RFC 6055 : IAB Thoughts on Encodings for Internationalized Domain Names

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 22 février 2011

Date de publication du RFC : Février 2011

https://www.bortzmeyer.org/6055.html

Ce RFC de l'IAB fait le point sur un petit problème lié aux IDN, problème qui, semble t-il, n'avait jamais été traité en détail avant. La norme actuelle (RFC 5890 [1] et suivants) est connue sous le nom d'**IDNA** pour "*IDN in Applications*" car elle ne change rien au protocole et que tout le travail doit être fait dans l'application. Toutefois, « application » est un terme un peu vague. Par exemple, un programme écrit en C doit-il convertir le "*U-label*" (forme Unicode du nom) en "*A-label*" (forme ASCII) avant l'appel à la fonction de bibliothèque `getaddrinfo()` ? Ou bien est-ce que `getaddrinfo()` doit le faire seul ? (Pour les impatients, la réponse, qui arrive au terme d'un long RFC, est que l'application ne doit **pas** convertir en Punycode avant d'être **certaine** que la résolution de noms se fera avec le DNS, dans l'espace public.)

Les deux méthodes mènent en effet à des résultats différents dès qu'un autre protocole que le DNS est utilisé pour la résolution de nom. En effet, contrairement à ce qu'on lit parfois, l'appel de `getaddrinfo()` n'est pas un appel au DNS mais aux mécanismes de résolution de nom locaux. Par exemple, sur un système d'exploitation qui utilise la GNU libc comme Debian, le mécanisme se nomme NSS (pour "*Name Service Switch*") et le fichier `/etc/nsswitch.conf` permet de le configurer. Si ce fichier contient une ligne comme :

```
hosts:      files ldap dns
```

1. Pour voir le RFC de numéro NNN, `https://www.ietf.org/rfc/rfcNNN.txt`, par exemple `https://www.ietf.org/rfc/rfc5890.txt`

alors, un appel à `getaddrinfo()` par une application qui veut récupérer l'adresse IP correspondant à un nom de domaine ne produira pas forcément une requête DNS, le nom sera peut-être trouvé dans `/etc/hosts` ou dans LDAP d'abord. Dans ce cas, traduire le nom en ACE ("*ASCII compatible encoding*", c'est-à-dire traduire `palais-congrès.fr` en `xn--palais-congrs-7gb.fr`, selon le RFC 3492) serait une erreur car LDAP, par exemple, est nativement Unicode depuis le début et n'a nul besoin de l'ACE.

La section 1 du RFC rappelle la terminologie, les textes de référence (comme le RFC 2130) et les concepts, notamment la notion d'encodage. Elle rappelle aussi que des opérations aussi simples que l'égalité sont plus complexes en Unicode qu'en ASCII <https://www.bortzmeyer.org/pourquoi-idn-et-pas-html>. Cette même section revient sur la question des API, résumée plus haut. La lecture du RFC 3490 pourrait faire croire à l'implémenteur d'IDN que la situation est simple (figure 1 du RFC) avec l'application parlant au "*stub resolver*" DNS qui parle à l'Internet. La réalité est en fait plus riche comme le montre la figure 2, avec un niveau d'indirection supplémentaire entre le sous-programme appelé par l'application (par exemple `getaddrinfo()`, normalisé dans le RFC 3493) et les techniques de résolution utilisées, le DNS mais aussi LLMNR (RFC 4795), Netbios (RFC 1001), `/etc/hosts` (RFC 952), etc.

Tiens, justement, ces API, elles prennent des noms de domaine sous quelle forme ? UTF-8 ? Punycode ? Autre ? La section 1.1 se penche sur ce problème. La description de `getaddrinfo()` dit juste que le nom (le paramètre `nodename`) est un `char *`, une chaîne de caractères, sans indiquer d'encodage. Les règles des chaînes de caractères en C font que l'octet nul ne peut pas en faire partie, ce qui élimine des encodages comme UTF-16 et UTF-32. En pratique, le nom passé peut être de l'ASCII (c'est le cas s'il a été traité par Punycode), de l'ISO 8859, du ISO-2022-JP (très répandu au Japon) ou de l'UTF-8. On peut noter qu'il est possible de distinguer algorithmiquement **certains** de ces différentes encodages avec des règles comme « Si le nom comporte un ESC, U+001B, alors, c'est de l'ISO-2022-JP, sinon si n'importe quel octet a un bit de poids fort à un, c'est de l'UTF-8, sinon, si le nom commence par `xn--`, c'est du Punycode, sinon enfin c'est de l'ASCII traditionnel. » Ces règles ne sont pas parfaites (celle-ci ne tient pas compte des ISO 8859, d'autre part un nom ASCII traditionnel peut théoriquement commencer par `xn--` sans être un "*A-label*") et on peut préférer des règles plus sophistiquées comme celle exposée dans « "*The Properties and Promizes of UTF-8*" <http://www.ifi.unizh.ch/mml/mduerst/papers/PDF/IUC11-UTF-8.pdf> ». Quant à ISO 8859, c'est le plus ennuyeux, car il n'y a aucun moyen fiable de reconnaître un membre du jeu ISO 8859 d'un autre. Dans un texte long, des heuristiques sont possibles mais les noms de domaines sont trop courts pour fournir assez d'information pour distinguer, par exemple, ISO 8859-1 de ISO 8859-15.

