dnstap, un journal de l'activité d'un serveur DNS

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 8 juin 2017

http://www.bortzmeyer.org/dnstap.html

Comment voir en temps réel l'activité d'un serveur DNS, les requêtes qu'il reçoit et les réponses qu'il envoie ? Il existe plusieurs méthodes, cet article présent rapidement la plus récente, dnstap <http://dnstap.info/>.

Est-ce que je présente d'abord dnstap, ou d'abord les autres solutions, afin d'expliquer leurs faiblesses et de dire pourquoi dnstap a été développé ? Je choisis d'être positif et de commencer par les bonnes choses : dnstap <http://dnstap.info/>, ce qu'il fait, et comment.

dnstap est un moyen d'obtenir un flux d'informations de la part d'un serveur DNS (qu'il s'agisse d'un résolveur ou bien d'un serveur faisant autorité). Le serveur doit être modifié pour journaliser son activité, et configuré ensuite pour émettre un flux dnstap. Typiquement, ce flux est envoyé vers une prise Unix, où un lecteur dnstap l'attendra pour stocker ce flux, ou pour le formater et l'afficher joliment. Ce flux est encodé en "*Protocol Buffers*", eux-mêmes transportés dans des "*Frame Streams*" <https://github.com/farsightsec/fstrm>.

C'est pas assez concret ? OK, alors jouons un peu avec un résolveur DNS, Unbound. On le configure ainsi :

```
dnstap:
   dnstap-enable: yes
   dnstap-socket-path: "/var/run/unbound/dnstap.sock"
   dnstap-send-identity: yes
   dnstap-send-version: yes
   dnstap-log-resolver-response-messages: yes
   dnstap-log-client-query-messages: yes
```

(Les quatre dernières lignes sont optionnelles. Ici, avec les deux dernières lignes, on ne journalise qu'une partie des requêtes et des réponses.) Et on relance le serveur. Il va dire quelque chose comme :

```
[1496927690] unbound[32195:0] notice: attempting to connect to dnstap socket /var/run/unbound/dnstap.sock
[1496927690] unbound[32195:0] notice: dnstap identity field set to "gpd-01"
[1496927690] unbound[32195:0] notice: dnstap version field set to "unbound 1.5.10"
[1496927690] unbound[32195:0] notice: dnstap Message/RESOLVER_RESPONSE enabled
[1496927690] unbound[32195:0] notice: dnstap Message/CLIENT_QUERY enabled
```

Si on l'interroge (par exemple avec dig), le serveur DNS va envoyer les messages dnstap à la prise, ici `/var/run/unbound/dnstap.sock`. Pour les lire, on va utiliser le client en ligne de commande `dnstap` :

```
% dnstap -q -u /var/run/unbound/dnstap.sock
...
15:17:09.433521 CQ ::1 UDP 41b "witches.town." IN A
...
15:18:23.050690 CQ ::1 UDP 52b "fete.lutte-ouvriere.org." IN A
15:18:23.055833 RR 2001:4b98:abcb::1 UDP 147b "fete.lutte-ouvriere.org." IN A
```

Pour la première requête, l'information était dans le cache (la mémoire) du résolveur DNS. Il n'y a donc eu qu'une seule requête, depuis dig vers le résolveur (CQ = "*Client Query*"). Dans le second cas, la réponse ne se trouvait pas dans le cache, notre résolveur a dû aller demander à un serveur faisant autorité pour le domaine (en l'occurrence le serveur `2001:4b98:abcb::1`, dont on voit la réponse au résolveur).

