RFC 8618 : Compacted-DNS (C-DNS): A Format for DNS Packet Capture

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 4 septembre 2019

Date de publication du RFC : Septembre 2019

https://www.bortzmeyer.org/8618.html

Lorsque l'opérateur d'un service DNS veut conserver les données de trafic, il peut demander au serveur d'enregistrer requêtes et réponses (mais, la plupart du temps, le serveur n'écrit qu'une petite partie des informations) ou bien écouter le trafic réseau et enregistrer le pcap. Le problème est que le format pcap prend trop de place, est de trop bas niveau (une connexion TCP, par exemple, va être éclatée en plusieurs paquets), et qu'il est difficile de retrouver les informations spécifiquement DNS à partir d'un pcap. D'où la conception de ce format de stockage, spécifique au DNS, et qui permet d'enregistrer la totalité de l'information, dans un format optimisé en taille et de plus haut niveau. C-DNS s'appuie sur CBOR pour cela.

Le DNS est un service d'infrastructure absolument critique. Il est donc nécessaire de bien le connaitre et de bien l'étudier. Cela passe par une récolte de données, en l'occurrence le trafic entrant et sortant des serveurs DNS, qu'ils soient des résolveurs ou bien des serveurs faisant autorité. Ce genre de récolte peut être coordonnée par l'OARC <https://www.dns-oarc.net/> pour des projets comme DITL <https://www.dns-oarc.net/oarc/data/ditl> (≪ un jour dans la vie de l'Internet ≫). Un exemple d'une telle récolte, faite avec le classique tcpdump (qui, en dépit de son nom, ne fait pas que du TCP) :

```
% tcpdump -w /tmp/dns.pcap port 53
```

Le fichier produit (ici, sur un serveur faisant autorité pour `eu.org`), au format pcap, contient les requêtes DNS et les réponses, et peut être analysé avec des outils comme tcpdump lui-même, ou comme Wireshark. Il y a aussi des outils spécifiques au DNS comme PacketQ <https://github.com/dotse/PacketQ> ou comme dnscap <https://www.dns-oarc.net/tools/dnscap>. Ici, avec tcpdump :

```
% tcpdump -n -r /tmp/dns.pcap
15:35:22.432746 IP6 2001:db8:aa:101::.40098 > 2400:8902::f03c:91ff:fe69:60d3.53: 41209% [1au] A? tracker.tor
15:35:22.432824 IP6 2400:8902::f03c:91ff:fe69:60d3.53 > 2001:db8:aa:101::.40098: 41209- 0/4/5 (428)
```

Au lieu des outils tous faits, on peut aussi développer ses propres programmes en utilisant les nombreuses bibliothèques qui permettent de traiter du pcap (attention si vous analysez du trafic Internet : beaucoup de paquets sont mal formés, par accident ou bien délibérément, et votre analyseur doit donc être robuste). C'est ce que font en général les chercheurs qui analysent les données DITL.

Le problème du format pcap (ou pcapng <https://github.com/pcapng/pcapng>) est qu'il y a à la fois trop de données et pas assez. Il y a trop de données car il inclut des informations probablement rarement utiles, comme les adresses MAC et car il ne minimise pas les données. Et il n'y en a pas assez car il ne stocke pas les informations qui n'étaient pas visibles sur le réseau mais qui l'étaient uniquement dans la mémoire du serveur DNS. Ainsi, on ne sait pas si la réponse d'un résolveur avait été trouvée dans le cache ou pas. Ou bien si les données étaient dans le bailliage ou pas (cf. RFC 8499 [1], section 7). Les captures DNS peuvent être de très grande taille (10 000 requêtes par seconde est banal, 100 000, ça arrive parfois) et on désire les optimiser autant que possible, pour permettre leur rapatriement depuis les serveurs éloignés, puis leur stockage parfois sur de longues périodes. (Les formats « texte » comme celui du RFC 8427 ne conviennent que pour un message isolé, ou un tout petit nombre de messages.)

