RFC 5321 : Simple Mail Transfer Protocol

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 1 octobre 2008

Date de publication du RFC : Octobre 2008

http://www.bortzmeyer.org/5321.html

Ce RFC normalise la version actuelle de SMTP, le protocole de transfert de courrier électronique qui est un des grands succès de l'Internet et toujours une de ses applications les plus populaires, malgré les attaques des spammeurs et la concurrence des blogs et de la messagerie instantanée.

Ce protocole a été successivement normalisé par les RFC 788 [1] (qui avait lui-même été précédé par des RFC décrivant des protocoles qui étaient très proches de SMTP), RFC 821, RFC 2821 et désormais notre RFC 5321. Le RFC 788 datant de novembre 1981, SMTP a donc vingt-sept ans.

Parmi ses caractéristiques principales, on trouve la simplicité, qui lui vaut son nom (écrire un client SMTP est simple), le fait qu'il permette le relayage du courrier par des serveurs intermédiaires et le fait qu'il ne spécifie que le **transport** du message, pas son contenu, qui fait l'objet du RFC 5322.

Il est intéressant de noter que, s'il a gagné en fonctions, SMTP en a aussi perdu dans certains cas. C'est ensuite que les fonctions de messagerie instantanée (voir annexe F.6) ont disparu avec le RFC 2821, le routage par la source également (annexe F.2) ou que la simple soumission du courrier par le logiciel client est désormais traitée par le RFC 6409, le SMTP complet étant réservé aux communications entre serveurs de messagerie.

La section 2 du RFC traite en détail du modèle de fonctionnement de SMTP. Le principe est que l'émetteur trouve le serveur distant (en général grâce aux enregistrements MX du DNS) puis ouvre une connexion TCP avec ce serveur distant, connexion sur laquelle on transmet l'adresse de l'expéditeur, celle du destinataire et le message (ces deux adresses forment donc ce qu'on appelle l'**enveloppe**, qui ne fait pas partie du message ; elles ne sont pas forcément identiques aux adresses qu'on trouve dans les

1. Pour voir le RFC de numéro NNN, https://www.ietf.org/rfc/rfcNNN.txt, par exemple https://www.ietf.org/rfc/rfc788.txt

champs `From:` et `To:` du message). (L'annexe B revient également sur certains aspects de la séparation entre l'enveloppe et le message. Ainsi, faire suivre un message en se fiant aux en-têtes `To:` du message, au lieu d'utiliser l'enveloppe, va très probablement produire des boucles sans fin.)

Le serveur distant n'est pas forcément la destination finale, il peut être un simple **relais** (un relais utilise SMTP des deux côtés) ou une **passerelle** (la passerelle parle SMTP d'un côté mais utilise un autre protocole, par exemple un protocole privé, de l'autre côté, cf. section 2.3.10).

En théorie (et la section 2.1 insiste sur ce point), le serveur qui a accepté un message **doit** le délivrer ou bien prévenir l'expéditeur. Aujourd'hui, avec la prolifération du spam et des "*joe jobs*", ce principe n'est plus tenable et cette partie du RFC est donc celle qui est la plus ignorée. Une discussion sur ce point figure dans le RFC, aux sections 6.1, 6.2 et 7.8, qui admettent que « "*In today's world, [...] those principles may not be practical.*" » en rappelant toutefois qu'il faut être prudent, contrairement à beaucoup d'opérateurs de serveurs de messagerie qui jettent de nombreux messages sans garder de trace, ce qui jette un sérieux doute sur la fiabilité du service de courrier.

SMTP est simple, mais extensible. La section 2.2 résume le modèle d'extensibilité de SMTP, qui avait été intégré à partir du RFC 2821. Il repose sur une nouvelle commande pour s'annoncer au serveur distant, `EHLO`, qui remplace l'ancien `HELO`, et qui permet d'annoncer les extensions que l'ou ou l'autre MTA accepte. La section 2.2.1 insiste toutefois sur le fait que l'une des forces de SMTP est sa simplicité et qu'il ne faut donc peut-être pas trop le charger d'extensions, si elles ne sont pas clairement utiles.

La (longue) section 2.3 est consacrée à la terminologie. Elle rappelle notamment la distinction entre enveloppe et message, signalée plus haut. C'est ainsi que la définition des **en-têtes** du message comme `Subject:` ou bien `Date:` est laissée au RFC 5322, alors que celle du **corps** du message est du domaine de MIME (RFC 2045).

