# rfc2460

Spécification IPv6

IPv6 succède à IPv4 et apporte les changements suivant :

**Capacités d'adressage étendus** IPv6 étend la taille d'adresse de 32 bits à 128 bits, pour supporter un plus grand nombre de nœuds adressables, et une auto-configuration plus simple. Le routage multicast est amélioré en ajoutant un scope aux adresses multicast. Un nouveau type d'adresse appelés adresse anycast est définis, utilisé pour envoyer un paquet à n'importe qui dans un groupe de nœud.

**Simplification du format d'en-tête** Certains champs IPv4 ont été supprimés, d'autre rendus optionnels, pour réduire le coût de traitement de la manipulation des paquets et pour limiter le coût de la bande passante de l'en-tête IPv6.

**Support amélioré pour les extensions et les options** Change la manière dont les options de l'en-tête sont encodées pour un forwarding plus efficace et une plus grande flexibilité.

**Capacité de labélisation de flux** Permet de labéliser les paquets appartenant à un trafic particulier pour lequel l'émetteur demande une manipulation spéciale.

**Authentification et confidentialité** Les extensions pour supporter l'authentification, l'intégrité des données, et la confidentialité des données.

# Terminologie

**node** Un périphérique qui implémente IPv6

**router** Un nœud qui transmet les paquets IPv6 qui ne lui sont pas adressés

**host** Tout nœud qui n'est pas un routeur

**upper layer** Un protocole de la couche immédiatement au dessus d'IPv6, comme TCP ou UDP.

**link** Un moyen de communication sur lequel les nœud peuvent communiquer, ex : Ethernet, PPP, X.25, Frame Relay, ou ATM

**neighbors** Un nœud attaché au même lien

**interface** L'attachement d'un nœud à un lien

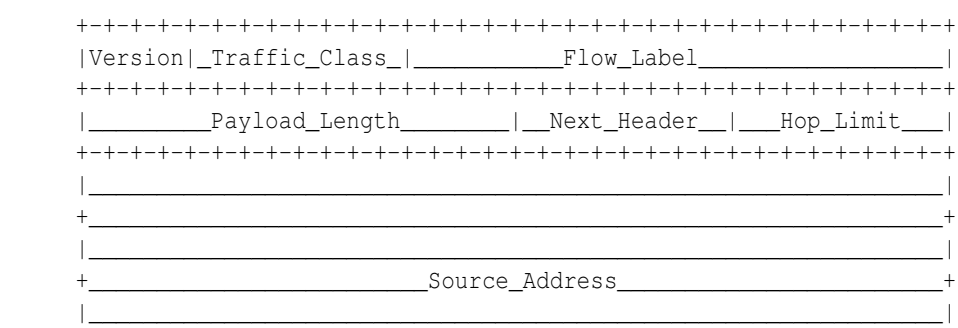
**address** Un identifiant de la couche IPv6 pour une interface ou un jeu d'interfaces.

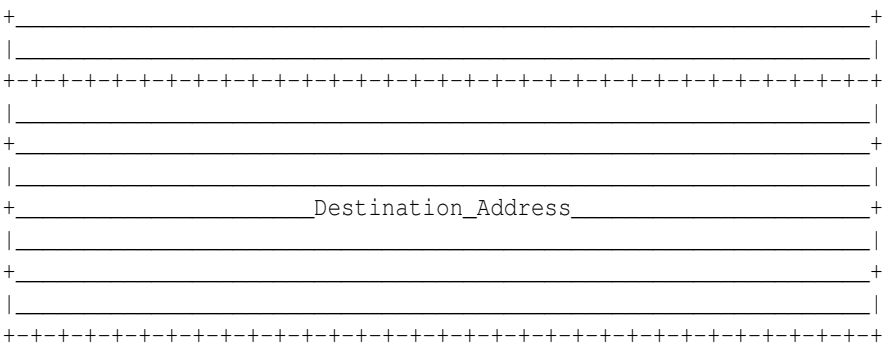
**packet** Un en-tête IPv6 plus le payload

**link MTU** Maximum Transmission Unit : taille max d'un paquet qui peut être véhiculé sur un lien.

**path MTU** Le MTU le plus faible de tous les liens dans le chemin entre un nœud source et un nœud de destination.

# Format de l'en-tête






**Version** 4bits - numéro de version du protocole (6)

**Traffic Class** 8bits - Champ de classe de trafic

**Flow Label** 20bits - Label

**Payload Length** 16bits - Longueur du payload IPv6

**Next Header** 8bits - Identifie le type d'en-tête qui suit. Utilise les même valeurs que IPv4

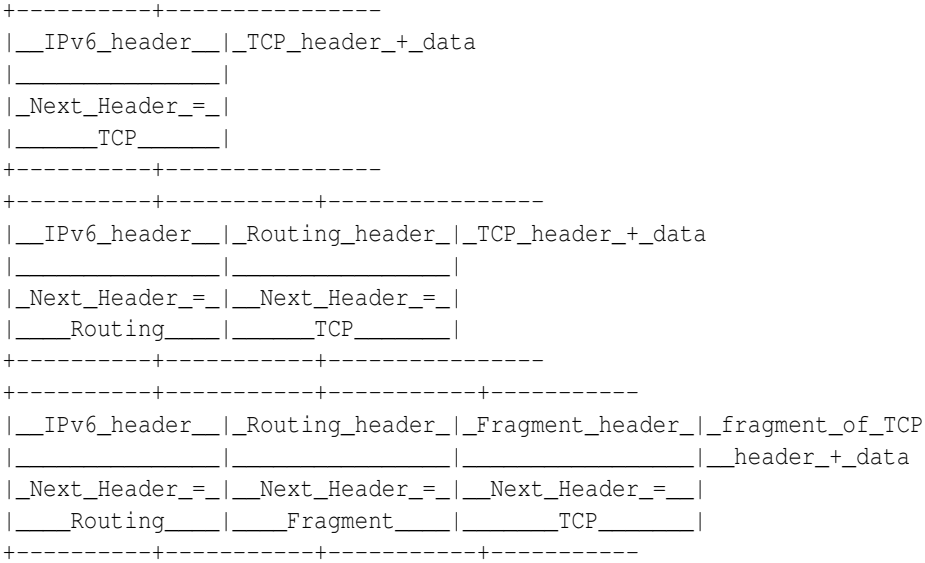
**Hop Limit** 8bits - Décrémenté de 1 par chaque nœud qui forward le paquet.

