Groupe de travail Réseau
Request for Comments : 3124
Catégorie : En cours de normalisation
Traduction Claude Brière de L'Isle

H. Balakrishnan, MIT LCS
S. Seshan, CMU
juin 2001

Le gestionnaire d'encombrement

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

Résumé
Le présent document décrit le gestionnaire d'encombrement (CM, *Congestion Manager*) un module de système d'extrémité qui :
(i) permet à un ensemble de plusieurs flux concurrents provenant d'un envoyeur et destinés au même receveur et partageant les mêmes propriétés d'encombrement d'effectuer un évitement et un contrôle d'encombrement appropriés, et
(ii) permet aux applications de s'adapter facilement à l'encombrement du réseau.

1. Conventions utilisées dans le document

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 la [RFC2119].

Flux
C'est un groupe de paquets qui partagent tous la même adresse IP de source et de destination, le type de service IP, le protocole de transport, et les numéros d'accès de source et destination de couche transport.

Macroflux
C'est un groupe de flux à capacité de gestionnaire d'encombrement (CM) qui utilisent tous les mêmes algorithmes de gestion d'encombrement et de programmation, et partagent les informations d'état d'encombrement. Actuellement, les flux destinés à des receveurs différents appartiennent à des macroflux différents. Les flux destinés au même receveur PEUVENT appartenir à des macroflux différents. Lorsque le gestionnaire d'encombrement est utilisé, les flux qui ont un comportement d'encombrement identique et utilisent le même algorithme de contrôle d'encombrement DEVRAIENT appartenir au même macroflux.

Application
Tout module logiciel qui utilise le CM. Cela inclut des applications de niveau utilisateur telles que les serveurs de la Toile ou des serveurs audio/vidéo, ainsi que des protocoles du noyau tels que TCP [RFC0793] qui utilisent le CM pour le contrôle d'encombrement.

Application qui se comporte bien
C'est une application qui ne transmet que quand c'est permis par le CM et qui tient précisément le compte de toutes les données qu'elle a envoyé au receveur en informant le CM à l'aide de l'API CM.

Unité maximum de transmission de chemin (PMTU, *Path Maximum Transmission Unit*)
C'est la taille du plus grand paquet que l'envoyeur peut transmettre sans qu'il soit fragmenté en route vers le receveur. Elle inclut les tailles de tous les en-têtes et des données excepté celle de l'en-tête IP.

Fenêtre d'encombrement (cwnd, *congestion window*)
Variable d'état du CM qui module la quantité de données en suspens entre l'envoyeur et le receveur.

Fenêtre en suspens (ownd, *outstanding window*)
C'est le nombre d'octets qui ont été transmis par la source, mais dont on ne sait pas si ils ont été reçus par la destination ou perdus dans le réseau.

Fenêtre initiale (IW, *Initial Window*)
C'est la taille de la fenêtre d'encombrement de l'envoyeur au début d'un macroflux.

Syntaxe des types de données
On utilise "u64" pour les quantités de 64 bits non signées, "u32" pour celles de 32 bits non signées, "u16" pour 16 bits non signés, "u8" pour 8 bits non signés, "i32" pour 32 bits signés, "i16" pour 16 bits signés, "float" pour les valeurs à virgule flottante IEEE. Le type "void" est utilisé pour indiquer qu'aucune valeur en retour n'est attendue d'un appel. On se réfère aux pointeurs en utilisant la syntaxe "*", suivant la convention du langage C.

On souligne que toutes les fonctions d'API décrites dans le présent document sont des appels "abstraits" et que les mises en œuvre conformes de CM peuvent différer dans les détails spécifiques de mise en œuvre.

2. Introduction

Le cadre décrit dans le présent document intègre la gestion d'encombrement à travers les applications et les protocoles de transport. Le CM entretient les paramètres d'encombrement (bande passante disponible agrégée et par flux, délais d'aller retour par receveur, etc.) et exporte une interface de programmation d'application (API, *application programming interface*) qui permet aux applications d'apprendre les caractéristiques du réseau, de passer les informations au CM, de partager entre elles les informations d'encombrement, et de programmer les transmissions de données. Le présent document se concentre sur les protocoles d'application et de transport avec leurs propres informations indépendantes de numéro de séquence par octet ou par paquet, et n'exige pas de modifications à la pile de protocoles du receveur. Cependant, l'application receveuse doit fournir des retours à l'application d'envoi sur la réception des paquets et les pertes, et cette dernière est supposée utiliser l'API CM pour mettre à jour l'état de CM. Le présent document ne vise pas les réseaux avec réservations ou différenciation de services.

Le CM est un module de système d'extrémité qui permet à un ensemble de plusieurs flux concurrents d'effectuer un évitement et un contrôle d'encombrement stable, et qui permet aux applications d'adapter facilement leurs transmissions aux conditions qui prévalent dans le réseau. Il intègre la gestion d'encombrement à travers toutes les applications et protocoles de transport. Il entretient les paramètres d'encombrement (bande passante agrégée disponible et par flux, délais d'aller-retour par receveur, etc.) et exporte une API qui permet aux applications d'apprendre les caractéristiques du réseau, de passer les informations au CM, de se partager les information d'encombrement, et de programmer les transmissions de données. Lorsque le CM est utilisé, toutes les transmissions de données soumises au CM doivent être faites avec le consentement explicite du CM via cette API pour s'assurer d'un comportement d'encombrement approprié.

Les systèmes PEUVENT choisir d'utiliser CM, et si ils le font, ils DOIVENT suivre la présente spécification.

Le présent document se concentre sur les applications et réseaux qui respectent les conditions suivantes :

1. Les applications se comportent bien avec leurs propres informations de numéro de séquence indépendantes par octet ou par paquet, et utilisent l'API de CM pour mettre à jour leur état interne dans le CM.

2. Les réseaux sont au mieux sans discrimination ou réservations de service. En particulier, il ne traite pas des situations où différents flux entre la même paire d'hôtes traversent des chemins avec des caractéristiques différentes.

Le cadre de gestionnaire d'encombrement peut être étendu pour prendre en charge des applications qui ne fournissent pas leurs propres retours et aux réseaux à services différenciés. Ces extensions seront traitées dans des documents ultérieurs.

