Équipe d'ingénierie de l'Internet (IETF)
Request for Comments : 7761
STD 83
RFC rendue obsolète : 4601
Catégorie : Norme
ISSN : 2070-1721
Traduction Claude Brière de L'Isle

B. Fenner, Arista Networks
M. Handley, UCL
H. Holbrook & I. Kouvelas, Arista Networks
R. Parekh, Cisco Systems, Inc.
Z. Zhang, Juniper Networks
L. Zheng, Huawei Technologies
mars 2016

Diffusion groupée indépendante du protocole – mode épars (PIM-SM) : spécification du protocole (révisé)

Résumé
Le présent document spécifie la diffusion groupé indépendante du protocole - mode épars (PIM-SM). PIM-SM est un protocole d'acheminement de diffusion groupée qui peut utiliser la base d'informations d'envoi individuel sous-jacente ou une base d'informations d'acheminement capable de diffusion groupée séparée. Il construit des arborescences partagées unidirectionnelles qui prennent racine à un point de rendez-vous (RP) par groupe, et crée facultativement des arborescences de plus court chemin par source.

Le présent document rend obsolète la RFC 4601 et la remplace ; il incorpore ses errata et retire les caractéristiques facultatives (*,*,RP), de routeur bordure de diffusion groupée PIM et l'authentification avec IPsec qui n'avaient pas une expérience de déploiement suffisante (voir l'Appendice A), et passe la spécification de PIM au statut de norme de l'Internet.

Statut de ce mémoire
Ceci est une norme de l'Internet.

Le présent document a été produit par l'équipe d'ingénierie de l'Internet (IETF). Il représente le consensus de la communauté de l'IETF. Il a subi une révision publique et sa publication a été approuvée par le groupe de pilotage de l'ingénierie de l'Internet (IESG). Plus d'informations sur les normes de l'Internet sont disponibles à la Section 2 de la [RFC5741].

Les informations sur le statut actuel du présent document, tout errata, et comment fournir des réactions sur lui peuvent être obtenues à http://www.rfc-editor.org/info/rfc7761

Table des matières

1. Introduction

Le présent document spécifie un protocole pour l'acheminement efficace des groupes de diffusion groupée qui peuvent s'étendre sur des internets de large zone (et des inter-domaines). Ce protocole s'appelle diffusion groupée indépendante du protocole en mode épars (PIM-SM, *Protocol Independent Multicast - Sparse Mode*) parce que, bien qu'il puisse utiliser l'acheminement d'envoi individuel sous-jacent pour fournir les informations de chemin inverse pour la construction de l'arborescence de diffusion groupée, il ne dépend d'aucun protocole d'acheminement en envoi individuel particulier.

PIM-SM version 2 était spécifié dans la RFC 4601 comme proposition de norme. Le présent document est destiné à traiter les erreurs rapportées et à supprimer le (*,*,RP) facultatif, les caractéristiques de routeur bordure de diffusion groupée PIM et l'authentification avec IPsec qui manquent d'une expérience de déploiement suffisante, pour avancer PIM-SM au rang de norme de l'Internet.

Le présent document spécifie le même protocole que la RFC 4601, et les mises en œuvre de la spécification du présent document seront capables d'interopérer avec succès avec les mises en œuvre de la RFC 4601.

2. Terminologie

Dans ce document, les mots clés "DOIT", "NE DOIT PAS", "EXIGE", "DEVRA", "NE DEVRA PAS", "DEVRAIT", "NE DEVRAIT PAS", "RECOMMANDE", "PEUT", et "FACULTATIF" sont à interpréter comme décrit dans la [RFC2119].

2.1 Définitions

Les termes qui suivent ont une signification particulière pour PIM-SM :

Point de rendez-vous (RP, *Rendezvous Point*) :
Un RP est un routeur qui a été configuré pour être utilisé comme racine de l'arborescence de la répartition non spécifique de source pour un groupe de diffusion groupée. Les messages Join provenant des receveurs pour un groupe sont envoyés vers le RP, et les données provenant des envoyeurs sont envoyées au RP afin que les receveurs puissent découvrir qui sont les envoyeurs et commencer à recevoir le trafic destiné au groupe.

Routeur désigné (DR) :
Un LAN à support partagé comme Ethernet peut avoir plusieurs routeurs PIM-SM connectés. Un seul de ces routeurs, le DR, va agir au nom des hôtes directement connectés par rapport au protocole PIM-SM. Un seul DR est choisi par interface (LAN ou autre) en utilisant un processus d'élection simple.

Base de données d'informations d'acheminement de diffusion groupée (MRIB, *Multicast Routing Information Base*)
C'est le tableau de la topologie de la diffusion groupée, qui est normalement déduite du tableau d'acheminement d'envoi individuel, ou de protocoles d'acheminement tels que le protocole de routeurs frontières multi protocoles (MBGP, *Multiprotocol Border Gateway Protocol*) qui portent des informations de topologie spécifiques de la diffusion groupée. Dans PIM-SM, la MRIB est utilisée pour décider où envoyer les messages Join/Prune. Une fonction secondaire de la MRIB est de

fournir des métriques d'acheminement pour les adresses de destination ; ces métriques sont utilisées lors de l'envoi et du traitement des messages Assert.

Voisin RPF
RPF *(Reverse Path Forwarding)* signifie "transmission sur le chemin inverse". Le voisin RPF d'un routeur par rapport à une adresse est le voisin qu'indique la MRIB comme devant être utilisé pour transmettre des paquets à cette adresse. Dans le cas d'un groupe de diffusion groupée PIM-SM, le voisin RPF est le routeur sur lequel un message Join pour ce groupe serait dirigé, en l'absence d'un état modifiant Assert.

TIB *(Tree Information Base)* Base de données d'informations d'arborescence
C'est la collection des états d'un routeur PIM qui a été créée en recevant des messages PIM Join/Prune, des messages PIM Assert, et des informations du protocole de gestion de groupe sur Internet (IGMP, *Internet Group Management Protocol*) ou de découverte des appareils en veille sur la diffusion groupée (MLD, *Multicast Listener Discovery*) provenant des hôtes locaux. Elle mémorise essentiellement l'état de toutes les arborescences de distribution de diffusion groupée chez ce routeur.

MFIB *(Multicast Forwarding Information Base)* Base de données d'informations de transmission de diffusion groupée
La TIB détient tous les états qui sont nécessaires pour transmettre des paquets en diffusion groupée à un routeur. Cependant, bien que la présente spécification définisse la transmission en termes de TIB, transmettre réellement des paquets en utilisant la TIB est très inefficace. Bien sûr, une mise en œuvre réelle de routeur va normalement construire une MFIB efficace à partir de l'état de la TIB pour effectuer la transmission. La façon dont cela est fait est spécifique de la mise en œuvre et n'est pas discuté dans ce document.

En amont : Vers la racine de l'arborescence. La racine de l'arborescence peut être la source ou le RP, selon le contexte.

En aval : En s'éloignant de la racine de l'arborescence.

GenID *(Generation Identifier)* Identifiant de génération, utilisé pour détecter les réamorçages.

2.2 Notation en pseudo code

On utilise cette notation en plusieurs endroits de cette spécification.
A (+) B est l'union de deux ensembles, A et B.
A (-) B sont les éléments de l'ensemble A qui ne sont pas dans l'ensemble B.
NUL est l'ensemble ou liste vide.

De plus, on utilise une syntaxe dans le style de celle du langage C :
= note l'affectation d'une variable.
== note une comparaison pour égalité.
!= note une comparaison pour inégalité.
Les accolades { et } sont utilisées pour grouper.

Sauf mention contraire, les opérations spécifiées par des déclarations qui ont plusieurs opérateurs (+) et (-) devraient être évaluées de gauche à droite, c'est-à-dire, A (+) B (-) C est l'ensemble résultant de l'union des ensembles A et B moins les éléments de l'ensemble C.

3. Généralités sur le protocole PIM-SM

Cette section donne une vue générale du comportement PIM-SM. Elle est destinée à constituer une introduction au fonctionnement de PIM-SM, et elle N'EST PAS définitive. Pour la spécification définitive, voir la Section 4.

PIM s'appuie sur un protocole de rassemblement de la topologie sous-jacente pour remplir de chemins un tableau d'acheminement. Ce tableau d'acheminement est appelé la base de données d'informations d'acheminement de diffusion groupée (MRIB, *Multicast Routing Information Base*). Dans ce tableau les chemins peuvent être pris directement du tableau d'acheminement d'envoi individuel, ou ils peuvent être différents et fournis par un protocole d'acheminement distinct tel que MBGP [RFC4760]. Sans considération de la façon dont elle est créée, le principal rôle de la MRIB dans le protocole PIM est de fournir le routeur du prochain bond ainsi qu'un chemin à capacité de diffusion groupée pour chaque sous-réseau de destination. La MRIB est utilisée pour déterminer le voisin de prochain bond auquel est envoyé tout message PIM Join/Prune. Les flux de données s'écoulent le long du chemin inverse de celui des messages Join. Donc, à l'opposé de la RIB d'envoi individuel qui spécifie le prochain bond qu'un paquet de données va prendre vers un certain sous-réseau, la MRIB donne les

informations du chemin inverse et indique le chemin qu'un paquet de données de diffusion groupée devrait prendre, de son sous-réseau d'origine au routeur qui détient la MRIB.

Comme tous les protocoles d'acheminement de diffusion groupée qui mettent en œuvre le modèle de service de la [RFC1112], PIM-SM doit être capable d'acheminer les paquets de données des sources aux receveurs sans que ni les sources ni les receveurs ne connaissent à priori l'existence les uns des autres. Cela est essentiellement fait en trois phases, bien que, comme les envoyeurs et les receveurs peuvent aller et venir à tout moment, les trois phases puissent survenir simultanément.

3.1 Phase une : arborescence de RP

Dans la phase une, un receveur de diffusion groupée exprime son intérêt pour la réception de trafic destiné à un groupe de diffusion groupée. Normalement, il le fait en utilisant IGMP [RFC3376] ou MLD [RFC2710], mais d'autres mécanismes peuvent aussi servir à cette fin. Un des routeurs locaux du receveur est choisi comme routeur désigné (DR) pour ce sous-réseau. À réception de l'expression d'intérêt du receveur, le DR envoie alors un message PIM Join *(se joindre à la PIM)* vers le RP pour ce groupe de diffusion groupée. Ce message Join est appelé un (*,G) Join parce que il se joint au groupe G pour toutes les sources vers ce groupe. Le (*,G) Join voyage bond par bond vers le RP pour le groupe, et dans chaque routeur qu'il traverse, l'état de l'arborescence de diffusion groupée pour le groupe G est instancié. Finalement, le (*,G) Join atteint le RP ou atteint un routeur qui a déjà l'état (*,G) Join pour ce groupe. Lorsque de nombreux receveurs rejoignent le groupe, leurs messages Join convergent sur le RP et forment un arbre de distribution pour le groupe G qui a sa racine au RP. Ceci est appelé l'arborescence de RP (RPT, *RP Tree*) et est aussi appelé l'arborescence partagée parce qu'elle est partagée par toutes les sources qui envoient à ce groupe. Les messages Join sont renvoyés périodiquement tant que le receveur reste dans le groupe. Lorsque tous les receveurs sur un réseau d'extrémité quittent le groupe, le DR va envoyer un message PIM (*,G) Prune *(élaguer PIM)* vers le RP pour ce groupe de diffusion groupée. Cependant, si le message Prune n'est pas envoyé pour une raison quelconque, l'état va finalement arriver à expiration.

Un envoyeur de données de diffusion groupée commence simplement par envoyer des données destinées à un groupe de diffusion groupée. Le routeur local (DR) de l'envoyeur prend ces paquets de données, les encapsule en envoi individuel, et les envoie directement au RP. Le RP reçoit ces paquets de données encapsulés, les désencapsule, et les transmet sur l'arborescence partagée. Les paquets suivent alors l'état d'arborescence de diffusion groupée (*,G) dans les routeurs sur l'arborescence RP, étant dupliqués chaque fois que l'arborescence RP a des embranchements, et atteignant finalement tous les receveurs pour ce groupe de diffusion groupée. Le processus d'encapsulation des paquets de données au RP est appelé enregistrement, et les paquets d'encapsulation sont appelés paquets d'enregistrement PIM.

À la fin de la phase une, le trafic de diffusion groupée s'écoule encapsulé vers le RP, puis naturellement tout le long de l'arborescence de RP jusqu'aux receveurs de diffusion groupée.

3.2 Phase deux : Register-Stop

L'enregistrement-encapsulation des paquets de données est inefficace pour deux raisons :

- L'encapsulation et la désencapsulation peuvent être des opérations relativement coûteuses à effectuer pour un routeur, selon que le routeur a ou non le matériel approprié pour ces tâches.
- Voyager jusqu'au RP, et ensuite retourner le long de l'arborescence partagée peut avoir pour résultat que les paquets voyagent sur une distance relativement longue pour atteindre des receveurs qui sont proches de l'envoyeur. Pour certaines applications, cette latence ou consommation de bande passante accrue est indésirable.

Bien que l'enregistrement-encapsulation puisse continuer indéfiniment, pour ces raisons, le RP va normalement choisir de passer à la transmission native. Pour faire cela, lorsque le RP reçoit un paquet de données à enregistrer-encapsuler de la source S sur le groupe G, il va normalement initier un (S,G) Join spécifique de source vers S. Ce message Join voyage bond par bond vers S, instanciant l'état d'arborescence de diffusion groupée (S,G) dans les routeurs le long du chemin. L'état d'arborescence de diffusion groupée (S,G) n'est utilisé que pour transmettre des paquets pour le groupe G si ces paquets viennent de la source S. Le message Join atteint finalement le sous réseau de S ou un routeur qui est déjà dans l'état d'arborescence de diffusion groupée (S,G), et ensuite les paquets commencent à s'écouler à partir de S en suivant l'état d'arborescence (S,G) vers le RP. Ces paquets de données peuvent aussi atteindre des routeurs avec l'état (*,G) le long du chemin vers le RP ; si ils le font, ils peuvent couper à ce point sur l'arborescence du RP.

Tandis que le RP est dans le processus de jonction à l'arborescence spécifique de source pour S, les paquets de données vont continuer d'être encapsulés vers le RP. Lorsque les paquets provenant de S commencent aussi à arriver en natif au RP, le RP va recevoir deux copies de chacun de ces paquets. À ce point, le RP commence à éliminer la copie encapsulée de ces paquets, et il renvoie un message Register-Stop *(arrêter d'enregistrer)* au DR de S pour empêcher le DR d'encapsuler inutilement les paquets.

À la fin de la phase deux, le trafic va s'écouler de façon native de S le long de l'arborescence spécifique de source jusqu'au RP, et de là le long de l'arborescence partagée jusqu'aux receveurs. Si les deux arborescences se coupent, le trafic peut se transférer de l'arborescence spécifique de source à l'arborescence de RP et donc éviter de faire un long détour via le RP.

Noter qu'un envoyeur peut commencer d'envoyer avant ou après qu'un receveur se joint au groupe, et donc la phase deux peut se produire avant la construction de l'arborescence partagée jusqu'au receveur.

3.3 Phase trois : arborescence de plus court chemin

Bien que d'avoir le RP qui rejoint la source supprime la redondance d'encapsulation, cela n'optimise pas complètement les chemins de transmission. Pour de nombreux receveurs, le chemin via le RP peut impliquer un détour significatif lorsque on le compare au plus court chemin de la source au receveur.

Pour obtenir de plus faibles latences ou une utilisation plus efficace de la bande passante, un routeur sur le LAN du receveur, normalement le DR, peut facultativement initier un transfert de l'arborescence partagée à une arborescence de plus court chemin (SPT, *shortest-path tree*) spécifique de source. Pour ce faire, il produit un (S,G) Join vers S. Cela instancie l'état dans les routeurs le long du chemin vers S. Finalement, ce Join atteint soit le sous réseau de S, soit un routeur qui a déjà l'état (S,G). Lorsque cela arrive, les paquets de données provenant de S commencent à s'écouler en suivant l'état (S,G) jusqu'à ce qu'ils atteignent le receveur.

À ce point, le receveur (ou un routeur en amont du receveur) va recevoir deux copies des données : une de la SPT et une de la RPT. Lorsque le premier trafic commence à arriver de la SPT, le DR ou le routeur en amont commence à éliminer les paquets pour G provenant de S qui arrivent via l'arborescence du RP. De plus, il envoie un message (S,G) Prune au RP. Ceci est appelé un (S,G,rpt) Prune. Le message Prune *(élaguer)* voyage bond par bond, instanciant l'état le long du chemin vers le RP, indiquant que du trafic de S pour G NE DEVRAIT PAS être transmis dans cette direction. L'élagage est propagé jusqu'à ce qu'il atteigne le RP ou un routeur qui a encore besoin du trafic de S pour d'autres receveurs.

À partir de là, le receveur va recevoir le trafic de S le long de l'arborescence de plus court chemin entre le receveur et S. De plus, le RP reçoit le trafic provenant de S, mais ce trafic n'atteint plus le receveur le long de l'arborescence du RP. Pour autant que le receveur soit concerné, c'est l'arborescence de distribution finale.

3.4 Jonctions spécifiques de source

IGMPv3 permet à un receveur de rejoindre un groupe et de spécifier qu'il veut recevoir du trafic pour un groupe seulement si ce trafic provient d'une source particulière. Si un receveur fait cela, et si aucun autre receveur sur le LAN n'exige tout le trafic pour le groupe, alors le DR peut omette d'effectuer un (*,G) Join pour établir l'arborescence partagée, et produire à la place seulement un (S,G) spécifique de source.

La gamme des adresses de diffusion groupée de 232.0.0.0 à 232.255.255.255 est actuellement laissée de côté pour la diffusion groupée spécifique de source dans IPv4. Pour les groupes dans cette gamme, les receveurs devraient seulement produire des Join IGMPv3 spécifiques de source. Si un routeur PIM reçoit un Join non spécifique de source pour un groupe dans cette gamme, il devrait l'ignorer.

3.5 Élagages spécifiques de source

IGMPv3 permet aussi à un receveur de se joindre à un groupe et de spécifier qu'il veut seulement recevoir du trafic pour un groupe si ce trafic ne vient pas d'une ou plusieurs sources spécifiques. Dans ce cas, le DR va effectuer un (*,G) Join normal, mais peut combiner cela avec un (S,G,rpt) Prune pour chacune des sources que le receveur ne souhaite pas recevoir.

3.6 LAN de transit multi accès

On a vu jusqu'à présent les liaisons de transit en point à point. Cependant, l'utilisation de LAN multi accès comme un Ethernet pour le transit n'est pas rare. Cela peut causer des complications pour trois raisons :

- Deux routeurs ou plus sur le LAN peuvent produire des (*,G) Join à différents routeurs en amont sur le LAN parce qu'ils ont des entrées de MRIB incohérentes quant à la façon d'atteindre le RP. Les deux chemins sur l'arborescence du RP vont être établis, causant l'apparition de deux copies de tout le trafic de l'arborescence partagée sur le LAN.
- Deux routeurs ou plus sur le LAN peuvent produire des (S,G) Join à différents routeurs en amont sur le LAN parce qu'ils ont des entrées de MRIB incohérentes quant à la façon d'atteindre la source S. Les deux chemins sur l'arborescence spécifique de source vont être établis, causant l'apparition de deux copies de tout le trafic provenant de S sur le LAN.

- Un routeur sur le LAN peut produire un (*,G) Join à un routeur en amont sur le LAN, et un autre routeur sur le LAN peut produire un (S,G) Join à un routeur amont différent sur le même LAN. Le trafic provenant de S peut atteindre le LAN sur le RPT et le SPT à la fois. Si le receveur derrière le routeur (*,G) aval ne produit pas un (S,G,rpt) Prune, cette condition va alors persister.

Tous ces problèmes sont causés par le fait qu'il y a plus d'un routeur amont avec l'état Join pour le groupe ou la paire source-groupe. PIM n'empêche pas que se produisent de telles jonctions dupliquées ; lorsque des paquets de données dupliqués apparaissent sur le LAN provenant de différents routeurs, ces routeurs le remarquent puis choisissent un seul transmetteur. Ce choix est effectué en utilisant les messages PIM Assert, qui résolvent le problème en faveur du routeur amont qui a l'état (S,G) ; ou si ni l'un ni l'autre, ou si les deux routeurs ont l'état (S,G), le problème est résolu en faveur de celui qui a la meilleure métrique pour le RP pour les arborescences RP, ou la meilleure métrique pour la source pour les arborescences spécifiques de source.

Ces messages Assert sont aussi reçus par les routeurs en aval sur le LAN, et ils causent l'envoi des messages Join suivants au routeur amont qui a gagné le Assert.

3.7 Découverte du RP

Les routeurs PIM-SM ont besoin de savoir l'adresse du RP pour chaque groupe pour lequel ils ont l'état (*,G). Cette adresse est obtenue automatiquement (par exemple, RP incorporé) à travers un mécanisme d'amorçage, ou par configuration statique.

Une façon dynamique de faire cela est d'utiliser le mécanisme de routeur d'amorçage (BSR, *Bootstrap Router*) [RFC5059]. Un routeur dans chaque domaine PIM est choisi comme BSR par un simple processus d'élection. Tous les routeurs dans le domaine qui sont configurés à être candidats comme RP envoient périodiquement en individuel leur candidature au BSR. À partir des candidats, le BSR tire un ensemble de RP, et annonce périodiquement cet ensemble dans un message Bootstrap. Les messages Bootstrap sont arrosés bond par bond dans tout le domaine jusqu'à ce que tous les routeurs du domaine connaissent l'ensemble de RP.

Pour faire correspondre un groupe à un RP, un routeur hache les adresses du groupe dans l'ensemble de RP en utilisant une fonction de hachage qui préserve l'ordre (qui minimise les changements si l'ensemble de RP change). Le RP résultant est celui qu'il utilise comme RP pour ce groupe.

4. Spécification du protocole

La spécification de PIM-SM est séparée en plusieurs parties :
- Le paragraphe 4.1 détaille l'état de protocole mémorisé.
- Le paragraphe 4.2 spécifie les règles de transmission du paquet de données.
- Le paragraphe 4.3 spécifie l'élection du routeur désigné (DR) et les règles d'envoi et de traitement des messages Hello.
- Le paragraphe 4.4 spécifie les règles de génération et de traitement d'enregistrement PIM.
- Le paragraphe 4.5 spécifie les règles de génération et de traitement d'enregistrement des Join/Prune PIM.
- Le paragraphe 4.6 spécifie les règles de génération et de traitement d'enregistrement de PIM Assert.
- Le paragraphe 4.7 spécifie les mécanismes de découverte de RP.
- Le paragraphe 4.8 décrit PIM-SSM, le sous ensemble de PIM requis pour la prise en charge de la diffusion groupée spécifique de source.
- Le paragraphe 4.9 spécifie les formats de paquet PIM.
- Le paragraphe 4.10 donne un résumé des temporisateurs PIM-SM, et le paragraphe 4.11 donne leurs valeurs par défaut.

4.1 État de protocole PIM

Ce paragraphe spécifie tous les états de protocole qu'une mise en œuvre de PIM devrait entretenir afin de fonctionner correctement. On appelle cet état la base de données d'information d'arborescence (TIB, *Tree Information Base*) car il détient les états de toutes les arborescences de distribution de diffusion groupée à ce routeur. Dans la présente spécification, on définit les mécanismes PIM en termes de TIB. Cependant, seule une mise en œuvre très simple traiterait les opérations de transmission de paquets selon les termes de cet état. La plupart des mises en œuvre vont utiliser cet état pour construire un tableau de transmission de diffusion groupée, qui va alors être mis à jour lorsque les états pertinents changent dans la TIB.

Bien qu'on spécifie précisément l'état à conserver, cela ne signifie pas qu'une mise en œuvre de PIM-SM a besoin de conserver l'état sous cette forme. C'est en fait une définition d'état abstraite, qui est nécessaire pour spécifier le comportement du routeur. Une mise en œuvre de PIM-SM est libre de conserver tout état interne qu'elle requiert et sera quand même conforme à la

présente spécification pour autant que cela résulte en le même comportement de protocole visible de l'extérieur que celui d'un routeur abstrait qui détient l'état suivant.

On divise l'état TIB en trois sections :

état (*,G) : état qui conserve l'arborescence de RP pour G.

état (S,G) : état qui conserve une arborescence spécifique de source pour la source S et le groupe G.

état (S,G,rpt) : état qui conserve les informations spécifiques de source sur la source S sur l'arborescence de RP pour G. Par exemple, si une source est reçue sur l'arborescence spécifique de source, elle va normalement avoir été élaguée de l'arborescence de RP. Cet état élagué est l'état (S,G,rpt).

L'état qui devrait être conservé est décrit ci-dessous. Bien sûr, les mises en œuvre vont seulement conserver l'état lorsque il est pertinent pour les opérations de transmission ; par exemple, l'état "NoInfo" peut être déduit de l'absence d'autres informations d'état plutôt que d'être détenu explicitement.

4.1.1 État non spécifique

Un routeur détient l'état non spécifique de groupe suivant :

Pour chaque interface :
- o Intervalle d'outrepassement effectif
- o Délai de propagation effectif
- o État de suppression d'interface : un de {"Activé", "Désactivé"}

État voisin :
Pour chaque voisin :
- o Information provenant du Hello du voisin
- o Identifiant de génération (GenID, *Generation Identifier*) du voisin
- o Temporisateur de vie de voisin (NLT, *Neighbor Liveness Timer*)

État de routeur désigné (DR) :
- o Adresse IP du routeur désigné
- o Priorité de DR du DR

L'intervalle d'outrepassement effectif, le délai de propagation effectif, et l'état de suppression d'interface sont décrits au paragraphe 4.3.3. L'état de routeur désigné est décrit au paragraphe 4.3.

4.1.2. État (*,G)

Pour tout groupe G, un routeur conserve l'état suivant :
état (*,G) : pour chaque interface :
adhésion locale : état : un de {"NoInfo", "Include"}

état PIM (*,G) Join/Prune :
- o état : un de {"NoInfo" (NI), "Join" (J), "Prune-Pending" (PP)}
- o temporisateur d'élagage en cours (PPT, *Prune-Pending Timer*)
- o temporisateur d'expiration (ET, *Expiry Timer*) de Join/Prune

état gagnant d'assertion (*,G)
- o état : un de {"NoInfo" (NI), "Assertion perdue" (L), "Assertion gagnée" (W)}
- o temporisateur d'assertion (AT, *Assert Timer*)
- o adresse IP de gagnant d'assertion (AssertWinner)
- o métrique d'assertion de gagnant d'assertion (AssertWinnerMetric)

Non spécifique de l'interface :
état (*,G) Join/Prune amont :
- o état : un de {"NonJoint(*,G)", "Joint(*,G)"}
- o temporisateur de Join/Prune amont (JT)
- o dernier RP utilisé
- o dernier voisin RPF vers le RP qui a été utilisé

L'adhésion locale est le résultat du mécanisme d'adhésion locale (comme IGMP ou MLD) qui fonctionne sur cette interface. Il n'est pas nécessaire de le conserver si ce routeur n'est pas le DR sur cette interface sauf si ce routeur a gagné une assertion (*,G) sur cette interface pour ce groupe, bien qu'une mise en œuvre puisse facultativement conserver cet état pour le cas où il deviendrait le DR ou le gagnant d'assertion. Il est RECOMMANDÉ de mémoriser ces informations si possible, car cela réduit la latence pour converger vers un fonctionnement stable après qu'une défaillance a causé un changement de DR. Ces informations sont utilisées par la macro pim_include(*,G) décrite au paragraphe 4.1.5.

L'état PIM (*,G) Join/Prune est le résultat de la réception des messages PIM (*,G) Join/Prune sur cette interface et est spécifié au paragraphe 4.5.1. L'état est utilisé par les macros qui calculent la liste d'interfaces sortantes au paragraphe 4.1.5, et dans la macro JoinDesired(*,G) (définie au paragraphe 4.5.4) qui est utilisée pour décider si un Join(*,G) devrait être envoyé en amont.

L'état gagnant d'assertion (*,G) est le résultat de l'envoi ou de la réception de messages Assertion (*,G) sur cette interface. Il est spécifié au paragraphe 4.6.2.

L'état (*,G) Join/Prune amont reflète l'état de l'automate à états (*,G) décrit au paragraphe 4.5.4.

Le temporisateur (*,G) Join/Prune amont est utilisé pour envoyer des messages périodiques Join(*,G), et pour outrepasser les messages Prune(*,G) provenant des homologues sur l'interface amont de LAN.

Le dernier RP utilisé doit être mémorisé parce que si le RP change, l'état doit alors être supprimé et reconstruit pour les groupes dont le RP change.

Le dernier voisin RPF vers le RP est mémorisé parce que si la MRIB change, alors le voisin RPF vers le RP peut changer. Si il le fait, on a alors besoin de déclencher un nouveau Join(*,G) pour le nouveau voisin amont et un Prune(*,G) à l'ancien voisin amont. De même, si un routeur détecte par un GenID changé dans un message Hello que le voisin amont vers le RP a réamorcé, il DEVRAIT alors réinstancier l'état en envoyant un Join(*,G). Ces mécanismes sont spécifiés au paragraphe 4.5.4.

4.1.3 État (S,G)

Pour toute paire source/groupe (S,G), un routeur conserve l'état suivant :

état (S,G) :
Pour chaque interface :
adhésion locale : état : un de {"NoInfo", "Include"}
état PIM (S,G) Join/Prune :
- état : un de {"NoInfo" (NI), "Join" (J), "Prune-Pending" (PP)}
- temporisateur élagage en cours (PPT, *Prune-Pending Timer*)
- temporisateur d'expiration Join/Prune (ET)

état gagnant d'assertion (S,G)
- état : un de {"NoInfo" (NI), "Assertion perdue" (L), "Assertion gagnée" (W)}
- temporisateur d'assertion (AT)
- adresse IP de gagnant d'assertion (AssertWinner)
- métrique d'assertion de gagnant d'assertion (AssertWinnerMetric)

Non spécifique d'interface :
état (S,G) Join/Prune amont :
- état : un de {"NonJoint(S,G)", "Joint(S,G)"}
- temporisateur de (S,G) Join/Prune amont (JT)
- dernier voisin RPF vers S utilisé
- bit SPT (indique que l'état (S,G) est actif)
- temporisateur de garde en vie de (S,G) (KAT, *Keepalive Timer*)

état (S,G) supplémentaire au DR :
- enregistrer l'état : un de {"Join" (J), "Prune" (P), "Join-Pending" (JP), "NoInfo" (NI)}
- temporisateur arrêter d'enregistrer (RST, *Register-Stop Timer*)

L'adhésion locale est le résultat du mécanisme d'adhésion spécifique de source locale (comme IGMP version 3) fonctionnant sur cette interface et qui spécifie que cette source particulière devrait être incluse. Comme il est mémorisé ici, cet état est celui qui résulte après que toutes les incohérences IGMPv3 ont été résolues. Il n'est pas nécessaire de le conserver si ce routeur n'est

pas le DR sur cette interface sauf si ce routeur a gagné une assertion (S,G) sur cette interface pour ce groupe. Cependant, il est RECOMMANDÉ de mémoriser ces informations si possible, car cela réduit la latence en convergeant vers des conditions de fonctionnement stable après qu'une défaillance a causé un changement de DR. Ces informations sont utilisées par la macro pim_include(S,G) décrite au paragraphe 4.1.5.

L'état PIM (S,G) Join/Prune est le résultat de la réception de messages PIM (S,G) Join/Prune sur cette interface et il est spécifié au paragraphe 4.5.2. L'état est utilisé par les macros qui calculent la liste des interfaces sortantes du paragraphe 4.1.5, et dans la macro JoinDesired(S,G) (définie au paragraphe 4.5.5) qui est utilisée pour décider si un Join(S,G) devrait être envoyé en amont.

L'état gagnant d'assertion (S,G) est le résultat de l'envoi ou de la réception de messages d'assertion (S,G) sur cette interface. Il est spécifié au paragraphe 4.6.1.

L'état (S,G) Join/Prune amont reflète l'état de l'automate à états (S,G) décrit au paragraphe 4.5.5.

Le temporisateur (S,G) Join/Prune amont est utilisé pour envoyer des messages périodiques Join(S,G), et pour outrepasser les messages Prune(S,G) provenant des homologues sur une interface amont de LAN.

