RFC 8490 : DNS Stateful Operations

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 18 mars 2019

Date de publication du RFC : Mars 2019

https://www.bortzmeyer.org/8490.html

Autrefois, le DNS était toujours cité comme exemple d'un protocole sans état. On envoie une requête, on reçoit une réponse, et le client et le serveur oublient aussitôt qu'ils ont échangé, ils ne gardent pas de trace de cette communication. Mais, dans certains cas, maintenir un état sur une durée plus longue qu'un simple échange requête/réponse peut être utile. Ce nouveau RFC propose un mécanisme pour des **sessions** DNS, le mécanisme DSO ("*DNS Stateful Operations*"). Il introduit donc une nouvelle notion dans le DNS, la persistance des sessions.

Ne pas avoir d'état a de nombreux avantages : cela simplifie les programmes, cela augmente les performances considérablement (pas besoin de chercher dans une table l'état actuel d'un dialogue, le contenu de la requête est suffisant pour donner une réponse, on peut répondre à la vitesse de l'éclair) et cela permet de résister aux DoS, qui réussissent souvent lorsqu'elles arrivent à épuiser le système qui dépend d'un état. (C'est pour cela que c'est souvent une mauvaise idée de mettre un pare-feu à état devant un serveur Internet, et c'est même franchement absurde quand il s'agit d'un serveur DNS.) Le DNS « habituel », tournant sur UDP et sans maintenir d'état, doit une partie de son succès à son caractère sans état.

Mais ne pas avoir d'état a aussi des inconvénients : toutes les options, tous les choix doivent être répétés dans chaque requête. Et cela rend impossible de négocier des paramètres entre les deux parties, par exemple dans le cas d'une session cryptographiquement protégée. Bref, dans certains cas, on aimerait bien avoir une vraie session, de durée relativement longue (plusieurs secondes, voire plusieurs minutes). Le DNS a un mécanisme de connexion de longue durée, en utilisant TCP (RFC 7766 [1]), et peut utiliser TLS pour sécuriser cette communication (DoT, « "*DNS over TLS*" », RFC 7858) mais les requêtes à l'intérieur de cette connexion n'en profitent pas, elles ne savent pas qu'elles sont liées par le fait qu'elles sont dans la même connexion. D'où ce nouveau système.

1. Pour voir le RFC de numéro NNN, https://www.ietf.org/rfc/rfcNNN.txt, par exemple https://www.ietf.org/rfc/rfc7766.txt

Le principe de DSO ("*DNS Stateful Operations*") est de permettre à une requête DNS de créer une session, avec des paramètres communs à toute la session (comme la durée maximale d'inactivité). La session est balisée par des requêtes DNS utilisant l'"*opcode*" DSO, de numéro 6 (la création d'un nouvel opcode <https://www.iana.org/assignments/dns-parameters/dns-parameters.xml#dns-parameters est très rare). Les paramètres sont encodés en TLV (une nouveauté dans le monde DNS ; les traditionnels "*Query count*" et "*Answer count*", avec les sections correspondantes, ne sont pas utilisés). La longueur du message DSO est indiquée par les deux premiers octets du message. Les messages DSO peuvent solliciter une réponse (même si c'est un simple accusé de réception) ou pas. Cette sollicitation est faite par un "*Message ID*" différent de zéro. Si, par contre, le "*Message ID*" DNS est à zéro, il s'agit d'un message DSO unidirectionnel (retenez ce terme, il va souvent servir dans ce RFC), qui n'attend pas de réponse. (Rappelez-vous que le "*Message ID*" sert à faire correspondre requêtes et réponses DNS. Si on n'attend pas de réponse, pas besoin d'un "*Message ID*". Si par contre le message est bidirectionnel, il doit mettre un "*Message ID*" non nul.)

DSO ("*DNS Stateful Operations*", sessions - avec état, donc - pour le DNS) ne s'applique qu'avec certains transports sous-jacents (section 4 du RFC). UDP est évidemment exclu, car il faut maintenir l'ordre des messages, et il faut qu'il y ait une connexion à gérer. Cela peut être TCP (RFC 1035, section 4.2.2 et RFC 7766) ou DoT (DNS sur TLS, RFC 7858). DoH (DNS sur HTTPS, RFC 8484) est par contre exclu car HTTP a ses propres mécanimes de gestion de session. (D'autre part, la section 9.2 décrit les conséquences que cela a pour l'"*anycast*".)

