Programmer pour IPv6 ou tout simplement programmer à un niveau supérieur ?

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 10 janvier 2006. Dernière mise à jour le 25 novembre 2008

https://www.bortzmeyer.org/network-high-level-programming.html

---

On explique souvent les problèmes du protocole réseau IPv6 par le faible nombre d'applications portées sur IPv6. Mais faut-il vraiment suggérer aux développeurs de se pencher sur un protocole peu répandu ou plutôt leur proposer de travailler à un niveau d'abstraction supérieur ?

IPv6 est présenté comme le protocole de l'avenir pour Internet depuis dix ou quinze ans. Mais, à l'heure d'aujourd'hui, il reste très peu utilisé. Une des raisons, pointées par exemple par Matthieu Herrb dans son excellent exposé <http://2005.jres.org/paper/72.pdf> à JRES 2005 est que beaucoup d'applications ne sont pas encore compatibles. Notamment, pour traduire un nom en adresse IP, les applications utilisent encore trop souvent l'antédiluvienne fonction `gethostbyname()` au lieu de `getaddrinfo()`, qui avait été normalisée dans le RFC 3493[1], il y a bientôt six ans ! Cela indique la difficulté à faire bouger l'enseignement (des étudiants apprennent encore avec des cours où on leur montre `gethostbyname()`). Bien sûr, les grosses applications très orientées réseaux comme BIND ou Apache ne sont pas concernées mais les ennuis viennent souvent des innombrables applications dont le réseau n'est pas le point principal mais qui ont besoin, par exemple, de récupérer un fichier (songeons aux processeurs XML, qui ont besoin de récupérer schémas ou bien feuilles de style, alors que le gourou XML qui les programme n'est pas forcément un gourou réseau).

Demander aux programmeurs de changer leur code juste pour accepter IPv6, alors que ce protocole est très peu utilisé aujourd'hui, ce n'est pas très motivant. Il vaudrait mieux attaquer le problème différemment. La vérité est que programmer en C avec les fonctions `socket()`, `getaddrinfo()` et autres `connect()`, c'est faire de l'assembleur, c'est travailler à un niveau trop bas. Cette technique a tous les inconvénients de l'assembleur, notamment le manque de robustesse face aux changements futurs. Au contraire, le programmeur devrait :

1. Pour voir le RFC de numéro NNN, `https://www.ietf.org/rfc/rfcNNN.txt`, par exemple `https://www.ietf.org/rfc/rfc3493.txt`

— Le mieux serait qu'il programme dans un langage de plus haut niveau que C. Par exemple, en Python, en utilisant la bibliothèque standard httplib <http://docs.python.org/lib/module-httplib.html>, le programme passe à IPv6 sans même que le programmeur s'en rende compte, juste lors d'une mise à jour de Python (à noter que la bibliothèque urllib2 <http://docs.python.org/lib/module-urllib2.html> est d'un niveau encore plus élevé et que Python a des bibliothèques pour beaucoup d'autres protocoles <http://docs.python.org/lib/internet.html>).
— S'il programme en C, plutôt que de tout faire lui-même (souvent mal), le programmeur devrait compter sur des bibliothèques existantes comme libcurl <http://curl.haxx.se/libcurl/> ou Neon <http://www.webdav.org/neon/> (ces deux excellentes bibliothèques acceptent IPv6 depuis longtemps). (Un exemple de client REST avec curl se trouve en (en ligne sur https://www.bortzmeyer.org/files/get-station.c) et permet de récupérer des informations sur les stations Vélib'.)

D'accord, si on programme juste un client. Mais si on veut faire un serveur ? Ond[Caractère Unicode non montré [2]]ej Sur[Caractère Unicode non montré] me rappelle que Apache, surtout connu comme serveur HTTP, est, depuis sa version 2, bien plus que cela. Il est désormais une plate-forme de développement de serveurs Internet. On peut développer des modules Apache pour écrire des serveurs pour d'autres protocoles, Apache se chargeant de tous les détails. Il est livré avec des modules pour POP et echo et le registre de .cz, où travaille Ond[Caractère Unicode non montré]ej, a réalisé son serveur whois et son serveur EPP en modules Apache.

En travaillant ainsi, le programmeur pourrait oublier les aspects réseaux et se concentrer sur son code « métier ». Et il aurait, gratuitement, IPv6, HTTPS et demain sans doute beaucoup d'autres choses qu'il n'aurait jamais développées lui-même (comme la future séparation de l'identificateur et du localisateur <https://www.bortzmeyer.org/separation-identificateur-localisateur.html>). Espérons que c'est ce qui sera enseigné aux futurs programmeurs.

À titre d'annexe, voici un petit programme tout bête (quasiment l'équivalent de wget) en Python, qui marche en IPv4 et IPv6 sans effort de la part du programmeur :

```
import httplib
import sys
host = sys.argv[1]
path = sys.argv[2]
conn = httplib.HTTPConnection(host)
conn.request("GET", path)
r = conn.getresponse()
print r.status, r.reason
```

Une étude très détaillée de la programmation réseau indépendante de la version (et qui marche donc avec v4 et v6) se trouve dans l'étude d'EGEE, "*IPv6 compliance with C/C++, Java, Python and Perl*" <https://edms.cern.ch/document/971407>. J'ai bien aimé l'approche de créer une petite bibliothèque qui assure l'indépendance par rapport à la version d'IP, de façon à ce que l'essentiel du programme soit identique pour v4 et v6. L'étude de cas concerne un protocole spécifique à EGEE. Si on utilise un protocole courant comme HTTP, il y a souvent encore plus simple : utiliser une bibliothèque comme libcurl ou httplib cités plus haut. Leur étude inclut le code des serveurs. Vous pouvez comparer leur code avec mon serveur echo en Python (en ligne sur https://www.bortzmeyer.org/files/echoserver.py).

Pour terminer sur une nuance, peut-on citer des applications qui ont besoin d'appeler les fonctions de bas niveau, de manipuler directement des adresses IP ? Oui, toutes les applications de débogage, comme tcpdump, de surveillance comme ping ou bien de mesure comme echoping <http://echoping.sourceforge.net/>. Toute règle a ses exceptions mais celles-ci ne doivent pas faire oublier que la règle s'applique toujours pour la grande majorité des applications.

2. Car trop difficile à faire afficher par LaTeX