d'encombrement à chaque RTT à mi chemin entre la valeur actuelle de cwnd et la valeur maximum utilisée.

Avant que la fenêtre d'encombrement soit réduite, ssthresh est réglé au maximum de sa valeur actuelle et des 3/4 de cwnd. Si l'envoyeur a alors plus de données à envoyer que ce que permet la cwnd diminuée, le TCP va faire un démarrage lent (effectuer une augmentation exponentielle) d'au moins la moitié de la vieille valeur de cwnd.

La justification de cette valeur de "3/4 de la cwnd" est que 3/4 de la cwnd est une estimation prudente de la valeur moyenne récente de la fenêtre d'encombrement, et le TCP devrait être capable en toute sécurité de faire un démarrage lent au moins jusqu'à ce point. Pour un TCP en régime permanent qui a réduit sa fenêtre d'encombrement chaque fois que celle-ci a atteint une valeur maximum 'maxwin', la fenêtre d'encombrement moyenne a été de 3/4 maxwin. En moyenne, lorsque la connexion devient limitée par l'application, cwnd va être 3/4 maxwin, et dans ce cas, cwnd elle-même représente la valeur moyenne de la fenêtre d'encombrement. Cependant, si il se trouve que la connexion devient limitée par l'application lorsque cwnd égale maxwin, alors la valeur moyenne de la fenêtre d'encombrement est donnée par 3/4 cwnd.

Une autre possibilité serait de régler ssthresh au maximum de la valeur courante de ssthresh et de la vieille valeur de cwnd, permettant au TCP de faire le démarrage lent sur tout le chemin vers la vieille valeur de cwnd. D'autres expériences peuvent être utilisées pour évaluer ces deux options de réglage de ssthresh.

Pour la question distincte de l'augmentation de la fenêtre d'encombrement en réponse à un accusé de réception, on pense que le comportement correct est que l'envoyeur n'augmente la fenêtre d'encombrement que si la fenêtre était pleine lorsque l'accusé de réception est arrivé.

On appelle cet ensemble de modifications à TCP la validation de fenêtre d'encombrement (CWV, *Congestion Window Validation*) parce qu'elles se rapportent à la vérification que la fenêtre d'encombrement est toujours un reflet valide de l'état actuel du réseau tel que perçu par la connexion.

3.1 Algorithme de base pour réduire la fenêtre d'encombrement

Une question clé de l'algorithme CWV est de déterminer comment appliquer les lignes directrices de la réduction de fenêtre d'encombrement une fois pour chaque délai d'aller-retour où le flux est limité par l'application. On utilise le temporisateur de retransmission (RTO) de TCP comme limite supérieure raisonnable du délai d'aller-retour, et on réduit la fenêtre d'encombrement en gros une fois par RTO.

Cet algorithme de base pourrait être mis en œuvre comme suit dans TCP : lorsque TCP envoie un nouveau paquet, il vérifie pour voir si plus de RTO secondes se sont écoulées depuis l'envoi du paquet précédent. Si RTO secondes se sont écoulées, ssthresh est réglé au maximum de 3/4 cwnd et de la valeur actuelle de ssthresh, et ensuite la fenêtre d'encombrement est divisée par deux pour chaque RTO qui s'est écoulé depuis l'envoi du paquet précédent. De plus, T_prev est réglé à l'heure actuelle, et W_used est remis à zéro. T_prev sera utilisé pour déterminer le temps écoulé depuis la dernière fois que l'envoyeur a été limité par le réseau ou a réduit cwnd après une période d'inactivité. Lorsque l'envoyeur est limité par l'application, W_used contient la fenêtre d'encombrement maximum réellement utilisée depuis la dernière fois que l'envoyeur a été limité par le réseau.

Le mécanisme pour déterminer le nombre de RTO dans la plus récente période d'inactivité pourrait aussi être mis en œuvre en utilisant un temporisateur qui arrive à expiration tous les RTO après l'envoi du dernier paquet au lieu d'une vérification par paquet ; les contraintes d'efficacité sur les différents systèmes d'exploitation peuvent dicter ce qu'il est plus efficace de mettre en œuvre.

Après que TCP a envoyé un paquet, il vérifie aussi pour voir si ce paquet a rempli la fenêtre d'encombrement. Si c'est le cas, l'envoyeur est limité par le réseau, et règle la variable T_prev à l'heure actuelle de l'horloge TCP, et la variable W_used à zéro.

Lorsque TCP envoie un paquet qui ne remplit pas la fenêtre d'encombrement, et que la file d'attente d'envoi de TCP est vide, l'envoyeur est alors limité par l'application. L'envoyeur vérifie pour voir si la quantité de données non acquittées est supérieure à W_used ; si il en est ainsi, W_used est réglé à la quantité de données non acquittées. De plus, TCP vérifie pour voir si le temps écoulé depuis T_prev est supérieur à RTO. Si il l'est, le TCP n'a pas réduit sa fenêtre d'encombrement suite à une période d'inactivité. Le TCP a été limité par l'application plutôt que limité par le réseau pendant au moins un intervalle entier de RTO, mais moins que deux intervalles RTO. Dans ce cas, TCP règle ssthresh au maximum de 3/4 cwnd et de la valeur actuelle de ssthresh, et réduit sa fenêtre d'encombrement à (cwnd+W_used)/2. W_used est alors réglé à zéro, et T_prev est réglé à l'heure actuelle, de sorte qu'une autre réduction n'aura pas lieu avant qu'au moins une autre période de RTO soit écoulée. Donc, durant une période limitée par l'application, l'algorithme de CWV réduit la fenêtre d'encombrement une fois par RTO.