C'est également dans cette section de terminologie que sont définis des termes comme MTA ou MUA même si le RFC note (section 2.3.3) que, dans ce monde peu organisé, il ne faut pas toujours prendre les mots trop sérieusement.

La section suivante, 2.4, couvre les principes de base de la syntaxe SMTP, comme le fait que les commandes sont du texte, pas du binaire, qu'elles sont insensibles à la casse, et que les réponses à ces commandes sont des nombres de trois chiffres, conçus pour être interprétés par un programme (par exemple, 500 signifie « Erreur de syntaxe »).

Place ensuite, en section 3, aux procédures SMTP. Le cœur du protocole est là, notamment dans la section 3.3 qui explique l'enchaînement des commandes. Quatre sont particulièrement importantes, `EHLO` pour se connecter (section 4.1.1.1), `MAIL FROM` pour indiquer l'expéditeur (section 4.1.1.2), `RCPT TO` pour le destinataire (section 4.1.1.3) et `DATA` pour envoyer le message (section 4.1.1.4).

Voici un exemple d'une session SMTP, telle que l'affiche, pour nous aider au déboguage, le client SMTP msmtp <`http://msmtp.sourceforge.net/`>, lorsqu'il est lancé par `echo TEST | msmtp --from=foobar@example.org --debug stephane@bortzmeyer.org` (les lignes préfixées par `<--` sont envoyées par le serveur SMTP de réception, celles commençant par `-->` par celui d'émission) :

```
<-- 220 relay1.nic.fr ESMTP Postfix
--> EHLO bortzmeyer.nic.fr
<-- 250-relay1.nic.fr
<-- 250-PIPELINING
<-- 250-SIZE 10240000
<-- 250-VRFY
<-- 250-ETRN
<-- 250-ENHANCEDSTATUSCODES
<-- 250-8BITMIME
<-- 250 DSN
--> MAIL FROM:<foobar@example.org>
--> RCPT TO:<stephane@bortzmeyer.org>
--> DATA
<-- 250 2.1.0 Ok
<-- 250 2.1.5 Ok
<-- 354 End data with <CR><LF>.<CR><LF>
--> TEST
--> .
<-- 250 2.0.0 Ok: queued as 92859A1D95D
--> QUIT
<-- 221 2.0.0 Bye
```

On voit, et c'est une des caractéristiques importantes de SMTP, que l'émetteur peut indiquer ce qu'il veut comme expéditeur. Cela permet les usurpations d'identité mais il n'est pas évident de résoudre ce problème de sécurité sans supprimer en même temps bien des usages utiles du courrier. Si le serveur SMTP initial peut à la rigueur authentifier ses clients, les serveurs suivants n'ont guère le choix, sinon de tout accepter.

Et voici ce qu'affiche un programme Python qui utilise la bibliothèque smtplib <http://docs.python.org/lib/module-smtplib.html>. Notons que les commandes de la bibliotèque portent le nom des commandes SMTP. Ici, on n'utilise que la commande de bienvenue initiale, `EHLO` :

```
% python
...
>>> import smtplib
>>> s = smtplib.SMTP("mail.example.org")
>>> s.set_debuglevel(True)
>>> s.ehlo("mail.foobar.example")
send: 'ehlo mail.foobar.example\r\n'
reply: '250-horcrux\r\n'
reply: '250-VRFY\r\n'
reply: '250-ETRN\r\n'
reply: '250-STARTTLS\r\n'
reply: '250-ENHANCEDSTATUSCODES\r\n'
reply: '250-8BITMIME\r\n'
reply: retcode (250); Msg: mail.example.org
VRFY
ETRN
STARTTLS
ENHANCEDSTATUSCODES
8BITMIME
(250, 'mail.example.org\nVRFY\nETRN\nSTARTTLS\nENHANCEDSTATUSCODES\n8BITMIME')
```

On trouve d'ailleurs des mises en œuvre de SMTP pour tous les langages de programmation, pour Emacs Lisp <http://www.gnu.org/software/emacs/manual/html_node/smtpmail/index.html>, pour Haskell <http://darcs.haskell.org/SoC/haskellnet/HaskellNet/>...

