RFC 6891 : Extension Mechanisms for DNS (EDNS(0))

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 17 avril 2013

Date de publication du RFC : Avril 2013

https://www.bortzmeyer.org/6891.html

L'extension EDNS au traditionnel DNS a plus de treize ans et fête cette longue durée avec une nouvelle version, essentiellement cosmétique. EDNS a servi à faire sauter un certain nombre de barrières qui limitait la croissance du DNS, notamment l'antédiluvienne limite des réponses à seulement 512 octets.

Le protocole DNS, dans sa forme originale, spécifiée dans le RFC 1034 [1], ne permettait pas de négocier des options, d'indiquer au serveur ce que sait faire le client, en plus des capacités minimales qu'impose le protocole. La norme originale (RFC 1035, section 2.3.4) imposait une limite de 512 octets aux messages DNS envoyés sur UDP. Une telle limite est bien trop basse, depuis longtemps, par exemple pour DNSSEC (section 3 de notre RFC), et ne correspondait pas aux capacités des réseaux et des machines modernes. Le DNS avait d'autres limites comme des champs de taille fixe, ne permettant qu'un petit nombre de valeurs possibles, désormais presque toutes définies (par exemple, avant EDNS, les codes de réponse - "*RCODE*" - ne faisaient que quatre bits, donc étaient presque tous épuisés <https://www.iana.org/assignments/dns-parameters/dns-parameters.xml#dns-parameters-6>).

EDNS0 est un mécanisme d'extension du DNS et une première extension, pour indiquer une taille supérieure aux 512 octets. L'extension se fait en squattant des champs inutilisés du paquet (DNS est un format binaire rigide, il ne permet donc pas facilement d'ajouter de nouvelles possibilités) et en créant un pseudo-type d'enregistrement, le type OPT. Déclaré comme indispensable par certaines autres extensions (notamment DNSSEC), EDNS fait aujourd'hui partie du bagage nécessaire à toute mise en œuvre du DNS.

L'extension pour indiquer la taille permet au client de spécifier la quantité d'octets qu'il est capable de recevoir (section 4.3). Avec le client DNS dig, cela se fait avec l'option `bufsize` (désormais activée par défaut dans les versions récentes de dig). Notre RFC recommande une valeur par défaut de 4 096 octets (section 6.2.5).

Prenons par exemple le TLD `.hk` car c'est un des plus gros en nombre de serveurs de noms. Si je demande cette liste :

1. Pour voir le RFC de numéro NNN, `https://www.ietf.org/rfc/rfcNNN.txt`, par exemple `https://www.ietf.org/rfc/rfc1034.txt`

```
% dig NS hk.

; <<>> DiG 9.8.1-P1 <<>> NS hk.
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 65212
;; flags: qr rd ra; QUERY: 1, ANSWER: 10, AUTHORITY: 0, ADDITIONAL: 14

;; QUESTION SECTION:
;hk. IN NS

;; ANSWER SECTION:
hk. 69658 IN NS X.HKIRC.NET.hk.
hk. 69658 IN NS T.HKIRC.NET.hk.
hk. 69658 IN NS S.HKIRC.NET.hk.
hk. 69658 IN NS A.HKIRC.NET.hk.
hk. 69658 IN NS B.HKIRC.NET.hk.
hk. 69658 IN NS W.HKIRC.NET.hk.
hk. 69658 IN NS Z.HKIRC.NET.hk.
hk. 69658 IN NS U.HKIRC.NET.hk.
hk. 69658 IN NS Y.HKIRC.NET.hk.
hk. 69658 IN NS V.HKIRC.NET.hk.

;; ADDITIONAL SECTION:
A.HKIRC.NET.hk. 69658 IN A 203.119.2.18
B.HKIRC.NET.hk. 69658 IN A 203.119.87.19
B.HKIRC.NET.hk. 69658 IN AAAA 2001:dca:1000::cb77:5713
S.HKIRC.NET.hk. 69658 IN A 128.32.136.3
S.HKIRC.NET.hk. 69658 IN AAAA 2607:f140:ffff:fffe::3
T.HKIRC.NET.hk. 69658 IN A 128.32.136.14
T.HKIRC.NET.hk. 69658 IN AAAA 2607:f140:ffff:fffe::e
U.HKIRC.NET.hk. 69658 IN A 210.201.138.58
U.HKIRC.NET.hk. 69658 IN AAAA 2404:0:10a0::58
V.HKIRC.NET.hk. 69658 IN A 204.61.216.46
V.HKIRC.NET.hk. 69658 IN AAAA 2001:500:14:6046:ad::1
W.HKIRC.NET.hk. 69658 IN A 202.12.28.140
W.HKIRC.NET.hk. 69658 IN AAAA 2001:dc0:1:0:4777::140
X.HKIRC.NET.hk. 69658 IN A 202.45.188.39

;; Query time: 1 msec
;; SERVER: 130.129.5.6#53(130.129.5.6)
;; WHEN: Tue Mar 12 19:30:33 2013
;; MSG SIZE  rcvd: 486
```

