Quelques éléments d'histoire sur le DNS

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 4 juillet 2010. Dernière mise à jour le 1 novembre 2012

http://www.bortzmeyer.org/bind-dns-history.html

Le DNS est à la base de la plupart des transactions sur Internet. Presque toutes commencent en effet par une requête DNS. Par exemple si vous récupérez un fichier via BitTorrent (dont on pourrait penser qu'il n'utilise pas le DNS), le "*tracker*" est souvent trouvé via une URL donc via le DNS. Même si vous utilisez un système récent, "*tracker-less*" (celui-ci étant remplacé par une DHT), vous avez probablement trouvé les coordonnées du fichier qui vous intéressait <http://www.thepiratebay.org/> via le Web donc via le DNS. Bref, si le DNS est en panne, il n'y a presque plus rien qui marche. Pourtant, à une époque, l'Internet fonctionnait sans le DNS. Comment s'est faite la transition ? Qu'est-ce qu'il y avait avant ?

Vue l'énorme quantité d'articles soi-disant historiques de vulgarisation sur les origines du DNS, vue la quantité de bêtises journalistiques qui ont été écrites sur ce sujet, il peut sembler difficile de remonter aux vraies origines et de trouver ce qui s'est passé. Pourtant, la plupart des acteurs de ce changement sont encore vivants et beaucoup ont documenté leurs efforts.

Commençons par les pères du DNS eux-même. Paul Mockapetris et Kevin Dunlap ont documenté les débuts de ce protocole en 1988, dans « "*Development of the Domain Name System*" <http://cseweb.ucsd.edu/classes/wi01/cse222/papers/mockapetris-dns-sigcomm88.pdf> » (le fichier PDF en question est une numérisation de l'original papier, il ne semble pas que la forme numérique originale aie été conservée ; même chose pour les autres articles de la série).

Tout commença, racontent les auteurs, en 1982, lorsque l'ancien système de nommage, HOSTS.TXT (qui était spécifié dans le RFC 608 [1]), montrait trop clairement ses limites, avec l'augmentation de taille de l'Internet. Plusieurs candidats avaient été envisagés pour son remplacement comme IEN 116 <http://www.postel.org/ien/pdf/ien116.pdf> ou comme le système Grapevine de Xerox. Le premier était jugé trop simple, le second beaucoup trop compliqué (il essayait de tout faire, contrairement à la

1. Pour voir le RFC de numéro NNN, https://www.ietf.org/rfc/rfcNNN.txt, par exemple https://www.ietf.org/rfc/rfc608.txt

méthode qui a assuré le succès de l'Internet, faire des systèmes simples qui assurent 95 % des fonctions, sans chercher à tout résoudre). Une curiosité : dans Grapevine, les noms étaient hiérarchiques, comme dans le DNS, mais le nombre de niveaux était fixe (alors qu'un nom de domaine peut avoir un seul composant - par exemple `dk`, deux, trois, ou plus - par exemple `ns1.dnsar.ati.tn`). Grapevine n'avait que deux niveaux, le registre et la machine. Le travail commença donc sur le protocole DNS. (Cela n'apparait pas dans l'article mais, oralement, Paul Mockapetris raconte qu'il avait hérité de cette tâche car il était le plus débutant et que ce service DNS n'était pas considéré comme essentiel.) Et c'est là qu'on dispose d'une autre source, les fameux RFC puisque tous ont été conservés (parfois sous forme papier, nécessitant une renumérisation). Le DNS est donc né officiellement avec les RFC 882 et RFC 883 (la norme actuelle figure dans les RFC 1034 et RFC 1035).

Il semble que ce soit vers 1985 que certaines machines commencèrent à n'utiliser **que** le DNS pour accéder à l'information. (Je me souviens bien qu'au début des années 1990, il était fréquent que des documents comme les FAQ indiquent les adresses IP des serveurs cités, car certains « vieux de la vieille » n'avaient pas confiance dans le DNS qui, il est vrai, n'avait pas toujours la fiabilité d'aujourd'hui.)

L'article de Mockapetris et Dunlap détaille ensuite les principes fondateurs du DNS : nommage hiérarchique, compromis entre la simplicité et les demandes de certains enthousiastes qui voulaient en faire une vraie base de données distribuée, absence de sémantique dans les noms (un serveur SMTP n'est pas forcé de s'appeler `smtp.example.net` et on ne peut pas savoir, juste en regardant un nom, où sont les frontières des zones).

