RFC 6762 : Multicast DNS

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 20 février 2013

Date de publication du RFC : Février 2013

http://www.bortzmeyer.org/6762.html

Après une très longue négociation (l'IETF n'était pas enthousiaste pour entériner le coup de force d'Apple), le protocole de résolution de noms « "*Multicast DNS*" » est désormais normalisé dans ce RFC. "*Multicast DNS*" permet de résoudre un nom de domaine en données (par exemple en adresses IP) sur le réseau local, sans faire appel à un serveur DNS configuré. Le demandeur claironne la question et la machine qui se reconnaît répond. Les noms résolus par "*Multicast DNS*" utilisent presque toujours le pseudo-TLD `.local`, décidé par Apple.

Le problème que tente de résoudre "*Multicast DNS*" est celui du réseau local rempli de petites machines ("*smartphones*", tablettes, grille-pains, etc) et pas d'administrateur réseaux compétent pour configurer le tout. De tels réseaux sont de plus en plus fréquents. Trouver l'adresse IP du grille-pain par les mécanismes DNS classiques (un administrateur réseau installe un serveur DNS, puis ajoute `grille-pain IN AAAA 2001:db8:1337::2` dans sa configuration) n'est pas envisageable. L'idée de base de "*Multicast DNS*" est de faire en sorte que chaque machine réponde aux requêtes pour son propre nom. Ainsi, pas besoin d'un serveur DNS configuré manuellement. "*Multicast DNS*" est donc une solution « zéro infrastructure ».

"*Multicast DNS*" est issu du projet de remplacement du protocole NBP d'AppleTalk, projet décrit dans le RFC 6760 [1]. L'idée de base, connue sous le nom de « Zeroconf » (le produit commercial d'Apple aujourd'hui se nomme « Bonjour ») était de rendre les réseaux IP aussi simples de configuration (pour un petit réseau sans importance) que l'étaient les réseaux AppleTalk. Au niveau de l'adressage, cela était déjà fait dans les RFC 3927 et RFC 4862, il restait à le faire pour le nommage.

"*Multicast DNS*" est du DNS (RFC 1034 et RFC 1035) au sens où il réutilise le format des paquets, les types de données et la syntaxe des noms. En revanche, le protocole utilisé pour la résolution de noms

1. Pour voir le RFC de numéro NNN, `https://www.ietf.org/rfc/rfcNNN.txt`, par exemple `https://www.ietf.org/rfc/rfc6760.txt`

est nouveau. Il n'y a pas de récursion, la machine demandeuse diffuse simplement sur le réseau local sa question et la machine qui reconnaît son nom répond. "*Multicast DNS*", qui n'est donc pas le DNS, tourne sur un autre port, le 5353.

"*Multicast DNS*" vise donc à être simple pour l'utilisateur (réseau sans administration). Mais le RFC est long et compliqué et plein de détails. Courage.

Avant de décrire plus en détail "*Multicast DNS*", un peu de vocabulaire (section 2) :

— Un enregistrement DNS **partagé** est un enregistrement où plusieurs machines peuvent légitimement répondre, pour le même nom et type.
— Un enregistrement **unique** est un enregistrement qui est censé n'être géré que par une seule machine. "*Multicast DNS*" étant conçu pour des réseaux non administrés, les collisions seront sans doute fréquentes (tout le monde appelera son imprimante « "*printer*" »). Avant de servir un enregistrement unique, les machines devront donc s'assurer de cette unicité, en interrogeant le réseau pour savoir si quelqu'un d'autre répond déjà pour ce nom et ce type. (Pour des raisons de tolérance aux pannes, c'est permis, si les données servies sont les mêmes. Autrement, il y a conflit.)

La section 3 du RFC couvre un point délicat, les noms à utiliser. Le concurrent de "*Multicast DNS*", LLMNR (RFC 4795) utilise des noms normaux comme `gandalf.middleearth.example`. Les concepteurs de "*Multicast DNS*" considèrent que tout le monde n'a pas de nom de domaine où y enregistrer ses machines et qu'il est donc nécessaire d'avoir un espace spécial pour mettre les engins qui répondent en "*Multicast DNS*", le pseudo-TLD `.local`. "*Multicast DNS*" peut résoudre des noms habituels mais le cas privilégié est celui des noms se terminant par `.local`. Normalement, réserver un TLD nécessite une procédure lente et très coûteuse (185 000 $ uniquement pour déposer le dossier!), mais Apple semble avoir des privilèges particuliers et a pris ce TLD sans demander son avis à l'ICANN, ni suivre aucune procédure habituelle. C'était un des aspects les plus brûlants de cette norme, et qui avait suscité le plus de débats. Voir aussi annexe G.