Le CM répond à deux objectifs principaux :
(i) Permettre un multiplexage efficace. De plus en plus, la tendance de l'Internet est à l'envoi individuel de données (par exemple, les serveurs de la Toile) pour transmettre des types hétérogènes de données aux receveurs, allant de contenus non fiables en temps réel à des pages et appliquettes fiable de la Toile. Il en résulte que de nombreux flux logiquement différents partagent le même chemin entre envoyeur et receveur. Pour que l'Internet reste stable, chacun de ces flux doit incorporer des protocoles de contrôle qui enquêtent en toute sécurité sur les économies de bande passante et réagissent à l'encombrement. Malheureusement, ces flux concurrents sont normalement en compétition les uns avec les autres pour les ressources du réseau, plutôt qu'ils ne les partagent efficacement. De plus, ils n'apprennent pas les uns des autres l'état du réseau. Même si ils mettent chacun en œuvre indépendamment le contrôle d'encombrement (par exemple, un groupe de connexions TCP mettant chacune en œuvre les algorithmes de [Jacobson88], [RFC2581]) l'ensemble des flux

tendent à être plus agressifs en face d'encombrement qu'une seule connexion TCP qui met en œuvre le contrôle et l'évitement d'encombrement TCP standard [Balakrishnan98].

(ii) Permettre l'adaptation des applications à l'encombrement. De plus en plus, des applications populaires en continu en temps réel fonctionnent sur UDP en utilisant leur propres protocoles de transport de niveau utilisateur pour de bonnes performances d'application, mais dans la plupart des cas aujourd'hui, elles ne s'adaptent pas ou ne réagissent pas correctement à l'encombrement du réseau. En mettant en œuvre un algorithme stable de contrôle et en présentant une API adaptée, le CM permet une adaptation facile des applications à l'encombrement. Les applications adaptent les données qu'elles transmettent aux conditions actuelles du réseau.

Le cadre de CM s'appuie sur les récents travaux sur le partage de bloc de contrôle TCP [RFC2140], le contrôle d'encombrement TCP intégré (TCP-Int) [Balakrishnan98] et les sessions TCP [Padmanabhan98]. La [RFC2140] plaide en faveur du partage de certains états dans le bloc de contrôle TCP pour améliorer les performance transitoires de transport et décrit le partage à travers un ensemble de connexions TCP. [Balakrishnan98], [Padmanabhan98], et [Eggert00] décrivent plusieurs expériences qui quantifient les avantages du partage d'état d'encombrement, incluant une stabilité accrue en présence d'encombrement et une meilleure récupération de perte. Intégrer la récupération de perte entre des connexions concurrentes améliore significativement les performances parce que les pertes sur une connexion peuvent être détectées en remarquant que des données en retard envoyées sur une autre connexion ont été reçues et acquittées. Le cadre de CM étend ces idées de deux façons significatives : (i) il étend la gestion d'encombrement aux flux non TCP, qui deviennent de plus en plus courants et souvent ne mettent pas correctement en œuvre la gestion d'encombrement, et (ii) il fournit une API pour que les applications adaptent leurs transmissions aux conditions actuelles du réseau. Pour un exposé complet des motivations du CM, son architecture, l'API, et les algorithmes, voir [Balakrishnan99] ; pour une description d'une mise en œuvre et les résultats des performances, voir [Andersen00].

L'architecture résultante du protocole d'hôte d'extrémité chez l'envoyeur est montrée à la Figure 1. Le CM aide à assurer la stabilité du réseau en mettant en œuvre des algorithmes d'évitement et de contrôle d'encombrement stables qui sont "compatibles TCP" [Mahdavi98] sur la base des algorithmes décrits dans la [RFC2581]. Cependant, il ne tente pas de mettre en application un comportement approprié à l'encombrement pour toutes les applications (mais il n'empêche pas une régulation chez l'hôte qui effectue cette tâche). Noter qu'alors que le régulateur chez l'hôte d'extrémité peut utiliser le CM, le réseau doit être protégé contre la compromission du CM et du régulateur chez les hôtes d'extrémité, une tâche qui exige des mécanismes dans les routeurs [Floyd99a]. On ne traite pas plus avant de cette question dans ce document.

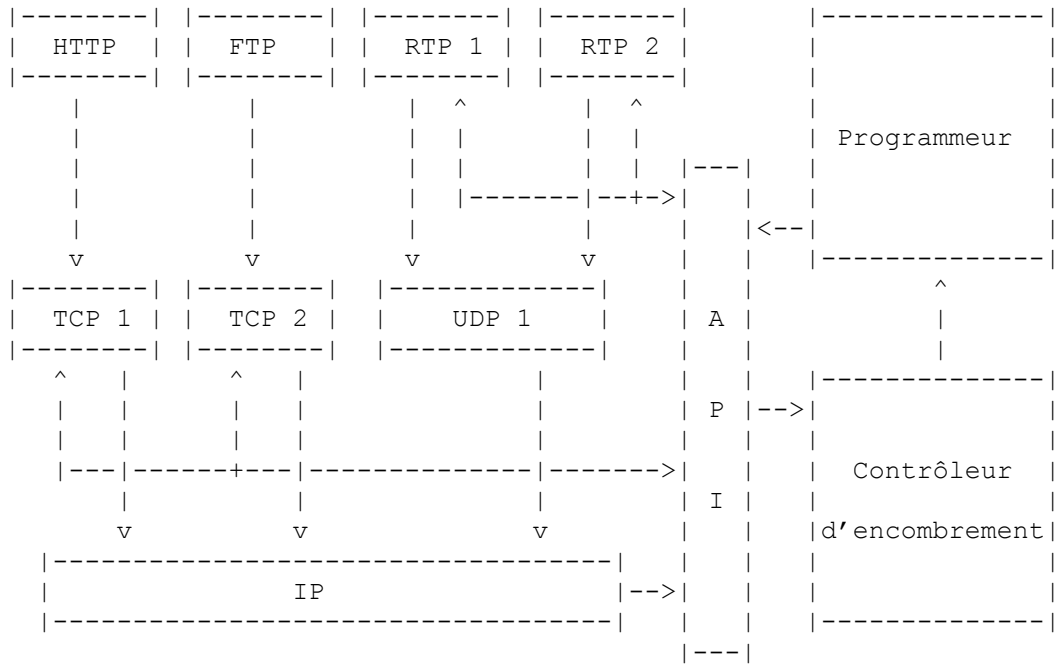

Figure 1

Les composants clés du cadre de CM sont (i) l'API, (ii) le contrôleur d'encombrement, et (iii) le programmeur. L'API est (en partie) motivée par les exigences de verrouillage de niveau application (ALF, *Application-Level Framing*) [Clark90], et est décrite à la Section 4. L'intérieur du CM (Section 5) comporte un contrôleur d'encombrement (paragraphe 5.1) et un programmeur pour orchestrer la transmissions des données entre les flux concurrents dans un macroflux (paragraphe 5.2). Le contrôleur d'encombrement ajuste le taux agrégé de transmission entre envoyeur et receveur sur la base de son estimation de l'encombrement dans le réseau. Il obtient des retours sur ses transmissions passées de la part des applications elles-mêmes via l'API. Le programmeur partage la bande passante disponible entre les différents flux au sein de chaque

macroflux et notifie aux applications quand il leur est permis d'envoyer des données. Le présent document se concentre sur les applications qui se comportent bien ; un document futur décrira le protocole envoyeur-receveur et les formats d'en-tête qui s'appliqueront aux applications qui n'incorporent pas leurs propres retours au CM.

