RFC 6763 : DNS-Based Service Discovery

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 20 février 2013

Date de publication du RFC : Février 2013

https://www.bortzmeyer.org/6763.html

Ce RFC normalise une technique pour trouver des **ressources** sur un réseau. Soit une machine qui arrive sur ce réseau. Elle cherche une imprimante, un serveur de fichiers, comment fait-elle ? Il ne s'agit pas juste de trouver une information technique unique (comme le routeur par défaut, qui est indiqué par DHCP ou RA, les "*Router Advertisement*" du RFC 4862[1]) mais de récolter une liste de mises en œuvre possible de ce service, liste parmi laquelle on choisira ensuite. SD ("*Service Discovery*") fonctionne en interrogeant le DNS, avec le nom du service, la réponse indiquant les machines qui fournissent ce service.

Ce protocole est fondé sur le DNS et utilise du DNS standard : même format, mêmes messages, etc. (Rappelons que le DNS, contrairement à ce que racontent les articles des médias et les textes « DNS pour les nuls », ne sert pas qu'à trouver les adresses IP, c'est au contraire un protocole de base de données très général.) SD est donc uniquement un ensemble de **conventions** (notamment de nommage) au dessus du DNS. SD est souvent utilisé au dessus du "*Multicast DNS*" du RFC 6762, mais peut aussi être utilisé sans lui. Avec "*Multicast DNS*", SD fournit une solution « zéro configuration ».

L'idée de base est que les services sont décrits par des enregistrements SRV (RFC 2782) dont le nom est le nom d'une instance du service et dont le contenu indique le nom du serveur rendant ce service, ainsi que le port à utiliser. Un enregistrement TXT permet, si nécessaire, d'indiquer d'autres informations, qui ne rentrent pas dans un enregistrement SRV. Et pour trouver toutes les instances d'un service ? On se sert d'un enregistrement PTR dont le nom est celui du service.

Par exemple, si une machine "*Multicast DNS*" cherche une imprimante (service IPP, cf. RFC 2910), elle va faire une requête `PTR _ipp._tcp.local` (où `.local` est le pseudo-TLD de "*Multicast DNS*"). Mettons qu'elle obtienne deux réponses, `Imprimante comptabilité.local` et `myprinter.local`. Si

1. Pour voir le RFC de numéro NNN, `https://www.ietf.org/rfc/rfcNNN.txt`, par exemple `https://www.ietf.org/rfc/rfc4862.txt`

l'utilisateur, via une interface de sélection, choisit l'imprimante de la comptabilité, l'étape suivante sera de faire une requête `SRV Imprimante comptabilité.local` (rappelez-vous que le DNS permet caractères Unicode et espace, contrairement à ce qu'on lit parfois, et c'est courant dans le monde "*Multicast DNS*"). Mettons que cette requête donne comme résultat `a56grthg43.local, port 9818`, alors la machine qui veut imprimer se connectera à l'adresse de `a56grthg43.local`, port 9818. Voilà, en quelques mots, le fonctionnement de "*Multicast DNS*". La section 13, décrite plus loin, fournit des exemples réels.

Pour mieux comprendre SD, il est utile de lire la section 3, qui expose le cahier des charges de ce protocole :

— Permettre l'énumération de toutes les instances qui rendent un certain service dans un certain domaine,
— Pour une instance donnée, pouvoir résoudre ce nom en des informations techniques permettant de contacter l'instance (typiquement, une adresse IP et un port ; l'adresse IP peut être IPv4 ou IPv6, SD gère les deux, voir section 14),
— Identifier les instances par un nom relativement stable (≪ Imprimante comptabilité ≫) de manière à ce que l'utilisateur n'ait rien à faire si ladite instance change d'adresse,
— Être suffisamment simple pour être mis en œuvre dans des engins ayant des ressources limitées (comme, justement, des imprimantes). Plus facile à dire qu'à faire, le RFC fait quand même 54 pages.

