Groupe de travail Réseau
Request For Comments : 815
Traduction Claude Brière de L'Isle

David D. Clark, MIT Laboratory for Computer Science
juillet 1982

Algorithmes de réassemblage de datagrammes IP

1. Introduction

Un des mécanismes de IP est la fragmentation et le réassemblage. Dans certaines circonstances, un datagramme transmis à l'origine comme une seule unité va arriver à sa destination finale coupé en plusieurs fragments.

La couche IP chez l'hôte récepteur doit accumuler ces fragments jusqu'à ce qu'assez en soient arrivés pour reconstituer complètement le datagramme d'origine.

Le document de spécification de IP donne une description complète du mécanisme de réassemblage et contient plusieurs exemples. Il donne aussi un algorithme possible pour le réassemblage, fondé sur le suivi des fragments arrivants dans un vecteur de bits. Le présent document décrit une autre approche qui devrait se révéler plus convenable pour certaines machines.

Un examen superficiel du processus de réassemblage peut suggérer qu'il est assez compliqué. D'abord, il est nécessaire de garder trace de tous les fragments, ce qui suggère un petit travail de comptabilité. Ensuite, lorsque un nouveau fragment arrive, il peut se combiner avec les fragments existants d'un certain nombre de façons différentes. Il peut précisément remplir l'espace entre deux fragments, ou il peut se chevaucher avec des fragments existants, ou dupliquer complètement des fragments existants, ou remplir partiellement un espace entre deux fragments sans s'abouter à l'un ou l'autre. Donc, il peut sembler que le processus de réassemblage pourrait impliquer de concevoir un algorithme très compliqué qui vérifie un certain nombre d'options différentes.

En fait, le processus de réassemblage est extrêmement simple. Le présent document décrit une façon de traiter le réassemblage qui réduit le problème de la comptabilité à un minimum, qui exige pour la mémorisation seulement une mémoire tampon égale en taille au datagramme final à réassembler, qui peut réassembler un datagramme à partir de tout nombre de fragments arrivant dans n'importe quel ordre avec tous les schémas possibles de chevauchement et de duplication, et qui est approprié pour presque toutes les sortes de systèmes d'exploitation.

Le lecteur devrait consulter le document de spécification du protocole IP pour être sûr qu'il est complètement familiarisé avec le concept général de réassemblage, et avec les champs d'en-tête particuliers et le vocabulaire utilisé pour décrire le processus.

2. L'algorithme

Afin de définir cet algorithme de réassemblage, il est nécessaire de définir quelques termes. Un datagramme partiellement réassemblé consiste en certaines séquences d'octets qui sont déjà arrivées, et certaines zones qui sont encore à venir. On va appeler ces zones manquantes des "trous". Chaque trou peut être caractérisé par deux nombres, "premier.trou", le numéro du premier octet dans le trou, et "dernier.trou", le numéro du dernier octet dans le trou. Cette paire de nombres sera appelée le "descripteur de trou", et on va supposer que tous les descripteurs de trou pour un datagramme particulier sont rassemblés dans la "liste de descripteurs de trou".

La forme générale de l'algorithme est la suivante. Lorsque un nouveau fragment du datagramme arrive, il peut remplir un ou plusieurs des trous existants. On va examiner chacune des entrées dans la liste de descripteurs de trou pour voir si le trou en question est éliminé par ce fragment entrant. Si oui, on va supprimer cette entrée de la liste. Finalement, un fragment va arriver qui élimine toutes les entrées de la liste. À ce point, le datagramme a été complètement réassemblé et peut être passé aux niveaux supérieurs du protocole pour la suite du traitement.

L'algorithme va être décrit en deux phases. Dans la première partie, on va montrer la séquence des étapes qui sont exécutées lorsque un nouveau fragment arrive, afin de déterminer si un des trous existants est rempli ou non par le nouveau fragment. Dans la seconde partie de cette description, on va montrer un algorithme d'une simplicité ridicule pour la gestion de la liste de descripteurs de trou.

3. Algorithme de traitement de fragment

Un fragment arrivant peut remplir n'importe lequel des trous existants d'un certain nombre de façons. Le plus simplement, il peut boucher complètement un trou. Autrement, il peut laisser un certain espace restant soit au début, soit à la fin d'un trou existant. Ou finalement, il peut se poser au milieu d'un trou existant, coupant le trou en deux et laissant un plus petit trou à chaque bout.

À cause de ces possibilités, il peut sembler qu'un certain nombre de vérifications doivent être faites lorsque arrive un nouveau fragment, ce qui conduit à un algorithme assez compliqué. En fait, si il est exprimé de façon appropriée, l'algorithme peut comparer chaque trou au fragment arrivant avec seulement quatre vérifications.