Le cahier des charges du format C-DNS ("*Compacted DNS*") est donc :
— Minimiser la taille des données,
— Minimiser le temps de traitement pour compacter et décompacter.
La section du RFC détaille les scénarios d'usage de C-DNS. En effet, la capture de données DNS peut être faite dans des circonstances très différentes. Le serveur peut être une machine physique, une virtuelle, voire un simple conteneur. La personne qui gère le capture peut avoir le contrôle des équipements réseau (un commutateur, par exemple, pour faire du "*port mirroring*"), le serveur peut être surdimensionné ou, au contraire, soumis à une attaque par déni de service qui lui laisse peu de ressources. Le réseau de collecte des données capturées peut être le même que le réseau de service ou bien un réseau différent, parfois avec une capacité <https://www.bortzmeyer.org/capacite.html> plus faible. Bref, il y a beaucoup de cas. C-DNS est optimisé pour les cas où :
— La capture des données se fait sur le serveur lui-même, pas sur un équipement réseau,
— Les données seront stockées localement, au moins temporairement, puis analysées sur une autre machine.
Donc, il est crucial de minimiser la taille des données récoltées. Mais il faut aussi faire attention à la charge que représente la collecte : le serveur de noms a pour rôle de répondre aux requêtes DNS, la collecte est secondaire, et ne doit donc pas consommer trop de ressources CPU.

Compte-tenu de ces contraintes, C-DNS a été conçu ainsi (section 4 du RFC) :
— L'unité de base d'un fichier C-DNS est le couple R/R, {requête DNS, réponse DNS} ("*Q/R data item*"), ce qui reflète le fonctionnement du protocole DNS, et permet d'optimiser le stockage, puisque bien des champs ont des valeurs communes entre une requête et une réponse (par exemple le nom de domaine demandé). Notez qu'un couple R/R peut ne comporter que la requête, ou que la réponse, si l'autre n'a pas pu être capturée, ou bien si on a décidé délibérément de ne pas le faire.
— C-DNS est optimisé pour les messages DNS syntaxiquement corrects. Quand on écrit un analyseur de paquets DNS, on est frappés du nombre de messages incorrects qui circulent sur le réseau. C-DNS permet de les stocker sous forme d'un « blob » binaire, mais ce n'est pas son but principal, il ne sera donc efficace que pour les messages corrects.

1. Pour voir le RFC de numéro NNN, https://www.ietf.org/rfc/rfcNNN.txt, par exemple https://www.ietf.org/rfc/rfc8499.txt

— Pratiquement toute les données sont optionnelles dans C-DNS. C'est à la fois pour gagner de la place, pour tenir compte du fait que certains mécanismes de capture ne gardent pas toute l'information, et pour permettre de minimiser les données afin de préserver la vie privée.
— Les couples R/R sont regroupés en blocs, sur la base d'élements communs (par exemple l'adresse IP source, ou bien la réponse, notamment les NXDOMAIN), qui permettent d'optimiser le stockage, en ne gardant cet élément commun qu'une fois par bloc. (Par contre, cela complexifie les programmes. On n'a rien sans rien.)

C-DNS repose sur CBOR (RFC 7049). La section 5 du RFC explique pourquoi :

— Format binaire, donc prenant moins d'octets que les formats texte comme JSON (l'annexe C discute des autres formats binaires),
— CBOR est un format normalisé et répandu,
— CBOR est simple et écrire un analyseur peut se faire facilement, si on ne veut pas dépendre d'une bibliothèque extérieure,
— CBOR a désormais un langage de schéma, CDDL (RFC 8610), qu'utilise notre RFC.

Avec la section 6 du RFC commence la description du format. Classiquement, un fichier C-DNS commence par un en-tête, puis une série de blocs. Chaque bloc comprend un certain nombre de tables (par exemple une table d'adresses IP pour les adresses apparaissant dans le bloc), suivies des éléments R/R. Ceux-ci référencent les tables. Ainsi, une requête de `2001:db8:1::cafe` à `2001:db8:ffff::beef` pour le nom `www.example.org` contiendra des pointeurs vers les entrées `2001:db8:1::cafe` et `2001:db8:ffff::beef` de la table des adresses IP, et un pointeur vers l'entrée `www.example.org` de la table des noms de domaine. S'il n'y a qu'un seul élément R/R dans le bloc, ce serait évidemment une complication inutile, mais l'idée est de factoriser les données qui sont souvent répétées.

On l'a vu, dans C-DNS, plein de choses sont optionnelles, car deux dispositifs de capture différents ne récoltent pas forcément les mêmes données. Ainsi, par exemple, un système de capture situé dans le logiciel serveur n'a pas forcément accès à la couche IP et ne peut donc pas enregistrer le nombre maximal de sauts ("*hop limit*"). Cela veut dire que, quand on lit un fichier C-DNS :

— Il faut déterminer si une donnée est présente ou pas, et ne pas supposer qu'elle l'est forcément,
— Il peut être utile de déterminer si une donnée manquante était absente dès le début, ou bien si elle a été délibérément ignorée par le système de capture. Si un message ne contient pas d'option EDNS (RFC 6891), était-ce parce que le message n'avait pas cette option, ou simplement parce qu'on ne s'y intéressait pas et qu'on ne l'a pas enregistrée ?

