RFC 6168 : Requirements for Management of Name Servers for the DNS

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 4 mai 2011

Date de publication du RFC : Mai 2011

https://www.bortzmeyer.org/6168.html

Traditionnellement, les serveurs DNS étaient gérés par des mécanismes spécifiques au logiciel utilisé. BIND a son `rndc`, Unbound son `unbound-control` (voir des exemples à la fin), d'autres logiciels utilisent d'autres méthodes et on n'avait pas de moyen facile de gérer un ensemble hétérogène de serveurs. En outre, ces programmes n'étaient pas prévus pour des autorisations précises (modifier telle zone mais pas telle autre) et cela rendait difficile de déléguer la gestion d'une **partie** du service à un tiers. L'IETF est donc lancée depuis quelques années dans un effort pour mettre en place un mécanisme standard de gestion à distance des serveurs DNS et ce RFC 6168 [1] en est le cahier des charges formel.

Car c'est un vieux projet ! Une des étapes avait été la publication, il y a plus de quatre ans, du brouillon « "*Requirments for the Nameserver Communication protocol*" <https://www.bortzmeyer.org/ncp-first-draft.html> ». Il y a eu ensuite un groupe de travail restreint, nommé DCOMA, dont j'ai fait partie. Le projet a avancé à une allure de tortue mais, au moins, il a désormais un cahier des charges figé. Avant même le document cité plus haut, un effort de normalisation avait mené à deux MIB, décrites dans les RFC 1611 et RFC 1612, qui n'avaient jamais été réellement déployées et ont été officiellement abandonnées par le RFC 3197.

L'annexe A rassemble quelques études de cas, pour illustrer à quoi pourrait servir un tel protocole. Par exemple, en A.1 se trouve le cas de zones locales non-standard comme un `.prive` ou `.local`. Si des TLD locaux sont une mauvaise idée <https://www.bortzmeyer.org/pourquoi-le-tld-local-n-est-pas-une-html>, ils n'en sont pas moins très utilisés et des domaines purement locaux peuvent avoir un sens dans certains cas. Actuellement, pour qu'un tel système fonctionne, il faut mettre des directives `stub` ou `forward` dans **tous** les résolveurs de l'organisation, et garder ces résolveurs (qui peuvent être de marques différentes) synchronisés. D'une façon générale, gérer un "*pool*" de résolveurs DNS qui doivent

1. Pour voir le RFC de numéro NNN, `https://www.ietf.org/rfc/rfcNNN.txt`, par exemple `https://www.ietf.org/rfc/rfc6168.txt`

avoir la même configuration (par exemple les mêmes clés DNSSEC, cf. annexe A.3) est une tâche courante pour un FAI et plutôt pénible. Le futur protocole de contrôle pourrait faciliter l'automatisation de cette tâche.

A.2 propose un autre cas, dans l'optique de résilience maximale d'une zone DNS : le RFC 2182 recommande à juste titre qu'une zone DNS soit servie par plusieurs serveurs de noms, ne partageant pas de SPOF. Une solution fréquente est l'hébergement réciproque des zones d'un partenaire. Par exemple (au 12 avril 2011), l'AFNIC héberge un serveur secondaire de `.nl` et SIDN ("*Stichting Internet Domeinregistratie Nederland*") a un serveur secondaire de `.fr`. La configuration correspondante est actuellement maintenue à la main, ou par des logiciels ad hoc. Ainsi, si un TLD dont le serveur secondaire est à l'AFNIC, veut tout à coup autoriser le transfert de zones, ou au contraire le restreindre à certaines adresses IP, il doit envoyer un message et attendre qu'un technicien de l'AFNIC ait changé la configuration. De même, si deux hébergeurs DNS assurent réciproquement un service de secours en hébergeant les zones de l'autre, la création d'une nouvelle zone doit actuellement être automatisée par des moyens spécifiques du logiciel utilisé. À noter que ce cas nécessite des autorisations à grain fin, comme rappelé en section 4.4.