On voit que la réponse était proche des 512 octets et que, pour qu'elle tienne dans cette limite, le serveur a dû sérieusement réduire la taille de la section additionnelle ("*additional section*"). Si le serveur avait dû réduire encore plus, jusqu'à retirer des enregistrements de la section réponse ("*answer section*"), il aurait dû mettre le bit TC (troncation) à VRAI, imposant ainsi au client de reessayer en TCP.

Mais EDNS0 permet d'avoir la totalité de la section additionnelle (notez la pseudo-section lié à l'enregistrement OPT) :

```
% dig +bufsize=4096 NS hk.

; <<>> DiG 9.8.1-P1 <<>> +bufsize=4096 NS hk.
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 50455
;; flags: qr rd ra; QUERY: 1, ANSWER: 10, AUTHORITY: 0, ADDITIONAL: 20
```

```
;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;hk. IN NS

;; ANSWER SECTION:
hk. 69632 IN NS Z.HKIRC.NET.hk.
hk. 69632 IN NS S.HKIRC.NET.hk.
hk. 69632 IN NS A.HKIRC.NET.hk.
hk. 69632 IN NS X.HKIRC.NET.hk.
hk. 69632 IN NS T.HKIRC.NET.hk.
hk. 69632 IN NS V.HKIRC.NET.hk.
hk. 69632 IN NS W.HKIRC.NET.hk.
hk. 69632 IN NS Y.HKIRC.NET.hk.
hk. 69632 IN NS U.HKIRC.NET.hk.
hk. 69632 IN NS B.HKIRC.NET.hk.

;; ADDITIONAL SECTION:
A.HKIRC.NET.hk. 69632 IN A 203.119.2.18
B.HKIRC.NET.hk. 69632 IN A 203.119.87.19
B.HKIRC.NET.hk. 69632 IN AAAA 2001:dca:1000::cb77:5713
S.HKIRC.NET.hk. 69632 IN A 128.32.136.3
S.HKIRC.NET.hk. 69632 IN AAAA 2607:f140:ffff:fffe::3
T.HKIRC.NET.hk. 69632 IN A 128.32.136.14
T.HKIRC.NET.hk. 69632 IN AAAA 2607:f140:ffff:fffe::e
U.HKIRC.NET.hk. 69632 IN A 210.201.138.58
U.HKIRC.NET.hk. 69632 IN AAAA 2404:0:10a0::58
V.HKIRC.NET.hk. 69632 IN A 204.61.216.46
V.HKIRC.NET.hk. 69632 IN AAAA 2001:500:14:6046:ad::1
W.HKIRC.NET.hk. 69632 IN A 202.12.28.140
W.HKIRC.NET.hk. 69632 IN AAAA 2001:dc0:1:0:4777::140
X.HKIRC.NET.hk. 69632 IN A 202.45.188.39
X.HKIRC.NET.hk. 69632 IN AAAA 2405:3001:1:58::1:39
Y.HKIRC.NET.hk. 69632 IN A 137.189.6.21
Y.HKIRC.NET.hk. 69632 IN AAAA 2405:3000:3:60::21
Z.HKIRC.NET.hk. 69632 IN A 194.146.106.70
Z.HKIRC.NET.hk. 69632 IN AAAA 2001:67c:1010:17::53

;; Query time: 1 msec
;; SERVER: 130.129.5.6#53(130.129.5.6)
;; WHEN: Tue Mar 12 19:30:59 2013
;; MSG SIZE  rcvd: 613
```