3.2 Pseudo-code pour réduire la fenêtre d'encombrement

Initialement :
T_last = tcpnow, T_prev = tcpnow, W_used = 0

Après l'envoi d'un segment de données :

```
Si tcpnow - T_last ≥ RTO
   (L'envoyeur a été inactif.)
   ssthresh =  max(ssthresh, 3*cwnd/4)
   Pour i=1  à (tcpnow - T_last)/RTO
      win =  min(cwnd, fenêtre maximum déclarée du receveur)
      cwnd =  max(win/2, MSS)
   T_prev = tcpnow
   W_used = 0

T_last = tcpnow

Si la fenêtre est pleine
   T_prev = tcpnow
   W_used = 0
Autrement
   Si il n'y a plus de données disponibles à l'envoi
      W_used =  max(W_used, quantité de données non acquittées)
      SI tcpnow - T_prev ≥ RTO
         (L'envoyeur a été limité par l'application.)
         ssthresh =  max(ssthresh, 3*cwnd/4)
         win =  min(cwnd, fenêtre maximum déclarée du receveur)
         cwnd = (win + W_used)/2
         T_prev = tcpnow
         W_used = 0
```

4. Simulations

La proposition de CWV a été mise en œuvre comme une option dans le simulateur de réseau NS [NS]. Les simulations dans la suite d'essais de validation pour CWV peuvent être lancées avec la commande "./test-all-tcp" dans le répertoire "tcl/test". Les simulations montrent l'utilisation de CWV pour réduire la fenêtre d'encombrement après une période où la connexion TCP a été limitée par l'application, et pour limiter l'augmentation de la fenêtre d'encombrement lorsque un transfert est limité par l'application. Comme l'illustrent les simulations, l'utilisation de ssthresh pour conserver l'historique de connexion est une partie critique de l'algorithme de validation de fenêtre d'encombrement. [HPF99] expose trois simulations plus en détail.

5. Expériences

On a mis en œuvre le mécanisme CWV dans l'application de TCP avec la machine FreeBSD 3.2. [HPF99] expose ces expériences plus en détail.

La première expérience examine les effets des mécanismes de validation de fenêtre d'encombrement pour limiter les augmentations de cwnd durant les périodes limitées par l'application. L'expérience a utilisé une connexion ssh réelle à travers une liaison modem émulée avec Dummynet [Dummynet]. La vitesse de la liaison est de 30 kbit/s et la liaison a des mémoires tampon de cinq paquets disponibles. La plupart des modems d'aujourd'hui ont une capacité de mémoire tampon supérieure à cela, mais les situations de limitation de mémoire tampon surviennent parfois avec des modems plus anciens. Dans la première moitié du transfert, l'usager envoie sa frappe sur la connexion. À environ la moitié du temps, l'usager propose l'envoi d'un fichier modérément grand, qui cause la transmission d'une grosse salve de trafic.

Pour le TCP non modifié, tout retour d'ACK durant la première partie du transfert résulte en une augmentation de la cwnd. Il en résulte que la grosse salve de données qui arrive de l'application à la couche transport est envoyée comme autant de paquets dos à dos, dont la plupart sont perdus et ensuite retransmis.

Pour le TCP modifié avec la validation de fenêtre d'encombrement, la fenêtre d'encombrement n'est pas augmentée lorsque la fenêtre n'est pas pleine, et a été diminuée durant les périodes limitées par l'application à un niveau plus proche de ce que l'usager a réellement utilisé. La salve de trafic est maintenant contrainte par la fenêtre d'encombrement, d'où il résulte un meilleur comportement du flux avec une perte minimale. Le résultat final est que le transfert est approximativement de 30 % plus rapide que celui sans CWV, du fait qu'il évite les temporisations de retransmission.

La seconde expérience utilise une réelle connexion ssh sur une connexion ppp numérotée réelle, où le banc de modems a plus de mémoire tampon. Pour le TCP non modifié, la salve initiale provenant du gros fichier ne cause pas de pertes, mais fait que le RTT augmente à approximativement 5 secondes, lorsque la connexion devient limitée par la fenêtre du receveur.

Pour le TCP modifié avec la validation de fenêtre d'encombrement, le flux se comporte beaucoup mieux, et ne produit pas de grosse salve de trafic. Dans ce cas, l'augmentation linéaire de cwnd résulte en une lente augmentation du RTT à mesure que la mémoire tampon se remplit.

