Guide pratique du multicast sur les réseaux TCP/IP

Adaptation française du *Multicast over TCP/IP HOWTO*

Juan-Mariano de Goyeneche `<jmseyas CHEZ dit POINT upm POINT es>`

Adaptation française: Antoine Duval

`<aduval CHEZ altern POINT org>`

Relecture de la version française: Fabrice Gadaud

`<fabrice CHEZ gadaud POINT org>`

Préparation de la publication de la v.f.: Jean-Philippe Guérard

`<jean TIRET philippe POINT guerard CHEZ corbeaunoir POINT org>`

Version : 1.0.fr.1.1

14 octobre 2003

Historique des versions

Version v1.0.fr.1.1	2003-10-14	JPG
Ajout de titres aux renvois inclus dans le texte. Reformatage des exemples de code.		
Version v1.0.fr.1.0	2003-09-30	AD,FG,JPG
Première version française.		
Version v1.0	1998-03-20	JMdG

Ce guide pratique couvre les principaux aspects relatifs au multicast sur les réseaux TCP/IP. De nombreuses informations contenues dans ce document ne sont pas spécifiques à Linux (au cas où vous n'utiliserez pas encore le système Linux/GNU…). Le multicast est encore un domaine de recherche et les standards ne sont encore que des brouillons. Gardez cela à l'esprit à la lecture des lignes qui suivent.

Table des matières

1. Introduction

Dans ce document, je vais essayer de vous donner des informations les plus à jour possible sur le multicast au travers des réseaux TCP/IP. Tout retour est le bienvenu. Si vous trouvez des erreurs dans ce document, si vous avez des commentaires ou des mises à jour à faire quant à son contenu, n'hésitez pas à m'en faire part à l'adresse que vous trouverez dans l'entête du document.

1.1. Qu'est ce que le multicast ?

Le multicast est… un besoin. Ou du moins dans certains cas. Si vous avez des informations (beaucoup d'informations, le plus souvent) qui doivent être transmises à un ensemble de machines (mais le plus souvent pas toutes) au travers d'internet, alors le multicast est une bonne solution. Un cas typique d'utilisation du multicast, est la diffusion audio ou vidéo temps réel à un ensemble de machines donné qui ont précédemment rejoint une conférence distribuée.

Il est à noter que le multicast ressemble à la radio ou à la télévision, en ce sens que seuls les utilisateurs qui ont réglé leur récepteur (en sélectionnant une fréquence particulière) reçoivent l'information. De ce fait, vous n'entendez que le canal qui vous intéresse, pas les autres.

1.2. Le problème d'unicast

Unicast est tout ce qui n'est ni du broadcast ni du multicast. Bien sûr, cette définition n'est pas vraiment claire… Expliquons nous : lorsque vous envoyez un paquet et qu'il y a seulement un émetteur (vous) et un récepteur (la personne à qui vous envoyez le paquet), alors il s'agit d'un mode de transmission unicast. TCP est par nature orienté unicast. UDP est déclinable selon plus de modèles, mais si vous envoyez des paquets UDP et qu'il n'y a qu'une personne qui est censée les recevoir, alors il s'agit aussi d'une transmission unicast.

Pendant des années les transmissions unicast ont semblé être suffisantes pour Internet. Cela fut le cas jusqu'en 1993 où la première mise en œuvre du multicast vit le jour à l'occasion de la parution de la distribution BSD 4.4. Apparemment personne n'en avait jamais eu besoin avant cette date. Quels sont donc les nouveaux problèmes liés au multicast ?

Il n'est peut-être pas nécessaire de dire ici, qu'internet a beaucoup changé depuis ses « premiers jours ». Particulièrement, l'apparition du web a considérablement transformé la situation : les personnes ne voulaient plus seulement se connecter à des hôtes distants, serveurs de mails ou encore FTP. Ils voulaient maintenant voir les images des personnes qu'ils côtoient sur les pages de leurs sites web personnels, et plus tard les voir et les entendre.

Avec la technologie d'aujourd'hui il est possible de supporter le coût d'établissement d'une connexion avec ceux qui veulent voir votre page web personnelle. Cependant, si vous diffusez de l'audio et de la vidéo, vous aurez besoin d'un volume de bande passante énorme comparé aux applications web. Vous avez alors -ou vous aviez, si vous utilisez déjà le multicast- deux solutions : établir une connexion unicast avec chacun de vos destinataires, ou utiliser le broadcast. La première solution n'est pas envisageable : si nous nous accordons sur le fait qu'une seule connexion audio/vidéo consomme énormément de bande passante, imaginez alors le volume consommé par une centaine ou un millier de ces connections. Aussi bien l'ordinateur, émetteur des données, que le réseau lui étant lié succomberait [ndt : dans une grande souffrance].

Le broadcast, ou envoi par diffusion, semble être une solution, mais pas nécessairement la solution. Si vous souhaitez que toutes les machines de votre réseau écoutent la conférence, vous pouvez utiliser le broadcast. Dans ce cas, les paquets ne sont émis qu'une seule fois et chaque hôte les reçoit sur une adresse de broadcast ; adresse sur laquelle ils pourront, de la même manière, émettre. Cependant, il se peut que seulement peu d'utilisateurs soient intéressés par ces paquets. Plus encore, il se peut que certains utilisateurs soient vraiment intéressés par votre conférence, mais qu'ils se trouvent en dehors de votre réseau local, à seulement quelques routeurs de là. Vous n'êtes pas sans savoir que si le broadcast fonctionne bien à l'intérieur d'un réseau local, des problèmes surviennent lorsque vous désirez diffuser des paquets qui doivent être routés au travers de différents réseaux locaux.

La meilleure solution serait alors d'émettre des paquets à une adresse spéciale, telle une fréquence donnée comme dans le cas d'une transmission radio ou de télévision. Dans ce cas, tous les hôtes qui ont décidé de joindre la conférence seront demandeurs des paquets portant cette adresse de destination ; ils les lisent lorsqu'ils passent sur le réseau et les transmettent à la couche IP pour qu'ils soient démultiplexés. Cette technique est similaire au broadcast en ce sens que vous n'émettez qu'un seul paquet en diffusion et que tous les hôtes du réseau peuvent le reconnaître et le lire ; elle est différente en ce sens que tous les paquets multicast ne sont pas lus et traités, mais seulement ceux qui ont été précédemment enregistrés dans le noyau comme étant intéressants.

Ces paquets spéciaux sont routés au niveau du noyau comme tout autre paquet car il s'agit de paquets IP. L'unique différence réside dans l'algorithme de routage qui indique au noyau qui doit être routé et que faire des paquets à router.

2. Explications relatives au multicast

2.1. Adresses multicast

Comme vous le savez probablement, les adresses IP sont divisées en « classes ». Ces classes sont basées sur l'ordre des bits de poids fort de l'adresse IP :

Bit ->	0					31	Plage d'adresses :
	0	Adresses de classe A					0.0.0.0 - 127.255.255.255
	1	0	Adresses de classe B				128.0.0.0 - 191.255.255.255
	1	1	0	Adresses de classe C			192.0.0.0 - 223.255.255.255
	1	1	1	0	Adresses multicast		224.0.0.0 - 239.255.255.255
	1	1	1	1	0	Réservé	240.0.0.0 - 247.255.255.255

La classe qui nous intéresse est la classe D. Chaque datagramme dont l'adresse de destination (codée sur 32 bits) commence par « 1110 » est un datagramme IP Multicast.

Les 28 autres bits identifient le *groupe* auquel le datagramme est envoyé. Poursuivant avec la précédente analogie, il est nécessaire de régler sa radio pour entendre un programme qui est retransmis sur une fréquence donnée. Dans notre cas, il convient de procéder de la même façon : vous devez « régler » votre noyau pour qu'il reçoive les paquets qui sont émis à destination d'un groupe multicast donné. Lorsque cette opération est effectuée, il est dit que l'hôte a *joint* ce groupe ; ceci sur une interface spécifiée : vous trouverez plus d'informations sur ce propos par la suite.

Il existe des groupes multicast spéciaux, dits « groupes multicast bien connus », vous ne devez pas les utiliser pour vos applications, du fait de l'usage bien spécifique auquel ils sont destinés :

224.0.0.1
: est le groupe *all-hosts*. Si vous pingez ce groupe, tous les hôtes sur le réseau capables d'émettre en multicast répondront, car tout hôte gérant le multicast *doit* joindre ce groupe au démarrage sur toutes ses interfaces.

224.0.0.2
: est le groupe *all-routers*. Tous les routeurs multicast joignent ce groupe sur toutes leurs interfaces multicast.

224.0.0.4
: est le groupe de tous les routeurs DVMRP.

224.0.0.5
: est le groupe de tous les routeurs OSPF.

224.0.0.13
: est le groupe de tous les routeurs PIM, etc.

Tous ces groupes multicast spéciaux sont régulièrement publiés dans le RFC « Assigned Numbers »

Dans tous les cas, l'intervalle allant de 224.0.0.0 à 224.0.0.255 est reservé pour les contenus locaux (comme des tâches administratives ou de maintenance) et les datagrammes destinés à ces adresses ne sont jamais retransmis par les routeurs multicast. De la même manière, l'intervalle allant de 239.0.0.0 à 239.255.255.255 est reservé pour des raisons d'administration (cf section 2.3.1 pour de plus amples imformations à ce propos).

2.2. Niveaux de conformité

Il existe trois niveaux de conformité différents, que les hôtes peuvent implémenter, en accord avec la spécification multicast, permettant de gérer les différents services mis en commun.

