RFC 6250 : Evolution of the IP Model

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 20 mai 2011

Date de publication du RFC : Mai 2011

https://www.bortzmeyer.org/6250.html

Qu'offre l'Internet aux développeurs d'application et aux concepteurs de protocole ? Le **modèle de service** de l'Internet n'a pas toujours été bien documenté, et les préjugés et idées fausses abondent chez les programmeurs (l'essentiel du RFC est d'ailleurs composé de l'exposé des affirmations erronées sur le modèle Internet.) En outre, l'Internet ayant crû de manière analogue à un écosystème, sans suivre strictement un plan pré-établi, ce modèle a largement changé, sans que ce changement n'ait été intégré par les développeurs. D'où ce RFC, qui ambitionne de documenter le modèle de service de l'Internet tel qu'il est en 2011. Ce RFC devrait être lu par tous les développeurs d'applications, afin qu'ils apprennent le service que leur offre réellement l'Internet.

Tout le monde sait que l'Internet offre un service de niveau 3 (le protocole IP). Même si les applications y accèdent en général via un service de niveau 4 (typiquement TCP), bien des caractéristiques de la couche 3 apparaissent aux applications (comme les adresses IP). (Le RFC utilise d'ailleurs parfois le terme « application » pour parler de tout ce qui est au dessus de la couche 3.) Quelles sont exactement les propriétés de ce service ? Par exemple, les caractéristiques de la communication (latence, taux de perte de paquets, etc) mesurées au début sont-elles constantes (réponse : non) ? Quelle est la taille d'une adresse IP (ceux qui répondent « quatre octets » peuvent abandonner l'informatique pour l'élevage des chèvres en Lozère) ? Si `192.0.2.81` se connecte chez moi, pourrais-je me connecter à mon tour à cette machine (réponse : plutôt oui, à l'origine, malheureusement non aujourd'hui) ? Si certaines applications (par exemple une application qui inclus un simple client HTTP pour récupérer un fichier distant) peuvent ignorer sans problème ce genre de questions, pour d'autres (logiciel de transfert de fichiers pair-à-pair, client SIP, etc), ces questions sont cruciales et les développeurs ne connaissent pas forcément les bonnes réponses.

Revenons à l'histoire, comme le fait la section 1 du RFC. Le modèle original d'IP (et qui est encore souvent enseigné comme tel dans pas mal d'établissements d'enseignement) était partiellement documenté en 1978 dans l'IEN28 <http://www.rfc-editor.org/ien/ien28.pdf> (les premières publications sur le TCP/IP actuel, les RFC 791 [1] et RFC 793 ne sont venus qu'après). Un des principes était

1. Pour voir le RFC de numéro NNN, https://www.ietf.org/rfc/rfcNNN.txt, par exemple https://www.ietf.org/rfc/rfc791.txt

que les applications étaient très différentes mais que toutes tournaient sur IP, qui à son tour tournait sur tous les réseaux physiques possibles (modèle dit « du sablier », IP représentant le goulet d'écoulement du sablier). Ce modèle a nécessité plusieurs clarifications dans les RFC ultérieurs, par exemple le RFC 1122 en 1989 ou le RFC 3819. Résultat, aujourd'hui, il n'y a pas de document unique qui décrive le modèle de service, ce qu'offre IP aux applications.

Il est d'autant plus nécessaire de comprendre ce modèle qu'il est désormais difficile de le changer. Au début de l'Internet, on pouvait changer un concept et obtenir de tous les développeurs de toutes les applications de s'adapter. Aujourd'hui, la taille de l'Internet et son absence de centre rendent une telle opération impossible. L'Internet s'est ossifié.

Bon, en quoi consiste ce fameux « modèle de service » ? Il est décrit par la section 2. Un modèle de service est défini comme la description du service que rend une couche aux couches supérieures, ici le service que rend IP aux applications. Le schéma n° 1 résume cette notion de couches et la place d'IP dans l'architecture de l'Internet. Et le comportement d'IP ? Sa description officielle était dans la section 2.2 du RFC 791 : un service sans connexion (orienté paquets), acceptant des paquets de taille variable, et les délivrant (mais pas toujours) sans garantir leur ordre ou leur absence de duplication. Bref, un service qui « fait au mieux » (”*best effort*” dans la langue de Jack London). Les émetteurs n'ont pas besoin de se signaler aux destinataires ou au réseau avant d'émettre et les récepteurs n'ont pas besoin de signaler quoi que ce soit au réseau avant de recevoir.