Ce cahier des charges concerne à la fois les serveurs faisant autorité (par exemple ceux de `.FR`) que les récurseurs (par exemple les résolveurs de votre FAI). La section 2.2 donne des détails sur les types de serveurs qui seront gérés (les maîtres, les esclaves, et les récurseurs). Les "*stub resolvers*" (la bibliothèque de résolution de noms sur votre machine) sont exclus du champ d'application de ce projet. De même, le futur protocole qui mettre en œuvre ces exigences ne s'occupera que de la configuration des serveurs, pas des données servies (pour lesquelles il existe déjà des protocoles standard de transmission, cf. RFC 2136 et RFC 5936).

Le RFC est écrit sous forme d'une suite d'exigences, chacune dans sa propre section. Donc, le numéro de section identifie l'exigence. Voyons les principales.

L'exigence 2.1.1 concerne la taille : parmi les serveurs faisant autorité, il y en a avec peu de zones mais beaucoup de données dans ces zones (les serveurs des TLD par exemple) et d'autres avec beaucoup de toutes petites zones (un hébergeur DNS qui accueille des centaines de milliers de `mapetiteentreprise.com`). La solution doit convenir pour toutes les tailles.

La solution pourra permettre la découverte automatique des serveurs de noms existant sur le réseau mais ce n'est pas obligatoire (exigence 2.1.2).

Elle devra fonctionner pour des serveurs dont la configuration change souvent (nombreuses zones ajoutées et retirées par heure, par exemple, exigence 2.1.3).

Pour contrôler quelque chose, il faut un modèle décrivant la chose en question. L'exigence 2.1.5 impose donc la création d'un **modèle de données** d'un serveur de noms. Enfin, 2.1.6 impose que la nouvelle solution n'interfère pas le service principal, le DNS lui-même.

La section 3 décrit les classes d'opérations de gestion à implémenter :
— Contrôle (arrêter, redémarrer),
— Configuration (ajouter une zone, ajouter une clé de confiance DNSSEC, etc),
— Surveillance de l'état du serveur (pour intégrer dans des solutions comme Nagios),
— Déclenchement d'alarmes.
Le futur protocole devra gérer toutes ces classes.

Quelles sont les opérations à faire sur un serveur de noms ? La section 3.1 liste les plus importantes :

— Démarrer et arrêter le serveur,
— Recharger la configuration,
— Recharger une zone (ou toutes les zones),

À noter que l'opération « démarrer le serveur » est la seule qui ne puisse pas être mise en œuvre dans le serveur lui-même.

Comme certaines de ces opérations peuvent être très longues (la seule lecture du fichier de zone d'un gros TLD peut prendre du temps), la même section demande en outre que le futur protocole ait un mécanisme de notification asynchrone (« fais ça et préviens-moi quand c'est fini »).

Et question configuration, que doit-on pouvoir configurer ? La section 3.2 donne une liste :

— Les zones à servir (pour un serveur maître) ; à noter qu'il existe déjà des protocoles standard pour transmettre les données contenues dans ces zones (RFC 5936, RFC 1995 et RFC 2136),
— La liste des zones à servir ; on doit pouvoir ajouter et retirer des noms à cette liste (« tu fais maintenant autorité pour `foobar.example` et ton maître est `2001:db8:2011:04:11::53` »), ce qui n'est pas possible de manière standard actuellement,
— Les clés de confiance utilisées dans des protocoles comme DNSSEC,
— Les clés utilisées par TSIG (RFC 8945),
— Les autorisations : qui a le droit de faire des mises à jour dynamiques (RFC 2136), qui a le droit de transférer le contenu de la zone, qui a le droit de faire des requêtes récursives, etc.

