Groupe de travail Réseau
Request for Comments : 3517
Catégorie : En cours de normalisation
Traduction Claude Brière de L'Isle

E. Blanton, Purdue University
M. Allman, BBN/NASA GRC
K. Fall, Intel Research
L. Wang, University of Kentucky
avril 2003

Algorithme prudent de récupération de pertes fondé sur l'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 (RFC2581) mais permet aux envoyeurs TCP de récupérer plus efficacement lorsque plusieurs segments sont perdus dans un seul envoi de données.

Terminologie
Dans le présent document, les mots clés "DOIT", "NE DOIT PAS", "EXIGE", "DEVRA", "NE DEVRA PAS", "DEVRAIT", "NE DEVRAIT PAS", "RECOMMANDÉ", "PEUT", et "FACULTATIF" sont à interpréter comme décrit dans le BCP 14, [RFC2119].

1. Introduction

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, *selective acknowledgment*). Bien que le SACK TCP [RFC2018] soit activement déployé dans l'Internet [All00], il est prouvé que des hôtes n'utilisent pas les informations de SACK lorsque ils prennent des décisions de retransmission et de contrôle d'encombrement [PF01]. Le but du présent document est de présenter une méthode directe pour que les mises en œuvre TCP utilisent les informations de SACK pour augmenter les performances.

La [RFC2581] permet à TCP d'utiliser des algorithmes évolués de récupération de pertes [RFC793] pourvu qu'ils suivent l'esprit des algorithmes de contrôle d'encombrement de TCP [RFC2581], [RFC2914]. La [RFC2582] présente un de ces algorithmes évolués de récupération appelé NewReno. Le présent document présente un algorithme de récupération de pertes qui utilise l'option TCP SACK [RFC2018] pour améliorer la récupération de pertes de TCP. L'algorithme présenté dans le présent document, qui s'appuie fortement sur l'algorithme décrit dans [FF96], est un remplacement prudent de l'algorithme de récupération rapide [Jac90], [RFC2581]. L'algorithme spécifié dans le présent document est une stratégie directe de récupération de pertes fondé sur SACK qui suit les lignes directrices établies dans la [RFC2581] et peut en toute sécurité être utilisé par les mises en œuvre de TCP. D'autres méthodes de récupération de pertes fondées sur le SACK peuvent être utilisées dans TCP si les mises en œuvre le jugent utile (pour autant que les algorithmes de remplacement suivent les lignes directrices fournies dans la [RFC2581]). Prière de noter, cependant, que les décisions fondées sur SACK dans le présent document (comme quels segments sont à envoyer et à quel moment) sont largement découplées des algorithmes de contrôle d'encombrement, et à ce titre peuvent être traitées comme des questions distinctes, si on le désire.

2. Définitions

Le lecteur est supposé familier avec les définitions données dans la [RFC2581].

Le lecteur est supposé familier avec les accusés de réception sélectifs spécifiés dans la [RFC2018].

Afin d'expliquer l'algorithme de récupération de pertes fondé sur SACK, on définit quatre variables que mémorise un envoyeur TCP :

- "HighACK" est le numéro de séquence du plus fort octet de données qui a été acquitté cumulativement à un moment donné.
- "HighData" est le plus fort numéro de séquence transmis à un moment donné.
- "HighRxt" est le plus fort numéro de séquence qui a été retransmis durant la phase courante de récupération de pertes.
- "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 spécifié dans la [RFC2581]. L'algorithme est souvent appelé "algorithme pipe".

Pour les besoins de cette spécification on définit un "accusé de réception dupliqué" comme un segment qui arrive sans données et un numéro d'accusé de réception (ACK) qui est égal à la valeur actuelle de HighACK, comme décrit dans la [RFC2581].

On définit une variable "DupThresh" qui contient le nombre d'accusés de réception dupliqués exigé pour déclancher une retransmission. Selon la [RFC2581] ce seuil est défini comme étant de 3 accusés de réception dupliqués. Cependant, les mises en œuvre devraient consulter toute mise à jour de la [RFC2581] pour déterminer la valeur actuelle de DupThresh (ou la 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 de SACK

Pour qu'un envoyeur TCP mette en œuvre l'algorithme défini dans la section suivante il doit garder une structure de données pour mémoriser les informations des accusés de réception sélectifs entrants connexion par connexion. Une telle structure de données est couramment appelée le "tableau des résultats" *(scoreboard)*. Les spécificités de la structure de données de 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).

Note : le présent document se réfère à la prise en compte (marquage) des octets individuels de données transférés à travers une connexion TCP. Une mise en œuvre réelle du tableau des résultats préfèrera probablement gérer ces données comme des gammes de numéros de séquence. Les algorithmes présentés ici le permettent, mais exigent que des gammes arbitraires de numéros de séquence soient marquées comme ayant été acquittées sélectivement.