On l'a dit, l'architecture réelle n'est plus vraiment celle du RFC 791, qui était de toute façon décrite de manière très brève dans ce RFC. D'autres documents ont approfondi la question comme le RFC 1958 ou comme le rapport « ”*New Arch : Future Generation Internet Architecture*” <http://www.dtic.mil/cgi-bin/GetTRDoc?AD=ADA426770&Location=U2&doc=GetTRDoc.pdf> » de 2004.

En fait, il est plus facile de décrire ce qu'il n'y a **pas** dans le modèle de service d'IP. La section 3, consacrée aux préjugés erronés, est donc beaucoup plus longue que la section 2. Je ne vais pas la répéter dans son entièreté mais simplement citer quelqu'uns de ces préjugés, en commençant par ceux concernant la connectivité.

Une des suppositions non dites les plus fréquentes dans les applications est que la possibilité d'envoyer un paquet (”*reachability*”) est **symétrique** (section 3.1.1). Un protocole comme FTP (RFC 959), avec sa notion de « voici mon adresse IP, rappelle-moi », repose largement là-dessus. Si la machine A peut envoyer un paquet à la machine B, le contraire est aussi vrai. Ce principe était largement vrai au début de l'Internet (attention, la possibilité d'envoyer était symétrique mais pas le chemin suivi) mais est tout à fait faux aujourd'hui. C'est même plutôt le contraire : la machine typique est coincée derrière un routeur NAT et un pare-feu qui empêchent tout envoi initié par l'extérieur. L'annexe A du RFC 5694 discute plus en détail ce problème et ses conséquences notamment pour les applications pair-à-pair. Bref, la seule chose que laquelle les applications peuvent relativement compter, est qu'une réponse à une requête initiée depuis l'intérieur a des chances d'arriver (c'est le modèle de HTTP). Les requêtes initiées de l'extérieur sont par contre impossibles dans le cas général (RFC 2993).

À noter que, même en l'absence de boîtiers intermédiaires qui bloquent les paquets, certains liens physiques ne sont pas symétriques, notamment en radio, comme l'illustre la section 5.5 du rapport « ”*The mistaken axioms of wireless-network research*” <http://pdos.csail.mit.edu/decouto/papers/kotz03.pdf> ».

Autre supposition fréquente mais erronée, la transitivité. L'idée est que, si une machine A peut joindre B, et que b peut joindre C, alors A peut joindre C (section 3.1.2). Exactement comme pour la

symétrie, ce principe était vrai au début, et ne l'est plus maintenant, en raison du NAT et des pare-feux, mais aussi de protocoles physiques comme 802.11 qui ne garantissent pas une telle transitivité.

Autre erreur que font certains développeurs d'applications, parce qu'ils confondent le modèle original de l'Internet et la triste réalité d'aujourd'hui, la conviction que les paquets signalant une erreur arriveront à l'émetteur du paquet qui a déclenché l'erreur (section 3.1.3). Ces paquets signalant une erreur sont typiquement portés par le protocole ICMP. Des exemples de leur usage ? La découverte de la MTU du chemin (RFC 1191 et RFC 1981) ou bien traceroute (RFC 1812).

Pourquoi ne peut-on plus compter sur ces messages ? D'abord, bien des routeurs ont été configurés pour limiter le rythme d'envoi, avant de rendre plus difficile leur utilisation dans une attaque DoS (RFC 2923, section 2.1. Ensuite, bien des pare-feux ont été stupidement configurés par des incompétents et bloquent tout l'ICMP (malgré les recommandations des RFC 2979, section 3.1.1, et RFC 4890). Résultat, il a fallu développer des mécanismes alternatifs pour la découverte de la MTU (RFC 4821).

