# Traiter des options EDNS nouvelles dans un programme

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 18 juin 2021

https://www.bortzmeyer.org/edns-option.html

---

Le protocole DNS permet d'attacher des métadonnées aux questions ou aux réponses par le biais de l'extension EDNS, normalisée dans le RFC 6891 [1]. Un certain nombre d'options sont déjà normalisées et peuvent être manipulées depuis un programme, via une bibliothèque DNS. Mais s'il s'agit d'une option EDNS nouvelle, pas encore traitée par la bibliothèque ?

On va voir cela avec l'option « RRSERIAL » actuellement en cours de discussion à l'IETF, dans le "*draft*" draft-ietf-dnsop-rrserial. Son but est de récupérer le numéro de série de la zone correspondant à une réponse. Elle est très simple, la valeur associée étant vide dans la question, et uniquement un entier sur 32 bits dans la réponse. Un serveur expérimental existe utilisant cette option, 200.1.122.30.

Déjà, on peut tester sans programmer avec dig, et son option +ednsopt. Les options EDNS ont un code, enregistré à l'IANA <https://www.iana.org/assignments/dns-parameters/dns-parameters.xml#dns-parameters-11>. RRSERIAL n'en a pas encore, donc le serveur de test utilise le code temporaire 65024 :

```
% dig +ednsopt=65024 @200.1.122.30 dateserial.example.com
...
;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags: do; udp: 4096
; OPT=65024: 78 49 7a 79 ("xIzy")
```

1. Pour voir le RFC de numéro NNN, https://www.ietf.org/rfc/rfcNNN.txt, par exemple https://www.ietf.org/rfc/rfc6891.txt

Ça marche, on a obtenu une réponse. dig ne sait pas la formater proprement (≪ xIzy ≫ est la représentation de 78 49 7a 79 interprété comme de l'ASCII, alors qu'il s'agit d'un entier, le nombre 2018081401), mais c'est déjà ça. (Le "*draft*" précise que l'option RRSERIAL a une valeur nulle dans la requête, seule sa présence compte. S'il avait fallu donner une valeur, dig permet de le faire avec `+ednsopt=CODE:VALEUR`.) Donc, le serveur fonctionne bien. Maintenant, on voudrait faire mieux et donc utiliser un client DNS adapté (d'autant plus qu'il y a des cas à traiter comme la réponse NXDOMAIN, où le numéro de série de la zone est dans un enregistrement SOA, pas dans l'option EDNS). On va donc programmer.

Commençons en Python avec la bibliothèque dnspython `<https://www.dnspython.org/>`. On peut fabriquer l'option dans la requête avec la classe `GenericOption` :

```
opts = [dns.edns.GenericOption(dns.edns.RRSERIAL, b'')]
...
message = dns.message.make_query(qname, qtype, options=opts)
```

(Le `b''` indique une valeur binaire vide.) Pour lire l'option dans la réponse :

```
for opt in response.options:
   if opt.otype == dns.edns.RRSERIAL:
      print("Serial of the answer is %s" % struct.unpack(">I", opt.data)[0])
```

On a donc juste à convertir la valeur binaire en chaine (le `>I` signifie un entier gros-boutien). Le code Python complet est en .

Pour Go, on va utiliser la bibliothèque godns `<https://github.com/miekg/dns>`. Créer l'option et l'ajouter à la requête DNS (`m` dans le code) se fait ainsi :

```
m.Question = make([]dns.Question, 1)
// Tout pseudo-enregistrement EDNS a pour nom "." (la racine)
o.Hdr.Name = "."
o.Hdr.Rrtype = dns.TypeOPT
o.SetUDPSize(4096)
// Option EDNS générique
e := new(dns.EDNS0_LOCAL)
e.Code = otype
// Requête vide
e.Data = []byte{}
o.Option = append(o.Option, e)
// Extra est la section Additionnelle
m.Extra = append(m.Extra, o)
```

Et pour lire le résultat :

```
opt := msg.IsEdns0()
for _, v := range opt.Option {
    // Merci à Tom Thorogood pour le rappel qu'il faut forcer le type
    // et donc avoir une nouvelle variable v (le ':=').
    switch v := v.(type) {
        case *dns.EDNS0_LOCAL:
            if v.Option() == otype {
                serial := binary.BigEndian.Uint32(v.Data)
                fmt.Printf("EDNS rrserial found, \"%d\"\n", serial)
            ...
```

Le code Go complet est en .