Parmi les choix qui se présentaient aux auteurs du nouveau protocole, la détermination du niveau de souplesse nécessaire. Il était prévu que d'autres acteurs que le monde Internet utilisent le nouveau protocole, ce qui avait mené à l'introduction des **classes**. Il y avait même l'idée qu'il y aurait une classe ISO alors que, finalement, le projet OSI avait développé sa propre usine à gaz, X.500, qui a sombré avec le reste de ce projet. Aujourd'hui, les classes continuent à utiliser deux octets dans chaque enregistrement DNS, sans servir à rien (et il est probable que la plupart des logiciels seraient très déroutés par d'autres classes que la classe `IN`, contrairement à ce que pensaient les promoteurs de Net4D <`http://www.bortzmeyer.org/net4d.html`>).

La fin de l'article décrit le déploiement du DNS en 1988. Le RFC 1031 contenait une liste des mises en œuvre du DNS (les deux plus importantes étant BIND, toujours là, et Jeeves, bien oublié). À ce moment, `HOSTS.TXT` contenait encore 5 500 noms alors que le DNS en avait... 20 000 (depuis, le comptage est devenu impossible mais il y a certainement aujourd'hui des centaines de millions de noms).

Une section entière est dédiée aux serveurs de noms de la racine, ceux qui doivent répondre pour tous les noms, en général en renvoyant vers les serveurs du TLD. Ils n'étaient que sept à l'époque (aujourd'hui, il est difficile de donner le nombre exact <`http://www.bortzmeyer.org/combien-serveurs-racin`
`html`>). Quatre étaient des Unix avec BIND, trois des TOPS-20 avec Jeeves. Le trafic sur un serveur racine atteignait la valeur pharamineuse de... une requête par seconde (aujourd'hui, chaque machine physique voit passer plusieurs milliers, voire dizaines de milliers de requêtes par seconde).

Une fois le protocole défini, il restait une importante décision à prendre, sur le schéma de nommage. Le RFC 920 a été le premier à se pencher sur la question, sur laquelle on peut aussi consulter le passionnant article d'Elizabeth Feinler <`http://www.bortzmeyer.org/history-naming.html`>.

Cela, c'était du protocole et de l'organisation. Et le code ? Il existe d'autres articles sur les premières mises en œuvre du DNS :

`http://www.bortzmeyer.org/bind-dns-history.html`

— « *"The Berkeley Internet Name Domain Server"* <http://www.eecs.berkeley.edu/Pubs/TechRpts/1984/CSD-84-182.pdf> »,
— « *"A Name Server Database"* <http://www.eecs.berkeley.edu/Pubs/TechRpts/1984/CSD-84-174.pdf> »,
— « *"The Design and Implementation of a "Domain Names" Resolver"* <http://www.eecs.berkeley.edu/Pubs/TechRpts/1984/CSD-84-176.pdf> »,
— « *"The Design and Implementation of the BIND Servers"* <http://www.eecs.berkeley.edu/Pubs/TechRpts/1984/CSD-84-177.pdf> »,
— « *"Distributed Name Servers : Naming and Caching in Large Distributed Computing Environments"* <http://www.eecs.berkeley.edu/Pubs/TechRpts/1985/CSD-85-228.pdf> ».

"The Berkeley Internet Name Domain Server" est l'article général sur l'ajout du DNS au système d'exploitation BSD 4.2 (le premier qui avait TCP/IP). Des points qui ne seront normalisés que longtemps plus tard (comme la mise à jour dynamique) y sont déjà discutés. L'article discute aussi du mécanisme de stockage, envisageant une base de données généraliste (suggestion rejetée puisqu'un serveur de noms n'a aucun besoin de toutes les fonctions d'une telle base) et choisissant finalement de mettre toutes les données en mémoire (les nouveaux - à l'époque - VAX avaient bien plus de mémoire que leurs prédécesseurs). Ensuite, pour accéder rapidement aux données, une table de hachage indexée par les FQDN était utilisée.