Le dernier voisin RPF vers S est mémorisé parce que si la MRIB change, le voisin RPF vers S peut changer. Si il le fait, on a alors besoin de déclencher un nouveau Join(S,G) pour le nouveau voisin amont et un Prune(S,G) à l'ancien voisin amont. De même, si le routeur détecte par un changement de GenID dans un message Hello que le voisin en amont vers S a réamorcé, il DEVRAIT alors réinstancier l'état par l'envoi d'un Join(S,G). Ces mécanismes sont spécifiés au paragraphe 4.5.5.

Le bit SPT est utilisé pour indiquer si la transmission a lieu sur l'arborescence de plus court chemin (SPT) de (S,G) ou sur l'arborescence (*,G). Un routeur peut avoir l'état (S,G) et transmettre quand même sur l'état (*,G) durant l'intervalle où l'arborescence spécifique de source est en construction. Lorsque le bit SPT est FAUX, seul l'état de transmission (*,G) est utilisé pour transmettre les paquets de S à G. Lorsque le bit SPT est VRAI, les deux états de transmission (*,G) et (S,G) sont utilisés.

Le temporisateur de garde en vie de (S,G) est mis à jour par les données transmises en utilisant cet état de transmission (S,G). Il est utilisé pour garder l'état (S,G) en vie en l'absence de Join(S,G) explicites. Entre autres choses, ceci est nécessaire pour ce qu'on appelle les "règles de contournement" – lorsque le RP utilise des Join(S,G) pour arrêter l'encapsulation, et ensuite des élagages de (S,G) pour empêcher que le trafic atteigne sans nécessité le RP.

Sur un DR, l'état (S,G) Register est utilisé pour garder trace de la décision d'encapsuler les données pour le RP sur le tunnel Register ; le temporisateur (S,G) Register-Stop mesure le temps avant que l'encapsulation recommence pour un certain (S,G).

4.1.4 État (S,G,rpt)

Pour chaque paire source/groupe (S,G) pour lequel un routeur a aussi l'état (*,G), il conserve aussi l'état suivant :

état (S,G,rpt) :
pour chaque interface :
adhésion locale : état: un de {"NoInfo", "Exclude"}
état PIM (S,G,rpt) Join/Prune :
- état : un de {"NoInfo", "Élagué", "Élagage en cours"}
- temporisateur élagage en cours (PPT)
- temporisateur d'expiration Join/Prune (ET)

Non spécifique d'interface :
état (S,G,rpt) Join/Prune amont :
- état : un de {"RPTNonJoint(G)", "NonÉlagué(S,G,rpt)", "Élagué(S,G,rpt)"}
- Temporisateur d'outrepassement (OT, *Override Timer*)

L'adhésion locale est le résultat du mécanisme d'adhésion spécifique de source locale (comme IGMPv3) qui fonctionne sur cette interface et spécifie que bien qu'il y ait l'état (*,G) Include, cette source particulière devrait être exclue. Comme il est mémorisé ici, cet état est celui qui résulte après que toutes les incohérences IGMPv3 entre les membres du LAN ont été résolues. Il n'est pas nécessaire de le conserver si ce routeur n'est pas le DR sur cette interface sauf si ce routeur a gagné une (*,G) Assert sur cette interface pour ce groupe. Cependant, on RECOMMANDE de mémoriser ces informations si possible, car cela réduit la latence et converge vers des conditions de fonctionnement stable après une défaillance ayant causé un changement de DR. Ces informations sont utilisées par la macro pim_exclude(S,G) décrite au paragraphe 4.1.5.

L'état PIM (S,G,rpt) Join/Prune est le résultat de la réception de messages PIM (S,G,rpt) Join/Prune sur cette interface et est spécifié au paragraphe 4.5.3. L'état est utilisé par les macros qui calculent la liste d'interfaces sortantes du paragraphe 4.1.5, et dans les règles pour ajouter des messages Prune(S,G,rpt) aux messages Join(*,G) spécifiées au paragraphe 4.5.6.

L'état (S,G,rpt) Join/Prune amont est utilisé avec le temporisateur d'outrepassement pour envoyer les messages Outrepasser corrects en réponse aux messages Join/Prune envoyés par les homologues en amont sur un LAN. Cet état et ce comportement sont spécifiés au paragraphe 4.5.7.

4.1.5 Macros de résumé d'état

En utilisant cet état, on donne les définitions de "macro" suivantes, qu'on utilisera dans les descriptions des automates à états et le pseudo code dans les paragraphes qui suivent.

Les plus importantes macros sont celles qui définissent la liste d'interfaces sortantes (ou "olist") pour l'état concerné. Une olist peut être "immédiate" si elle est construite directement à partir de l'état du type pertinent. Par exemple, la immediate_olist(S,G) est la olist qui serait construite si le routeur avait seulement l'état (S,G) et pas (*,G) ou (S,G,rpt). À l'opposé, la olist "inherited" hérite de l'état d'autres types. Par exemple, la inherited_olist(S,G) est la olist qui est pertinente pour transmettre un paquet de S à G en utilisant les deux états, spécifique de source, et spécifique de groupe.

Il n'y a pas de immediate_olist(S,G,rpt), car l'état (S,G,rpt) est un état négatif ; il supprime les interfaces dans la olist (*,G) de la olist qui est en fait utilisée pour transmettre le trafic. La inherited_olist(S,G,rpt) est donc la liste d'interfaces sortantes qui serait utilisée pour la transmission d'un paquet de S à G sur l'arborescence de RP. C'est un strict sous ensemble de immediate_olist(*,G).

D'une façon générale, les inherited_olists sont utilisées pour la transmission, et les immediate_olists sont utilisées pour prendre des décisions sur la conservation d'état.

immediate_olist(*,G) = joins(*,G) (+) pim_include(*,G) (-) lost_assert(*,G)

immediate_olist(S,G) = joins(S,G) (+) pim_include(S,G) (-) lost_assert(S,G)

inherited_olist(S,G,rpt) = (joins(*,G) (-) prunes(S,G,rpt)) (+) (pim_include(*,G) (-) pim_exclude(S,G)) (-) (lost_assert(*,G) (+) lost_assert(S,G,rpt))

inherited_olist(S,G) = inherited_olist(S,G,rpt) (+) joins(S,G) (+) pim_include(S,G) (-) lost_assert(S,G)

Les macros pim_include(*,G) et pim_include(S,G) indiquent les interfaces auxquelles le trafic peut être transmis parce que les hôtes sont des membres locaux sur cette interface. Noter que normalement, seulement le DR se soucie des adhésions locales, mais lorsque une assertion se produit, le gagnant de l'assertion prend la responsabilité de la transmission du trafic aux membres locaux qui ont demandé du trafic sur un groupe ou une paire source/groupe.

pim_include(*,G) = { toutes interfaces I telles que : ((je_suis_DR(I) ET lost_assert(*,G,I) == FAUX) OU AssertWinner(*,G,I) == moi) ET local_receiver_include(*,G,I) }

pim_include(S,G) = { toutes interfaces I telles que : ((je_suis_DR(I) ET lost_assert(S,G,I) == FAUX) OU AssertWinner(S,G,I) == moi) ET local_receiver_include(S,G,I) }

pim_exclude(S,G) = { toutes interfaces I telles que : ((je_suis_DR(I) ET lost_assert(*,G,I) == FAUX) OU AssertWinner(*,G,I) == moi) ET local_receiver_exclude(S,G,I) }

La clause "local_receiver_include(S,G,I)" est vraie si le module IGMP/MLD ou autre mécanisme d'adhésion local a déterminé que les membres locaux sur l'interface I désirent recevoir le trafic envoyé spécifiquement par S à G. "local_receiver_include(*,G,I)" est vrai si le module IGMP/MLD ou autre mécanisme d'adhésion local a déterminé que les membres locaux sur l'interface I désirent recevoir tout le trafic envoyé à G (éventuellement excluant le trafic provenant d'un ensemble spécifique de sources). "local_receiver_exclude(S,G,I)" est vrai si "local_receiver_include(*,G,I)" est vrai mais qu'aucun des membres locaux ne désire recevoir de trafic de S.

L'ensemble "joins(*,G)" est l'ensemble de toutes les interfaces sur lesquelles le routeur a reçu des (*,G) Join :

joins(*,G) = { toutes interfaces I telles que DownstreamJPState(*,G,I) est soit Join, soit Prune-Pending }

DownstreamJPState(*,G,I) est l'état de l'automate à états finis du paragraphe 4.5.1.

L'ensemble "joins(S,G)" est l'ensemble de toutes les interfaces sur lesquelles le routeur a reçu des (S,G) Joins :

joins(S,G) = { toutes les interfaces I telles que DownstreamJPState(S,G,I) est soit Join, soit Prune-Pending }

DownstreamJPState(S,G,I) est l'état de l'automate à états finis du paragraphe 4.5.2.

L'ensemble "prunes(S,G,rpt)" est l'ensemble de toutes les interfaces sur lesquelles le routeur a reçu des (*,G) Joins et des (S,G,rpt) Prunes:

prunes(S,G,rpt) = { toutes les interfaces I telles que DownstreamJPState(S,G,rpt,I) est Prune ou PruneTmp }

DownstreamJPState(S,G,rpt,I) est l'état de l'automate à états finis du paragraphe 4.5.3.

L'ensemble "lost_assert(*,G)" est l'ensemble de toutes les interfaces sur lesquelles le routeur a reçu des (*,G) Joins mais a perdu une (*,G) assert. La macro lost_assert(*,G,I) est définie au paragraphe 4.6.5.

lost_assert(*,G) = { toutes les interfaces I telles que lost_assert(*,G,I) == VRAI }

L'ensemble "lost_assert(S,G,rpt)" est l'ensemble de toutes les interfaces sur lesquelles le routeur a reçu des (*,G) Joins mais a perdu une (S,G) assert. La macro lost_assert(S,G,rpt,I) est définie au paragraphe 4.6.5.

lost_assert(S,G,rpt) = { toutes les interfaces I telles que lost_assert(S,G,rpt,I) == VRAI }

L'ensemble "lost_assert(S,G)" est l'ensemble de toutes les interfaces sur lesquelles le routeur a reçu des (S,G) Joins mais a perdu une (S,G) assert. La macro lost_assert(S,G,I) est définie au paragraphe 4.6.5.

lost_assert(S,G) = { toutes les interfaces I telles que lost_assert(S,G,I) == VRAI }

Les définitions suivantes de macros en pseudo code sont aussi utilisées dans de nombreux endroits de cette spécification. Par principe, RPF' est le voisin RPF vers un RP ou une source sauf si une PIM-Assert a outrepassé le choix normal du voisin.

```
voisin RPF'(*,G) {
   si ( je_suis_perdant_de_Assert(*, G, RPF_interface(RP(G))) ) {
      retourner AssertWinner(*, G, RPF_interface(RP(G)) )
   } autrement {
      retourner NBR( RPF_interface(RP(G)), MRIB.next_hop( RP(G) ) )
   }
}

voisin RPF'(S,G,rpt) {
   si ( je_suis_perdant_de_Assert(S, G, RPF_interface(RP(G)) ) ) {
      retourner AssertWinner(S, G, RPF_interface(RP(G)) )
   } autrement {
      retourner RPF'(*,G)
   }
}

voisin RPF'(S,G) {
   si ( je_suis_perdant_de_Assert(S, G, RPF_interface(S) )) {
      retourner AssertWinner(S, G, RPF_interface(S) )
   } autrement {
      retourner NBR( RPF_interface(S), MRIB.next_hop( S ) )
   }
}
```

RPF'(*,G) et RPF'(S,G) indiquent le voisin à partir duquel les paquets de données devraient venir et auquel ils devraient être envoyés sur l'arborescence de RP et de SPT, respectivement.

RPF'(S,G,rpt) est à la base le RPF'(*,G) modifié par le résultat d'une Assert(S,G) sur RPF_interface(RP(G)). Dans un tel cas, les paquets provenant de S auront pour origine un routeur différent de RPF'(*,G). Si on a seulement l'état actif (*,G) Join, on doit accepter les paquets provenant de RPF'(S,G,rpt) et ajouter un Prune(S,G,rpt) aux messages périodiques Join(*,G) qu'on envoie à RPF'(*,G) (voir le paragraphe 4.5.6).

La fonction MRIB.next_hop(S) retourne une adresse du voisin PIM de prochain bond vers l'hôte S, comme indiqué par la MRIB actuelle. Si S est directement adjacent, MRIB.next_hop(S) retourne alors NUL. Au RP pour G, MRIB.next_hop(RP(G)) retourne NUL.

La fonction NBR(I, A) utilise les informations collectées par les messages PIM Hello pour transposer l'adresse IP A d'un routeur voisin PIM directement connecté sur l'interface I à la principale adresse IP du même routeur (paragraphe 4.3.4). La principale adresse IP d'un voisin est l'adresse qu'il utilise comme source de ses messages PIM Hello. Noter que l'adresse IP d'un voisin peut n'être pas unique au sein de la base de données de voisins PIM à cause de problèmes de portée. L'adresse doit, cependant, être unique parmi les adresses de tous les voisins PIM sur une interface spécifique.

je_suis_perdant_de_Assert(S, G, I) est vrai si l'automate à états Assert (au paragraphe 4.6.1) pour (S,G) sur l'interface I est dans l'état "je suis le perdant de l'Assert".

je_suis_perdant_de_Assert(*, G, I) est vrai si l'automate à états Assert (au paragraphe 4.6.2) pour (*,G) sur l'interface I est dans l'état "je suis le perdant de l'Assert".

4.2 Règles de transmission des paquets de données

Les règles de transmission de paquet PIM-SM sont définies ci-dessous en pseudo code
iif est l'interface entrante du paquet,
S est l'adresse de source du paquet,
G est l'adresse de destination du paquet (adresse de groupe),
RP est l'adresse du point de rendez-vous pour ce groupe,
RPF_interface(S) est l'interface que la MRIB indique comme étant utilisée pour acheminer les paquets à S,
RPF_interface(RP) est l'interface que la MRIB indique comme étant utilisée pour acheminer les paquets au RP, sauf au RP lorsque c'est l'interface de désencapsulation (l'interface "virtuelle" sur laquelle les paquets Register sont reçus).

D'abord, on redémarre (ou démarre) le temporisateur Garder en vie si la source est sur un sous réseau directement connecté.

Ensuite, on vérifie si le bit SPT devrait être établi parce qu'on est maintenant passé de l'arborescence de RP à celle de SPT.

Ensuite, on vérifie si le paquet devrait être accepté sur la base de l'état TIB et de l'interface sur laquelle le paquet est arrivé.

Si le paquet devrait être transmis en utilisant l'état (S,G), on construit alors une liste d'interfaces sortantes pour le paquet. Si cette liste n'est pas vide, on relance le temporisateur de garde en vie d'état (S,G).

Si le paquet devrait être transmis en utilisant l'état (*,G), on construit alors une liste d'interfaces sortantes pour le paquet. On vérifie aussi si on devrait initier un basculement pour commencer à recevoir cette source sur une arborescence de plus court chemin.

Finalement, on supprime l'interface entrante de la liste d'interfaces sortantes qu'on a créée, et si la liste d'interfaces sortantes résultante n'est pas vide, on transmet le paquet par ces interfaces.

À réception des données de S à G sur l'interface iif : si (DirectlyConnected(S) == VRAI ET iif == RPF_interface(S)) { régler KeepaliveTimer(S,G) à Keepalive_Period
# Note : une transition d'état Register ou une transition UpstreamJPState(S,G) peut survenir par suite du redémarrage du temporisateur de garde en vie *(KeepaliveTimer)*, et doit être traitée ici.
}

si (iif == RPF_interface(S) ET UpstreamJPState(S,G) == Joined ET inherited_olist(S,G) != NUL) {régler KeepaliveTimer(S,G) à Keepalive_Period
}

Update_SPTbit(S,G,iif)
  oiflist = NUL

si (iif == RPF_interface(S) ET SPTbit(S,G) == VRAI) { oiflist = inherited_olist(S,G)
} autrement si(iif == RPF_interface(RP(G)) ET SPTbit(S,G) == FAUX { oiflist = inherited_olist(S,G,rpt) CheckSwitchToSpt(S,G)
} autrement {

```
# Note : La vérification RPF a échoué. Une transition dans un automate à états Assert peut causer l'envoi d'un message
        Assert(S,G) ou Assert(*,G) sur l'interface iif. Voir les détails au paragraphe 4.6.

  si ( SPTbit(S,G) == VRAI ET iif est dans inherited_olist(S,G) ) { envoyer Assert(S,G) sur iif
  } autrement si ( SPTbit(S,G) == FAUX ET iif est dans inherited_olist(S,G,rpt) ) { envoyer Assert(*,G) sur iif
 }
}
```

oiflist = oiflist (-) iif transmet le paquet sur toutes les interfaces dans oiflist

Ce pseudo code emploie plusieurs définitions de "macro" :

DirectlyConnected(S) est VRAI si la source S est sur un sous ensemble qui est directement connecté à ce routeur (ou pour les paquets générés sur ce routeur).

inherited_olist(S,G) et inherited_olist(S,G,rpt) sont définies au paragraphe 4.1.

Fondamentalement, inherited_olist(S,G) est la liste d'interfaces sortantes pour les paquets transmis sur l'état (S,G), en prenant en compte l'état (*,G), les assertions, etc.

inherited_olist(S,G,rpt) est la liste d'interfaces sortantes pour les paquets transmis sur l'état (*,G), en prenant en compte l'état (S,G,rpt) Prune, les assertions, etc.

Update_SPTbit(S,G,iif) est défini au paragraphe 4.2.2.

CheckSwitchToSpt(S,G) est défini au paragraphe 4.2.1.

UpstreamJPState(S,G) est l'état de l'automate à états finis du paragraphe 4.5.5.

Keepalive_Period est définie au paragraphe 4.11.

Les messages PIM-Assert déclenchés par les données envoyées du code de transmission ci-dessus DEVRAIENT être limités en débit d'une façon qui dépend de la mise en œuvre.

4.2.1 Basculement de dernier bond sur le SPT

En mode épars PIM, les routeurs de dernier bond se joignent à l'arborescence partagée vers le RP. Une fois que le trafic provenant des sources pour les groupes adhérents arrive au routeur de dernier bond, il a l'option de basculer pour recevoir le trafic sur une arborescence de plus court chemin (SPT).

La décision d'un routeur de basculer sur la SPT est contrôlée comme suit :

```
void
CheckSwitchToSpt(S,G) {
 si ( ( pim_include(*,G) (-) pim_exclude(S,G)
       (+) pim_include(S,G) != NUL )
      ET SwitchToSptDesired(S,G) ) {
# Note : Redémarrer le KAT aura pour résultat le basculement sur la SPT.
       régler le KeepaliveTimer(S,G) à Keepalive_Period
  }
}
```

SwitchToSptDesired(S,G) est une fonction de politique qui est définie par la mise en œuvre. Une politique de "seuil infini" peut être mise en œuvre en faisant que SwitchToSptDesired(S,G) retourne toujours faux. Une politique de "basculer sur le premier paquet" peut être mise en œuvre en faisant que SwitchToSptDesired(S,G) retourne vrai une fois qu'un seul paquet a été reçu pour la source et le groupe.

4.2.2 Établissement et suppression du bit SPT (S,G)

Le bit SPT (S,G) est utilisé pour distinguer si il faut transmettre sur l'état (*,G) ou l'état (S,G). Lorsque on bascule de l'arborescence de RP à l'arborescence de source, il y a une période de transition lorsque les données arrivent, due à l'état (*,G) amont pendant l'établissement de l'état (S,G), durant laquelle un routeur devrait continuer à ne transmettre que sur l'état (*,G).

Cela empêche des trous noirs temporaires qui seraient causés par l'envoi d'un Prune(S,G,rpt) avant que l'état (S,G) amont ait fini d'être établi.

Donc, lorsque un paquet arrive, le bit SPT (S,G) est mis à jour comme suit :

```
void
Update_SPTbit(S,G,iif) {
  si ( iif == RPF_interface(S)
       ET JoinDesired(S,G) == VRAI
       ET ( DirectlyConnected(S) == VRAI
            OU RPF_interface(S) != RPF_interface(RP(G))
            OU inherited_olist(S,G,rpt) == NUL
            OU ( ( RPF'(S,G) == RPF'(*,G) ) ET
                 ( RPF'(S,G) != NUL ) )
            OU ( je_suis_perdant_de_Assert(S,G,iif) ) ) ) {
    Régler le bit SPT(S,G) à VRAI
  }
}
```

De plus, un routeur peut régler le bit SPT(S,G) à VRAI dans d'autres cas, comme lorsque il reçoit une Assert(S,G) sur RPF_interface(S) (voir le paragraphe 4.6.1).

JoinDesired(S,G) est défini au paragraphe 4.5.5 et indique si on a l'état (S,G) Join approprié auquel envoyer un Join(S,G) en amont.

Fondamentalement, Update_SPTbit(S,G,iif) va établir le bit SPT si on a l'état (S,G) Join approprié, et si le paquet est arrivé sur l'interface amont correcte pour S, et si une ou plusieurs des conditions suivantes s'appliquent :

1. La source est directement connectée, auquel cas la bascule sur la SPT n'a pas de sens.

2. L'interface RPF pour S est différente de l'interface RPF au RP. Le paquet est arrivé sur RPF_interface(S), et donc la SPT doit avoir été achevée.

3. Personne ne veut le paquet sur l'arborescence de RP.

4. RPF'(S,G) == RPF'(*,G). Dans ce cas, le routeur ne va jamais être capable de dire si la SPT a été achevée, de sorte qu'il devrait juste basculer immédiatement. La vérification de RPF'(S,G) != NUL assure que le bit SPT n'est établi que si le voisin RPF vers S est valide.

Dans le cas où l'interface RPF est la même pour le RP et pour S, mais RPF'(S,G) et RPF'(*,G) diffèrent, on attend une Assert(S,G), qui indique que le routeur amont avec l'état (S,G) estime que la SPT a été achevée. Cependant, le point (3) ci-dessus est nécessaire parce que il peut n'y avoir aucun état (*,G) pour déclencher l'arrivée d'une Assert(S,G).

Le bit SPT est supprimé dans l'automate à états (S,G) amont (voir au paragraphe 4.5.5) lorsque JoinDesired(S,G) devient FAUX.

4.3 Routeurs désignés (DR) et messages Hello

Un LAN à supports partagés comme Ethernet peut avoir plusieurs routeurs PIM-SM qui lui sont connectés. Un seul de ces routeurs, le DR, va agir au nom des hôtes directement connectés par rapport au protocole PIM-SM. Parce que la distinction entre interfaces de LAN et point à point peut parfois être brouillée, et parce que les routeurs peuvent aussi avoir des fonctionnalités d'hôte de diffusion groupée, la spécification PIM-SM ne fait pas de distinction entre les deux. Donc, l'élection du DR va se faire sur toutes les interfaces, de LAN ou autres.

L'élection du DR est effectuée en utilisant les messages Hello. Les messages Hello sont aussi le moyen par lequel a lieu la négociation des options dans PIM, de sorte que des fonctionnalités supplémentaires peuvent être activées, ou des paramètres être réglés.

4.3.1 Envoi des messages Hello

Les messages PIM Hello sont envoyés périodiquement à chaque interface à capacité PIM. Cela permet à un routeur de connaître les routeurs PIM voisins sur chaque interface. Les messages Hello sont aussi le mécanisme utilisé pour élire un DR, et négocier des capacités supplémentaires. Un routeur doit enregistrer les informations de Hello reçues de chaque voisin PIM.

Les messages Hello DOIVENT être envoyés sur toutes les interfaces actives, incluant les liaisons physiques point à point, et sont envoyés en diffusion groupée à l'adresse de groupe "TOUS-LES-ROUTEURS-PIM" ('224.0.0.13' pour IPv4 et 'ff02::d' pour IPv6).

On note que certaines mises en œuvre n'envoient pas de message Hello sur les interfaces point à point. C'est un comportement non conforme. Un routeur PIM conforme DOIT envoyer des messages Hello, même sur les interfaces point à point.

Un temporisateur Hello (HT(I)) par interface est utilisé pour déclencher l'envoi des messages Hello sur chaque interface active. Lorsque PIM est activé sur une interface ou lorsque un routeur démarre pour la première fois, le temporisateur Hello de cette interface est réglé à une valeur aléatoire entre 0 et Délai_Hello_Déclenché. Cela empêche la synchronisation des messages Hello si plusieurs routeurs sont mis sous tension simultanément. Après l'intervalle initial aléatoire, les messages Hello DOIVENT être envoyés toutes les Hello_Period secondes. Le temporisateur Hello NE DEVRAIT PAS être réinitialisé sauf lorsque il arrive à expiration.

Noter que les voisins ne vont pas accepter de messages Join/Prune ou Assert d'un routeur sauf si ils ont déjà entendu un message Hello de ce routeur. Donc, si un routeur a besoin d'envoyer un message Join/Prune ou Assert sur une interface sur laquelle il n'a pas encore envoyé un message Hello avec l'adresse IP actuellement configurée, il DOIT alors immédiatement envoyer le message Hello pertinent sans attendre que le temporisateur Hello arrive à expiration, suivi par le message Join/Prune ou Assert.

L'option Priorité de DR permet à un administrateur de réseau de donner la préférence à un routeur particulier dans le processus d'élection de DR en lui donnant une priorité de DR numériquement supérieure. L'option Priorité de DR DEVRAIT être incluse dans tout message Hello, même si aucune priorité de DR n'est explicitement configurée sur cette interface. Ceci est nécessaire parce que l'élection de DR fondée sur la priorité n'est activée que lorsque tous les voisins sur une interface annoncent qu'ils sont capables d'utiliser l'option Priorité de DR. La priorité par défaut est 1.

L'option Identifiant de génération (GenID, *Generation Identifier*) DEVRAIT être incluse dans tous les messages Hello. L'option GenID contient une valeur de 32 bits générée au hasard qui est régénérée chaque fois qu'une transmission PIM est lancée ou relancée sur l'interface, y compris lorsque le routeur lui-même redémarre. Lorsque un message Hello avec un nouveau GenID est reçu d'un voisin, toutes les vieilles informations de Hello sur ce voisin DEVRAIENT être éliminées et remplacées par les informations provenant du nouveau message Hello. Cela peut causer le choix d'un nouveau DR sur cette interface.

L'option ^Délai d'élagage de LAN DEVRAIT être incluse dans tous les messages Hello envoyés sur des LAN multi accès. Cette option annonce la capacité d'un routeur à utiliser des valeurs autres que par défaut pour le délai de propagation et l'intervalle d'outrepassement, qui affectent le réglage des temporisateurs Élagage en cours *(Prune-Pending)*, jonction amont *(Upstream Join)*, et Outrepassement *(Override)* (définis au paragraphe 4.10).

L'option Liste d'adresses annonce toutes les adresses secondaires associées à l'interface de source du routeur qui génère le message. L'option DOIT être incluse dans tous les messages Hello si il y a des adresses secondaires associées à l'interface de source et PEUT être omise si il n'existe aucune adresse secondaire.

Pour permettre que des routeurs nouveaux ou qui réamorcent apprennent rapidement les voisins PIM, lorsque un message Hello est reçu d'un nouveau voisin, ou qu'un message Hello avec un nouveau GenID est reçu d'un voisin existant, un nouveau message Hello DEVRAIT être envoyé sur cette interface après un délai aléatoire entre 0 et Délai_Hello_Déclenché. Ce message déclenché n'a pas besoin de changer le rythme des messages périodiques programmés. Si un routeur a besoin d'envoyer un Join/Prune au nouveau voisin ou d'envoyer un message Assert en réponse à un message Assert provenant du nouveau voisin avant l'expiration de ce délai aléatoire, il DOIT alors envoyer immédiatement le message Hello pertinent sans attendre l'expiration du temporisateur Hello, suivi par le message Join/Prune ou Assert. Si il ne fait pas cela, le nouveau voisin va alors éliminer le message Join/Prune ou Assert.

Avant qu'une interface s'arrête ou change d'adresse principale IP, un message Hello avec un temps de garde *(HoldTime)* de zéro DEVRAIT être envoyé immédiatement (avec l'ancienne adresse IP si l'adresse IP a changée). Cela va amener les voisins PIM à supprimer ce voisin (ou sa vieille adresse IP) immédiatement. Après qu'une interface a changé son adresse IP, elle DOIT envoyer un message Hello avec sa nouvelle adresse IP. Si une interface change une de ses adresses IP secondaires, un message Hello avec une option Liste d'adresses mise à jour et un temps de garde différent de zéro DEVRAIT être envoyé immédiatement. Cela va amener les voisins PIM à mettre à jour cette liste d'adresses secondaires du voisin immédiatement.

4.3.2 Élection de DR

Lorsque un message Hello PIM est reçu sur l'interface I, les informations suivantes sur le voisin envoyeur sont enregistrées :

neighbor.interface : l'interface sur laquelle le message Hello est arrivé.

neighbor.primary_ip_address : adresse IP que le voisin PIM a utilisé comme adresse de source du message Hello.

neighbor.genid : identifiant de génération du voisin PIM.

neighbor.dr_priority : champ Priorité de DR du voisin PIM, si il est présent dans le message Hello.

neighbor.dr_priority_present : fanion indiquant si le champ Priorité de DR était présent dans le message Hello.

neighbor.timeout : valeur de temporisateur pour marquer la fin de l'état du voisin quand il devient périmé, aussi appelé le temporisateur de vie de voisin (NLT, *Neighbor Liveness Timer*).

Le temporisateur de vie de voisin (NLT(N,I)) est remis à Hello_Holdtime (à partir de l'option Hello Holdtime) chaque fois qu'un message Hello est reçu qui contient une option Holdtime, ou à Default_Hello_Holdtime si le message Hello ne contient pas l'option Holdtime.

L'état de voisin est supprimé lorsque le temporisateur de voisin arrive à expiration.

La fonction pour calculer le DR sur l'interface I est :

```
 hôte
 DR(I) {
    dr = moi
    pour chaque voisin sur l'interface I {
      si ( dr_est_meilleur( voisin, dr, I ) == VRAI ) {
          dr = voisin
       }
    }
    retourner dr
 }
```

La fonction utilisée pour comparer les "métriques" de DR sur l'interface I est :

```
  bool
  dr_est_meilleur(a,b,I) {
    si( il y a un voisin n sur I pour lequel n.dr_priorité_présente est faux ) {
          retourner a.principale_ip_adresse > b.principale_ip_addresse
    } autrement {
       retourner ( a.dr_priorité > b.dr_priorité ) OU
            ( a.dr_priorité == b.dr_priorité ET
              a.principale_ip_adresse > b.principale_ip_adresse )
    }
  }
```

La fonction triviale Je_suis_DR(I) est définie pour faciliter la lisibilité :

```
bool
  Je_suis_DR(I) {
    retourner DR(I) == moi
  }
```

La Priorité de DR est un nombre non signé de 32 bits, et la priorité numériquement la plus forte est toujours préférée. L'idée d'un routeur du DR en cours sur une interface peut changer lorsque un message Hello PIM est reçu, lorsque un voisin arrive en fin de temporisation, ou lorsque la propre priorité de DR d'un routeur change. Si le routeur devient le DR ou cesse d'être le DR, cela va normalement causer le changement de l'état de l'automate à états du DR Register. Les actions suivantes sont déterminées par cet automate à états.

On note que certaines mises en œuvre de PIM n'envoient pas de message Hello sur les interfaces point à point et donc ne peuvent pas effectuer l'élection de DR sur de telles interfaces. Ce comportement n'est pas conforme. L'élection de DR DOIT être effectuée sur TOUTES les interfaces PIM-SM actives.

4.3.3 Réduction du délai de propagation d'élagage sur les LAN

En plus des informations enregistrées pour l'élection de DR, les informations suivantes par voisin sont obtenues de l'option Délai d'élagage de Hello de LAN :

neighbor.lan_prune_delay_present : fanion indiquant si l'option Délai d'élagage de LAN était présente dans le message Hello.

neighbor.tracking_support : fanion qui mémorise la valeur du bit T dans l'option Délai d'élagage de LAN si elle est présente dans le message Hello. Cela indique la capacité du voisin à désactiver la suppression de message Join.

neighbor.propagation_delay : champ Délai de propagation de l'option Délai d'élagage de LAN (si elle est présente) dans le message Hello.

neighbor.override_interval : champ Override_Interval de l'option Délai d'élagage de LAN (si elle est présente) dans le message Hello.