Deux importantes utilisations de DSO sont prévues :

— Gestion de sessions et des paramètres associés : DSO va permettre de définir des paramètres comme les durées maximales d'inactivité avant qu'on ne coupe la connexion de transport sous-jacente. Dans ce cas, DSO est une alternative au RFC 7828.
— Abonnements de longue durée à des services comme la découverte (RFC 6763).

La section 5 est le gros du RFC, elle décrit tous les détails du protocole. Pour établir une session DSO, il faut :

— Établir une connexion avec un protocole comme TCP ; on est alors connecté (on peut envoyer des messages DNS et recevoir des réponses) mais sans session DSO,
— On envoie une demande DSO,
— Si le correspondant est d'accord, on reçoit une réponse DSO, et la session est établie, et les paramètres comme la durée d'inactivité maximale sont désormais contrôlés par DSO ; on peut envoyer des messages DSO unidirectionnels (non sollicités, et ne demandant pas de réponse),
— Si par contre le correspondant refuse DSO, on continue avec une connexion normale.

Si on sait à l'avance que le correspondant gère DSO, on peut se considérer comme en session dès l'établissement de la connexion. Mais, souvent on ne sait pas ou on n'est pas sûr et il faut donc explicitement ouvrir une session. Cela se fait avec un message DSO (un message où l'"*opcode*" DNS vaut 6 ; ces "*opcodes*" sont décrits dans le RFC 1035, section 4.1.1). L'acceptation prend la forme d'un message DSO avec un "*Message ID*" qui correspond et un code de réponse 0 ("*rcode*" = NOERROR). Si le code de réponse est autre chose que NOERROR (par exemple 4, NOTIMP, « type de requête inconnu » ou 5, REFUSED, « je connais peut-être DSO mais je n'ai pas envie d'en faire »), c'est que notre correspondant ne peut pas ou ne veut pas établir une session.

Il n'y a pas de message DSO dédié à l'ouverture de session. On envoie un message DSO de n'importe quel type (par exemple "*Keepalive*"). Il peut donc arriver que le copain en face connaisse DSO mais pas ce type particulier. Dans ce cas, il va répondre DSOTYPENI ("*DSO Type Not Implemented*", code 11, une nouveauté dans le registre <https://www.iana.org/assignments/dns-parameters/dns-parameters.xml#dns-parameters-6>). La session n'est pas établie et le client doit recommencer avec un autre type (comme "*Keepalive*", qui a l'avantage d'être normalisé depuis le début et d'être obligatoire, donc il marchera partout).

Il y a des cas plus gênants : un serveur qui couperait la connexion de transport sous-jacente, ou bien qui ne répondrait pas aux messages DSO. Ce cas risque de se produire si un boitier intermédiaire

bogué est sur le trajet. Il peut être alors nécessaire d'adopter des mesures de contournement comme celles qu'utilisaient les résolveurs DNS avec les serveurs ne gérant pas bien EDNS, mesures de contournement qui ont été abandonnées récemment avec le "*DNS Flag Day*" <https://www.afnic.fr/fr/ressources/blog/1er-fevrier-2019-le-dns-va-t-il-trembler.html>.

Si, par contre, tout se passe bien, la session DSO est établie, et des paramètres comme le délai d'inactivité doivent désormais suivre les règles de DSO et plus celles de normes précédentes comme le RFC 7766 (c'est pour cela que notre RFC met à jour le RFC 7766).

La section 5 détaille également le format des messages DSO. Ce sont des messages DNS ordinaires, commençant par le "*Message ID*" sur deux octets, avec l'"*opcode*" qui vaut DSO (code numérique 6). Les champs qui indiquent le nombre d'enregistrements dans les différentes sections doivent tous être mis à zéro. Les données DSO sont situées après l'en-tête DNS standard, et sont sous forme de TLV. Le logiciel peut donc analyser ces données même s'il ne connait pas un type DSO spécifique. Dans une requête DSO, il y a toujours au moins un TLV, le « TLV primaire », qui indique le type d'opérations. Les autres éventuels TLV (« TLV additionnels ») sont là pour préciser le message. Rappelons qu'il y a deux sortes de messages DSO, les unidirectionnels et les autres. Les unidirectionnels ont le "*Message ID*" à zéro et n'ont jamais de réponse. (Avec le "*Message ID*" à zéro, on ne saurait de toute façon pas à quelle demande correspond une réponse.)

Chaque TLV comprend trois champs :
— Le type, sur deux octets (la liste des types possibles figure dans un registre IANA <https://www.iana.org/assignments/dns-parameters/dns-parameters.xml#dns-dso-type-codes> créé par ce RFC),
— La longueur des données, sur deux octets,
— Les données.