Pour la seconde expérience, les deux TCP, modifié et non modifié, finissent par livrer les données précisément au même instant. Cela parce que la liaison a été pleinement utilisée dans les deux cas, car la mémoire tampon du modem est plus grande que la fenêtre du receveur. Il est clair qu'une mémoire tampon de modem de cette taille n'est pas souhaitable à cause de son effet sur le RTT des flux concurrents, mais elle est nécessaire avec les mises en œuvre TCP actuelles qui produisent des salves similaires à celles montrées dans le graphique plus haut.

6. Conclusions

Le présent document a présenté plusieurs algorithmes TCP pour la validation de fenêtre d'encombrement, à utiliser après une période de repos ou une période dans laquelle l'envoyeur a été limité par l'application, et avant une augmentation de la fenêtre d'encombrement. Le but de ces algorithmes est que la fenêtre d'encombrement TCP reflète les connaissances récentes de la connexion TCP sur l'état du chemin réseau, tout en gardant en même temps des souvenirs (c'est-à-dire, dans ssthresh) sur l'état précédent du chemin. On estime que ces modifications seront avantageuses à la fois pour le réseau et les flux TCP eux-mêmes, en empêchant les abandons inutiles de paquet dus à l'échec de l'envoyeur TCP à mettre à jour ces information (ou à l'absence de ces informations) sur les conditions actuelles du réseau. Des travaux futurs documenteront et étudieront le bénéfice apporté par ces algorithmes, en utilisant des simulations et des expériences. Des travaux futurs supplémentaires décriront une version plus complexe de l'algorithme de validation de fenêtre d'encombrement pour les mises en œuvre de TCP où l'envoyeur n'a pas une estimation précise du délai d'aller-retour TCP.

7. Références

[Dummynet] Luigi Rizzo, "Dummynet and Forward Error Correction", Freenix 98, juin 1998, New Orleans. URL " http://info.iet.unipi.it/~luigi/ip_dummynet/ ".

[FF96] Fall, K., and Floyd, S., "Simulation-based Comparisons of Tahoe, Reno, and SACK TCP", Computer Communication Review, V. 26 N. 3, juillet 1996, pp. 5-21. URL " http://www.aciri.org/floyd/papers.html ".

[HPF99] Mark Handley, Jitendra Padhye, Sally Floyd, "TCP Congestion Window Validation", UMass CMPSCI Technical Report 99-77, septembre 1999. URL " ftp://www-net.cs.umass.edu/pub/Handley99-tcpq-tr-99-77.ps.gz ".

[HTH98] Amy Hughes, Joe Touch, John Heidemann, "Issues in TCP Slow-Start Restart After Idle", non publié.

[J88] Jacobson, V., "Congestion Avoidance and Control", à l'origine dans Proceedings of SIGCOMM '88 (Palo Alto, CA, août 1988) et révisé en 1992. URL " http://www-nrg.ee.lbl.gov/nrg-papers.html ".

[JKBFL96] Raj Jain, Shiv Kalyanaraman, Rohit Goyal, Sonia Fahmy, and Fang Lu, Comments on "Use-it or Lose-it", ATM

Forum Document Number : ATM Forum/96-0178, URL " http://www.netlab.ohio-state.edu/~jain/atmf/af_rl5b2.htm ".

[JKGFL95] R. Jain, S. Kalyanaraman, R. Goyal, S. Fahmy, and F. Lu, "A Fix for Source End System Rule 5", AF-TM 95-1660, décembre 1995, URL " http://www.netlab.ohio-state.edu/~jain/atmf/af_rl52.htm ".

[MSML99] Matt Mathis, Jeff Semke, Jamshid Mahdavi, and Kevin Lahey, "The Rate-Halving Algorithm for TCP Congestion Control", juin 1999. URL " http://www.psc.edu/networking/ftp/papers/draft-ratehalving.txt ".

[NS] NS, "the UCB/LBNL/VINT Network Simulator". URL " http://www-mash.cs.berkeley.edu/ns/ ".

[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)

[VH97] Vikram Visweswaraiah and John Heidemann. "Improving Restart of Idle TCP Connections", Rapport technique 97-661, University of Southern California, novembre 1997.

8. Considérations pour la sécurité

Les considérations générales pour la sécurité concernant le contrôle d'encombrement TCP sont discutées dans la RFC2581. Le présent document décrit un algorithme pour un aspect de ces procédures de contrôle d'encombrement, et donc les considérations décrites dans la RFC2581 s'appliquent aussi à cet algorithme. Il n'y a pas de problème de sécurité supplémentaire connu pour cet algorithme spécifique.

9. Adresse des auteurs

Mark Handley
(ACIRI)
téléphone : +1 510 666 2946
mél : mjh@aciri.org
URL : http://www.aciri.org/mjh/

Jitendra Padhye
(ACIRI)
téléphone : +1 510 666 2887
mél : padhye@aciri.org
URL: http://www-net.cs.umass.edu/~jitu/

Sally Floyd
(ACIRI)
téléphone : +1 510 666 2989
mél : floyd@aciri.org
URL : http://www.aciri.org/floyd/

10. Déclaration de droits de reproduction

Remerciement

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