L'état supplémentaire décrit ci-dessus est supprimé avec l'état de voisin de DR lorsque le temporisateur de voisin expire.

Tout comme pour l'option Priorité de DR, les informations fournies dans l'option Délai d'élagage de lAN ne sont utilisées que si tous les voisins sur une liaison annoncent l'option. La fonction ci-dessous calcule cet état :

```
bool
lan_delay_enabled(I) {
   pour chaque voisin sur l'interface I {
      si ( neighbor.lan_prune_delay_present == faux ) {
         retourner faux
      }
   }
   retourner vrai
}
```

Le délai de propagation inséré par un routeur dans l'option Délai d'élagage de LAN exprime le délai attendu de propagation de message sur la liaison et DEVRAIT être configurable par l'administrateur de système. Il est utilisé par les routeurs amont pour représenter le temps qu'ils devraient attendre un message Outrepassement de Join avant d'élaguer une interface.

Les mises en œuvre de PIM DEVRAIENT appliquer une limite inférieure aux valeurs permises pour ce délai pour permettre de programmer et traiter les délais dans leur routeur. De tels délais peuvent causer un retard de traitement des messages reçus ainsi qu'un retard non prévu de l'envoi des messages déclenchés. Régler ce délai de propagation à une valeur trop faible peut résulter en des pannes temporaires de transmission parce qu'un routeur aval ne sera pas capable d'outrepasser le message Prune d'un voisin avant que le voisin amont cesse de transmettre.

Lorsque tous les routeurs sur une liaison sont en position de négocier un délai de propagation différent de celui par défaut, on choisit la plus grande valeur parmi celles annoncées par chaque voisin. La fonction de calcul du délai de propagation effectif de l'interface I est :

```
time_interval
Effective_Propagation_Delay(I) {
   si ( lan_delay_activé(I) == faux ) {
      retourner Propagation_delay_défaut
   }
   delay = Propagation_Delay(I)
   pour chaque voisin sur l'interface I {
      si ( neighbor.propagation_delay > delay ) {
         delay = neighbor.propagation_delay
      }
   }
   retourner delay
```

```
}
```

Pour éviter la synchronisation des messages d'outrepassement lorsque plusieurs routeurs vers l'aval partagent une liaison multi accès, l'envoi de tels messages est retardé d'un petit délai aléatoire. La période aléatoire devrait représenter la taille de la population de routeurs PIM sur la liaison. Chaque routeur exprime sa vision de la quantité de délai aléatoire nécessaire dans le champ Intervalle d'outrepassement de l'option Délai d'élagage de LAN.

Lorsque tous les routeurs d'une liaison sont en position de négocier un intervalle d'outrepassement différent de celui par défaut, on choisit la plus grande valeur parmi celles annoncées par chaque voisin. La fonction de calcul de l'intervalle d'outrepassement effectif de l'interface I est :

```
time_interval
Effective_Override_Interval(I) {
   si ( lan_delay_enabled(I) == faux ) {
      retourner t_override_default
   }
   delay = Override_Interval(I)
   pour chaque voisin sur l'interface I {
      si ( neighbor.override_interval > delay ) {
         delay = neighbor.override_interval
      }
   }
   retourner delay
}
```

Bien que les mécanismes n'en soient pas spécifiés dans le présent document, il est possible aux routeurs amont de tracer explicitement l'adhésion des routeurs aval individuels si la suppression de Join est désactivée. Un routeur peut annoncer sa volonté de désactiver la suppression de Join en utilisant le bit T dans l'option Hello de Délai d'élagage. Sauf si tous les routeurs PIM sur une liaison négocient cette capacité, le traçage explicite et la désactivation du mécanisme de suppression de Join ne sont pas possibles. La fonction de calcul de l'état de Suppression sur l'interface I est :

```
bool
Suppression_Enabled(I) {
   si ( lan_delay_enabled(I) == faux ) {
      retourner vrai
   }
   pour chaque voisin sur l'interface I {
      si ( neighbor.tracking_support == faux ) {
         retourner vrai
      }
   }
   retourner faux
}
```

Noter que le réglage de Suppression_Enabled(I) affecte la valeur de t_suppressed (voir le paragraphe 4.11).

4.3.4 Maintenance des listes d'adresses secondaires

La communication des adresses secondaires d'interface d'un routeur à ses voisins PIM est nécessaire pour fournir aux voisins un mécanisme pour transposer les informations de prochain bond obtenues de leur MRIB en une adresse principale qui puisse être utilisée comme destination pour les messages Join/Prune. La transposition est effectuée au moyen de la macro NBR. L'adresse principale d'un voisin PIM est obtenue de l'adresse IP de source utilisée dans ses messages PIM Hello. Les adresses secondaires sont portées dans le message Hello dans une option Hello Address List. L'adresse principale de l'interface de source du routeur NE DOIT PAS figurer sur la liste dans l'option Liste d'adresses de Hello.

En plus des informations enregistrées pour l'élection de DR, les informations par voisin suivantes sont obtenues de l'option Liste d'adresses de Hello :

neighbor.secondary_address_list : liste des adresses secondaires utilisées par le voisin PIM sur l'interface à travers laquelle le message Hello a été transmis.

Lors du traitement d'un message Hello PIM reçu qui contient une option Liste d'adresses de Hello, la liste des adresses secondaires dans le message remplace complètement toutes les adresses secondaires précédemment associées pour ce voisin. Si un message Hello PIM reçu ne contient pas d'option Liste d'adresses de Hello, toutes les adresses secondaires associées au voisin DOIVENT alors être supprimées. Si un message Hello PIM reçu contient une option Liste d'adresses de Hello qui comporte l'adresse principale du routeur d'envoi dans la liste des adresses secondaires (bien que ce ne soit pas attendu) les adresses énumérées dans le message, à l'exclusion de l'adresse principale, sont alors utilisées pour mettre à jour les adresses secondaires associées pour ce voisin.

Toutes les adresses secondaires annoncées dans les messages Hello reçus doivent être confrontées à celles annoncées précédemment par tous les autres voisins PIM sur cette interface. Si il y a un conflit et si les mêmes adresses secondaires ont été annoncées précédemment par un autre voisin, seule la transposition des plus récentes reçues DOIT être conservée, et un message d'erreur DEVRAIT être enregistré pour l'administrateur en débit limité.

Au sein d'une option Liste d'adresses de Hello, toutes les adresses DOIVENT être de la même famille d'adresses. Il n'est pas permis de mélanger des adresses IPv4 et IPv6 au sein du même message. De plus, la famille d'adresses des champs du message DEVRAIT être la même que l'adresse IP de source et de destination de l'en-tête du paquet.

4.4 Messages PIM Register

Le routeur désigné (DR) sur un LAN ou une liaison point à point encapsule des paquets en diffusion groupée provenant de sources locales au RP pour le groupe pertinent sauf si il a récemment reçu un message Register-Stop pour ce (S,G) ou (*,G) du RP. Lorsque le DR reçoit un message Register-Stop provenant du RP, il lance un temporisateur Register-Stop pour conserver cet état. Juste avant l'arrivée à expiration du temporisateur Register-Stop, le DR envoie un message Null-Register au RP pour permettre au RP de rafraîchir les informations de Register-Stop au DR. Si le temporisateur Register-Stop arrive en fait à expiration, le DR va reprendre l'encapsulation des paquets provenant de la source pour le RP.

4.4.1 Envoi des messages Register provenant du DR

Chaque routeur PIM-SM a la capacité d'être un DR. L'automate à états ci-dessous est utilisé pour mettre en œuvre la fonction Register. Pour les besoins de la spécification, on représente le mécanisme d'encapsulation des paquets pour le RP comme une interface Register-Tunnel, qui est ajoutée ou retirée à la olist (S,G). L'interface de tunnel prend alors part aux règles normales de transmission de paquet comme spécifié au paragraphe 4.2.

Si l'état Register est conservé, il ne l'est que pour les sources directement connectées et il est par (S,G). Il y a quatre états dans l'automate à états Register par (S,G) du DR :

Join (J) ! Le tunnel Register est "joint" (le joint est en fait implicite, mais le DR agit comme si le RP avait joint le DR sur l'interface tunnel).

Prune (P) : Le tunnel Register est "élagué" (cela arrive lorsque un Register-Stop est reçu).

Join-Pending (JP) : Le tunnel Register est élagué mais le DR envisage de le rajouter.

NoInfo (NI) : Pas d'informations. C'est l'état initial, et l'état lorsque le routeur n'est pas le DR.

De plus, un temporisateur d'arrêt d'enregistrement (RST, *Register-Stop Timer*) est gardé si l'automate à états n'est pas dans l'état NoInfo.

Figure 1 : Automate à états Register par (S,G) chez le DR

État précédent	Événement				
	Expiration du RST	Pourrait enregistrer -->Vrai	Pourrait enregistrer -->Faux	Register-Stop reçu	RP changé
NoInfo (NI)	-	->état J ajout tunnel Reg	-	-	-
Join (J)	-	-	->état NI supprime tunnel reg	-> état P supprime tunnel reg ; établit RST(*)	-> état J met à jour tunnel reg
Join-Pending (JP)	-> état J ajout tunnel reg	-	-> état NI	-> état P établit RST(*)	-> état J ajout tunnel reg ; annule RST
Prune (P)	-> état JP établit RST(**) ; envoie Null-Register	-	-> état NI	-	-> état J. Ajoute tunnel register ; annule RST

Notes :
(*) Le temporisateur Register-Stop est réglé à une valeur aléatoire choisie uniformément dans l'intervalle (0,5 * Register_Suppression_Time, 1,5 * Register_Suppression_Time) moins Register_Probe_Time. La soustraction de Register_Probe_Time est un peu inutile parce qu'il est vraiment petit par rapport à Register_Suppression_Time, mais c'était dans la vieille spécification et on l'a gardé pour la compatibilité.
(**) Le temporisateur Register-Stop est réglé à Register_Probe_Time.

Les trois actions suivantes sont définies :

Add Register Tunnel : Une interface virtuelle Register-Tunnel, VI, est créée (si elle n'existe pas déjà) avec sa cible d'encapsulation RP(G). DownstreamJPState(S,G,VI) est réglé à l'état Join, ce qui cause l'ajout de l'interface tunnel à immediate_olist(S,G) et inherited_olist(S,G).

Remove Register Tunnel : VI est l'interface Register-Tunnel virtuelle avec la cible d'encapsulation de RP(G). DownstreamJPState(S,G,VI) est réglé à l'état NoInfo, ce qui cause la suppression de l'interface tunnel de immediate_olist(S,G) et inherited_olist(S,G). Si DownstreamJPState(S,G,VI) est NoInfo pour tous les (S,G), alors VI peut être supprimée.

Update Register Tunnel : Cette action survient lorsque RP(G) change.

VI_old est l'interface Register-Tunnel virtuelle avec la cible d'encapsulation old_RP(G). Une interface virtuelle Register-Tunnel, VI_new, est créée (si elle n'existe pas déjà) avec sa cible d'encapsulation new_RP(G). DownstreamJPState(S,G,VI_old) est réglé à l'état NoInfo, et DownstreamJPState(S,G,VI_new) est réglé à l'état Join. Si DownstreamJPState(S,G,VI_old) est NoInfo pour toutes les (S,G), alors VI_old peut être supprimée.

Noter qu'on ne peut pas simplement changer la cible d'encapsulation de VI_old parce que tous les groupes qui utilisent ce tunnel d'encapsulation ne seront pas déplacés sur le même nouveau RP.

CouldRegister(S,G) : La macro "CouldRegister" dans l'automate à états est définie comme :

```
 bool CouldRegister(S,G) {
   retourner ( Je_suis_DR( RPF_interface(S) ) ET
         KeepaliveTimer(S,G) fonctionne ET
         DirectlyConnected(S) == VRAI )
 }
```

Noter qu'à réception d'un paquet au DR d'une source directement connectée, KeepaliveTimer(S,G) doit être établi par les règles de transmission de paquet avant de calculer CouldRegister(S,G) dans l'automate à états Register, ou le premier paquet provenant d'une source ne sera pas enregistré.

Encapsulation des paquets de données dans le tunnel Register
Conceptuellement, le tunnel Register est une interface avec une plus petite MTU que l'interface IP sous-jacente vers le RP. La fragmentation IP sur les paquets transmis sur le tunnel Register est effectuée sur la base de cette plus petite MTU. Le DR encapsulant peut effectuer la découverte de la MTU du chemin sur le RP pour déterminer la MTU effective du tunnel. La fragmentation pour la plus petite MTU devrait prendre en compte à la fois la redondance d'en-tête IP externe et de l'en-tête PIM Register. Si un paquet de diffusion groupée est fragmenté sur le chemin dans le tunnel Register, chaque fragment est encapsulé individuellement afin qu'il contienne les en-têtes IP, PIM, et l'en-tête IP interne.

Dans IPv6, le DR DOIT effectuer la découverte de la MTU du chemin, et un message ICMP Paquet trop gros DOIT être envoyé par le DR encapsulant si il reçoit un paquet qui ne va pas tenir dans la MTU effective du tunnel. Si la MTU entre le DR et le RP résulte en une MTU effective de tunnel plus petite que 1280 (MTU IPv6 minimum) le DR DOIT envoyer des messages Fragmentation exigée avec une valeur de MTU de 1280 et DOIT fragmenter ses messages PIM Register comme exigé, en utilisant un en-tête de fragmentation IPv6 entre l'en-tête externe IPv6 et l'en-tête PIM Register.

Le TTL d'un paquet de données transmis est décrémenté avant son encapsulation dans le tunnel Register. Le paquet encapsulant utilise le TTL normal que le routeur aurait utilisé pour tout paquet IP généré localement.

Les bits de notification explicite d'encombrement (ECN, *Explicit Congestion Notification*) IP devraient être copiés du paquet d'origine à l'en-tête IP du paquet encapsulant. Ils NE DEVRAIENT PAS être réglés indépendamment par le routeur encapsulant.

Le codet Diffserv (DSCP, *DiffServ Code Point*) devrait être copié du paquet d'origine dans l'en-tête IP du paquet encapsulant. Il PEUT être réglé indépendamment par le routeur encapsulant, sur la base de la configuration statique ou de la classification du trafic. Voir dans la [RFC2983] l'exposé sur le réglage du DSCP sur les tunnels.

Traitement des messages Register-Stop(*,G) au DR
Un ancien RP pourrait envoyer un message Register-Stop avec l'adresse de source réglée tout à zéro. C'était la façon d'agir normale dans la RFC2362 lorsque le message Register correspondait à l'état (*,G) au RP, et il était défini comme signifiant "arrêter d'encapsuler toutes les sources pour ce groupe". Cependant, le comportement d'un tel Register-Stop(*,G) est ambigu ou incorrect dans certaines circonstances.

On spécifie qu'un RP ne devrait pas envoyer de messages Register-Stop(*,G) mais pour la compatibilité, un DR devrait être capable d'en accepter un si il est reçu.

Un Register-Stop(*,G) devrait être traité comme un Register-Stop(S,G) pour tous les automates à états Register (S,G) qui ne sont pas dans l'état NoInfo. Un routeur ne devrait pas appliquer un Register-Stop(*,G) aux sources qui deviennent actives après la réception du Register-Stop(*,G).

4.4.2 Réception des messages Register au RP

Lorsque un RP reçoit un message Register, la suite des actions est décidée conformément au pseudo code suivant :

```
paquet_arrive_sur_rp_tunnel( pkt ) {
   si( outer.dst n'est pas une de mes adresses ) {
      éliminer le paquet en silence.          # Note : Ce peut être une tentative frauduleuse.
   }
   si( Je_suis_RP(G) ET outer.dst == RP(G) ) {
        sentRegisterStop = FAUX;
        si ( SPTbit(S,G) OU ( SwitchToSptDésiré(S,G) ET ( inherited_olist(S,G) == NUL ))) {
          envoyer Register-Stop(S,G) à outer.src
          sentRegisterStop = VRAI;
        }
        si ( SPTbit(S,G) OU SwitchToSptDésiré(S,G) ) {
            si ( sentRegisterStop == VRAI ) {
                régler KeepaliveTimer(S,G) à RP_Keepalive_Period ;
            } autrement {
                régler KeepaliveTimer(S,G) à Keepalive_Period ;
            }
        }
        si( !SPTbit(S,G) ET ! pkt.NullRegisterBit ) {
            désencapsuler et transmettre le paquet interne à inherited_olist(S,G,rpt)      # Note (+)
        }
   } autrement {
      envoyer Register-Stop(S,G) à outer.src             # Note (*)
   }
}
```

outer.dst est l'adresse de destination IP de l'en-tête encapsulant.
outer.src est l'adresse IP de source de l'en-tête encapsulant, c'est-à-dire, l'adresse du DR.
Je_suis_RP(G) est vrai si la transposition de groupe en RP indique que ce routeur est le RP pour le groupe.

Note (*) : Cela peut bloquer le trafic de S pour Register_Suppression_Time si le DR a appris une nouvelle transposition de groupe en RP avant le RP. Cependant, cela n'a d'importance que si on imagine un moyen pour que le RP accepte aussi les jonctions (*,G) lorsque il n'a pas encore réalisé qu'il est sur le point de devenir le RP pour G. Cela sera réglé une fois que le RP aura appris la nouvelle transposition de groupe en RP. On a décidé de ne rien faire là dessus et juste d'accepter le fait que PIM peut subir une interruption de connexité (*,G) à la suite du changement de RP.

Note (+) : Les mises en œuvre NE DEVRAIENT PAS faire de cela un cas particulier, mais DEVRAIENT faire que ce chemin rejoigne le chemin de transmission normal de paquet. Toutes les actions appropriées découlant du pseudo code "À réception de données de S à G sur l'interface iif" au paragraphe 4.2 devraient être effectuées.

Le temporisateur KeepaliveTimer(S,G) est redémarré au RP lorsque les paquets arrivent sur l'interface tunnel approprié et que le RP désire passer à la SPT ou lorsque le bit SPT est déjà établi. Cela peut causer le déclenchement d'une jonction par l'automate à états (S,G) amont si la inherited_olist(S,G) n'est pas NUL.

Un RP devrait préserver l'état (S,G) qui a été créé en réponse à un message Register pendant au moins (3 * Register_Suppression_Time) ; autrement, le RP peut arrêter la jonction (S,G) avant que le DR pour S ait recommencé d'envoyer des Register. Le trafic serait alors interrompu jusqu'à l'arrivée à expiration du temporisateur Register-Stop au DR.

Donc, au RP, KeepaliveTimer(S,G) devrait être redémarré à (3 * Register_Suppression_Time + Register_Probe_Time).

Lors de la transmission d'un paquet du tunnel Register, le TTL du paquet de données d'origine est décrémenté après qu'il est désencapsulé.

Les bits ECN IP devraient être copiés de l'en-tête IP du paquet Register au paquet désencapsulé.

Le DSCP devrait être copié de l'en-tête IP du paquet Register au paquet désencapsulé. Le RP PEUT conserver le DSCP du paquet interne ou reclassifier le paquet et appliquer un DSCP différent. Les scénarios où chacune de ces solutions pourrait être utile sont discutés dans la [RFC2983].

4.5 Messages PIM Join/Prune

Un message PIM Join/Prune consiste en une liste de groupes et une liste de sources jointes et élaguées pour chaque groupe. Lors du traitement d'un message Join/Prune reçu, chaque source jointe ou élaguée pour un groupe est effectivement considérée individuellement, et s'applique à un ou plusieurs des automates à états suivants. Lorsque on examine un message Join/Prune dont le champ Adresse de voisin amont vise ce routeur, les Join et Prune (*,G) peuvent affecter à la fois l'automate à états (*,G) et (S,G,rpt) vers l'aval, tandis que les Join et Prune (S,G), et (S,G,rpt) peuvent seulement affecter leurs automates à états avals respectifs. Lorsque on examine un message Join/Prune dont le champ Adresse de voisin amont vise un autre routeur, la plupart des messages Join ou Prune pourraient affecter chaque automate à états vers l'amont.

En général, un message PIM Join/Prune ne devrait être accepté pour traitement que si il vient d'un voisin PIM connu. Un routeur PIM entend parler des voisins PIM par les messages PIM Hello. Si un routeur reçoit un message Join/Prune d'une certaine adresse IP de source et qu'il n'a pas vu de message Hello PIM provenant de cette adresse de source, le message Join/Prune DEVRAIT alors être éliminé sans autre traitement. De plus, si le message Hello provenant d'un voisin a été authentifié (voir au paragraphe 6.3) alors tous les messages Join/Prune provenant de ce voisin DOIVENT aussi être authentifiés.

On note que certaines vieilles mises en œuvre PIM échouent de façon incorrecte à envoyer des messages Hello sur les interfaces point à point, aussi on RECOMMANDE qu'une option de configuration soit fournie pour permettre l'interopération avec ces vieux routeurs, mais cette option de configuration NE DEVRAIT PAS être activée par défaut.

4.5.1 Réception des messages Join/Prune (*,G)

Lorsque un routeur reçoit un Join(*,G), il doit d'abord vérifier si le RP dans le message correspond à RP(G) (l'idée du routeur de qui est le RP). Si le RP dans le message ne correspond pas au RP(G), le Join(*,G) devrait être éliminé en silence. (Noter que d'autres entrées de liste de source, comme (S,G,rpt) ou (S,G) dans le même ensemble spécifique de groupe, devraient quand même être traitées.) Si un routeur n'a pas d'informations de RP (par exemple, il n'a pas reçu de message BSR récent) il peut alors choisir d'accepter le Join(*,G) et traiter le RP dans le message comme RP(G). Les messages Prune(*,G) reçus sont traités même si le RP dans le message ne correspond pas au RP(G).

L'automate à états par interface pour recevoir les messages (*,G) est donné ci-dessous. Il y a trois états :

NoInfo (NI) : L'interface n'a pas d'état Join (*,G) et pas de temporisateur en cours.

Join (J) : L'interface a l'état Join (*,G) qui va causer la transmission par le routeur des paquets destinés à G provenant de cette interface excepté si il y a aussi des informations de Prune (S,G,rpt) (voir au paragraphe 4.5.3) ou si le routeur a perdu une assertion sur cette interface.

Prune-Pending (PP) : Le routeur a reçu un Prune(*,G) sur cette interface d'un voisin aval et il attend pour voir si l'élagage sera outrepassé par un autre routeur aval. Pour les besoins de la transmission, l'état Prune-Pending fonctionne exactement comme l'état Join.

De plus, l'automate à états utilise deux temporisateurs :

Temporisateur d'expiration (ET, *Expiry Timer*) : ce temporisateur est redémarré lorsque un Join(*,G) valide est reçu. L'expiration du temporisateur d'expiration cause le retour de l'état de l'interface à NoInfo pour ce groupe.

Temporisation d'élagage en cours (PPT, *Prune-Pending Timer*) : ce temporisateur est lancé lorsque un Prune(*,G) valide est reçu. L'expiration du temporisateur d'élagage en cours cause le retour de l'état de l'interface à NoInfo pour ce groupe.

Figure 2 : Automate à états aval par interface (*,G)

État précédent	Événement			
	Reçoit Join(*,G)	Reçoit Prune(*,G)	PPTexpire	ET expire
NoInfo (NI)	-> État J :lance ET	-> État NI	-	-
Join (J)	-> État J : relance ET	-> État PP :lance PPT	-	-> État NI|
Prune-Pending (PP)	-> État J :relance ET	-> État PP : envoie Prune-Echo(*,G)	-> État NI |	-> État NI

Les événements de transition "Reçoit Join(*,G)" et "Reçoit Prune(*,G)" impliquent de recevoir un Join ou Prune ciblé sur l'adresse IP principale de ce routeur sur l'interface reçue. Si le champ Adresse de voisin amont n'est pas correcte, ces transitions d'état dans cet automate NE DOIVENT PAS se produire, bien que voir un tel paquet puisse causer des transitions d'état dans d'autres automates à états.

Sur les interfaces non numérotées sur les liaisons point à point, l'adresse du routeur devrait être la même que l'adresse de source qu'il a choisie pour le message Hello qu'il a envoyé sur cette interface. Cependant, sur les liaisons point à point, il est RECOMMANDÉ pour la rétro compatibilité que les messages PIM Join/Prune avec un champ Adresse de voisin amont tout à zéro soient aussi acceptés.

Transitions de l'état NoInfo

Dans l'état NoInfo, les événements suivants peuvent déclencher une transition :

Receive Join(*,G) : un Join(*,G) est reçu sur l'interface I avec son Adresse de voisin amont réglée à l'adresse IP principale du routeur sur I.

L'automate à états (*,G) aval sur l'interface I passe à l'état Join. Le temporisateur d'expiration (ET) est lancé et réglé à HoldTime *(temps de garde)* à partir du message Join/Prune déclencheur.

Transitions de l'état Join

Dans l'état Join, les événements suivants peuvent déclencher une transition :

Reçoit Join(*,G) : un Join(*,G) est reçu sur l'interface I avec son adresse de voisin amont réglée à l'adresse IP principale du routeur sur I. L'automate à états (*,G) aval sur l'interface I reste dans l'état Join, et le temporisateur d'expiration (ET) est relancé. Le ET est réglé au maximum de sa valeur courante et du HoldTime provenant du message Join/Prune déclencheur.

Reçoit Prune(*,G) : un Prune(*,G) est reçu sur l'interface I avec son adresse de voisin amont réglée à l'adresse IP principale du routeur sur I. L'automate à états (*,G) aval sur l'interface I passe à l'état Prune-Pending. Le temporisateur Prune-Pending est lancé. Il est réglé à J/P_Override_Interval(I) si le routeur a plus d'un voisin sur cette interface ; autrement, il est réglé à zéro, causant son expiration immédiate.

Expiry Timer Expire : le temporisateur d'expiration pour l'automate à états (*,G) aval sur l'interface I arrive à expiration. L'automate à états (*,G) aval sur l'interface I passe à l'état NoInfo.

Transitions de l'état Prune-Pending

Dans l'état Prune-Pending, les événements suivants peuvent déclencher une transition :

Reçoit Join(*,G) : un Join(*,G) est reçu sur l'interface I avec son adresse de voisin amont réglée à l'adresse IP principale du routeur sur I. L'automate à états (*,G) aval sur l'interface I passe à l'état Join. Le temporisateur Prune-Pending est annulé (sans déclencher d'événement d'expiration). Le temporisateur d'expiration (ET) est redémarré et est alors réglé au maximum de sa valeur courante et de HoldTime provenant du message Join/Prune déclencheur.

Expiry Timer Expire : le temporisateur d'expiration pour l'automate à états (*,G) sur l'interface I expire. L'automate à états (*,G) aval sur l'interface I passe à l'état NoInfo.

Prune-Pending Timer expire : le temporisateur Élagage en cours pour l'automate à états (*,G) aval sur l'interface I expire. L'automate à états (*,G) aval sur l'interface I passe à l'état NoInfo. Un PruneEcho(*,G) est envoyé au sous réseau connecté à l'interface I.

L'action "Envoie PruneEcho(*,G)" est déclenchée lorsque le routeur arrête de transmettre sur une interface par suite d'un élagage. Un PruneEcho(*,G) est simplement un message Prune(*,G) envoyé par le routeur amont sur un LAN avec sa propre adresse dans le champ Adresse de voisin amont. Son objet est d'ajouter de la fiabilité afin que si un Prune qui aurait dû être outrepassé par un autre routeur est perdu localement sur le LAN, le PruneEcho puisse être reçu et causer la réalisation de l'outrepassement. Un PruneEcho(*,G) n'a pas besoin d'être envoyé sur une interface qui ne contient qu'un seul voisin PIM pendant que cet automate à états est dans l'état Prune-Pending.

4.5.2 Réception des messages (S,G) Join/Prune

L'automate à états par interface pour recevoir des messages Join/Prune (S,G) est montré ci-dessous et est presque identique à celui pour les messages (*,G). Il y a trois états :

NoInfo (NI) : l'interface n'a pas d'état Join (S,G) et pas de temporisateur (S,G) qui fonctionne.

Join (J) : l'interface a l'état Join (S,G), qui va causer la transmission par le routeur des paquets de S destinés à G à partir de cette interface si l'état (S,G) est actif (le bit SPT est établi) sauf si le routeur a perdu une assertion sur cette interface.

Prune-Pending (PP) : le routeur a reçu un Prune(S,G) sur cette interface d'un voisin en aval et attend pour voir si l'élagage sera outrepassé par un autre routeur vers l'aval. Pour les besoins de la transmission, l'état Prune-Pending fonctionne exactement comme l'état Join.

De plus, il y a deux temporisateurs :

Temporisateur d'expiration (ET) : ce temporisateur est établi lorsque est reçu un Join(S,G) valide. L'arrivée à expiration de Expiry Timer cause le retour de cet automate à états à l'état NoInfo.

Temporisateur d'élagage en cours (PPT) : ce temporisateur est établi lorsque est reçu un Prune(S,G) valide. L'arrivée à expiration du temporisateur d'élagage en cours cause le retour de cet automate à états à l'état NoInfo.

Figure 3 : Automate à états par interface (S,G) vers l'aval

	Événement			
État antérieur	Reçoit Join(S,G)	Reçoit Prune(S,G)	Expiration du PPT	Expiration du ET
NoInfo (NI)	-> L'état J : lance le ET	-> État NI	-	-
Join (J)	-> L'état J : relance le ET	-> État PP : lance le PPT	-	-> État NI
Prune-Pending (PP)	-> L'état J : relance le ET	-> État PP	-> État NI envoie Prune-Echo(S,G)	-> État NI

Les événements de transition "Reçoit Join(S,G)" et "Reçoit Prune(S,G)" impliquent de recevoir un Join ou Prune ciblé sur l'adresse IP principale de ce routeur sur l'interface reçue. Si le champ d'adresse du voisin amont n'est pas correcte, ces transitions d'état dans cet automate à états NE DOIVENT PAS se produire, bien que de voir un tel paquet puisse causer des transitions d'état dans les autres automates à états.

Sur les interfaces non numérotées de liaisons point à point, l'adresse du routeur DEVRAIT être la même que l'adresse de source qu'il a choisie pour le message Hello qu'il a envoyé sur cette interface. Cependant, sur les liaisons point à point il est RECOMMANDÉ pour la rétro compatibilité que les messages PIM Join/Prune avec un champ d'adresse de voisin amont tout à zéro soient aussi acceptés.

Transitions de l'état NoInfo

Dans l'état NoInfo, les événements suivants peuvent déclencher une transition :

Reçoit Join(S,G) : un Join(S,G) est reçu sur l'interface I avec son adresse de voisin amont réglée à l'adresse principale IP du routeur sur I. L'automate à états (S,G) aval sur l'interface I passe à l'état Join. Le temporisateur d'expiration (ET) est lancé et réglé au HoldTime *(temps de garde)* à partir du message Join/Prune déclencheur.

Transitions de l'état Join

Dans l'état Join, les événements suivants peuvent déclencher une transition :

Reçoit Join(S,G) : un Join(S,G) est reçu sur l'interface I avec son adresse de voisin amont réglée à l'adresse IP principale du routeur sur I. L'automate à états aval (S,G) sur l'interface I reste dans l'état Join. Le temporisateur d'expiration (ET) est relancé et est alors réglé au maximum de sa valeur actuelle et du temps de garde provenant du message Join/Prune déclencheur.

Reçoit Prune(S,G) : un Prune(S,G) est reçu sur l'interface I avec son adresse de voisin amont réglée à l'adresse IP principale du routeur sur I. L'automate à états aval (S,G) sur l'interface I passe à l'état Prune-Pending. Le temporisateur d'élégage en cours est lancé. Il est réglé à J/P_Override_Interval(I) si le routeur a plus de un voisin sur cette interface ; autrement, il est réglé à zéro, causant son expiration immédiate.

Expiration du temporisateur d'expiration : le temporisateur d'expiration pour l'automate à états aval (S,G) sur l'interface I arrive à expiration. L'automate à états aval (S,G) sur l'interface I passe à l'état NoInfo.

Transitions à partir de l'état Prune-Pending

Dans l'état Prune-Pending, les événements suivants peuvent déclencher une transition :

Reçoit Join(S,G) : un Join(S,G) est reçu sur l'interface I avec son adresse de voisin amont réglée à l'adresse IP principale du routeur sur I. L'automate à états aval (S,G) sur l'interface I passe à l'état Join. Le temporisateur d'élagage en cours est annulé (sans déclencher d'événement d'expiration). Le temporisateur d'expiration (ET) est relancé et est alors réglé au maximum de sa valeur actuelle et du temps de garde provenant du message Join/Prune déclencheur.