L'article discute aussi du résolveur (qu'on nomme aujourd'hui le *"stub resolver"*), les bibliothèques installées sur le client et qui permettent aux applications de faire des requêtes DNS. Mais l'idée d'un démon tournant en permanence, avec un cache des réponses, était déjà envisagée (ce qu'on nomme un résolveur aujourd'hui).

Les plans étaient ambitieux, jusqu'à envisager de stocker les listes de diffusion dans le DNS. Cela n'a jamais été fait mais un autre projet décrit dans l'article, stocker les noms d'utilisateurs dans le DNS a été réalisé avec Hesiod (bien abandonné aujourd'hui).

Dans *"A Name Server Database"*, l'auteur, David Riggle, se penche sur la partie « Stockage et recupération des données » d'un serveur de noms. L'article est très détaillé techniquement et discute même les plus petits détails. Au moment de l'article, en 1984, les TLD que nous connaissons n'existaient pas encore. Le premier schéma montre donc un arbre où l'Université de Berkeley est sous `.ARPA`. De même, la classe donnée comme exemple n'est pas `IN` (Internet) mais `CS` (CSnet). Et l'article mentionne comme « utile » l'option de recherche inverse du DNS, `IQUERY` (RFC 1035, section 6.4.2), bien oubliée aujourd'hui (car inutilisable en pratique : il faut connaitre le serveur faisant autorité dès le début, elle ne permet pas de le trouver en partant de la racine ; cf. RFC 3425) mais qui a été consciencieusement implémentée. Enfin, les chiffres n'étaient pas du tout les mêmes qu'aujourd'hui. L'auteur cite ainsi un bon résultat : faire tenir 300 noms de domaine dans 64k de mémoire... Le programme avait été écrit en C sur un Vax 11/780 Unix BSD 4.2. Pas question évidemment de réserver une telle machine pour un seul projet et les essais devaient coexister avec plein d'autres travaux, la machine ayant en permanence une charge entre 5 et 10...

Comme dans les autres textes sur le DNS de l'époque, des tas d'utilisations sont envisagées comme de stocker la liste des abonnés d'une liste de diffusion ou bien d'associer employé et numéro de bureau. On voit donc que rien n'est plus faux que la théorie selon laquelle le DNS n'aurait été conçu que pour trouver une adresse IP à partir d'un nom.

La première idée de l'implémenteur est de faire une base de données sur disque qui soit maintenue à la main, en éditant un fichier. Le serveur de noms, lui, ne la modifie jamais. En cas de crash, il n'y a donc pas de problème, il peut toujours repartir du fichier.

La seconde idée importante est de ne pas stocker les données sous la forme « évidente » d'un arbre mais sous la forme d'une table de hachage, indexée par le FQDN. La plupart des noms cherchés sont

en effet des feuilles de l'arbre du DNS et la recherche des données nécessiterait donc un parcours de l'arbre dans toute sa profondeur. (La possibilité d'utiliser une « vraie » base de données comme Ingres est considérée dans l'article et écartée pour des raisons de performance.)

Dans "*The design and implementation of the Berkeley Internet Name Domain servers*", Songnian Zhou met l'accent est mis sur le logiciel qui va utiliser cette base pour répondre aux questions posées. Mais, comme les autres articles de la série, il parle aussi de questions plus fondamentales, voire philosophiques. De nos jours, les articles seraient bien spécialisés, avec des rôles bien déterminés. Mais, à l'époque, on pouvait commencer un article sur l'implémentation du serveur en parlant des règles de nommage et en évoquant celles d'UUCP (plus exactement, celles d'UUCP à l'époque, fondées sur des noms relatifs, indiquant le chemin à parcourir).

Et le serveur lui-même, dans tout cela ? Le DNS de l'époque ne faisait pas de distinction claire entre un serveur faisant autorité et un cache récursif et le logiciel allait donc faire les deux, ce que BIND va faire jusqu'à la version 9 incluse (dans la version 10, les deux fonctions sont complètement séparées, et on peut même compiler le logiciel avec une seule des deux). Par contre, une fonction non prévue par les RFC allait être ajoutée : la mise à jour dynamique des données, qui ne sera normalisée que dans le RFC 2136, sous une forme très proche de ce que décrit l'article (qui explique, par exemple, pourquoi seul le serveur maître peut traiter ces mises à jour).

