RFC 7676 : IPv6 Support for Generic Routing Encapsulation (GRE)

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 23 octobre 2015

Date de publication du RFC : Octobre 2015

https://www.bortzmeyer.org/7676.html

Le protocole GRE est le plus simple mécanisme de tunnel normalisé. Il est mis en œuvre sur d'innombrables systèmes, et très largement répandu. Pour reprendre la formule de James Raftery, GRE est « The "*AK-47 of guerrilla networking. Cheap, simple, gets all sorts of dirty jobs done.*" ». Il ne fonctionnait traditionnellement qu'avec IPv4 mais bien des mises en œuvre de GRE avaient été adaptées à IPv6 depuis longtemps. Ce RFC décrit cette adaptation.

GRE, spécifié dans les RFC 2784[1] et RFC 2890, permet de transporter n'importe quoi sur n'importe quoi. On peut faire de l'IPv4 dans IPv4, de l'IPv6 dans l'IPv6, de l'IPv4 dans IPv6 ou le contraire. En termes GRE, IPv6 peut être la charge utile ("*payload*", IPv6 sur autre chose) ou bien le support ("*delivery*", quelque chose sur IPv6).

Le format des en-têtes GRE (RFC 2784, section 2) ne change pas avec ce RFC. Il y a juste la forte recommandation de ne **pas** utiliser la somme de contrôle GRE. Cela diminue la charge de calcul pour les machines aux extrémités du tunnel GRE, charge qui était peu utile car cette somme de contrôle GRE est largement redondante avec celle que fournissent IP (pour IPv4 seulement), TCP ou UDP. Néanmoins, on a toujours le droit de mettre une somme de contrôle, en cas d'environnement très hostile. (Cf. RFC 6935 à propos des sommes de contrôle et des tunnels.)

D'abord, le cas où IPv6 est la charge utile (section 3 de notre RFC), encapsulé dans... ce qu'on veut, IPv4 ou IPv6. Dans l'en-tête GRE, le champ "*Protocol Type*" (RFC 2784, section 2.4) doit être mis à 0x86DD (c'est la même valeur que lorsque IPv6 est porté sur Ethernet, cf. RFC 7042).

1. Pour voir le RFC de numéro NNN, https://www.ietf.org/rfc/rfcNNN.txt, par exemple https://www.ietf.org/rfc/rfc2784.txt

Qui dit tunnel dit problèmes de MTU. Pour limiter les dégâts, notre RFC impose que tout tunnel GRE qui transporte de l'IPv6 ait une MTU d'au moins 1 280 octets, ce qui est la valeur minimum requise par IPv6. (Plus précisement, notre RFC demande que le tunnel puisse transporter des paquets de 1 280 octets sans utiliser la fragmentation IP. Cela peut se faire en ayant une MTU ¿= 1 280 octets ou bien en ayant un système de fragmentation et réassemblage sous IP.) Avec cette règle, on est au moins sûrs que les paquets IPv6 prudents (ceux limités à 1 280 octets <https://www.bortzmeyer.org/fragmentation-ip-1280.html>) passeront.

En théorie, une mise en œuvre standard de GRE doit tester cette possibilité, avant d'envoyer des paquets IPv6. (Celles d'aujourd'hui ne le font apparemment pas souvent, cf. RFC 7588.)

Et si le point d'entrée du tunnel GRE reçoit un paquet qui est plus grand que la MTU du tunnel ? Rappelez-vous qu'une des principales différences entre IPv4 et IPv6 est que, en IPv6, les routeurs sur le trajet n'ont pas le droit de fragmenter : si c'est nécessaire, cela doit être fait à la source. (Notez aussi que les machines d'entrée et de sortie du tunnel font un travail proche de celui d'un routeur mais que ce ne sont pas exactement des routeurs.) Donc, si le paquet trop gros arrive, le point d'entrée du tunnel doit faire ce que fait un routeur IPv6 : jeter le paquet et envoyer un message ICMP "*Packet Too Big*".

Et le contraire, IPv6 comme réseau sous-jacent, pour transporter n'importe quoi, de l'IPv4 ou de l'IPv6 (section 4 de notre RFC) ? Le paquet IPv6 va comporter l'en-tête IPv6, zéro, un ou davantage en-têtes d'extensions, puis la charge utile, faite d'un en-tête GRE et des données (un paquet IPv4 ou IPv6). S'il n'y a pas d'en-têtes d'extension IPv6, le champ "*Next Header*" d'IPv6 vaut 47 (la valeur enregistrée pour GRE <https://www.iana.org/assignments/protocol-numbers/protocol-numbers.xml>). S'il y a un ou plusieurs en-têtes d'extension, le dernier doit avoir, dans son champ "*Next Header*", cette même valeur 47.