SD se veut l'héritier du protocole NBP d'Apple (celui popularisé par le "*Chooser*"/Sélecteur des anciens Macintosh) et une plus longue discussion de cet héritage figure dans le RFC 6760.

La section 4 explique comment fonctionne l'énumération des instances d'un service. On l'a vu, le client SD doit faire une requête de type PTR pour un nom qui est la concaténation du service et du domaine. Si le client cherchait à accéder au service `foobar`, il fera des requêtes PTR pour `_foobar._tcp.local` en "*Multicast DNS*" et `_foobar._tcp.example.net` autrement (si `example.net` est le domaine du client). La requêtes renverra des noms d'instance. Ces noms d'instance sont en Unicode (dans le profil du RFC 5198, qui exclut les caractères de contrôle mais pas grand'chose d'autres ; voir l'annexe C pour une discussion de ce choix et l'annexe E pour ses conséquences). Ainsi, un nom d'instance peut comprendre des points (voir la section 4.3 pour les précautions à prendre). Notez que, vue la façon dont fonctionne l'encodage UTF-8, les limites de taille du DNS peuvent réduire le nom à quinze caractères, dans le pire des cas (caractères Unicode en dehors du Plan Multilingue de Base). Notez que le DNS a deux limites, une de 63 octets sur la longueur d'un composant du FQDN et une de 255 octets sur le FQDN (section 7.2 pour les détails). Il y a aussi une taille imposée par la raison : si le but est de présenter une liste d'instances d'un service à un utilisateur, une liste de, mettons, cent entrées, n'a guère de sens.

Le nom de l'instance doit pouvoir être choisi par l'utilisateur, et ne devrait pas être un nom unique généré à l'usine, nom qui serait probablement incompréhensible (genre dérivé de l'adresse MAC ou d'un autre identificateur technique, par exemple `Mac064389AB5`). L'annexe D explique pourquoi ces noms d'usine sont une mauvaise idée. Notamment, le fonctionnement de "*Multicast DNS*" et SD fait que des noms uniques ne sont pas nécessaires.

Quand au nom de domaine à la fin (`.local` avec "*Multicast DNS*" ou un nom de domaine habituel autrement), il n'a pas de règles particulières. Notez encore que le FQDN n'est pas limité aux lettres-chiffres-tiret du RFC 952. En effet, contrairement à ce que dit une légende tenace `<https://www.bortzmeyer.org/pourquoi-idn-et-pas-un-dns-unicode.html>`, le DNS n'a jamais été limité à ASCII. Des noms comme `[Caractère Unicode non montré [2] ][Caractère Unicode non montré ][Caractère Unicode non montré ][Caractère Unicode non montré ][Caractèr`

2. Car trop difficile à faire afficher par LaTeX

Unicode non montré][Caractère Unicode non montré][Caractère Unicode non montré][Caractère Unicode non montré][Caractère Unicode non montré].local ou Building A, 1st sont donc parfaitement valables. Comme indiqué plus haut, s'ils comportent des caractères non-ASCII, ils sont stockés dans le DNS en UTF-8 (RFC 5198, qui normalise le profil Unicode de l'Internet, incluant la canonicalisation choisie). Un client SD qui n'obtient pas de réponse pour une requête en UTF-8 a toutefois le droit de reessayer en Punycode (RFC 3492) mais ce n'est pas obligatoire.

Les RFC normalisent des protocoles tels qu'ils s'expriment sur le câble, pas des interfaces utilisateurs. Néanmoins, la section 4.2 et l'annexe F donnent quelques conseils sur la présentation des résultats à l'utilisateur. Typiquement, le logiciel fait sa requête PTR, récolte une liste d'instances, et montre la liste à l'utilisateur pour qu'il choisisse. A priori, seul le premier composant est montré, pas le FQDN (donc, [Caractère Unicode non montré][Caractère Unicode non montré][Caractère Unicode non montré][Caractère Unicode non montré][Caractère Unicode non montré][Caractère Unicode non montré][Caractère Unicode non montré][Caractère Unicode non montré][Caractère Unicode non montré] et pas [Caractère Unicode non montré][Caractère Unicode non montré][Caractère Unicode non montré][Caractère Unicode non montré][Caractère Unicode non montré][Caractère Unicode non montré][Caractère Unicode non montré][Caractère Unicode non montré].local ou bien Building A, 1st floor printer et non pas Building A, 1st floor printer.example.com).