Niveau 0
: signifie « qu'aucun support n'est disponible pour l'IP multicast ». Beaucoup d'hôtes et de routeurs sur Internet sont dans cet état, car le support multicast n'est pas exigé pour l'IPv4 (il l'est cependant pour l'IPv6). Ainsi, les hôtes qui sont dans cet état ne peuvent ni émettre, ni recevoir de paquets multicast. Ils ignorent donc les paquets émis par les autres hôtes pouvant émettre en multicast.

Niveau 1
: signifie « qu'il existe un support pour envoyer mais pas pour recevoir des datagrammes IP multicast ». Dans ce cas, il n'est pas nécessaire de joindre un quelconque groupe multicast pour envoyer des datagrammes. Peu d'ajouts sont nécessaires à un module IP pour passer un hôte du « niveau 0 » au « compatible niveau 1 », comme on pourra le voir dans la section 2.3.

Niveau 2
: signifie qu'un « support complet est disponible pour l'IP multicast ». Les hôtes de niveau 2 sont aussi bien capables d'émettre, que de recevoir du trafic multicast. Ils savent comment joindre et quitter les groupes multicast, ainsi que propager cette information aux routeurs multicasts. De même, ils incluent une implémentation de IGMP (Internet Group Management Protocol) dans leur pile de protocoles TCP/IP.

2.3. Envoyer des datagrammes multicast.

Il est évident que le trafic multicast doit être encapsulé par une couche transport comme UDP. En effet, TCP fournit des connections point-à-point, ce qui n'est pas envisageable pour du trafic multicast. Il est à noter que des recherches sont effectuées dans ce domaine pour définir et implémenter un nouveau protocole de transport orienté multicast. Consultez la section Protocoles de transport multicast pour plus de détails.

En principe, une application nécessite seulement l'ouverture d'une prise réseau UDP qu'elle remplit de données, qui seront envoyées à destination d'une adresse multicast (classe D). Cependant, certaines opérations doivent être contrôlées lors du processus d'envoi.

2.3.1. TTL

Le champ *TTL* (*Time To Live*, ou temps restant à vivre) de l'entête IP a une double signification pour le multicast. Comme toujours, il contrôle le temps restant à vivre pour un datagramme empêchant ainsi les boucles infinies dues aux erreurs de routage. Chaque routeur décrémente le TTL de tout datagramme le traversant et lorsque cette valeur devient égale à 0 le paquet est détruit.

Le TTL dans notre cas (sur IPv4) a aussi une signification de « seuil ». Éclairons son utilisation à l'aide d'un exemple : supposez que vous fassiez une longue conférence vidéo (consommant beaucoup de bande passante) entre tous les hôtes de votre département. Vous désirez que tout ce trafic puisse être supporté par votre réseau local. Peut-être que votre département est suffisamment grand pour avoir différents réseaux locaux. Dans ce cas vous désirerez sûrement que les hôtes appartenant à chacun de vos réseaux locaux puissent suivre la conférence, sans pour autant saturer Internet tout entier avec votre trafic multicast. Il est donc nécessaire de limiter la portée du trafic multicast au travers des routeurs. C'est ce à quoi est destiné le TTL. Chaque routeur a un seuil TTL assigné sur chaque interface, et seuls les datagrammes qui ont un TTL supérieur au seuil de l'interface sont retransmis au travers de ce routeur. Notez que lorsqu'un datagramme traverse un routeur avec une valeur de seuil donnée, le TTL du datagramme n'est *pas* décrémenté de la valeur de ce seuil. Seule une comparaison est faite. (Comme précedemment, le TTL est décrémenté de 1 à chaque passage au travers d'un routeur).

Une liste de seuils TTL et de leur signification suit :

TTL	Signification
0	Restriction au même hôte. Le paquet ne sera émis sur aucune interface.
1	Restriction au même sous réseau. Le paquet ne pourra être routé
<32	Restriction au même site, organisation ou département.
<64	Restriction à la même région.
<128	Restriction au même continent.
<255	Aucune restriction. Global.

Aucune signification exacte n'a été donnée au terme de « site » ou de « région ». Il est à la charge des administrateurs de décider des limites qui s'appliquent.

Il est à noter que l'astuce liée au TTL n'est peut être pas toujours assez souple pour correspondre à tous les besoins, et plus spécialement s'il est nécessaire de traiter avec des régions qui se chevauchent et prenent en compte à la fois des limites géographiques, topologiques ou encore liées à une gestion de bande passante. Afin de résoudre ce problème, des régions multicast à caractère administratif ont été établies depuis 1994. (cf D.Meyer's « Administratively Scoped IP Multicast » Internet draft). Ce découpage est basé sur les adresses multicast au lieu des TTL. L'intervalle allant de 239.0.0.0 à 239.255.255.255 est reservé à ce découpage administratif.

2.3.2. Loopback

Lorsqu'un hôte est conforme au niveau 2 et qu'il est membre du groupe auquel il envoie des datagrammes, alors une copie des datagrammes lui est retransmise par défaut. Cela ne signifie pas que l'interface lit sa propre transmission depuis le réseau (en reconnaissant les datagrammes comme étant d'un groupe auquel appartient l'interface). Au contraire, c'est la couche IP qui, par défaut, reconnaît le datagramme à envoyer, le copie puis le met sur la file d'entrée IP avant de l'envoyer.

Cette fonctionnalité est appréciable dans certains cas, mais pas pour tous. De ce fait il est possible d'activer ou de désactiver ce processus d'envoi selon ses souhaits.

2.3.3. Sélection de l'interface.

Les hôtes qui sont reliés à plus d'un réseau doivent pouvoir décider de l'interface réseau à utiliser pour envoyer les informations. Si rien n'est spécifié, alors le noyau en choisit une par défaut. Ce choix est basé sur la configuration faite par l'administrateur.

2.4. Recevoir des datagrammes multicast

2.4.1. Joindre un groupe multicast

La diffusion en broadcast est (en comparaison) plus simple à implémenter que la diffusion en multicast. En effet dans le premier cas, il n'est pas nécessaire que les processus donnent au noyau des règles concernant la gestion des paquets diffusés. Le noyau sait ce qu'il doit faire avec *tous* les paquets : les lire et les délivrer aux applications concernées.

En diffusion multicast il est nécessaire d'aviser le noyau des groupes multicast qui nous intéressent. Le noyau « joint » alors ces groupes. Puis selon la nature du matériel sous-jacent, les datagrammes multicast sont filtrés soit par le matériel, soit par la couche IP (et dans certains cas, par les deux). Seuls les datagrammes dont le groupe de destination a précédemment été rejoint sont acceptés.

Essentiellement, lorsque l'on joint un groupe, nous disons au noyau « D'accord, je sais que par défaut tu ignores les datagrammes multicast, mais souviens toi que *ce* groupe multicast m'intéresse. De ce fait, lit et délivre (à tous les processus intéressés, pas seulement moi) tous les datagrammes que tu vois sur ton interface réseau et qui sont adressés au groupe multicast en question ».

Quelques remarques : tout d'abord, notez que vous ne faites pas que rejoindre un groupe. Vous joignez un groupe sur une interface réseau particulière. Bien sûr, il est possible de joindre le même groupe sur plusieurs interfaces. Si vous ne spécifiez pas concrètement une interface, alors le noyau en choisira une (selon sa table de routage) lorsqu'il sera nécessaire d'envoyer des datagrammes. Par ailleurs, il est possible que plus d'un processus rejoignent le même groupe multicast par une même interface. Ces processus reçoivent alors tous les datagrammes envoyés à ce groupe au travers de cette interface.

Comme dit précédemment, tous les hôtes gérant le multicast joignent le groupe *all-hosts* au démarrage, de ce fait on peut « pinger » tous les hôtes gérant le multicast au travers de l'adresse 224.0.0.1.

Finalement, considérez le fait suivant : pour qu'un processus puisse recevoir des datagrammes multicast, il doit demander au noyau de joindre le groupe désiré, et doit se relier au port par lequel les datagrammes seront envoyés. La couche UDP se base à la fois l'adresse de destination et le numéro de port pour démultipexer les paquets reçus et décider à quelle(s) connection(s) les délivrer.

2.4.2. Quitter un groupe multicast

Quand un processus n'est plus intéressé par un groupe multicast, il informe le noyau qu'il désire quitter ce groupe. Il est important de comprendre que cela ne signifie pas que le noyau n'acceptera plus les datagrammes multicast destinés à ce groupe. En effet, il continuera à le faire si d'autres processus restent intéressés par ce groupe. Dans ce cas l'hôte reste membre de ce groupe, et cela jusqu'à ce que tous les processus décident de le quitter.

L'explication ici est que joindre un groupe multicast nécessite seulement une adresse IP. C'est la couche de liaison de données qui, après une demande explicite au matériel dans certains cas, accepte les datagrammes multicast destinés à ce groupe. L'appartenance à un groupe ne se fait donc pas par processus, mais par hôte.