**Source Address** 128bits - Adresse de l'émetteur

**Destination Address** 128bits - Adresse du destinataire

Dans IPv6, les information optionnelles sont encodées dans des en-tête séparées qui peuvent être placés entre l'en-tête IPv6 et l'en-tête de la couche supérieur. Il peut y avoir 0, 1 ou plusieurs extensions, chacun identifié par le champ next header :

# Format de l'en-tête




À une exception, les en-tête d'extension ne sont pas examinés ou traités par les nœuds le long du chemin, jusqu'à ce que le paquet atteigne le nœud identifié dans le champ destination address. Donc, un démultiplexage normal dans le champ next header invoque le module pour traiter la première extension, ou l'en-tête du upper-layer s'il n'y a pas d'extension. Le contenu et les sémantiques de chaque extension détermine si ou non on doit traiter le next header. Cependant, les en-têtes d'extension doivent être traités strictement dans l'ordre auquel ils apparaissent dans le paquet.

L'extension Hop-by-Hop est une exception, qui gère les informations qui doivent être examinés et traités par tous les nœuds dans le

chemin de livraison, incluant les nœuds source et de destination. Cet extension doit suivre immédiatement l'en-tête IPv6. Sa présence est indiquée par la valeur 0 dans le champs next header.

Si, en résultat du traitement d'un en-tête, un nœud nécessite de traiter le next header mais que la valeur de next header n'est pas reconnue par le nœud, il devrait détruire le paquet et envoyer un message ICMP Parameter Problem à la source du paquet, avec un code ICMP de 1 et un Pointeur ICMP contenant l'offset de la valeur non reconnue dans le paquet. La même action devrait être faite si un nœud rencontre un next header de 0 dans un en-tête autre que l'en-tête IPv6

Chaque en-tête d'extension est un entier multiple de 8 octets, pour pouvoir maintenir un alignement 8 octets pour les autres en-têtes. Une implémentation complète d'IPv6 inclus l'implémentation des en-tête d'extension suivantes :

**Hop-by-Hop Options**
**Routing (Type 0)**
**Fragment**
**Destination Options**
**Authentication**
**Encapsulating Security Payload**

Les 4 premiers sont spécifiés dans ce document, les 2 derniers sont spécifiés dans la rfc2402 et la rfc2406.

# Ordre d'en-tête d'extension

Quand plus d'un en-tête d'extension sont utilisé dans le même paquet, il est recommandé qu'ils apparaissent dans l'ordre suivant :

**IPv6 header**
**Hop-by-Hop Options header**
**Destination Options header (note 1)**
**Routing header**
**Fragment header**
**Authentication header (note 2)**
**Encapsulating Security Payload header (note 2)**
**Destination Options header (note 3)**
**upper-layer header**

**note 1 :** Pour les options à traiter par la première destination qui apparaît dans le champ Destination Address plus les destinations suivante listée dans l'en-tête Routing.

**note 2 :** Des recommandation additionnelles au regard le l'ordre des en-tête d'authentification et d'encapsulation sont donnés dans la rfc2406.

**note 3 :** Pour les options à traiter seulement par la destination finale du paquet.

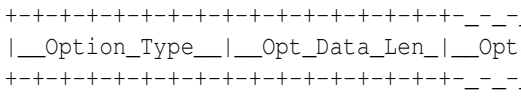
Chaque en-tête d'extension devrait exister un seule fois au plus, excepté pour l'en-tête Destination Options qui devrait exister au plus 2 fois.

Si l'en-tête upper-layer est un autre en-tête IPv6, il peut seulement être suivant par ses propres extensions, qui sont sujet aux même recommandation d'ordre.

Si et quand d'autre en-tête d'extension sont définis, leur contrainte d'ordre relative aux en-tête définis plus haut doivent être spécifiés.

# Options

2 des en-tête d'extension définis ici – Hop-by-Hop et Destination Header – gère un nombre variable d'options type-length-value (TLV) encodés, dont le format est le suivant :




**Option Type** 8bits - identifiant du type d'option

**Opt Data Len** 8bits - Longueur du champ Option Data en octets

**Option Data** Champ de longueur variable.

La séquence d'options dans un en-tête doit être traité strictement dans l'ordre qu'ils apparaissent dans l'en-tête. Les identifiant d'Option Type sont encodés en interne de manière à ce que leur 2 bit MSB spécifient l'action à effectuer si le nœud le traitant ne reconnaît pas l'Option Type :

**00** Passe l'option et continue à traiter l'en-tête

**01** Supprime le paquet

**10** Détruit le paquet et envoie un ICMP parameter Problem, Code 2.

**11** Détruit le paque et envoie un ICMP parameter Problem, Code 2 seulement si de Destination Address n'est pas une adresse multicast.

Le bit suivant spécifie si l'Option Data de cette option peut changer en cour de route vers la destination finale. Quand un en-tête d'authentification est présent dans le paquet, pour une option dont la donnée peut changer en cour de route, tout son champ Option Data doit être traité comme une série de 0 en calculant ou en vérifiant la valeur d'authentifiant le paquet.