3. API de gestionnaire d'encombrement

Par convention, l'IETF ne normalise pas les interfaces de programmation d'application (API). Cependant, il est considéré qu'il est important d'avoir les exigences de l'API CM et de l'algorithme de CM dans un seul document cohérent. La section qui suit sur l'API CM utilise les termes DOIT, DEVRAIT, etc., mais ces termes sont censés s'appliquer au sein du contexte d'une mise en œuvre de l'API CM. La section ne s'applique pas aux mises en œuvre de contrôle d'encombrement en général, mais seulement à celles qui offrent l'API CM.

En utilisant l'API CM, les flux peuvent déterminer leur part de la bande passante disponible, demander et avoir leurs transmissions de données programmées, informer le CM de la réussite de leurs transmissions, et être informés des changements d'estimation de la bande passante du chemin par le CM. Donc, le CM libère les applications de la conservation des informations sur l'état d'encombrement et la bande passante disponible le long de tout chemin.

Les prototypes de fonction ci-dessous suivent les conventions standard du langage C. On souligne que ces fonctions d'API sont des appels abstraits et que les mises en œuvre conformes de CM peuvent différer dans les détails spécifiques, pour autant que soient fournies des fonctionnalités équivalentes.

Lorsque un nouveau flux est créé par une application, il passe des informations au CM via l'appel d'API cm_open(stream_info). Actuellement, stream_info comporte les informations suivantes :
(i) l'adresse IP de source,
(ii) l'accès de source,
(iii) l'adresse IP de destination,
(iv) l'accès de destination,
(v) le numéro de protocole IP.

3.1 Maintenance d'état

1. Ouvrir : Toutes les applications DOIVENT invoquer cm_open(stream_info) avant d'utiliser l'API CM. Cela retourne une commande, cm_streamid, que l'application va utiliser pour toutes les invocations d'API CM ultérieures pour ce flux. Si le cm_streamid retourné est -1, le cm_open() a alors échoué et le flux ne peut pas utiliser le CM.

Tous les autres appels d'un flux au CM utilisent le cm_streamid retourné de l'appel cm_open().

2. Clore : Lorsque un flux se termine, l'application DEVRAIT invoquer cm_close(cm_streamid) pour informer le CM de la terminaison du flux.

3. Taille de paquet : cm_mtu(cm_streamid) retourne la PMTU estimée du chemin entre envoyeur et receveur. En interne, ces informations DEVRAIENT être obtenues via la découverte de la MTU de chemin [RFC1191]. Elle PEUT être statiquement configurée en l'absence d'un tel mécanisme.

3.2 Transmission des données

Le CM s'accommode de deux types d'envoyeurs adaptatifs, permettant aux applications d'adapter de façon dynamique leur contenu sur la base des conditions prévalentes dans le réseau, et de prendre en charge les applications fondées sur l'API.

1. Transmission fondée sur le rappel. L'API de transmission fondée sur le rappel donne au flux le plein contrôle sur la décision de ce qui est à transmettre à chaque instant. Pour réaliser cela, le CM ne met aucune donnée en mémoire tampon, mais permet plutôt aux flux de s'adapter au tout dernier moment aux changements inattendus survenus dans le réseau. Cela permet donc aux flux de "tirer leur épingle du jeu" et de repaquetiser les données lorsqu'elles apprennent un changement du débit, ce qui est difficile à faire une fois que les données ont été mises en mémoire tampon. Le CM doit mettre en œuvre un appel cm_request(i32 cm_streamid) pour les flux qui souhaitent envoyer des données dans ce style. Après un certain temps, selon le débit, le CM DOIT invoquer un rappel en utilisant cmapp_send(), qui est l'accord donné au flux d'envoyer jusqu'à PMTU octets. L'API de style rappel est le choix recommandé pour les flux fondés sur ALF. Noter que cm_request() ne prend pas comme argument le nombre d'octets ou les unités de mesure de la MTU ; chaque appel à cm_request() est une demande implicite de l'envoi de jusqu'à PMTU octets. Le CM PEUT fournir une interface de remplacement, cm_request(int k). Le rappel cmapp_send pour cette demande se voit accorder le droit

d'envoyer jusqu'à k segments de la taille de la PMTU. Le paragraphe 4.3 expose la durée pendant laquelle l'accord de transmission est valide, et le paragraphe 5.2 décrit comment ces demandes sont programmées et comment sont faits les rappels.

2. Style synchrone. L'API fondée sur le rappel s'accommode d'une classe de flux ALF qui sont "asynchrones". Les émetteurs asynchrones ne transmettent pas sur la base d'une horloge périodique, mais le font lorsque déclanchés par des événements asynchrones tels que des lectures de fichiers ou de trames capturées. D'un autre côté, il y a de nombreux flux qui sont des émetteurs "synchrones", qui émettent périodiquement sur la base de leurs propres temporisateurs internes (par exemple, un envoyeur audio qui envoie à un taux d'échantillonnage constant). Bien que les rappels de CM puissent être configurés pour interrompre périodiquement de tels émetteurs, la boucle de transmission de telles applications est moins affectée si ils conservent leur boucle originale fondée sur le temporisateur. De plus, cela complique l'API CM d'avoir un flux qui exprime la périodicité et la granularité de ses rappels. Donc, le CM DOIT exporter une API qui permet à de tels flux d'être informés des changements de débit en utilisant la fonction cmapp_update(u64 newrate, u32 srtt, u32 rttdev) où newrate est le nouveau débit en bits par seconde pour ce flux, srtt est l'estimation actuelle lissée de délai d'aller-retour en microsecondes, et rttdev est l'écart type linéaire lissé de l'estimation de délai d'aller retour calculé en utilisant le même algorithme que dans TCP [RFC2988]. La valeur newrate rapporte un débit instantané calculé, par exemple, en prenant le ratio de cwnd et srtt, et en divisant par la fraction de ce ratio allouée au flux.