2.4.3. Transcrire une adresse IP Multicast dans une adresse Ethernet/FDDI

Les trames Ethernet, aussi bien que les trames FDDI, ont une adresse de destination codée sur 48 bits. Dans le but d'éviter une sorte d'ARP multicast translatant les adresses IP multicast en adresses ethernet/FDDI, l'IANA a réservé une plage d'adresses pour le multicast : chaque trame ethernet/FDDI dont l'adresse de destination appartient à l'intervalle allant de 01-00-5e-00-00-00 à 01-00-5e-ff-ff-ff contient des données à destination d'un groupe multicast. Le préfixe 01-00-5e identifie la trame comme étant multicast, le bit suivant étant toujours à 0, il reste donc 23 bits de disponibles pour les adresses multicast. Comme les groupes multicast sont définis sur 28 bits, la translation ne peut pas être une à une. Seuls les 23 bits les moins significatifs du groupe IP multicast sont placés dans la trame. Les 5 bits d'ordre supérieur restant sont ignorés, ce qui a pour résultat que 32 groupes multicast différents peuvent être translatés par une même adresse ethernet/FDDI. Cela signifie que la couche ethernet agit tel un filtre imparfait, et que la couche IP doit décider si elle accepte les datagrammes de la couche de liaison de données qui lui sont donnés. La couche IP agit tel un filtre final parfait.

Tous les détails concernant l'IP Multicast au travers de FDDI sont donnés dans le RFC 1390 « Transmission of IP and ARP over FDDI Networks » (transmission IP et ARP au travers de réseaux FDDI). Pour plus d'information concernant la translation d'adresses IP multicast vers ethernet, vous pouvez consulter le draft-ietf-mboned-intro-multicast-03.txt: « Introduction to IP Multicast Routing » (introduction au routage IP multicast).

Si vous êtes intéressé par l'IP multicast sur les réseaux locaux Token-Ring, consultez le RFC 1469.

3. Besoins et configuration du noyau

Linux est, bien sûr (doutez-vous de cela ?), pleinement compatible avec le multicast niveau 2. Il satisfait toutes les exigences liées à l'envoi, à la réception et au routage (grâce à mrouter) des datagrammes multicast.

Si vous désirez émettre et recevoir des paquets IP multicast, vous devez activer l'option « IP: multicasting » lors de la configuration de votre noyau. Si vous désirez aussi que votre Linux agisse tel un routeur multicast, vous aurez alors besoin d'activer le routage multicast du noyau en sélectionnant « IP: forwarding/gatewaying » (transfert/passerelle), « IP: multicast routing » (routage multicast) et « IP: tunneling » (tunnel), cette dernière option est nécessaire car les nouvelles versions de mrouted peuvent aussi retransmettre les paquets au travers de tunnels IP ; les datagrammes IP multicast sont alors encapsulés dans des datagrammes unicast. Il est nécessaire d'établir des tunnels entre des hôtes multicast s'ils sont séparés par un réseau ou par plusieurs routeurs incapables de transférer des datagrammes multicast. (*mrouted* est un service qui implémente un algorithme de routage multicast - politique de routage- et informe le noyau sur la façon de router les datagrammes multicast.

Certaines versions du noyau considèrent le routage multicast comme étant « EXPERIMENTAL », de ce fait vous devez activer l'option « Prompt for development and/or incomplete code/drivers » dans la section « Code maturity level options ».

Si, durant l'exécution de *mrouted*, le trafic généré sur le réseau (où est connectée votre station linux) est correctement transféré aux autres réseaux, mais il vous est impossible de « voir » le trafic des autres réseaux sur votre réseau local, vérifiez alors si vous recevez des messages d'erreur du protocole ICMP. Il s'agit d'une erreur due, le plus fréquemment, à la non activation de l'IP tunneling de votre routeur Linux. C'est le genre d'erreur qui paraît stupide lorsque vous la connaissez mais, croyez moi, cela prend beaucoup de temps pour la détecter quand vous l'ignorez, car il n'y a pas de raison apparente signifant la cause de l'erreur. Un renifleur (ou « sniffeur ») s'avère utile dans ce genre de situation.

Vous trouverez plus d'informations dans la section politiques de routage et techniques de retransmission ; *mrouted* et les tunnels sont aussi expliqués dans la section consacrée à MBone et la section applications multicast.

Une fois votre nouveau noyau compilé et installé, vous devez définir une route par défaut pour votre trafic multicast. Ce qui revient à ajouter une route au réseau 224.0.0.0.

Le problème auquel le plus de personnes font face à ce stade de la configuration est la valeur du masque à appliquer. Si vous avez lu précédemment l'excellent NET-3-HOWTO de Terry Dawson, il ne vous sera pas difficile de trouver la bonne valeur. Comme expliqué, le masque de sous réseau est un nombre codé sur 32 bits rempli avec des « 1 » sur la partie réseau de votre adresse IP, et avec des « 0 » sur la partie hôte. En référence à la section 2.1 une adresse multicast de classe D n'a pas de division réseau/hôte. À la place, il a un groupe d'identification codé sur 28 bits et un identifiant de classe D codé sur 4 bits. Le masque de sous réseau voulu est donc 11110000000000000000000000000000 ou d'une manière plus facilement lisible : 240.0.0.0. De ce fait la commande complète est :

```
route add 224.0.0.0 netmask 240.0.0.0 dev eth0
```

Selon la version de votre programme *route*, il peut être nécessaire d'ajouter l'option *-net* après l'option *add*.

Dans l'exemple, nous supposons que l'interface *eth0* supporte le multicast et, si rien d'autre n'est spécifié, nous voulons que le trafic multicast sorte par cette interface. Si cela n'est pas votre cas, changez alors le paramètre *dev* de manière appropriée.

Le système de fichiers */proc* s'avère utile une fois encore : il est possible de vérifier dans le fichier */proc/net/igmp* les groupes auxquels votre machine est actuellement connectée.

4. MBone

L'utilisation d'une nouvelle technologie apporte son lot d'avantages et d'inconvénients. Les avantages du multicast sont -je le pense- clairs. Le principal inconvénient est qu'actuellement des centaines d'hôtes, et plus spécialement les routeurs, ne supportent pas le multicast. Par conséquent, les personnes qui commencent à travailler avec du multicast, et qui de fait ont acheté de nouveaux équipements et modifié leurs systèmes d'exploitation ; on construit des *îlots multicast* dans leurs locaux. C'est alors qu'ils ont découvert qu'il leur était difficile de communiquer avec des personnes faisant de même car il suffit d'un routeur ne supportant pas le multicast pour que rien ne fonctionne.

La solution est alors apparue comme évidente : ils décidèrent de contruire un réseau multicast virtuel au-dessus d'Internet. De fait : les sites reliés par des routeurs multicasts peuvent communiquer entre-eux directement. Mais les sites joigniables uniquement par le biais de routeurs unicast doivent encapsuler leur trafic multicast dans des paquets unicast pour communiquer avec les autres îlots multicast. Les routeurs traversés ne poserons pas de problèmes car ils savent gérer ce trafic unicast. Finalement, le site de réception désencapsulera le trafic, et le retransmettra à l'îlot selon la méthode multicast originale. Ainsi, les deux extrémités convertissent du multicast vers de l'unicast et à nouveau de l'unicast vers du multicast, définissant ainsi ce qui est appelé un tunnel multicast.

Le MBone ou épine dorsale multicast (Multicast BackBone) est un réseau virtuel multicast basé sur l'interconnection d'îlots multicast reliés entre eux par des tunnels multicast.

Diverses utilisations sont faites du MBone de nos jours, on peut ainsi remarquer la profusion de téléconférences audio/vidéo en temps réel qui ont lieu au travers d'Internet. Un exemple de ces téléconférences est la retransmission (en direct !) de la conférence de Linus Torvalds qui a été donnée au groupe d'utilisateurs de Linux de la Silicon Valley [ndt : en 1999 ?].

Pour plus d'informations à propos du MBone, consultez : http://www.mediadesign.co.at/newmedia/more/mbone-faq.html

5. Applications multicast

La plupart des personnes ayant affaire avec le multicast, décident tôt ou tard de se connecter au MBone, et de ce fait ont tradionnellement besoin de *mrouted*. Vous aurez sûrement besoin de lui si vous ne possédez pas de routeur capable de faire du multicast et que vous voulez que le travail effectué sur l'un de vos sous réseaux soit « entendu » par un autre réseau. *mrouted* circonscrit le problème d'envoi de trafic au travers des routeurs unicast -il encapsule les datagrammes multicast dans des datagrammes unicast (IP dans IP)-. Il ne s'agit pas de la seule fonctionnalité de *mrouted*. La plus importante, est qu'il indique au noyau la façon de router (ou de ne pas router) les datagrammes multicast selon leurs adresses d'origine et de destination. De ce fait, même si vous avez un routeur multicast, *mrouted* peut être utilisé pour lui dire *quoi faire* des datagrammes. (notez que j'ai dit *quoi*, et non *comment*, *mrouted* dit « transmet ceci au réseau connecté à cette interface », mais le transfert se fait actuellement par le noyau). Il existe une distinction à faire entre le transfert en lui-même et l'algorithme qui décide de ce qui doit être transféré et de la façon de le faire. Cette distinction est très utile car elle permet d'écrire un code unique permettant le transfert, en l'insérant dans le noyau ; tandis que les algorithmes et les politiques de transfert sont implémentés en tant que service utilisateur, ce qui permet un changement de politique sans recompiler le noyau.

Vous pouvez vous procurer une version de *mrouted* pour Linux depuis : ftp://www.video.ja.net/mice/mrouted/Linux/. Des mirroirs de ce site existent tout autour de la planète. Lisez bien le fichier ftp://www.video.ja.net/mice/README.mirrors pour choisir le mirroir le plus près de chez vous.

Vous trouverez ci-dessous une liste des applications qui ont été écrites afin de se connecter au MBone et qui ont été portées sous Linux. Cette liste est issue de la page d'information « Linux Multicast Information » écrite par Michael Esler : http://www.cs.virginia.edu/~mke2e/multicast/.Je vous recommande cette page pour toutes les informations et ressources à propos du multicast et de Linux.