La programmation de ce nouveau logiciel n'avait pas été évidente. Par exemple, le serveur de noms est naturellement parallèle puisqu'il doit répondre à des requêtes simultanées de plusieurs clients. Le mécanisme normal de parallélisme sur les Unix BSD de l'époque était le processus mais BSD n'avait pas de mémoire partagée, ce qui rendait difficile (du moins c'est ainsi qu'on le voyait à l'époque) la communication entre les processus. Finalement, BIND aura un seul processus et utilisera `select()` pour traiter les questions. Dans un autre ordre d'idées, c'est dans cet article qu'apparait la spécification du format de fichier de configuration que BIND gardera jusqu'à la version 4 incluse. Quant à la mise en œuvre de la mise à jour dynamique, elle était fort complexe et fait l'objet de plusieurs pages.

La partie la plus amusante de la lecture d'un article aussi ancien est évidemment celle consacrée aux projets futurs. Ainsi, l'auteur perçoit bien l'importance d'ajouter des ACL et des mécanismes d'authentification (complètement absents de la première version). Moins bien analysée est l'idée d'utiliser le DNS pour stocker les informations sur l'utilisateur, auxquelles on accédait à l'époque avec la commande finger (qui avait été normalisée dans le RFC 742 et qui est aujourd'hui dans le RFC 1288). De nombreuses années avant ENUM ou `.tel`, la vision de l'enregistrement DNS "*User Information*", UINFO (qui ne sera finalement pas adopté) est stimulante.

Quant à "*The Design and Implementation of a "Domain Names" Resolver*" de Mark Painter, il se focalise sur la partie client, le résolveur (rappelez-vous qu'à l'époque, le vocabulaire n'était pas bien fixé et on disait souvent "*resolver*" pour ce qu'on appelerait aujourd'hui "*stub resolver*"). Là encore, il y avait de grandes idées qui ne se sont pas concrétisées (comme d'utiliser le DNS pour nommer les processus du système, avec évidemment mise à jour dynamique puisque des processus sont créés et détruits tout le temps).

Mais le gros débat dans cet article concernait l'architecture du résolveur. L'auteur envisageait trois possibilités :

— Que le code du résolveur soit entièrement dans l'espace mémoire de chaque processus qui résoud des noms (c'est la méthode adoptée à l'époque),
— Que le code du résolveur soit dans le noyau,
— Que le résolveur soit un processus séparé, avec un petit peu de code dans chaque processus, pour appeler le résolveur (c'est largement la solution qui a été adoptée ensuite sur Unix).

`http://www.bortzmeyer.org/bind-dns-history.html`

L'avantage de la première méthode était qu'un appel au résolveur est peu coûteux (juste un appel de sous-programme). Mais cela interdisait le partage des données (le cache DNS) entre les processus d'une même machine (Unix BSD à cette époque n'avait pas encore de mémoire partagée). La deuxième méthode (dans le noyau) a été écartée vue la complexité du noyau BSD.

Les programmes qui voulaient utiliser le service de résolution de noms avaient à leur disposition une nouvelle API tournant autour d'une routine nommée `std_query` (qui n'a pas été conservée par la suite). Notons que la spécification de cette API limitait explicitement les noms au jeu de caractères ASCII (alors que le DNS, lui, a toujours accepté des jeux plus larges). C'est, je crois, la première mention de cette limitation.

Il y avait aussi une routine `inv_query` pour les requêtes inverses, qui ont finalement été abandonnée dans le RFC 3425 (l'article expliquait déjà pourquoi c'était une mauvaise idée).

Une autre routine, `set_resopt`, permettait de changer le comportement du résolveur, par exemple pour exiger uniquement des réponses faisant autorité, ou au contraire pour n'interroger que le cache local.

Ces nouvelles routines ne concernaient que le résolveur DNS. Unix avait déjà `gethostbyname` comme accès général à la résolution de noms mais sa sémantique légèrement différente avait posé bien des problèmes pour y inclure le DNS.

L'auteur a aussi fait des essais quantitatifs, mesurant que, pour cinquante requêtes, il fallait une demi-seconde, des chiffres bien éloignés des résultats d'aujourd'hui (la seule page d'accueil de `cnn.com` nécessite une centaine de requêtes DNS). À l'époque, on avait nettement moins de ressources et l'aricle note avec inquiétude la taille « substantielle » du résolveur : près de cent kilo-octets...