Le principe de robustesse, une bonne ou une mauvaise idée ?

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 1 octobre 2013

https://www.bortzmeyer.org/principe-robustesse.html

On attribue souvent le succès de l'Internet, et notamment sa **résilience**, au **principe de robustesse**. Ce principe, attribué à Jon Postel, s'énonce « "*Be conservative in what you do, be liberal in what you accept from others.*" ». Que veut-il dire ? Était-ce un bon principe d'ingéniérie à l'époque ? Et l'est-il toujours aujourd'hui ?

On trouve ce principe formellement énoncé dans plusieurs RFC, parfois avec une formulation légèrement différente. Ainsi, le RFC 793 [1] utilise le texte ci-dessus alors que le RFC 791 (également écrit par Postel) dit « "*In general, an implementation must be conservative in its sending behavior, and liberal in its receiving behavior.*" ». En se rappelant que "*liberal*" en anglais veut bien dire ce qu'il veut dire (contrairement au français où il désigne en général un partisan de laisser les forts écraser les faibles), ce principe veut dire qu'un programme qui met en œuvre un protocole réseau devrait être dur avec lui-même mais indulgent avec les autres. Par exemple, si la norme dit « un nombre entier est envoyé sous forme texte et peut comprendre des zéros au début si le nombre est inférieur à 1 000 », le programme qui suit Jon Postel ne va jamais envoyer ces zéros initiaux (afin d'être sympa avec des programmes qui n'ont pas fait attention à ce détail de la norme) mais, lorsqu'il reçoit des entiers, il acceptera des zéros initiaux. S'il est très « postelien », il les acceptera même si le nombre est supérieur à 1 000. Le « postelisme » est donc le contraire du pédantisme : le but est l'**interopérabilité**, que les programmes arrivent à se parler et à travailler ensemble. C'est plus important que de respecter rigoureusement la norme. (Merci à Anahi pour l'exemple, qui vient du proverbe espagnol, « "*como un cero a la izquierda*" », qui veut dire « inutile comme un zéro à gauche ».)

On comprend mieux le principe de robustesse de Postel lorsqu'on compare avec le projet qui était le principal concurrent des protocoles TCP/IP, le projet OSI de l'ISO. La culture dominante du monde OSI était au contraire d'extrême pédantisme et les programmeurs prenaient un malin plaisir à refuser les messages des autres programmes, en arguant du fait qu'ils n'étaient pas parfaitement conformes à la norme. Résultat, l'interopérabilité était très faible. C'est ainsi que, au début des années 90, un stagiaire

1. Pour voir le RFC de numéro NNN, https://www.ietf.org/rfc/rfcNNN.txt, par exemple https://www.ietf.org/rfc/rfc793.txt

et moi avons pu constater que deux mises en œuvre de FTAM (le concurrent OSI de FTP) développées par le même constructeur (Digital), ne pouvaient pas échanger un fichier... Et cela n'avait l'air de gêner personne dans le monde OSI.

Donc, oui, le choix de Postel s'expliquait et a en effet contribué au succès de l'Internet, OSI n'étant plus qu'un énorme gaspillage bien oublié.

Mais l'histoire ne s'arrête pas là. Car le principe de robustesse, comme n'importe quel **bon** principe, peut aussi, si on le pousse jusqu'au bout, donner des résultats ridicules. Car de savoir que les récepteurs seront indulgents peut pousser les envoyeurs à ne pas faire attention et à envoyer n'importe quoi, se disant que le récepteur se débrouillera. Le code du récepteur suit le principe de robustesse ? On va envoyer des entiers avec des zéros initiaux, puis on va envoyer des flottants, puis des chiffres suivis par des lettres en comptant que l'analyseur du récepteur s'arrêtera proprement. Et, rapidement, les programmes bien élevés et qui veulent interopérer seront de facto obligés de gérer ces horreurs, sous peine de ne pas pouvoir interopérer. Un exemple historique parfait est celui du langage HTML. Les premiers navigateurs acceptaient n'importe quoi, donc les webmestres ont pris l'habitude d'écrire du HTML sans faire attention à la syntaxe et, aujourd'hui, la plupart des pages Web sont incorrectes syntaxiquement (malgré d'excellents services comme le validateur du W3C `<http://validator.w3.org/>`) et les navigateurs sont obligés d'accepter cela : un navigateur qui rejetterait les pages mal formées ne pourrait regarder qu'une petite partie du Web. Et aucune autorité (et certainement pas le W3C, dans ce cas, où l'IETF pour les protocoles réseau) ne peut décider autrement, elles n'ont ni le pouvoir, ni l'envie. Résultat, le code est inutilement compliqué et fragile. Et HTML n'est certainement pas le seul exemple.

Le principe de robustesse est particulièrement délicat à appliquer lorsqu'il s'agit de sécurité. Une façon de résumer le principe de robustesse est de dire « ne soyez pas un fasciste psycho-rigide, essayez de comprendre votre interlocuteur au lieu d'insister qu'il a tort ». Bref, il pousse à deviner ce que voulait dire le programme d'en face. En sécurité, c'est souvent une mauvaise idée car on peut deviner mal et ouvrir ainsi une faille de sécurité. Ainsi, le protocole DNSSEC vise à permettre d'authentifier, par des signatures cryptographiques, les enregistrements DNS envoyés. Pour éviter les attaques par rejeu, les signatures DNSSEC ont une durée de vie maximale. Bien des administrateurs DNS ont signé leurs zones sans prêter suffisamment attention à la nécessité de re-signer les enregistrements avant l'expiration. Résultat, les résolveurs DNS validants n'acceptaient plus ces enregistrements. Vu l'ampleur de ce problème, le résolveur DNS validant Unbound `<https://www.bortzmeyer.org/unbound.html>`, par défaut, accepte des enregistrements expirés (jusqu'à 10 % de leur durée de vie totale, c'est réglable avec les paramètres `val-sig-skew-min` et `val-sig-skew-max`). C'est sympa. Mais on voit le paradoxe : un logiciel de sécurité qui décide d'accepter des enregistrements mal signés, pour ne pas être trop méchant... Si Unbound était le résolveur le plus utilisé, on verrait sans doute les administrateurs DNS ne pas trop s'inquiéter de l'expiration des signatures, disant « oui, c'est expiré mais ça va marcher encore quelque temps, grâce au principe de robustesse ».

On lit parfois que ce principe de robustesse n'avait de sens qu'autrefois, dans un Internet mythifié où de joyeux hackers hippies échangeaient des paquets librement et sans se soucier du lendemain. Mais je ne pense pas que cela soit la raison pour laquelle ce principe marchait à l'époque, et marche moins bien aujourd'hui. C'est plutôt que le principe était bon mais que les principes ne doivent pas être appliqués aveuglément, comme des règles religieuses. Ils sont un guide pour l'ingénieur, pas un moyen d'éviter de penser. C'est pour avoir oublié cela que tant de logiciels aujourd'hui doivent se battre avec des pairs mal écrits et qui les bombardent de messages bizarres.

Un débat au sujet de cet article a lieu sur SeenThis `<http://seenthis.net/messages/180689>`.