Autre erreur : certaines applications croient qu'elles peuvent utiliser la diffusion IPv4 pour transmettre un message à toutes les machines (section 3.1.5). Autrefois, des adresses comme `10.1.255.255` (les seize derniers bits tous à un) envoyaient en théorie un message à toutes les machines du réseau `10.1.0.0/16`, et les routeurs devaient faire suivre ce message au delà du lien local. Comme ce mécanisme facilitait trop les attaques par déni de service, il a été abandonné par le RFC 2644 en 1999. Depuis, la diffusion ne marche plus que sur le lien local. Et encore, car certains liens, dits NBMA, ne la permettent pas. Plus drôle, certains liens comme 802.11 en mode ad hoc autorisent la diffusion mais, en pratique, le message n'atteint pas toutes les machines du lien. Bref, si on veut écrire à toutes les machines d'un groupe, il vaut mieux le faire soi-même, sans compter sur la diffusion.

À ce sujet, la section 3.1.6 note que, contrairement à une croyance courante, un ensemble de paquets "*unicast*" n'est pas forcément plus coûteux ou moins rapide qu'un "*multicast*" ou qu'un "*broadcast*". Sur les liens filaires traditionnels, un seul paquet "*multicast*" sera sans doute plus rapide. Ce n'est plus le cas sur les liens radio où le "*multicast*" devra être transmis au débit de base, même s'il existe des machines ayant une interface radio plus rapide, et qui a négocié un débit plus élevé.

Un problème subtil est causé par le cas où l'application compte sur la latence <`https://www.bortzmeyer.org/latence.html`> des paquets, par exemple pour faire de la visioconférence. Une application qui mesure cette latence peut avoir tendance à mesurer le premier paquet et à la considérer comme représentatif. Mais c'est souvent à tort (section 3.1.6). En effet, pour le premier paquet d'un flot de données, il y a souvent un temps d'établissement (dû à la résolution ARP, par exemple). Et ce sera pire avec les futurs systèmes de séparation de l'identificateur et du localisateur <`https://www.bortzmeyer.org/separation-identificateur-localisateur.html`> qui, tous, prévoient une forte latence pour le premier paquet, lorsqu'il faut résoudre l'identificateur en localisateur.

Autre caractéristique d'un chemin suivi par les paquets : le taux de réarrangement, qui mesure le fait que certains paquets arrivent avant des paquets partis plus tôt. Des études comme celle de Bennett, J., Partridge, C., et N. Shectman, « "*Packet reordering is not pathological network behavior*" <`ftp://ftp.cs.berkeley.edu/ucb/projects/tenet/users/mccanne/tmp/sigcom.ps`> » ou comme les RFC 2991 ou RFC 3819, section 15, montrent que le réarrangement a des conséquences pour les applications. Pour un programme de "*streaming*", par exemple, il oblige à augmenter la taille des tampons. TCP voit ses performances se dégrader si les paquets n'arrivent pas dans l'ordre. Mais le réarrangement est un fait : des tas de raisons peuvent le causer (un routeur qui fait de la répartition de charges entre deux liens, par exemple) et les applications doivent s'y attendre.

Et le taux de pertes des paquets ? Sur quoi peut-on compter ? Beaucoup de gens pensent que les pertes sont rares, et probabilistiques (c'est-à-dire que le destin frappe un paquet au hasard, et qu'en

reessayant, ça va marcher). Mais ce n'est pas forcément le cas (section 3.1.9). D'abord, si l'application a un comportement non-uniforme (des grands moments de silence, puis de brusques envois de nombreux paquets), les pertes se concentreront pendant ces envois, qui rempliront les files d'attente. Ensuite, il existe des phénomènes qui mènent à la perte préférentielle du premier paquet, comme ces routeurs de cœur qui, pour éviter une attaque DoS (envoi de paquets vers beaucoup d'adresses qui ne répondent pas), ne gardent pas en mémoire les paquets qui déclenchent une résolution ARP. Le premier paquet vers une nouvelle destination sera donc forcément perdu. Un autre exemple d'un comportement analogue est le "*Wake-on-LAN*", où le paquet qui déclenche l'allumage de la machine n'est en général pas gardé.