5.1. Conférences audio

- NeVoT - Terminal réseau pour la voix http://www.fokus.gmd.de/step/nevot
- RAT - UCL outil audio robuste http://www-mice.cs.ucl.ac.uk/mice/rat
- vat - LBL outil audio visuel http://www-nrg.ee.lbl.gov/vat/

5.2. Conférences vidéo

- ivs - Système de vidéo conférence de l'Inria http://www.inria.fr/rodeo/ivs.html
- nv - Outil réseau pour la vidéo ftp://ftp.parc.xerox.com/pub/net-research/
- nv w/ Meteor - Version de nv w/ pour Matrox Meteor (UVa) ftp://ftp.cs.virginia.edu/pub/gwtts/Linux/nv-meteor.tar.gz
- vic - LBL Outil de vidéo conférence http://www-nrg.ee.lbl.gov/vic/
- vic w/ Meteor - Version de vic w/ support pour Matrox Meteor (UVa) ftp://ftp.cs.virginia.edu/pub/gwtts/Linux/vic2.7a38-meteor.tar.gz

5.3. Autres utilitaires

- mmphone service téléphonique multimédia http://www.eit.com/software/mmphone/phoneform.html
- wb - LBL tableau blanc partagé http://www-nrg.ee.lbl.gov/wb/
- webcast - Application multicast fiable pouvant relier les navigateurs Mosaic http://www.ncsa.uiuc.edu/SDG/Software/XMosaic/CCI/webcast.html

5.4. Outils de session

Les outils de gestion de sessions ont été placés après les autres, car ils nécessitent quelques explications. Lorsque qu'une conférence a lieu, il est assigné pour chaque service (audio, vidéo, tableau-blanc partagé, etc) un ou plusieurs groupes multicast, eux-mêms associés à un numéro de port. Les conférences qui vont avoir lieu sont périodiquement annoncées sur le MBone. Ces annonces contiennent des informations à propos des groupes multicasts, des numéros de ports et des programmes qui peuvent être utilisés (vic, vat, …). Les outils de gestion de sessions « écoutent » ces informations et vous les annoncent de manière à ce qu'elles soient facilement joignables. Vous pouvez ainsi décider quelle conférence vous intéresse. Ainsi, au lieu de démarrer chaque programme qui sera utilisé et lui indiquer les groupes multicast et les numéros de ports qu'il vous faut joindre, il vous suffit de cliquer sur l'annonce et l'outil de gestion de sessions démarre les bonnes applications, en leur communiquant toutes les informations nécessaires pour joindre la conférence. Les outils de gestion de sessions vous permettent généralement d'annoncer votre propre conférence sur le MBone.

- gwTTS - système de télé-enseignement de Université de Virginie http://www.cs.Virginia.EDU/~gwtts
- isc - contrôleur de session intégré http://www.fokus.gmd.de/step/isc
- mmcc - contrôle de conférences multimédia ftp://ftp.isi.edu/confctrl/mmcc
- sd - outil de référencement de session LBL ftp://ftp.ee.lbl.gov/conferencing/sd
- sd-snoop - utilitaire de référencement de sessions du Tenet Group ftp://tenet.berkeley.edu/pub/software
- sdr - nouvelle génération de référencement de sessions d'UCL ftp://cs.ucl.ac.uk/mice/sdr

6. Programmation multicast

Programmation multicast… ou comment écrire vos propres applications multicast.

Diverses extensions de l'API de programmation sont nécessaires pour utiliser le multicast. Ces extensions sont utilisables par le biais de deux appels systèmes : setsockopt() (utilisé pour envoyer des informations au noyau) et getsockopt() (utilisé pour recevoir des informations du voisinage multicast). Cela ne signifie pas que ces 2 nouveaux appels systèmes ont été ajoutés pour le fonctionnement du multicast. La paire *setsockopt() / getsockopt()* est là depuis des années. Depuis la BSD 4.2 au moins. L'ajout consiste en un nouveau jeu d'options (options multicast) qui est passé à ces appels systèmes et que le noyau doit comprendre.

Les deux lignes suivantes donnent le prototypage des fonctions *setsockopt() / getsockopt()*.

```
int getsockopt(int s, int level, int optname, void* optval, int* optlen);
int setsockopt(int s, int level, int optname, const void* optval, int optlen);
```

Le premier paramètre, *s*, est la prise réseau (socket) à laquelle l'appel système s'applique. Pour le multicast, la prise doit être de la famille *AF_INET* et peut être de type *SOCK_DGRAM* ou *SOCK_RAW*. Le plus courant est l'utilisation d'une prise *SOCK_DGRAM*, mais si votre but est d'écrire un démon de routage ou d'en modifier un, vous aurez plutôt besoin d'une prise de type *SOCK_RAW*.

Le deuxième paramètre, *level*, identifie la couche qui capturera l'option, le message, ou la requète, ou tout ce que vous désirez appeler. De fait, *SOL_SOCKET* est pour la couche de la prise réseau, *IPPROTO_IP* est pour la couche IP, etc. Pour la programmation multicast, la valeur est toujours *IPPROTO_IP*.

optname identifie l'option que nous passons ou demandons. Sa valeur, soit fournie par le programme, soit retournée par le noyau, est *optval*. Les valeurs pour *optname* qui peuvent être invoquées pour la programmation multicast sont les suivantes :

	setsockopt()	getsockopt()
IP_MULTICAST_LOOP	oui	oui
IP_MULTICAST_TTL	oui	oui
IP_MULTICAST_IF	oui	oui
IP_ADD_MEMBERSHIP	oui	non
IP_DROP_MEMBERSHIP	oui	non

optlen transporte la taille de la structure de données à laquelle *optval* fait référence. Notez que pour *getsockopt()*, c'est un résultat et non une donnée : le noyau écrit la valeur de *optname* dans la zone tampon pointée par *optval* et nous informe de la longueur de la valeur par *optlen*.

Aussi bien *setsockopt()* que *getsockopt()* retournent 0 en cas de succès et -1 en cas d'erreur.

6.1. IP_MULTICAST_LOOP

Vous devez décider, en tant que programmeur, si les données que vous voulez émettre doivent être retournées ou non sur votre hôte. Si vous pensez avoir plus d'un processus ou utilisateur en écoute, alors un retour doit être activé. À contrario, si vous émettez des images que votre caméra vidéo produit, vous ne nécessiterez probablement pas de retour, à moins que vous désiriez les voir sur votre écran. Dans ce dernier cas, votre application recevra déjà certainement les images depuis le périphérique connecté à votre ordinateur pour les envoyer à la prise réseau. De ce fait, l'application possède déjà les données, et cela devient improbable que vous vouliez les recevoir de nouveau au travers de la prise réseau.

Il est à noter que le retour est activé par défaut.

Comme *optval* est un pointeur, vous ne pouvez pas écrire :

```
setsockopt(socket, IPPROTO_IP, IP_MULTICAST_LOOP, 0, 1);
```

pour désactiver le loopback vous devez écrire :

```
u_char loop;
setsockopt(socket, IPPROTO_IP, IP_MULTICAST_LOOP, &loop, sizeof(loop));
```

et donnez la valeur 1 à *loop* pour activer le retour et 0 pour le désactiver.

Pour savoir si une prise réseau a activé ou non son retour, utilisez quelque chose comme :

```
u_char loop;
int size;
getsockopt(socket, IPPROTO_IP, IP_MULTICAST_LOOP, &loop, &size)
```

6.2. IP_MULTICAST_TTL

Si rien n'est précisé, les datagrammes multicast sont envoyés avec comme valeur par défaut 1, dans le but d'éviter qu'ils soient transférés sur tout le réseau local. Pour changer la valeur du *TTL* (de 0 à 255), mettez cette valeur dans une variable (ici nommée « ttl ») et écrivez dans votre programme :

```
u_char ttl;
setsockopt(socket, IPPROTO_IP, IP_MULTICAST_TTL, &ttl, sizeof(ttl));
```

Le comportement avec *getsockopt()* est similaire à celui vu avec *IP_MULTICAST_LOOP*.

6.3. IP_MULTICAST_IF

Traditionnellement, l'administrateur système indique l'interface multicast par défaut sur laquelle les datagrammes sont envoyés. Le programmeur peut surcharger cela et choisir ainsi à l'aide de cette option une interface de sortie concrète pour une prise réseau donnée.

```
struct in_addr interface_addr;
setsockopt (socket, IPPROTO_IP, IP_MULTICAST_IF, &interface_addr,
            sizeof(interface_addr));
```

Ainsi, tout le trafic multicast généré par cette prise réseau sera envoyé sur l'interface choisie. Pour revenir au comportement par défaut et laisser le noyau choisir l'interface de sortie (choix basé sur la configuration de l'administrateur système), il suffit d'appeler *setsockopt()* avec la même option et *INADDR_ANY* comme valeur pour l'interface.

Pour connaitre ou sélectionner une interface de sortie, les *ioctls* suivants peuvent être utililisés : *SIOCGIFADDR* (pour obtenir les adresses des interfaces), *SIOCGIFCONF* (pour obtenir la liste de toutes les interfaces) et *SIOCGIFFLAGS* (pour obtenir les fonctionnalités d'une interface et, de fait, permet de déterminer si l'interface supporte le multicast ou non. Fonction indiquée par l'option *IFF_MULTICAST*).