**0** Option Data ne change pas en cour de route

**1** Option Data peut changer en cour de route

Les 3 bits MSB décris ci-dessus sont traités comme partie de l'Option Type, et non indépendamment. Le même espace de numérotation d'Option Type est utilisé par Hop-by-Hop et Destination Options. Cependant, la spécification d'une option particulière peut restreindre son utilisation à seulement un de ces en-tête.

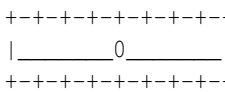
Des options individuelles peuvent avoir des besoin d'alignement spécifiques, pour s'assurer que les valeurs multi-octets dans Option Data conserve les limites naturelles. L'alignement requis d'une option est spécifiée en utilisant la notation xn+y, signifiant que l'Option Type doit apparaître comme un multiple entier de x octets du début de l'en-tête, plus y octets. Par exemple :

**2n** Signifie un offset 2 octets depuis le début de l'en-tête

**8n+2** Signifie un offset 8 octets depuis le début de l'en-tête, plus 2 octets.

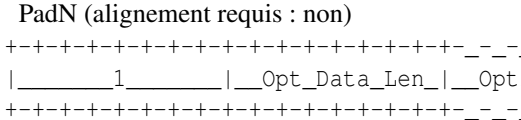
Il y a 2 options de padding qui sont utilisées si nécessaire pour aligner les options suivantes et aligner l'en-tête à un multiple de 8 octets. Ces options de padding doivent être reconnues par toutes les implémentations IPv6 :

Pad1 (alignement requis : non)




Note : le format de l'option Pad1 est un cas spéciale, il n'a pas de champs de longueur et valeur. Cette option est utilisée pour insérer un octet de padding dans les options de l'en-tête. Si plus d'un octet de padding sont requis, l'option PadN, ci-dessous, doit être utilisé :

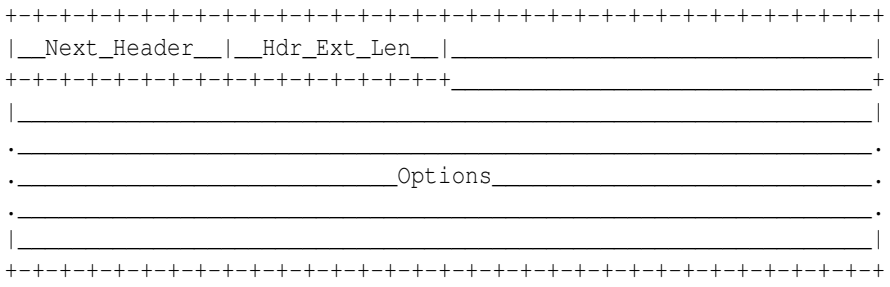
PadN (alignement requis : non)



L'option PadN est utilisé pour insérer 2 ou plusieurs octets de padding dans les options de l'en-tête. Pour N octets de padding, le champs Opt Data Len contient la valeur N-2, et l'Option Data consiste de N-2 octets de valeur 0.

# Hop-by-Hop

L'option Hop-by-Hop est utilisé pour gérer les information optionnelles qui doivent être examinés par tous les nœud le long du chemin de livraison du paquet. Cette option est identifiée par la valeur Next Header 0 dans l'en-tête IPv6, et a le format suivant :



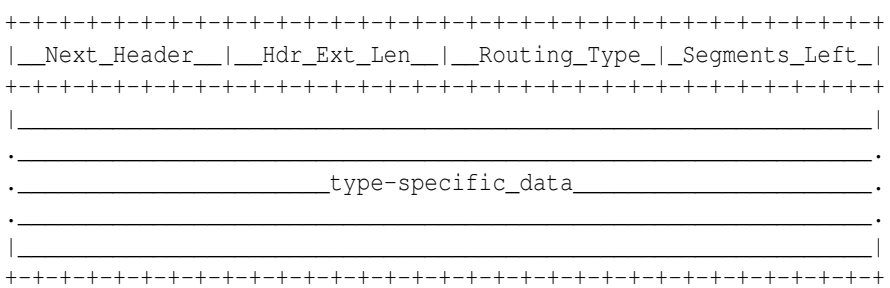

**Next Header** 8bits - identifie le type d'en-tête immédiatement suivant l'option Hop-by-Hop

**Hdr Ext Len** 8bits - Longueur de l'option Hop-by-Hop en unité de 8 octets, n'incluant pas les 8 premiers octets.

**Options** Variable, contient une ou plusieurs options TLV.

# Routing

L'en-tête Routing est utilisé par une source IPv6 pour lister un ou plusieurs nœud intermédiaire à visiter sur le chemin vers la destination du paquet. Cette fonction est très similaire à Loos Source et Record Route d'IPv4. Cet en-tête est identifié par la valeur Next Header de 43 et a le format suivant :