Après l'énumération des instances, la section 5 explique la résolution du nom d'une instance. Dans la section précédente, on a obtenu le nom Building A, 1st floor printer.example.com et on va chercher à contacter la machine. C'est là que le client SD va faire une requête SRV (pour obtenir nom de machine et port) et une requête TXT (pour obtenir des détails qui n'étaient pas prévus dans l'enregistrement SRV). Les enregistrements SRV (RFC 2782) sont très précieux car ils évitent des numéros de port fixes (il n'y en a que 65 535 et ce n'est pas possible d'en réserver un à l'usage exclusif de chaque service, même si ça se faisait au début de l'Internet). N'importe quel service peut tourner sur n'importe quel port, le client trouvera le port dans l'enregistrement SRV. Autre avantage des SRV : les champs Priorité et Poids qui permettent d'assurer la répartition de charge et la résistance aux pannes. Notez que, si vous êtes programmeur et que cela vous fatigue de gérer les règles de priorité et de poids, il existe des bibliothèques qui font cela pour vous, comme RULI <http://www.nongnu.org/ruli/>.

La section 6 décrit les enregistrements TXT. Les SRV sont très utiles, notamment parce qu'ils sont normalisés depuis longtemps et déjà mis en œuvre dans certains logiciels. Mais ils ne permettent pas d'enregistrer toutes les nuances qu'on voudrait spécifier. D'où l'usage des enregistrements TXT, du texte libre où on peut encoder ce qu'on veut. Par exemple, si un client SD cherche une imprimante, savoir le nom de la machine et le port ne suffit pas. Le vieux protocole LPR (RFC 1179 impose également un nom de file d'attente d'impression. De même, un serveur de fichiers peut avoir plusieurs volumes à servir. On va alors les stocker dans un TXT, avec des règles exactes qui dépendent de l'application (impression, service de fichiers, etc).

Cet enregistrement TXT est obligatoire, même si on n'a rien à dire. Cela permet notamment de choisir le TTL de cette information (autrement, c'est celui de la zone qui s'appliquerait). Le RFC recommande de le garder d'une taille raisonnable (quelques centaines d'octets), même si la théorie permet davantage. Mais, par exemple, il existe des mises en œuvre de "*Multicast DNS*" dans le matériel, qui permettent à une machine d'être découvrable sur le réseau même lorsqu'elle est en hibernation et ces mises en œuvre ont souvent des limites ridicules (par exemple 256 octets maximum pour un TXT).

Et le contenu de ce TXT, alors ? Ce sont des couples clé=valeur, contenant des informations qui dépendent du service (par exemple, pour le service d'impression, on peut consulter "*Bonjour Printing Specification*" <http://developer.apple.com/networking/bonjour/bonjourprinting.

pdf>). La valeur est facultative. Donc, on peut avoir un couple `PlugIns=JPEG,MPEG2,MPEG4` (indiquant quels formats sont acceptés) ou `PaperSize=A4`, mais aussi uniquement `passreq` (juste une clé, sans valeur, ce qui est utile pour les attributs booléens, cet enregistrement est équivalent à `passreq=true`). La valeur peut être également de taille nulle (`PlugIns=` pour indiquer qu'aucun format supplémentaire n'est disponible).