C-DNS permet donc d'indiquer quels éléments des données initiales ont été délibérément ignorés. Cela permet, par exemple, à un programme de lecture de fichiers C-DNS de savoir tout de suite si le fichier contient les informations qu'il veut.

Ces indications contiennent aussi des informations sur un éventuel échantillonnage (on n'a gardé que X % des messages), sur une éventuelle normalisation (par exemple tous les noms de domaine passés en caractères minuscules) ou sur l'application de techniques de minimisation, qui permettent de diminuer les risques pour la vie privée. Par exemple, au lieu de stocker les adresses IP complètes, on peut ne stocker qu'un préfixe (par exemple un /32 au lieu de l'adresse complète), et il faut alors l'indiquer dans le fichier C-DNS produit, pour que le lecteur comprenne bien que `2001:db8::` est un préfixe, pas une adresse.

La section 7 du RFC contient ensuite le format détaillé. Quelques points sont à noter (mais, si vous écrivez un lecteur C-DNS, lisez bien tout le RFC, pas juste mon article !) Ainsi, toutes les clés des objets ("*maps*") CBOR sont des entiers, jamais des chaînes de caractère, pour gagner de la place. Et ces entiers sont toujours inférieurs à 24, pour tenir sur un seul octet en CBOR (lisez le RFC 7049 si vous voulez savoir pourquoi 24). On peut aussi avoir des clés négatives, pour les extensions au format de base, et elles sont comprises entre -24 et -1.

La syntaxe complète, rédigée dans le format CDDL du RFC 8610, figure dans l'annexe A de notre RFC.

On peut reconstruire un fichier pcap à partir de C-DNS. Une des difficultés est qu'on n'a pas forcément toutes les informations, et il va donc falloir être créatif (section 9). Une autre raison fait qu'on ne pourra pas reconstruire au bit près le fichier pcap qui aurait été capturé par un outil comme tcpdump : les noms de domaines dans les messages DNS étaient peut-être comprimés (RFC 1035, section 4.1.4) et C-DNS n'a pas gardé d'information sur cette compression. (Voir l'annexe B pour une discussion détaillée sur la compression.) Pareil pour les informations de couche 3 : C-DNS ne mémorise pas si le paquet UDP était fragmenté, s'il était dans un ou plusieurs segments TCP, s'il y avait des messages ICMP liés au trafic DNS, etc.

Si vous voulez écrire un lecteur ou un producteur de C-DNS, la section 11 du RFC contient des informations utiles pour la programmeuse ou le programmeur. D'abord, lisez bien le RFC 7049 sur CBOR. C-DNS utilise CBOR, et il faut donc connaitre ce format. Notamment, la section 3.9 du RFC 7049 donne des conseils aux implémenteurs CBOR pour produire un CBOR « canonique ». Notre RFC en retient deux, représenter les entiers, et les types CBOR, par la forme la plus courte possible (CBOR en permet plusieurs), mais il en déconseille deux autres. En effet, la section 3.9 du RFC 7049 suggérait de trier les objets selon la valeur des clés, et d'utiliser les tableaux de taille définie (taille indiquée explicitement au début, plutôt que d'avoir un marqueur de fin). Ces deux conseils ne sont pas réalistes pour le cas de C-DNS. Par exemple, pour utiliser un tableau de taille définie, il faudrait tout garder en mémoire jusqu'au moment où on inscrit les valeurs, ce qui augmenterait la consommation mémoire du producteur de données C-DNS. (D'un autre côté, le problème des tableaux de taille indéfinie est qu'ils ont un marqueur de fin ; si le programme qui écrit du C-DNS plante et ne met pas le marqueur de fin, le fichier est du CBOR invalide.)

Le RFC a créé plusieurs registres IANA <https://www.iana.org/assignments/c-dns/c-dns.xml> pour ce format, stockant notamment les valeurs possibles pour le transport utilisé, pour les options de stockage (anonymisé, échantillonné...), pour le type de réponse (issue de la mémoire du résolveur ou pas).