Si le cache est froid (le résolveur vient de démarrer), on verra que le résolveur devra faire plein d'autres requêtes (« requêtes tertiaires », dit le RFC 7626 [1]), ici, on a demandé `www.sinodun.com`, et le résolveur devra trouver les adresses IP des serveurs de `.com` (les serveurs de la racine étaient déjà connus, ici, c'est le serveur `I.root-servers.net` qui a répondu) :

```
11:47:39.556098 CQ ::1 UDP 44b "www.sinodun.com." IN A
11:47:39.560778 RR 192.36.148.17 UDP 1097b "." IN NS
11:47:39.598590 RR 192.112.36.4 UDP 867b "www.sinodun.com." IN A
11:47:39.605212 RR 2001:500:9f::42 UDP 852b "m.gtld-servers.net." IN AAAA
11:47:39.611999 RR 2001:503:a83e::2:30 UDP 789b "m.gtld-servers.net." IN AAAA
11:47:39.611999 RR 199.7.91.13 UDP 852b "l.gtld-servers.net." IN AAAA
11:47:39.611999 RR 192.35.51.30 UDP 789b "j.gtld-servers.net." IN AAAA
11:47:39.616442 RR 192.35.51.30 UDP 771b "av4.nstld.com." IN A
11:47:39.618318 RR 192.12.94.30 UDP 771b "av1.nstld.com." IN A
11:47:39.619118 RR 192.26.92.30 UDP 517b "www.sinodun.com." IN A
11:47:39.620192 RR 192.82.134.30 UDP 286b "av4.nstld.com." IN A
11:47:39.625671 RR 192.82.134.30 UDP 115b "m.gtld-servers.net." IN AAAA
11:47:39.628389 RR 192.54.112.30 UDP 789b "f.gtld-servers.net." IN AAAA
11:47:39.628974 RR 192.54.112.30 UDP 789b "d.gtld-servers.net." IN AAAA
```

Et si je veux voir les réponses, pas seulement les questions ? Demandons à `dnstap` d'être plus bavard :

1. Pour voir le RFC de numéro NNN, `https://www.ietf.org/rfc/rfcNNN.txt`, par exemple `https://www.ietf.org/rfc/rfc7626.txt`

```
% dnstap -y -u /var/run/unbound/dnstap.sock
type: MESSAGE
identity: "gpd-01"
version: "unbound 1.5.10"
message:
  type: CLIENT_QUERY
  query_time: !!timestamp 2017-06-08 13:24:56.664846
  socket_family: INET6
  socket_protocol: UDP
  query_address: ::1
  query_port: 52255
  query_message: |
    ;; opcode: QUERY, status: NOERROR, id: 32133
    ;; flags: rd ad; QUERY: 1, ANSWER: 0, AUTHORITY: 0, ADDITIONAL: 1

    ;; QUESTION SECTION:
    ;www.kancha.de. IN  A

    ;; ADDITIONAL SECTION:

    ;; OPT PSEUDOSECTION:
    ; EDNS: version 0; flags: do; udp: 4096
---
type: MESSAGE
identity: "gpd-01"
version: "unbound 1.5.10"
message:
  type: RESOLVER_RESPONSE
  query_time: !!timestamp 2017-06-08 13:24:56.664838
  response_time: !!timestamp 2017-06-08 13:24:56.697349
  socket_family: INET
  socket_protocol: UDP
  response_address: 85.13.128.3
  response_port: 53
  query_zone: "kancha.de."
  response_message: |
    ;; opcode: QUERY, status: NOERROR, id: 6700
    ;; flags: qr aa; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

    ;; QUESTION SECTION:
    ;www.kancha.de. IN  A

    ;; ANSWER SECTION:
    www.kancha.de. 7200 IN A 85.13.153.248

    ;; ADDITIONAL SECTION:

    ;; OPT PSEUDOSECTION:
    ; EDNS: version 0; flags: do; udp: 1680
---
```

(Eh oui, c'est du YAML.) Notons quelques points importants :

— C'est le client dnstap pas le résolveur DNS, qui formate ce résultat comme il veut. On peut donc écrire d'autres clients dnstap, qui feraient autre chose avec les données (calculer des résultats agrégés, présenter comme tcpdump, etc). C'est une propriété importante de dnstap : le serveur journalise, le client dnstap en fait ce qu'il veut.

— dnstap permet d'ajouter des informations qui n'apparaissent à aucun moment dans la requête ou dans la réponse. C'est le cas de `query_zone`, qui indique le bailliage (la zone d'où vient l'information sur ce nom) ou des temps de réponse.