Le protocole, on l'a vu, doit également permettre la surveillance du bon fonctionnement du serveur. Des exemples de données utiles sont (section 3.3) :

— État du serveur (« fonctionnement normal », « en cours d'arrêt », etc),
— Statistiques de base, comme le nombre de requêtes traitées, le nombre d'erreurs, etc,
— Liste des zones actuellement servies,
— Alertes de sécurité.

Le serveur doit pouvoir transmettre ces informations à la demande du logiciel de surveillance, mais aussi pouvoir sonner des alarmes sans qu'on lui ait rien demandé (section 3.4) :

— Changement d'état du serveur (« ça y est, je suis prêt » ou au contraire « plus de mémoire, j'arrête tout »),
— Problème de sécurité (tentative de transfert d'une zone refusée, par exemple).

Un tel protocole, riche en fonctions, peut attirer l'attention des méchants : arrêter à distance un serveur de noms, sans avoir besoin de se connecter sur la machine, est tentant. D'où la section 4 qui pose les exigences de sécurité du protocole (voir aussi section 6). La plus importante est l'exigence d'une authentification mutuelle : le serveur de noms doit pouvoir authentifier le logiciel de gestion et réciproquement. Il est également nécessaire que le protocole soit confidentiel : des informations secrètes (comme les clés TSIG) seront parfois transmises par ce canal. Et, une fois l'authentification faite, le protocole doit permettre d'exprimer des autorisations détaillées : tel client peut ajouter des zones mais pas redémarrer le serveur, tel client peut lire toutes les informations mais n'en modifier aucune, etc.

La section 5 est ensuite un pot-pourri d'exigences supplémentaires pour le futur protocole de contrôle des serveurs de noms. On y trouve les grands classiques comme l'extensibilité dans le temps (futures fonctions) et dans l'espace, via des extensions spécifiques à un vendeur (un logiciel donné peut toujours avoir des fonctions très spéciales, trop spéciales pour être normalisées, mais qu'il serait intéressant de contrôler via le protocole).

Pour avoir une idée plus précise des fonctions d'un tel protocole, on peut regarder ce qui existe aujourd'hui. Comme exemple de serveur faisant autorité, essayons BIND. Son outil de contrôle à distance se nomme `rndc`. Il y a plusieurs moyens de le configurer (la plupart des paquetages binaires de BIND font tout cela automatiquement), ici, je vais générer une clé d'authentification et un fichier de configuration à la main. La clé est une clé secrète partagée, on la génère avec `dnssec-keygen` :

```
% dnssec-keygen -a hmac-md5 -n HOST -b 128 samplekey
```

Le fichier produit, ici `Ksamplekey.+157+06915.private`, contient le secret :

```
% cat Ksamplekey.+157+06915.private
...
Key: 7fCXQYhOXW+jkpEx9DuBdQ==
...
```

On va alors créer le fichier de configuration de rndc. Comme il contient un secret (la clé), il faudra s'assurer que ce fichier ait les bonnes protections :

```
options {
      default-server  localhost;
      default-key     samplekey;
};

server localhost {
      key             samplekey;
      addresses   { ::1 port 9153; };
};

key samplekey {
      algorithm       hmac-md5;
      secret          "7fCXQYhOXW+jkpEx9DuBdQ==";
};
```

Et il faut configurer le serveur de noms, dans son `named.conf`. Ici, le protocole de contrôle n'acceptera que `localhost` (`::1`) sur le port 9153 (la valeur par défaut est 953) :

```
key samplekey {
      algorithm       hmac-md5;
      secret          "7fCXQYhOXW+jkpEx9DuBdQ==";
};

controls {
     inet ::1 port 9153
             allow {::1;}
             keys {samplekey;};
};
```

Je me répète, attention à la sécurité. Car, s'il peut lire la clé, un utilisateur pourra arrêter BIND, supprimer des zones, etc.