Et voilà, tout le monde est désormais content.

La section 6 décrit le pseudo-enregistrement OPT, dont la présence marque un paquet comme conforme à EDNS. Il est situé dans la section additionnelle du message DNS, a le type 41, le nom de domaine est forcément . (la racine), la classe est détournée de son rôle normal pour indiquer la taille des paquets que l'envoyeur pourra recevoir en retour et le champ TTL est également détourné de son usage normal pour offrir de nouvelles options et de nouveaux codes de retour (rappelez-vous qu'il n'y avait que quatre bits pour ces codes dans le DNS original). Pour l'instant, une seule <https://www.iana.org/assignments/dns-parameters/dns-parameters.xml#dns-parameters-13> nouvelle option, le bit DO ("*DNSSEC OK*") qui indique la capacité de l'émetteur à traiter les signatures DNSSEC (il avait été normalisé dans le RFC 3225).

L'ex-champ TTL sert aussi à indiquer le numéro de version d'EDNS, zéro actuellement (d'où le nom EDNS0 qu'on utilise parfois). Programmeurs, attention, certaines API (par exemple celle de DNS Python <https://www.bortzmeyer.org/dnspython.html>) nécessitent d'indiquer le numéro de version pour activer EDNS et il faut donc indiquer zéro pour dire qu'on veut de l'EDNS, ce qui peut être déroutant.

Ensuite, les données peuvent contenir plusieurs options. À l'heure actuelle, l'écrasante majoritéé des paquets EDNS n'en contiennent aucune. Mais, si c'est le cas, elles sont codées en TLV, un code indiquant l'option (les valeurs possibles étant dans un registre IANA <https://www.iana.org/assignments/dns-parameters/dns-parameters.xml#dns-parameters-11>), une longueur et les données spécifiques à l'option. Un exemple d'une telle option est le NSID du RFC 5001.

On notera qu'EDNS est ce que le DNS appelle une extension "*hop by hop*" ce qui veut dire qu'elle s'applique entre deux logiciels adjacents, pas de bout en bout. Si un client DNS demande à un résolveur local, qui demande à son tour à un serveur de `.re`, les options EDNS mises par le client sont transmises uniquement au résolveur, qui fabriquera ensuite une requête différente (peut-être avec d'autres options), pour le serveur faisant autorité pour `.re`.

Encore aujourd'hui, il existe des serveurs qui ne gèrent pas EDNS. Par exemple, ceux de `microsoft.com` répondent `FORMERR` ("*Format Error*") :

```
% dig +bufsize=4096 @ns2.msft.net. SOA microsoft.com.

; <<>> DiG 9.8.1-P1 <<>> +bufsize=4096 @ns2.msft.net. SOA microsoft.com.
; (2 servers found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: FORMERR, id: 54935
...
```

La section 6.2.2 de notre RFC précise donc qu'un émetteur intelligent peut alors se rabattre sur du DNS classique et ne pas envoyer l'enregistrement OPT :

```
% dig @ns2.msft.net. SOA microsoft.com.

; <<>> DiG 9.8.1-P1 <<>> @ns2.msft.net. SOA microsoft.com.
; (2 servers found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 22484
;; flags: qr aa rd; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 2
;; WARNING: recursion requested but not available

;; QUESTION SECTION:
;microsoft.com. IN SOA

;; ANSWER SECTION:
microsoft.com. 3600 IN SOA ns1.msft.net. msnhst.microsoft.com. 2013031202 300 600 2419200 3600
...
```