En réponse, le flux DOIT adapter sa taille de paquet ou changer son intervalle de temporisateur pour se conformer au débit admis (c'est-à-dire, ne pas le dépasser). Bien sûr, il peut choisir de ne pas utiliser tout ce débit. Noter que le CM n'est pas sur le chemin des données de la transmission réelle.

Pour éviter des rappels cmapp_update() inutiles que l'application va seulement ignorer, le CM DOIT fournir une fonction cm_thresh(float rate_downthresh, float rate_upthresh, float rtt_downthresh, float rtt_upthresh) qu'un flux peut utiliser à tout moment dans son exécution. En réponse, le CM DEVRAIT n'invoquer le rappel que lorsque le débit diminue jusqu'à moins que (rate_downthresh * lastrate) ou augmente jusqu'à plus que (rate_upthresh * lastrate), où lastrate est le débit notifié en dernier au flux, ou lorsque le délai d'aller-retour change en conséquence des seuils exigés. Ces informations sont utilisées comme conseils par le CM, dans ce sens que cmapp_update() peut être invoqué même si ces conditions ne sont pas satisfaites.

Le CM DOIT mettre en œuvre une fonction cm_query(i32 cm_streamid, u64* rate, u32* srtt, u32* rttdev) pour permettre à une application d'interroger l'état actuel du CM. Cela règle la variable de débit à l'estimation de débit en cours en bits par seconde, la variable srtt à l'estimation actuelle du délai d'aller-retour actuel lissé en microsecondes, et rttdev à l'écart type linéaire moyen. Si le CM n'a pas d'estimation valide pour le macroflux, il donne des valeurs négatives pour le débit, srtt, et rttdev.

Noter qu'un flux peut utiliser en même temps plus d'une des API de transmission ci-dessus. En particulier, la connaissance d'un débit tenable est utile pour les flux asynchrones comme pour les synchrones ; par exemple, un serveur asynchrone de la Toile qui dissémine des images en utilisant TCP peut utiliser cmapp_send() pour programmer ses transmissions et cmapp_update() pour décider si il envoie une image à faible ou haute résolution. Une mise en œuvre TCP qui utilise le CM est décrite au paragraphe 6.1.1, et les avantages de l'API de rappel cm_request() pour TCP y sont apparents.

Le lecteur remarquera que l'API CM de base ne fournit pas d'interface pour les transmissions à contrôle d'encombrement mises en mémoire tampon. C'est intentionnel, car ce mode de transmission peut être mis en œuvre en utilisant la primitive fondée sur le rappel. Le paragraphe 6.1.2 décrit comment des prises UDP à contrôle d'encombrement peuvent être mises en œuvre en utilisant l'API CM.

3.3 Notification d'application

Lorsque un flux reçoit des retours des receveurs, il DOIT utiliser cm_update(i32 cm_streamid, u32 nrecd, u32 nlost, u8 lossmode, i32 rtt) pour informer le CM d'événements tels que des pertes d'encombrement, des réceptions réussies, du type de perte (événement de fin de temporisation, notification explicite d'encombrement [RFC2481], etc.) et des échantillons de délai d'aller-retour. Le paramètre nrecd indique combien d'octets ont été bien reçus par le receveur depuis le dernier appel cm_update, tandis que le paramètre nrecd identifie combien d'octets ont été reçus/perdus pendant la même période. La valeur rtt indique le délai d'aller-retour mesuré durant la transmission de ces octets. La valeur rtt doit être réglée à -1 si aucun échantillon d'aller-retour valide n'a été obtenu par l'application. Le paramètre lossmode est un indicateur du nombre de pertes détectées. Une valeur de CM_NO_FEEDBACK indique que l'application n'a pas reçu de retour pour toutes ses données en suspens et elle est rapportée au CM. Par exemple, un TCP qui a subi une fin de temporisation va utiliser ce paramètre pour en informer le CM. Une valeur de CM_LOSS_FEEDBACK indique que l'application a subi des pertes, ce qu'elle pense dû à l'encombrement, mais toutes les données en suspens n'ont pas été perdues. Par exemple, une perte de segment TCP détectée en utilisant les accusés de réception dupliqués (sélectifs) ou d'autres techniques pilotées par les

données rentre dans cette catégorie. Une valeur de CM_EXPLICIT_CONGESTION indique que le receveur a fait écho d'un message de notification explicite d'encombrement. Finalement, une valeur de CM_NO_CONGESTION indique qu'aucune perte en rapport avec de l'encombrement n'est survenue. Le paramètre lossmode DOIT être rapporté comme vecteur binaire où les bits correspondent à CM_NO_FEEDBACK, CM_LOSS_FEEDBACK, CM_EXPLICIT_CONGESTION, et CM_NO_CONGESTION. Noter que sur les liaisons (chemins) qui subissent des pertes pour des raisons autres que l'encombrement, une application DEVRAIT informer le CM des pertes, avec le champ CM_NO_CONGESTION établi *(à 1)*.

cm_notify(i32 cm_streamid, u32 nsent) DOIT être invoquée lorsque des données sont transmises de l'hôte (par exemple, dans le sous-programme de sortie IP) pour informer le CM que nsent octets ont été émis sur un certain flux. Cela permet au CM de mettre à jour son estimation du nombre d'octets en suspens pour le macroflux et pour le flux.

Une allocation cmapp_send() de la part du CM à une application n'est valide que pour un certain temps, égal au plus grand du délai d'aller-retour et d'un seuil dépendant de la mise en œuvre communiqué comme argument de la fonction de rappel cmapp_send(). L'application NE DOIT PAS envoyer de données sur la base de ce rappel après que ce délai a expiré. De plus, si l'application décide de ne pas envoyer de données après avoir reçu ce rappel, elle DEVRAIT invoquer cm_notify(stream_info, 0) pour permettre au CM d'autoriser d'autres flux du macroflux à transmettre des données. Le contrôleur d'encombrement du CM DOIT être robuste face aux applications qui oublient d'invoquer correctement cm_notify(stream_info, 0), ou aux applications qui crashent ou disparaissent après avoir fait un appel cm_request().

3.4 Interrogation

Si les applications souhaitent apprendre la bande passante disponible par flux et le temps d'aller-retour, elles peuvent utiliser l'invocation cm_query(i32 cm_streamid, i64* rate, i32* srtt, i32* rttdev) du CM, qui contient les quantités désirées. Si le CM n'a pas d'estimations valides pour le macroflux, il met des valeurs négatives pour le débit, srtt, et rttdev.