Idéalement, ces informations devraient être facultatives, de simples optimisations du service, et la négociation entre le client et le serveur devrait suffire. En pratique, certains protocoles, comme LPR cité plus haut, imposent des informations qu'on ne trouve que dans le TXT (par exemple, le protocole LPR ne fournit aucun moyen pour une imprimante de dire à son client qu'elle accepte PostScript ou pas). En outre, lorsqu'une information est distribuée via ce TXT, il est important de veiller à ce qu'elle soit cohérente avec ce qui est réellement disponible sur la machine qui répond aux requêtes SD.

Dans les couples clé=valeur, la clé est courte (il n'est pas prévu de la montrer aux utilisateurs, elle n'a pas à être « conviviale »), en ASCII uniquement, et unique. Elle est insensible à la casse. La valeur peut être n'importe quoi, y compris du binaire (attention donc si vous l'affichez). Mais c'est souvent du texte, ASCII ou UTF-8.

Si un service risque d'évoluer dans le futur, et de manière incompatible, le RFC recommande de mettre un attribut indiquant la version, de clé `txtvers`, par exemple `txtvers=1`. Cela aidera le client à savoir à quel genre de serveur il parle.

Maintenant, discutons du noms des services. La section 7 dit qu'ils sont faits de deux composants, le nom proprement dit (suivant le registre <https://www.iana.org/assignments/service-names-port-numb > décrit dans le RFC 6335, cf. section 16), précédé d'un tiret bas, puis `_tcp` ou `_udp`. Et les autres protocoles, comme SCTP ? Ils n'ont malheureusement pas de domaine pour eux, tout ce qui n'est pas TCP doit être mis sous `_udp`. Le point est discuté dans la section 7 mais l'argumentation est invraisemblable, je ne sais même pas comment l'IESG a laissé passer cela, bien que c'était explicitement mentionné dans les revues du futur RFC.

Un point plus sérieux est celui du choix du nom d'un protocole. Il est courant de bâtir un protocole applicatif au dessus d'un autre, souvent au dessus de HTTP, à la fois pour bénéficier des nombreuses bibliothèques existantes, et pour être raisonnablement sûr de passer les pare-feux. C'est ainsi que le protocole de partage de musique DAAP, utilisé par iTunes, est construit au dessus de HTTP. Le service, dans les enregistrements SF, doit-il être `_http._tcp.DOMAIN` ou bien `_daap._tcp.DOMAIN` ? La réponse est claire, c'est le second choix qui est le bon : DAAP n'a de sens qu'avec un client iTunes (un navigateur Web normal, par exemple, serait bien embêté est se connectant à un serveur DAAP, et un client iTunes ne saurait pas quoi faire d'un serveur HTTP générique), et le fait que ce soit du HTTP sous-jacent n'est qu'un détail technique, qui ne se reflète pas dans le nom du service. SD publie le service, pas la façon dont il est mis en œuvre.

À partir d'ici, le RFC continue mais il s'agit plutôt de points de détail, qui n'affectent pas la compréhension globale de SD.

Dans certains cas, il peut être utile de restreindre la demande à un sous-ensemble des machines implémentant un service. Prenons l'exemple d'une imprimante, dotée d'un serveur HTTP pour en permettre l'administration. Si un navigateur Web qui cherche les imprimantes à administrer demande `_http._tcp.DOMAIN`, il va trouver des tas de serveurs n'ayant rien à voir avec les imprimantes. SD permet donc d'ajouter un champ, le sous-type, pour faire une requête `_printer._sub._http._tcp.DOMAIN` pour « sous-service Imprimante du service HTTP ». (Notez le `_sub`, dont le RFC n'explique pas le rôle.) Notez que c'est un problème différent de celui du paragraphe précédent : ici, le

service d'administration des imprimantes n'est pas juste « bâti sur HTTP », il est en HTTP et un navigateur HTTP générique peut l'utiliser (la section 7.1 détaille dans quels cas les sous-types sont utiles et dans quels cas ils ne le sont pas). Le navigateur Safari sur MacOS fonctionne ainsi. En parlant de MacOS, la commande `dns-sd` permet de publier sur le réseau

local un nom de service. Par exemple, ici, on publie le service HTTP "*A web page*" :

```
% dns-sd -R "A web page"           _http._tcp          local 100
```