Dans `.local`, toute machine peut « enregistrer » un nom, simplement en s'en proclamant propriétaire et en répondant aux requêtes "*Multicast DNS*" pour ce nom. Ainsi, si je nomme mon "*smartphone*" `dumbledore`, il répondra aux requêtes "*Multicast DNS*" `dumbledore.local`, et ce nom pourra être utilisé en argument des commandes comme ping, ou mis dans un URL (`http://dumbledore.local/`). L'utilisation d'un seul composant avant `.local` (ici, `dumbledore`) est recommandé.

Voici un dialogue "*Multicast DNS*" entre le PC/Ubuntu « tyrion » et le Raspberry Pi <`http://www.bortzmeyer.org/raspberry-pi.html`> « alarmpi ». Tous les deux ont le logiciel Avahi. tyrion veut pinguer alarmpi :

```
09:11:25.650228 IP (tos 0x0, ttl 255, id 0, offset 0, flags [DF], proto UDP (17), length 59)
    192.168.2.26.5353 > 224.0.0.251.5353: [udp sum ok] 0 A (QM)? alarmpi.local. (31)
09:11:25.651092 IP (tos 0x0, ttl 255, id 0, offset 0, flags [DF], proto UDP (17), length 69)
    192.168.2.4.5353 > 224.0.0.251.5353: [bad udp cksum db2!] 0*- [0q] 1/0/0 alarmpi.local. (Cache flush) [2
```

On voit que `192.168.2.26` (tyrion) a demandé à la cantonade (`224.0.0.251`) qui était `alarmpi.local`. La question était posée en "*multicast*" (`QM`). La réponse contenait l'adresse d'alarmpi, `192.168.2.4`. Ensuite, tyrion veut résoudre cette adresse en nom, il va faire une requête `PTR` :

```
09:11:25.779570 IP (tos 0x0, ttl 255, id 0, offset 0, flags [DF], proto UDP (17), length 70)
    192.168.2.26.5353 > 224.0.0.251.5353: [udp sum ok] 0 PTR (QM)? 4.2.168.192.in-addr.arpa. (42)
09:11:25.780463 IP (tos 0x0, ttl 255, id 0, offset 0, flags [DF], proto UDP (17), length 91)
    192.168.2.4.5353 > 224.0.0.251.5353: [bad udp cksum 8d1!] 0*- [0q] 1/0/0 4.2.168.192.in-addr.arpa. (Cach
```

`.local`, quoique non enregistré dans le RFC 2606 est donc un nom spécial. Les noms dans ce pseudo-TLD n'ont pas de signification globale, un peu comme les adresses IP locales au lien des RFC 3927 et RFC 4291. Les requêtes pour un nom se terminant en `.local` sont donc envoyées en "*Multicast DNS*" aux adresses de diffusion restreinte `224.0.0.251` ou `ff02::fb` (désormais enregistrées à l'IANA <https://www.iana.org/assignments/multicast-addresses/multicast-addresses.xml> - idem en IPv6 <https://www.iana.org/assignments/ipv6-multicast-addresses/ipv6-multicast-addresses.xml>, cf. section 22). Les autres noms peuvent aussi être résolus en "*Multicast DNS*" mais ce n'est pas l'usage. Cela permettrait à la résolution de noms locaux de fonctionner, même en cas de coupure de l'accès Internet. Mais cela poserait de gros problèmes de sécurité puisque, en "*Multicast DNS*", n'importe quelle machine du réseau local peut répondre « oui, je suis `onlinebanking.banqueserieuse.example` ». Le RFC demande que DNSSEC, ou un mécanisme équivalent, soit alors utilisé...

Naturellement, il peut y avoir des conflits, puisqu'il n'y a pas de registre chargé d'attribuer les noms de manière unique. "*Multicast DNS*" résoud ce problème en testant un nom sur le réseau local avant de l'utiliser. S'il est déjà utilisé, la machine "*Multicast DNS*" fait marche arrière et choisit un autre nom.

"*Multicast DNS*" peut aussi être utilisé pour la résolution d'adresse en nom (section 4). Dans ce cas, il n'utilise pas `.local` mais les noms en `.arpa` des adresses IP locales au lien, `254.169.in-addr.arpa` et de `8.e.f.ip6.arpa` à `b.e.f.ip6.arpa`.