4. Traitement et actions sur les informations de SACK

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

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

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

IsLost (SeqNum) :
Ce sous-programme retourne 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 quand (DupThresh * 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 des 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 parcourues pour chaque octet 'S1' dans l'espace des numéros de séquence entre HighACK et HighData qui n'ont pas fait l'objet d'un SACK :

(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 fait l'objet d'un accusé de réception sélectif et n'ont pas été déterminés comme ayant été 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.

Note : Les octets retransmis sans être considérés comme perdus sont comptés deux fois par le mécanisme ci-dessus.

NextSeg () :
Ce sous programme utilise la structure de données de tableau de résultats conservée par la fonction Update() pour déterminer ce qu'il convient de transmettre sur la base des informations de SACK qui sont arrivées du receveur des données (et donc ont été marquées dans le tableau de résultats). NextSeg () DOIT retourner la gamme de numéros de séquence du prochain segment 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 les pertes, la gamme 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 haut octet couvert par tout SACK reçu.
   (1.c) IsLost (S2) retourne Vrai.

(2) Si il n'existe aucun numéro de séquence 'S2' selon la règle (1) mais si il existe des données non envoyées disponibles et si la fenêtre annoncée du receveur le permet, la gamme de séquence d'un segment de jusqu'à SMSS octets de données non envoyées précédemment en commençant par le numéro de séquence HighData+1 DOIT être retournée.

(3) Si les conditions pour les règles (1) et (2) échouent, mais si il existe un numéro de séquence 'S3' non acquitté sélectivement qui satisfait aux critères pour détecter la 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 PEUT être retourné.

Note : La règle (3) est une sorte de retransmission "en dernier recours". Elle permet la retransmission des numéros de séquence même lorsque l'envoyeur a une moindre certitude qu'un segment a été perdu que avec la règle (1). La retransmission des segments via la règle (3) va aider à maintenir l'horloge des ACK de TCP et donc peut éventuellement aider à éviter des fins de temporisation 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 de Pipe). 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 a quitté le réseau (une des deux transmissions du paquet ou les deux transmissions du paquet). Donc l'envoyeur peut sous estimer Pipe en considérant que les deux segments ont quitté le réseau alors qu'il est possible que seul un des deux l'ait fait. On pense que le déclanchement de la règle (3) sera rare et que ses implications seront probablement limitées à des cas marginaux par rapport à l'algorithme de récupération entier. Donc, on laisse la décision d'utiliser ou non la règle (3) aux mises en œuvre.

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

Note : L'algorithme de récupération de pertes fondé sur le SACK exposé dans le présent document exige plus de ressources de calcul que les stratégies précédentes de récupération de pertes de TCP. Cependant, on pense que la structure de 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 la mémoire) dans la plupart des mises en œuvre TCP.

5. Détail 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 ().

À réception du premier ACK dupliqué (DupThresh - 1), le tableau de résultats est mis à jour normalement. Noter que le premier et le second ACK dupliqués peuvent aussi être utilisés pour déclancher la transmission de segments précédemment non envoyés en utilisant l'algorithme Transmission limitée [RFC3042].

Lorsque un envoyeur TCP reçoit l'ACK dupliqué correspondant aux ACK DupThresh, le tableau des résultats DOIT être mis à jour avec les informations du nouveau SACK (via Update ()). Si aucun événement de perte précédent n'est survenu sur la connexion ou si le point d'accusé de réception cumulatif est au delà de la dernière valeur de RecoveryPoint, une phase de récupération de pertes DEVRAIT être initiée, selon l'algorithme de retransmission rapide présenté dans la [RFC2581]. Les étapes suivantes DOIT être effectuées :

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

(2) ssthresh = cwnd = (FlightSize / 2)
La fenêtre d'encombrement (cwnd) et le seuil de démarrage lent (ssthresh) sont réduit à la moitié de FlightSize selon la [RFC2581].

(3) Retransmettre le premier segment de données présumé abandonné -- le segment commençant avec le numéro de séquence HighACK + 1. Pour empêcher la répétition de la retransmission des mêmes données, régler HighRxt au plus fort numéro de séquence dans le segment retransmis.

(4) Run SetPipe ()
Régler une variable "pipe" au nombre d'octets en instance actuellement "dans le tuyau" ; ce sont les données qui ont été envoyées par l'envoyeur TCP mais pour lesquelles aucun accusé de réception cumulatif ni sélectif n'a été reçu et dont les données n'ont pas été déterminées avoir été abandonnées dans le réseau. On suppose que les données sont encore en train de traverser le réseau.

(5) Afin de tirer parti d'éventuelle cwnd potentiellement disponible, procéder à l'étape (C) ci-dessous.