Alors qu'ici on publie un « sous-service » `printer` nommé "*A printer's web page*" (ouvrez bien l'œil, il y a une virgule, pas un point) :

```
% dns-sd -R "A printer's web page" _http._tcp,_printer local 101
```

Avec la syntaxe standard des fichiers de zone du DNS (RFC 1035, section 5), les mêmes déclarations seraient :

```
; One PTR record advertises "A web page"
_http._tcp.local. PTR A\032web\032page._http._tcp.local.

; Two different PTR records advertise "A printer's web page"
_http._tcp.local. PTR A\032printer's\032web\032page._http._tcp.local.
_printer._sub._http._tcp.local.  PTR A\032printer's\032web\032page._http._tcp.local.
```

Que fait SD lorsque plusieurs protocoles assurent à peu près la même fonction (section 8) ? C'est un cas courant, pour des raisons historiques, ou bien lorsque plusieurs équipes n'ont pas pu résister à la tentation de développer leur propre jouet. Par exemple, la fonction d'impression peut être faite par LPR (RFC 1179) ou IPP (RFC 2910), sans compter des protocoles spécifiques d'un vendeur. Le principe de SD dans ce cas, est de décider qu'un des protocoles est le « vaisseau amiral ». Une instance a alors tout intérêt à mettre en œuvre ce protocole privilégié pour assurer le service. Toutefois, si ce n'est pas le cas, elle doit quand même créer un enregistrement SRV pour ce protocole, avec un port de zéro (signifiant qu'il n'est pas utilisable) afin que la détection de duplicata fonctionne.

En général, un client SD donné ne s'intéresse qu'à un seul service. S'il veut imprimer, il cherche une imprimante, et se moque des autres services existants sur le réseau. Toutefois, lorsqu'on veut analyser et déboguer un réseau où il y a du SD, il est pratique de pouvoir demander « toutes les instances de tous les services ». Cela se fait (section 9) en demandant `_services._dns-sd._udp.DOMAIN`.

Au fait, comment est-ce que les informations SD arrivent dans le DNS (section 10) ? Fondamentalement, ce n'est pas notre problème : le RFC 6763 normalise les interactions sur le réseau, pas la façon dont chaque machine est configurée, et dont cette configuration arrive dans le DNS. Il peut y avoir deux méthodes : si on utilise "*Multicast DNS*", chaque machine connait sa propre information et y répond. Sinon, il faut configurer un serveur DNS traditionnel. Cela peut se faire à la main (en éditant le fichier de zone) mais SD est plutôt conçu pour des réseaux non-gérés, où il n'y a pas d'administrateur système (même pas d'administrateur système incompétent). Il faut donc plutôt envisager des mécanismes automatiques, par exemple un programme qui balaie le réseau (en utilisant d'autres protocoles) puis produit le fichier de zone (cette possibilité est citée par le RFC mais semble peu déployée en pratique).

Comme un des buts de SD est de permettre l'énumération de choses inconnues, par exemple lorsqu'un portable arrive sur un nouveau réseau, et sans configuration manuelle préalable, comment apprendre les noms de domaine à utiliser dans les requêtes SD (section 11) ? S'il n'y a pas de serveur DHCP qui répond, il faut sans doute utiliser le `.local` de "*Multicast DNS*". S'il y a un serveur DHCP, il indique un nom par l'option 15, `Domain` (RFC 2132), ou par l'option 119 `Domain Search` (RFC 3397). À noter qu'en IPv6, ce sera sans doute RA et non pas DHCP qui sera utilisé, et qu'il peut aussi indiquer un domaine (RFC 8106).