Une fois les noms décidés, reste à envoyer les requêtes et à y répondre. Il existe deux acteurs, le demandeur, ou client ("*Querier*") et le répondeur, ou serveur ("*Responder*"). "*Multicast DNS*" a deux modes (section 5) : envoi unique et envoi continu. Le premier est très proche du fonctionnement actuel du DNS. Le demandeur "*Multicast DNS*" envoie un paquet vers `224.0.0.251:5353` ou `[ff02::fb]:5353` et attend la réponse. Le client n'a même pas besoin de savoir que c'est une adresse "*multicast*". Il suffit donc de changer quelques lignes de code dans le client, pour dire « si le nom se termine par `.local`, envoie le paquet DNS à `[ff02::fb]:5353`, sinon envoie au résolveur normal ». (Il faut aussi éviter d'utiliser le port source 5353 car celui-ci indique un client "*Multicast DNS*" complet, qui utilise le second mode.) Rappelez-vous que "*Multicast DNS*" utilise le format habituel des paquets DNS donc la construction du paquet de requête et l'analyse de la réponse ne changent pas.

Un tel client simple acceptera probablement la première réponse qui arrivera. Cela a quelques inconvénients. Si on réclame `myprinter.local` et que plusieurs imprimantes sur le réseau local portent ce nom, seule la plus rapide à répondre sera prise en compte. Il est donc préférable de "*patcher*" un peu plus le demandeur et d'en faire un client "*Multicast DNS*" complet, utilisant le second mode.

Ce deuxième mode de fonctionnement, en continu, ne va pas se contenter de la première réponse, mais va continuer jusqu'à une conclusion satisfaisante. Par exemple, si un navigateur Web veut se connecter à `http://myprinter.local/controlpanel`, la conclusion est satisfaisante quand la page est affichée. Si l'utilisateur est en train de naviguer dans le réseau local à la recherche de ressources (par exemple avec le "*Service Discovery*" du RFC 6763), la conclusion n'arrive que lorsque la fenêtre de navigation est fermée (tant que ce n'est pas le cas, il peut y avoir des nouvautés à afficher et il faut continuer l'interrogation). Dans le cas de recherche d'une imprimante cité plus haut, cela permet de ne pas se limiter à l'imprimante qui a le répondeur "*Multicast DNS*" le plus rapide, mais de s'assurer au contraire qu'on a reçu de l'information de toutes les imprimantes.

Pour cela, le répondeur va donc tourner en permanence, et répéter les questions, notant les nouvelles réponses, jusqu'à ce que le programme qui lui a demandé une résolution qui signale qu'il n'est plus intéressé.

Pour indiquer la gestion complète de "*Multicast DNS*", y compris le mode d'interrogation en continu, le demandeur utilise le port source 5353 dans ses requêtes. "*Multicast DNS*" détourne un bit (section 5.4) dans le champ « classe » des paquets (champ quasi-inutilisé puisque seule la classe IN - pour Internet - est réellement déployée) pour indiquer si la réponse doit elle-même être envoyée en diffusion ou en "*unicast*". Le fonctionnement normal est que la réponse elle-même soit diffusée, pour permettre aux autres clients "*Multicast DNS*" du réseau local d'en prendre connaissance, économisant ainsi du trafic. Le bit est mis alors à zéro dans ces requêtes QM ("*Query Multicast*", comme dans l'exemple entre tyrion et alarmpi plus haut), et c'est en général ce que vous verrez dans la sortie de tcpdump si vous surveillez le trafic "*Multicast DNS*". Sinon, si le bit est mis à un, les requêtes sont QU ("*Query Unicast*") et la réponse sera envoyée en "*unicast*". Voici le dialogue précédent, entre tyrion et alarmpi, vu par tshark : (en ligne sur `http://www.bortzmeyer.org/files/mdns1-tshark.txt`).

Et le répondeur, que doit-il faire (section 6) ? "*Multicast DNS*" n'a pas la distinction entre résolveur/cache et serveur faisant autorité, qui est si importante dans le DNS. Un répondeur fait toujours autorité, il ne met jamais dans la réponse les informations qu'il a apprises sur le réseau. Avec "*Multicast DNS*", on ne sert que ses propres informations. Si le répondeur a l'information demandé, il répond avec autorité (bit AA - "*Authoritative Answer*" - mis à un), sinon, il ne répond tout simplement pas. C'est donc un comportement très différent de celui d'un serveur DNS habituel.