Expiration du temporisateur d'expiration : le temporisateur d'expiration pour l'automate à états aval (S,G) sur l'interface I expire. L'automate à états aval (S,G) sur l'interface I passe à l'état NoInfo.

Le temporisateur d'élagage en cours expire

Le temporisateur d'élagage en cours pour l'automate à états aval (S,G) sur l'interface I expire.

L'automate à états aval (S,G) sur l'interface I passe à l'état NoInfo. Un PruneEcho(S,G) est envoyé sur le sous réseau connecté à l'interface I. L'action "Send PruneEcho(S,G)" est déclenchée lorsque le routeur arrête de transmettre sur une interface par suite d'un élagage. Un PruneEcho(S,G) est simplement un message Prune(S,G) envoyé par le routeur amont sur un LAN avec sa propre adresse dans le champ Adresse de voisin amont. Son but est d'ajouter une fiabilité supplémentaire afin que si un élagage qui aurait dû être outrepassé par un autre routeur est perdu en local sur le LAN, le PruneEcho peut alors être reçu et causer l'outrepassement. Un PruneEcho(S,G) n'a pas besoin d'être envoyé sur une interface qui contient seulement un voisin PIM durant le temps où cet automate à états était dans l'état Prune-Pending.

4.5.3 Réception des messages Join/Prune (S,G,rpt)

L'automate à états par interface pour recevoir les messages Join/Prune (S,G,rpt) est donné ci-dessous. Il y a cinq états :

NoInfo (NI) : l'interface n'a pas d'état Prune (S,G,rpt) et aucun temporisateur (S,G,rpt) en cours.

Prune (P) : l'interface a l'état Prune (S,G,rpt) qui va faire que le routeur ne transmet pas de paquet de S destiné à G à partir de cette interface même si l'interface a l'état Join (*,G) actif.

Prune-Pending (PP) : le routeur a reçu un Prune(S,G,rpt) sur cette interface d'un voisin en aval et il attend pour voir si l'élagage sera outrepassé par un autre routeur aval. Pour les besoins de la transmission, l'état Prune-Pending fonctionne exactement comme l'état NoInfo.

PruneTmp (P') : cet état est un état transitoire qui pour les besoins de transmission se comporte exactement comme l'état Prune. Un Join (*,G) a été reçu (qui peut annuler le Prune (S,G,rpt)). Lorsque on analyse le message Join/Prune de haut en bas, on entre d'abord dans cet état si le message contient un Join (*,G). Plus loin dans le message, on va normalement rencontrer un Prune (S,G,rpt) pour réinstaller l'état Prune. Cependant, si on atteint la fin du message sans rencontrer un tel Prune (S,G,rpt) on va revenir à l'état NoInfo dans cet automate à états. Comme on ne passe pas de temps dans cet état, aucun temporisateur ne peut arriver à expiration.

Prune-Pending-Tmp (PP') : cet état est transitoire et est identique à P' excepté qu'il est associé à l'état PP plutôt qu'à l'état P. Pour les besoins de la transmission, PP' se comporte exactement comme l'état PP.

De plus, il y a deux temporisateurs :

Temporisateur d'expiration (ET) : ce temporisateur est lancé lorsque un Prune(S,G,rpt) valide est reçu. L'expiration du temporisateur d'expiration cause le retour de cet automate à états à l'état NoInfo.

Temporisateur d'élagage en cours (PPT) : ce temporisateur est lancé lorsque un Prune(S,G,rpt) valide est reçu. L'expiration du temporisateur d'élagage en cours cause le passage de cet automate à états à l'état Prune.

Figure 4 : Automate à états par interface (S,G,rpt) aval

État antérieur	Événement					
	Reçoit Join(*,G)	Reçoit Join (S,G,rpt)	Reçoit Prune (S,G,rpt)	Fin de message	Expiration du PPT	Expiration du ET
NoInfo (NI)	-	-> état PP lance PPT ; lance ET	-	-	-	-
Prune (P)	-> état P	-> état NI	-> état P relance ET	-	-	-> état NI
Prune-Pending (PP)	-> état PP'	-> état NI	-	-	-> état P	-
PruneTmp (P')	-	-	-> état P relance ET	-> état NI	-	-
Prune-Pending-Tmp (PP')	-	-	-> état PP relance ET	-> état NI	-	-

Les événements de transition "Reçoit Join(S,G,rpt)", "Reçoit Prune(S,G,rpt)", et "Reçoit Join(*,G)" impliquent de recevoir un Join ou Prune ciblé sur l'adresse IP principale de ce routeur sur l'interface reçue. Si le champ Adresse de voisin amont n'est pas correct, ces transitions d'état dans cet automate à états NE DOIVENT PAS se faire, bien que voir un tel paquet puisse causer des transitions d'état dans d'autres automates à états.

Sur des interfaces non numérotées de liaisons point à point, l'adresse du routeur devrait être la même que l'adresse de source qu'il a choisie pour le message Hello qu'il a envoyé sur cette interface. Cependant, sur les liaisons point à point il est RECOMMANDÉ que les messages Join/Prune PIM avec un champ d'adresse de voisin amont tout à zéro soient aussi acceptés.

Transitions à partir de l'état NoInfo
Dans l'état NoInfo (NI) les événements suivants peuvent déclencher une transition :
Reçoit Prune(S,G,rpt)
Un Prune(S,G,rpt) est reçu sur l'interface I avec son adresse de voisin amont réglée à l'adresse IP principale du routeur sur I.
L'automate à états aval (S,G,rpt) sur l'interface I passe à l'état Prune-Pending. Le temporisateur d'expiration (ET) est lancé et réglé au temps de garde provenant du message Join/Prune déclencheur. Le temporisateur d'élagage en cours est lancé. Il est réglé à J/P_Override_Interval(I) si le routeur a plus d'un voisin sur cette interface ; autrement, il est réglé à zéro, causant son expiration immédiate.

Transitions à partir de l'état Prune-Pending
Dans l'état Prune-Pending (PP) les événements suivants peuvent déclencher une transition :
Reçoit Join(*,G)
Un Join(*,G) est reçu sur l'interface I avec son adresse de voisin amont réglée à l'adresse IP principale du routeur sur I.
L'automate à états aval (S,G,rpt) sur l'interface I passe à l'état Prune-Pending-Tmp tandis qu'est traité le reste du message Join/Prune composé contenant le Join(*,G).
Reçoit Join(S,G,rpt)
Un Join(S,G,rpt) est reçu sur l'interface I avec son adresse de voisin amont réglée à l'adresse IP principale du routeur sur I.
L'automate à états aval (S,G,rpt) sur l'interface I passe à l'état NoInfo. ET et PPT sont annulés.
Le temporisateur d'élagage en cours expire.
Le temporisateur d'élagage en cours pour l'automate à états aval (S,G,rpt) sur l'interface I expire.
L'automate à états aval (S,G,rpt) sur l'interface I passe à l'état Prune.

Transitions à partir de l'état Prune
Dans l'état Prune (P), les événements suivants peuvent déclencher une transition :
Reçoit Join(*,G)
Un Join(*,G) est reçu sur l'interface I avec son adresse de voisin amont réglée à l'adresse IP principale du routeur sur I.
L'automate à états aval (S,G,rpt) sur l'interface I passe à l'état PruneTmp tandis qu'est traité le reste du message Join/Prune composé contenant le Join(*,G).
Reçoit Join(S,G,rpt)
Un Join(S,G,rpt) est reçu sur l'interface I avec son adresse de voisin amont réglée à l'adresse IP principale du routeur sur I.
L'automate à états aval (S,G,rpt) sur l'interface I passe à l'état NoInfo. ET et PPT sont annulés.
Reçoit Prune(S,G,rpt)
Un Prune(S,G,rpt) est reçu sur l'interface I avec son adresse de voisin amont réglée à l'adresse IP principale du routeur sur I.
L'automate à états aval (S,G,rpt) sur l'interface I reste dans l'état Prune. Le temporisateur d'expiration (ET) est redémarré et est réglé au maximum de sa valeur actuelle et du HoldTime du message Join/Prune déclencheur.
ET arrive à expiration
Le temporisateur d'expiration pour l'automate à états aval (S,G,rpt) sur l'interface I expire.
L'automate à états aval (S,G,rpt) sur l'interface I passe à l'état NoInfo.

Transitions à partir de l'état Prune-Pending-Tmp
Dans l'état Prune-Pending-Tmp (PP') et en traitant un message Join/Prune composé, les événements suivants peuvent déclencher une transition :
Reçoit Prune(S,G,rpt)
Le message Join/Prune composé contient un Prune(S,G,rpt) qui est reçu sur l'interface I avec son adresse de voisin amont réglée à l'adresse IP principale du routeur sur I.
L'automate à états aval (S,G,rpt) sur l'interface I repasse à l'état Prune-Pending. Le temporisateur ET est redémarré et est alors réglé au maximum de sa valeur actuelle et du HoldTime provenant du message Join/Prune déclencheur.

Fin du message
La fin du message Join/Prune composé est atteinte.
L'automate à états aval (S,G,rpt) sur l'interface I passe à l'état NoInfo. ET et PPT sont annulés.

Transitions à partir de l'état PruneTmp
Dans l'état PruneTmp (P') et en traitant un message Join/Prune composé, les événements suivants peuvent déclencher une transition :
Reçoit Prune(S,G,rpt)
Le message Join/Prune composé contient un Prune(S,G,rpt).
L'automate à états aval (S,G,rpt) sur l'interface I revient à l'état Prune. Le temporisateur ET est redémarré et est alors réglé au maximum de sa valeur actuelle et du HoldTime provenant du message Join/Prune déclencheur.

Fin du message : la fin du message composé Join/Prune est atteinte.
L'automate à états aval (S,G,rpt) sur l'interface I passe à l'état NoInfo. ET est annulé.

Note : la réception d'un Prune(*,G) n'affecte pas l'automate à états (S,G,rpt) aval.

4.5.4 Envoi des messages (*,G) Join/Prune

Les automates à états par interfaces pour (*,G) tiennent l'état Join des routeurs PIM aval. Cet état détermine alors si un routeur a besoin de propager un Join(*,G) en amont vers le RP.

Si un routeur souhaite propager un Join(*,G) vers l'amont, il doit aussi surveiller les messages sur son interface amont en provenance des autres routeurs sur ce sous réseau, et ceux-ci peuvent modifier son comportement. Si il voit un Join(*,G) au voisin amont correct, il devrait supprimer son propre Join(*,G). Si il voit un Prune(*,G) au voisin amont correct, il devrait être prêt à écraser le "prune" par l'envoi d'un Join(*,G) presque immédiatement. Finalement, si il voit changer l'identifiant de génération (voir au paragraphe 4.3) du voisin amont correct, il sait que le voisin amont a perdu ; et il devrait être prêt à rafraîchir l'état en envoyant un Join(*,G) presque immédiatement.

Si une assertion (*,G) se produit sur l'interface amont, et que cela change l'idée que ce routeur se fait du voisin amont, il devrait être prêt à s'assurer que le gagnant de l'assertion est averti des routeurs vers l'aval en envoyant un Join(*,G) presque immédiatement.

De plus, si la MRIB change pour indiquer que le prochain bond vers le RP a changé, et si l'interface amont change ou si il n'y a pas de gagnant d'assertion sur l'interface amont, le routeur devrait élaguer le vieux prochain bond et se joindre au nouveau prochain bond.

L'automate à états (*,G) amont contient seulement deux états :
Not Joined *(non joint)* : l'automate à états aval indique que le routeur n'a pas besoin de se joindre à l'arborescence de RP pour ce groupe.
Joined *(joint)* : l'automate à états aval indique que le routeur devrait se joindre à l'arborescence de RP pour ce groupe.

En plus, un temporisateur JT(*,G) est conservé pour déclencher l'envoi d'un Join(*,G) au prochain bond amont vers le RP, RPF'(*,G).

Figure 5 : Automate à états (*,G) amont

État précédent	**Événement**	
	Join désiré(*,G) ->Vrai	Join désiré(*,G) ->Faux
NonJoint (NJ)	–> état J : Envoyer Join(*,G) ; régler le temporisateur Join à t_periodic	-
Joint (J)	-	-> état NJ : Envoyer Prune(*,G) ; annuler le temporisateur Join

De plus, nous avons les transitions suivantes, qui se produisent dans l'état Joined :

Dans l'état Joint (J)			
Temporisateur expire	Voit Join(*,G) en RPF'(*,G)	Voit Prune(*,G) à RPF'(*,G)	RPF'(*,G) change à cause d'une Assert
Envoie Join(*,G) ; règle Join Timer à t_periodic	Augmente Join Timer à t_joinsuppress	Diminue Join Timer à t_override	Diminue Join Timer à t_override

|

Dans l'état Joint (J)	
RPF'(*,G) change mais pas à cause d'une Assert	RPF'(*,G) GenID change
Envoie Join(*,G) au nouveau prochain bond ; envoie Prune(*,G) à l'ancien prochain bond ; règle Join Timer à t_periodic	Diminue Join Timer à t_override

Cet automate à états utilise la macro suivante :

```
bool JoinDesired(*,G) {
  si (immediate_olist(*,G) != NUL)
     retourner VRAI
  autrement
     retourner FAUX
}
```

JoinDesired(*,G) est vrai lorsque le routeur a transmis l'état qui lui ferait transmettre le trafic pour G en utilisant l'état d'arborescence partagée. Noter que bien que JoinDesired soit vrai, l'envoi du routeur d'un message Join(*,G) peut être supprimé par un autre routeur envoyant un Join(*,G) sur l'interface amont.

Transitions à partir de l'état NonJoint

Lorsque l'automate à états (*,G) amont est dans l'état NonJoint, l'événement suivant peut déclencher une transition d'état:

JoinDesired(*,G) devient vrai
La macro JoinDesired(*,G) devient vraie, par exemple, parce que l'état aval pour (*,G) a changé de sorte qu'au moins une interface est une immediate_olist(*,G).

L'automate à états (*,G) amont passe à l'état Joined. Envoie Join(*,G) au voisin amont approprié, qui est RPF'(*,G). Régler le temporisateur Join (JT) à expirer après t_periodic secondes.

Transitions à partir de l'état Joined

Lorsque l'automate à états (*,G) amont est dans l'état Joined, les événements suivants peuvent déclencher les transitions d'état :

JoinDesired(*,G) devient Faux
La macro JoinDesired(*,G) devient Faux, par exemple, parce que l'état aval pour (*,G) a changé de sorte qu'aucune interface n'est en immediate_olist(*,G).

L'automate à états (*,G) amont passe à l'état NonJoint. Envoyer Prune(*,G) au voisin amont approprié, qui est RPF'(*,G). Annuler le temporisateur Join (JT).

Le temporisateur Join arrive à expiration
Le temporisateur Join (JT) arrive à expiration, indiquant qu'il est temps d'envoyer un Join(*,G).

Envoyer Join(*,G) au voisin amont approprié, qui est RPF'(*,G). Redémarrer le temporisateur Join (JT) pour qu'il arrive à expiration après t_periodic secondes.

Voir Join(*,G) en RPF'(*,G)
Cet événement n'est pertinent que si RPF_interface(RP(G)) est un support partagé. Ce routeur voit un autre routeur sur RPF_interface(RP(G)) envoyer un Join(*,G) à RPF'(*,G). Cela amène ce routeur à supprimer son propre Join.

L'automate à états (*,G) amont reste dans l'état Joined.

Soit t_joinsuppress le minimum de t_suppressed et du HoldTime provenant du message Join/Prune qui déclenche cet événement. Si le temporisateur Join est réglé à expirer dans moins de t_joinsuppress secondes, le rétablir afin qu'il expire après t_joinsuppress secondes. Si le temporisateur Join est réglé à expirer dans plus de t_joinsuppress secondes, le laisser inchangé.

Voir Prune(*,G) en RPF'(*,G)
Cet événement n'est pertinent que si RPF_interface(RP(G)) est un support partagé. Ce routeur voit un autre routeur sur RPF_interface(RP(G)) envoyer un Prune(*,G) à RPF'(*,G). Comme ce routeur est dans l'état Joined, il doit outrepasser le Prune après un bref intervalle aléatoire.

L'automate à états (*,G) amont reste dans l'état Joined. Si le temporisateur Join est réglé à expirer dans plus de t_override secondes, le rétablir afin qu'il expire après t_override secondes. Si le temporisateur Join est réglé à expirer dans moins de t_override secondes, le laisser inchangé.

RPF'(*,G) change à cause d'un Assert.
Le prochain bond actuel vers le RP change à cause d'un Assert(*,G) sur la RPF_interface(RP(G)).

L'automate à états (*,G) amont reste dans l'état Joined.
Si le temporisateur Join est réglé à expirer dans plus de t_override secondes, le rétablir afin qu'il expire après t_override secondes. Si le temporisateur Join est réglé à expirer dans moins de t_override secondes, le laisser inchangé.

RPF'(*,G) change mais pas à cause d'un Assert.
Un événement s'est produit qui a causé le changement du prochain bond vers le RP pour G. Ce peut être causé par un changement dans la base de données d'acheminement MRIB ou par la transposition de groupe en RP. Noter que cette transition ne se produit pas si un Assert est actif et si l'interface amont ne change pas.

L'automate à états (*,G) amont reste dans l'état Joined. Envoyer Join(*,G) au nouveau voisin amont, qui est la nouvelle valeur de RPF'(*,G). Envoyer Prune(*,G) à l'ancien voisin amont, qui est l'ancienne valeur de RPF'(*,G). Utiliser la nouvelle valeur de RP(G) dans le message Prune(*,G) ou tout à zéro si RP(G) devient inconnu (l'ancienne valeur de RP(G) peut être utilisée à la place pour améliorer le comportement dans les routeurs qui mettent en œuvre d'anciennes versions de cette spécification). Régler le temporisateur Join (JT) à expirer après t_periodic secondes.

RPF'(*,G) GenID change. L'identifiant de génération du routeur qui est RPF'(*,G) change. Cela signifie normalement que ce voisin a perdu l'état, et donc, l'état doit être rafraîchi.

L'automate à états (*,G) amont reste dans l'état Joined. Si le temporisateur Join est réglé à expirer dans plus de t_override secondes, le rétablir afin qu'il expire après t_override secondes.

4.5.5 Envoi des messages (S,G) Join/Prune

L'automate à états par interfaces pour (S,G) détient l'état Join provenant des routeurs PIM aval. Cet état détermine alors si un routeur a besoin de propager un Join(S,G) amont vers la source.

Si un routeur souhaite propager un Join(S,G) vers l'amont, il doit aussi regarder les messages sur son interface amont provenant des autres routeurs sur ce sous réseau, et ceux-ci peuvent modifier son comportement. Si il voit un Join(S,G) pour le voisin amont correct, il devrait supprimer son propre Join(S,G). Si il voit un Prune(S,G), Prune(S,G,rpt), ou Prune(*,G) pour le voisin amont correct vers S, il devrait être prêt à outrepasser cet élagage en programmant un Join(S,G) à envoyer presque immédiatement. Finalement, si il voit que l'identifiant de génération de son voisin amont change, il sait que le voisin amont a perdu l'état, et il devrait rafraîchir l'état en programmant un Join(S,G) à envoyer presque immédiatement.

Si un (S,G) Assert se produit sur l'interface amont, et si cela change l'idée qu'a ce routeur du voisin amont, il devrait être prêt à s'assurer que le vainqueur de l'Assert est au courant des routeurs aval en programmant un Join(S,G) à envoyer presque immédiatement.

De plus, si les changements de la MRIB causent le changement du prochain bond vers la source, et si l'interface amont change ou si il n'y a pas de vainqueur de l'Assert sur l'interface amont, le routeur devrait envoyer un élagage à l'ancien prochain bond et un Join au nouveau prochain bond.

L'automate à états (S,G) amont contient seulement deux états :
NonJoint (NJ) : Les automates à états aval et les informations de membre locales n'indiquent pas que le routeur a besoin de se joindre à l'arborescence de plus court chemin pour ce (S,G).

Joint (J) : Les automates à états aval et les informations de membre locales indiquent que le routeur devrait se joindre à l'arborescence de plus court chemin pour ce (S,G).

De plus, un temporisateur JT(S,G) est conservé pour être utilisé à déclencher l'envoi d'un Join(S,G) au prochain bond amont vers S, RPF'(S,G).

Figure 6 : Automate à états amont (S,G)

	Événement	
État précédent	JoinDesired(S,G) ->Vrai	JoinDesired(S,G) ->Faux
NonJoint (NJ)	–> état J : Envoyer Join(S,G) ; régler le JT à t_periodic	-
Joint (J)	-	-> état NJ : Envoyer Prune(S,G) ; régler SPTbit(S,G) à FAUX annuler le JT

De plus, on a les transitions suivantes, qui se produisent dans l'état Joint :

Dans l'état Joint (J)			
Temporisateur expire	Voit Join(S,G) à RPF'(S,G)	Voit Prune(S,G) à RPF'(S,G)	Voit Prune (S,G,rpt) à RPF'(S,G)
Envoie Join(S,G) ; règle JT à t_periodic	Augmente JT à t_joinsuppress	Diminue JT à t_override	Diminue JT à t_override

Dans l'état Joint (J)			
Voit Prune(*,G) à RPF'(S,G)	RPF'(S,G) change, pas à cause d'un Assert	RPF'(S,G) GenID change	RPF'(S,G) change à cause d'un Assert
Diminue JT à t_override	Envoie Join(S,G) au nouveau prochain bond ; envoie Prune(S,G) à l'ancien prochain bond ; règle JT à t_periodic	Diminue JT à t_override	Diminue JT à t_override

Cet automate à états utilise la macro suivante :

```
bool JoinDesired(S,G) {
   retourner( immediate_olist(S,G) != NUL
         OU ( KeepaliveTimer(S,G) court
            ET inherited_olist(S,G) != NUL ) )
}
```

JoinDesired(S,G) est vrai lorsque le routeur a l'état transmission qui va l'amener à transmettre du trafic pour G en utilisant l'état d'arborescence de source. L'état d'arborescence de source peut être le résultat soit d'un état de jonction actif spécifique de la source, soit du temporisateur de garde en vie (S,G) et d'un état actif non spécifique de la source. Noter que bien que JoinDesired soit vrai, l'envoi par le routeur d'un message Join(S,G) peut être supprimé par l'envoi par un autre routeur d'un Join(S,G) sur l'interface amont.

Transitions à partir de l'état NonJoint

Lorsque l'automate à états amont (S,G) est dans l'état NonJoint, l'événement suivant peut déclencher une transition d'état :

JoinDesired(S,G) devient Vrai
La macro JoinDesired(S,G) devient Vrai, par exemple, parce que l'état aval pour (S,G) a changé de sorte qu'au moins une interface est en inherited_olist(S,G).

L'automate à état amont (S,G) transite à l'état Joint. Envoie Join(S,G) au voisin amont approprié, qui est RPF'(S,G). Règle le temporisateur Join (JT) à expirer après t_periodic secondes.

Transitions à partir de l'état Joint

Lorsque l'automate à état amont (S,G) est dans l'état Joint, les événements suivants peuvent déclencher une transition d'état :

JoinDesired(S,G) devient Faux.

La macro JoinDesired(S,G) devient Faux, par exemple, parce que l'état aval pour (S,G) a changé de sorte qu'aucune interface n'est dans inherited_olist(S,G).

L'automate à état amont (S,G) passe à l'état NonJoint. Il envoie Prune(S,G) au voisin amont approprié, qui est (S,G) de RPF. Il annule le temporisateur Join (JT), et règle le bit SPT(S,G) à FAUX.

Le temporisateur Join expire.
Le JT expire, indiquant qu'il est temps d'envoyer un Join(S,G).
Envoi de Join(S,G) au voisin amont approprié, qui est RPF'(S,G). Redémarrage du temporisateur Join (JT) à expirer après t_periodic secondes.

Voit Join(S,G) à RPF'(S,G)
Cet événement n'est pertinent que si RPF_interface(S) est un support partagé. Ce routeur voit qu'un autre routeur sur RPF_interface(S) a envoyé un Join(S,G) à RPF'(S,G). Cela cause la suppression par ce routeur de son propre Join.

L'automate à état amont (S,G) reste dans l'état Joint.

Soit t_joinsuppress le minimum de t_suppressed et du temps de garde provenant du message Join/Prune qui déclenche cet événement.

Si le JT est réglé à expirer dans moins de t_joinsuppress secondes, le rétablir pour qu'il expire après t_joinsuppress secondes. Si le JT est réglé à expirer dans plus de t_joinsuppress secondes, le laisser inchangé.

Voit Prune(S,G) à RPF'(S,G)
Cet événement n'est pertinent que si RPF_interface(S) est un support partagé. Ce routeur voit qu'un autre routeur sur RPF_interface(S) a envoyé un Prune(S,G) à RPF'(S,G). Comme ce routeur est dans l'état Joint, il doit outrepasser le Prune après un court intervalle aléatoire.

L'automate à état amont (S,G) reste dans l'état Joint. Si le JT est réglé à expirer dans plus de t_override secondes, le rétablir pour qu'il expire après t_override secondes.

Voit Prune(S,G,rpt) à RPF'(S,G)
Cet événement n'est pertinent que si RPF_interface(S) est un support partagé. Ce routeur voit qu'un autre routeur sur RPF_interface(S) a envoyé un Prune(S,G,rpt) à RPF'(S,G). Si le routeur amont est un routeur PIM conforme à la RFC2362, le Prune(S,G,rpt) va alors lui faire arrêter la transmission. Pour la rétro compatibilité, ce routeur devrait outrepasser le Prune pour que la transmission continue.

L'automate à états amont (S,G) reste dans l'état Joint. Si le JT est réglé à expirer dans plus de t_override secondes, le rétablir afin qu'il expire après t_override secondes.

Voit Prune(*,G) à RPF'(S,G)
Cet événement n'est pertinent que si RPF_interface(S) est un support partagé. Ce routeur voit qu'un autre routeur sur RPF_interface(S) a envoyé un Prune(*,G) à RPF'(S,G). Si le routeur amont est un routeur PIM conforme à la RFC2362, alors le Prune(*,G) va lui faire arrêter la transmission. Pour la rétro compatibilité, ce routeur devrait outrepasser le Prune afin que la transmission continue.

L'automate à état amont (S,G) reste dans l'état Joint. Si le JT est réglé à expirer dans plus de t_override secondes, le rétablir afin qu'il expire après t_override secondes.

RPF'(S,G) change à cause d'un Assert
Le prochain bond actuel vers S change à cause d'un Assert(S,G) sur la RPF_interface(S).

L'automate à états amont (S,G) reste dans l'état Joint. Si le JT est réglé à expirer dans plus de t_override secondes, le rétablir pour qu'il expire après t_override secondes. Si le JT est réglé à expirer dans moins de t_override secondes, le laisser inchangé.

RPF'(S,G) change mais pas à cause d'un Assert
Un événement s'est produit qui a causé un changement du prochain bond vers S. Noter que cette transition ne se produit pas si un Assert est actif et si l'interface amont ne change pas.

L'automate à état amont (S,G) reste dans l'état Joint. Envoi d'un Join(S,G) au nouveau voisin amont, qui est la nouvelle valeur de RPF'(S,G). Envoi d'un Prune(S,G) à l'ancien voisin amont, qui est l'ancienne valeur de RPF'(S,G). Régler le JT à expirer après t_periodic secondes.

RPF'(S,G) GenID change
L'identifiant de génération du routeur qui est RPF'(S,G) change. Cela signifie normalement que ce voisin a perdu l'état, et donc l'état doit être rafraîchi.

L'automate à état amont (S,G) reste dans l'état Joint. Si le temporisateur Join est réglé à expirer dans plus de t_override secondes, le rétablir afin qu'il expire après t_override secondes.

4.5.6 Messages périodiques (S,G,rpt)

Les Join et Prune (S,G,rpt) sont des Join ou Prune (S,G) envoyés sur l'arborescence de RP avec le bit RPT établi, soit pour modifier le résultat des Join(*,G), soit pour outrepasser le comportement des autres homologues de LAN en amont. Le paragraphe suivant décrit les règles d'envoi des messages déclenchés. Ce paragraphe décrit les règles pour inclure un message Prune(S,G,rpt) avec un Join(*,G).

Lorsque un routeur va envoyer un Join(*,G), il devrait utiliser le pseudo code suivant, pour chaque (S,G) pour lequel il a un état, pour décider si il inclut un Prune(S,G,rpt) dans le message Join/Prune composé :

```
 si( SPTbit(S,G) == VRAI ) {
# Note : Si on reçoit (S,G) sur le SPT, on élague seulement l'arborescence partagée si les voisins RPF diffèrent.
       si( RPF'(*,G) != RPF'(S,G) ) {
          ajouter Prune(S,G,rpt) au message composé
       }
   } autrement si ( inherited_olist(S,G,rpt) == NUL ) {
# Note : Toutes les interfaces (*,G) olist ont reçu des élagages de RPT pour (S,G).
     ajouter Prune(S,G,rpt) au message composé
   } autrement si ( RPF'(*,G) != RPF'(S,G,rpt) {
# Note : On s'est joint à l'arborescence partagée, mais il y avait un (S,G) qui affirme que le voisin RPF de l'arborescence partagée est différent.
     ajouter Prune(S,G,rpt) au message composé
   }
```

Noter que Join(S,G,rpt) n'est normalement pas envoyé comme message périodique, mais seulement comme message déclenché.

4.5.7 Automate à états pour messages déclenchés (S,G,rpt)

L'automate à états pour les messages déclenchés (S,G,rpt) est exigé selon (S,G) quand il y a un état (*,G) à un routeur, et le routeur ou un de ses homologues de LAN en amont souhaite élaguer S de l'arborescence de RP.

Il y a trois états dans l'automate à états. Un des états est quand il n'y a pas d'état Join(*,G) à ce routeur. Si il y a un état Join(*,G) au routeur, l'automate à états doit être dans un des deux autres états. Les trois états sont :

Élagué(S,G,rpt)
(*,G) Joint, mais (S,G,rpt) élagué.

NonÉlagué(S,G,rpt)
(*,G) Joint, et (S,G,rpt) non élagué.

RPTNonJoint(G)
(*,G) n'a pas été joint.

De plus, il y a un temporisateur d'outrepassement (OT, *Override Timer*) (S,G,rpt) OT(S,G,rpt), qui est utilisé pour retarder les messages déclenchés Join(S,G,rpt) pour empêcher l'explosion de messages déclenchés.

Figure 7 : Automate à états pour messages déclenchés en amont (S,G,rpt)

	Événement			
État précédent	PruneDesiré (S,G,rpt) -->Vrai	PruneDesiré (S,G,rpt) -->Faux	RPTJoinDesiré(G) -->Faux	olist_héritée (S,G,rpt) --> non NUL
RPTNonJoint (G) (NJ)	--> état P	-	-	--> état NP
Élagué(S,G,rpt) (P)	-	--> état NP envoi de Join (S,G,rpt)	--> état NJ	-
NonÉlagué(S,G,rpt) (NP)	--> état P envoi de Prune (S,G,rpt) ; annule OT	-	--> état NJ ; annule OT	-

De plus, on a les transitions suivantes au sein de l'état NonÉlagué(S,G,rpt), qui sont toutes utilisées pour élaguer le comportement d'outrepassement.

Dans l'état NonÉlagué(S,G,rpt)				
Expiration de OT	Voit Prune (S,G,rpt) à RPF' (S,G,rpt)	Voit Join (S,G,rpt) à RPF' (S,G,rpt)	Voit Prune (S,G) à RPF' (S,G,rpt)	RPF' (S,G,rpt) -> RPF' (*,G)
envoie Join (S,G,rpt) ; laisse OT non établi	OT = min(OT, t_override)	Annule OT	OT = min(OT, t_override)	OT = min(OT, t_override)

Noter que la fonction min dans l'automate à états ci-dessus considère un temporisateur non tournant comme ayant une valeur infinie (par exemple, min(not-running, t_override) = t_override).