3.5 Partage de granularité

Une des décisions que doit prendre le CM est la granularité à laquelle un macroflux est construit, en décidant quels flux appartiennent au même macroflux et partagent les informations d'encombrement. L'API fournit deux fonctions qui permettent aux applications de décider lesquels de leurs flux devraient appartenir au même macroflux.

cm_getmacroflow(i32 cm_streamid) retourne un identifiant univoque i32 de macroflux. cm_setmacroflow(i32 cm_macroflowid, i32 cm_streamid) établit le macroflux du flux cm_streamid à cm_macroflowid. Si le cm_macroflowid qui est passé au cm_setmacroflow() est -1, alors un nouveau macroflux est construit et cela est retourné à l'appelant. Chaque appel à cm_setmacroflow() subvertit la précédente association de macroflux pour le flux, s'il en existe un.

La méthode d'agrégation par défaut suggérée est d'agréger par adresse IP de destination, c'est-à-dire que tous les flux pour la même adresse de destination sont agrégés par défaut à un seul macroflux. Les appels cm_getmacroflow() et cm_setmacroflow() peuvent alors être utilisés pour changer cela en tant que de besoin. On note bien qu'il y a certains cas où ceci peut n'être pas optimal, même sur des réseaux au mieux. Par exemple, lorsque un groupe de receveurs est derrière un appareil de NAT *(traducteur d'adresse réseau)*, l'envoyeur les verra comme une seule adresse. Si les hôtes derrière le NAT sont en fait connectés sur des liaisons embouteillées différentes, certains de ces hôtes pourraient voir de plus mauvaises performances qu'avant. Il est possible de détecter de tels hôtes lorsque on utilise des estimations de délai et de pertes, mais les mécanismes spécifiques pour le faire sortent du domaine d'application du présent document.

L'objectif de cette interface est d'établir un partage pour des groupes qui ne partagent pas de politique de pondération relative des flux dans un macroflux. Ce dernier exige que le programmeur fournisse une interface pour établir la politique de partage. Cependant, parce qu'on veut prendre en charge de nombreux programmeurs différents (chacun d'eux peut avoir besoin d'informations différentes pour établir sa politique) on ne spécifie pas une API complète pour le programmeur (mais voir le paragraphe 5.2). Un document de lignes directrices ultérieur est supposé décrire quelques programmeurs simples (par exemple, round robin pondéré, programmation hiérarchique) et l'API qu'ils exportent pour fournir une priorité relative.

4. Composants internes du contrôleur d'encombrement

Cette section décrit les composants internes du CM. Il comporte un contrôleur d'encombrement et un programmeur, avec les interfaces abstraites bien définies qu'ils exportent.

4.1 Contrôleur d'encombrement

Un algorithme de contrôle d'encombrement est associé à chaque macroflux ; la collection de tous ces algorithmes comporte le contrôleur d'encombrement du CM. L'algorithme de contrôle décide quand et combien de données peuvent être transmises par un macroflux. Il utilise des notifications d'application (paragraphe 4.3) provenant des flux concurrents sur le même macroflux pour construire les informations sur l'état d'encombrement du chemin de réseau utilisé par le macroflux.

Le contrôleur d'encombrement DOIT mettre en œuvre un algorithme de contrôle d'encombrement "facile à utiliser pour TCP" [Mahdavi98]. Plusieurs macroflux PEUVENT (et bien sûr, vont souvent le faire) utiliser le même algorithme de contrôle d'encombrement mais chaque macroflux entretient l'état sur le réseau utilisé par ses flux.

Le module de contrôle d'encombrement DOIT mettre en œuvre les interfaces abstraites suivantes. On souligne que ceux-ci ne sont pas directement visibles aux applications ; ils sont au sein du contexte d'un macroflux, et sont différents des fonctions d'API de CM de la Section 4.

- void query(u64 *rate, u32 *srtt, u32 *rttdev) : cette fonction retourne le débit estimé (en bits par seconde) et le délai d'aller-retour lissé (en microsecondes) pour le macroflux.

- void notify(u32 nsent) : cette fonction DOIT être utilisée pour notifier au module de contrôle d'encombrement chaque fois que des données sont envoyées par une application. Le paramètre nsent indique le nombre d'octets qui viennent d'être envoyés par l'application.

- void update(u32 nsent, u32 nrecd, u32 rtt, u32 lossmode) : cette fonction est invoquée chaque fois qu'un des flux CM associé à un macroflux identifie que des données ont atteint le receveur ou ont été perdues en route. Le paramètre nrecd indique le nombre d'octets qui viennent d'arriver chez le receveur. Le paramètre nsent est la somme du nombre d'octets qui viennent d'être reçus et du nombre d'octets identifiés comme perdus en route. Le paramètre rtt est le délai d'aller-retour estimé en microsecondes durant le transfert. Le paramètre lossmode donne une indication de la façon dont une perte a été détectée (paragraphe 4.3).

Bien que ces interfaces ne soient pas visibles aux applications, le contrôleur d'encombrement DOIT mettre en œuvre ces interfaces abstraites pour assurer l'interopérabilité modulaire avec des programmeurs différents développés séparément.

Le module de contrôle d'encombrement DOIT aussi invoquer la fonction de programmation du programmeur associé (paragraphe 5.2) quand il pense que l'état d'encombrement actuel permet à un paquet de la taille de la MTU d'être envoyé.

4.2 Programmeur

Bien qu'il soit de la responsabilité du module de contrôle d'encombrement de déterminer quand et combien de données peuvent être transmises, il est de la responsabilité du module programmeur d'un macroflux de déterminer lequel des flux devrait obtenir l'opportunité de transmettre les données.

Le programmeur DOIT mettre en œuvre les interfaces suivantes :
- void schedule(u32 num_bytes) : lorsque le module de contrôle d'encombrement détermine que les données peuvent être envoyées, le programme schedule() DOIT être invoqué avec pas plus que le nombre d'octets qui peuvent être envoyés. À son tour, le programmeur PEUT invoquer la fonction cmapp_send() que les applications CM doivent fournir.
- float query_share(i32 cm_streamid) : cette invocation retourne la part du flux décrit de la bande passante disponible totale pour le macroflux. Cette invocation combinée avec l'appel d'interrogation du contrôleur d'encombrement fournit les informations pour satisfaire la demande cm_query() d'une application.
- void notify(i32 cm_streamid, u32 nsent) : cette interface est utilisée pour notifier le module programmeur chaque fois que des données sont envoyées par une application CM. Le paramètre nsent indique le nombre d'octets qui viennent d'être envoyés par l'application.

