RFC 8086 : GRE-in-UDP Encapsulation

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 9 mars 2017

Date de publication du RFC : Mars 2017

https://www.bortzmeyer.org/8086.html

Le protocole de tunnel GRE, normalisé dans les RFC 2784 [1] et RFC 7676, tourne normalement directement sur IP (numéro de protocole 47, TCP étant 6 et UDP 17). Cela présente quelques inconvénients, notamment la difficulté à passer certaines "*middleboxes*", et ce nouveau RFC propose donc une encapsulation de GRE dans UDP, et non plus directement dans IP. Un des autres avantages de cette encapsulation est que le port source UDP peut être utilisé comme une source d'entropie supplémentaire : sa vérification permet d'améliorer la (faible) sécurité de GRE. GRE sur UDP permet aussi l'utilisation de DTLS si on veut chiffrer (ce que ne fait pas GRE classiquement).

Un autre avantage est que l'encapsulation dans UDP peut améliorer les performances, dans le cas où il y a des répartiteurs de charge ECMP : ils pourront alors faire passer tous les paquets d'un même tunnel GRE par le même chemin, puisqu'ils prennent leurs décisions sur la base du tuple {protocole, adresse IP source, adresse IP destination, port source, port destination}.

Vu du réseau, un tunnel GRE sur UDP sera juste du trafic UDP normal. Attention, toutefois, le trafic UDP sur l'Internet public doit normalement obéir à certaines règles, notamment de contrôle de la congestion (ces règles figurent dans le RFC 8085). Avec TCP, c'est le protocole de transport qui s'en charge, avec UDP, c'est à l'application de le faire. Si on transporte du trafic quelconque, pas spécialement raisonnable, dans un tunnel GRE sur UDP, on viole les règles du RFC 8085. Il faut donc s'assurer que le trafic dans le tunnel a des mécanismes de contrôle de la congestion, ou bien réserver GRE sur UDP à des réseaux fermés, où on prend les risques qu'on veut. (Voir aussi la section 8 de notre RFC.)

Donc, on peut se servir de GRE sur UDP au-dessus d'IPv4 ou d'IPv6 (section 2 du RFC). La somme de contrôle UDP est très recommandée (elle est obligatoire en IPv6). On doit vérifier que le trafic transporté fera attention au contrôle de congestion. Le port source UDP doit être dans la plage des ports éphémères

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

(de 49 152 à 65 535, voir aussi la section 3.2.1). Utiliser un port par flot encapsulé facilite la tâche des équipements comme les répartiteurs de trafic. Mais on peut aussi n'utiliser qu'un seul port pour tout ce qui passe dans le tunnel et, dans ce cas, il faut le choisir de manière imprévisible, pour des raisons de sécurité (RFC 6056). Et en IPv6, merci de penser à utiliser le "*flow label*" (RFC 6438).

Le port de destination, lui, est par défaut 4754 pour de l'UDP ordinaire et 4755 pour du DTLS.

Ce protocole GRE sur UDP a eu une histoire longue et compliquée, pris dans des efforts pour fournir des mécanismes génériques d'encapsulation dans UDP (projet GUE), efforts qui n'ont guère débouché (cf. le RFC 7510 pour un autre exemple que GRE).

Voilà, après ces grands principes, le format exact (section 3). Au-dessus de l'en-tête IP (v4 ou v6), on met un en-tête UDP (RFC 768) et un en-tête GRE (RFC 2784).

La section 5 du RFC couvre le cas de DTLS (RFC 6347), qui a l'avantage de donner à GRE les moyens de chiffrer le trafic, sans modifier GRE lui-même.

Évidemment, dans l'Internet réellement existant, le problème, ce sont les "*middleboxes*" (section 7 du RFC). C'est d'ailleurs parfois uniquement à cause d'elles qu'il faut utiliser GRE sur UDP et pas GRE tout court, car certaines se permettent de bloquer les protocoles qu'elles ne connaissent pas (typiquement, tout sauf UDP et TCP).