Voyons maintenant des exemples d'utilisation :

```
% rndc status
version: 9.8.0
number of zones: 2
...
server is up and running
```

Cette commande était bien inoffensive. Mais on peut en faire d'autres comme :
— Recharger une zone (`rndc reload $MAZONE`),
— Arrêter le serveur (`rndc stop`),

— Ajouter une nouvelle zone à servir (`rndc addzone example.com '{ type master; file "example.com"; };'`),
— Et bien d'autres...

À la lumière du cahier des charges, que manque t-il à BIND ? Les grandes lignes y sont mais il manque pas mal de détails comme les notifications asynchrones, les changements d'ACL et peut-être la finesse des autorisations. `rndc addzone` nécessite une option spéciale dans `named.conf` mais elle s'applique à toutes les zones et, de toute façon, on ne peut pas la lier à certaines clés seulement. Une fois qu'on a un accès au serveur, c'est un accès total.

Comme exemple de serveur récursif, prenons Unbound. La configuration se fait dans le `unbound.conf` à peu près comme ceci (même adresse et port qu'avec BIND plus haut) :

```
remote-control:
control-enable: yes
control-interface: ::1
control-port: 9153
server-key-file: "unbound_server.key"
server-cert-file: "unbound_server.pem"
control-key-file: "unbound_control.key"
control-cert-file: "unbound_control.pem"
```

Les certificats X.509 utilisés pour l'authentification peuvent facilement être créés avec `unbound-control-setup`. Le programme de contrôle du serveur utilise le même fichier de configuration `unbound.conf` (contrairement à BIND) :

```
% unbound-control status
version: 1.4.5
verbosity: 4
threads: 1
modules: 2 [ validator iterator ]
uptime: 206 seconds
unbound (pid 5315) is running...
```

et on peut faire plein de choses avec ce programme comme arrêter le serveur (`unbound-control stop`), afficher d'utiles statistiques (nombre de requêtes, nombre de succès et d'échecs de cache, etc, avec `unbound-control stats`), activer ou désactiver le relayage vers un autre serveur (`unbound-control forward 2001:db8:42::53`), vider le cache pour une zone donnée (`unbound-control flush_zone example.com`, notez qu'`unbound_control` a d'autres commandes de vidage, plus ou moins subtiles), etc.

Par rapport aux exigences de notre RFC 6168, on note que `unbound-control` ne permet pas de changer les clés de confiance utilisées par le serveur. Il ne peut pas non plus afficher les zones définies localement sur le récurseur (mais il permet d'en ajouter). Et la finesse des autorisations est insuffisante (la possibilité de créer des zones locales est ouverte à tous ceux qui ont un accès.) Autrement, il a une bonne partie de ce qu'il faut pour un récurseur.

Maintenant que le cahier des charges est fait, quelles sont les solutions pour le mettre en œuvre ? Historiquement, le protocole officiel de gestion des machines, à l'IETF, était SNMP. Comme on l'a vu, il y a même eu une tentative de s'en servir avec les serveurs DNS, tentative qui n'est pas allé loin. Aujourd'hui, la mode est plutôt à un autre protocole de gestion, Netconf (RFC 6241). Il existe une proposition, documentée dans l'"*Internet-Draft*" `draft-dickinson-dnsop-nameserver-control`, qui utilisera dans le futur Netconf (une version précédente de cette proposition incluait le protocole, et même une implémentation, mais la version actuelle se concentre sur le modèle de données). Voir la

présentation au CENTR à Amsterdam en mai 2011 <https://www.centr.org/main/6282-CTR/version/default/part/AttachmentData/data/NSCP_CENTR.pdf>.

Aux diverses réunions IETF sur le sujet, une alternative ne nécessitant pas un nouveau protocole avait souvent été proposée : se servir d'outils de gestion de configuration comme Chef ou Puppet. La principale limite de ces outils est qu'ils ne conviennent qu'à l'intérieur d'une même organisation et ne règlent pas facilement des problèmes multi-organisations.