Si le répondeur a déterminé qu'un nom, mettons `example.local`, était unique et chez lui, il répond positivement aux requêtes sur ce nom, s'il a des données sur le type demandé. Il répond négativement s'il n'en a pas (par exemple à une requête pour des adresses IPv4 si le répondeur ne connait qu'une adresse IPv6, sections 6.1 et 20). Si le nom est partagé, et que le répondeur n'a pas d'information, il ne répond pas, il n'envoie pas de NXDOMAIN. S'il a de l'information (par exemple, il sait qu'il est le seul `majoliemachine.local` sur le réseau) et que la question porte sur un type qu'il n'a pas, il renvoie une réponse avec un enregistrement NSEC (voir l'annexe E pour les raisons du choix). Celui-ci joue un rôle analogue à celui qu'il a en DNSSEC : dire clairement qu'on n'a pas une information (voir section 20 aussi).

Comme tous les protocoles « zéro configuration », "*Multicast DNS*" peut être assez bavard et il y a donc des règles pour limiter le trafic, comme de retarder aléatoirement la réponse, pour éviter que tous les répondeurs ayant une information ne veulent émettre en même temps sur l'Ethernet.

À noter que, dans la grande majorité des cas, le répondeur ne répond pas en "*unicast*" mais en "*multicast*". Le but est de permettre aux autres machines "*Multicast DNS*" d'observer les réponses (cf. section 10.5 et annexe D). D'une manière générale, les répondeurs coopèrent en écoutant les réponses des autres. Si le nom est partagé, il n'y a pas de problème. Si un répondeur détecte une réponse pour un nom qui est censé être unique et dont il est détenteur, il y a conflit et on passe alors à la section 9. (Si les données dans la réponse sont identiques, par contre, il n'y a pas de conflit : cela permet de répartir le travail entre plusieurs répondeurs, par exemple pour la résistance aux pannes.)

Les requêtes de type `ANY` (où le demandeur veut la totalité des enregistrements, quel que soit leur type) sont un cas particulier (section 6.5). Dans le DNS, elles ont une sémantique subtile et beaucoup d'utilisateurs se trompent à ce sujet, croyant par exemple d'un récurseur va leur renvoyer tous les enregistrements qui existent, alors qu'il n'envoie que ceux qu'il a dans son cache. Avec "*Multicast DNS*", c'est plus simple : comme tous les répondeurs sont des serveurs faisant autorité, la réponse à une requête `ANY` est toujours complète, elle inclut tous les enregistrements pour un nom.

J'ai déjà parlé de l'intensité du trafic que peut générer "*Multicast DNS*" sur le réseau local. La section 7 regroupe un ensemble de techniques de « réduction de bruit », qui permettent de limiter ce trafic. Bien sûr, sur un LAN, la capacité réseau ne coûte pas trop cher. Néanmoins, il est toujours plus sympa de ne pas abuser de la ressource partagée. Par exemple, si un demandeur connait déjà une partie des

réponses, il les inclut dans le paquet de requête. Les répondeurs savent alors que ce n'est pas la peine d'y répondre. (Cela ne concerne en général que les noms partagés.) On économise ainsi un paquet (celui de la réponse).

Autre cas, si un répondeur s'apprête à répondre mais voit passer sur le réseau une réponse identique à celle qu'il allait donner, il se tait. Si plusieurs répondeurs ont les mêmes données, il n'est donc pas nécessaire, bien au contraire, qu'ils répondent tous. Au passage, c'est une excellente illustration de l'intérêt qu'il y a à envoyer les réponses en "*multicast*", pour que les autres puissent les lire.

Avant de commencer son service, un répondeur doit savoir ce qu'il y a sur le réseau local, et notamment si d'autres répondeurs traiteront le même nom. C'est le sondage ("*probing*") et c'est couvert en section 8. Le sondage est suivi de l'annonce ("*announcing*"). Ces deux opérations doivent être faites dès que la connectivité de la machine change (démarrage, arrivée sur un nouveau réseau, coupure/reconnexion du câble, etc).

D'abord, le sondage. Si le répondeur gère des noms censés être uniques, il doit envoyer une requête sur le réseau pour ces noms (avec le type `ANY`). Normalement, elle ne doit pas avoir de réponse. Avant d'envoyer la requête, on attend un court délai aléatoire (entre 0 et 250 ms), aléatoire afin d'éviter, en cas de coupure de courant, par exemple, ou de remise en service d'un commutateur, que tout le réseau ne hurle en même temps. Cette requête est répétée trois fois mais cela veut dire que le délai total, pour une machine existante, pour défendre son nom contre un nouveau venu, est relativement court, sur certains réseaux lents. Le sondage peut donc ne pas détecter tous les conflits et il est donc nécessaire, en phase de fonctionnement normal, d'écouter en permanence les requêtes des autres et d'y répondre immédiatement (pas de délai aléatoire, cette fois).