L'Internet garantit-il qu'un chemin existe, à un moment donné, vers une destination (section 3.1.10) ? Oui, c'est toujours largement vrai. Mais cela cesse de l'être dans des réseaux exotiques comme les DTN (cf. RFC 4838) où deux machines pourront communiquer sans jamais être allumées en même temps. Avec humour, le RFC mentionne également le cas de réseaux où les paquets sont transportés par des animaux, peut-être une allusion au RFC 1149.

Pourquoi ces suppositions, qui étaient en général vraies autrefois, ont cessé de l'être (section 3.1.11) ? Il y a deux classes de raisons, une située dans les couches 1 et 2, et une autre placée dans la couche 3. Pour la première, le RFC 3819 donnait d'utiles conseils aux concepteurs de réseaux physiques. En gros, il indique une liste de services que doivent rendre les couches basses à IP et, si celles-ci ne le font pas, comment la couche d'adaptation à IP peut compenser. Ainsi, les RFC 3077 et RFC 2491 décrivent des mécanismes pour compenser l'absence d'atteignabilité symétrique et donc restaurer les services auquel s'attend IP. Mais toutes les couches basses ne le font pas : par exemple, il n'existe pas de moyen standard, lorsqu'une machine est connectée en WiFi ad hoc, de compenser le manque de symétrie dans les liens. Un mécanisme, situé sous IP, reste à élaborer.

Autre classe de causes pour lesquelles les suppositions d'autrefois ne sont plus forcément respectées, les raisons situées dans la couche 3 elle-même. Cela peut être une conséquence d'une politique de sécurité (RFC 4948), qui mène au filtrage. Cela peut être aussi à cause du NAT. Plusieurs applications font beaucoup d'efforts pour contourner ces problèmes. Cela semble une bonne application du principe de robustesse (« soyez strict dans ce que vous envoyez mais laxiste dans ce que vous acceptez »), et, en effet, les protocoles qui réussissent à contourner les limites de la couche 3 ont en général plus de succès que les autres (RFC 5218). Mais le principe de robustesse a aussi l'inconvénient de masquer les problèmes, et de diminuer la pression pour les résoudre. Par exemple (qui n'est pas dans le RFC), Skype réussit à passer via des "*hotspots*" où seul le port 80 fonctionne en sortie, donc les clients de ces "*hotspots*" ne râlent pas contre l'opérateur de ce dernier et celui-ci continue donc à fournir un service de daube. Le RFC suggère, pour sortir de ce dilemme, de fournir un mode spécial pour l'application, activable à la demande, où l'application n'utilisera que des techniques propres et officielles, permettant ainsi de déboguer les réseaux mal fichus, sans que leurs problèmes soient masqués.

Cela, c'était pour la connectivité. Et pour l'adresssage ? La section 3.2 rassemble les suppositions erronées les plus fréquentes sur ce sujet.

D'abord, bien des applications considèrent l'adresse IP comme stable sur une assez longue période (ce qui était vrai autrefois, où les adresses étaient toutes fixes et configurées manuellement). Cette supposition (section 3.2.1) se reflète dans l'API : par exemple, `getaddrinfo()` ne fournit pas d'indication sur la durée de validité de l'adresse récupérée. Même avec plein de bonne volonté, l'application ne peut pas savoir si l'adresse IP récupérée en échange du nom peut être conservée pour une minute, une heure ou un jour (la plupart des applications appelent `getaddrinfo()` et ne remettent jamais en cause son résultat, gardant l'adresse IP pendant toute leur durée de vie, un phénomène connu sous le nom d'épinglage - "*pinning*" - et qui peut avoir des conséquences pour la sécurité <https://www.bortzmeyer.org/dns-rebinding-pinning.html>).

Or, aujourd'hui, avec des techniques comme DHCP, les adresses ne sont pas stables et changent, notamment lorsque la machine se déplace. Cela casse des applications comme SSH. Plusieurs protocoles ont été proposés pour conserver un identificateur stable même lorsque le localisateur change (par exemple HIP <https://www.bortzmeyer.org/hip-resume.html>).