Le programmeur PEUT mettre en œuvre de nombreuses interfaces supplémentaires. Avec l'augmentation de l'expérience des programmeurs CM, de futurs documents pourront ajouter et/ou changer certaines parties de l'API de programmeur.

5. Exemples

5.1 Exemples d'applications

Cette section décrit trois utilisations possibles de l'API CM par les applications. On décrit deux applications asynchrones --- une mise en œuvre d'un envoyeur TCP et une mise en œuvre de prises UDP à contrôle d'encombrement, et une application synchrone --- un serveur audio en continu. Plus de détails sur ces applications et les optimisations de mise en

œuvre de CM pour un fonctionnement efficace sont décrits dans [Andersen00].

Toutes les applications qui utilisent le CM DOIVENT incorporer les retours provenant du receveur. Par exemple, il doit périodiquement (normalement une ou deux fois par délai d'aller-retour) déterminer combien de ses paquets sont arrivés chez le receveur. Quand la source obtient ce retour, elle DOIT utiliser cm_update() pour informer le CM de ces nouvelles informations. Il en résulte que le CM met à jour ownd et il peut en résulter que le CM change ses estimations et appels de cmapp_update() des flux du macroflux.

Les protocoles de cette section sont des exemples et des suggestions pour la mise en œuvre, plus que des exigences pour toute mise en œuvre conforme.

5.1.1 TCP

Une mise en œuvre TCP qui utilise le CM devrait utiliser l'API de rappel cmapp_send(). TCP identifie seulement quelles données il devrait envoyer à l'arrivée d'un accusé de réception ou à l'expiration d'un temporisateur. Il en résulte que cela exige un contrôle étroit sur quand et si de nouvelles données ou des retransmissions sont envoyées.

Quand TCP se connecte ou accepte une connexion à un autre hôte, il effectue un appel cm_open() pour associer la connexion TCP à un cm_streamid.

Une fois qu'une connexion est établie, le CM est utilisé pour contrôler la transmission des donnée sortantes. Le CM élimine le besoin de suivre à la trace et de réagir à l'encombrement dans TCP, parce que le CM et son API de transmission assurent un comportement d'encombrement approprié. La récupération de perte est encore effectuée par TCP sur la base de retransmissions et récupérations rapides ainsi que des temporisations. De plus, TCP est aussi modifié pour avoir sa propre estimation de fenêtre en suspens (tcp_ownd). Chaque fois que des segments de données sont envoyés à partir de son rappel cmapp_send(), TCP met à jour sa valeur de tcp_ownd. La variable ownd est aussi mise à jour après chaque appel cm_update(). TCP tient aussi un compte du nombre de segments en suspens (pkt_cnt). À tout moment, TCP peut calculer la taille moyenne de paquet (avg_pkt_size) comme tcp_ownd/pkt_cnt. avg_pkt_size est utilisé par TCP pour aider à estimer la quantité de données en suspens. Noter que ceci n'est pas nécessaire si l'option SACK est utilisée sur la connexion, car ces informations sont explicitement disponibles.

Les sous-programmes de sortie de TCP sont modifiés comme suit :

1. Toutes les vérifications de fenêtre d'encombrement (cwnd) sont supprimées.

2. Quand des données d'application sont disponibles. Les sous programmes de sortie de TCP effectuent toutes les vérifications non liées à l'encombrement (algorithme de Nagle, vérifications de la fenêtre annoncée par le receveur, etc.). Si ces vérifications réussissent, le sous-programme de sortie met les données en file d'attente et invoque cm_request() pour le flux.

3. Si les données entrantes ou les temporisateurs résultent en la détection d'une perte, la retransmission est aussi placée dans une file d'attente et cm_request() est invoqué pour le flux.

4. Le rappel cmapp_send() pour TCP et réglé sur un sous-programme de sortie. Si une retransmission est en file d'attente, le sous-programme effectue la retransmission. Autrement, le sous-programme sort autant de nouvelles données que l'état de la connexion TCP le permet. Cependant, cmapp_send() n'envoie jamais plus d'un seul segment par appel. Ce sous-programme s'arrange pour que les autres calculs de sortie soient faits, tels que les calculs d'en-tête et d'options.

Le sous-programme de sortie IP de l'hôte invoque cm_notify() lorsque les paquets sont réellement envoyés. Comme il ne sait pas quel cm_streamid est responsable du paquet, cm_notify() prend stream_info comme argument (voir à la Section 4 ce que stream_info devrait contenir). Comme cm_notify() rapporte la taille de la charge utile IP, TCP garde trace de la taille totale d'en-tête et incorpore ces mises à jour.

Les sous-programmes d'entrée de TCP sont modifiés comme suit :

1. L'estimation du RTT est faite comme d'habitude en utilisant soit des horodatages soit l'algorithme de Karn. Toute estimation de rtt qui est générée est passée au CM via l'appel cm_update.

2. Toutes les mises à jour de cwnd et de seuil de démarrage lent (ssthresh) sont supprimées.

3. À l'arrivée d'un ack pour de nouvelles données, TCP calcule la valeur de in_flight (la quantité de données en cours dans le réseau) comme snd_max-ack-1 (c'est-à-dire, MAX Sequence Sent - Current Ack - 1). TCP invoque alors

cm_update(streamid, tcp_ownd - in_flight, 0, CM_NO_CONGESTION, rtt).

4. À l'arrivée d'un accusé de réception dupliqué, TCP doit vérifier son compte dupack (dup_acks) pour déterminer son action. Si dup_acks < 3, le TCP ne fait rien. Si dup_acks == 3, TCP suppose qu'un paquet est perdu et qu'au moins 3 paquets sont arrivés pour générer ces ack dupliqués. Donc, il invoque cm_update(streamid, 4 * avg_pkt_size, 3 * avg_pkt_size, CM_LOSS_FEEDBACK, rtt). La taille moyenne de paquet est utilisée car les accusés de réception n'indiquent pas exactement combien de données ont atteint l'autre côté. La plupart des mise en œuvre TCP interprètent un ACK dupliqué comme une indication qu'une pleine MSS *(Maximum Segment Size, taille de segment maximum)* a atteint sa destination. Quand un nouvel ACK est reçu, ces mises en œuvre d'envoyeur TCP peuvent se resynchroniser avec le receveur TCP. L'API CM ne fournit pas de mécanisme pour que TCP passe les informations sur cette resynchronisation. Donc, TCP peut seulement déduire l'arrivée d'une quantité avg_pkt_size de données de chaque ack dupliqué. TCP met aussi en file d'attente une retransmission du segment perdu et invoque cm_request(). Si dup_acks > 3, TCP suppose qu'un paquet a atteint l'autre côté et a causé l'envoi de cet ack. Il en résulte qu'il invoque cm_update(streamid, avg_pkt_size, avg_pkt_size, CM_NO_CONGESTION, rtt).