Cet automate à états utilise les macros suivantes :

```
bool RPTJoinDesired(G) {
  retourner (JoinDesired(*,G))
}
```

RPTJoinDesired(G) est vrai quand le routeur a un état de transmission qui causerait sa transmission de trafic pour G en utilisant l'état d'arborescence partagée (*,G).

```
bool PruneDésiré(S,G,rpt) {
    retourner ( RPTJoinDesiré(G) ET
          ( inherited_olist(S,G,rpt) == NUL
            OU (SPTbit(S,G)==VRAI
               ET (RPF'(*,G) != RPF'(S,G)) )))
}
```

PruneDésiré(S,G,rpt) peut seulement être vrai si RPTJoinDesired(G) est vrai. Si RPTJoinDesired(G) est vrai, alors PruneDésiré(S,G,rpt) est vrai soit si il n'y a pas d'interface sortante sur laquelle S pourrait être transmis, soit si le routeur a l'état de transmission actif (S,G) mais RPF'(*,G) != RPF'(S,G).

L'automate à états contient les événements de transition suivants :

Voit Join(S,G,rpt) à RPF'(S,G,rpt)
Cet événement n'est pertinent que dans l'état "NonÉlagué".
Le routeur voit un Join(S,G,rpt) provenant de quelqu'un d'autre que RPF'(S,G,rpt), qui est le voisin amont correct. Si on est dans l'état "NonÉlagué" et si le temporisateur d'outrepassement (S,G,rpt) fonctionne, c'est parce que l'envoi de notre propre Join(S,G,rpt) à RPF'(S,G,rpt) a été déclenché. Quelqu'un d'autre nous a battu, de sorte qu'il n'est plus besoin d'envoyer notre propre Join.
L'action est d'annuler le temporisateur d'outrepassement.

Voit Prune(S,G,rpt) à RPF'(S,G,rpt)
Cet événement n'est pertinent que dans l'état "NonÉlagué".
Le routeur voit un Prune(S,G,rpt) provenant de quelqu'un d'autre à RPF'(S,G,rpt), qui est le voisin amont correct. Si on est dans l'état "NonÉlagué", on veut alors continuer de recevoir le trafic provenant de S destiné à G, et ce trafic est fourni par RPF'(S,G,rpt). Donc, on a besoin d'outrepasser le Prune. L'action est d'établir le temporisateur d'outrepassement (S,G,rpt) à l'intervalle aléatoire d'outrepassement d'élagage, t_override. Cependant, si le temporisateur d'outrepassement est déjà en cours, on n'établit le temporisateur que si cela le règle à une valeur inférieure. À la fin de cet intervalle, si aucun autre n'a envoyé de Join, alors on le fera.

Voit Prune(S,G) à RPF'(S,G,rpt)
Cet événement n'est pertinent que dans l'état "NonÉlagué".
Cette transition et cette action sont les mêmes que la transition et l'action ci-dessus, excepté que le Prune n'a pas le bit RPT établi. Cette transition est nécessaire pour être compatible avec les mises en œuvre de routeurs de la RFC 2362 qui ne tiennent pas les états (S,G) et (S,G,rpt) séparés.

Le temporisateur d'outrepassement d'élagage (S,G,rpt) arrive à expiration.
Cet événement n'est pertinent que dans l'état "NonÉlagué".
Lorsque le temporisateur d'outrepassement expire, on doit envoyer un Join(S,G,rpt) à RPF'(S,G,rpt) pour outrepasser le message Prune qui a causé le démarrage du temporisateur. On n'envoie cela que si RPF'(S,G,rpt) égale RPF'(*,G) ; si ce n'est pas le cas, le Join pourrait alors être envoyé à un routeur qui n'a pas l'état (*,G) Join, et ainsi le comportement pourrait n'être pas bien défini. Si RPF'(S,G,rpt) n'est pas le même que RPF'(*,G), il peut alors arrêter de transmettre S. Cependant, si cela arrive, le routeur va alors envoyer un AssertCancel(S,G), qui va causer l'égalité de RPF'(S,G,rpt) et de RPF'(*,G) (voir ci-dessous).

RPF'(S,G,rpt) change pour devenir égal à RPF'(*,G).
Cet événement n'est pertinent que dans l'état "NonÉlagué".
RPF'(S,G,rpt) peut seulement être différent de RPF'(*,G) si un (S,G) Assert s'est produit, ce qui signifie que le trafic provenant de S arrive sur le SPT, et donc Prune(S,G,rpt) aura été envoyé à RPF'(*,G). Lorsque RPF'(S,G,rpt) change pour devenir égal à RPF'(*,G), on doit déclencher un Join(S,G,rpt) à RPF'(*,G) pour faire que ce routeur commence à transmettre S à nouveau.
L'action est de régler le temporisateur d'outrepassement (S,G,rpt) à l'intervalle aléatoire d'outrepassement d'élagage t_override. Cependant, si le temporisateur court déjà, on ne le règle que si cela le met à une valeur inférieure. À la fin de cet intervalle, si aucun autre n'a envoyé un Join, alors on le fait.

PruneDésiré(S,G,rpt)->VRAI
Voir la macro ci-dessus. Cet événement est pertinent dans les états "NonÉlagué" et "RPTNonJoint(G)".
Le routeur souhaite recevoir le trafic pour G mais ne souhaite par recevoir le trafic de S pour G. Cela cause la transition du routeur à l'état Élagué.
Si le routeur était précédemment dans l'état NonÉlagué, l'action est alors d'envoyer un Prune(S,G,rpt) à RPF'(S,G,rpt), et d'annuler le temporisateur d'outrepassement. Si le routeur était précédemment dans l'état RPTNonJoint(G), il n'est alors pas nécessaire de déclencher une action dans cet automate à états parce que l'envoi d'un Prune(S,G,rpt) est traité par les règles pour l'envoi du Join(*,G).

PruneDésiré(S,G,rpt)->FAUX
Voir la macro ci-dessus. Cette transition n'est pertinente que dans l'état "Élagué".
Si le routeur est dans l'état Élagué(S,G,rpt), et si PruneDésiré(S,G,rpt) se change en FAUX, ce pourrait être parce que le routeur n'a plus RPTJoinDesired(G) vrai, ou qu'il souhaite maintenant recevoir à nouveau le trafic provenant de S. Si c'est le premier cas, alors cette transition ne devrait pas avoir lieu, mais c'est plutôt la transition "RPTJoinDesired(G)->FAUX" qui devrait se produire. Donc, cette transition devrait être interprétée comme "PruneDesired(S,G,rpt)->FAUX ET RPTJoinDesired(G)=VRAI". L'action est d'envoyer un Join(S,G,rpt) à RPF'(S,G,rpt).

RPTJoinDesired(G)->FAUX
Cet événement est pertinent dans les états "Élagué" et "NonÉlagué".
Le routeur ne souhaite plus recevoir de trafic destiné à G sur l'arborescence de RP. Cela cause une transition à l'état RPTNonJoint(G), et le temporisateur d'outrepassement est annulé si il courait. Toutes les autres actions sont traitées par l'automate à états amont approprié pour (*,G).

inherited_olist(S,G,rpt) devient non NUL
Cette transition n'est pertinente que dans l'état RPTNonJoint(G).
Le routeur s'est joint à l'arborescence de RP (traitée comme approprié par l'automate à états (*,G) en amont) et veut recevoir le trafic de S. Cela ne déclenche aucun événement dans cet automate à états, mais cause une transition à l'état NonÉlagué (S,G,rpt).

4.6 Messages PIM Assert

Lorsque plusieurs routeurs PIM échangent du trafic sur un LAN partagé, il est possible que plus d'un routeur en amont ait un état de transmission valide pour un paquet, ce qui peut conduire à la duplication du paquet (voir au paragraphe 3.6). PIM ne tente pas d'empêcher cela d'arriver. Il détecte plutôt quand c'est arrivé et choisit un seul transmetteur parmi les routeurs amont pour empêcher d'autres duplications. Ce choix est effectué en utilisant les messages PIM Assert. Les messages Assert sont aussi reçus par les routeurs en aval sur le LAN, et ils causent l'envoi de messages Join/Prune au routeur amont qui a gagné le Assert.

En général, un message PIM Assert devrait n'être accepté au traitement que si il vient d'un voisin PIM connu. Un routeur PIM entend parler de ses voisins PIM par les messages PIM Hello. Si un routeur reçoit un message Assert d'une certaine adresse IP de source et si il n'a pas vu de message Hello PIM provenant de cette adresse de source, le message Assert DEVRAIT alors être éliminé sans autre traitement. De plus, si le message Hello provenant d'un voisin a été authentifié (voir au paragraphe 6.3), alors tous les messages Assert provenant de ce voisin DOIVENT aussi être authentifiés.
On note que certaines anciennes mises en œuvre de PIM échoue incorrectement à envoyer des messages Hello sur des interfaces point à point, de sorte qu'on RECOMMANDE aussi qu'une option de configuration soit fournie pour permettre l'interopération avec ces vieux routeurs, mais cette option de configuration NE DEVRAIT PAS être activée par défaut.

4.6.1 Automate à états de message Assert (S,G)

L'automate à états Assert (S,G) pour l'interface I est montré à la Figure 8. Il y a trois états :

NoInfo (NI) : ce routeur n'a pas d'état Assert (S,G) sur l'interface I.

Je suis le gagnant de l'assertion (W) : ce routeur a gagné un Assert (S,G) sur l'interface I. Il est maintenant responsable de la transmission du trafic de S destiné à G qui sort de l'interface I. Sans considération de si il est le DR pour I, lorsque un routeur est le gagnant de l'Assert, il est aussi responsable de la transmission du trafic à I au nom des hôtes locaux sur I qui ont fait des demandes d'adhésion qui se réfèrent spécifiquement à S (et G).

Je suis le perdant de l'Assert (L) : ce routeur a perdu un Assert (S,G) sur l'interface I. Il ne doit pas transmettre de paquets de S destinés à G sur l'interface I. Si il est le DR sur I, il n'est plus responsable de la transmission du trafic sur I pour satisfaire les hôtes locaux dont les demandes d'adhésion se réfèrent spécifiquement à S et G.

De plus, il y a aussi un temporisateur d'assertions (AT, *Assert Timer*) qui est utilisé pour périmer les assertions sur les perdants de Assert et pour renvoyer les assertions sur le gagnant de l'Assert.

Figure 8 : Automate à états par interface Assert (S,G)

Dans l'état NoInfo (NI)			
Reçoit InferiorAssert avec le bit RPT à zéro	Reçoit Assert avec bit RPT établi et CouldAssert (S,G,I)	Les données arrivent de S à G sur I et CouldAssert (S,G,I)	Reçoit Assert acceptable avec le bit RPT à zéro et AssTrDes (S,G,I)
-> état W [Actions A1]	-> état W [Actions A1]	-> état W [Actions A1]	-> état L [Actions A6]

Dans l'état Je suis le gagnant de Assert (W, *winner*)			
AT expire	Reçoit Assert inférieur	Reçoit Assert préféré	CouldAssert (S,G,I) -> FAUX
-> état W [Actions A3]	-> état W [Actions A3]	-> état L [Actions A2]	->état NI[Actions A4]

Dans l'état Je suis le perdant de Assert (L, *Loser*)				
|Reçoit Assert préféré	Reçoit Assert acceptable avec bit RPT à zéro du gagnant actuel	Reçoit InferiorAssert ou Assert Cancel du gagnant actuel	AT expire	GenIDChanges du gagnant actuel ou NLT expire
-> état L [Actions A2]	-> état L [Actions A2]	-> état NI [Actions A5]	-> état NI [Actions A5]	-> état NI [Actions A5]

Dans l'état Je suis le perdant de Assert (L)			
AssTrDes (S,G,I) -> FAUX	my_metric -> meilleure que la métrique du gagnant	RPF_interface(S) arrête d'être I	Reçoit Join(S,G) sur l'interface I
-> état NI [Actions A5]	-> état NI [Actions A5]	-> état NI [Actions A5]	-> état NI [Actions A5]

Noter que pour des raisons de concision, "AssTrDes(S,G,I)" est utilisé dans le tableau de l'automate à états pour se référer à AssertTrackingDésiré(S,G,I).

Terminologie :

Un "Assert préféré" est celui qui a une meilleure métrique que le gagnant actuel.

Un "Assert acceptable" est celui qui a une meilleure métrique que my_assert_metric(S,G,I). Un Assert n'est jamais considéré acceptable si sa métrique est infinie.

Un "Assert inférieur" est celui qui a une plus mauvaise métrique que my_assert_metric(S,G,I). Un Assert n'est jamais considéré inférieur si my_assert_metric(S,G,I) est infinie.

L'automate à états utilise les macros suivantes :

```
CouldAssert(S,G,I) =
   SPTbit(S,G)==VRAI
   ET (RPF_interface(S) != I)
   ET (I dans ( ( joins(*,G) (-) prunes(S,G,rpt) )
            (+) ( pim_include(*,G) (-) pim_exclude(S,G) )
            (-) lost_assert(*,G)
            (+) joins(S,G) (+) pim_include(S,G) ) )
```

CouldAssert(S,G,I) est vrai pour les interfaces en aval qui seraient dans la inherited_olist(S,G) si les informations de Assert(S,G) n'étaient pas prises en compte.

```
AssertTrackingDésiré(S,G,I) =
   (I dans ( joins(*,G) (-) prunes(S,G,rpt)
        (+) ( pim_include(*,G) (-) pim_exclude(S,G) )
         (-) lost_assert(*,G)
         (+) joins(S,G) ) )
   OU (local_receiver_include(S,G,I) == VRAI
      ET (Je_suis_DR(I) OU (AssertWinner(S,G,I) == moi)))
   OU ((RPF_interface(S) == I) ET (JoinDesired(S,G) == VRAI))
   OU ((RPF_interface(RP(G)) == I) ET (JoinDesired(*,G) == VRAI)
      ET (SPTbit(S,G) == FAUX))
```

AssertTrackingDésiré(S,G,I) est vrai sur toute interface dans laquelle un Assert(S,G) pourrait affecter le comportement du routeur sur cette interface.

Les trois premières lignes de AssertTrackingDésiré comptent pour les informations de (*,G) join et de membres locaux reçues sur I qui pourraient causer que le routeur va être intéressé aux assertions sur I.

La quatrième ligne compte pour les informations (S,G) join reçues sur I qui pourraient causer que le routeur va être intéressé par les assertions sur I.

La cinquième et la sixième lignes comptent pour les informations de membres locaux (S,G) sur I. Noter qu'on ne peut pas utiliser la macro pim_include(S,G) car elle utilise lost_assert(S,G,I) et il en résulterait que le routeur oublierait qu'il a perdu une assertion si la seule raison de son intérêt était les membres locaux. La vérification AssertWinner(S,G,I) force un gagnant d'assertion à continuer d'être responsable de la transmission tant que des receveurs locaux sont présents. Retirer cette vérification ferait que le gagnant d'assertion abandonnerait la transmission aussitôt que disparaîtrait l'information qui a causé à l'origine qu'il transmette, et la tâche de la transmission pour les receveurs locaux reviendrait au DR.

Les trois dernières lignes comptent pour le fait qu'un routeur doit garder trace des informations d'assertion sur les interfaces amont afin d'envoyer les Join et Prune au voisin approprié.

Transitions à partir de l'état NoInfo.
Lorsque dans l'état NoInfo, les événements suivants peuvent déclencher des transitions :

Reçoit Assert inférieur avec le bit RPT à zéro
Un assert est reçu pour (S,G) avec le bit RPT à zéro, dont la métrique est inférieure à celle de notre propre métrique d'assertion. Le bit RPT à zéro indique que l'envoyeur de l'assertion avait l'état de transmission (S,G) sur cette interface. Si l'assertion est inférieure à notre métrique, alors on doit aussi avoir l'état de transmission (S,G) (c'est-à-dire, CouldAssert(S,G,I)==VRAI) car les assertions (S,G) ont priorité sur les assertions (*,G) et donc on devrait être le gagnant de l'assertion. On transite à l'état "Je suis le gagnant de l'assertion" et on effectue les actions A1 (ci-dessous).

Reçoit Assert avec le bit RPT établi ET CouldAssert(S,G,I)==VRAI
Une assertion est reçue pour (S,G) sur I avec le bit RPT établi (il est une assertion (*,G)). CouldAssert(S,G,I) est VRAI seulement si on a l'état de transmission (S,G) sur cette interface, de sorte qu'on devrait être le gagnant de l'assertion. On transite à l'état "Je suis le gagnant de l'assertion" et on effectue les actions A1 (ci-dessous).

Un paquet de données (S,G) arrive sur l'interface I, ET CouldAssert(S,G,I)==VRAI

Un paquet de données (S,G) est arrivé sur une interface en aval qui est dans notre liste d'interfaces (S,G) sortantes. On suppose de façon optimiste qu'on va être le gagnant de l'assertion pour ce (S,G), et donc on transite à l'état "Je suis le gagnant de l'assertion" et on effectue les actions A1 (ci-dessous) qui vont initier la négociation d'assertion pour (S,G).

Reçoit Assert acceptable avec le bit RPT à zéro ET AssertTrackingDésiré(S,G,I)==VRAI
On est intéressé par les assertions (S,G) parce que I est une interface en aval pour laquelle on a l'état de transmission (S,G) ou (*,G) ou parce que I est l'interface amont pour S et qu'on a l'état de transmission (S,G). L'assertion reçue a une meilleure métrique que la nôtre, si bien qu'on ne gagne pas le Assert. On transite à "Je suis le perdant de l'assertion" et on effectue les actions A6 (ci-dessous).

Transitions à partir de l'état "Je suis le gagnant de l'assertion".
Lorsque dans l'état "Je suis le gagnant de l'assertion", les événements suivants déclenchent les transitions :

Expiration du temporisateur d'assertions
Le temporisateur d'assertions (S,G) expire. Comme on est dans l'état Gagnant, on doit toujours avoir l'état de transmission (S,G) qui est activement gardé en vie. On renvoie l'Assert (S,G) et on relance le temporisateur d'assertions (actions A3 ci-dessous). Noter que le temporisateur d'assertions du gagnant de l'assertion est conçu pour arriver à expiration peu avant les temporisateurs sur les perdants d'assertion ; cela empêche les secousses inutiles chez l'émetteur et les inondations périodiques de paquets dupliqués.

Reçoit Assert inférieur
On reçoit un Assert(S,G) ou (*,G) mentionnant à S qu'il a une métrique moins bonne que la nôtre. Celui qui a envoyé l'assertion se trompe, et donc on renvoie un (S,G) Assert et on redémarre le temporisateur d'assertions (actions A3).

Reçoit Assert préféré
On reçoit un AssertS,G) qui a une meilleure métrique que la nôtre. On transite à l'état "Je suis le perdant de l'assertion" et on effectue les actions A2 (ci-dessous). Noter que cela peut affecter la valeur de JoinDesired(S,G) et PruneDésiré(S,G,rpt), ce qui peut causer des transitions dans les automates à états (S,G) ou (S,G,rpt) en amont.

CouldAssert(S,G,I) -> FAUX
Notre état de transmission (S,G) ou interface RPF a changé de façon telle que CouldAssert(S,G,I) devient faux. On ne peut plus effectuer les actions du gagnant de l'assertion, et donc on transite à l'état NoInfo et on effectue les actions A4. Cela inclut d'envoyer une "assertion d'annulation" avec une métrique infinie.

Transitions à partir de l'état "Je suis le perdant de l'assertion".
Lorsque dans l'état "Je suis le perdant de l'assertion", les transitions suivantes peuvent se produire :

Reçoit Assert préféré.
On reçoit une assertion qui est meilleure que celle du gagnant d'assertion actuel. On reste dans l'état Perdant et on effectue les actions A2.

Reçoit Assert acceptable avec le bit RPT à zéro du gagnant actuel.
On reçoit une assertion du gagnant actuel de l'assertion qui est meilleure que notre propre métrique pour ce (S,G) (bien que la métrique puisse être pire que la précédente métrique du gagnant). On reste dans l'état Perdant et on effectue les actions A2.

Reçoit Assert inférieur ou Assert annulation du gagnant actuel.
On reçoit une assertion du gagnant actuel d'assertion qui est pire que notre propre métrique pour ce groupe (normalement, parce que la métrique du gagnant est devenue pire ou parce que il y a une annulation d'assertion). On transite à l'état NoInfo, supprimant les informations de Assert(S,G) et permettant aux mécanismes normaux Join/Prune de PIM de fonctionner. Généralement, on va finalement faire une réassertion et gagner quand les paquets de données provenant de S recommencent à s'écouler.

Temporisateur d'assertion arrive à expiration.
Le temporisateur d'assertion (S,G) arrive à expiration. On transite à l'état NoInfo, et on supprime les informations d'assertion (S,G) (Actions A5).

Le GenID du gagnant actuel change ou NLT arrive à expiration.
Le temporisateur de vivacité de voisin (NLT, *Neighbor Liveness Timer*) associé au gagnant actuel arrive à expiration ou on reçoit un message Hello du gagnant actuel qui rapporte un GenID différent de celui qu'il rapportait précédemment. Cela indique que l'interface ou routeur du gagnant actuel est défaillant (et peut être revenu à l'activité) et donc on doit supposer qu'il ne sait plus qu'il était le gagnant. On transite à l'état NoInfo, supprimant ces informations d'assertion (S,G) (Actions A5).

AssertTrackingDesired(S,G,I)->FAUX

AssertTrackingDesired(S,G,I) devient FAUX. Notre état de transmission a changé de sorte que les assertions (S,G) sur l'interface I ne nous intéressent plus. On transite à l'état NoInfo, supprimant les informations d'assertion (S,G).

Ma métrique devient meilleure que celle du gagnant de l'assertion. my_assert_metric(S,G,I) a changé de telle sorte que maintenant ma métrique d'assertion pour (S,G) est meilleure que la métrique mémorisée pour le gagnant actuel de l'assertion. Cela peut arriver quand la métrique d'acheminement sous-jacente change, ou quand CouldAssert(S,G,I) devient vrai, par exemple, quand SPTbit(S,G) devient vrai. On transite à l'état NoInfo, on supprime cet état d'assertion (S,G) (Actions A5) et on permet aux mécanismes normaux PIM Join/Prune de fonctionner. Généralement, on va finalement réaffirmer et gagner quand les paquets de données provenant de S recommencent à s'écouler.

RPF_interface(S) arrête d'être l'interface I.
L'interface I est utilisée comme étant l'interface RPF pour S, et maintenant elle ne l'est plus. On transite à l'état NoInfo, et on supprime cet état d'assertion (S,G) (Actions A5).

Reçoit Join(S,G) sur l'interface I.
On reçoit un Join(S,G) qui a le champ d'adresse de voisin amont réglé à ma principale adresse IP sur l'interface I. L'action est de transiter à l'état NoInfo, de supprimer cet état d'assertion (S,G) (action A5) , et de permettre aux mécanismes normaux PIM Join/Prune de fonctionner. Si l'envoyeur du Join était dans l'erreur, alors le mécanisme normal d'assertion va finalement être appliqué à nouveau, et on perdra encore l'assertion. Cependant, l'envoyeur de l'assertion peut savoir que le précédent gagnant de l'assertion est mort, et on peut donc finir par être le nouveau transmetteur.

Actions de l'automate à états d'assertions (S,G) :

A1 : envoie Assert(S,G).
Régler le temporisateur d'assertions à (Assert_Time - Assert_Override_Interval).
Mémoriser cela comme AssertWinner(S,G,I).
Mémoriser spt_assert_metric(S,I) comme AssertWinnerMetric(S,G,I).

A2 : Mémoriser le nouveau gagnant d'assertion comme AssertWinner(S,G,I) et la métrique de gagnant d'assertion comme AssertWinnerMetric(S,G,I).
Régler le temporisateur d'assertions à Assert_Time.

A3 : envoi de Assert(S,G).
Régler le temporisateur d'assertions à (Assert_Time - Assert_Override_Interval).

A4 : envoi de AssertCancel(S,G).
Supprimer les informations d'assertion (AssertWinner(S,G,I) et AssertWinnerMetric(S,G,I) vont alors retourner à leurs valeurs par défaut).

A5 : Supprimer les informations d'assertion (AssertWinner(S,G,I) et AssertWinnerMetric(S,G,I) vont alors retourner à leurs valeurs par défaut).

A6: Mémoriser le nouveau gagnant d'assertion comme AssertWinner(S,G,I) et affirmer la métrique du gagnant comme AssertWinnerMetric(S,G,I).
Régler le temporisateur d'assertions à Assert_Time.
Si (I est RPF_interface(S)) ET (UpstreamJPState(S,G) ==Joined) régler SPTbit(S,G) à VRAI.

Noter que certaines de ces actions peuvent causer un changement de la valeur de JoinDesired(S,G), PruneDesired(S,G,rpt), ou RPF'(S,G) ce qui pourrait causer d'autres transitions dans d'autres automates à états.

4.6.2 Automate à états de message Assert (*,G)

L'automate à états (*,G) Assert pour l'interface I est montré à la Figure 9. Il y a trois états :

NoInfo (NI)
Ce routeur n'a pas affirmé d'état (*,G) sur l'interface I.

Je suis le gagnant de l'assertion (W)
Ce routeur a gagné l'assertion (*,G) sur l'interface I. Il est maintenant responsable de la transmission du trafic destiné à G sur l'interface I à l'exception du trafic pour lequel il a l'état (S,G) "Je suis le perdant de l'assertion". Sans considération de si il est le DR pour I, il est aussi responsable du traitement des demandes d'adhésion pour G provenant des hôtes locaux sur I.

Je suis le perdant de l'assertion (L)
Ce routeur a perdu une assertion (*,G) sur l'interface I. Il ne doit pas transmettre de paquets pour G sur l'interface I à l'exception du trafic provenant de sources pour lesquelles il a l'état (S,G) "Je suis le gagnant de l'assertion". Si il est le DR, il n'est plus responsable du traitement des demandes d'adhésion pour le groupe G provenant des hôtes locaux sur I.

De plus, il y a aussi un temporisateur d'assertions (AT, *Assert Timer*) qui est utilisé pour périmer les assertions sur les perdants d'assertions et pour renvoyer les assertions sur le gagnant de l'assertion.

Lorsque un message Assert est reçu avec une adresse de source autre que zéro, une mise en œuvre de PIM doit d'abord la confronter aux événements possibles dans l'automate à états (S,G) Assert et traiter toutes les transitions et actions, avant de considérer si le message Assert correspond à l'automate à états (*,G) Assert.

Il est important de noter que AUCUNE TRANSITION NE PEUT SE PRODUIRE dans l'automate à états (*,G) par suite de la réception d'un message Assert sauf si l'automate à états (S,G) Assert pour le S et G pertinent est dans l'état "NoInfo" après que l'automate à états (S,G) a traité le message. Aussi, AUCUNE TRANSITION NE PEUT SE PRODUIRE dans l'automate à états (*,G) par suite de la réception d'un message Assert si ce message déclenche un changement d'état dans l'automate à états (S,G). Évidemment, quand l'adresse de source dans le message reçu est réglée à zéro, un automate à états (S,G) pour S et G n'existe pas et peut être supposé être dans l'état "NoInfo".

Par exemple, si les deux automates à états (S,G) et (*,G) Assert sont dans l'état NoInfo quand un message Assert arrive, et si le message cause la transition de l'automate à états (S,G) à l'état "W" ou "L", l'assertion ne va pas être traitée par l'automate à états (*,G) Assert.

Autre exemple : si l'automate à états (S,G) Assert est dans l'état "L" quand un message Assert est reçu, et si la métrique d'assertion dans le message est pire que my_assert_metric(S,G,I), alors l'automate à états (S,G) Assert va transiter à l'état NoInfo. Dans ce cas, si l'automate à états (*,G) Assert était dans l'état NoInfo, il pourrait paraître qu'il devrait transiter à l'état "W", mais ce n'est pas le cas parce que ce message a déjà déclenché une transition dans l'automate à états (S,G) Assert.

Figure 9 : Automate à états par interface Assert (*,G)

État NoInfo (NI)		
Reçoit Assert inférieur avec bit RPT établi et CouldAssert(*,G,I)	Données arrivent pour G sur I et CouldAssert (*,G,I)	Reçoit Assert acceptable avec bit RPT établi et AssTrDes (*,G,I)
| -> état W [Actions A1]	-> état W [Actions A1]	-> état L [Actions A2]

Dans l'état Je suis le gagnant de Assert (W)			
AT expire	Reçoit Assert inférieur	Reçoit Assert préféré	CouldAssert (*,G,I) -> FAUX
-> état W [Actions A3]	-> état W [Actions A3]	-> état L [Actions A2]	-> état NI [Actions A4]

Dans l'état Je suis le perdant de Assert (L)				
Reçoit Assert préféré avec bit RPT établi	Reçoit Assert acceptable du gagnant actuel avec le bit RPT établi	Reçoit Assert inférieur ou Assert Cancel du gagnant actuel	AT expire	Le GenID du gagnant actuel change ou NLT expire
-> état L [Actions A2]	-> état L [Actions A2]	-> état NI [Actions A5]	-> état NI [Actions A5]	-> état NI [Actions A5]

Dans l'état Je suis le perdant de Assert (L)			
AssTrDes (*,G,I) -> FAUX	my_metric -> meilleure que métrique du gagnant	RPF_interface (RP(G)) arrête d'être I	Reçoit Join(*,G) sur interface I
-> état NI [Actions A5]	-> état NI [Actions A5]	-> état NI [Actions A5]	-> état NI [Actions A5]

L'automate à états utilise les macros suivantes :

```
CouldAssert(*,G,I) =
   ( I dans ( joins(*,G) (+) pim_include(*,G)) )
   ET (RPF_interface(RP(G)) != I)
```

CouldAssert(*,G,I) est vrai sur les interfaces en aval pour lesquelles on a l'état (*,G) Join, ou les membres locaux qui ont demandé du trafic destiné à G.

```
AssertTrackingDésiré(*,G,I) =
   CouldAssert(*,G,I)
```

```
OU (local_receiver_include(*,G,I)==VRAI
   ET (Je_suis_DR(I) OU AssertWinner(*,G,I) == moi))
OU (RPF_interface(RP(G)) == I ET RPTJoinDesired(G))
```

AssertTrackingDesired(*,G,I) est vrai sur toute interface sur laquelle un (*,G) Assert peut affecter le comportement du routeur sur cette interface.

Noter que pour des raisons de compacité, "AssTrDes(*,G,I)" est utilisé dans l'automate à états pour se référer à AssertTrackingDesired(*,G,I).

Terminologie :

Une "assertion préférée" est celle qui a une meilleure métrique que le gagnant actuel.

Une "assertion acceptable" est celle qui a une meilleure métrique que my_assert_metric(*,G,I). Une assertion n'est jamais considérée comme acceptable si sa métrique est infinie.

Une "assertion inférieure" est celle qui a une métrique pire que my_assert_metric(*,G,I). Une assertion n'est jamais considérée comme inférieure si my_assert_metric(*,G,I) est infinie.

Transitions à partir de l'état NoInfo

Lorsque on est dans l'état NoInfo, les événements suivants déclenchent des transitions, mais seulement si l'automate à états (S,G) Assert est dans l'état NoInfo avant et après considération du message reçu :

Reçoit Assert inférieur avec le bit RPT établi ET CouldAssert(*,G,I)==VRAI
Une assertion inférieure (*,G) est reçue pour G sur l'interface I. Si CouldAssert(*,G,I) est VRAI, alors I est notre interface en aval, et on a l'état de transmission (*,G) sur cette interface, donc on devrait être le gagnant de l'assertion. On transite à l'état "Je suis le gagnant de l'assertion" et on effectue les actions A1 (ci-dessous).

Un paquet de données destiné à G arrive sur l'interface I, ET CouldAssert(*,G,I)==VRAI
Un paquet de données destiné à G est arrivé sur l'interface aval qui est dans notre liste d'interfaces sortantes (*,G). On croit donc qu'on devrait être le transmetteur pour ce (*,G), et on transite à l'état "Je suis le gagnant de l'assertion" et on effectue les actions A1 (ci-dessous).

Reçoit Assert acceptable avec le bit RPT établi ET AssertTrackingDesired(*,G,I)==VRAI
On est intéressé aux assertions (*,G) soit parce que I est l'interface aval pour laquelle on a l'état de transmission (*,G), soit parce que I est l'interface amont pour RP(G) et qu'on a l'état de transmission (*,G) . On reçoit une assertion (*,G) qui a une meilleure métrique que la nôtre, de sorte qu'on ne gagne pas Assert. On transite à "Je suis le perdant de l'assertion" et on effectue les actions A2 (ci-dessous).

Transitions à partir de l'état "Je suis le gagnant de l'assertion"

Lorsque dans l'état "Je suis le gagnant de l'assertion", les événements suivants déclenchent des transitions, mais seulement si l'automate à états (S,G) Assert est dans l'état NoInfo avant et après considération du message reçu :

Reçoit Assert inférieur
On reçoit une assertion (*,G) qui a une plus mauvaise métrique que la nôtre. Celui qui a envoyé l'assertion a perdu, et donc on renvoie un (*,G) Assert et on relance le temporisateur d'assertion (Actions A3 ci-dessous).