Si un hôte a plus d'une interface et que l'option *IP_MULTICAST_IF* n'est pas mentionnée, alors les transmissions multicast sont émises sur l'interface par défaut, bien que les interfaces restantes puissent être utilisées pour des retransmissions multicast si l'hôte joue le rôle de routeur multicast.

6.4. IP_ADD_MEMBERSHIP

Rappelez vous que vous devez informer le noyau des groupes multicast qui vous intéressent. Si aucun processus n'est intéressé par un groupe donné, les paquets provenant de ce groupe et qui arrivent sur notre hôte sont rejetés. Pour informer le noyau, des groupes qui vous intéressent, et de fait, pour devenir membre de ce groupe, vous devez d'abord remplir une structure *ip_mreq* qui sera passée plus tard au noyau dans le champ *optval* avec l'appel système *setsockopt()*.

La structure *ip_mreq* (provenant de /usr/include/linux/in.h) a les membres suivants :

```
struct ip_mreq
{
  struct in_addr imr_multiaddr;   /* adresse IP du groupe multicast */
  struct in_addr imr_interface;   /* adresse IP de l'interface locale */
};
```

Note : la définition « physique » de la structure est contenue dans le fichier spécifié ci-dessus. Vous ne devez en aucun cas inclure <linux/in.h> si vous souhaitez que votre code soit portable. En lieu et place, incluez <netinet/in.h>, qui inclut à son tour <linux/in.h>.

Le premier membre, *imr_multiaddr*, contient l'adresse du groupe que vous désirez joindre. Souvenez-vous que l'appartenance à un groupe se fait aussi par le biais d'une interface, pas uniquement sur l'adresse du groupe. C'est la raison pour laquelle vous devez fournir une valeur au second membre : *imr_interface*. De cette façon, si vous êtes sur un hôte ayant plusieurs interfaces multicast, vous pouvez joindre un même groupe par le biais de différentes interfaces. Si vous ne désirez pas indiquer d'interface, vous pouvez toujours remplir ce dernier membre avec un joker (*INADDR_ANY*), alors le noyau choisira de lui même l'interface.

Une fois la structure remplie (définie en tant que *struct ip_mreq mreq;*) vous avez juste à appeler *setsockopt()* de cette manière :

```
setsockopt (socket, IPPROTO_IP, IP_ADD_MEMBERSHIP, &mreq, sizeof(mreq));
```

Notez que vous pouvez joindre plusieurs groupes à travers la même prise réseau. Cette limite supérieure est fixée par *IP_MAX_MEMBERSHIPS* et, a pour la version 2.0.33 du noyau Linux, la valeur de 20.

6.5. IP_DROP_MEMBERSHIP

La procédure nécessaire pour quitter un groupe est semblable à celle nécessaire pour le joindre.

```
struct ip_mreq mreq;
setsockopt (socket, IPPROTO_IP, IP_DROP_MEMBERSHIP, &mreq, sizeof(mreq));
```

Où *mreq* est une structure contenant les mêmes données que celles utilisées lorsque ce groupe fût joint. Si le membre *imr_interface* contient la valeur *INADDR_ANY*, alors le premier groupe pouvant correspondre est supprimé.

Si vous avez rejoint un grand nombre de groupes sur une même prise réseau, il n'est pas nécessaire de supprimer toutes les souscriptions pour terminer. Lorsque vous fermez une prise réseau, toutes les souscriptions associées à cette prise sont supprimées par le noyau. Il en est de même si le processus qui a ouvert la prise est tué.

Cependant, gardez à l'esprit que si un processus abandonne la souscription à un groupe, cela n'implique pas forcément l'arrêt de la transmission des datagrammes du groupe en question à l'hôte. En effet, si une autre prise a joint ce groupe avec la même interface avant le *IP_DROP_MEMBERSHIP*, alors l'hôte continuera à être membre de ce groupe.

ADD_MEMBERSHIP (aussi bien que *DROP_MEMBERSHIP*) est une opération non-bloquante. Il renvoie un résultat, immédiatement, indiquant s'il a réussi ou échoué.

7. Fonctionnement interne

Ce chapitre délivre des informations, qui ne sont ni nécessaires à la compréhension du fonctionnement du multicast, ni nécessaires à l'écriture de programmes multicast, mais qui deumeurent cependant très intéressantes car elles permettent une compréhension des concepts sur lesquels s'appuient le multicast et ses implémentations. Cela permet ainsi de contourner les erreurs les plus fréquentes et de nombreuses incompréhensions.

7.1. IGMP

Lorsque nous avons parlé de *IP_ADD_MEMBERSHIP* et de *IP_DROP_MEMBERSHIP*, nous avions dit que l'information délivrée était utilisée par le noyau pour accepter ou refuser les datagrammes multicast. Cela est vrai mais pas complêtement. Une telle simplification impliquerait que les datagrammes multicast de tous les groupes multicast à travers le monde seraient reçus par notre hôte, et alors il vérifierait les souscriptions des processus s'exécutant sur lui et déciderait s'il délivre l'information ou non. Comme vous pouvez l'imaginer, cela serait du gaspillage de bande passante.

De fait actuellement, un hôte informe son routeur des groupes multicasts qui l'intéressent ; alors, ce routeur informe à son tour les routeurs en amont qu'il souhaite recevoir le trafic en question, et ainsi de suite. Les algorithmes employés servant à demander le trafic d'un groupe donné, ou à l'inverse ceux pour terminer une souscription, peuvent varier. Cependant, il y a quelque chose qui ne change jamais : la manière dont cette information est transmisse. IGMP est utilisé pour cela. IGMP signifie Internet Group Management Protocol. C'est un protocole similaire par de nombreux aspects à ICMP. Il porte un numéro de protocole égal à 2 et ses messages sont transportés dans des datagrammes IP. Tous les hôtes compatibles multicast niveau 2 doivent implémenter ce protocole.

Comme dit précédemment, IGMP est utilisé aussi bien par les hôtes pour donner des informations de souscription à ses routeurs, que par les routeurs pour communiquer entre eux. Par la suite, il ne sera question que des relations entre les hôtes et les routeurs, principalement parce que les informations décrivant les communications entre routeurs (autres que celles provenant du source code de mrouted, RFC-1075) et décrivant le protocole de routage multicast « distance-vecteur » sont maintenant obsolètes. mrouted implémente quant à lui une version de ce protocole non encore documentée.

La version 0 d'IGMP est spécifiée dans le RFC-988 qui est maintenant obsolète. Plus personne n'utilise la version 0 de nos jours.

La version 1 d'IGMP est décrite dans le RFC-1112 et, bien qu'elle soit mise-à-jour par le RFC-2236 (IGMP version 2), elle est encore largement utilisée.

Le noyau Linux implémente complètement IGMP version 1 et en partie la version 2.

Vous trouverez ci-dessous une description informelle du protocole. Vous pouvez consulter le RFC-2236 pour une description formelle, avec beaucoup de diagrammes d'états et d'expiration de délais.

Tous les messages IGMP ont la structure suivante :

<table>
<tr><td>0</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>1</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>2</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>3</td><td></td></tr>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>0</td><td>1</td></tr>
<tr><td colspan="8">Type</td><td colspan="8">Temp max. de réponse</td><td colspan="16">contrôle d'intégrité</td></tr>
<tr><td colspan="32">Adresse de groupe</td></tr>
</table>

Dans le cas d'IGMP version 1 (appelé ci-après IGMPv1) le champ « temp maximum de réponse » est marqué comme « inutilisé ». Il est rempli de zéros lors d'une émission et est ignoré à la réception. De plus, cette version d'IGMP coupe le champ « type » en deux sous champs de longueur 4 bits : « version » et « type ». Comme IGMPv1 identifie un message d'*interrogation de souscription* comme étant 0x11 (version 1, type 1) et IGMPv2 aussi, les 8 bits ont la même signification.

Je pense qu'il est plus instructif de donner d'abord une description d'IGMPv1 et de pointer ensuite les additions faites par IGMPv2.

À la lecture de la description faite ci-dessous, gardez à l'esprit qu'un routeur multicast reçoit *tous* les datagrammes IP multicast.

7.1.1. IGMP version 1

Les routeurs émettent périodiquement des requêtes IGMP sur les souscriptions des hôtes (IGMP Host Membership Queries). Ces requêtes sont faites sur le groupe *all-hosts* (224.0.0.1) avec un TTL de 1, ceci toutes les 1 ou 2 minutes. Tous les hôtes supportant le multicast entendent cela, mais ne répondent pas immédiatement pour éviter une tempête de réponses. Au lieu de cela, ils déclenchent un compte à rebours pour chaque groupe auquel ils appartiennent, et cela sur l'interface sur laquelle ils ont reçu la demande.

Tôt ou tard, le compte à rebours expirera sur un des hôtes, ce dernier envoiera alors un *IGMP Host Membership Report* (résultat d'appartenance) avec un TTL de 1 à l'adresse multicast du groupe devant être rapporté. Comme le résultat est envoyé au groupe, tous les hôtes qui ont rejoint ce dernier - et qui sont actuellement en train d'attendre que leur propre délai expire- le recoive aussi. Alors, ces hôtes stoppent leur compte à rebours, et ne génèrent pas d'autres rapports. Un seul rapport est généré - par l'hôte ayant le délai le plus court-, et cela est suffisant pour le routeur. Celui-ci doit seulement savoir s'il y a des membres de ce groupe dans le sous réseau, pas qui et combien ils sont.