Une fois qu'on a ce nom, on peut former des requêtes spéciales pour découvrir d'autres choses :
— `b._dns-sd._udp.DOMAIN` donne la liste de tous les domaines disponibles pour l'exploration,
— `db._dns-sd._udp.DOMAIN` donne le domaine recommandé pour l'exploration,
— `r._dns-sd._udp.DOMAIN` et `dr._dns-sd._udp.DOMAIN` les ou le domaine à utiliser pour enregistrer des noms via les mises à jour dynamiques du DNS.

Notons que beaucoup de clients SD n'utiliseront pas ces requêtes et se contenteront du premier domaine déterminé (`.local` ou bien celui appris par DHCP).

Le RFC recommande, en section 12, aux serveurs DNS qui répondront au requêtes SD d'inclure dans la section additionnelle des informations supplémentaires, non indispensables, mais qui peuvent diminuer le trafic en évitant au client de poser la question par la suite. Par exemple, lorsqu'on envoie une réponse SRV, inclure les adresses du serveur nommé dans l'enregistrement SRV est raisonnable. Bien sûr, le client est libre d'en tenir compte ou pas (il peut estimer que, pour des raisons de sécurité, il vaut mieux reposer la question : la section additionnelle est un bon endroit pour tenter d'empoisonner un client DNS avec des informations bidon).

La section 13 contient des exemples réels, en utilisant le domaine `dns-sd.org`, qui est servi par des BIND standard (SD ne nécessite aucune modification des serveurs de noms). Pour voir, on va l'interroger avec drill, livré avec ldns <http://nlnetlabs.nl/projects/ldns/> (le RFC utilise l'antédiluvien nslookup et moi je me sers habituellement de dig). Commençons par les services HTTP :

```
% drill -b 4096  PTR  _http._tcp.dns-sd.org.
...
;; ANSWER SECTION:
_http._tcp.dns-sd.org.  60      IN      PTR     \032*\032Google,\032searching\032the\032Web._http._tcp.dns-s
_http._tcp.dns-sd.org.  60      IN      PTR     \032*\032Amazon\.com,\032on-line\032shopping._http._tcp.dns-
_http._tcp.dns-sd.org.  60      IN      PTR     \032*\032Yahoo,\032maps,\032weather,\032and\032stock\032quot
_http._tcp.dns-sd.org.  60      IN      PTR     \032*\032SlashDot,\032News\032for\032Nerds,\032Stuff\032that
_http._tcp.dns-sd.org.  60      IN      PTR     \032*\032DNS\032Service\032Discovery._http._tcp.dns-sd.org.
...
```

L'option `-b` (indiquer au serveur la taille des réponses qu'on peut recevoir) est là car la réponse est de grande taille. Notez aussi que drill a représenté les espaces par un code pour éviter toute ambiguité. Affichée à un humain, cette liste donnerait :

```
* Google, searching the Web.
* Amazon.com, on-line shopping.
* Yahoo, maps, weather, and stock quotes.
* SlashDot, News for Nerds, Stuff that Matters.
* DNS Service Discovery.
...
```

Une fois qu'on a choisi un des services découverts, comment le contacter ?

```
% drill SRV '\032*\032DNS\032Service\032Discovery._http._tcp.dns-sd.org'
...
;; ANSWER SECTION:
\032*\032DNS\032Service\032Discovery._http._tcp.dns-sd.org.     60      IN      SRV     0 0 80 dns-sd.org.
...

% drill TXT '\032*\032DNS\032Service\032Discovery._http._tcp.dns-sd.org'
...
;; ANSWER SECTION:
\032*\032DNS\032Service\032Discovery._http._tcp.dns-sd.org.     60      IN      TXT     "txtvers=1" "path=/"
```

Bref, on peut joindre l'instance `DNS Service Discovery` du service HTTP sur le port 80 en contactant la machine `dns-sd.org`. Le TXT indique une option non triviale, `path=/`. À noter que les noms b et r cités plus haut ne donnent aucun résultat sur le domaine de test `dns-sd.org`.