Reçoit Assert préféré
On reçoit une assertion (*,G) qui a une meilleure métrique que la nôtre. On transite à l'état "Je suis le perdant de l'assertion" et on effectue les actions A2 (ci-dessous).

Lorsque dans l'état "Je suis le gagnant de l'assertion", les événements suivants déclenchent des transitions :

Expiration du temporisateur d'assertion
Le temporisateur d'assertion (*,G) arrive à expiration. Comme on est dans l'état gagnant, on doit alors toujours avoir l'état de transmission (*,G) qui est activement gardé en vie. Pour empêcher d'inutiles oscillations du transmetteur et l'inondation périodique de paquets dupliqués, on envoie à nouveau le (*,G) Assert et on relance le temporisateur d'assertions (Actions A3).

CouldAssert(*,G,I) -> FAUX

Notre état de transmission (*,G) ou l'interface RPF a changé de sorte que CouldAssert(*,G,I) est devenu faux. On ne peut plus effectuer les actions du gagnant de l'assertion, et donc on transite à l'état NoInfo et on effectue les actions A4 (ci-dessous).

Transitions à partir de l'état "Je suis le perdant de l'assertion"

Lorsque dans l'état "Je suis le perdant de l'assertion", les événements suivants déclenchent des transitions, mais seulement si l'automate à états (S,G) Assert est dans l'état NoInfo avant et après considération du message reçu :

Reçoit Assert préféré avec le bit RPT établi
On reçoit une assertion (*,G) qui est meilleure que l'assertion du gagnant actuel. On reste dans l'état perdant et on effectue les actions A2 ci-dessous.

Reçoit Assert acceptable du gagnant actuel avec le bit RPT établi
On reçoit une assertion (*,G) du gagnant actuel de l'assertion qui est meilleure que notre propre métrique pour ce groupe (bien que la métrique puisse être pire que la métrique précédante du gagnant). On reste dans l'état perdant et on effectue les actions A2 ci-dessous.

Reçoit Assert inférieur ou Assert Cancel du gagnant actuel
On reçoit une assertion du gagnant actuel de l'assertion qui est pire que notre propre métrique pour ce groupe (normalement parce que la métrique du gagnant est devenue pire ou est maintenant une assertion annulée). On transite à l'état NoInfo, on supprime cet état (*,G) Assert (actions A5) et on permet au mécanisme normal PIM Join/Prune d'opérer. Généralement, on va finalement refaire une assertion et gagner quand les paquets de données pour G recommencent à s'écouler.

Lorsque dans l'état "Je suis le perdant de l'assertion", les événements suivants déclenchent des transitions :

Expiration du temporisateur d'assertion (AT)
AT (*,G) expire. On passe à l'état NoInfo et on supprime ces informations de (*,G) (actions A5).

Le GenID du gagnant actuel change ou NLT expire.
Le temporisateur de vivacité de voisin (NLT, *Neighbor Liveness Timer*) associé au gagnant actuel expire, ou on reçoit un message Hello du gagnant actuel rapportant un GenID différent de celui rapporté précédemment. Cela indique que l'interface ou routeur du gagnant actuel a une défaillance (et peut avoir récupéré) et on doit donc supposer qu'il ne sait plus qu'il était le gagnant. On passe à l'état NoInfo, en supprimant les informations de (*,G) Assert (actions A5).

AssertTrackingDesired(*,G,I)->FAUX
AssertTrackingDesired(*,G,I) devient FAUX. Notre état de transmission a changé de sorte que les assertions (*,G) sur l'interface I ne nous intéressent plus. On transite à l'état NoInfo et on supprime ces informations de (*,G) Assert (actions A5).

Ma métrique devient meilleure que celle du gagnant de l'assertion
Ma métrique d'acheminement, rpt_assert_metric(G,I), a changé de sorte que maintenant ma métrique d'assertion pour (*,G) est meilleure que la métrique qu'on avait mémorisée pour le gagnant actuel de l'assertion. On transite à l'état NoInfo, on supprime cet état d'assertion (*,G) (actions A5), et on permet aux mécanismes normaux PIM Join/Prune de fonctionner. Généralement, on va finalement refaire une assertion et gagner quand les paquets de données pour G recommencent à s'écouler.

RPF_interface(RP(G)) arrête d'être l'interface I
L'interface I était utilisée comme étant l'interface RPF pour le RP(G), et maintenant elle ne l'est plus. On transite à l'état NoInfo et on supprime cet état (*,G) Assert (actions A5).

Reçoit Join(*,G) sur l'interface I
On reçoit un Join(*,G) qui a le champ Adresse de voisin amont réglé à ma principale adresse IP sur l'interface I. L'action est de passer à l'état NoInfo, de supprimer cet état (*,G) Assert (actions A5), et de permettre aux mécanismes normaux PIM Join/Prune de fonctionner. Si quelqu'un a envoyé le Join par erreur, alors le mécanisme normal d'assertion va finalement se ré-appliquer, et on va perdre encore l'assertion. Cependant, celui qui a envoyé l'assertion peut savoir que le précédant gagnant de l'assertion est mort, et donc on peut finir par être le nouveau transmetteur.

Actions de l'automate à états (*,G) Assert :

A1 : Envoyer Assert(*,G).
Régler le temporisateur d'assertions à (Assert_Time - Assert_Override_Interval).
Se mémoriser comme AssertWinner(*,G,I).
Mémoriser rpt_assert_metric(G,I) comme AssertWinnerMetric(*,G,I).

A2 : Mémoriset le nouveau gagnant de l'assertion comme AssertWinner(*,G,I) et la métrique de gagnant d'assertion comme AssertWinnerMetric(*,G,I).
Régler le temporisateur d'assertions à Assert_Time.

A3 : Envoyer Assert(*,G).
Régler le temporisateur d'assertions à (Assert_Time - Assert_Override_Interval).

A4 : Envoyer AssertCancel(*,G).
Supprimer les informations d'assertion (AssertWinner(*,G,I) et AssertWinnerMetric(*,G,I) vont alors retourner à leurs valeurs par défaut).

A5 : Supprimer les informations d'assertion (AssertWinner(*,G,I) et AssertWinnerMetric(*,G,I) vont alors retourner à leurs valeurs par défaut).

Noter que certaines de ces actions peuvent causer le changement de la valeur de JoinDesired(*,G) ou RPF'(*,G) ce qui pourrait causer d'autres transitions dans d'autres automates à états.

4.6.3 Métrique de Assert

Les métriques de Assert sont définies comme :

```
struct assert_metric {
  rpt_bit_flag;
  metric_preference;
  route_metric;
  ip_address;
};
```

Lorsque on compare les assert_metrics, les champs rpt_bit_flag, metric_preference, et route_metric sont comparés dans cet ordre, où la première valeur la plus faible gagne. Si tous les champs sont égaux, la principale adresse IP du routeur qui est la source du message Assert est utilisée comme départage, la plus forte adresse IP étant la gagnante.

Une métrique d'assertion pour (S,G) à inclure dans (ou comparer avec) un message Assert envoyé sur l'interface I devrait être calculée en utilisant le pseudo code suivant :

```
assert_metric
my_assert_metric(S,G,I) {
   si( CouldAssert(S,G,I) == VRAI ) {
      retourner spt_assert_metric(S,I)
   } autrement si( CouldAssert(*,G,I) == VRAI ) {
      retourner rpt_assert_metric(G,I)
   } autrement {
      retourner infinite_assert_metric()
   }
}
```

spt_assert_metric(S,I) donne la métrique d'assertion qu'on utilise si on envoie une assertion sur la base de l'état de transmission (S,G) actif :

```
assert_metric
spt_assert_metric(S,I) {
  retourner {0,MRIB.pref(S),MRIB.metric(S),my_ip_address(I)}
}
```

rpt_assert_metric(G,I) donne la métrique d'assertion qu'on utilise si on envoie une assertion sur la base seulement de l'état de transmission (*,G) :

```
assert_metric
rpt_assert_metric(G,I) {
   retourner {1,MRIB.pref(RP(G)),MRIB.metric(RP(G)),my_ip_address(I)}
}
```

MRIB.pref(X) et MRIB.metric(X) sont la préférence d'acheminement et les métriques d'acheminement associées au chemin pour une destination X particulière (en envoi individuel), comme déterminé par la MRIB. my_ip_address(I) est simplement l'adresse IP principale du routeur qui est associée à l'interface I locale.

infinite_assert_metric() est une métrique d'assertion que le routeur utilise pour un Assert qui ne correspond pas à l'état de transmission (S,G) ou (*,G) :

```
assert_metric
infinite_assert_metric() {
   retourner {1,infinity,infinity,0}
}
```

4.6.4 Messages AssertCancel

Un message AssertCancel est simplement un message Assert RPT mais avec une métrique infinie. Il est envoyé par le gagnant de l'assertion quand il supprime l'état de transmission qui a causé la production de l'assertion. Les autres routeurs vont voir cette métrique, et cela va causer l'envoi par tout autre routeur qui a l'état de transmission de sa propre assertion, et de prendre la transmission.

Un AssertCancel(S,G) est une métrique d'assertion infinie avec le bit RPT établi qui désigne S comme la source.

Un AssertCancel(*,G) est une métrique d'assertion infinie avec le bit RPT établi et la source réglée à zéro.

Les messages AssertCancel sont simplement une optimisation. Le mécanisme de fin de temporisation du Assert d'origine va permettre à un sous réseau de devenir finalement cohérent ; le mécanisme AssertCancel cause simplement une convergence plus rapide. Aucun traitement particulier n'est exigé pour un message AssertCancel, car il est simplement un message Assert du gagnant actuel.

4.6.5 Macros d'état Assert

Les macros lost_assert(S,G,rpt,I), lost_assert(S,G,I), et lost_assert(*,G,I) sont utilisées dans les calculs de olist du paragraphe 4.1 et sont définies comme :

```
bool lost_assert(S,G,rpt,I) {
  si ( RPF_interface(RP(G)) == I OU
       ( RPF_interface(S) == I ET SPTbit(S,G) == VRAI ) ) {
    retourner FAUX
  } autrement {
    retourner ( AssertWinner(S,G,I) != NUL ET
          AssertWinner(S,G,I) != moi )
  }
}

bool lost_assert(S,G,I) {
  si ( RPF_interface(S) == I ) {
    retourner FAUX
  } autrement {
    retourner ( AssertWinner(S,G,I) != NUL ET
          AssertWinner(S,G,I) != moi  ET
          (AssertWinnerMetric(S,G,I) est meilleur que spt_assert_metric(S,I) )
  }
}
```

Note : le terme "AssertWinnerMetric(S,G,I) est meilleur que spt_assert_metric(S,I)" est exigé pour traiter correctement la phase de transition quand un routeur a l'état (S,G) Join mais n'a pas encore le bit SPT établi. Dans ce cas, il doit ignorer l'état d'assertion si il veut gagner l'assertion une fois le bit SPT établi.

```
bool lost_assert(*,G,I) {
  si ( RPF_interface(RP(G)) == I ) {
    retourner FAUX
  } autrement {
    retourner ( AssertWinner(*,G,I) != NUL ET AssertWinner(*,G,I) != moi )
```

```
  }
}
```

AssertWinner(S,G,I) est l'adresse IP de source du paquet Assert(S,G) qui a gagné l'assertion.

AssertWinner(*,G,I) est l'adresse IP de source du paquet Assert(*,G) qui a gagné une assertion.

AssertWinnerMetric(S,G,I) est la métrique d'assertion du paquet Assert(S,G) qui a gagné une assertion.

AssertWinnerMetric(*,G,I) est la métrique d'assertion du paquet Assert(*,G) qui a gagné une assertion.

AssertWinner(S,G,I) est NUL par défaut et AssertWinnerMetric(S,G,I) est Infini par défaut dans l'état NoInfo.

Résumé des règles et raisons de Assert.

Cette section résume les règles clés pour l'envoi des assertions et des réactions aux assertions et les raisons de ces règles. Cette section n'est pas destinée à être et ne devrait pas être traitée comme une spécification définitive du comportement du protocole. Les automates à états et le pseudo code devraient être consultés pour cela. Cette section est plutôt destinée à documenter les aspects importants du comportement du protocole Assert et à fournir des informations qui pourraient se révéler utiles au lecteur pour comprendre en mettre en œuvre cette partie du protocole.

1. Comportement : Les voisins en aval envoient des messages périodiques Join(*,G) et Join(S,G) au voisin RPF' approprié, c'est-à-dire, le voisin RPF comme modifié par le processus d'assertion. Ils ne sont pas toujours envoyés au voisin RPF indiqué par la MRIB. Les règles normales de suppression et d'outrepassement s'appliquent.

Raison : en envoyant les messages Join périodiques et déclenchés au voisin RPF' au lieu du voisin RPF, le routeur en aval évite de re-déclencher le processus Assert avec chaque Join. Un effet collatéral de l'envoi des Join au gagnant de Assert est que le trafic ne va pas revenir au voisin RPF "normal" jusqu'à ce que le Assert arrive en fin de temporisation. Cela ne va pas arriver avant que les données arrêtent de s'écouler, si le point 8, ci-dessous, est mis en œuvre.

2. Comportement : le gagnant d'assertion pour (*,G) agit comme DR local pour (*,G) au nom des membres de IGMP/MLD.

Raison : ceci est exigé pour permettre à un seul routeur de fusionner les adhésions et départs PIM et IGMP/MLD. Sans cela, l'outrepassement ne fonctionne pas.

3. Comportement : le gagnant d'assertion pour (S,G) agit comme DR local pour (S,G) au nom des membres IGMPv3.

Raison : même raison qu'au point 2.

4. Comportement : les outrepassements d'élagage (S,G) et (*,G) sont envoyés au voisin RPF' et non au voisin RPF régulier.

Raison : même raison qu'au point 1.

5. Comportement : un outrepassement d'élagage (S,G,rpt) n'est pas envoyé (du tout) si RPF'(S,G,rpt) != RPF'(*,G).

Raison : cela évite de garder l'état en vie sur l'arborescence (S,G) quand il reste seulement des membres (*,G) en aval. Aussi, cela évite l'envoi de Join(S,G,rpt) à un routeur qui n'est plus sur l'arborescence (*,G). Ce comportement peut amener à confusion, bien que la présente spécification indique bien qu'un tel Join DEVRAIT être éliminé.

6. Comportement : un perdant d'assertion qui reçoit un Join(S,G) avec une adresse de voisin amont qui est sa principale adresse IP sur cette interface termine le temporisateur d'assertion (S,G).

Raison : ceci est nécessaire afin d'avoir une rapide convergence dans le cas où le routeur en aval qui a initialement envoyé un join au gagnant d'assertion précédent a entrepris un changement de topologie.

7. Comportement : un perdant d'assertion qui reçoit un Join(*,G) avec une adresse de voisin amont qui est sa principale adresse IP sur cet interface amène à expiration le temporisateur d'assertion (*,G) et tous les temporisateurs d'assertion (S,G) qui n'ont pas de messages Prune(S,G,rpt) correspondants dans le message Join/Prune composé.

Raison : même raison qu'au point 6.

8. Comportement : un gagnant d'assertion pour (*,G) ou (S,G) envoie une assertion d'annulation quand il est sur le point d'arrêter de transmettre sur une entrée (*,G) ou (S,G). Ce comportement ne s'applique pas au (S,G,rpt).

Raison : cela permet de revenir à l'arborescence partagée après le départ du dernier routeur SPT sur le LAN. Faire ainsi empêche les routeurs en aval sur l'arborescence partagée de garder l'état de SPT en vie.

9. Comportement : renvoi des messages d'assertion avant la fin de temporisation d'une assertion. (Ce comportement est facultatif.)

Raison : cela empêche les dupliqués périodiques qui se produiraient autrement chaque fois qu'une assertion arrive en fin de temporisation et est ensuite rétablie.

10. Comportement : lorsque RPF'(S,G,rpt) change pour être le même que RPF'(*,G), on doit déclencher un Join(S,G,rpt) pour RPF'(*,G).

Raison : cela permet de revenir au RPT après le départ du dernier membre de la SPT.

4.7 Bootstrap PIM et découverte de RP

Pour un fonctionnement correct, chaque routeur PIM au sein d'un domaine PIM doit être capable de transposer une certaine adresse de groupe de diffusion groupée dans le même RP. Si ce n'est pas le cas, des trous noirs peuvent alors apparaître, où des receveurs dans le domaine ne peuvent pas recevoir certains groupes. Un domaine dans ce contexte est un ensemble contigu de routeurs qui mettent tous en œuvre PIM et sont configurés pour opérer dans des limites communes.

Une exception notable à cela est lorsque un domaine PIM est coupé en plusieurs régions de portée administratives ; ce sont des régions où une frontière a été configurée afin qu'une gamme de groupes de diffusion groupée ne soient pas transmis à travers cette frontière. Pour plus d'informations sur la diffusion groupée IP limitée administrativement, voir la RFC 2365. Les critères modifiés des régions limitées administrativement sont que la région est convexe par rapport à la transmission fondée sur la MRIB, et que tous les routeurs PIM dans la région limitée transposent les groupes limités au même RP dans cette région.

La présente spécification ne rend pas obligatoire l'utilisation d'un seul mécanisme pour fournir aux routeurs les informations pour effectuer la transposition de groupe en RP. Actuellement, quatre mécanismes sont possibles, et tous les quatre ont des problèmes associés :

Configuration statique : un routeur PIM DOIT prendre en charge la configuration statique de transposition de groupe en RP. Ce mécanisme n'est pas robuste aux défaillances mais fournit au moins un mécanisme d'interopérabilité de base.

RP incorporé : cela définit une politique d'allocation d'adresse dans laquelle l'adresse du point de rendez-vous (RP) est codée dans une adresse IPv6 de groupe de diffusion groupée [RFC3956].

Auto-RP de Cisco : Auto-RP utilise un groupe de diffusion groupée PIM en mode dense (PIM-DM) pour annoncer les transpositions de groupe en RP à partir d'une localisation centrale. Ce mécanisme n'est pas utile si le mode PIM dense ne fonctionne pas en parallèle avec le mode PIM épars ; il était seulement destiné à être utilisé avec PIM en mode épars version 1. Aucune spécification standard n'existe actuellement.

Routeur Bootstrap (BSR) : la RFC 2362 spécifie un mécanisme “bootstrap” fondé sur l'élection automatique d'un BSR. Tout routeur dans le domaine qui est configuré à être un RP possible rapporte sa candidature au BSR, et ensuite un mécanisme d'arrosage à l'échelle du domaine distribue l'ensemble de RP choisi par le BSR dans tout le domaine. Comme spécifié dans la RFC 2362, le mécanisme de BSR est fautif par son traitement des régions limitées administrativement qui sont plus petites qu'un domaine PIM, mais le mécanisme fonctionne pour les groupes de portée non limitée.

Pour ce qui concerne PIM-SM, la seule exigence importante est que tous les routeurs dans le domaine (ou zone limitée administrativement pour les régions à limitation) reçoivent le même ensemble de transpositions de gamme de groupe à RP. Ceci peut être réalisé par l'utilisation d'un de ces mécanismes, ou par d'autres mécanismes non spécifiés actuellement.

Il doit être assuré au niveau du fonctionnement que toute adresse de RP configurée, apprise, ou annoncée est accessible à partir de tous les routeurs dans le domaine PIM.

4.7.1 Transposition de groupe à RP

En utilisant un des mécanismes décrits ci-dessus, un routeur PIM reçoit une ou plusieurs transpositions possibles de gamme de groupe à RP. Chaque transposition spécifie une gamme de groupes de diffusion groupée (exprimée par un groupe et un gabarit) et le RP auquel ces groupes devraient se transposer. Chaque transposition peut aussi avoir une priorité associée. Il est possible de recevoir plusieurs transpositions, dont toutes vont correspondre au même groupe de diffusion groupée ; c'est le cas courant avec le mécanisme de BSR. L'algorithme pour effectuer la transposition de groupe en RP est le suivant :

1. Effectuer la plus longue correspondance sur la gamme de groupes pour obtenir une liste des RP.

2. À partir de cette liste de RP correspondants, trouver ceux de plus haute priorité. Éliminer tous les RP de la liste qui ont les moindres priorités.

3. Si un seul RP reste dans la liste, utiliser ce RP.

4. Si plusieurs RP restent dans la liste, utiliser la fonction de hachage PIM pour en choisir un.

Donc, si deux transpositions de gamme de groupe en RP ou plus couvrent un groupe particulier, celui qui a le plus long gabarit est à utiliser. Si les transpositions ont la même longueur de gabarit, alors celui qui a la plus haute priorité est choisi. Si il y a plus d'une entrée qui correspond avec la même longueur de gabarit et si les priorités sont identiques, une fonction de hachage (voir au paragraphe 4.7.2) est alors appliquée pour choisir le RP.

Cet algorithme est invoqué par un DR quand il a besoin de déterminer un RP pour un certain groupe, par exemple, à réception d'un paquet ou d'une indication de membre IGMP/MLD pour un groupe pour lequel le DR ne connaît pas le RP.

De plus, la fonction de transposition est invoquée par tous les routeurs à réception d'un message Join/Prune (*,G).

Noter que si l'ensemble des transpositions possibles de gamme de groupe à RP change, chaque routeur va avoir besoin de vérifier si des groupes existants sont affectés. Cela peut, par exemple, faire qu'un DR ou agissant comme DR rejoigne un groupe, ou faire qu'il recommence l'encapsulation d'enregistrement au nouveau RP.

Note de mise en œuvre : le mécanisme bootstrap décrit dans la RFC 2362 omettait l'étape 1 ci-dessus. Cependant, des mises en œuvre connues, approximativement la moitié effectuent de toutes façons l'étape 1. Noter que les mises en œuvre de BSR qui omettent l'étape 1 ne vont pas interopérer correctement avec les mises en œuvre de la présente spécification quand elles sont utilisées avec le mécanisme de BSR décrit dans la [RFC5059].

4.7.2 Fonction de hachage

La fonction de hachage est utilisée par tous les routeurs au sein d'un domaine, pour transposer un groupe en un des RP de l'ensemble des transpositions correspondantes de gamme de groupes en RP (les transpositions de cet ensemble ont toutes la même plus grande longueur de gabarit et le même niveau de priorité). L'algorithme prend en entrée l'adresse de groupe, et les adresses des RP candidats provenant des transpositions, et donne en résultat une adresse de RP à utiliser.

Le protocole exige que tous les routeurs hachent le même RP au sein d'un domaine (sauf les transitoires). La fonction de hachage suivante doit être utilisée dans chaque routeur :

1. Pour les adresses de RP dans les transpositions correspondantes de gamme de groupes à RP, on calcule une valeur :

   Valeur(G,M,C(i))= (1103515245 * ((1103515245 * (G&M)+12345) OUX C(i)) + 12345) mod 2^31

   où C(i) est l'adresse de RP et M est un gabarit de hachage. Si BSR est utilisé, le gabarit de hachage est donné dans les messages Bootstrap. Si BSR n'est pas utilisé, le mécanisme de remplacement qui fournit les transpositions de gamme de groupes à RP peut fournir la valeur, ou autrement il donne par défaut un gabarit avec les 30 bits de poids fort à un pour IPv4 et les 126 bits de poids fort à un pour IPv6. Le gabarit de hachage permet à un petit nombre de groupes consécutifs (par exemple, 4) de toujours donner le même RP. Par exemple, des données codées hiérarchiquement peuvent être envoyées sur des adresses de groupe consécutives pour obtenir les mêmes caractéristiques de délai et de partage de sort.

   Pour les familles d'adresses autres que IPv4, un résumé de 32 bits va être utilisé car C(i) et G doivent d'abord être déduits du RP ou adresse de groupe actuel. Cette méthode de résumé doit être utilisée de façon cohérente à travers le domaine PIM. Pour les adresses IPv6, il est RECOMMANDÉ d'utiliser l'adresse IPv4 équivalente pour une adresse compatible IPv4, et le OU exclusif de chaque segment de 32 bits de l'adresse pour toutes les autres adresses IPv6. Par exemple, le résumé de l'adresse IPv6 3ffe:b00:c18:1::10 serait calculé comme 0x3ffe0b00 ^ 0x0c180001 ^ 0x00000000 ^ 0x00000010, où le symbole '^' représente l'opréation OU exclusif.

2. Le RP candidat avec la plus forte valeur de hachage résultante est alors le RP choisi par cette fonction de hachage. Si plus d'un RP a la même plus forte valeur de hachage, le RP avec la plus forte adresse IP est choisi.

4.8 Diffusion groupée spécifique de source

Le modèle de service de diffusion groupée spécifique de source (SSM, *Source-Specific Multicast*) [RFC4607] peut être mis en œuvre avec un sous ensemble strict de mécanismes du protocole PIM-SM. Les sémantiques de la diffusion IP régulière et de SSM peuvent coexister sur un seul routeur, et les deux peuvent être mises en œuvre en utilisant le protocole PIM-SM. Une gamme d'adresses de diffusion groupée, actuellement 232.0.0.0/8 dans IPv4 et ff3x::/32 pour IPv6, est réservée pour SSM, et le choix de la sémantique est déterminé par l'adresse du groupe de diffusion groupée dans les paquets de données et les messages PIM.

4.8.1 Modifications du protocole pour les adresses de destination SSM

Le règles suivantes outrepassent le comportement normal PIM-SM pour une adresse de diffusion groupée G dans la gamme SSM :

- un routeur NE DOIT PAS envoyer de message (*,G) Join/Prune pour quelque raison que ce soit.
- un routeur NE DOIT PAS envoyer de message (S,G,rpt) Join/Prune pour quelque raison que ce soit.
- un routeur NE DOIT PAS envoyer de message Register pour un paquet destiné à une adresse SSM.
- un routeur NE DOIT PAS transmettre de paquets sur la base de l'état (*,G) ou (S,G,rpt). Les macros de résumé d'état en rapport avec (*,G) et (S,G,rpt) sont NUL pour toute adresse SSM, pour les besoins de la transmission de paquet.
- un routeur agissant comme RP NE DOIT PAS transmettre de paquet encapsulé dans Register qui a une adresse de destination SSM et DEVRAIT répondre par un message Register-Stop à un tel message Register.
- un routeur PEUT optimiser la création et la maintenance des états (S,G,rpt) et (*,G) pour les adresses de destination SSM -- cet état n'est pas nécessaire pour les paquets SSM.

Les trois dernières règles sont présentes pour les routeurs "traditionnels" sans capacité SSM qui peuvent envoyer des (*,G) et (S,G,rpt) Join/Prune, ou des messages Register pour des adresses de destination SSM. Noter que la présente spécification ne tente pas d'aider un routeur "traditionnel" sans capacité SSM à fonctionner avec SSM.

4.8.2 Routeurs seulement PIM-SSM

On peut choisir de ne mettre en œuvre que le sous ensemble de mode PIM épars qui fournit la sémantique de transmission SSM.

Un routeur seulement PIM-SSM DOIT mettre en œuvre les portions suivantes de la présente spécification :

- automate à états (S,G) amont (paragraphe 4.5.5)
- automate à états (S,G) aval (paragraphe 4.5.2)
- automate à états (S,G) Assert (paragraphe 4.6.1)
- messages Hello, découverte de voisin, et élection de DR (paragraphe 4.3)
- règles de transmission de paquet (paragraphe 4.2)

Un routeur seulement PIM-SSM n'a pas besoin de mettre en œuvre les éléments de protocole suivants :

- automate à états Register (paragraphe 4.4)
- automate à états avals (*,G) et (S,G,rpt) (paragraphes 4.5.1 et 4.5.3)
- automates à états amont (*,G) et (S,G,rpt) (paragraphes 4.5.4, 4.5.6, et 4.5.7)
- automate à états (*,G) Assert (paragraphe 4.6.2)
- élection de RP Bootstrap (paragraphe 4.7)
- Temporisateur de maintien en vie
- bit SPT (paragraphe 4.2.2)

Le temporisateur de maintien en vie devrait être traité comme fonctionnant toujours, et le bit SPT devrait être traité comme toujours établi pour une adresse SSM. De plus, les règles de transmission de paquet du paragraphe 4.2 peuvent être simplifiées dans un routeur seulement PIM-SSM :

```
oiflist = NUL
si( iif == RPF_interface(S) ET UpstreamJPState(S,G) == Joined ) {
   oiflist = inherited_olist(S,G)
} autrement si( iif est dans inherited_olist(S,G) ) {
   send Assert(S,G) on iif
}

oiflist = oiflist (-) iif
transmettre le paquet sur toutes les interfaces dans oiflist
```

Ce n'est rien de plus que la réduction de la règle normale de transmission PIM-SM, avec toutes les clauses (S,G,rpt) et (*,G) remplacées par NUL.

4.9 Formats de paquet PIM

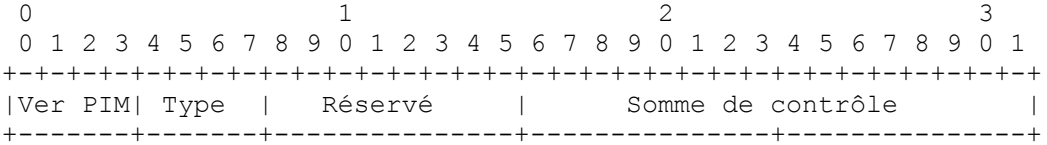
Ce paragraphe décrit les détails des formats de paquet pour les messages de contrôle PIM.

Tous les messages de contrôle PIM ont le numéro de protocole IP de 103.

Les messages PIM sont soit en envoi individuel (par exemple, les Register et Register-Stop) soit en diffusion groupée avec le TTL 1 au groupe “TOUS-LES-ROUTEURS-PIM” (par exemple, Join/Prune, Assert). L'adresse de source utilisée pour les messages en envoi individuel est une adresse accessible sur tout le domaine ; l'adresse de source utilisée pour les messages en diffusion groupée est l'adresse de liaison locale de l'interface sur laquelle le message est envoyé.

Le groupe IPv4 “TOUS-LES-ROUTEURS-PIM” est "224.0.0.13". Le groupe IPv6 “TOUS-LES-ROUTEURS-PIM” est "ff02::d".

L'en-tête PIM commun à tous les messages PIM est :



Ver PIM : le numéro de version PIM est 2.

Type : types des messages PIM spécifiques. Les types PIM sont :

Type de message	**Destination**
0 = Hello	Diffusion groupée à TOUS-LES-ROUTEURS-PIM
1 = Register	Envoi individuel au RP
2 = Register-Stop	Envoi individuel à la source du paquet Register
3 = Join/Prune	Diffusion groupée à TOUS-LES-ROUTEURS-PIM
4 = Bootstrap	Diffusion groupée à TOUS-LES-ROUTEURS-PIM
5 = Assert	Diffusion groupée à TOUS-LES-ROUTEURS-PIM
6 = Graft (utilisé seulement dans PIM-DM)	Envoi individuel au RPF'(S)
7 = Graft-Ack (utilisé seulement dans PIM-DM)	Envoi individuel à la source du paquet Graft
8 = Annonce de candidat RP	Envoi individuel au BSR du domaine

Réservé : réglé à zéro à l'émission. Ignoré à réception.

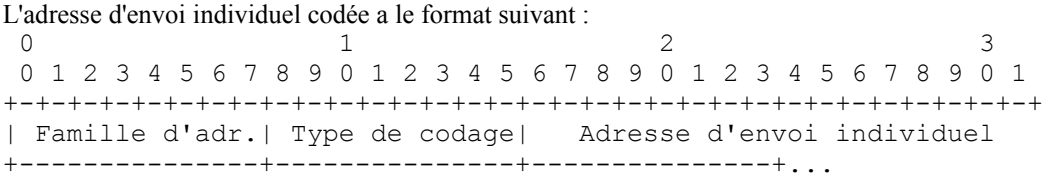
Somme de contrôle : c'est une somme de contrôle IP standard, c'est-à-dire, les 16 bits du complément à un de la somme des compléments à un du message PIM entier, à l'exclusion de la section "paquet de données en diffusion groupée" du message Register. Pour calculer la somme de contrôle, le champ Somme de contrôle est mis à zéro. Si la longueur du paquet n'est pas un nombre entier de mots de 16 bits, le paquet est bourré avec un octet de zéros en queue avant d'effectuer la somme de contrôle.