En théorie, le serveur SMTP ne devrait pas du tout toucher au message qu'il convoie. Mais il existe quelques exceptions. Par exemple, la section 3.7.2 couvre le cas des en-têtes `Received:` dans le message (également discutés en section 4.4). Très utiles au déboguage, ceux-ci doivent être ajoutés par chaque serveur SMTP qui a relayé le message. Pour le premier exemple ci-dessus, l'en-tête `Received:` ressemblera à :

```
Received: from bortzmeyer.nic.fr (batilda.nic.fr [192.134.4.69])
        by relay1.nic.fr (Postfix) with ESMTP id 92859A1D95D
        for <stephane@bortzmeyer.org>; Sun, 28 Sep 2008 03:56:11 +0200 (CEST)
```

Enfin, la section 4.1 décrit une par une toutes les commandes SMTP et leur syntaxe. La section 4.2 décrit, elle, les réponses. Composées de trois chiffres, le premier indique si la réponse est positive, négative ou incomplète (ainsi, `2xy` indique que le serveur qui répond est satisfait, `5xy` qu'il ne l'est pas et `4xy` qu'il y a eu un problème temporaire). Le second chiffre, lui, indique le genre de la réponse : `x0z` pour ce qui concerne la syntaxe, `x2z` pour le réseau, etc. On peut donc en déduire que `421` signifie un problème temporaire avec le réseau, forçant à fermer la connexion (section 3.8).

Tout cela est bien beau, mais cela ne fait que spécifier le dialogue avec un serveur distant qu'on a déjà trouvé. Et comment le trouver ? Si je suis un MTA et que je veux transmettre un message envoyé à `postmaster@gmail.com`, comment est-ce que je trouve l'adresse du ou des serveurs responsables de `gmail.com` ? Si on refaisait SMTP en partant de zéro aujourd'hui, j'utiliserai sans doute les enregistrements SRV du RFC 2782. Mais le courrier électronique a été normalisé longtemps avant ce RFC et il utilise donc une sorte d'enregistrements SRV spécifique, les enregistrements MX. Le MTA regarde donc dans le DNS les MX de `gmail.com` :

```
% dig MX gmail.com.
gmail.com. 3600 IN MX 5 gmail-smtp-in.l.google.com.
gmail.com. 3600 IN MX 10 alt1.gmail-smtp-in.l.google.com.
...
```

et sait donc qu'il doit contacter `gmail-smtp-in.l.google.com`. La section 5 du RFC est consacrée à ce mécanisme.

Que faire s'il n'y a pas d'enregistrement MX ? La question a toujours été chaudement discutée chez les passionnés de SMTP, mais particulièrement lors de la rédaction de ce RFC 5321. En effet, l'ancienne règle (section 5 du RFC 2821) était qu'on utilisait, s'il était présent, un enregistrement A (une adresse IPv4), qualifié de « MX implicite ». Ce mécanisme est très critiqué : les domaines ont souvent un enregistrement A pour le Web (afin de pouvoir taper `http://example.net/` et pas seulement `http://www.example.net/`) et la machine qu'il indique n'a pas forcément de serveur de courrier. En outre, que faut-il faire s'il n'y a pas d'enregistrement A mais un AAAA (une adresse IPv6) ? Notre RFC s'éloigne donc du RFC 2821 ici et décide de garder l'ancienne convention du MX implicite (au nom de la continuité), en l'étendant à IPv6 (voir la discussion en section 5.2). Les termes de « *"A RR"* » (*"A resource record"*, enregistrement de type A) ont ainsi disparus au profit de *"address RR"* qui regroupe A et AAAA.

On note qu'il n'existe pas de moyen normalisé de dire « Je ne veux pas recevoir de courrier » ; ne pas avoir de MX ne suffit pas, en raison de la règle maintenue d'un MX implicite ; la méthode la plus courante, mais non standard, est de publier un MX délibérement invalide, pointant par exemple vers `.` (la racine) ou vers `localhost`.

Notons que la section 5 précise qu'un émetteur SMTP doit essayer d'envoyer à toutes les **adresses** IP, pas uniquement à tous les MX (un serveur peut avoir plusieurs adresses).