On commence l'algorithme lorsque arrive le premier fragment du datagramme. On commence par créer une zone vide de mémoire tampon de données et en mettant une entrée dans sa liste de descripteurs de trou, l'entrée qui décrit le datagramme comme étant complètement manquant. Dans ce cas, premier.trou égale zéro, et dernier.trou égale l'infini. (Infini est supposé mis en œuvre par un très grand entier, supérieur à 576, au choix de la mise en œuvre.)

Le huit étapes suivantes sont alors parcourues pour insérer chacun des fragments arrivant dans la zone de mémoire tampon où le datagramme complet va être construit. Le fragment arrivant est décrit par premier.fragment, le premier octet du fragment, et dernier.fragment, le dernier octet du fragment.

1. Choisir le descripteur du prochain trou à partir de la liste de descripteurs de trou. Si il n'y a plus d'entrées, passer à l'étape huit.
2. Si premier.fragment est supérieur à dernier.trou, passer à l'étape une.
3. Si dernier.fragment est inférieur à premier.trou, passer à l'étape une.
   (Si les étapes deux ou trois sont vraies, le fragment qui vient d'arriver ne se recouvre d'aucune façon avec le trou, de sorte qu'on a pas besoin de plus s'occuper de ce trou. On retourne au début de l'algorithme et on choisit le prochain trou pour l'examiner.)
4. Supprimer l'entrée actuelle de la liste de descripteurs de trou.
   (Comme ni l'étape deux ni l'étape trois n'étaient vraies, le fragment qui vient d'arriver n'interagit en aucune façon avec ce trou. Donc, le descripteur courant ne sera plus valide. On va le détruire, et dans les deux étapes suivantes on va déterminer si il est ou non nécessaire de créer de nouveaux descripteurs de trou.)
5. Si premier.fragment est supérieur à premier.trou, créer alors un nouveau descripteur de trou "nouveau_trou" avec premier.nouveau_trou égal à premier.trou, et dernier.nouveau_trou égal à premier.fragment moins un.
   (Si la vérification de l'étape cinq est vraie, la première partie du trou original n'est alors pas comblée par ce fragment. On crée un nouveau descripteur pour ce plus petit trou.)
6. Si dernier.fragment est inférieur à dernier.trou, et si autre.fragment est vrai, créer alors un nouveau descripteur de trou "nouveau_trou", avec premier.nouveau_trou égal à dernier.fragment plus un et dernier.nouveau_trou égal à dernier.trou.
   (Cette vérification est le reflet de l'étape cinq avec une caractéristique supplémentaire. Initialement, on ne sait pas quelle longueur aura la datagramme réassemblé, et on a donc créé un trou allant de zéro à l'infini. On va finalement recevoir le dernier fragment du datagramme. À ce moment, le descripteur de trou qui va du dernier octet de la mémoire tampon à l'infini peut être éliminé. Le fragment qui contient le dernier fragment indique ce fait par un fanion dans l'en-tête Internet appelé "Autres fragments". La vérification de ce bit dans cette déclaration nous empêche de créer un descripteur pour le trou inutile qui décrit l'espace entre la fin du datagramme et l'infini.)
7. Passer à l'étape une.
8. Si la liste de descripteur de trou est maintenant vide, le datagramme est alors complet. Le passer au processeur du protocole de niveau supérieur pour la suite du traitement. Autrement, revenir à l'étape une.

4. Deuxième partie : gestion de la liste de descripteur de trou

La principale complexité de l'algorithme en huit étapes ci-dessus n'est pas d'effectuer les vérifications arithmétiques, mais d'ajouter et de supprimer les entrées de la liste des descripteurs de trou. On pourrait imaginer une mise en œuvre dans laquelle le paquetage de gestion de la mémorisation sera bien plus compliqué que le reste de l'algorithme, car il n'y a pas de limite supérieure spécifiée au nombre de descripteurs de trous qui vont exister pour un datagramme durant le réassemblage. Il y a cependant une façon très simple pour traiter les descripteurs de trou. Il suffit de mettre chaque descripteur de trou dans les premiers octets du trou lui-même. Noter que d'après la définition de l'algorithme de réassemblage, la taille minimum d'un trou est de huit octets. Pour mémoriser premier.trou et dernier.trou on peut supposer deux octets chacun. Deux octets supplémentaires seront requis pour lier ensemble les entrées sur la liste de descripteurs de trou. Cela laisse au moins deux octets de plus pour traiter les idiosyncrasies de la mise en œuvre.