Voici maintenant un exemple réel, vu avec tcpdump, d'une machine cherchant les instances du service `sane-port` en "*Multicast DNS*" :

```
21:06:04.877355 IP6 (hlim 255, next-header UDP (17) payload length: 47) \
    2a01:e35:8bd9:8bb0:21e:8cff:fe7f:48fa.5353 > ff02::fb.5353: [udp sum ok] 0 \
    PTR (QM)? _sane-port._tcp.local. (39)
```

tcpdump sait décoder le "*Multicast DNS*" et a vu la demande d'énumération (type PTR demandé) des instances du service `_sane-port._tcp.local`.

La section 15 résume ce qu'il faut savoir de la sécurité de DNS-SD. Le RFC est très court à ce sujet et oublie de préciser clairement que DNS-SD n'offre en soi aucune sécurité. S'il est utilisé sur "*Multicast DNS*", il en hérite l'absence totale de protection. Sinon, il a la sécurité du DNS (faible) et ses protections (DNSSEC mais, évidemment, les réseaux locaux sans administrateur n'auront pas DNSSEC, technique encore trop complexe à mettre en œuvre).

Chose rare dans les RFC, les pincipaux choix techhniques faits par les concepteurs de SD sont documentés dans le RFC, dans les annexes. Ce sont de très intéressantes lectures. Ainsi, l'annexe A explique pourquoi SD s'appuie sur le DNS et pas sur un nouveau protocole, peut-être plus adapté : c'est essentiellement pour pouvoir utiliser toute une infrastructure de normes (DNSSEC, mises à jour dynamiques), de logiciels (serveurs, clients comme drill, analyseurs) et de compétences, infrastructure qui a largement fait ses preuves. Technologie mûre, éprouvée, pour laquelle il existe beaucoup d'implémentations, beaucoup de livres et d'articles, et beaucoup de gens compétents : le DNS n'a aucun concurrent réaliste aujourd'hui.

L'annexe B explique pourquoi les noms interrogés sont écrits `INSTANCE.SERVICE.DOMAINE` et pas `SERVICE.INSTANCE.DOMAINE`. Il y a des raisons sémantiques (les noms de domaines sont petit-boutiens et le service est plus général que l'instance) mais aussi techniques (la compression de noms dans le DNS marche mieux dans ce cas).

L'annexe C est une très intéressante réflexion sur les services de découverte en général. C'est le genre de problèmes qui devrait être plus souvent discuté lors des enseignements de réseaux informatiques à l'Université (plus que les sempiternels réseaux de Petri). Certains services de découverte choisissent de donner à chaque instance un identificateur unique et opaque, typiquement représenté sous forme d'une longue chaîne de chiffres hexadécimaux. Le nom montré à l'utilisateur n'est alors que pour information,

le protocole travaille avec les identificateurs uniques et opaques. SD n'a pas choisi cette approche, car elle sépare trop radicalement ce que voit l'utilisateur de ce qui se passe réellement sur le réseau. Par exemple, si deux instances différentes (et ayant donc des identificateurs opaques différents) ont le même nom montré ? Que va penser l'utilisateur de voir deux « Corridor Printer » ? Laquelle est la bonne ? En pratique, dans ce cas et dans d'autres (comme le remplacement d'un engin par un autre à qui on donne le même nom montré mais qui a un identificateur différent), l'utilisateur va devoir tôt ou tard apprendre l'existence des identificateurs cachés et se colleter avec eux. Résultat, pour SD, le nom montré à l'utilisateur et celui utilisé dans le réseau sont toujours le même.