La section 6 est consacrée aux problèmes et à tout ce qui peut aller mal. Par exemple, la sous-section 6.4 parle des rapports avec les implémentations qui violent la norme. Elles sont très nombreuses (historiquement, Lotus Notes semble avoir la médaille des problèmes). Depuis que le courrier existe, il y a des polémiques entre les « éradicateurs » qui demandent que les serveurs SMTP refusent de compenser ces violations et stoppent tout dialogue avec les logiciels trop bogués et les « conciliateurs » qui, après avoir fait trois génuflexions vers la statue de Jon Postel et cité son « principe de robustesse » (« "*Be conservative in what you send and liberal in what you accept*" »), demandent qu'on essaie malgré tout de parler avec les programmes ratés. Le problème est d'autant plus difficile que tous les développeurs ne sont pas égaux : on peut difficilement refuser de parler aux serveurs de messagerie d'AOL, quelles que soient les curieuses pratiques qu'ils utilisent. Même chose avec des logiciels très répandus comme Microsoft Exchange.

Une autre partie du problème est que certains MUA ont des capacités limitées et qu'on ne peut pas trop exiger d'eux. Ainsi, les serveurs SMTP ont pris l'habitude de réparer les messages, en rectifiant par exemple les champs `Date:` invalides (une bogue très fréquente). Ces réparations devraient, dit le RFC, être limitées aux clients locaux, que l'on connait, et ne pas s'étendre aux machines d'autres domaines. Cette approche est cohérente avec le RFC 6409 (qui sépare la **soumission** d'un message depuis un MUA vers le premier MTA, de l'envoi d'un message entre MTA) mais avec certains logiciels, comme Postfix, elle ne peut pas être configurée.

Toute aussi brûlante est la question de la sécurité du courrier électronique, qui fait l'objet de la section 7. D'abord (section 7.1), il y a le problème de l'authentification : un serveur SMTP peut toujours prétendre que le courrier vient de `pape@nic.va`, il n'y a aucun moyen de vérifier. (De très nombreux projets ont visé à traiter ce problème, voir par exemple les RFC 7208 et RFC 6376 mais il est bien plus compliqué qu'il n'en a l'air, notamment parce qu'il n'existe pas de système d'identité international et reconnu.) Le RFC recommande donc d'authentifier ses messages, par exemple avec le RFC 4880.

La section 7.9, elle, couvre les questions opérationnelles liées à la sécurité. Par exemple, certains fournisseurs de courrier (notamment AOL, qui s'en vante bruyamment), refusent le courrier de certains sites, sur des critères décidés unilatéralement et sans prévenir leurs propres utilisateurs. D'une certaine façon, c'est compréhensible, puisque l'ampleur du spam nécessite des mesures souvent radicales. Mais cela peut, si on poursuit cette logique, mener à un système de courrier qui serait réservé à un oligopole de quelques fournisseurs qui s'acceptent entre eux (le projet du MAAWG).

C'est la même section qui mentionne les relais ouverts, en des termes étonnamment prudents (la pratique générale est de les supprimer). Comme autre exemple de code Python pour faire du SMTP, voici un programme pour tester si un serveur est un relais ouvert (en ligne sur `http://www.bortzmeyer.org/files/open-relay.py`).

Ce RFC ne marque guère de changements par rapport à son prédécesseur, le RFC 2821. Le but principal était de le faire avancer sur le chemin des normes, vers son nouveau statut de « Projet de norme ». Les changements ? Eh bien, le principal étant l'extension du « MX implicite » à IPv6.

Ce RFC a été longtemps retardé par une amusante et exaspérante « querelle des exemples » et a même fait l'objet d'un appel <`http://www.ietf.org/mail-archive/web/ietf/current/msg51960.html`> contre l'obstruction de l'IESG. Celle-ci estimait en effet que le RFC aurait dû utiliser, pour ses exemples, uniquement les noms de domaines du RFC 2606, malgré le fait que le RFC 2821 aie déjà utilisé des noms comme `isi.edu` (l'université de Jon Postel) ou `generic.com` depuis des années.

Des propositions d'architectures radicalement nouvelles, en général fondées sur un modèle où le récepteur « tire », comme avec Atom, plutôt que sur le modèle traditionnel du courrier où l'expéditeur « pousse » ont déjà été faites, par exemple par Dan Bernstein <http://cr.yp.to/im2000.html> ou par Wietse Venema <http://www.securityfocus.com/columnists/479/3> mais n'ont jamais débouché sur des protocoles finalisés, et encore moins sur des implémentations. Ces propositions visent en général à traiter le problème du spam en forçant l'expéditeur à garder le message chez lui, jusqu'au moment de la récupération par le destinataire. Cela résoudrait la question du stockage du spam, mais pas du tout celle de son traitement puisqu'il faut bien notifier le destinataire qu'il a un message, et qu'il doit bien le récupérer pour décider si c'est du spam ou pas.