5. À l'arrivée d'un accusé de réception partiel (qui n'excède pas le plus fort segment transmis au moment où la perte est survenue, comme défini dans la [RFC2582]) TCP suppose qu'un paquet a été perdu et que le paquet retransmis a atteint le receveur. Donc, il invoque cm_update(streamid, 2 * avg_pkt_size, avg_pkt_size, CM_NO_CONGESTION, rtt). CM_NO_CONGESTION est utilisé car la période de perte a déjà été rapportée. TCP met aussi en file d'attente une retransmission du segment perdu et invoque cm_request().

Lorsque le temporisateur de retransmission TCP expire, l'envoyeur identifie qu'un segment a été perdu et invoque cm_update(streamid, avg_pkt_size, 0, CM_NO_FEEDBACK, 0) pour signifier qu'aucun retour n'a été reçu de la part du receveur et qu'un segment a sûrement "quitté le tuyau". TCP met aussi en file d'attente une retransmission du segment perdu et invoque cm_request().

5.1.2 Contrôle d'encombrement pour UDP

UDP à contrôle d'encombrement est une application utile de CM, qu'on décrit dans le contexte des prises Berkeley [Stevens94]. Elles fournissent les mêmes fonctionnalités que les prises UDP Berkeley standard, mais au lieu d'envoyer immédiatement les données provenant du noyau de la file d'attente de paquets aux couches inférieures pour transmission, la mise en œuvre de mise en mémoire tampon fait des appels à l'API qui sont exportés par le CM à l'intérieur du noyau et obtiennent des rappels de la part du CM. Quand est créée une prise UDP de CM, elle est liée à un flux particulier. Plus tard, quand des données sont ajoutées à la file d'attente des paquets, cm_request() est invoqué sur le flux associé à la prise. Lorsque le CM programme ce flux pour la transmission, il invoque udp_ccappsend() dans le module UDP. Cette fonction transmet une MTU de la file d'attente des paquets, et programme la transmission de tous les paquets restants. La mise en œuvre interne au noyau de l'API UDP de CM ne devrait pas exiger de copies supplémentaires des données et devrait prendre en charge toutes les options UDP standard. Modifier les applications existantes pour utiliser UDP à contrôle d'encombrement exige la mise en œuvre d'une nouvelle option de prise sur la prise. Pour fonctionner correctement, l'envoyeur doit obtenir des retours sur l'encombrement. Cela peut être fait d'au moins deux façons : (i) l'application UDP receveuse peut fournir des retours à l'application envoyeuse, qui va informer le CM des conditions du réseau en utilisant cm_update() ; (ii) la mise en œuvre UDP receveuse peut fournir des retours à l'UDP envoyeur. Noter que cette dernière solution exige des changements à la pile réseau du receveur et que l'envoyeur UDP ne peut pas supposer que tous les receveurs prennent en charge cette option sans négociation explicite.

5.1.3 Serveur audio

Une application audio normale a souvent accès à l'échantillonnage d'une multitude de débits et qualités de données. L'objectif de l'application est alors de livrer la meilleure qualité audio possible (normalement le plus fort débit de données) à ses clients. Le choix de la version audio à transmettre devrait se fonder sur l'état d'encombrement actuel du réseau. De plus, la source va vouloir que l'audio soit livré à ses usagers à un taux d'échantillonnage cohérent. Il en résulte qu'il doit envoyer les données à un taux régulier, minimisant les délais de transmission et réduisant la mise en mémoire tampon avant l'exécution. Pour satisfaire à ces exigences, cette application peut utiliser l'API d'envoyeur synchrone (paragraphe 4.2).

Quand la source commence, elle utilise l'appel cm_query() pour obtenir une estimation initiale de la bande passante et des délais du réseau. Si d'autres flux de ce macroflux ont déjà été actifs, elle obtient alors une estimation initiale qui est valide ; autrement, elle obtient des valeurs négatives, qu'elle ignore. Elle choisit ensuite un codage qui n'excède pas ces estimations (ou, dans le cas d'une estimation invalide, elle utilise les valeurs initiales spécifiques de l'application) et elle commence à émettre les données. L'application met aussi en œuvre le rappel cmapp_update(). Lorsque le CM détermine que les caractéristiques du réseau ont changé, il invoque la fonction cmapp_update() de l'application et la passe à un nouveau débit et une nouvelle estimation de délai d'aller-retour. L'application doit changer son choix de codage audio pour s'assurer qu'il n'excède pas ces nouvelles estimations.

5.2 Exemple de module de contrôle d'encombrement

Pour illustrer les responsabilités d'un module de contrôle d'encombrement, ce qui suit décrit certaines des actions d'un simple module de contrôle d'encombrement de style TCP qui met en œuvre le contrôle d'encombrement à augmentation additive, diminution multiplicative (AIMD_CC) :

- query() : AIMD_CC retourne la fenêtre d'encombrement actuelle (cwnd) divisée par le rtt lissé (srtt) comme estimation de la bande passante. Il retourne l'estimation du rtt lissé comme srtt.
- notify() : AIMD_CC ajoute le nombre d'octets envoyés à sa fenêtre de données en cours (ownd).
- update() : AIMD_CC soustrait nsent de ownd. Si la valeur de rtt est différente de zéro, AIMD_CC met à jour srtt en utilisant le calcul TCP de srtt. Si la mise à jour indique que des données ont été perdues, AIMD_CC règle cwnd à 1 MTU si loss_mode est CM_NO_FEEDBACK et à cwnd/2 (avec un minimum de 1 MTU) si loss_mode est CM_LOSS_FEEDBACK ou CM_EXPLICIT_CONGESTION. AIMD_CC règle aussi sa variable ssthresh interne à cwnd/2. Si aucune perte n'est intervenue, AIMD_CC imite les modes de démarrage lent et de croissance linéaire de TCP. Il incrémente cwnd de nsent lorsque cwnd < ssthresh (bordé par un maximum de ssthresh-cwnd) et de nsent * MTU/cwnd lorsque cwnd > ssthresh.
- Lorsque cwnd ou ownd sont mis à jour et indiquent qu'au moins une MTU peut être transmise, AIMD_CC appelle le CM pour programmer une transmission.