**Next Header** 8bits - identifie le type d'en-tête immédiatement suivant l'option Routing

**Hdr Ext Len** 8bits - Longueur de l'option Routing en unité de 8 octets, n'incluant pas les 8 premiers octets.

**Routing Type** 8bits - identifant d'une variante d'en-tête Routing particulière

**Segments Left** 8bits - Nombre de segment de routes restant à visiter

**Type-specific data** Variable - le format est déterminé par Routing Type.

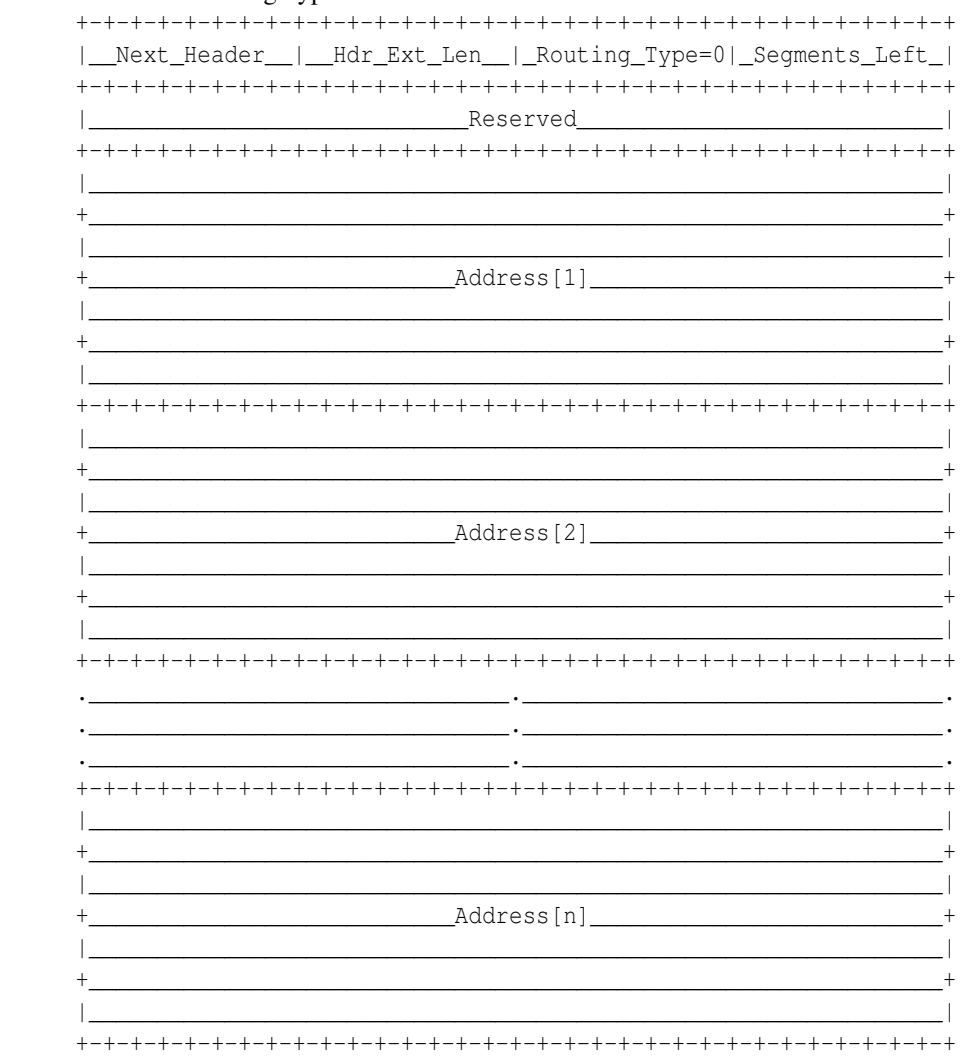
Si, en traitant un paquet reçu, un nœud rencontre un en-tête Routing avec un Routing Type inconnu, le comportement dépend de la valeur du champ Segments Left :

Si Segment Left vaut 0, le nœud doit ignorer l'en-tête Routing et traiter l'en-tête suivant dans le paquet.

Si Segment Left ne vaut pas 0, le nœud doit détruire le paquet en envoyer un ICMP Parameter Problem, code 0.

Si, une fois l'en-tête Routing traité un nœud intermédiaire détermine que le paquet doit être forwardé sur un lien dont le link MTU est inférieur à la taille du paquet, le nœud doit détruire le paquet et envoyer un ICMP Packet Too Big.

L'en-tête Routing Type 0 a le format suivant :




**Next Header** 8bits - Identifie le prochain header

**Hdr Ext Len** 8bits - Longueur de l'en-tête Routing en unité de 8 octets, n'incluant pas les 8 premiers octets

**Routing Type** 0

**Segment Left** 8bits - Nombre de segments restant

**Reserved** 32bits - Initialisé à 0 pour la tranmission ; ignorée à la reception

**Address [1..n]** Vecteur d'adresses 128 bits, numérotées de 1 à n

Les adresses multicast ne doivent pas apparaître dans l'en-tête Routing de type 0, ou dans le champ Destination Address d'un paquet gérant un en-tête Routing de type 0.

Un en-tête Routing n'est pas examiné ou traité tant qu'il n'atteind pas le nœud identifié dans le champ Destination Address. Dans ce nœud, le module Routing est chargé, et dans le cas du type 0, effectue l'algorithme suivant :

```
if Segments Left = 0 {
 traite le next header dans le paquet, dont le type est
 identifié par Next Header dans l'en-tête Routing
}
else if Hdr Ext Len is odd {
 Envoie un ICMP Parameter Problem, Code 0 à la source, pointant
 sur le champs Hdr Ext Len, et détruit le paquet
```

```
}
else {
 Calcule n, le nombre d'adresses dans l'en-tête Routing, en
 divisant Hdr Ext Len par 2

 if Segments Left is greater than n {
  Envoie un ICMP Parameter Problem, Code 0 à la source, pointant
  sur le champ Segments Left, et détruit le paquet
 }
 else {
  décrémente Segments Left de 1;
  Calcule i, l'index de la prochaine adresses à visiter dans
  le vecteur d'adresse, en soustrayant Segment Left de n

  if Address [i] or the IPv6 Destination Address is multicast {
   détruit le packet
  }
  else {
   Inverse Destination Address et Address[i]

   if the IPv6 Hop Limit is less than or equal to 1 {
    Envoie un ICMP Time Exceeded - Hop Limit Exceeded
    dans le message Transit à la source et détruit le paquet
   }
   else {
    décrémente Hop Limit de 1

    renvoie le paquet au module IPv6 pour le transmettre
    à la nouvelle destination
   }
  }
 }
}
```

Considérons le case suivant comme exemple de cet algorithme. Un nœud source S envoie un paquet à destination du nœud D, en utilisant en en-tête Routing pour forcer le paquet à être routé via les nœuds intermédiaire I1, I2 et I3. Les valeurs de l'en-tête IPv6 et des champs d'en-tête Routing sur chaque segment dans le chemin de livraison serait comme suit :