Bien sûr, récolter des données de trafic DNS soulève beaucoup de problèmes liés à la vie privée (cf. RFC 7626). Il est donc recommander de minimiser les données, comme imposé par des réglements comme le RGPD, ou comme demandé dans le rapport « "*Recommendations on Anonymization Processes for Source IP Addresses Submitted for Future Analysis*" <https://www.icann.org/en/system/files/files/rssac-040-07aug18-en.pdf> ».

Les passionnés de questions liées aux formats regarderont l'annexe C, qui liste des formats alternatifs à CBOR, qui n'ont finalement pas été retenus :

— Apache Avro <https://avro.apache.org/>, trop complexe car on ne peut lire les données qu'en traitant le schéma,
— Protocol Buffers, qui a le même problème,
— JSON (RFC 8259), qui n'est pas un format binaire, mais qui a été ajouté pour compléter l'étude.

Cette annexe décrit également le résultat de mesures sur la compression obtenue avec divers outils, sur les différents formats. C-DNS n'est pas toujours le meilleur, mais il est certainement, une fois comprimé, plus petit que pcap, et plus simple à mettre en œuvre qu'Avro ou Protocol Buffers.

Notez que j'ai travaillé sur ce format lors d'un hackathon de l'IETF <https://www.bortzmeyer.org/c-dns-tests.html>, mais le format a pas mal changé depuis (entre autres en raison des problèmes identifiés lors du hackathon).

Voyons maintenant une mise en œuvre de ce format, avec l'outil DNS-STATS <http://dns-stats.org/> plus exactement son "*Compactor*" (source sur Github <https://github.com/dns-stats/compactor>, et documentation <https://github.com/dns-stats/compactor/wiki>). Je l'ai installé sur une machine Debian :

```
aptitude install libpcap-dev libboost1.67-all-dev liblzma-dev libtins-dev
git clone https://github.com/dns-stats/compactor.git
cd compactor
sh autogen.sh
autoconf
automake
./configure
make
```

Et après, on peut l'utiliser pour transformer du C-DNS en pcap et réciproquement. J'ai créé un fichier pcap d'un million de paquets avec tcpdump sur un serveur faisant autorité, avec `tcpdump -w dns.pcap -c 1000000 port 53`. Puis :

```
%   ./compactor -o /tmp/dns.cdns  /tmp/dns.pcap
```

Et en sens inverse (reconstituer le pcap) :

```
%  ./inspector /tmp/dns.cdns
```

Cela nous donne :

```
% ls -lth /tmp/dns*
-rw-r--r-- 1 stephane stephane  98M Jul 31 08:13 /tmp/dns.cdns.pcap
-rw-r--r-- 1 stephane stephane 3.2K Jul 31 08:13 /tmp/dns.cdns.pcap.info
-rw-r--r-- 1 stephane stephane  27M Jul 31 07:27 /tmp/dns.cdns
-rw-r--r-- 1 root     root     339M Jul 30 20:05 /tmp/dns.pcap
```

Notez que `dns.cdns.pcap` est le pcap reconstitué, on remarque qu'il est plus petit que le pcap original, certaines informations ont été perdues, comme les adresses MAC. Mais il reste bien plus gros que la même information stockée en C-DNS. Le `/tmp/dns.cdns.pcap.info` nous donne quelques informations :

```
% cat /tmp/dns.cdns.pcap.info
CONFIGURATION:
  Query timeout        : 5 seconds
  Skew timeout         : 10 microseconds
  Snap length          : 65535
  Max block items      : 5000
  File rotation period : 14583
  Promiscuous mode     : Off
  Capture interfaces   :
  Server addresses     :
  VLAN IDs             :
  Filter               :
  Query options        :
  Response options     :
  Accept RR types      :
  Ignore RR types      :

COLLECTOR:
  Collector ID         : dns-stats-compactor 0.12.3
```

```
  Collection host ID   : ns1.example

STATISTICS:
  Total Packets processed                  : 1000000
  Matched DNS query/response pairs (C-DNS) : 484407
  Unmatched DNS queries            (C-DNS) : 98
  Unmatched DNS responses          (C-DNS) : 69
  Malformed DNS packets                    : 68
  Non-DNS packets                          : 0
  Out-of-order DNS query/responses         : 1
  Dropped C-DNS items (overload)           : 0
  Dropped raw PCAP packets (overload)      : 0
  Dropped non-DNS packets (overload)       : 0
```