Notez que la définition de chaque type doit préciser s'il est censé être utilisé en TLV primaire ou additionnel. Pour une réponse, il peut n'y avoir aucun TLV présent.

Toutes les sections « normales » d'un message DNS sont vides, y compris la section additionnelle qu'utilise EDNS (le champ `ARCOUNT` doit être à zéro). Il ne peut donc pas y avoir d'options EDNS dans un message DSO (pour éviter la confusion qui se produirait si une option EDNS et un message DSO donnaient des valeurs différentes au même service). Si on veut le service équivalent à une option EDNS, il faut créer un nouveau type DSO (section 10.3 du RFC pour les détails) et le faire enregistrer <https://www.iana.org/assignments/dns-parameters/dns-parameters.xml#dns-dso-type-codes>.

Combien de temps durent les sessions DSO ? D'un côté, il faut qu'elles soient aussi longues que possible, pour amortir le coût de créer et de maintenir des sessions sur un grand nombre de requêtes, d'un autre, il ne faut pas gaspiller des ressorces à maintenir une session ouverte si elle ne sert plus à rien. La section 6 du RFC discute cette question. DSO a un délai maximal d'inactivité et, quand le délai est dépassé sans activité, le client DSO est censé couper la connexion. (S'il ne le fait pas, le serveur le fera, après un délai plus long.) Le client a évidemment le droit de couper la session avant l'expiration du délai, s'il sait qu'il n'en aura plus besoin.

Le délai maximal d'inactivité est fixé par les messages DSO de type 1. Deux cas spéciaux : zéro indique qu'on doit fermer la connexion immédiatement après la première requête, et 0xFFFFFFFF indique que la session peut être gardée ouverte aussi longtemps qu'on le souhaite.

DSO permet également de spécifier l'intervalle de génération des messages "*keepalives*", messages envoyés périodiquement uniquement pour que les boitiers de traduction d'adresse gardent leur état et ne suppriment pas une correspondance adresse interne ¡-¿ adresse externe en pensant qu'elle ne sert

plus. Si on sait qu'il n'y a pas de NAT sur le trajet, on peut mettre un intervalle très élevé. Le client peut aussi se dire « j'ai une adresse RFC 1918, le serveur a une adresse IP publique, il y a donc sans doute un machin NAT sur le trajet, je demande des "*keepalives*" fréquents ».

Enfin, le client doit être préparé à ce que le serveur ferme la session à sa guise, parce que le serveur estime que le client exagère (il ne ferme pas la session alors que le délai d'inactivité est dépassé, et qu'il n'envoie pas de requêtes), ou bien parce que le serveur va redémarrer. Normalement, c'est le client DSO qui ferme la session mais, dans certains cas, le serveur peut décider de le faire.

La section 7 du RFC décrit les trois TLV de base qui doivent être présents dans toutes les mises en œuvre de DSO : "*keepalive*", délai avant de réessayer, et remplissage. La section 8.2 indique dans quels cas ils peuvent être utilisés par le client ou par le serveur.

Le TLV "*keepalive*" contrôle l'envoi de messages servant uniquement à indiquer que la session est toujours ouverte, afin notamment de rassurer les routeurs NAT. Ce même TLV sert également à indiquer le délai d'inactivité maximal. Comme ce type de TLV est obligatoire, c'est un bon candidat pour le message initial d'ouverture de session (il n'y a pas de message particulier pour cette ouverture : on envoie juste un message ordinaire). Il a le type 1 et comprend deux champs de données, le délai maximal d'inactivité, en millisecondes, sur quatre octets, et l'intervalle d'émission des "*keepalives*", également en millisecondes, et sur quatre octets. Il peut être utilisé comme TLV primaire, et il requiert une réponse, le "*Message ID*" doit donc être différent de zéro. La valeur du délai maximal d'inactivité émise par le client est un souhait, la valeur à utiliser est celle qui figure dans la réponse du serveur. Si le client ne la respecte pas par la suite, le serveur aura le droit de fermer la session. Notez qu'EDNS avait déjà un mécanisme équivalent, pour définir une durée d'inactivité maximale dans les connexions TCP, normalisé dans le RFC 7828. Mais les limites d'EDNS, comme le fait que les options EDNS ne s'appliquent normalement qu'au message en cours, rendent cette solution peu satisfaisante. Cet ancien mécanisme ne doit donc pas être utilisé avec DSO, qui dispose, d'un autre système, celui utilisant les valeurs spécifiées par un message portant le TLV "*Keepalive*".