Lorsqu'aucune réponse n'est reçue pour un groupe donné après un certain nombre de requêtes, le routeur déduit qu'il n'y a pas de membre de ce groupe sur ce sous réseau, et de ce fait n'a pas à transmettre le trafic. Notez que pour IGMPv1 il n'y a pas de message de fin d'appartenance de groupe.

Lorsqu'un hôte rejoint un *nouveau* groupe, le noyau envoie un rapport concernant ce groupe, ainsi il n'est pas nécessaire d'attendre une ou deux minutes jusqu'a ce que la demande d'appartenance suivante soit formulée. Comme vous pouvez le constater le paquet IGMP est généré par le noyau comme étant une réponse à la commande *IP_ADD_MEMBERSHIP*, comme vu dans la section IP_ADD_MEMBERSHIP. Notez l'importance de l'adjectif *nouveau* : si la commande *IP_ADD_MEMBERSHIP* est effectuée sur un hôte déjà membre de ce groupe, aucun paquet IGMP n'est envoyé car nous recevons déjà le trafic pour ce groupe ; à la place, un compteur sur le nombre d'utilisateurs de ce groupe est incrémenté. *IP_DROP_MEMBERSHIP* ne génère aucun datagramme dans IGMPv1.

Les requêtes sur les souscriptions des hôtes sont identifiées par le type 0x11, et les rapports d'appartenance par le type 0x12.

Aucun rapport n'est envoyé pour le groupe *all-hosts*. L'appartenance à ce dernier groupe est permanente.

7.1.2. IGMP version 2

L'ajout majeur dans cette version du protocole est l'inclusion de messages de fin d'appartenance de groupe (*Leave Group message*) noté 0x17. La raison de cet ajout est d'éviter un gaspillage de bande passante entre le moment où le dernier hôte du sous-réseau quitte ses souscriptions et le moment où le routeur s'aperçoit qu'il n'y a plus de membre de ce groupe présent dans ce sous réseau (*leave latency*). Les messages de fin d'appartenance doivent être adressés au groupe *all-routers* (224.0.0.2) plus qu'au groupe que l'on vient de quitter, car l'information n'a pas d'utilité pour les autres membres de ce groupe (les versions du noyau ultérieures à 2.0.33 envoyent l'information au groupe ; et bien que cela ne porte pas préjudice aux hôtes, cela constitue une perte de temps car ces messages doivent êtres traités, sans pour autant donner d'information utile). Il est à noter qu'il existe certains détails subtiles à propos du moment où il est approprié ou non d'envoyer des Messages de fin d'appartenance ; si cela vous intéresse, reportez-vous au RFC.

Quant un router IGMPv2 reçoit un message de fin d'appartenance de groupe, il envoie une requête « spécifique de groupe » (*Group-Specific Queries*) au groupe à quitter. Cela contitue un autre ajout. IGMPv1 n'a pas ces requêtes spécifiques de groupes. Toutes ces requêtes sont envoyées au groupe *all-hosts*. Le champ *type* de l'entête IGMP n'est pas amené à changer (0x11 comme précédemment), mais le champ « adresse du groupe » est rempli avec l'adresse du groupe multicast à quitter.

Le champ *Temps Maximum de Réponse*, qui dans IGMPv1 était mis à 0 pour la transmission et ignoré à la réception, devient significatif seulement dans les messages de requêtes en appartenance (ou Membership Query). Ce champ informe du temps maximum alloué en 1/10 de secondes avant l'envoi d'un rapport. Il est donc utilisé comme mécanisme d'écoute.

IGMPv2 introduit un nouveau type de message : 0x16. Il s'agit d'un « rapport d'appartence IGMPv2 » envoyé par les hôtes IGMPv2 si ceux-ci détectent qu'un routeur IGMPv2 est présent (un hôte IGMPv2 sait qu'un routeur IGMPv1 est présent s'il reçoit une requête avec le champ « Réponse Max » à 0).

Lorsque plus d'un routeur se réclame comme agissant en interrogateur, IGMPv2 fournit un mécanisme pour écourter les discussions : le routeur possédant la plus petite adresse IP est désigné comme étant l'interrogateur. Les autres routeurs restent à l'écoute d'éventuels dépassements de temps de réponse. Ainsi, si le routeur ayant la plus petite adresse IP « tombe » ou est éteint, la décision pour connaitre le nouvel interrogateur est prise après que les délais aient expiré.

7.2. Aspects concernant le noyau

Cette sous-section donne le point de départ de l'étude de l'implémentation du multicast dans le noyau Linux. Cela n'explique pas l'implémentation faite. Elle indique juste où trouver les éléments.

L'étude a été menée sur la version 2.0.32, de ce fait il se peut qu'elle soit dépassée au moment où vous allez la lire (le code réseau du noyau semble avoir beaucoup changé dans les versions 2.1.x).

Le code multicast dans les noyaux Linux est toujours entouré par une paire de balises #ifdef CONFIG_IP_MULTICAST / #endif, de cette façon vous pouvez l'inclure ou l'exclure de votre noyau selon vos besoins (cette inclusion/exclusion est faite à l'étape de compilation, comme vous le savez probablement… Les #ifdefs sont interprétés par le pré-processeur. La décision est prise selon les choix que vous avez effectués lors du make config, make menuconfig ou make xconfig).

Vous désirez probablement activer les fonctionnalités multicast, mais si votre machine Linux n'a pas à se comporter comme un routeur multicast, vous n'aurez sûrement pas besoin d'activer le routage multicast du noyau. Le code concernant le routage multicast est contenu entre la paire de balises #ifdef CONFIG_IP_MROUTE / #endif.

Les sources du noyau sont couramment placées dans le répertoire /usr/src/linux. Cependant, l'emplacement peut changer, de ce fait l'emplacement du répertoire de base des sources du noyau sera designé ici comme étant LINUX. De ce fait, LINUX/net/ipv4/udp.c correspond à /usr/src/linux/net/ipv4/udp.c si vous avez extrait les sources du noyau dans le répertoire /usr/src/linux.

Toutes les interfaces multicast désignées dans cette section et dédiées à la programmation d'applications multicast, s'utilisent au travers des appels systèmes *setsockopt() / getsockopt()*.

L'un comme l'autre est implémenté au moyen de fonctions qui effectuent quelques tests pour vérifier les paramètres qui leurs sont passés et qui, à tour de rôle, appellent une autre fonction qui effectue quelques tests complémentaires, démultiplexant l'appel passé dans le paramètre *level* pour chaque appel système, et appelle alors une autre fonction qui… (si vous êtes intéressé par tous ces sauts, vous pouvez les suivre dans LINUX/net/socket.c (fonctions *sys_socketcall()* et *sys_setsockopt()*), LINUX/net/ipv4/af_inet.c (fonction *inet_setsockopt()*) et LINUX/net/ipv4/ip_sockglue.c (fonction *ip_setsockopt()*)).

Le fichier qui nous intéresse ici est LINUX/net/ipv4/ip_sockglue.c. Nous y trouvons *ip_setsockopt()* et *ip_getsockopt()* qui sont essentiellement des switchs (après quelques vérifications d'erreurs) vérifiant chaque valeur possible pour *optname*. Tout comme les options unicast, les options multicast sont capturées : *IP_MULTICAST_TTL*, *IP_MULTICAST_LOOP*, *IP_MULTICAST_IF*, *IP_ADD_MEMBERSHIP* et *IP_DROP_MEMBERSHIP*. Avant le switch, un test est fait pour savoir si les options sont spécifiques au routeur multicast, et dans ce cas, elles sont redirigées vers les fonctions *ip_mroute_setsockopt()* et *ip_mroute_getsockopt()* (fichier LINUX/net/ipv4/ipmr.c).

Dans le fichier LINUX/net/ipv4/af_inet.c nous pouvons voir les valeurs par défaut -dont nous parlions dans les sections précédentes (loopback activé, TTL=1)- données à la prise réseau lorsqu'elle est crée (prises depuis la fonction *inet_create()* de ce fichier) :

```
#ifdef CONFIG_IP_MULTICAST
  sk->ip_mc_loop=1;
  sk->ip_mc_ttl=1;
 *sk->ip_mc_name=0;
  sk->ip_mc_list=NULL;
#endif
```

L'assertion stipulant que « la fermeture d'une prise réseau implique que le noyau n'accepte plus les souscriptions faites par cette prise » est corroborée par :

```
#ifdef CONFIG_IP_MULTICAST
/* Les applications oublient de quitter les groupes avant de partir */
  ip_mc_drop_socket(sk);
#endif
```

Extrait de la fonction *inet_release()*, dans le même fichier que précédemment.

Les opérations indépendantes de la couche de liaison et relatives aux périphériques sont gardées dans le fichier LINUX/net/core/dev_mcast.c.

Deux fonctions sont encore manquantes : les fonctions concernant le traîtement des paquets multicast en entrée et en sortie. Comme tous les autres paquets, les paquets entrants sont transmis depuis les pilotes de périphériques à la fonction *ip_rcv()* (LINUX/net/ipv4/ip_input.c). Cette fonction contient le filtrage parfait appliqué aux paquets multicasts qui ont passé la couche du périphérique (souvenez-vous que les couches les plus basses ne fournissent qu'un filtrage approximatif et ce n'est que la couche IP qui sait à 100% si nous sommes intéressés ou non par ce groupe multicast). Si l'hôte fonctionne comme un routeur multicast, alors la fonction décide aussi si le paquet doit être transmis. Il appelle alors *ipmr_forward()*. Cette dernière fonction est implémentée dans le fichier LINUX/net/ipv4/ipmr.c.