Comme l'en-tête IPv6 n'a pas de somme de contrôle, une distribution du paquet à la mauvaise machine est toujours possible. Le comportement normal de cette machine qui reçoit un paquet inattendu est de le jeter silencieusement. Si elle parle GRE elle-même, elle peut faire la décapsulation et faire suivre le paquet (cela me semble très dangereux, permettant de faire relayer les paquets par des machines ≪ ouvertes ≫).

Enfin, la section 6 de notre RFC se penche sur la sécurité de GRE. Cela va vite car elle est à peu près nulle (cf. RFC 6169). Notamment, contrairement à beaucoup de tunnels modernes, GRE ne chiffre pas. Si on veut de la confidentialité pour les données transportées, il ne faut pas utiliser GRE. GRE reste très utile quand on veut simplement contourner le routage IP normal et établir une forme de routage manuellement contrôlée (j'ai décrit un exemple réel d'un tel cas <https://www.bortzmeyer.org/tunnel-over-tunnel.html>).

Comme indiqué plus haut, il existe déjà plusieurs mises en œuvre de GRE IPv6. Faisons quelques essais avec des machines Linux. D'abord, IPv6 dans IPv4 (oui, il existe d'autres protocoles que GRE pour tunneler IPv6 dans IPv4, je suis au courant, cf. RFC 7059 pour une comparaison des tunnels), donc en mode ≪ "*IPv6 payload*" ≫.

```
% sudo ip tunnel add tun0 mode gre remote  10.254.112.180 \
            local 192.168.43.49  ttl 255
% sudo ip link set tun0 up
```

On a ici créé un tunnel GRE entre la machine locale, `192.168.43.49` et l'autre extrémité du tunnel, `10.254.112.180`. On attribue maintenant des adresses IPv6 à l'interface `tun0` :

```
% sudo ip -6 addr add fda1:1a36:de6d::2 dev tun0
% sudo ip -6 route add  fda1:1a36:de6d::/48 dev tun0
```

On fait pareil de l'autre côté du tunnel (en inversant évidemment `remote` et `local`). On peut maintenant se pinguer en IPv6 même si le FAI ne le fournissait pas (c'est le cas ici, avec un VPS OVH) :

```
% ping6  fda1:1a36:de6d::2
PING fda1:1a36:de6d::2(fda1:1a36:de6d::2) 56 data bytes
64 bytes from fda1:1a36:de6d::2: icmp_seq=1 ttl=64 time=1.22 ms
64 bytes from fda1:1a36:de6d::2: icmp_seq=2 ttl=64 time=0.761 ms
64 bytes from fda1:1a36:de6d::2: icmp_seq=3 ttl=64 time=0.689 ms
^C
--- fda1:1a36:de6d::2 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2002ms
rtt min/avg/max/mdev = 0.689/0.893/1.229/0.239 ms
```

Plus amusant, tcpdump sait décoder le GRE et affiche donc bien ce qui s'est passé dans le tunnel :

```
16:46:02.240707 IP 10.254.112.180 > 192.168.43.49: GREv0, length 108: \
    IP6 fda1:1a36:de6d::1 > fda1:1a36:de6d::2: ICMP6, echo request, seq 1, length 64
16:46:02.240957 IP 192.168.43.49 > 10.254.112.180: GREv0, length 108: \
    IP6 fda1:1a36:de6d::2 > fda1:1a36:de6d::1: ICMP6, echo reply, seq 1, length 64
```

tcpdump a décapsulé le GRE, trouvé les paquets ICMP d'IPv6 et les a affichés.

Essayons maintenant l'inverse, « "*IPv6 delivery*" », c'est-à-dire de l'IPv4 encapsulé dans IPv6 (cela pourra servir lorsqu'on verra apparaître des FAI uniquement IPv6). Créons le tunnel (notez que le mode est `ip6gre` et plus `gre`) :

```
% sudo  ip -6 tunnel add tun0 mode ip6gre remote 2001:db8:2:245b::42 \
      local 2001:db8:8bd9:8bb0:666:6c7c:9bed:b390  ttl 255
% sudo ip -6 link set tun0 up
```

Donnons des adresses IPv4 :

```
% sudo ip  addr add 172.17.0.2 dev tun0
% sudo ip route add 172.17.0.0/24 dev tun0
```

Et on peut désormais pinguer en IPv4. tcpdump, là encore, sait décapsuler et afficher ce qui se passe dans le tunnel :

```
21:19:17.158171 IP6 2001:db8:2:245b::42 > 2001:db8:8bd9:8bb0:666:6c7c:9bed:b390: GREv0, length 88: \
         IP 172.17.0.1 > 172.17.0.2: ICMP echo request, id 32762, seq 1, length 64
21:19:17.238950 IP6 2001:db8:8bd9:8bb0:666:6c7c:9bed:b390 > 2001:db8:2:245b::42: GREv0, length 88: \
         IP 172.17.0.2 > 172.17.0.1: ICMP echo reply, id 32762, seq 1, length 64
```