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

(A) Un ACK cumulatif entrant pour un numéro de séquence supérieur à RecoveryPoint signale la fin de la récupération de pertes et la phase de récupération de pertes DOIT être terminée. Aucune information contenue dans le tableau des résultats pour des numéros de séquence supérieurs à la nouvelle valeur de HighACK NE DEVRAIT être supprimée en quittant la phase de récupération de pertes.

(B) À réception d'un ACK qui ne couvre pas RecoveryPoint, les actions suivantes DOIT être effectuées :
  (B.1) Utiliser Update () pour enregistrer les nouvelles informations de SACK convoyées par le 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 de résultats DOIT être interrogé via NextSeg () sur la gamme de numéros de séquence du prochain segment à transmettre (si il en est) et le segment en question envoyé. Si NextSeg () retourne un échec (pas de données à envoyer) retour sans envoi (c'est-à-dire, terminer les étapes C.1 à C.5).
  (C.2) Si 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.
  (C.3) Si des octets de données envoyés en (C.1) sont au-dessus de HighData, HighData doit être mis à jour pour refléter la transmission des données non envoyées précédemment.
  (C.4) L'estimation de la quantité de données en cours 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 en (C.1)

5.1 Temporisations de retransmission

Afin d'éviter des blocages de 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 expurger les informations de SACK collectées d'un receveur lors d'une expiration de temporisation de retransmission "car la fin de temporisation peut indiquer que le receveur des données a renoncé". De plus, un envoyeur TCP DOIT "ignorer les informations de SACK antérieures en déterminant quelles données retransmettre". Cependant, un envoyeur TCP de SACK DEVRAIT quand même utiliser toutes les informations de SACK disponibles durant la phase de démarrage lent de récupération de pertes suivant un RTO.

Si un RTO survient durant la récupération de pertes 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 pertes exposé dans le présent document DOIT être terminé. Une nouvelle phase de récupération (comme décrit à la section 5) NE DOIT PAS être initiée jusqu'à ce que HighACK soit supérieur ou égal à la nouvelle valeur de RecoveryPoint.

Comme décrit aux Sections 4 et 5, Update () DEVRAIT continuer d'être utilisé de façon approprié à réception des ACK. Cela permettra à la période de démarrage lent de récupération de bénéficier de toutes les informations disponibles fournies par le receveur, en dépit du fait que les informations de SACK ont été expurgées à cause du RTO.

Si il y a des segments manquants dans la mémoire tampon du receveur suite au traitement des segments retransmis, le ACK correspondant va contenir les 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 du démarrage lent. L'algorithme exact pour cette sélection n'est pas spécifié dans le présent document (spécifiquement, NextSeg () est inapproprié durant le démarrage lent après un RTO). Une approche relativement directe consistant à "remplir" l'espace des numéros de séquence rapportés comme manquants devrait être considérée comme raisonnable.

6. Gestion du temporisateur de RTO

L'estimateur de fin de temporisation de retransmission (RTO, *Retransmission TimeOut*) TCP standard est défini dans la [RFC2988]. Du fait que l'algorithme de SACK du présent document peut avoir un impact sur le comportement de l'estimateur, les mises en œuvre peuvent souhaiter examiner comment est géré le temporisateur. La [RFC2988] demande que le temporisateur de RTO soit réarmé chaque fois qu'arrive un ACK qui avance le point d'ACK cumulatif. Parce que l'algorithme exposé dans le présent document peut conserver l'horloge des ACK en fonctionnement à travers un événement de perte très significatif (relativement plus long que l'algorithme décrit dans la [RFC2581]) sur certains réseaux, l'événement de pertes pourrait durer plus longtemps que le RTO. Dans ce cas, le temporisateur de RTO va expirer prématurément et un segment qui n'a pas besoin d'être retransmis sera renvoyé.

Donc, on laisse aux mises en œuvre la latitude d'utiliser la gestion standard [RFC2988] de RTO ou, facultativement, une variante plus prudente qui réarme le temporisateur de RTO sur chaque retransmission qui est envoyée durant la récupération PEUT être utilisée. Cela donne une temporisation plus prudente que celle spécifiée dans la [RFC2988], et peut n'être pas toujours une alternative séduisante. Cependant, dans certains cas, cela peut empêcher des retransmissions inutiles, des retours de -N transmissions et réduire encore la fenêtre d'encombrement.

7. Recherches

L'algorithme spécifié dans le présent document est analysé dans [FF96], qui montre que l'algorithme ci-dessus est efficace pour réduire le temps de transfert sur le Reno TCP standard [RFC2581] lorsque plusieurs segments sont abandonnés d'une fenêtre de données (spécialement lorsque le nombre d'abandons augmente). [AHKO97] montre que l'algorithme défini dans le présent document peut grandement améliorer le débit dans les connexions qui traversent des canaux par satellite.