Un autre problème à la frontière entre l'ergonomie et l'informatique est celui des noms configurés à l'usine, avant que la machine ne soit livrée au client (annexe D). Certains fabricants, croyant bien faire, configurent des noms « inamicaux » par exemple en incluant un numéro de série ou bien une adresse MAC. Leur idée est de garantir l'absence de conflit de noms. Toutefois, lorsque SD utilise "*Multicast DNS*", c'est tout à fait inutile, puisque "*Multicast DNS*" dispose d'un mécanisme de détection et de résolution de conflits (RFC 6762, section 9). Et ces noms inamicaux sont incompréhensibles par l'utilisateur. Le RFC prend l'exemple d'un utilisateur n'ayant qu'une seule imprimante. Le nom « Printer » sera unique et un nom globalement unique comme « Printer 18 :03 :73 :66 :e5 :68 » ne servirait qu'à troubler l'utilisateur. S'il a deux imprimantes, la détection de conflits de "*Multicast DNS*" les nommera « Printer » et « Printer (2) », ce qui est certainement mieux que « Printer 18 :03 :73 :66 :e5 :68 » et « Printer 18 :03 :73 :98 :a3 :12 ». Et, s'il a dix imprimantes, il va de toute façon devoir passer un peu de temps pour les nommer et les configurer.

Le RFC recommande donc simplement de choisir des noms d'usine qui décrivent le produit, comme « Xerox Phaser 6200DX » ou « HP LaserJet 4600 » et de compter sur la résolution de conflits de "*Multicast DNS*" pour le cas où deux HP LaserJet se retrouvent sur le même réseau.

On l'a déjà dit, SD n'est pas du tout limité à l'ASCII et peut utiliser Unicode. L'annexe E rappelle le contexte et insiste sur le fait qu'Unicode est une possibilité, pas une obligation. Si l'administrateur des machines n'aime pas Unicode et veut nommer ses machines uniquement en ASCII, il a évidemment le droit.

Normalement, l'IETF ne se préoccupe que de ce qui passe sur le câble et pas des interfaces utilisateur. On trouve donc peu de RFC qui mentionnent des problèmes d'IHM. Mais l'annexe F de ce RFC 6763 estime nécessaire d'insister sur une question qui a une influence directe sur le protocole et les implémentations : le fait que la découverte de services avec SD est un processus **continu**. On ne peut pas se contenter de présenter une liste à l'utilisateur et de le laisser choisir une fois pour toutes. Des nouvelles machines peuvent apparaître à tout instant, et d'autres disparaître (c'est évidemment particulièrement fréquent avec "*Multicast DNS*"). Plus important, des machines peuvent répondre avec retard et un service de découverte qui s'arrêterait au bout de quelques secondes risquerait de les rater. De plus, les concepteurs de SD considèrent que ce n'est pas sympa que de demander à l'utilisateur de relancer une découverte explicitement.

Le modèle privilégié par SD est donc : on affiche **immédiatement** la liste des instances trouvées (même si elle est initialement vide) et on met à jour cette liste en permanence (plus exactement jusqu'à ce que l'utilisateur ferme la fenêtre). La liste affichée doit représenter en permanence l'état actuel du réseau (on débranche le câble : elle se vide, on le rebranche, quelques secondes après les instances réapparaissent).

Ce RFC 6763 arrive longtemps après les premiers déploiements de SD. On n'est pas dans le cas d'un RFC qui propose une nouvelle technologie, encore à implémenter, mais dans celui d'un RFC qui décrit a posteriori un protocole qu'Apple a imposé unilatéralement. L'annexe G décrit la longue histoire de SD, depuis les réflexions originales en 1997, jusqu'au RFC actuel.

SD est aujourd'hui mis en œuvre sur Mac OS, Windows mais aussi sur de nombreux autres systèmes. Il existe par exemple une implémentations en C, mDNSResponder <http://opensource.apple.com/tarballs/mDNSResponder/>, une version Unix, Avahi, et plein d'autres.

À noter que j'avais été le relecteur Gen-ART (cf. RFC 6385) pour ce document : <http://www.ietf.org/mail-archive/web/gen-art/current/msg05815.html>.