Le code se chargeant des paquets émis en sortie est contenu dans le fichier LINUX/net/ipv4/ip_output.c. On y trouve l'effet possible de l'option des *IP_MULTICAST_LOOP*, selon que l'on souhaite ou non faire une boucle locale (fonction *ip_queue_xmit()*). Ainsi la valeur du *TTL* du paquet sortant est donné selon qu'il s'agit d'un paquet unicast ou multicast. Dans ce dernier cas, la valeur de l'argument *IP_MULTICAST_TTL* passé en option est utilisée (fonction *ip_build_xmit()*).

Durant un travail à l'aide de mrouted (un programme fournissant des informations liées au noyau sur la façon de router des datagrammes multicast), nous avons détecté que les paquets multicast provenant du réseau local sont routés correctement exceptés ceux venant de la machine Linux agissant en tant que routeur multicast ! ip_input.c fonctionne bien, mais il semble que ce n'est pas le cas de ip_output.c. En lisant le code source des fonctions de sorties, nous avons trouvé que les paquets sortant n'étaient pas passés à *ipmr_forward()*, fonction décidant si les paquets doivent être routés ou non. Les paquets sont sortis sur le réseau local mais, comme les cartes réseaux sont le plus souvent incapables de lire les données transmises, ces datagrammes ne sont alors jamais routés. Nous avons ajouté le code nécessaire à la fonction ip_build_xmit() et le tour était joué. (Disposer du code source du noyau n'est pas un luxe, mais une nécessité !)

ipmr_forward() a déjà été mentionné un certain nombre de fois. C'est une fonction importante car elle permet de résoudre un important problème de compréhension nécessitant d'être plus largement expliqué. En routant du trafic multicast, ce n'est pas mrouted qui fabrique les copies puis les expédie à ses propres destinataires. mrouted reçoit tout le trafic multicast et, basé sur cette information, calcule les tables de routages multicast puis informe le noyau de la manière de router : « les datagrammes pour ce groupe provenant de cette interface doivent être transférés à ces interfaces ». Cette information est transmise au noyau par l'appel de *setsockopt()* sur la file d'une prise réseau ouverte par le daemon mrouted (le protocole spécifié lors de la création de la file de la prise réseau doit être *IPPROTO_IGMP*). La prise en compte d'options s'effectue par l'appel à la fonction *ip_mroute_setsockopt()* de LINUX/net/ipv4/ipmr.c. Cette première option (il est préférable de les appeler commandes plutôt qu'options) provenant de cette prise réseau doit être *MRT_INIT*. Toutes les autres commandes sont ignorées (retournant *-EACCES*) si *MRT_INIT* n'est pas précisé en premier. Seulement une seule instance de mrouted peut être exécutée à la fois sur un même hôte. Pour garder une trace de cela, lorsque le premier *MRT_INIT* est reçu, une variable importante, *struct sock* mroute_socket*, est pointée sur la prise réseau ou le *MRT_INIT* a été reçu. Si *mroute_socket* n'est pas nul lors de l'attente d'un *MRT_INIT* cela signifie qu'un autre mrouted est en cours d'exécution, un *-EADDRINUSE* est alors renvoyé. Toutes les commandes s'appuyant sur le même principe (*MRT_DONE*, *MRT_ADD_VIF*, *MRT_DEL_VIF*, *MRT_ADD_MFC*, *MRT_DEL_MFC* et *MRT_ASSERT*) retournent *-EACCES* si elles proviennent d'une prise réseau différente de *mroute_socket*.

Comme les datagrammes multicast routés peuvent être reçus/envoyés au travers d'interfaces physiques ou de tunnels, une abstraction commune a été définie : les VIFs, InterFaces Virtuelles (ou Virtual InterFaces). mrouted communique les structures VIFs au noyau, en indiquant les interfaces physiques ou tunnels pour qu'il les ajoute à sa table de routage. Les entrées de retransmissions multicast indiquent où transmettre les datagrammes.

Les VIFs sont ajoutées à l'aide de *MRT_ADD_VIF* et supprimées avec *MRT_DEL_VIF*. L'un comme l'autre passent un struct vifctl au noyau (défini dans /usr/include/linux/mroute.h) avec les informations suivantes :

```
struct vifctl {
  vifi_t  vifc_vifi;            /* Index du VIF */
  unsigned char vifc_flags;      /* Attributs VIFF_ */
  unsigned char vifc_threshold;  /* Seuil du ttl */
  unsigned int vifc_rate_limit;  /* Valeur de débit limite : non implémenté */
  struct in_addr vifc_lcl_addr;  /* Notre adresse */
```

```
  struct in_addr vifc_rmt_addr;  /* Adresse du tunnel IPIP */
};
```

Avec cette information une structure *vif_device* est construite :

```
struct vif_device
{
  struct device   *dev;                   /* le périphérique que nous utilisons */
  struct route    *rt_cache;              /* Tunnel route cache */
  unsigned long   bytes_in,bytes_out;
  unsigned long   pkt_in,pkt_out;         /* Statistiques */
  unsigned long   rate_limit;             /* Forme du trafic ; non implémenté */
  unsigned char   threshold;              /* seuil du TTL */
  unsigned short  flags;                  /* attributs de contrôle */
  unsigned long   local,remote;           /* Adresses(distantes pour les tunnels)*
};
```

Notez l'entrée *dev* dans la structure. La structure *device* est définie dans le fichier /usr/include/linux/netdevice.h. C'est une grosse structure, mais le champ qui nous intéresse est :

```
struct ip_mc_list*    ip_mc_list;   /* chaîne des filtres IP multicast */
```

La structure *ip_mc_list* -définie dans /usr/include/linux/igmp.h- est comme suit :

```
struct ip_mc_list
{
  struct device *interface;
  unsigned long multiaddr;
  struct ip_mc_list *next;
  struct timer_list timer;
  short tm_running;
  short reporter;
  int users;
};
```

Ainsi, le membre *ip_mc_list* de la structure *dev* est un pointeur sur une liste chaînée de structures *ip_mc_list*, chacune contenant une entrée pour chaque groupe multicast pour laquelle l'interface est membre. Ici nous voyons une fois de plus que les appartenances sont associées à des interfaces. LINUX/net/ipv4/ip_input.c parcourt cette liste chaînée pour décider si le datagramme reçu est destiné à un groupe auquel l'interface appartient :

```
#ifdef CONFIG_IP_MULTICAST
  if(!(dev->flags&IFF_ALLMULTI) && brd==IS_MULTICAST
     && iph->daddr!=IGMP_ALL_HOSTS
     && !(dev->flags&IFF_LOOPBACK))
  {
    /*
     * Verification de l'appartenance à un de nos groupes
     */
     struct ip_mc_list *ip_mc=dev->ip_mc_list;
     do
     {
       if(ip_mc==NULL)
       {
         kfree_skb(skb, FREE_WRITE);
         return 0;
       }
       if(ip_mc->multiaddr==iph->daddr)
         break;
       ip_mc=ip_mc->next;
```

```
      }
      while(1);
   }
#endif
```

Le champ *users* de la strucuture *ip_mc_list* est utilisé pour implémenter ce que nous avons dit dans la section IGMP version 1 : si un processus joint un groupe et que l'interface est déjà membre de ce groupe (ie, un autre processus joint le même groupe sur la même interface que précédemment) seul le compteur du nombre de membres (*users*) est incrémenté. Aucun message IGMP n'est envoyé, comme vous pouvez le voir dans le code suivant (extrait de *ip_mc_inc_group()*, appelé par *ip_mc_join_group()*, provenant du fichier LINUX/net/ipv4/igmp.c) :

```
for(i=dev->ip_mc_list;i!=NULL;i=i->next)
  {
    if(i->multiaddr==addr)
    {
      i->users++;
      return;
    }
  }
```

À chaque suppression d'une souscription de groupe, le compteur est décrémenté. Des opérations supplémentaires sont appliquées seulement si le compteur est devient égal à 0 (fonction *ip_mc_dec_group()*).

MRT_ADD_MFC et *MRT_DEL_MFC* ajoutent ou suppriment des entrées de retransmission dans la table de routage multicast. L'un comme l'autre passent un *struct mfcctl* au noyau (défini dans /usr/include/linux/mroute.h) avec cette information :

```
struct mfcctl
{
  struct in_addr mfcc_origin;       /* Origine du mcast      */
  struct in_addr mfcc_mcastgrp;     /* Groupe en question    */
  vifi_t  mfcc_parent;              /* Où cela arrive     */
  unsigned char mfcc_ttls[MAXVIFS]; /* Où cela va t-il    */
};
```

Avec toute cette information en main, *ipmr_forward()* parcourt les VIFs, et si une correspondance est trouvée, on duplique le datagramme et on appelle *ipmr_queue_xmit()* qui, à tour de rôle, utilise le périphérique de sortie indiqué par la table de routage et une adresse de destination propre si le paquet est à envoyer au travers d'un tunnel (il s'agit en fait de l'adresse de destination unicast de l'autre coté du tunnel).

La fonction *ip_rt_event()* (qui n'est pas directement apparentée à la sortie des datagrammes, mais qui se trouve pourtant dans ip_output.c) reçoit les évènements relatifs aux périphériques réseaux, tel que le branchement du périphérique par exemple. Cette fonction assure que le périphérique joint bien le groupe multicast *ALL-HOSTS*.

Les fonctions IGMP sont implémentées dans le fichier LINUX/net/ipv4/igmp.c. Les informations importantes concernant ces fonctions se trouvent dans /usr/include/linux/igmp.h et /usr/include/linux/mroute.h. L'entrée IGMP dans le répertoire /proc/net est créée à l'aide de la fonction *ip_init()* contenue dans le fichier LINUX/net/ipv4/ip_output.c.