Si un conflit est détecté, la machine doit renoncer à ce nom et en chercher un autre (par exemple en demandant à son utilisateur humain).

Ça, c'était le sondage, pour voir si quelqu'un utilisait « mon » nom. Et l'annonce ? Décrite en section 8.3, elle consiste, une fois que le sondage est fait, à répondre (sans qu'il y ait eu de question) en diffusant tous ses enregistrements. Un écoutant passif saura alors que ce nom est pris et pourra mettre à jour son cache. Pour être sûr que cette « réponse » soit reçue, on en transmet au moins deux exemplaires. Par contre, on ne répète pas cette réponse non sollicitée périodiquement ; elle n'est renvoyéee que si la connectivité réseau change ou bien si les données changent (section 8.4). Dans ce dernier cas, il n'est pas nécessaire de répéter l'étape du sondage, puisqu'on sait que le nom est déjà unique.

Maintenant, voyons un cas très difficile mais qui est fréquent sur les réseaux non-gérés pour lesquels "*Multicast DNS*" est fait : le conflit (section 9). Un répondeur a un nom unique, `monmactrescher.local` et il entend tout à coup sur le réseau une réponse "*Multicast DNS*" pour le même nom ! Il y a conflit ! Quelqu'un d'autre a pris ce nom ! Si les données sont les mêmes, ce n'est pas grave (c'est même une technique utilisée par "*Multicast DNS*" pour la résistance aux pannes). Mais si elles sont différentes, que faire ? Si un demandeur envoie une requête pour l'adresse IP de `monmactrescher.local`, que va-t-il recevoir ? La mienne ou celle du méchant qui a pris le même nom pour sa machine ? Le résultat ne sera pas du tout le même ! Le RFC impose que, dès qu'un conflit est détecté, toutes les machines en conflit repartent à la section 8 et recommencent les étapes de sondage et d'annonce. Une des deux machines va alors l'emporter, l'autre devra changer de nom. Ce changement peut se faire manuellement (« Une autre machine sur le réseau se nomme `monmactrescher`. Choisissez un autre nom. ») ou automatiquement (le RFC suggère d'ajouter « 2 » à la fin du nom, « 3 » si `monmactrescher2.local` existe déjà, etc.) Dans le cas où le nom a été changé automatiquement, le RFC suggère de prévenir l'utilisateur, par une boîte de dialogue si la machine a une interface graphique, par des moyens adaptés sinon (SNMP, par exemple).

Une des métadonnées dans les enregistrements DNS est le TTL. Quelle valeur doit être utilisée en "*Multicast DNS*" (section 10) ? Le RFC recommande, pour les enregistrements liés à un nom de machine, notamment les adresses IP, 2 minutes (valeur faible, mais rappelez-vous qu'il n'existe pas d'équivalent des résolveurs/caches en "*Multicast DNS*" ; chaque machine fait autorité et a un cache). Pour les autres, la valeur demandée est de 75 minutes. Cette valeur, au contraire de la première, peut sembler élevée. Des informations dépassées pourraient rester plus d'une heure dans les caches. Mais plusieurs techniques permettent de limiter le risque :

— Les paquets d'adieu : lorsqu'une machine sait que ses enregistrements vont changer (par exemple lorsqu'elle est en train de s'éteindre), elle envoie une réponse avec ses données, mais un TTL de zéro, pour vider les caches.
— Les réponses gratuites : lorsqu'une machine connait un changement, dans ses données ou dans sa connectivité, elle envoie une réponse non sollicitée, qui va remplacer les vieilles valeurs qui se trouvaient dans les caches.
— L'observation passive des échecs : la section 10.5 décrit un algorithme astucieux. Si un répondeur a une information dans son cache (« l'adresse IP de `le Mac de Mic.local` est `2001:db8:1337::2` »), qu'il voit passer une requête par un tiers (« quelle est l'adresse IP de `le Mac de Mic.local` ? ») mais qu'il ne voit passer aucune réponse (le Mac de Mic est éteint...), alors ce répondeur sait qu'il peut virer l'enregistrement de son cache.