Bon, maintenant, quelques petits détails techniques sur ce qu'il a fallu faire pour obtenir cela. D'abord, Unbound, sur presque tous les systèmes, a été compilé **sans** dnstap, probablement pour ne pas ajouter une dépendance supplémentaire (vers les "*Protocol Buffers*"). Si vous essayez de configurer Unbound pour dnstap, vous aurez un « "*fatal error : dnstap enabled in config but not built with dnstap support*" ».

Il faut donc le compiler soi-même, avec l'option `--enable-dnstap`. Cela nécessite d'installer "*Protocol Buffers*" et "*Frame Streams*" <https://github.com/farsightsec/fstrm> (paquetages Debian `libprotobuf-c-dev` et `libfstrm-dev`).

Quant au client dnstap, celui utilisé est écrit en Go et nécessite d'installer le paquetage golang-dnstap <https://github.com/dnstap/golang-dnstap> :

```
% go get github.com/dnstap/golang-dnstap
% go install github.com/dnstap/golang-dnstap/dnstap
```

Quelles sont les autres mises en œuvre dont on dispose ? Le serveur DNS faisant autorité Knot a dnstap (mais ce n'est pas le cas du résolveur du même nom). Le serveur BIND peut également être compilé avec dnstap (testé avec la 9.11.1). Comme client, il existe aussi un client fondé sur ldns <https://www.nlnetlabs.nl/projects/ldns/> (paquetage Debian `dnstap-ldns`).

Enfin, pour terminer, voyons les autres solutions qui existent pour afficher ce que fait le serveur DNS. La plus évidente est de demander au serveur de journaliser lui-même. Avec le formatage des données, cela peut imposer une charge sévère au serveur, alors qu'un serveur DNS doit rester rapide, notamment en cas d'attaque par déni de service (avec dnstap, le formatage à faire est minime, vu l'efficacité des "*Protocol Buffers*"). Comme le note la documentation d'Unbound, « "*note that it takes time to print these lines which makes the server (significantly) slower*" ». Mais il y a une autre limite : on ne peut voir que les requêtes, pas les réponses, ni les requêtes tertiaires.

Avec Unbound, cela se configure ainsi :

```
log-queries: yes
```

Et, si on demande le MX de `sonic.gov.so`, on obtient :

```
Jun 08 15:53:01 unbound[2017:0] info: ::1 sonic.gov.so. MX IN
```

Avec BIND, c'est :

```
08-Jun-2017 16:22:44.548 queries: info: client @0x7f71d000c8b0 ::1#39151 (sonic.gov.so): query: sonic.gov.so
```

Une autre solution pour voir le trafic DNS, et celle que je recommandais avant, est de "*sniffer*" le trafic réseau et de l'analyser. Il vaut mieux ne pas faire cela sur le serveur, que cela peut ralentir, mais sur une autre machine, avec du "*port mirroring*" pour envoyer une copie du trafic à la machine d'observation. Cette technique a le gros avantage de n'imposer absolument aucune charge au serveur. Mais elle a quatre défauts :

- Il faut réassembler les réponses fragmentées, phénomène relativement fréquent avec le DNS, où les données peuvent dépasser la MTU et où l'utilisation d'UDP fait que la couche transport ne découpe pas les données.
- Il faut reconstituer le trafic TCP, qui se retrouve réparti en plusieurs paquets. Avec le RFC 7766, l'utilisation de TCP deviendra sans doute de plus en plus fréquente pour le DNS.
- Enfin, si le trafic est chiffré (RFC 7858), le "*sniffer*" est aveugle.
- Certaines choses (comme la bailliage) ne se voient pas du tout dans le trafic, seul le serveur les connait.

Les trois premières raisons sont relativement récentes (autrefois, on ne chiffrait pas, on n'utilisait pas TCP, et les données étaient plus petites) et justifient le retour à un système où le serveur va devoir faire une partie du travail.