8. Politiques de routage et techniques de retransmission

Un algorithme trivial pour rendre le trafic multicast disponible partout est de l'envoyer… partout, sans se soucier des besoins des utilisateurs. Comme cela ne semble pas être bien optimisé, plusieurs algorithmes de routage et techniques de retransmission ont été implémentés.

DVMRP (Distance Vector Multicast Routing Protocol, ou Protocole de routage multicast basé sur des vecteurs de distances) est, peut-être, le plus utilisé par les routeurs multicast de nos jours. Il s'agit d'un protocole de routage en mode dense, de fait, il convient bien dans les environnements à large bande passante et dont les membres sont densément distribués. Cependant, dans des scénarios moins denses, il souffre de problèmes de changement d'échelles.

Il existe d'autres protocoles de routage denses, tels que MOSPF (Extentions multicast à OSPF -Open Shortest Path First) et PIM-DM (Protocol-Independent Multicast Dense Mode).

Pour pouvoir faire du routage dans un environnement économe, il existe PIM-SM (Protocol Independent Multicast Sparse Mode) et CBT (Core Based Tree).

OSPF version 2 est expliqué dans les RFC 1583, et MOSPF dans le RFC 1584. Les spécifications de PIM-SM et CBT peuvent être trouvées respectivement dans les RFC 2117 et 2201.

Tous ces protocoles de routage utilisent différents types de retransmissions multicast, tels que le flooding (technique par indondation), RTB (ou Reverse Path Broadcasting), TRPB (ou Truncated Reverse Path Broadcasting), RPM (ou Reverse Path Multicasting) ou Shared Trees.

Il serait trop long d'expliquer toutes ces techniques ici et, comme de courtes descriptions sont publiquement disponibles, je vous recommende la lecture du texte draft-ietf-mboned-in.txt.

De même vous pouvez trouver les RFCs correspondants, expliquant en détails toutes les techniques et politiques employées.

9. Protocoles de transport multicast

Jusqu'à présent nous avons parlé des transmissions multicast via UDP. Cela est une pratique courante, car il est impossible de faire de même avec TCP. Cependant, une recherche importante a lieu depuis un certain nombre d'années et dont le but est de développer de nouveaux protocoles de transport multicast.

Plusieurs de ces protocoles ont été implémentés et testés. Une bonne leçon à tirer est qu'il ne semble pas y avoir de protocoles de transport multicast génériques et suffisament bons pour tous les types d'applications multicast.

Si les protocoles de transports sont complexes et difficiles à optimiser, imaginez une façon de gérer les délais, la perte de données, l'ordonnancement, les retransmissions, le contrôle de flux et des congestions, le management des groupes, etc, lorsque le destinataire n'est pas unique, mais peut être constitué de centaines ou de milliers d'hôtes dispersés. Le changement d'échelle est donc un problème. De nouvelles techniques sont à implémenter, tel que ne pas délivrer des accusés de réception à chaque paquet reçu mais, à la place, envoyer des accusés de non-réception (negative acknowledges, ou NACKs) pour les données qui n'ont pas été reçues. Le RFC 1458 donne des propositions relatives aux conditions nécessaires à l'implémentation de tels protocoles multicasts.

Donner la description de ces protocoles multicast sort de la portée de cette section. À la place, vous trouverez ci-dessous les noms de certains d'entre eux et quelques sources d'informations complémentaires : RTP, le protocole de transport temps-réel (ou Real-Time Transport Protocole) concerne les conférences multimédia multi-partite, SRM (Scalable Reliable Multicast) est utilisé par *wb* (l'outil servant de tableau blanc partagé, voyez la section applications multicast), URGC (Uniform Reliable Group Communication Protocol) permet des transactions fiables et ordonnées -ce protocole est basé sur un contrôle centralisé-, Muse a été développé comme un protocole d'application spécifique au transfert de nouvelles (articles) au travers du MBone, le MFTP (Protocole Multicast de Transport de Fichiers, ou Multicast File Transport Protocol) se décrit de lui-même ; les personnes « joignent » un transfert de fichier (précédemment annoncé) de la même façon qu'ils joignent une conférence, LBRM (Log-Based Receiver-reliable Multicast) est un protocole curieux qui permet de garder une trace de tous les paquets envoyés par un serveur « loggant » ces derniers -ce serveur indique à l'émetteur s'il doit retransmettre les données ou s'il peut les supprimer proprement dans le cas où tous les destinataires ont bien reçu les données. Un protocole avec un drôle de nom -spécialement pour un protocole multicast- est STORM (STructure-Oriented Resilient Multicast). De nombreux protocoles multicast peuvent être trouvés en cherchant sur Internet, de même que d'intéressants papiers proposant de nouveaux champs d'applications au Multicast (par exemple, la distribution de pages web en multicast).

Une bonne page fournissant des comparaisons entre les différents protocoles multicastes fiables est http://www.tascnets.com/mist/doc/mcpCompare.html.

Un très bon site (à jour !), avec des liens intéressants (Internet drafts, RFCs, papiers, liens vers d'autres sites) est : http://research.ivv.nasa.gov/RMP/links.html.

http://hill.lut.ac.uk/DS-Archive/MTP.html est aussi une bonne source d'informations sur ce sujet.

L'article de Katia Obraczka « Multicast Transport Protocols: A Survey and Taxonomy » donne de courtes descriptions pour chaque protocole et essaye de les classifer selon diverses fonctionnalités. Vous pouvez le lire dans le magazine IEEE Communication, Janvier 1998, vol 36, No 1.

10. Bibliographie

10.1. RFCs

- RFC 1112 « Host Extensions for IP Multicasting ». Steve Deering. Août 1989.
- RFC 2236 « Internet Group Management Protocol, version 2 ». W. Fenner. Novembre 1997.
- RFC 1458 « Requirements for Multicast Protocols ». Braudes, R and Zabele, S. Mai 1993.
- RFC 1469 « IP Multicast over Token-Ring Local Area Networks ». T. Pusateri. Juin 1993.
- RFC 1390 « Transmission of IP and ARP over FDDI Networks ». D. Katz. Janvier 1993.
- RFC 1583 « OSPF Version 2 ». John Moy. Mai 1994.
- RFC 1584 « Multicast Extensions to OSPF ». John Moy. Mai 1994.
- RFC 1585 « MOSPF: Analysis and Experience ». John Moy. Mai 1994.
- RFC 1812 « Requirements for IP version 4 Routers ». Fred Baker, Editor. Juin 1995.
- RFC 2117 « Protocol Independent Multicast-Sparse Mode (PIM-SM): Protocol Specification ». D. Estrin, D. Farinacci, A. Helmy, D. Thaler; S. Deering, M. Handley, V. Jacobson, C. Liu, P. Sharma, and L. Wei. Juillet 1997.
- RFC 2189 « Core Based Trees (CBT version 2) Multicast Routing ». A. Ballardie. Septembre 1997.
- RFC 2201 « Core Based Trees (CBT) Multicast Routing Architecture ». A. Ballardie. Septembre 1997.

10.2. Brouillons Internet (Internet Drafts)

- « Introduction to IP Multicast Routing ». draft-ietf-mboned-intro-multicast-03.txt. T. Maufer, C. Semeria. Juillet 1997
- « Administratively Scoped IP Multicast ». draft-ietf-mboned-admin-ip-space-03.txt. D. Meyer. 10 Juin 1997.

10.3. Pages web

- Page d'accueil du Multicast sous Linux. http://web.archive.org/web/20010209032950/http://www.cs.washington.edu/homes/esler/multicast/
- Foire Aux Questions (FAQ) Multicast sous Linux. http://andrew.triumf.ca/pub/linux/multicast-FAQ
- Multicast et MBONE sous Linux. http://web.archive.org/web/19990128143933/http://www.teksouth.com/linux/multicast/
- Liens sur les protocoles multicast fiables http://www.nard.net/~tmont/rm-links.html
- Protocoles de Transport Multicast http://web.archive.org/web/20020124084206/http://hill.lut.ac.uk/DS-Archive/MTP.html

10.4. Livres

- « TCP/IP Illustrated: Volume 1 The Protocols ». Stevens, W. Richard. Addison Wesley Publishing Company, Reading MA, 1994
- « TCP/IP Illustrated: Volume 2, The Implementation ». Wright, Gary and W. Richard Stevens. Addison Wesley Publishing Company, Reading MA, 1995
- « UNIX Network Programming Volume 1. Networking APIs: Sockets and XTI ». Stevens, W. Richard. Second Edition, Prentice Hall, Inc. 1998.
- « Internetworking with TCP/IP Volume 1 Principles, Protocols, and Architecture ». Comer, Douglas E. Second Edition, Prentice Hall, Inc. Englewood Cliffs, New Jersey, 1991

11. Licence et mise en garde

Si vous publiez ce document sur un CD-ROM ou sous la forme d'une copie physique, une copie complémentaire sera la bienvenue ; envoyez-moi un courrier électronique (en anglais) à l'adresse `<jmseyas CHEZ dit POINT upm POINT es>`. N'hesitez pas non plus à faire une donation au Projet de documentation Linux (LDP) ou à la Free Software Foundation pour apporter votre soutien au projet de documentation libre pour Linux/GNU. Contactez le Projet de documentation Linux `<feedback CHEZ en POINT tldp POINT org>` pour plus d'informations.