Il y a seulement un piège évident dans cette stratégie de mémorisation. On doit exécuter les huit étapes de l'algorithme ci-dessus avant de copier les données du fragment dans la mémoire tampon de réassemblage. Si on copiait d'abord les

données, elles pourraient écraser un ou plusieurs descripteurs de trou. Une fois que l'algorithme ci-dessus a été effectué, tout les descripteurs de trou qui devraient être écrasés ont déjà été rendus obsolètes.

5. Extrémités libres

Éparpiller les descripteurs de trou dans la mémoire tampon de réassemblage elle-même exige qu'ils soient liés par une sorte de liste afin qu'on puisse les trouver. Cela implique à son tour qu'il doit y avoir un pointeur sur la tête de liste. Dans de nombreux cas, ce pointeur peut être mémorisé dans une sorte de bloc descripteur que la mise en œuvre associe à chaque mémoire tampon de réassemblage. Si une telle mémorisation n'est pas disponible, un truc sordide mais efficace est de mémoriser la tête de la liste dans une partie de l'en-tête Internet dans la mémoire tampon de réassemblage qui n'est plus nécessaire. Une localisation évidente est le champs de somme de contrôle.

Lorsque le fragment final du datagramme arrive, le champ Longueur du paquet dans l'en-tête Internet devrait être rempli.

6. Options

La description précédente a fait une simplification inacceptable. Elle suppose qu'il n'y a pas d'option Internet associée au datagramme qu'on veut réassembler. La difficulté avec les options est que jusqu'à ce qu'on reçoive le premier fragment du datagramme, on ne peut pas dire quelle sera la taille de l'en-tête Internet. Cela parce que, bien que certaines options soient copiées à l'identique dans chaque fragment d'un datagramme, d'autres options, comme "record route", sont mises dans le seul premier fragment. (Le "premier fragment" est celui qui contient zéro octet du datagramme original.)

Jusqu'à ce qu'on sache la taille de l'en-tête Internet, on ne sait pas où copier les données de chaque fragment dans la mémoire tampon de réassemblage. Si le premier fragment à arriver se trouve être le premier fragment, ce n'est alors pas un problème. Autrement, il y a deux solutions. D'abord, on peut laisser de l'espace dans la mémoire tampon de réassemblage pour le maximum possible d'en-tête Internet. En fait, la taille maximum n'est pas très grande, 64 octets. Autrement, on peut simplement parier que le premier fragment ne va pas contenir d'options. Si, lorsque le premier fragment arrive finalement, il y a des options, on peut alors faire glisser les données dans la mémoire tampon à une distance suffisante pour permettre de les intégrer. Le seul péril au moment de la copie des données est que certaines perturbent les pointeurs qui lient ensemble les descripteurs de trou. Il est aisé de voir comment démêler les pointeurs.

Les options chemin de source et enregistrement de chemin ont une caractéristique intéressante, car différents fragments peuvent suivre des chemins différents, ils peuvent arriver avec des chemins de retour différents enregistrée dans les différents fragments. Normalement, ces informations sont plus que ce dont a besoin le module Internet receveur. La procédure spécifiée est de prendre le chemin de retour enregistré dans le premier fragment et d'ignorer les autres versions.

7. Algorithme complet

En plus de l'algorithme décrit ci-dessus, il y a deux parties au processus de réassemblage. D'abord, lorsque un fragment arrive, il est nécessaire de trouver la mémoire tampon de réassemblage associée à ce fragment. Cela requiert qu'un mécanisme recherche toutes les mémoires tampon de réassemblage existantes. La mémoire tampon de réassemblage correcte est identifiée par l'égalité des champs suivants : Adresse Internet distante et locale, Identifiant de protocole, et Identification. La partie finale de l'algorithme est une sorte de temporisateur fondé sur un mécanisme qui décrémente le champ Durée de vie pour chaque datagramme partiellement réassemblé, de sorte que les datagrammes incomplets qui ont outrepassé leur durée de vie peuvent être détectés et supprimés. On peut créer un automate qui s'active une fois par seconde et décrémente de un le champ dans chaque datagramme, ou on peut lire l'horloge lorsque chaque premier fragment arrive, et mettre en file d'attente une sorte d'appel au temporisateur, en utilisant un mécanisme système approprié, pour supprimer le datagramme lorsque son heure est venue.

Une mise en œuvre de l'algorithme complet comprenant toutes ces parties a été construite dans BCPL à titre d'essai. L'algorithme complet a pris moins d'une demie page de listing, et a généré approximativement 400 instructions machine. Cette portion de l'algorithme effectivement impliquée dans la gestion de descripteur de trou est d'environ 20 lignes de code.

La version de l'algorithme décrite ici est en fait une simplification de la version originale de l'auteur, grâce à une observation pertinente de Elizabeth Martin du MIT.