Pour IPv6, la somme de contrôle inclut aussi le "pseudo en-tête" IPv6, comme spécifié au paragraphe 8.1 de la [RFC2460]. Ce "pseudo en-tête" est pré ajouté à l'en-tête PIM pour les besoins du calcul de la somme de contrôle. Le champ "Longueur de paquet de couche supérieure" dans le pseudo en-tête est réglé à la longueur du message PIM, sauf dans les messages Register où il est réglé à la longueur de l'en-tête PIM Register (8). La valeur de Prochain en-tête utilisée dans le pseudo en-tête est 103.

Si un message est reçu avec un champ Version PIM ou Type non reconnu, ou si la destination d'un message ne correspond pas au tableau ci-dessus, le message DOIT être éliminé, et un message d'erreur DEVRAIT être enregistré pour l'administrateur en débit limité.

4.9.1 Formats de source codée et d'adresse de groupe

L'adresse d'envoi individuel codée a le format suivant :

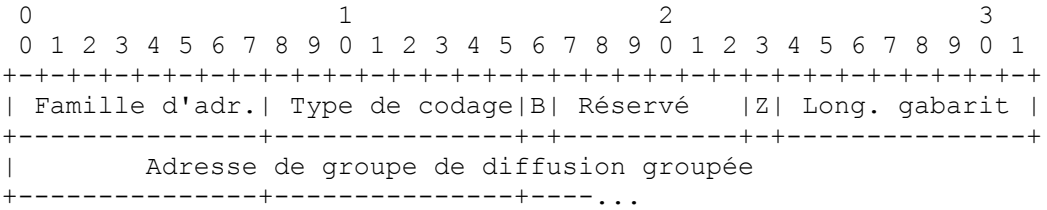


Famille d'adresses : famille d'adresses PIM du champ "Adresse d'envoi individuel" de cette adresse. Les valeurs 0-127 sont celles allouées par l'IANA pour les familles d'adresses Internet dans [IANA]. Les valeurs 128-250 sont réservées pour être allouées par l'IANA pour les familles d'adresses spécifiques de PIM. Les valeurs 251 à 255 sont destinées à l'usage privé. Comme il n'y a pas d'autorité d'allocation pour cet espace, on devrait s'attendre à des collisions.

Type de codage : le type de codage utilisé dans une famille d'adresses spécifique. La valeur '0' est réservée pour ce champ et représente le codage natif de la famille d'adresses.

Adresse d'envoi individuel : c'est celle représentée par la famille d'adresses et le type de codage donnés.

L'adresse de groupe codée a le format suivant :



Famille d'adresses : décrite plus haut

Type de codage : décrit plus haut

PIM [B]idirectionnel : indique que la gamme de groupes utilise PIM bidirectionnel [RFC5015]. Pour PIM-SM comme défini dans la présente spécification, ce bit DOIT être zéro.

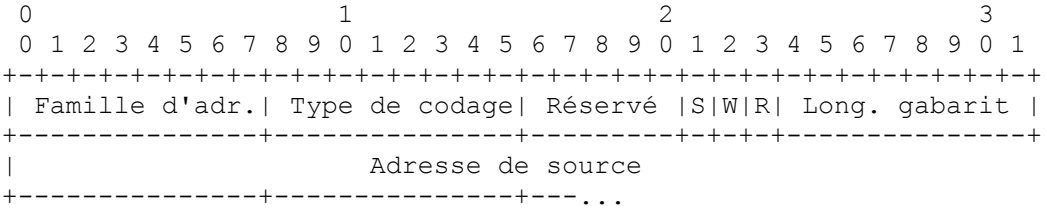
Réservé : zéro à l'émission, ignoré à réception.

[Z]one de portée administrative : Indique que la gamme de groupes est une zone de portée limitée administrativement. C'est utilisé seulement dans le mécanisme de routeur Bootstrap [RFC5059]. Pour tous les autres objets, ce bit est réglé à zéro et ignoré à réception.

Longueur de gabarit : c'est un champ de 8 bits. La valeur est le nombre de bits un contigus qui sont justifiés à gauche et utilisés comme gabarit ; quand ils sont combinés à l'adresse de groupe, elle décrit une gamme de groupes. Elle est inférieure ou égale à la longueur d'adresse en bits pour la famille d'adresses et le type de codage donnés. Si le message est envoyé pour un seul groupe, la longueur de gabarit doit alors être égale à la longueur d'adresse en bits pour la famille d'adresses et le type de codage donnés (par exemple, 32 pour le codage IPv4 natif, 128 pour le codage IPv6 natif).

Adresse de groupe de diffusion groupée : contient l'adresse du groupe.

L'adresse de source codée a le format suivant :



Famille d'adresses : décrite ci-dessus.

Type de codage : décrit ci-dessus.

Réservé : zéro à l'émission, ignoré à réception.

S : le bit "épars" est une valeur de 1 bit, réglé à 1 pour PIM-SM. Il est utilisé pour la compatibilité avec PIM version 1.

W : le bit W (pour WildCard, caractère générique) est une valeur de 1 bit à utiliser avec les messages PIM Join/Prune (voir au paragraphe 4.9.5.1).

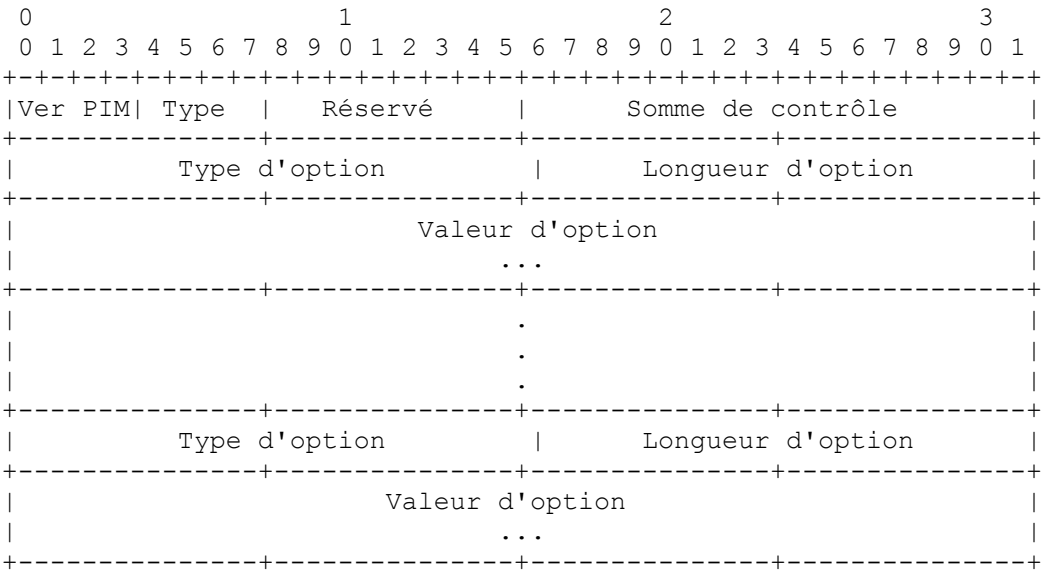
R : le bit RPT (pour Rendezvous Point Tree, arborescence de point de rendez-vous) est une valeur de 1 bit à utiliser avec les messages PIM Join/Prune (voir au paragraphe 4.9.5.1). Si le bit W est 1, le bit RPT DOIT être 1.

Longueur de gabarit : c'est un champ de 8 bits. La valeur est le nombre de bits un contigus qui sont justifiés à gauche et utilisés comme gabarit ; quand elle est combinée à l'adresse de source, elle décrit un sous réseau de source. La longueur de gabarit DOIT être égale à la longueur de gabarit pour la famille d'adresses et le type de codage donnés (32 pour IPv4 natif et 128 pour IPv6 natif). Un routeur DEVRAIT ignorer tout message reçu avec une autre longueur de gabarit.

Adresse de source : l'adresse de la source.

4.9.2 Format du message Hello

Un message Hello est envoyé périodiquement par les routeurs sur toutes les interfaces.



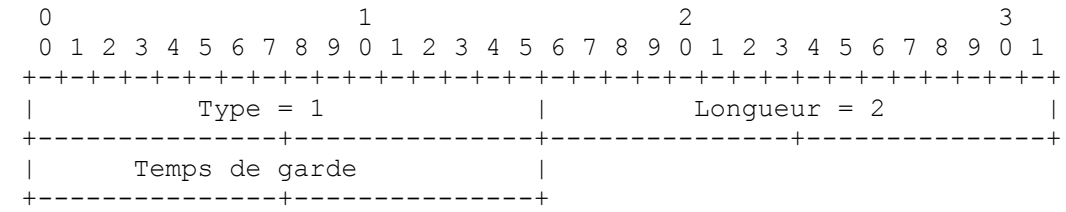
Version PIM, Type, Réservé, Somme de contrôle : décrits au paragraphe 4.9.

Type d'option : c'est le type de l'option qui figure dans le champ suivant, Valeur d'option.

Longueur d'option : longueur du champ Valeur d'option en octets.

Valeur d'option : champ de longueur variable qui porte la valeur de l'option. Le champ Valeur d'option peut contenir les valeurs suivantes :

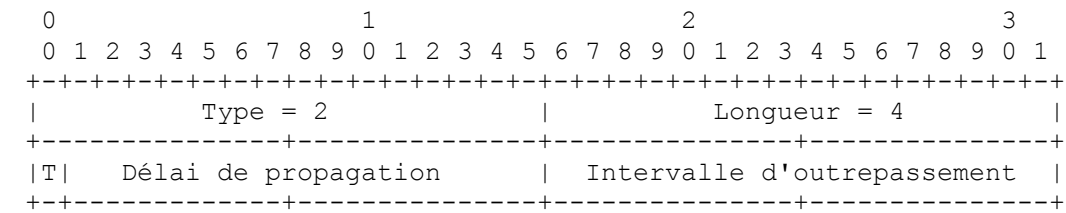
- Type d'option 1 : Temps de garde



Temps de garde est la durée pendant laquelle un receveur doit garder accessible le voisin, en secondes. Si le temps de garde est réglé à "0xffff", le receveur de ce message ne périme jamais le voisin. Cela peut être utilisé avec des liaisons en numérotation à la demande pour éviter de garder la liaison active avec des messages Hello périodiques.
Une mise en œuvre PEUT fournir un mécanisme de configuration pour rejeter un message Hello avec un temps de garde de 0xffff, et/ou fournir un mécanisme pour supprimer un voisin.

Les messages Hello avec une valeur de temps de garde réglée à "0" sont aussi envoyés par un routeur sur une interface qui est sur le point de fermer ou de changer d'adresse IP (voir au paragraphe 4.3.1). Ce sont effectivement des messages d'adieu, et les routeurs qui les reçoivent DEVRAIT immédiatement périmer les informations de voisin pour l'envoyeur.

- Type d'option 2 : Délai d'élagage de LAN



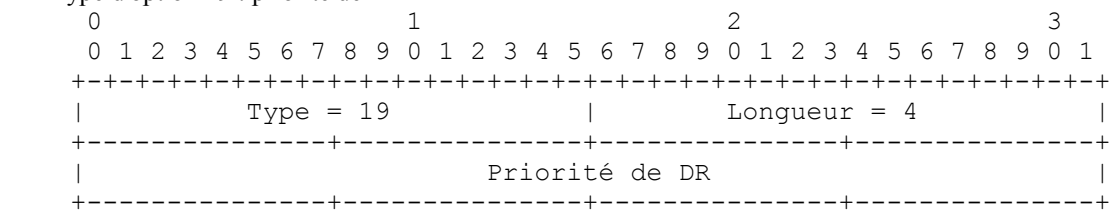
L'option Délai d'élagage de LAN est utilisée pour régler le délai de propagation d'élagage sur les LAN multi accès. Le bit T spécifie la capacité du routeur envoyeur de désactiver la suppression de Join. Délai de propagation et Intervalle d'outrepassement sont des intervalles de temps en unités de millisecondes. Un routeur qui génète une option Délai d'élagage de LAN sur l'interface I règle le champ Délai de propagation à la valeur configurée de Délai de propagation(I) et la valeur du champ Intervalle d'outrepassement à la valeur de Intervalle d'outrepassement(I). Sur un routeur receveur, les valeurs de ces champs sont utilisées pour régler la valeur de l'intervalle effectif d'outrepassement(I) et ses valeurs de temporisateur déduites.

Le paragraphe 4.3.3 décrit comment ces valeurs affectent le comportement d'un routeur.

- Type d'option 3 à 16 : réservé; à définir dans de futures versions de ce document.

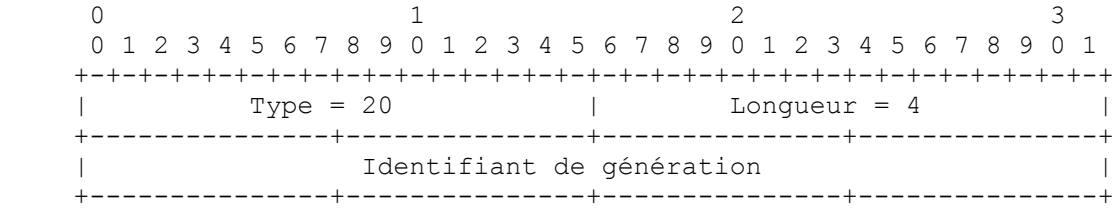
- Type d'option 18 : déconseillé et ne devrait pas être utilisé.

- Type d'option 19 : priorité de DR



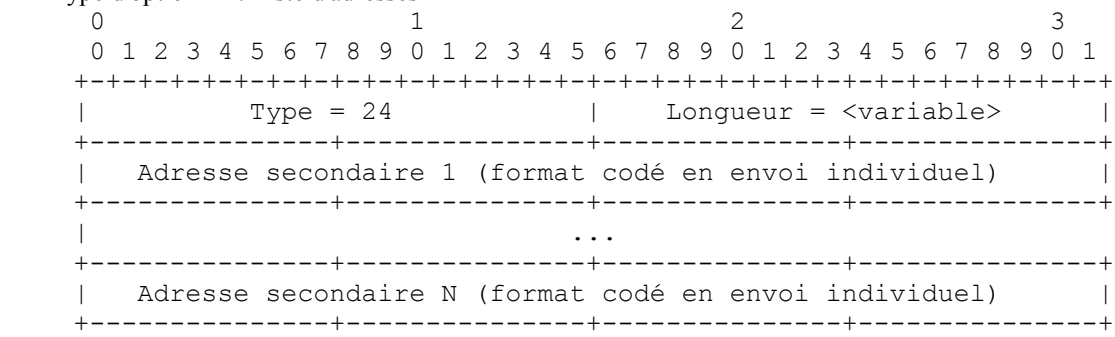
Priorité de DR est a nombre de 32 bits non signé qui devrait être pris en compte dans l'élection de DR comme décrit au paragraphe 4.3.2.

- Type d'option 20 : Identifiant de génération



Identifiant de génération est une valeur aléatoire de 32 bits pour l'interface sur laquelle le message Hello est envoyé. L'identifiant de génération est régénéré chaque fois que commence ou recommence la transmission PIM sur l'interface.

- Type d'option 24 : Liste d'adresses

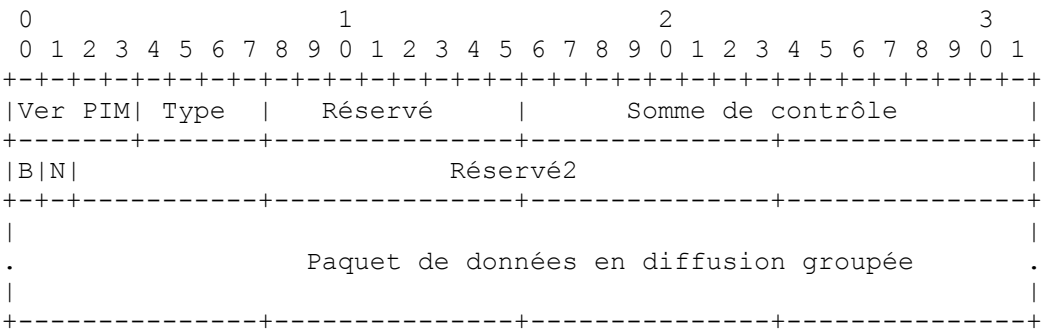


Le contenu de l'option Hello de liste d'adresse est décrit au paragraphe 4.3.4. Toutes les adresses au sein d'une seule liste d'adresse doivent appartenir à la même famille d'adresses.

Les types d'option 17 à 65000 sont allouées par l'IANA. Les types d'option de 65001 à 65535 sont réservés pour utilisation privée, comme défini dans la [RFC5226]. Les options inconnues DOIVENT être ignorées et NE DOIVENT PAS empêcher une relation de voisin d'être formée. L'option Temps de garde DOIT être mise en œuvre ; les options Priorité de DR et Identifiant de génération DEVRAIENT être mises en œuvre. L'option Liste d'adresses DOIT être mise en œuvre pour IPv6.

4.9.3 Format de message Register

Un message Register est envoyé par le DR au RP quand un paquet en diffusion groupée doit être transmis sur l'arborescence de RP. L'adresse IP de source est réglée à l'adresse du DR, l'adresse de destination à l'adresse du RP. Le TTL IP du paquet PIM est le TTL normal d'envoi individuel du système.



Version PIM, Type, Réservé, Somme de contrôle : décrits au paragraphe 4.9. Noter qu'afin de réduire les frais généraux d'encapsulation, la somme de contrôle pour les Register n'est faite que sur les 8 premiers octets du paquet, incluant l'en-tête PIM et les 4 octets suivants, excluant la portion Données du paquet. Pour des raisons d'interopérabilité, un message qui porte une somme de contrôle calculée sur le message Register PIM entier devrait aussi être acceptée. Lors du calcul de la somme de contrôle, le pseudo en-tête IPv6 "Longueur de paquet de couche supérieure" est réglé à 8.

B : bit Bordure. La présente spécification déconseille le bit Bordure. Un routeur DOIT régler le bit B à 0 à l'émission et DOIT ignorer ce bit à réception.

N : bit Null-Register. Réglé à 1 par un DR qui vérifie le RP avant de laisser arriver à expiration son temporisateur local de suppression de Register. Réglé à 0 autrement.

Réservé2 : Transmis à zéro, ignoré à réceiption.

Paquet de données en diffusion groupée : le paquet d'origine envoyé par la source. Ce paquet doit être d'une des mêmes familles d'adresse que le paquet PIM encapsulant, par exemple, un paquet de données IPv6 doit être encapsulé dans un paquet PIM IPv6. Noter que le TTL du paquet d'origine est décrémenté avant l'encapsulation, juste comme tout autre paquet transmis. De plus, le RP décrémente le TTL après la désencapsulation, avant de transmettre le paquet le long de l'arborescence partagée.

Pour les Null-Register (S,G), la portion du paquet de données en diffusion groupée contient un en-tête IP factice avec S comme adresse de source et G comme adresse de destination. Lors de la génération d'un message IPv4 Null-Register, les champs dans l'en-tête IPv4 factice DEVRAIENT être remplis en accord avec le tableau suivant. Les autres champs d'en-tête IPv4 peuvent contenir toute valeur valide pour ce champ.

Champ	**Valeur**
Version IP	4
Longueur d'en-tête	5
Somme de contrôle	somme de contrôle de l'en-tête
Décalage de fragmentation	0
Plus de fragments	0
Longueur totale	20
Protocole IP	103 (PIM)

À réception d'un Null-Register (S,G), si le champ Somme de contrôle d'en-tête n'est pas zéro, le receveur DEVRAIT vérifier la somme de contrôle et éliminer les Null-Register qui ont une mauvaise somme de contrôle. Le receveur NE DEVRAIT PAS vérifier la valeur d'un champ individuel ; une somme de contrôle correcte d'en-tête IP est suffisante. Si le champ Somme de contrôle d'en-tête est zéro, le recipient NE DOIT PAS vérifier la somme de contrôle.

Avec IPv6, une mise en œuvre génère un en-tête IP factice suivi par un en-tête PIM factice avec des valeurs en accord avec le tableau suivant en plus de la source et du groupe. Les autres champs d'en-tête IPv6 peuvent contenir toute valeur valide pour ce champ.

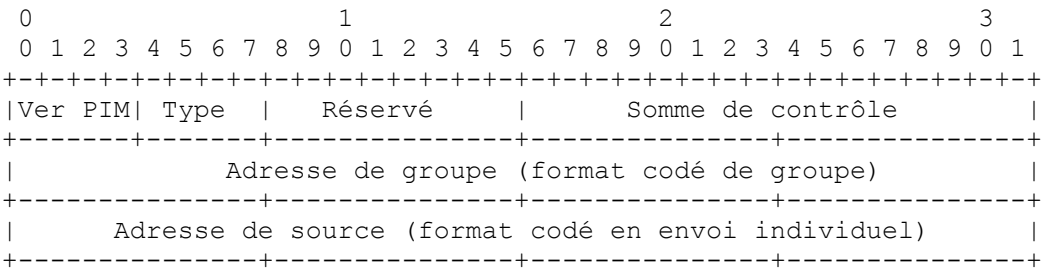
Champ d'en-tête	**Valeur**
Version IP	6
Prochain en-tête	103 (PIM)

Longueur	4
Version PIM	0
Type PIM	0
Réservé PIM	0
Somme de contrôle PIM	somme de contrôle PIM, incluant le "pseudo en-tête" Ipv6 ; voir au paragraphe 4.9

À réception d'un Null-Register (S,G) IPv6, si l'en-tête PIM factice est présent, le receveur DEVRAIT vérifier la somme de contrôle et éliminer les Null-Register qui ont une mauvaise somme de contrôle.

4.9.4 Format de message Register-Stop

Un Register-Stop est en envoi individuel du RP à l'envoyeur du message Register. L'adresse IP de source est l'adresse à laquelle le Register était adressé. L'adresse de destination IP est l'adresse de source du message Register.



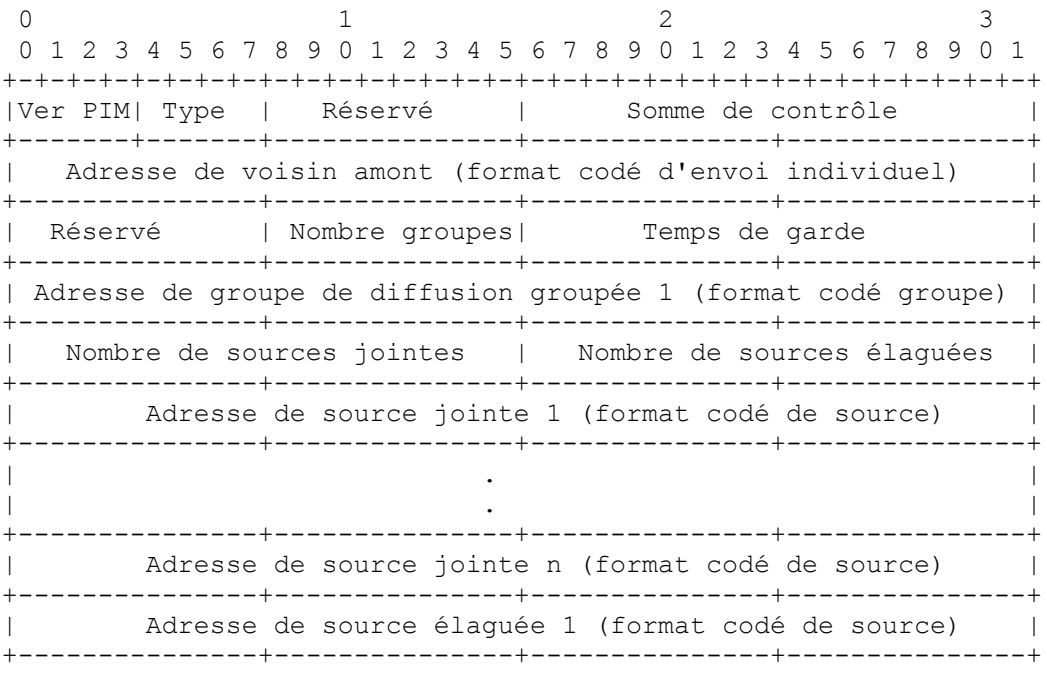
Version PIM, Type, Réservé, Somme de contrôle : décrits au paragraphe 4.9.

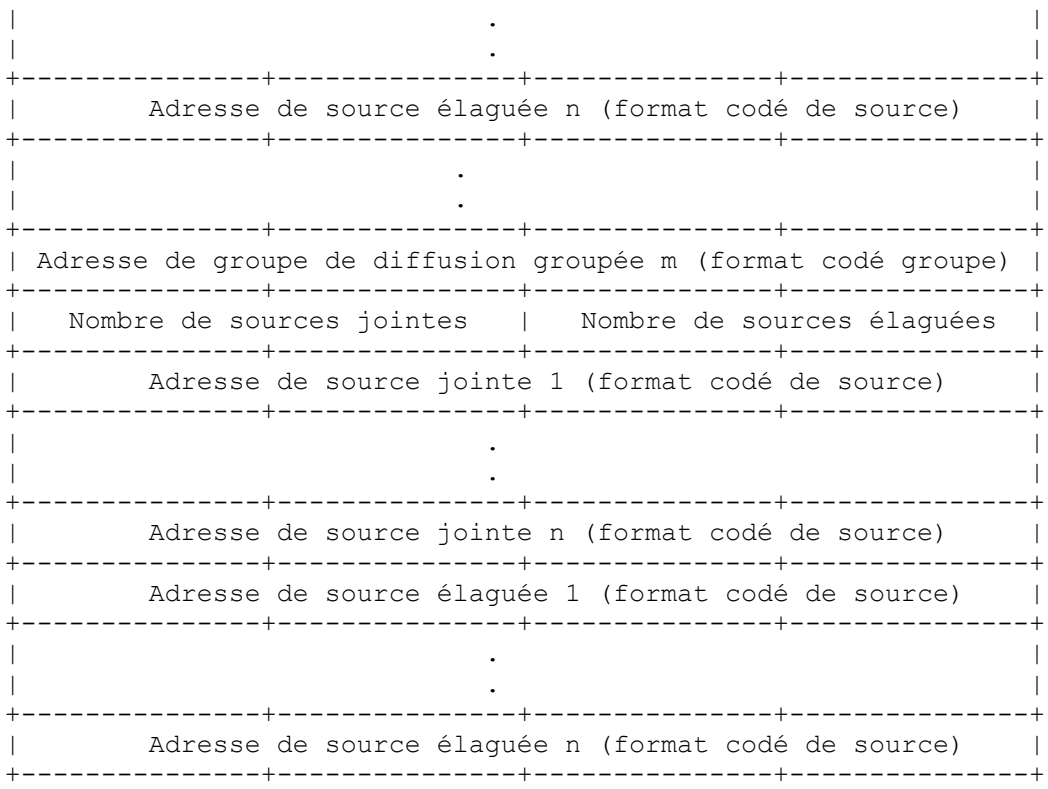
Adresse de groupe : adresse de groupe provenant du paquet de données en diffusion groupée dans le Register. Le format de cette adresse est décrit au paragraphe 4.9.1. Noter que pour les Register-Stop, le champ Longueur de gabarit contient la longueur d'adresse complète * 8 (par exemple, 32 pour le codage IPv4 natif) si le message est envoyé pour un seul groupe.

Adresse de source : adresse de l'hôte de la source du paquet de données en diffusion groupée dans le Register. Le format de cette adresse est donné dans l'adresse codée en envoi individuel au paragraphe 4.9.1. Une valeur spéciale de caractère générique consistant en un champ d'adresse tout de zéros peut être utilisé pour indiquer "toute source".

4.9.5 Format de message Join/Prune

Un message Join/Prune est envoyé par les routeurs aux sources et RP en amont. Les Join sont envoyés pour construire les arborescences partagées (arborescence de RP) ou les arborescences de source (SPT). Les messages Prune sont envoyés pour élaguer les arborescences de source quand des membres quittent les groupes ainsi que les sources qui n'utilisent pas l'arborescence partagée.





Version PIM, Type, Réservé, Somme de contrôle : décrits au paragraphe 4.9.

Adresse de voisin amont en envoi individuel : adresse principale du voisin amont qui est la cible du message. Le format de cette adresse est donné dans l'adresse codée en envoi individuel au paragraphe 4.9.1.

Réservé : transmis à zéro, ignoré à réception.

Temps de garde : durée pendant laquelle un receveur DOIT conserver l'état Join/Prune en vie, en secondes. Si il est réglé à "0xffff", le receveur de ce message DEVRAIT conserver l'état jusqu'à ce qu'il soit annulé par le message d'annulation Join/Prune approprié, ou qu'il arrive en fin de temporisation conformément à la politique locale. Cela peut être utilisé avec des liaisons à numérotation à la demande, pour éviter de garder la liaison active avec des messages Join/Prune périodiques. Noter que le temps de garde DOIT être supérieur au J/P_Override_Interval(I).

Nombre de groupes : nombre d'ensembles de groupes de diffusion groupée contenus dans le message.

Adresse de groupe de diffusion groupée : voir la description du format au paragraphe 4.9.1.

Nombre de sources jointes : nombre des adresses de source jointes mentionnées pour un groupe donné.

Adresse de source jointe 1 .. n : cette liste contient les sources pour un certain groupe desquelles le routeur d'envoi va transmettre les datagrammes en diffusion groupée si elles sont reçues sur l'interface sur laquelle le message Join/Prune est envoyé. Voir au paragraphe 4.9.1 la description du format de l'adresse de source codée.

Nombre de sources élaguées : nombre d'adresses de source élaguées mentionnées pour un groupe.

Adresse de source élaguées 1 .. n : cette liste contient les sources pour un certain groupe desquelles le routeur d'envoi ne veut pas transmettre les datagrammes en diffusion groupée quand ils sont reçus sur l'interface sur lequel le message Join/Prune est envoyé.

Au sein d'un message Join/Prune PIM, toutes les adresses de groupe de diffusion groupée, les adresses de source jointe, et les adresses de source élaguées DOIVENT être de la même famille d'adresses. Il N'est PAS PERMIS de mixer des adresses IPv4 et IPv6 dans le même message. De plus, la famille d'adresses des champs dans le message DEVRAIT être la même que les adresses IP de source et de destination du paquet. Cela permet une souplesse maximum de mise en œuvre pour les routeurs à double pile IPv4/IPv6. Si un routeur reçoit un message avec une famille d'adresses mixtes, il DEVRAIT traiter seulement les adresses qui sont de la même famille que l'adresse d'envoi individuel du voisin amont.

4.9.5.1 Règles de liste de source d'ensemble de groupe

Comme décrit plus haut, les messages Join/Prune sont composés d'un ou plusieurs ensembles de groupes. Chaque ensemble contient deux listes de sources : les sources jointes et les sources élaguées. Ce paragraphe décrit les différents types d'ensembles de groupes et les entrées de liste de source qui peuvent exister dans un message Join/Prune.

Il y a un type d'ensemble de groupe valide : ensemble spécifique de groupe.
Un ensemble spécifique de groupe est représenté par une adresse valide de diffusion groupée IP dans le champ Adresse de groupe et la longueur complète de l'adresse IP dans le champ Longueur de gabarit de l'adresse de groupe de diffusion groupée. Aucun message Join/Prune NE DEVRAIT contenir plus d'un ensemble spécifique de groupe pour la même adresse IP de diffusion groupée. Chaque ensemble spécifique de groupe peut contenir des entrées de liste de source (*,G), (S,G,rpt), et (S,G) dans les listes de joints ou élagués.

(*,G) : l'entrée de liste de source (*,G) est utilisée dans les messages Join/Prune envoyés au RP pour le groupe spécifié. Elle exprime l'intérêt (ou son absence) à recevoir le trafic envoyé au groupe à travers l'arborescence partagée du RP. Il DOIT seulement y avoir une de ces entrées dans les deux listes de joints et d'élagués d'un ensemble spécifique de groupe. Les entrées de liste de source (*,G) ont l'adresse de source réglée à l'adresse du RP pour le groupe G, la longueur de gabarit d'adresse de source réglée à la pleine longueur de l'adresse IP, et les bits WC et RPT de l'adresse de source codée établis.