Enfin, il n'y a pas que le `getaddrinfo()` du RFC 3493, il y a d'autres sous-programmes comme, sur Windows, `GetAddrInfoW` qui accepte de l'UTF-16.

Tout cela, c'était pour expliquer que des sous-programmes de résolution de noms comme `getaddrinfo()` ne reçoivent pas toujours assez d'information pour savoir ce qu'ils doivent faire. L'autre moitié du problème est décrit dans la section 2 : il y a d'autres protocoles de résolution que le DNS. Par exemple, la technologie privée d'Apple, Bonjour, a des noms entièrement en UTF-8 (ce qui est conforme aux recommandations du RFC 2277). Une application qui traduirait les noms Unicode en ACE empêcherait donc Bonjour de trouver la machine portant ce nom. Même chose avec les autres protocoles comme le fichier `hosts` (RFC 952. Ainsi, si je mets dans mon `/etc/hosts` sur Unix :

```
64.170.98.32 café-crème
```

je peux l'utiliser sans problème :

```
% ping -c 1 café-crème
PING café-crème (64.170.98.32) 56(84) bytes of data.
64 bytes from café-crème (64.170.98.32): icmp_req=1 ttl=70 time=155 ms

--- café-crème ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 155.568/155.568/155.568/0.000 ms
```

Traduire trop tôt est donc une méthode déconseillée : la traduction en ACE devrait se faire bien plus tard, lorsqu'on connait le protocole de résolution employé. Et cela ne peut pas se faire dans l'application à proprement parler, puisqu'elle ne connait jamais ce protocole.

D'ailleurs, puisque des protocoles comme LDAP utilisent Unicode de bout en bout, pourquoi le DNS n'en fait-il pas autant ? La section 3 décrit ce problème (voir aussi mon article « Pourquoi avoir développé IDNA au lieu d'utiliser directement l'Unicode dans le DNS ? <https://www.bortzmeyer.org/pourquoi-idn-et-pas-un-dns-unicode.html> »). Elle rappelle que, contrairement à ce que disent souvent les ignorants, le DNS est parfaitement capable, depuis le début, de faire passer autre chose que l'ASCII, a fortiori autre chose que le sous-ensemble LDH ("*Letters, Digits and Hyphen*") de l'ASCII. Cette erreur est tellement répandue qu'il a fallu, en 1997, consacrer une part du RFC 2181 (section 11) à tordre le coup à cette idée. Le DNS peut donc gérer des octets où le huitième bit est à un et donc, par exemple, des noms en UTF-8 (il n'est donc pas en contradiction avec le RFC 2277 qui pose comme principe que UTF-8 est l'encodage Unicode de l'Internet). Il y a des raisons techniques pour ne pas utiliser UTF-16 ou UTF-32 (présence d'octets nuls qui pertuberaient les bibliothèques écrites en C pour lesquelles c'est une marque de fin de chaîne) mais UTF-8 ne pose pas de problème. En prime, UTF-8 est un sur-ensemble d'ASCII, toute chaîne ASCII est forcément une chaîne UTF-8, ce qui fait qu'une application peut tout traiter comme de l'UTF-8, même les noms existants.

Donc, le DNS ne pose pas de restrictions ASCII ou LDH. Par contre, les applications peuvent, elles, en imposer. Ainsi, pour le Web, l'ancienne norme des URI, le RFC 2396 (remplacé depuis par le RFC 3986), limitait (dans sa section 3.2.2) le nom de machine présent dans l'URL au jeu LDH (lettres, chiffres et tiret). C'est en raison de cette limitation (présente dans d'autres protocoles) que beaucoup de registres n'autorisent que ces noms à l'enregistrement.