Lorsque le paquet va de S à I1 :

```
Source Address = S Hdr Ext Len = 6
Destination Address = I1 Segments Left = 3
     Address[1] = I2
     Address[2] = I3
     Address[3] = D
```

Lorsque le paquet va de I1 à I2 :

```
Source Address = S Hdr Ext Len = 6
Destination Address = I2 Segments Left = 2
     Address[1] = I1
     Address[2] = I3
     Address[3] = D
```

Lorsque le paquet va de I2 à I3 :

```
Source Address = S Hdr Ext Len = 6
Destination Address = I3 Segments Left = 1
     Address[1] = I1
```

```
    Address[2] = I2
    Address[3] = D
```

Lorsque le paquet va de I3 à D :

```
Source Address = S Hdr Ext Len = 6
Destination Address = D Segments Left = 0
    Address[1] = I1
    Address[2] = I2
    Address[3] = I3
```

# Fragment

L'en-tête Fragment est utilisé par une source IPv6 pour envoyer un paquet plus grand que dans le path MTU vers sa destination. (À la différence d'IPv4, la fragmentation n'est effectuée que par les nœuds sources). L'en-tête Fragment est identifié par une valeur Next Header de 44 et a le format suivant :

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|__Next_Header__|___Reserved____|______Fragment_Offset____|Res|M|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|_______________________________Identification_____________________|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

**Next Header** 8bits - identifie le prochain en-tête

**Reserved** 8bits - Initialisé à 0 pour la transmission, ignorée à la reception

**Fragment Offset** 13bits - l'offset, en unités de 8 octet de la donnée suivant cet en-tête, relatif au début de la partie Fragmentable du paquet original

**Res** 2bits - Initialisé à 0 pour la transmission, ignorée à la reception

**M flag** 1 = d'autres fragments, 0 = dernier fragment.

**Identification** 32bits - Voir ci-dessous

Pour envoyer un paquet qui est trop grand pour le MTU du chemin vers sa destination, un nœud source peut diviser le paquet en fragments et envoyer chaque fragment comme paquet séparé, à réassemble par le destinataire.

Pour tout paquet qui est fragmenté, le nœud source génère une valeur d'identification. Cette identification doit être différente qui tout autre paquet fragmenté envoyé récemment avec la même adresse source et de destination. Si un en-tête Routing est présent, l'adresse de destination à se préoccuper est celle de la destination finale.

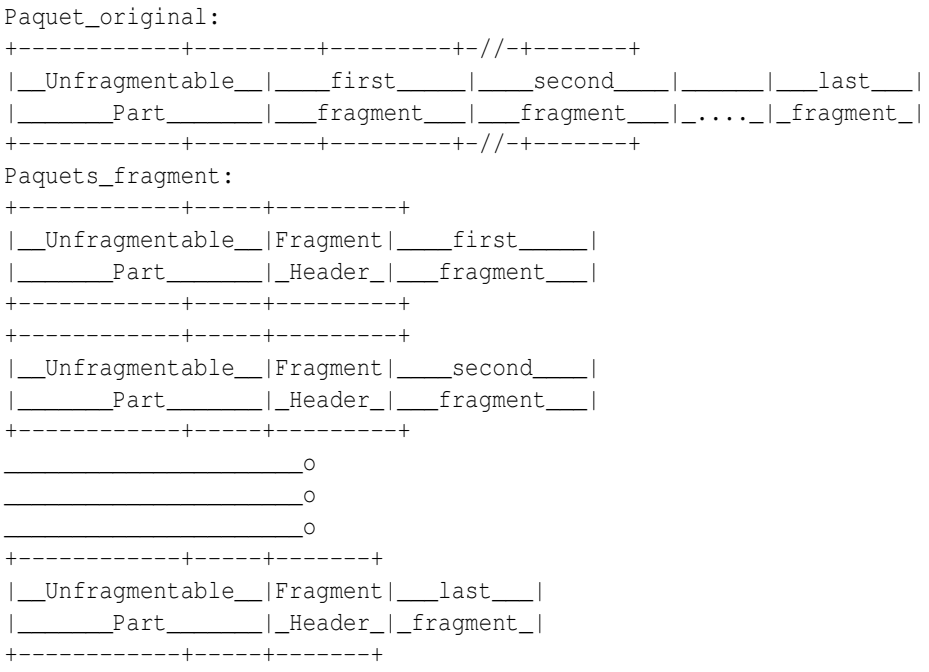
Le paquet initial, non-fragmenté est référé au paquet original, et il consiste de 2 parties :

```
+------------+---------------//---------------+
|__Unfragmentable__|____________________Fragmentable____________________|
|_______Part_______|_________________________Part_____________________________|
+------------+---------------//---------------+
```

La partie non-fragmentable consiste de l'en-tête IPv6 plus les en-tête d'extension qui doivent être traités par les nœuds sur le chemin de la destination.

La partie fragmentable consiste du reste du paquet, c'est à dire les en-tête d'extension qui doivent être traités seulement par la destination, plus le upper-layer.

La partie Fragmentable du paque original est divisée en fragments, chacun, possiblement excepté du dernier, devient un entier multiple de 8 octets de long. Les fragments sont transmis dans des paquets fragment séparés :




Chaque fragment est composé de :

**1-** La partie non-fragmentable du paquet original, avec le Payload Length de l'en-tête original changé pour contenir la longueur de ce paquet fragment (excluant la longueur de l'en-tête IPv6), et le champ Next Header de dernier en-tête de la partie non-fragmentée changée à 44.

**2-** Un en-tête Fragment contenant :

**-** La valeur Next Header qui identifie le premier header de la partie Fragmentable du paquet original

**-** Un Fragment Offset contenant l'offset du fragment, en unités de 8 octets, relatif au début de la partie fragmentable du paquet originale. Le Fragment Offset du premier fragment est 0.