Vous avez déjà noté (je reviendrais là-dessus plus tard en discutant la section 21 du RFC) que "*Multicast DNS*" n'offre aucune sécurité : n'importe quelle machine peut répondre n'importe quoi. Pour limiter les risques, le RFC impose quelques vérifications sur l'origine du paquet (section 11). Le test que le TTL était à 255, qui était dans les premières versions du protocole, a été abandonné (j'ignore pourquoi car ce test, décrit dans le RFC 5082, est simple et efficace). En revanche, le récepteur d'un paquet "*Multicast DNS*" envoyé à une adresse "*unicast*" doit vérifier que l'adresse IP source est bien sur le réseau local (un test qui, pour être utile, suppose que le pare-feu interdit l'entrée du réseau local aux paquets ayant une adresse IP source de ce réseau). Si le paquet a une adresse de destination "*multicast*", pas besoin d'autre test, on compte que le routeur d'entrée de site ne l'aurait pas laissé passer, de toute façon.

Les noms utilisés par "*Multicast DNS*" ont quelques caractéristiques spéciales par rapport au DNS auquel on est habitué. La section 12 liste ces caractéristiques, qui peuvent dérouter l'administrateur système qui connait le DNS (s'il ne ne connnait pas, il peut sauter cette section). D'abord, les noms en `.local` n'ont une signification que... locale. Autrefois, le DNS était prévu pour qu'un nom ait partout le même sens et que sa résolution donne partout le même résultat. Aujourd'hui, avec des bricolages comme le "*split view*" de BIND, ou comme les pare-feux, ce n'est déjà plus le cas. Avec "*Multicast DNS*" c'est encore plus marqué. `machine à café.local` ne pointera donc pas vers les mêmes données selon le lieu où on se trouve.

D'autre part, il n'y a pas de délégation dans "*Multicast DNS*" et donc pas d'enregistrements NS. Si une machine répond à `machine à café.local`, rien ne garantit qu'elle réponde à `sucre.machine à café.loca`

Ah, et un point de multilinguisme : le FQDN n'est pas prévu pour être montré aux utilisateurs. Le RFC précise donc qu'il ne faut pas traduire le `.local`...

Comme tout le monde n'a pas forcément envie de faire du "*Multicast DNS*", la section 13 du RFC précise que le fait de passer en "*Multicast DNS*" pour un nom qui ne se termine **pas** en `.local` devrait être configurable par l'utilisateur. Et, pour des raisons de sécurité, devrait être débrayée par défaut. (Sinon, `www.mabanque.example` pourrait être résolue par "*Multicast DNS*", avec son absence de sécurité...)

Dans les exemples cités ci-dessus, vous avez déjà vu des noms qui peuvent sembler surprenants, par exemple parce qu'ils comportents des espaces. La section 16 discute du cas des noms ayant des caractères sortant du traditionnel LDH ("*Letter-Digit-Hyphen*" avec "*Letter*" restreint à US-ASCII). Il faut bien noter que, contrairement à ce qu'affirme une légende tenace <`http://www.bortzmeyer.org/`

pourquoi-idn-et-pas-un-dns-unicode.html>, le DNS a toujours permis tous les caractères. La restriction à LDH est une tradition, maintenue de crainte de planter les applications qui se croient dans un monde purement anglophone. "*Multicast DNS*" a l'avantage de la nouveauté et n'a pas d'héritage à assumer : dès le début, tous les noms peuvent être en Unicode, encodé en UTF-8 (dans le profil du RFC 5198). Notez bien que "*Multicast DNS*" n'utilise **pas** Punycode (l'annexe F discute cet excellent choix).

Cela a toutefois quelques conséquences qui peuvent sembler déroutantes. Ainsi, "*Multicast DNS*", comme le DNS, prétend être insensible à la casse. Mais ce n'est vrai que pour les caractères ASCII. La machine `MyPrinter.local` va répondre aux requêtes "*Multicast DNS*" pour `myprinter.local` mais `Machine À Café.local` ne répondra pas pour `machine à café.local`! Pour faire une telle équivalence, il faurait fallu s'aventurer dans la jungle des conversions de casse en Unicode, un sujet très complexe (dans certains cas, cela dépend de la langue).

Pour les mêmes raisons, "*Multicast DNS*" ne fournit pas d'équivalence automatique, par exemple entre version accentuée et non accentuée d'un nom. Si on cherche `machine à café.local`, on ne trouvera pas `machine a cafe.local`. Si on veut que cela fonctionne, il faut que le répondeur décide de répondre à tous ces noms (ou bien que quelqu'un ait créé des enregistrements CNAME).