Encore plus répandue, l'idée comme quoi une adresse IP ferait quatre octets (section 3.2.2). Faites le test lors d'un entretien d'embauche : demandez au candidat à écrire un petit programme C qui gère des adresses IP. S'il déclare une variable de type `uint32_t`, cela montre qu'il ne connait pas IP (si la variable est de type `int`, cela montre qu'il ne connait pas C...). Cette idée était juste il y a douze ans, mais ne l'est plus depuis que l'Internet est redevenu multi-protocole, avec la coexistence d'IPv4 et IPv6 (dont les adresses font seize octets).

Plus subtile, la question du nombre d'adresses par interface. Pas mal de programmeurs tiennent pour acquis que « l'adresse IP identifie une interface » (ce qui est faux) et écrivent un code qui ne gère pas vraiment le cas d'adresses multiples. Ce préjugé n'a jamais été réellement justifié (le RFC 791 mentionne explicitement la possibilité d'avoir plusieurs adresses IP par interface) mais il l'est encore moins maintenant, où l'existence de plusieurs adresses IP sur une interface est devenue banale (surtout pour IPv6). On note que HIP, cité plus haut, donnerait raison au préjugé, s'il était déployé massivement : l'identificateur serait en effet unique (par machine virtuelle).

En parlant de l'idée d'une adresse IP par machine, si une machine peut avoir plusieurs adresses IP, est-ce qu'une adresse IP peut être utilisée par plusieurs machines ? Cela mène à la notion d'identité d'une machine. Ce concept n'existe pas réellement dans les protocoles TCP/IP mais pas mal d'applications font comme si (section 3.2.4). Par exemple, une ACL sur un pare-feu, bloquant une adresse IP, signifie que l'administrateur du pare-feu considère que l'ennemi peut être identifié par son adresse IP, ce qui n'est pourtant pas toujours le cas.

Parmi les évolutions technologiques qui empêchent ce préjugé d'être exact : l'"*anycast*" (RFC 4786), ou bien certaines solutions de haute disponibilité. Des RFC comme le RFC 2101 ou le RFC 2775 discutent de cette question de l'unicité de l'adresse.

Plus ridicule mais néanmoins fréquente <https://www.bortzmeyer.org/hostname-physical-location.html>, l'idée que l'adresse IP d'une machine identifie sa position géographique (section 3.2.5). C'est clairement faux aujourd'hui (pensez aux tunnels, par exemple).

Plus subtile, l'idée que l'adresse IP a un rapport avec le routage (section 3.2.6). Par exemple, une application qui déduirait de la proximité de deux adresses la proximité des machines. Avec des protocoles comme Mobile IPv6 ou HIP (déjà cité), cette supposition serait complètement fausse. D'une manière générale, tout système de séparation de l'identificateur et du localisateur casserait cette idée. Dans les textes originels (IEN019 <http://www.rfc-editor.org/ien/ien19.txt> ou IEN023 <http://www.rfc-editor.org/ien/ien23.pdf>),l'adresse IP était présentée comme un localisateur, alors que TCP, par exemple, l'utilise comme identificateur. C'est là la source de cette confusion d'aujourd'hui.

Ces erreurs sur l'adressage sont fréquentes. La section 3.2.10 fournit une perspective plus large, en partant du RFC 1958, dont la section 4.1 avait été le premier texte à insister sur l'importance d'utiliser des noms plutôt que des adresses (celles-ci ayant une sémantique trop floue). Notre RFC 6250 réinsiste fortement sur ce point : c'est bien à tort que des API comme `getaddrinfo()` exposent les adresses IP aux applications. La plupart du temps, celles-ci ne devraient pas avoir accès à ces adresses, ce qui éliminerait la tentation de les utiliser de manière incorrecte. Ceci dit, le conseil du RFC 1958 n'est pas

possible à suivre intégralement aujourd'hui : bien des machines IP (au hasard, mon téléphone portable) n'ont pas de FQDN à elles.

On l'a vu, le terme « applications », dans ce RFC 6250 est utilisé de manière souple, pour décrire tout ce qui se trouve au dessus de la couche 3. la section 3.3 aborde les problèmes spécifiques à une partie de ces « applications », la couche transport (couche 4). Quelles sont les erreurs courantes à propos de cette couche 4 ?