**-** La valeur du flag M à 0 si le fragment est le dernier, sinon 1

**-** La valeur d'identification générée pour le paquet original

**3-** Le fragment lui-même

À la destination, les paquets fragment sont réassemblés dans leur version non-fragmenté. Les règles suivantes gouvernent le ré-assemblage.

- Un paquet original est ré-assemblé seulement depuis les paquets fragments qui ont la même adresse source, de destination, et d'identification
- La partie non-fragmentable du paquet ré-assemblé consiste de tous les en-tête jusqu'à, mais n'incluant pas, l'en-tête Fragment du premier paquet fragment (c'est à dire avec Fragment Offset à 0), avec les 2 changements suivants :

**-** Le champ Next Header du dernier en-tête de la partie non-fragmentable est obtenu du champ Next Header de l'en-tête Fragment du premier fragment.

**-** La longueur du paylod du paquet ré-assemblé est calcudé avec la longueur de la partie non-fragmentable, et la longueur et l'offset du dernier fragment. Par exemple, une formule pour calculer la longueur du payload est :

**PL.orig = PL.first - FL.first - 8 + (8 addentry articles autoadd autofind autoprod createalpha createbeta createdb createprod finde**

où :

**PL.orig=** champ Payload Length du paquet réassemblé

**PL.first=** champ Payload Length du premier fragment réassemblé

**FL.first=** Longueur du fragment suivant l'en-tête Fragment du premier paquet fragment.

**FO.last*** Champ Fragment Offset de l'en-tête fragment du dernier paquet fragment

**FL.last=** Longueur de fragment suivant l'en-tête fragment du dernier paquet fragment.

- La partie Fragmentable du paquet ré-assemblé est construite depuis les fragments suivant les en-tête fragment dans chaque paquet fragment. La longueur de chaque fragment est calculé en soustrayant du Payload Length du paquet la longueur des en-tête entre l'en-tête IPv6 et le fragment.
- L'en-tête fragment n'est pas présent dans le paquet final ré-assemblé.

Les erreurs suivant peuvent se produire en réassemblant des paquets fragmentés :

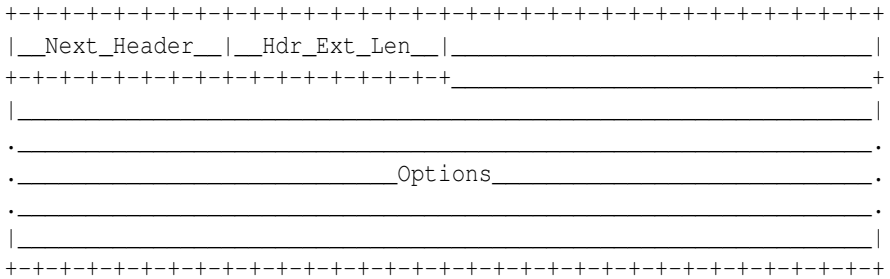
- Si les fragments reçus sont insuffisant pour réassembler un paquet dans les 60 secondes après la réception du premier fragment de ce paquet, le ré-assemblage du paquet doit être abandonné et tous les paquets fragments détruits. Si le premier fragment (celui avec un Fragment Offset à 0) a été reçu, un ICMP Time Exceeded – Fragment Reassembly Time Exceeded devrait être envoyé à la source de ce fragment.
- Si la longueur d'un fragment, comme dérivé du champ Payload Length du paquet fragment, n'est pas un multiple de 8 octets et que le flag M est à 1, alors ce fragment doit être détruit et un message ICMP Parameter Problem, code 0 doit être envoyé à la source, pointant sur le champ Payload Length du paquet fragment.
- Si la longueur et l'offset d'un fragment sont définis de sorte que le Payload Length du paquet ré-assemblé excède 65535 octets, ce fragment doit être détruit est un ICMP Parameter Problem, code 0 doit être envoyé à la source, pointant sur le champ Fragment Offset du paquet fragment.

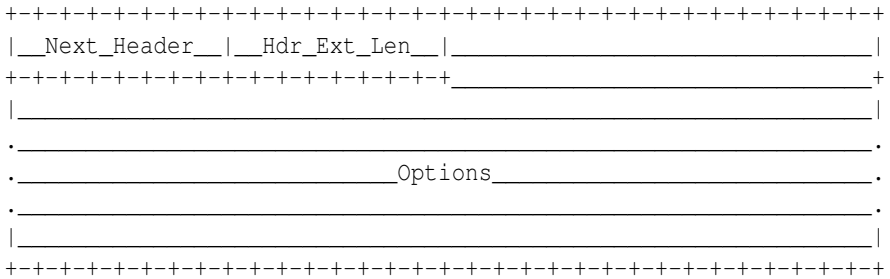
Les conditions suivante ne sont pas sensé se produire, mais ne sont pas considérés comme des erreurs :

- Le nombre et le contenu des en-tête précédant l'en-tête fragment de différents fragment du même paquet original peut différer. Les en-têtes présent avant l'en-tête fragment dans chaque paquet fragment sont traités quand le paquet arrive, avant la mise en queue des fragment pour ré-assemblage. Seul ces en-tête dans le paquet fragment d'offset 0 sont retenus dans le paquet réassemblé.
- Les valeurs Next Header dans les en-tête Fragment de différents fragments du même paquet original peuvent différer. Seul la valeur du fragment d'offset 0 est utilisé pour le ré-assemblage.

# Destination Options

L'en-tête Destination Options est utilisé pour gérer des informations optionnelles qui doivent être examinés seulement par le nœud destinataire du paquet. Cet en-tête est identifié par la valeur Next Header 60 et a le format suivant :




**Next Header** 8bits - Identifie le prochain type d'en-tête

**Hdr Ext Len** 8bits - Longueur de l'en-tête en unité de 8 octets, n'incluant pas les 8 premiers octets

**Options** Variable

Les seules options de destination définies dans ce document sont Pad1 et PanN décris plus haut.