Continuous cette longue exploration des points de détails qui font que "*Multicast DNS*" est plus compliqué qu'il n'en a l'air à première vue. La section 17 étudie le problème de la taille des paquets. Au tout début, il y a très longtemps, du temps des dinosaures, les paquets DNS étaient limités à 512 octets. Cette limite a sauté il y a des années, et, de toute façon, elle n'a guère de sens en réseau local, le domaine de "*Multicast DNS*". Il n'y a pas non plus de problème de découverte de la MTU du chemin. Les paquets "*Multicast DNS*" peuvent donc avoir la taille de la MTU du réseau. La fragmentation IP est bien sûr possible, mais déconseillée.

Quel est le format exact des paquets "*Multicast DNS*" (section 18)? Bien sûr, c'est du DNS. Mais il y a quelques particularités. Je ne vais pas répéter toute la section 18 mais je note deux points :

— Le bit TC (troncation) a une sémantique différente de celle du DNS : utilisé dans une question, il indique que le demandeur connait d'autres réponses, qu'il va les envoyer, et que le répondeur devrait donc attendre un peu avant de répondre.
— Le champ Classe de la question a été détourné : son bit de plus fort poids indique, s'il est mis à un, qu'on préfère une réponse en "*unicast*". Il n'est donc pas possible, avec "*Multicast DNS*", d'avoir des classes ¿ 127 (pas en problème en pratique, seules trois classes ont été enregistrées jusqu'à présent).

D'ailleurs, quelles sont les différences entre "*Multicast DNS*" et le DNS? La section 19 offre un utile résumé. Quelques points qui méritent d'être retenus (la liste exacte est bien plus longue) :

— Port 5353 et pas 53,
— Un TLD dédié, `.local`,
— Pas de SOA, pas de NS,
— UTF-8 systématique,
— Plusieurs questions dans un paquet Question,
— Les requêtes de type ANY donnent toujours le même résultat,
— Il peut y avoir des réponses sans qu'aucune question n'ait été posé (annonces gratuites),
— Surveillance des questions et des réponses des autres, pour apprendre ce qui se passe.

Et, par contre, ce qui ne change pas :

— Même syntaxe des noms de domaine,
— Mêmes types d'enregistrement,
— Mêmes API.

Arrivé à ce stade, il est temps de faire un tour complet de la sécurité de "*Multicast DNS*". La section 21 est consacrée à ce point. Pour résumer, "*Multicast DNS*" n'offre aucune sécurité, encore moins que le DNS normal. "*Multicast DNS*" suppose que tous les participants coopèrent sincèrement. En cas de désaccord, il n'existe par définition aucune autorité supérieure qui pourrait trancher. "*Multicast DNS*" est une parfaite an-archie et, comme pour tous les protocoles « zéro-configuration », est donc très difficile à sécuriser. Si on craint la présence d'un ou plusieurs méchants sur le réseau local, la première solution est d'utiliser un autre protocole. Si on connait les gentils, on peut toujours utiliser IPsec, pour être sûr de ne communiquer qu'avec les gentils (je trouve cette idée curieuse, IPsec n'étant pas spécialement « zéro-configuration »...) DNSSEC est également bien sûr une solution possible (mais il a le même problème). DNSSEC est évidemment impératif si on utilise "*Multicast DNS*" pour résoudre des noms en dehors de `.local`. Notons au passage que ce modèle de sécurité pose un problème à l'utilisateur : celui-ci devra prêter attention aux noms et savoir que `.local` est moins sûr. Un défi pour Monsieur Toutlemonde. C'est en raison de cette sécurité inexistante de `.local` que le RFC demande que les logiciels n'ajoutent pas automatiquement `.local` à un nom qui soit déjà un FQDN (par exemple `www.example.com`) mais seulement à des noms faits d'un seul composant.

La section 22.1 revient sur les domaines que "*Multicast DNS*" considère comme spéciaux et sur leur usage : si les applications peuvent les traiter à part, elles ne sont pas obligées ("*Multicast DNS*" fonctionne donc avec les applications existantes). (Tous les détails sur ces « noms spéciaux » figurent dans le RFC 6761.) Pour que "*Multicast DNS*" fonctionne bien, c'est la bibliothèque de résolution de noms (la libc sur Unix) qui doit reconnaitre ces noms comme spéciaux, et les passer à "*Multicast DNS*". D'autres acteurs de l'Internet doivent également veiller à considérer ces noms comme spéciaux : par exemple, un bureau d'enregistrement ne doit pas permettre leur location, un hébergeur DNS ne doit pas permettre de les gérer, etc.