(S,G,rpt) : l'entrée de liste de source (S,G,rpt) est utilisée dans les messages Join/Prune envoyés au RP pour le groupe spécifié. Elle exprime l'intérêt (ou son absence) à recevoir le trafic à travers l'arborescence partagée envoyé à ce groupe par la source spécifiée. Pour chaque adresse de source, l'entrée DOIT exister dans une seule des listes de sources jointes et élaguées d'un ensemble spécifique de groupe, mais pas les deux. Les entrées de liste de source (S,G,rpt) ont l'adresse de source réglée à l'adresse de la source S, la longueur de gabarit d'adresse de source réglée à la pleine longueur de l'adresse IP, et le bit WC à zéro et le bit RPT établi dans l'adresse de source codée.

(S,G) : l'entrée de liste de source (S,G) est utilisée dans les messages Join/Prune envoyés à la source spécifiée. Elle exprime l'intérêt (ou son absence) à recevoir le trafic à travers l'arborescence de plus court chemin envoyé par la source au groupe spécifié. Pour chaque adresse de source, l'entrée DOIT exister dans une seule des listes de sources jointes et élaguées d'un ensemble spécifique de groupe, mais pas les deux. Les entrées de liste de source (S,G) ont l'adresse de source réglée à l'adresse de la source S, la longueur de gabarit d'adresse de source réglée à la pleine longueur de l'adresse IP, et les deux bits WC et RPT de l'adresse de source codée réglés à zéro.

Les règles décrites ci-dessus sont suffisantes pour empêcher des combinaisons invalides d'entrées de liste de sources dans des ensembles spécifiques de groupes. Il y a cependant, un certain nombre de combinaisons qui ont une interprétation valide mais ne sont pas générées par le protocole comme décrit dans la présente spécification :

- Combiner une entrée de Join (*,G) et de Join (S,G,rpt) dans le même message est redondant, car l'entrée (*,G) couvre les informations fournies dans l'entrée (S,G,rpt).
- La même chose s'applique pour un Prune (*,G) et un Prune (S,G,rpt).
- La combinaison d'un Prune (*,G) et d'un Prune (S,G,rpt) n'est aussi pas générée. Les Join (S,G,rpt) ne sont envoyés que quand le routeur reçoit tout le trafic pour un groupe sur l'arborescence partagée et qu'il souhaite indiquer un changement pour cette source particulière. Comme un Prune (*,G) indique que le routeur ne souhaite plus recevoir le trafic de l'arborescence partagée, le Join (S,G,rpt) n'aurait pas de sens.
- Comme les messages Join/Prune sont ciblés sur un seul voisin PIM, inclure à la fois un Join (S,G) et un Prune (S,G,rpt) dans le même message est généralement redondant. Le Join (S,G) informe le voisin que l'envoyeur souhaite recevoir cette source particulière sur l'arborescence de plus court chemin. Il est donc inutile que le routeur dise qu'il ne souhaite plus le recevoir sur l'arborescence partagée. Cependant, il y a une interprétation valide pour cette combinaison d'entrées. Un routeur en aval peut devoir donner pour instruction à ceux en amont de ne commencer à transmettre à une source spécifique qu'une fois qu'il aura commencé à recevoir la source sur l'arborescence de plus court chemin.
- La combinaison d'un Prune (S,G) et d'un Join (S,G,rpt) pourrait éventuellement être utilisée par un routeur pour repasser de la réception d'une source particulière sur l'arborescence de plus court chemin à sa réception sur l'arborescence partagée (pourvu que le voisin RPF pour les arborescences de plus court chemin et partagée soient communes). Cependant, PIM en mode épars ne fournit pas de mécanisme pour commuter explicitement sur l'arborescence partagée.

Les règles sont résumées dans le tableau ci-dessous.

	Join (*,G)	Prune (*,G)	Join (S,G,rpt)	Prune (S,G,rpt)	Join(S,G)	Prune(S,G)
Join *,G)	-	non	?	oui	oui	oui
Prune (*,G)	non	-	?	?	oui	oui
Join (S,G,rpt)	?	?	-	non	oui	?
Prune (S,G,rpt)	oui	?	non	-	oui	?
Join (S,G)	oui	oui	oui	oui	-	non
Prune (S,G)	oui	oui	?	?	non	-

oui : Admis et attendu.

non : la combinaison n'est pas admise par le protocole et NE DOIT PAS être générée par un routeur. Un routeur PEUT accepter ces messages, mais le résultat est indéfini. Un message d'erreur PEUT être enregistré pour l'administrateur en débit limité.

? : la combinaison n'est pas prévue par le protocole, mais est bien définie. Un routeur PEUT l'accepter mais il NE DEVRAIT PAS la générer.

L'ordre des entrées de liste de sources dans une liste de sources d'ensemble de groupe n'est pas important, sauf quand il est limité par le format de paquet lui-même.

4.9.5.2 Fragmentation d'ensemble de groupe

Lors de la construction d'un Join/Prune pour un voisin particulier, un routeur devrait essayer d'inclure dans le message autant d'informations dont il a besoin pour arriver au voisin que possible. Cela implique d'ajouter un ensemble de groupes pour chaque groupe de diffusion groupée qui a des informations sur la transmission et au sein de chaque ensemble incluant toutes les entrées pertinentes de liste de sources.

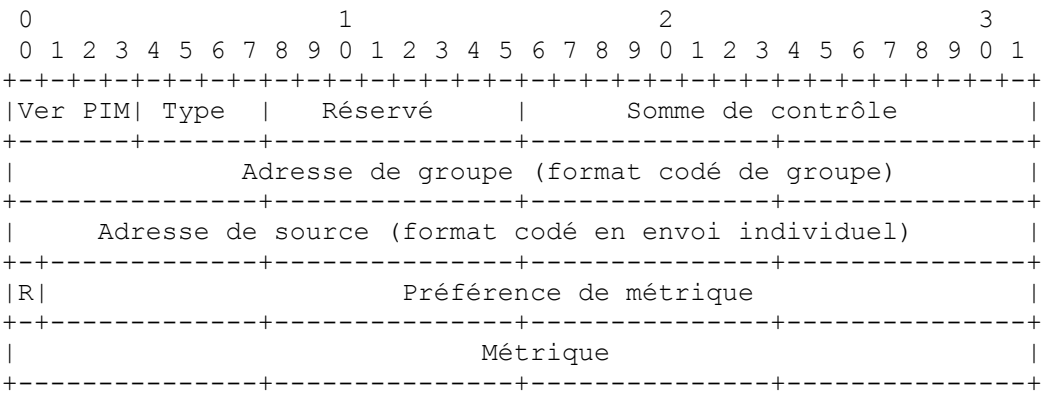
Sur un routeur qui a une grande quantité d'état de diffusion groupée, le nombre d'entrées qui doivent être incluses peut résulter en paquets plus grands que la taille maximum de paquet IP. Dans la plupart des cas, les informations peuvent être partagées sur plusieurs messages.

Il y a une exception avec les ensembles de groupes qui contiennent une entrée de liste de sources Join (*,G) . L'ensemble de groupes exprime l'intérêt du routeur à recevoir tout le trafic pour le groupe spécifié sur l'arborescence partagée, et il DOIT inclure une entrée de liste de sources Prune (S,G,rpt) pour chaque source que le routeur ne souhaite pas recevoir. Cette liste d'entrées de liste de source Prune (S,G,rpt) NE DOIT PAS être partagée sur plusieurs messages.

Si seulement N entrées Prune (S,G,rpt) tiennent dans un message Join/Prune de taille maximum, mais si le routeur a plus de N Prune (S,G,rpt) à ajouter, alors le routeur DOIT choisir d'inclure les N premières adresses IP (numériquement les plus petites dans l'ordre des octets du réseau) et le reste est ignoré (elles ne sont pas incluses).

4.9.6 Format du message Assert

Le message Assert est utilisé pour résoudre les conflits de transmission entre les routeurs sur une liaison. Il est envoyé quand un routeur reçoit un paquet de données de diffusion groupée sur une interface sur laquelle le routeur aurait normalement transmis ce paquet. Les messages Assert peuvent aussi être envoyés en réponse à un message Assert provenant d'un autre routeur.



Version PIM, Type, Réservé, Somme de contrôle : décrits au paragraphe 4.9.

Adresse de groupe : adresse de groupe pour laquelle le routeur souhaite résoudre le conflit de transmission. C'est une adresse de groupe codée, comme spécifié au paragraphe 4.9.1.

Adresse de source : adresse de source pour laquelle le routeur souhaite résoudre le conflit de transmission. L'adresse de source PEUT être réglée à zéro pour les assertions (*,G) (voir ci-dessous). Le format de cette adresse est donné dans l'adresse codée en envoi individuel au paragraphe 4.9.1.

R : le bit RPT est une valeur de 1 bit. Il est réglé à 1 pour les messages Assert (*,G) et à 0 pour les messages Assert(S,G).

Préférence de métrique : valeur de préférence allouée au protocole d'acheminement en envoi individuel qui a fourni le chemin de la source ou point de rendez-vous de diffusion groupée.

Métrique : métrique du tableau d'acheminement en envoi individuel associé au chemin utilisé pour atteindre la source ou point de rendez-vous de diffusion groupée. La métrique est dans les unités applicables au protocole d'acheminement en envoi individuel utilisé.

Les messages Assert peuvent être envoyés pour résoudre un conflit de transmission pour tout le trafic pour un groupe donné ou pour une source et groupe spécifiques.

Assert(S,G)
Les assertions spécifiques de source sont envoyées par des routeurs qui transmettent une source spécifique sur l'arborescence de plus court chemin (le bit SPT est VRAI). Les Assert (S,G) ont le champ Adresse de groupe réglé au groupe G et le champ Adresse de source réglé à la source S. Le bit RPT est réglé à 0, la préférence de métrique est réglée à MRIB.pref(S), et la métrique est réglée à MRIB.metric(S).

Assert(*,G)
Les assertions spécifiques de groupe sont envoyées par des routeurs qui transmettent des données pour le groupe et la ou les sources en compétition sur l'arborescence partagée. Les Assert(*,G) ont le champ Adresse de groupe réglé au groupe G. Pour les assertions déclenchées par des données, le champ Adresse de source PEUT être réglé à l'adresse IP de source du paquet de données qui a déclenché le Assert et est réglé à zéro autrement. Le bit RPT est réglé à 1, la préférence de métrique est réglée à MRIB.pref(RP(G)), et la métrique est réglée à MRIB.metric(RP(G)).

4.10 Temporisateurs PIM

PIM-SM tient les temporisateurs suivants, comme discuté au paragraphe 4.11. Tous les temporisateurs sont à décompte ; ils sont réglés à une valeur et décomptent jusqu'à zéro, moment où ils déclenchent normalement une action. Bien sûr, ils peuvent très facilement être mis en œuvre comme temporisateurs compteurs, où le moment absolu d'expiration est mémorisé et comparé à une horloge de l'heure réelle, mais le langage de la présente spécification suppose qu'ils décomptent jusqu'à zéro.

Temporisateurs globaux
Par interface (I) : temporisateur Hello : HT(I)
Par voisin (N) : temporisateur de vie de voisin (NLT, *Neighbor Liveness Timer*) : NLT(N,I)
Par groupe (G) : temporisateur d'expiration de Join (*,G) : ET(*,G,I)
temporisateur d'élagage en cours (*,G) : PPT(*,G,I)
temporisateur d'assertion (*,G) *(Assert Timer)* : AT(*,G,I)
Par source (S) : temporisateur d'expiration de Join (S,G) : ET(S,G,I)
temporisateur d'élagage en cours (S,G) : PPT(S,G,I)
temporisateur d'assertion (S,G) : AT(S,G,I)
temporisateur d'expiration d'élagage (S,G,rpt) : ET(S,G,rpt,I)
temporisateur d'élagage en cours (S,G,rpt) : PPT(S,G,rpt,I)
Par groupe (G) : temporisateur de Join en amont (*,G) : JT(*,G)
Par source (S) : temporisateur de Join en amont (S,G) : JT(S,G)
temporisateur de garde en vie (S,G) : KAT(S,G)
temporisateur d'outrepassement en amont (S,G,rpt) : OT(S,G,rpt)

Chez les DR ou les gagnants d'Assert pertinents seulement :
Par source, paire de groupes (S,G) : temporisareur de Register-Stop : RST(S,G)

4.11 Valeurs de temporisateurs

Lorsque des temporisateurs sont lancés ou relancés, ils sont réglés aux valeurs par défaut. Ce paragraphe résume ces valeurs par défaut. Noter que des événements de protocole ou la configuration peuvent changer la valeur par défaut d'un temporisateur sur une interface spécifique. Lorsque les temporisateurs sont initialisés dans ce document, la valeur spécifique de l'interface dans le contexte doit être utilisée.

Certains des temporisateurs mentionnés ci-dessous (Prune-Pending, Upstream Join, Upstream Override) peuvent être réglés à des valeurs qui dépendent des réglages du délai de propagation et de l'intervalle d'outrepassement de l'interface correspondante.

Les valeurs par défaut pour ceux-ci sont données ci-dessous.

Nom de la variable : Propagation_Delay(I)

Nom de la valeur	Valeur	Explication
Propagation_delay_default	0,5 s	Délai de propagation attendu sur la liaison locale.

La valeur par défaut de Propagation_delay_default est choisie pour être relativement large pour assurer la compatibilité avec les anciennes mises en œuvre de PIM.

Nom de la variable : Override_Interval(I)

Nom de la valeur	Valeur	Explication
t_override_default	2,5 s	Intervalle de délai par défaut sur lequel on se place au hasard pour programmer un message Join retardé.

Nom du temporisateur : Temporisateur de Hello (HT(I))

Nom de la valeur	Valeur	Explication
Hello_Period	30 s	Intervalle périodique pour les messages Hello.
Délai_Hello_Déclenché	5 s	Intervalle aléatoire pour le message Hello initial à l'amorçage ou du message Hello déclenché à un voisin qui réamorce.

À la mise sous tension du système, le temporisateur est initialisé à rand(0, Délai_Hello_Déclenché) pour empêcher la synchronisation. Lorsque un voisin nouveau ou qui réamorce est détecté, un Hello de réponse est envoyé dans rand(0, Délai_Hello_Déclenché).

Nom du temporisateur : Temporisateur de vie de voisin *(Neighbor Liveness Timer)* (NLT(N,I))

Nom de la valeur	Valeur	Explication
Default_Hello_Holdtime	3,5 * Hello_Period	Temps de garde par défaut pour garder l'état de voisin vivant.
Hello_Holdtime	D'après le message	Temps de garde provenant de l'option de message Hello Holdtime.

Le temps de garde dans un message Hello devrait être réglé à (3,5 * Hello_Period) donnant une valeur par défaut de 105 s.

Nom du temporisateur : Temporisateur d'expiration (ET(*,G,I), ET(S,G,I), ET(S,G,rpt,I))

Nom de la valeur	Valeur	Explication
J/P_HoldTime	D'après le message	Temps de garde pour le message Join/Prune

La valeur de J/P Holdtime qui est incluse dans les messages Join/Prune est spécifiée ci-dessous, dans la description du "temporisateur Join amont (JT(*,G), JT(S,G))".

Nom du temporisateur : Temporisateur d'élagage en cours (PPT(*,G,I), PPT(S,G,I), PPT(S,G,rpt,I))

Nom de la valeur	Valeur	Explication
J/P_Override_Interval(I)	Par défaut : Effective_Propagation_Delay(I) + Effective_Override_Interval(I)	Courte période après un Join ou Prune pour permettre aux autres routeurs sur le LAN d'outrepasser le Join ou Prune.

Noter que Effective_Propagation_Delay(I) et Effective_Override_Interval(I) sont toutes deux des valeurs spécifiques de l'interface qui peuvent changer quand des messages Hello sont reçus (voir au paragraphe 4.3.3).

Nom du temporisateur : Temporisateur d'assertion (AT(*,G,I), AT(S,G,I))

Nom de la valeur	Valeur	Explication
Assert_Override_Interval	Par défaut : 3 s	Bref intervalle avant qu'une assertion arrive en fin de temporisation où le gagnant d'assertion renvoie un message Assert.
Assert_Time	Par défaut : 180 s	Période après le dernier Assert avant que l'état Assert soit périmé.

Noter que pour des raisons historiques, le message Assert manque d'un champ Temps de garde. Donc, changer le Assert_Time de la valeur par défaut n'est pas recommandé.

Nom du temporisateur : Temporisateur Joint amont (JT(*,G), JT(S,G))

Nom de la valeur	Valeur	Explication
t_periodic	Par défaut : 60 s	Période entre les messages Join/Prune
t_suppressed	Rand(1,1 * t_periodic, 1,4 *t_periodic) quand Suppression_Enabled(I) est vrai, 0 autrement	Période de suppression quand quelqu'un d'autre envoie un message J/P de sorte qu'on a pas besoin de le faire.
t_override	rand(0, Effective_Override_Interval(I))	Délai aléatoire pour empêcher l'explosion de réponses lors de l'envoi d'un message Join pour outrepasser le message Prune de quelqu'un d'autre.

t_periodic peut être réglé à prendre en compte des choses comme la bande passante configurée et le nombre moyen attendu d'entrées de chemins de diffusion groupée pour le réseau ou la liaison rattaché (par exemple, la période serait plus longue pour les liaisons moins rapides, ou pour les routeurs dans le centre du réseau qui s'attendent à avoir un plus grand nombre d'entrées). Si le Join/Prune-Period est modifié durant le fonctionnement, ces changements devraient être faits relativement rarement, et le routeur devrait continuer de rafraîchir ses Join/Prune-Period précédents pendant au moins Join/Prune-Holdtime, afin de permettre au routeur en amont de s'adapter.

Le temps de garde spécifié dans un message Join/Prune devrait être réglé à (3,5 * t_periodic).
t_override dépend de l'intervalle d'outrepassement effectif sur l'interface amont, qui peut changer quand des messages Hello sont reçus.
t_suppressed dépend de l'état de suppression de l'interface amont (paragraphe 4.3.3) et devient zéro quand la suppression est désactivée.

Nom du temporisateur : Temporisateur d'outrepassement amont *(Upstream Override Timer)* (OT(S,G,rpt))

Nom de la valeur	Valeur	Explication
t_override	Voir temporisateur Join amont	Voir temporisateur Join amont

Le temporisateur d'outrepassement amont est toujours réglé à la valeur t_override ; cette valeur est définie plus haut sous "Noms de temporisateurs : temporisateur Join amont (JT(*,G), JT(S,G))".

Nom du temporisateur : temporisateur de garde en vie *(Keepalive Timer)* (KAT(S,G))

Nom de la valeur	Valeur	Explication
Keepalive_Period	Par défaut : 210 s	Période après le dernier paquet de données (S,G) durant lequel l'état Join (S,G) va être maintenu même en l'absence de messages Join (S,G).
RP_Keepalive_Period	(3 * Register_Suppression_Time) + Register_Probe_Time	Comme Keepalive_Period, mais au RP quand un Register-Stop est envoyé.

La période de garde en vie normale pour KAT(S,G) est par défaut de 210 secondes. Cependant, au RP, la période de garde en vie doit être au moins de Register_Suppression_Time, ou le RP peut périmer l'état (S,G) avant que le prochain Null-Register arrive. Donc, KAT(S,G) est réglé à max(Keepalive_Period, RP_Keepalive_Period) quand un Register-Stop est envoyé.

Nom du temporisateur : temporisateur de Register-Stop (RST(S,G))

Nom de la valeur	Valeur	Explication
Register_Suppression_Time	défaut : 60 s	Période durant laquelle un DR arrête d'envoyer des données encapsulées dans Register au RP après réception d'un message Register-Stop.
Register_Probe_Time	défaut : 5 s	Temps avant que RST expire quand un DR peut envoyer un Null-Register au RP pour lui faire renvoyer un message Register-Stop.

Si le Register_Suppression_Time ou le Register_Probe_Time est configuré à des valeurs autres que par défaut, il DOIT être assuré que la valeur du Register_Probe_Time est inférieure à la moitié de la valeur de Register_Suppression_Time pour empêcher une possible valeur négative dans le réglage du temporisateur de Register-Stop.

5. Considérations relatives à l'IANA

5.1 Famille d'adresses PIM

Le champ Famille d'adresses PIM a été choisi de 8 bits comme compromis entre le format de paquet et l'utilisation des numéros alloués par l'IANA. Parce que quand le format de paquet PIM a été conçu, seules 15 valeurs étaient allouées pour les familles d'adresses, et qu'un grand nombre de nouvelles familles d'adresses n'était pas envisagé, 8 bits semblaient suffisants. Cependant, l'IANA alloue les familles d'adresse dans un champ de 16 bits. Donc, la famille d'adresses PIM est allouée comme suit :
Les valeurs de 0 à 127 sont destinées à avoir la même signification que les numéros de famille d'adresses alloués par l'IANA [IANA].
Les valeurs de 128 à 250 sont destinées à être allouées pour PIM par l'IANA sur la base de l'approbation de l'IESG, comme défini dans la [RFC5226].
Les valeurs 251 à 255 sont destinées à utilisation privée, comme défini dans la [RFC5226].

5.2 Options PIM Hello

Les valeurs 17 à 65000 sont à allouer par l'IANA. Comme l'espace est grand, elles peuvent être allouées au “premier arrivé, premier servi”, comme défini dans la [RFC5226]. De telles allocations sont valides pour un an et peuvent être renouvelées. Des allocations permanentes exigent une spécification (voir "Spécification exigée" dans la [RFC5226]).

6. Considérations sur la sécurité

Cette Section décrit divers problèmes de sécurité possibles relatifs au protocole PIM-SM. Le lecteur se reportera aux [RFC5796], [RFC4609], et [RFC5294] pour une discussion plus approfondie de la sécurité de PIM-SM et de la diffusion groupée.

Noter que PIM s'appuie sur une MRIB remplie en-dehors de PIM ; donc, sécuriser les sources de changement de la MRIB est RECOMMANDÉ.

6.1 Attaques fondées sur des messages falsifiés

L'étendue des dommages possibles dépend du type de messages contrefaits acceptés. On considère ensuite l'impact de possibles falsifications, incluant des messages falsifiés de liaison locale (Join/Prune, Hello, et Assert) et des messages en envoi individuel (Register et Register-Stop).

6.1.1 Messages falsifiés de liaison locale

Les messages Join/Prune, Hello, et Assert sont tous envoyés à l'adresse de diffusion groupée de liaison locale TOUS-LES-ROUTEURS-PIM et donc ne sont pas transmis par un routeur conforme. Un message falsifié de ce type peut seulement atteindre un LAN si il a été envoyé par un hôte local ou si il a été admis sur le LAN par un routeur compromis ou non conforme.

1. Un message Join/Prune falsifié peut être cause que le trafic de diffusion groupée sera livré à des liaisons où il n'y a pas de demandeur légitime, gaspillant potentiellement la bande passante sur cette liaison. Un message Leave falsifié sur un LAN multi accès n'est généralement pas une attaque significative dans PIM, parce que tout routeur légitimement joint sur le LAN va outrepasser le Leave avec un Join avant que le routeur en amont arrête de transmettre les données au LAN.

2. En falsifiant un message Hello, un routeur non autorisé peut faire qu'il soit lui-même choisi comme routeur désigné sur un LAN. Le routeur désigné sur un LAN est (en l'absence de Assert) responsable de la transmission du trafic à ce LAN au nom de tous les membres locaux. Le routeur désigné est aussi responsable de l'enregistrement-encapsulation au RP de tous les paquets qui sont générés par les hôtes sur le LAN. Donc, la capacité des hôtes locaux d'envoyer et recevoir du trafic de diffusion groupée peut être compromis par un message Hello falsifié.

3. En falsifiant un message Assert sur un LAN multi accès, un attaquant pourrait amener un transmetteur légitimement désigné à arrêter de transmettre le trafic au LAN. Une telle falsification empêcherait tout hôte en aval de ce LAN de recevoir du trafic.

6.1.2 Messages en envoi individuel falsifiés

Les messages Register et Register-Stop sont transmis par les routeurs intermédiaires à leur destination en utilisant la transmission IP normale. Sans authentification de l'origine des données, un attaquant situé n'importe où dans le réseau peut être capable de falsifier un message Register ou Register-Stop. On examine ci-dessous l'effet d'une falsification de chacun de ces messages.

1. En falsifiant un message Register, un attaquant peut causer l'injection par le RP de trafic falsifié sur l'arborescence partagée de diffusion groupée.

2. En falsifiant un message Register-Stop, un attaquant peut empêcher un DR légitime d'enregistrer des paquets au RP. Ceci peut empêcher les hôtes locaux sur ce LAN d'envoyer des paquets en diffusion groupée.

Les deux messages PIM ci-dessus ne sont pas changés par les routeurs intermédiaires et ont seulement besoin d'être examinés par le receveur prévu. Donc, ces messages peuvent être authentifiés de bout en bout. Les attaques sur les messages Register et Register-Stop ne s'appliquent pas à une mise en œuvre de PIM-SSM seulement, car ces messages ne sont pas exigés pour PIM-SSM.

6.2 Mécanismes d'authentification non cryptographiques

Un routeur PIM DEVRAIT fournir une option pour limiter l'ensemble de voisins de qui il va accepter des messages Join/Prune, Assert, et Hello. La configuration statique des adresses IP ou une association de sécurité IPsec PEUT être utilisée. De plus, un routeur PIM NE DEVRAIT PAS accepter des messages de protocole d'un routeur de qui il n'a pas encore reçu un message Hello valide.

Un routeur désigné NE DOIT PAS enregistrer-encapsuler un paquet et l'envoyer au RP sauf si l'adresse de source du paquet est une adresse légale pour le sous réseau sur lequel le paquet a été reçu. De même, un routeur désigné NE DEVRAIT PAS accepter un paquet Register-Stop dont l'adresse IP de source n'est pas une adresse de RP valide pour le domaine local.

Une mise en œuvre DEVRAIT fournir un mécanisme pour permettre à un RP de restreindre la gamme des adresses de source desquelles il accepte les paquets Register encapsulés.

Toutes les options qui restreignent la gamme des adresses d'où les paquets sont acceptés DOIVENT par défaut permettre tous les paquets.

6.3 Authentification

Le présent document se réfère à la [RFC5796], qui spécifie des mécanismes pour authentifier les messages PIM-SM de liaison locale en utilisant l'encapsulation de charge utile de sécurité (ESP, *Encapsulating Security Payload*) ou (facultativement) l'en-tête d'authentification (AH, *Authentication Header*) de IPsec . Il souligne aussi que les messages PIM-SM non de liaison locale (c'est-à-dire, les messages Register et Register-Stop) peuvent être sécurisés par une association de sécurité IPsec normale en envoi individuel entre deux parties communicantes.

6.4 Attaques de déni de service

Un certain nombre d'attaques de déni de service sont possibles contre PIM qui peuvent être causées par la génération de faux messages de protocole PIM ou même en générant du faux trafic. Authentifier le trafic de protocole PIM empêche certaines de ces attaques, mais pas toutes. Deux attaques possibles sont :

- L'envoi très rapide de paquets à de nombreuses adresses de groupe différentes peut être une attaque de déni de service en soi. Cela va causer de nombreux paquets à enregistrer-encapsuler, surchargeant le DR, le RP, et les routeurs entre le DR et le RP.
- Falsifier un message Join peut causer l'établissement d'une arborescence de diffusion groupée. Un grand nombre de Join falsifiés peut consommer les ressources du routeur et résulter en un déni de service.

7. Références

[IANA] IANA, "Address Family Numbers", < http://www.iana.org/assignments/address-family-numbers >.

[RFC1112] S. Deering, "Extensions d'hôte pour diffusion groupée sur IP", STD 5, août 1989. *(Mise à jour par la* RFC2236*)*

[RFC2119] S. Bradner, "Mots clés à utiliser dans les RFC pour indiquer les niveaux d'exigence", BCP 14, mars 1997.

[RFC2460] S. Deering et R. Hinden, "Spécification du protocole Internet, version 6 (IPv6) ", décembre 1998. *(MàJ par* 5095, 6564 *; D.S)*

[RFC2710] S. Deering, W. Fenner et B. Haberman, "Découverte d'écouteur de diffusion groupée (MLD) pour IPv6", octobre 1999.

[RFC2983] D. Black, "Services différenciés et tunnels", octobre 2000. *(Information)*

[RFC3376] B. Cain et autres, "Protocole Internet de gestion de groupe, IGMP version 3", octobre 2002. *(P.S.)*

[RFC3956] P. Savola, B. Haberman, "Incorporation de l'adresse de point de rendez-vous (RP) dans une adresse de diffusion groupée IPv6", novembre 2004. *(P.S.)*

[RFC4607] H. Holbrook, B. Cain, "Diffusion groupée spécifique de source pour IP", août 2006. *(P.S.)*

[RFC4609] P. Savola et autres, "Diffusion groupée indépendante du protocole - Mode épars (PIM-SM) : questions de sécurité de l'acheminement de la diffusion groupée et améliorations", octobre 2006. *(Information)*

[RFC4760] T. Bates, R. Chandra, D. Katz et Y. Rekhter, "Extensions multi protocoles pour BGP-4", janvier 2007.

[RFC5015] M. Handley et autres, "Diffusion groupée bidirectionnelle indépendante du protocole (BIDIR-PIM)", octobre 2007. *(P.S.)*

[RFC5059] N. Bhaskar et autres, "Mécanisme de routeur d'amorçage (BSR) pour la diffusion groupée indépendante du protocole (PIM)", janvier 2008. (*Remplace* RFC2362) (*MàJ* RFC4601) *(P.S.)*

[RFC5226] T. Narten et H. Alvestrand, "Lignes directrices pour la rédaction d'une section Considérations relatives à l'IANA dans les RFC", BCP 26, mai 2008. (*Remplace* RFC2434)

[RFC5294] P. Savola, J. Lingard, "Menaces sur la diffusion groupée indépendante du protocole (PIM) au niveau de l'hôte", août 2008. *(Information)*

[RFC5796] W. Atwood, S. Islam, M. Siami, "Authentification et confidentialité dans les messages de liaison locale du mode de diffusion groupée éparse indépendante du protocole (PIM-SM)", mars 2010. (*MàJ* RFC4601). *(P. S.)*

[RFC7063] L. Zheng, J. Zhang, R. Parekh, "Rapport d'enquête sur la mise en œuvre et le déploiement de la diffusion groupé indépendante du protocole – mode épars (PIM-SM)", décembre 2013. *(Information)*

Appendice A. Fonctionnalités de la RFC4601 retirées

Sur la base d'une enquête sur la mise en œuvre et les déploiements de PIM [RFC7063] conduite par le groupe de travail PIM de l'IETF, les fonctions suivantes de la RFC 4601 ont été supprimées par suite d'un manque d'expérience suffisante de mise en œuvre et de déploiement :

- état (*,*,RP)
- routeur bordure de diffusion groupée PIM (PMBR, *PIM Multicast Border Router*)
- authentification en utilisant IPsec.

Remerciements

PIM-SM a été conçu sur de nombreuses années par un large groupe de personnes, incluant des idées, commentaires, et corrections de la part de Deborah Estrin, Dino Farinacci, Ahmed Helmy, David Thaler, Steve Deering, Van Jacobson, C. Liu, Puneet Sharma, Liming Wei, Tom Pusateri, Tony Ballardie, Scott Brim, Jon Crowcroft, Paul Francis, Joel Halpern, Horst Hodel, Polly Huang, Stephen Ostrowski, Lixia Zhang, Girish Chandranmenon, Brian Haberman, Hal Sandick, Mike Mroz, Garry Kump, Pavlin Radoslavov, Mike Davison, James Huang, Christopher Thomas Brown, et James Lingard.

Nos remerciements à la American Licorice Company, pour son rôle, obscur mais essentiel dans la création du présent document.

Adresse des auteurs

Bill Fenner
Arista Networks
mél : fenner@arista.com

Mark Handley
Department of Computer Science
University College London
Gower Street
London WC1E 6BT
United Kingdom
mél : M.Handley@cs.ucl.ac.uk

Hugh Holbrook
Arista Networks
5453 Great America Parkway
Santa Clara, CA 95054
mél : holbrook@arista.com

Isidor Kouvelas
Arista Networks
5453 Great America Parkway
Santa Clara, CA 95054
mél : kouvelas@arista.com

Rishabh Parekh
Cisco Systems, Inc.
170 W. Tasman Drive
San Jose, CA 95134
mél : riparekh@cisco.com

Zhaohui Zhang
Juniper Networks
10 Technology Park Drive
Westford, MA 01886
mél : zzhang@juniper.net

Lianshu Zheng
Huawei Technologies Co., Ltd
156 Beiqing Road, Hai-dian District
Beijing 100089 China
mél : vero.zheng@huawei.com