Noter qu'il y a 2 manières possible d'encoder les informations de destination optionnelles dans un paquet IPv6 : soit commme option dans l'en-tête Destination Header, ou comme en-tête d'extension séparée. L'en-tête fragment et l'en-tête Authentication sont des examples de cette approche. L'approche utilisée dépend de l'action désirée du nœud destination qui ne comprend pas les information optionnelles :

- Si l'action désirée est pour que le nœud de destination supprime le paquet et, seulement si l'adresse de destination n'est pas une adresse multicast, envoie un ICMP Unrecognized Type à l'adresse source, alors l'information peut être encodé dans un en-tête séparé ou comme option dans Destination Options dont l'Option Type un MSB à 11.
- Si une autre action est souhaitée, le informations doivent être encodées comme option dans Destination Options dont Option Type a le MSB à 00, 01, ou 10.

# No Next Header

La valeur 59 dans le champ Next Header indique qu'il n'y pas rien après cet en-tête. Si le champ Payload Length de l'en-tête IPv6 indique la présence d'octets après un en-tête dont le Next Header est à 59, ces octets doivent être ignorés, et passés tel quel si le paquet doit être forwardé.

# Problèmes de taille de paquet

IPv6 nécessite que tout lien dans l'internet ait un MTU de 1280 octets ou supérieur. Dans un lien qui ne peut pas convoyer un paquet 1280 octets en un paquet, une fragmentation et un ré-assemblage doit être fournis au niveau de la couche sous IPv6.

Les lients qui ont un MTU configurable (par exemple PPP) doivent être configurés pour avoir un MTU d'au moins 1280 octets ; Il est recommandé que le MTU soit de 1500 octets ou supérieur, pour gérer les encapsulation (ex : tunneling) sans générer de fragmentation IPv6.

Sur chaque lien sur lequel un nœud est directement attaché, le nœud doit être capable d'accepter les paquets aussi grand que le MTU du lien

Il est fortement recommandé que les nœuds IPv6 implémentent Path MTU Discovery (rfc1981), pour pouvoir découvrir les MTU supérieurs à 1280. Cependant, une implémentation minimale d'IPv6 peut simplement restreindre à envoyer des paquets de 1280 octets au plus, et omettre la découverte de MTU.

POur envoyer un paquet plus grand que le MTU du chemin, un nœud peut utiliser l'en-tête Fragment pour fragmenter le paquet à la source. Cependant, l'utilisation d'une telle fragmentation est découragée dans toute application capable d'ajuster ses paquets à la taille du Path MTU.

Un nœud doit être capable d'accepter un paquet fragmenté qui, après ré-assemblage, est plus grand que 1500 octets. Un nœud est autorisé à accepter des paquets fragmentés qui ré-assemblent à plus de 1500 octets. Un protocole upper-layer ou une application qui dépend de la fragmentation IPv6 pour envoyer des paquets supérieurs au MTU d'un chemin ne devrait pas envoyer des paquets supérieur à 1500 octets sauf s'il a l'assurance que la destination est capable de ré-assembler les paquets de taille plus grande.

En réponse à un paquet IPv6 qui est envoyé à une destination IPv4 (ex : un paquet qui est traduit d'IPv6 vers IPv4), l'émetteur IPv6 peut recevoir un ICMP Packet Too Big reportant un Next-Hop MTU inférieur à 1280. Dans ce cas, le nœud IPv6 n'est pas obligé de réduire la taille des paquets sous-jacent à moins de 1280, mais doit inclure un en-tête fragment dans ses paquets pour que le routeur traduisant IPv6 vers IPv4 puisse obtenir une valeur d'identification correcte pour les fragments IPv4. Noter que cela signifie que le payload peut nécessiter d'être réduit à 1232 octets (1280 moins 40 pour l'en-tête IPv6 et 8 pour l'en-tête fragment), et plus petits si d'autres en-têtes sont utilisés.

# Flow Labels

Le champ 20bits Flow Label dans l'en-tête IPv6 peut être utilisé par une source pour labéliser les séquences de paquets pour lesquels il demande une manipulation spéciale par les routeurs IPv6, tel qu'un QOS différent, ou un service temps-réel. Cet aspect d'IPv6 est, au moment de la rédaction, expérimental est sujet à changement. Les hôtes ou routeurs qui ne supportent pas les fonctions de Flow Label, ignorent le champs.