D'abord, un certain nombre d'acteurs de l'Internet croient qu'il est possible d'ajouter de nouveaux protocoles de transport, des concurrents des classiques TCP et UDP (section 3.2.1), tels que ceux enregistrés à l'IANA <https://www.iana.org/assignments/protocol-numbers>. C'était certainement le but originel de l'architecture de TCP/IP. Mais, aujourd'hui, ce n'est plus vrai. L'Internet s'est terriblement ossifié et les nouveaux protocoles de transport ont très peu de chances de passer à travers toutes les "*middleboxes*" mal fichues qui abondent sur le réseau. Souvent, on doit même tout tunneler au dessus d'HTTP. C'est au point où le RFC suggère même de ne pas hésiter à normaliser cette utilisation de HTTP (quelque chose que, traditionnellement, l'IETF avait toujours refusé de faire, au nom de la qualité technique des normes). XMPP le fait, par exemple, dans son XEP 0124 <http://xmpp.org/extensions/xep-0124.html>. SIP n'a pas de méthode standard pour cela, et ça lui a parfois été reproché, alors que le système fermé Skype n'hésite pas à détourner HTTP. Un groupe de travail de l'IETF, HyBi <http://tools.ietf.org/wg/hybi>, travaille sur un mécanisme générique d'utilisation de HTTP pour contourner ces "*middleboxes*" stupides.

Bref, le point le plus étroit du sablier, celui par lequel tout devait passer, et qui était IP dans l'architecture originale, est de plus en plus souvent TCP, voire HTTP. C'est parfois fait au nom de la sécurité (par des gens qui ne connaissent pas forcément ce sujet). L'IETF continue ses efforts pour casser cette dangereuse évolution mais, au cas où ces efforts ne seraient pas couronnés de succès, il faut se préparer à s'adapter.

En parlant de sécurité, quelles sont les idées reçues sur la sécurité (section 3.4) ? Elles sont évidemment nombreuses (et une analyse plus complète figure dans le RFC 3552). Par exemple, bien des applications se comportent encore comme si les paquets n'étaient jamais modifiés en route, à part pour quelques champs bien définis comme le TTL. Le NAT change malheureusement cette situation. On peut parfois restaurer le comportement correct avec des tunnels (RFC 4380) ou avec IPsec en mode transport (RFC 4301).

À défaut d'être respectés dans leur intégrité, les paquets sont-ils au moins privés (section 3.4.2) ? Pas davantage. Il a toujours été possible d'écouter les communications et pourtant on trouve toujours des protocoles et des applications qui s'obstinent à envoyer des mots de passe en clair. Là encore, IPsec, s'il était déployé, pourrait fournir une solution générique, ne nécessitant pas de changer les applications.

Autre erreur que font souvent les débutants en sécurité : croire que, puisqu'un paquet prétend venir de 198.51.100.66, alors c'est vrai (section 3.4.3). La vérité est qu'il est relativement facile de mettre une fausse adresse IP source (la plupart des lecteurs de mon blog le savent depuis longtemps mais on trouve pourtant régulièrement des articles qui parlent d'une attaque par déni de service en disant « elle vient de tel pays » simplement parce que les adresses IP source indiquent ce pays). Il existe des mécanismes techniques qui pourraient permettre d'avoir des adresses sûres, par exemples les adresses cryptographiques du RFC 3972.

En conclusion, que faut-il retenir de ce RFC ? La section 5 fournit les points essentiels :

— Vu le nombre d'applications qui dépendent de TCP/IP pour des travaux critiques, tout changement dans le modèle de service ne devrait être fait qu'avec les plus extrêmes précautions.
— Les invariants (les concepts essentiels sur lesquels on peut compter) devraient être mieux documentés, au lieu de dépendre du savoir informel des vieux sages.
— Chaque couche devrait expliciter clairement le service qu'elle fournit aux couches supérieures, et le service qu'elle attend des couches inférieures.
— Et il faudra de toute façon recommencer le travail de temps en temps, l'évolution étant inévitable.