Dans la plupart des RFC, il n'y a pas de justification des choix effectués, et on passe souvent beaucoup de temps à méditer devant la norme en se demandant « pourquoi X et pas Y ? » Ce n'est pas le cas ici, et ce RFC fournit plusieurs annexes (non normatives) détaillant les raisons des différents choix. Une très bonne lecture pour comprendre. Ainsi, l'annexe A explique pourquoi utiliser un port spécifique (5353) et pas le standard 53. Réutiliser l'ancien port aurait permis de ne pas modifier du tout les bibliothèques de résolution de noms mais :

— "*Multicast DNS*" n'est pas le DNS,
— Utiliser le même port aurait rendu difficile d'utiliser à la fois un répondeur "*Multicast DNS*" et un serveur DNS classique sur la même machine,
— Le port 53, sur certains systèmes, est réservé à root.

L'annexe B, elle, répond à une question plus rigolote : pourquoi utiliser une seule adresse "*Multicast*" (`ff02::fb` en IPv6) plutôt que de condenser le nom demandé, utilisant le résultat de cette condensation comme adresse, diminuant ainsi la charge sur les répondeurs (pas besoin de traiter les paquets qui arrivent pour d'autres noms que ceux qu'on gère) ? IPv6 fait cela dans le RFC 4861. Mais cela a des défauts : les machines ayant beaucoup de noms peuvent avoir besoin de s'inscrire à beaucoup de groupes "*multicast*", on ne peut plus utiliser plusieurs questions dans un même paquet si elles portent sur des noms différents (ou, plus exactement, se condensent en des adresses différentes), et surtout des tas de fonctions de réduction de trafic (comme l'écoute du trafic des autres) ne fonctionnent plus.

L'annexe E explique comment NSEC a été choisi pour encoder les réponses négatives. Le code de retour `NXDOMAIN` est global au paquet et ne convient donc pas s'il y avait plusieurs questions dans une même requête. Une autre possibilité envisagée avait été d'utiliser des enregistrements dont les données étaient de taille nulle. Mais, dans certains cas, on peut avoir besoin de tels enregistrements, sans qu'ils soient interprétés comme une réponse négative. Il ne restait donc que les enregistrements NSEC.

Et l'annexe G ? Elle discute des conflits possibles entre des usages locaux du TLD `.local` et "*Multicast DNS*". Un certain nombre de sites ont en effet utilisé un pseudo-TLD local pour nommer les machines internes. `.local` semble le nom le plus répandu mais on en trouve d'autres (`.prive`, `.intranet`,

.home, etc). Le RFC note que c'est bien sûr une mauvaise idée <http://www.bortzmeyer.org/pourquoi-le-tld-local-n-est-pas-une-bonne-idee.html>, entre autres parce qu'elle rentre en collision avec "*Multicast DNS*". Malheureusement, cette annexe a aussi la mauvaise idée de suggérer d'autres noms possibles, alors qu'ils ont des problèmes analogues.

L'annexe H décrit la longue histoire de "*Multicast DNS*" : depuis la première proposition <http://www.stuartcheshire.org/rants/NBPIP.html> de faire tourner NBP sur IP, en 1997, aux premières mises en œuvre et au premier "*Internet-Draft*" en 2000, jusqu'aux premiers vrais déploiements en 2002.

Finalement, l'IETF a entériné l'existant (processus connu sous le nom peu flatteur de "*rubber-stamping*", allusion aux tampons de l'administration), marquant son mécontentement uniquement par de longs délais.

Aujourd'hui, il existe de très nombreuses mises en œuvre de "*Multicast DNS*". Citons parmi d'autres :
— Le code source Apple, diffusé sous licence Apple Public Source License,
— Avahi, très répandu sur les Unix libres,
— En Mono, le projet Mono Zeroconf <http://mono-project.com/Mono_Zeroconf>,
— En Go, un projet sur github <https://github.com/davecheney/mdns>,
— Et tous ceux listés sur le site officiel <http://www.multicastdns.org/>.

Si vous aimez regardez des pcap, attention sur pcapr.net <http://www.bortzmeyer.org/pcapr.html>, il y a plein de traces étiquetées mdns qui n'en sont pas. Mais d'autres le sont par exemple <http://www.pcapr.net/view/tyson.key/2009/8/2/8/WebSense_and_mDNS.pcap.html> ou <http://www.pcapr.net/view/tyson.key/2009/9/6/6/Storage_Protocols_2.html>.