# Classes de trafic

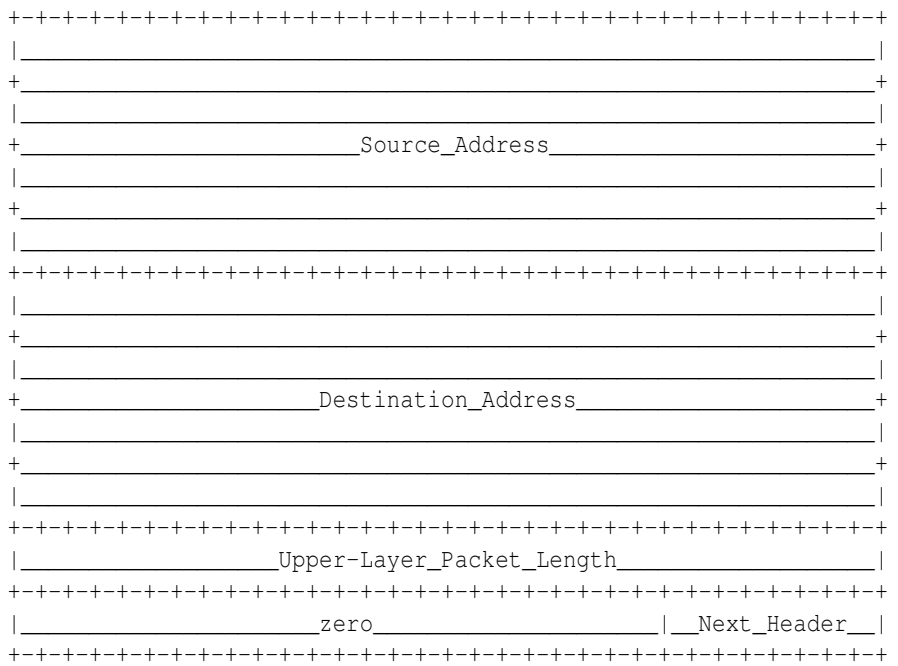
Le champ 8bits Traffic Class dans l'en-tête IPv6 est disponible par les nœuds émetteur et/ou les routeurs pour identifier et distinguer différentes classes ou priorités de paquets IPv6. Au moment de la rédaction de ce document, il y a des expérimentations dans l'utilisation de Type of Service d'IPv4 et/ou les bits Precedence pour fournir diverses formet de services différenciés pour les paquets IP. Le champ Traffic Class dans l'en-tete IPv6 est prévu pour permettre une fonctionnalité similaire dans IPv6.

Les pré-requis suivants s'appliquent au champ traffic class :

- L'interface de service IPv6 d'un nœud doit fournir un moyen pour un protocole upper-layer de founir une valeur de Traffic Class. La valeur par défaut doit être 0.
- Les nœuds qui supportent une utilisation spécifique de certains bits Traffic Class sont autorisés à changer la valeur de ces bits dans les paquets qu'ils émettent, forwardent, ou reçoivent. Les nœud devraient ignorer et laisser les bits de Traffic Class inchangé pour les paquets qui ne supportent aucune utilisation spécifique.
- Un protocole upper-layer ne doit pas assumer que la valeur des bits Traffic Class dans un paquet reçu soient les même que la valeur envoyée par la source du paquet.

# Problématiques des protocole upper-layer

Tout protocole transport ou upper-layer qui incluent les adresses de l'en-tête IP dans sont checksum doivent être modifiés pour IPv6, pour inclure des adresses IPv6 128bits. En particulier, l'illustration suivante montre le pseudo-header TCP et UDP pour IPv6 :



- Si le paquet IPv6 contient un en-tête Routing, l'adresse de destination utilisé dans le pseudo-entête est celui de la destination finale. Dans le nœud émetteur, l'adresse sera dans le dernier élément de l'en-tête Routing.
- La valeur Next Header dans le pseudo-entête identifie le protocole upper-layer (ex : 6 pour TCP, 17 pour UDP). Il va différer de la valeur Next Header dans l'en-tête IPv6 s'il y a des en-tête d'extension entre l'en-tête IPv6 et les données.
- À la différence d'IPv4, quand des paquets UDP sont émis par un nœud IPv6, le checksum IDP n'est pas optionnel. C'est à dire, quand un paquet UDP est émis, le nœud IPv6 doit calculer un checksum UDP sur le paquet et le pseudo-header, et, si ce calcul résulte à 0, il doit être changé pour 0xFFFF dans l'en-tête UDP. Les destinataires IPv6 doivent détruire les paquets UDP contenant un checksum 0, et devraient logger l'erreur.

La version IPv6 d'ICMP inclus le pseudo-header ci-dessus dans son calcul de checksum ; c'est un changement par rapport à ICMP d'ipv4. La raison pour ce changement est de protéger ICMP contre les problèmes de livraison ou la corruption des champs d'en-tête d'IPv6 sur lequel il dépend, qui, à la différence d'IPv4, ne sont pas couverts par un checksum de la couche internet. Le champ Next Header dans le pseudo-header pour ICMP contient la valeur 58.

# Durée de vie d'un paquet

À la différence d'IPv4, Les nœuds IPv6 ne sont pas obligés de forcer une durée de vie de paquet maximum. C'est la raison pour laquelle le champ Time-to-Live d'IPv4 a été renommé Hop Limit dans IPv6. Dans la pratique, très peu d'implémentations IPv4 sont conformes à cette durée de vie.

# Taille de payload upper-layer maximum

En calculant la taille du payload maximum disponible pour les données upper-layer, un protocole upper-layer doit prendre en compte la taille de l'en-tête IPv6 relative à l'en-tête IPv4. Par exemple, dans IPv4, l'option MSS de TCP est calculée comme taille de paquet maximum (une valeur par défaut ou une valeur apprise via Path MTU Discovery) moins 40 octets (20 cotets pour IPv4, et 20 octets pour TCP). En utilisant TCP sur IPv6, le MSS doit être calculé comme taille de paquet maximum moins 60 octets, parce que l'en-tête IPv6 minimum (sans extensions) fait 20 octets de plus que IPv4.

# Répondre aux paquets avec des en-têtes Routing

Quand un protocole upper-layer envoie un ou plusieurs paquets en réponse à un paquet reçus qui inclus un en-tête Routing, les réponses ne doivent pas inclure un en-tête Routing qui a été automatiquement dérivé en inversant l'en-tête Routing sauf si l'integrité et l'authenticité de Source Address et de l'en-tête Routing a été vérifiée. En d'autre termes, seul les paquets de types suivants sont autorisés à répondre à un paquet ayant un en-tête routing :

- Les paquets réponse qui n'ont pas d'en-tête Routing
- Les paquets réponse qui ont un en-tête Routing qui n'est pas dérivé de l'en-tête Routing du paquet reçu
- Les paquets réponse qui ont un en-tête Routing dérivé de l'en-tête Routing si et seulement si l'intégrité et l'authenticité de Source Address et de l'en-tête Routing a été vérifié.