Une fois la durée d'émission des "*keepalives*" fixée, les messages de "*keepalive*" seront des messages unidirectionnels (pas de réponse) et donc envoyés avec un "*Message ID*" nul.

Deuxième type de TLV obligatoire, le délai avant de réessayer de se connecter, qui a le code 2. C'est un message unidirectionnel, envoyé par le serveur pour indiquer qu'il va couper et qu'il ne faut pas réessayer avant la durée indiquée en valeur du TLV.

Et enfin, le troisième type (code 3) qui doit être présent dans toute mise en œuvre de DSO est le remplissage. Le but est d'améliorer la protection de la vie privée en insérant des données bidon dans les messages DNS, pour rendre plus difficile l'analyse des données chiffrées. Il n'a évidemment de sens que si la session sous-jacente est chiffrée, par exemple avec le RFC 7858. Pour la longueur du remplissage à choisir, voir le RFC 8467.

Comme toujours sur l'Internet, une grande partie des problèmes opérationnels viendront des "*middleboxes*". Le RFC rappelle à juste titre que la meilleure solution serait de ne pas avoir de "*middleboxes*" mais, comme c'est un idéal lointain, en attendant, il faut se pencher sur ce que font ces fichus boitiers intermédiaires, qui se permettent parfois d'intercepter automatiquement le trafic DNS et de le modifier. Si le boitier gère DSO et répond correctement aux spécifications de ce RFC, tout va bien. Si le boitier ne comprend pas DSO et renvoie un NOTIMP ou équivalent, cela empêche d'utiliser DSO mais, au moins, cela ne viole pas la norme : le client réagira comme si le serveur ne connait pas DSO. Si le boitier ne connait pas le DNS, et n'essaie pas de le comprendre, ça devrait marcher si, bien sûr, il établit bien une

connexion et une seule pour chaque connexion entrante (c'est ce que fait un routeur NAT qui ne regarde pas les couches supérieures).

Dès que le boitier ne respecte pas ces règles, on peut prévoir des ennuis, et qui seront très difficiles à déboguer. Par exemple si un répartiteur de charge DNS reçoit des connexions TCP, les ouvre, et envoie chaque requête DNS qu'elles contenaient à un serveur différent, le client DSO va certainement souffrir. Il croira avoir une session alors qu'il n'en est rien.

Autre problème pratique qui se posera peut-être : les optimisations de TCP. Deux d'entre elles ont des chances sérieuses de créer des ennuis, l'algorithme de Nagle et les accusés de réception retardés (on attend un peu de voir si un autre segment arrive, pour pouvoir accuser réception des deux avec un seul paquet, RFC 1122, section 4.2.3.2). Pour les messages DSO bidirectionnels, pas de problème. Pour les unidirectionnels, en revanche, le retard de l'accusé de réception pourra atteindre 200 millisecondes, ce qui est énorme dans un centre de données typique, avec des liens qui peuvent débiter plus d'un gibabit par seconde. L'algorithme de Nagle fera qu'on n'enverra pas de données tout de suite, attendant s'il n'y a pas quelque chose à transmettre et, avec l'accusé de réception retardé, la combinaison des deux retardera sérieusement l'envoi <http://www.stuartcheshire.org/papers/nagledelayedack/>.

Débrayer l'algorithme de Nagle, ou bien les accusés de réception retardés, résoudrait le problème mais ferait perdre d'utiles optimisations. En fait, la seule solution propre serait que les API permettent aux applications de dire à TCP « il n'y aura pas de réponse à ce message, envoie l'accusé de réception tout de suite ».

Enfin, un petit mot sur la sécurité pour finir. DSO nécessite des connexions permanentes et, potentiellement, cela peut consommer pas mal de ressources sur le serveur. Pour se protéger, le serveur a donc parfaitement le droit de limiter le nombre de connexions maximal, et de fermer des sessions quand ça lui chante.

Toujours sur la sécurité, DSO permet des établissements de connexion sans aller-retour, avec TCP "*Fast Open*" (RFC 7413) et TLS 1.3 (RFC 8446). C'est très rapide, c'est très bien mais les données envoyées avec le premier paquet ("*early data*") ne sont pas forcément bien sécurisées et la définition de chaque type de TLV doit donc indiquer s'il est sûr ou pas de l'utiliser dans le premier paquet.

Il semble qu'à l'heure actuelle, il n'y a pas encore de mise en œuvre de cette technique DSO.