Au moins, les serveurs de `microsoft.com` répondent, même si c'est par un refus. Plus embêtant, les serveurs qui ne répondent plus du tout lorsque la requête est en EDNS, en général en raison d'une boîte noire mal programmée et mal gérée installée devant le serveur (un pare-feu par exemple : beaucoup d'administrateurs réseaux ne supportent pas que le serveur DNS marche bien et mettent donc une "*middlebox*" boguée devant, cf. section 8 de notre RFC, ainsi que le RFC 5625). Un logiciel comme BIND, lorsqu'il ne reçoit pas de réponse, réessaie sans EDNS pour voir si c'est cela la cause du problème. Cette

possibilité est décrite en section 6.2.5 qui recommande d'essayer d'abord EDNS avec une taille plus petite (inférieure à la MTU, au cas où le problème soit lié à la fragmentation), puis enfin sans EDNS.

L'EDNS original, du RFC 2671, prévoyait également des nouvelles façons de stocker les composants d'un nom de domaine, en plus des deux méthodes DNS traditionnelles, comprimée ou non (sections 4.2 et 5 de notre RFC et section 4.1.4 du RFC 1035). Cette pratique a été très peu utilisée, en raison des difficultés de déploiement (cf. RFC 3363 et RFC 3364). Notre RFC 6891 abandonne donc cette possibilité et reclasse le RFC 2673 (le seul qui avait utilisé un nouveau type de composants, les « composants binaires ») dans le cimetière des RFC dépassés. C'est le principal changement de ce nouveau RFC (les autres étant plutôt des détails, cf. annexe A.) Notons aussi que ce nouveau RFC est désormais « Norme Internet » et plus simplement « Proposition de norme ».

Pour les amateurs de programmation, du code C d'analyse d'un paquet DNS contenant de l'EDNS est dans mon article « Décoder les paquets DNS capturés avec pcap <https://www.bortzmeyer.org/pcap-decodage-dns.html> ».

Quelques exemples de code pour finir. Pour analyser un enregistrement OPT d'EDNS, voir comment c'est fait dans DNSmezzo <https://github.com/AFNIC/DNSwitness/commit/97c7ff17881c7f75436b4e26567
Ensuite, pour envoyer des requêtes EDNS, en Go, avec godns <https://github.com/miekg/dns> :

```
const (
EDNSBUFFERSIZE  uint16  = 4096
)
...
m := new(dns.Msg)
...
m.SetEdns0(EDNSBUFFERSIZE, true)
```

En Python, avec DNSpython <https://www.bortzmeyer.org/dnspython.html> (notez qu'on indique la version d'EDNS donc zéro pour activer EDNS n'est pas une erreur) :

```
message = dns.message.make_query(name, type, use_edns=0, payload=4096)
```

Et pour finir, en C, si vous assemblez le paquet à la main (il y a évidemment des façons plus simples de faire de l'EDNS), le code pourra ressembler à :

```
 /* OPT pseudo-RR */
 after_qname[4] = 0;     /* root domain */
 /* OPT = type 41 */
 after_qname[5] = 0;
 after_qname[6] = 41;
 /* Class stores the payload size */
 bufsize_wire = htons(bufsize);
 memmove(after_qname + 7, &bufsize_wire, sizeof(bufsize));
 /* TTL store the RCODE and flags */
 after_qname[9] = 0;
 after_qname[10] = 0;
 if (dnssec) {
     after_qname[11] = 128;      /* DO: request DNSSEC signatures */
 } else {
     after_qname[11] = 0;
 }
 after_qname[12] = 0;
 /* Resource data length (empty, here) */
 after_qname[13] = 0;
 after_qname[14] = 0;
```