Même en mettant GRE dans UDP, tous les problèmes ne sont pas résolus. Le trafic GRE est unidirectionnel (il y a en fait deux tunnels différents, chacun à sens unique). Il n'y est pas censé avoir des réponses au port source du trafic. Mais certaines "*middleboxes*" vont insister pour que ce soit le cas. Une solution possible, pour ces "*middleboxes*" pénibles, est de n'utiliser qu'un seul port source.

Il existe des mises en œuvre de ce RFC pour Linux et BSD. Les tests suivants ont été faits sur des machines Linux, noyaux 4.4 et 4.8. `ip tunnel` ne fournit pas de choix pour « GRE sur UDP ». Il faut passer par le système FOU ("*Foo-over-UDP*", cf. cet article de LWN <`https://lwn.net/Articles/614348/`>), qui a l'avantage d'être plus générique :

```
# modprobe fou
# lsmod|grep fou
fou                    20480  0
ip_tunnel              28672  1 fou
ip6_udp_tunnel         16384  1 fou
udp_tunnel             16384  1 fou
```

La machine qui va recevoir les paquets doit configurer FOU pour indiquer que les paquets à destination de tel port UDP sont en fait du GRE :

```
# ip fou add port 4754 ipproto 47
```

(47 = GRE) La machine émettrice, elle, doit créer une interface GRE encapsulée grâce à FOU :

```
#  ip link add name tun1 type gre \
          remote $REMOTE local $LOCAL ttl 225 \
          encap fou encap-sport auto encap-dport 4754
# ip link   set tun1 up
```

Et il faut évidemment configurer une route passant par cette interface `tun1`, ici pour le préfixe `172.16.0.0/24` :

```
# ip route add  172.16.0.0/24 dev tun1
```

Avec cette configuration, lorsque la machine émettrice pingue `172.16.0.1`, les paquets arrivent bien sur la machine réceptrice :

```
12:10:40.138768 IP (tos 0x0, ttl 215, id 10633, offset 0, flags [DF], proto UDP (17), length 116)
    172.19.7.106.46517 > 10.17.124.42.4754: [no cksum] UDP, length 88
```

On peut les examiner plus en détail avec Wireshark :

```
User Datagram Protocol, Src Port: 1121 (1121), Dst Port: 4754 (4754)
    Source Port: 1121
    Destination Port: 4754
    Length: 96
    Checksum: 0x0000 (none)
        [Good Checksum: False]
        [Bad Checksum: False]
    [Stream index: 0]
Data (88 bytes)

0000  00 00 08 00 45 00 00 54 3e 99 40 00 40 01 ef 6f   ....E..T>.@.@..o
...
```

Wireshark ne connait apparemment pas le GRE sur UDP. Mais, dans les données, on reconnait bien l'en-tête GRE (les quatre premiers octets où presque tous les bits sont à zéro, le bit C étant nul, les quatre octets suivants optionnels ne sont pas inclus, le 0x800 désigne IPv4, cf. RFC 2784), et on voit un paquet IPv4 ensuite. Pour que ce paquet soit correctement traité par la machine réceptrice, il faut le transmettre à GRE. Comme ce dernier n'a pas de mécanisme permettant de mettre plusieurs tunnels sur une même machine (l'en-tête GRE n'inclut pas d'identificateurs), il faut activer l'unique interface GRE :

```
# ip link set gre0 up
```

On voit bien alors notre ping qui arrive :

```
# tcpdump -vv -n -i gre0
tcpdump: listening on gre0, link-type LINUX_SLL (Linux cooked), capture size 262144 bytes
14:02:06.922367 IP (tos 0x0, ttl 64, id 47453, offset 0, flags [DF], proto ICMP (1), length 84)
    10.10.86.133 > 172.16.0.1: ICMP echo request, id 13947, seq 17, length 64
```

Voilà, je vous laisse faire la configuration en sens inverse.

Si vous voulez en savoir plus sur la mise en œuvre de FOU, voyez cet excellent exposé d'un des auteurs <https://www.netdev01.org/docs/herbert-UDP-Encapsulation-Linux.pdf>, Tom Herbert, cet article du même <http://people.netfilter.org/pablo/netdev0.1/papers/UDP-Encapsulation
pdf>, et enfin sa vidéo <https://www.youtube.com/watch?v=hKTD9W2C5s8>.