8. Considérations sur la sécurité

L'algorithme présenté dans cet article partage les considérations sur la sécurité de la [RFC2581]. Une différence clé est qu'un algorithme fondé sur les SACK est plus robuste contre les attaques qui forgent des ACK dupliqués pour forcer l'envoyeur TCP à réduire le cwnd. Avec les SACK, les envoyeurs TCP ont une vérification supplémentaire de la légitimité d'un ACK particulier. Bien qu'il ne soit pas à toute épreuve, le SACK fournit une certaine protection dans ce domaine.

Remerciements

Les auteurs souhaitent remercier Sally Floyd pour ses encouragements au présent document et ses commentaires sur les premiers projets. L'algorithme décrit dans le présent document est vaguement fondé sur un algorithme présenté par Kevin Fall et Sally Floyd dans [FF96], bien que les auteurs du présent document assument la responsabilité de toute erreur du texte ci-dessus. Murali Bashyam, Ken Calvert, Tom Henderson, Reiner Ludwig, Jamshid Mahdavi, Matt Mathis, Shawn Ostermann, Vern Paxson et Venkat Venkatsubra ont fourni de précieux retours sur les premières versions du présent document. Nous remercions Matt Mathis et Jamshid Mahdavi pour la mise en œuvre du tableau de résultats et pour avoir guidé nos efforts pour conserver la trace de l'état SACK.

Le premier auteur tient à remercier l'Université de l'Ohio et le groupe de recherche d'interréseautage de l'Université de l'Ohio pour le soutien apporté au travail sur ce projet.

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. (*Remplace* RFC1072) *(P.S.)*

[RFC2026] S. Bradner, "Le processus de normalisation de l'Internet -- Révision 3", (BCP0009) octobre 1996. *(Remplace* RFC1602, RFC1871) (*MàJ par* RFC3667, RFC3668, RFC3932, RFC3979, RFC3978, RFC5378, RFC6410)

[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)

Références pour information

[AHKO97] Mark Allman, Chris Hayes, Hans Kruse, Shawn Ostermann. "TCP Performance Over Satellite Links". Proceedings of the Fifth 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.

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

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

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

[RFC2582] S. Floyd, T. Henderson, "Modification NewReno à l'algorithme de récupération rapide de TCP", avril 1999. (*Obsolète, voir* RFC3782) *(Expérimentale)*

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

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

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

Considérations sur la propriété intellectuelle

Une revendication de droits de propriété intellectuelle a été notifiée à l'IETF à l'égard de tout ou partie de la spécification contenue dans le présent document. Pour d'autres informations, consulter la liste en ligne des revendications de droits.

L'IETF ne prend position sur la validité ou la portée d'aucun droit de propriété intellectuelle ou d'autres droits qui pourraient être revendiqués au titre de la mise en œuvre ou l'utilisation de la technologie décrite dans le présent document ou sur la mesure dans laquelle toute licence sur de tels droits pourrait être ou non disponible ; pas plus qu'elle ne prétend qu'elle ait fait aucun effort pour identifier de tels droits. Des informations sur les procédures de l'IETF au sujet des droits dans la documentation en cours de normalisation et en rapport avec les normes peuvent être trouvées dans le BCP-11. Des copies des revendications de droits peuvent être disponibles à la publication et toutes les assurances de licences peuvent être rendues disponibles, ou le résultat de tentatives d'obtention d'une licence ou permission générale pour l'utilisation de tels droits de propriété par les mises en œuvre ou utilisateurs de la présente spécification peuvent être obtenus auprès du secrétariat de l'IETF.

L'IETF invite toute partie intéressée à porter à son attention tous droits de reproduction, brevets ou applications de brevets,

ou autres droits de propriété qui pourraient couvrir une technologie qui pourrait être nécessaire pour mettre en pratique la présente norme. Prière d'adresser les information au Directeur Général de l'IETF.

Adresses des auteurs

Ethan Blanton
Purdue University Computer Sciences
1398 Computer Science Building
West Lafayette, IN 47907
mél : eblanton@cs.purdue.edu

Mark Allman
BBN Technologies/NASA Glenn Research Center
Lewis Field
21000 Brookpark Rd. MS 54-5
Cleveland, OH 44135
mél : mallman@bbn.com

Kevin Fall
Intel Research
2150 Shattuck Ave., PH Suite
Berkeley, CA 94704
mél : kfall@intel-research.net

Lili Wang
Laboratory for Advanced Networking
210 Hardymon Building
University of Kentucky
Lexington, KY 40506-0495
mél : lwang0@uky.edu

Déclaration complète de droits de reproduction

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