Si le DNS n'est pas limité à l'ASCII, pourquoi ne pas faire des IDN uniquement en mettant carrément de l'UTF-8 dans les noms ? Certes, la représentation des chaînes de caractères dans le DNS <https://www.bortzmeyer.org/representation-texte.html> ne permet pas d'indiquer l'encodage utilisé mais, après tout, le RFC 2277 dit clairement que, sur l'Internet, en l'absence de mention contraire, tout est en UTF-8. Cette méthode a effectivement été utilisée dans certaines zones, notamment privées (non visibles depuis l'Internet public). Elle marche : l'application passe de l'UTF-8 au résolveur qui la transmet aveuglément au DNS, qui sait répondre à ces requêtes. Le principal problème de cette méthode (et qui est à peine esquissé dans notre RFC 6055) est la canonicalisation. Le fait que les requêtes DNS soient insensibles à la casse ne marche pas pour l'Unicode, où les règles sont bien plus complexes (pensons au [Caractère Unicode non montré [2]] allemand dont la majuscule comprend deux lettres, SS, voir le RFC 5198 pour un point de vue plus large). C'est cette absence d'une canonicalisation satisfaisante (qui affecte également les autres protocoles comme LDAP), bien plus qu'une soi-disant incapacité du DNS à gérer l'Unicode, qui explique pourquoi l'actuelle norme IDN n'utilise pas UTF-8 <https://www.bortzmeyer.org/pourquoi-idn-et-pas-un-dns-unicode.html>.

Parmi les autres surprises que nous réserve la résolution de noms, la section 3 note aussi que le nom actuellement résolu n'est pas forcément celui tapé par l'utilisateur. Il est en effet fréquent que le nom tapé

2. Car trop difficile à faire afficher par LaTeX

soit complété par des noms de domaines locaux. Par exemple, sur Unix, le fichier `/etc/resolv.conf` contient souvent `unedirectivesearch` qui indique les noms « suffixes » à essayer (voir la section 6 du RFC 1536, le RFC 3397 et la section 4 du RFC 3646 pour d'autres exemples). Si ce fichier contient `search foo.example bar.example` et qu'un utilisateur tape le nom de machine `toto`, le résolveur essaiera successivement `toto.foo.example` et `toto.bar.example`. Si `foo.example` et `bar.example` utilisaient des règles différentes (par exemple que l'un de ces domaines autorise de l'UTF-8 brut et pas l'autre), la pauvre application qui reçoit le nom `toto` n'aurait pas de moyen de prévoir ce qui va se passer. (Ici, `toto` est en ASCII. Mais si on le remplace par `café`?)

La section 3.1 donne une longue liste d'exemples détaillés des comportements actuels des applications. On y voit entre autres ce qui peut se passer lorsque l'application est « trop maligne » et tente de traduire trop tôt les noms. Par exemple, l'application reçoit le nom `crème`, le traduit en Punycode (`xn--crme-6oa`) mais `resolv.conf` contient un nom en UTF-8, mettons `café.example` et le résolveur (qui ne fait pas de conversion en Unicode) va alors chercher la chaîne incohérente `xn--crme-6oa.café.ex`

Après toutes ses considérations, très détaillées, est-il encore possible de donner une recommandation ? Oui, et elle tient en un paragraphe dans la section 4 : « **Une application qui va appeler un sous-programme de résolution de noms ne doit pas convertir le nom en Punycode si elle n'est pas absolument certaine que la résolution se fera uniquement avec le DNS public.** ». En d'autres termes, l'application plus haut qui appelerait aveuglément, par exemple, le sous-programme `idna_to_ascii_8z()` (qui, avec la GNU libidn `<http://www.gnu.org/software/libidn/>`, convertit du jeu de caractères local vers Punycode) avant de faire un `getaddrinfo()` a tort.

La section 4 de notre RFC laisse une possibilité à l'application d'essayer plusieurs méthodes, comme de tenter `getaddrinfo()` sur le nom brut, puis sur le nom punycodé. Mais ce n'est pas obligatoire. Même recommandation pour les traductions d'adresses IP en nom : une application « intelligente » peut se préparer à recevoir de l'UTF-8 ou du Punycode et réagir proprement dans les deux cas.

Dans tous les cas, une autre recommandation importante du RFC est que les API de résolution de noms spécifient clairement l'encodage qu'elles attendent. Sur Windows, `GetAddrInfoW()` spécifie bien qu'il prend de l'UTF-16 alors que la norme de `getaddrinfo()` (le RFC 3493) n'en parle pas, faisant que la mise en œuvre de `getaddrinfo` sur Windows utilise la page de code Windows alors que celle de MacOS utilise UTF-8.

On peut noter que la norme IDN a été révisée récemment `<https://www.bortzmeyer.org/idnabis.html>` mais, pour les questions discutées dans ce RFC, cela n'a pas de conséquence.

Aujourd'hui, l'erreur qui consiste à traduire en ACE sans faire attention semble assez fréquente. C'est le cas par exemple d'echoping `<http://echoping.sourceforge.net/>` avec sa bogue #3125516 `<https://sourceforge.net/tracker/?func=detail&aid=3125516&group_id=4581&atid=104581>`.

Merci à Pascal Courtois pour sa découverte d'une bogue gênante dans ce texte (pas dans le RFC).