5.3 Exemple de module programmeur

Pour préciser les responsabilités d'un module programmeur, on décrit certaines des actions d'un module simple de programmeur round robin (RR_sched) :

- schedule() : RR_sched programme autant de flux que possible en round robin.
- query_share() : RR_sched retourne 1/(nombre de flux dans le macroflux).
- notify() : RR_sched ne fait rien. La programmation round robin n'est pas affectée par la quantité de données envoyée.

6. Considérations pour la sécurité

Le gestionnaire d'encombrement (CM) fournit beaucoup des mêmes services que le contrôle d'encombrement dans TCP. À ce titre, il est vulnérable à beaucoup des mêmes problèmes de sécurité. Par exemple, des rapports de pertes et de transmissions incorrects vont donner au CM une image imprécise de l'état d'encombrement du réseau. En donnant au CM une estimation forte de l'encombrement, un attaquant peut dégrader les performances observées par les applications. Par exemple, un flux sur un hôte peut ralentir arbitrairement tout autre flux sur le même macroflux, une forme de déni de service.

La forme la plus dangereuse d'attaque survient lorsque une application donne au CM une estimation faible de l'encombrement. Cela amènerait le CM à être trop agressif et à permettre que les données soient envoyées beaucoup plus vite que ne le permettait des politiques raisonnables de contrôle d'encombrement.

La [RFC2140] décrit un certain nombre des problèmes de sécurité qui surviennent avec le partage des informations d'encombrement. Une faiblesse supplémentaire (non couverte par la [RFC2140]) apparaît parce que les applications ont accès à travers l'API de CM à l'état partagé de contrôle qui va affecter les autres applications sur le même ordinateur. Par exemple, une application UDP mal conçue, éventuellement compromise, ou intentionnellement malveillante pourrait détourner cm_update() pour affamer et/ou causer un comportement trop agressif aux autres utilisateurs du macroflux.

7. Références

[Andersen00] Balakrishnan, H., "System Support for Bandwidth Management and Content Adaptation in Internet Applications", Proc. 4th Symp. on Operating Systems Design and Implementation, San Diego, CA, octobre 2000. Disponible à http://nms.lcs.mit.edu/papers/cm-osdi2000.html

[Balakrishnan98] Balakrishnan, H., Padmanabhan, V., Seshan, S., Stemm, M., and Katz, R., "TCP Behavior of a Busy Web Server: Analysis and Improvements", Proc. IEEE INFOCOM, San Francisco, CA, mars 1998.

[Balakrishnan99] Balakrishnan, H., Rahul, H., and Seshan, S., "An Integrated Congestion Management Architecture for Internet Hosts", Proc. ACM SIGCOMM, Cambridge, MA, septembre 1999.

[Clark90] Clark, D. and Tennenhouse, D., "Architectural Consideration for a New Generation of Protocols", Proc. ACM

SIGCOMM, Philadelphia, PA, septembre 1990.

[Eggert00] Eggert, L., Heidemann, J., and Touch, J., "Effects of Ensemble TCP", ACM Computer Comm. Review, janvier 2000.

[Floyd99a] Floyd, S. and Fall, K.," Promoting the Use of End-to-End Congestion Control in the Internet", IEEE/ACM Trans. on Networking, 7(4), août 1999, pp. 458-472.

[Jacobson88] Jacobson, V., "Congestion Avoidance and Control", Proc. ACM SIGCOMM, Stanford, CA, août 1988.

[Mahdavi98] Mahdavi, J. and Floyd, S., "The TCP Friendly Website", http://www.psc.edu/networking/tcp_friendly.html

[Padmanabhan98] Padmanabhan, V., "Addressing the Challenges of Web Data Transport", Thèse de doctorat, Univ. of California, Berkeley, décembre 1998.

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

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

[RFC2026] S. Bradner, "Le processus de normalisation de l'Internet -- Révision 3", (BCP0009) octobre 1996. (*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.

[RFC2140] J. Touch, "Interdépendance de bloc de contrôle TCP", avril 1997. *(Information)*

[RFC2481] K. Ramakrishnan S. Floyd, "Proposition d'ajout de la notification d'encombrement explicite (ECN) à IP", janvier 1999.

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

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

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

[Stevens94] Stevens, W., "TCP/IP Illustrated, Volume 1". Addison-Wesley, Reading, MA, 1994.

8. Remerciements

Nous remercions David Andersen, Deepak Bansal, et Dorothy Curtis de leur travail sur la conception et la mise en œuvre du gestionnaire d'encombrement. Nous remercions Vern Paxson de ses commentaires détaillés, de ses réactions, et de sa patience, et Sally Floyd, Mark Handley, et Steven McCanne pour les réactions utiles sur l'architecture du CM. Allison Mankin et Joe Touch ont fourni plusieurs commentaires utiles sur des projets précédents de ce document.

9. Adresse des auteurs

Hari Balakrishnan
Laboratory for Computer Science
200 Technology Square
Massachusetts Institute of Technology
Cambridge, MA 02139
mél : hari@lcs.mit.edu
Web: http://nms.lcs.mit.edu/~hari/

Srinivasan Seshan
School of Computer Science
Carnegie Mellon University
5000 Forbes Ave.
Pittsburgh, PA 15213
mél : srini@cmu.edu
Web: http://www.cs.cmu.edu/~srini/

Propriété intellectuelle

L'IETF ne prend pas position sur la validité et la portée de tout droit de propriété intellectuelle ou autres droits qui pourrait 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 n'être pas disponible ; pas plus qu'elle ne prétend avoir accompli aucun effort pour identifier de tels droits. Les informations sur les procédures de l'ISOC au sujet des droits dans les documents de l'ISOC figurent dans les BCP 78 et BCP 79.

Des copies des dépôts d'IPR faites au secrétariat de l'IETF et toutes assurances de disponibilité de licences, ou le résultat de tentatives faites pour obtenir une licence ou permission générale d'utilisation de tels droits de propriété par ceux qui mettent en œuvre ou utilisent la présente spécification peuvent être obtenues sur répertoire en ligne des IPR de l'IETF à http://www.ietf.org/ipr.

L'IETF invite toute partie intéressée à porter son attention sur tous copyrights, licences ou applications de licence, ou autres droits de propriété qui pourraient couvrir les technologies qui peuvent être nécessaires pour mettre en œuvre la présente norme. Prière d'adresser les informations à l'IETF à ietf-ipr@ietf.org.

Déclaration complète de droits de reproduction

Remerciement

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