RFC 1305 : Network Time Protocol (Version 3) Specification, Implementation

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 10 avril 2009

Date de publication du RFC : Mars 1992

https://www.bortzmeyer.org/1305.html

Dans tout système technique complexe, comme l'Internet, on a besoin d'horloges correctes. À tout moment, on lit l'heure et on en tient compte. Par exemple, l'ingénieur système qui lit les journaux d'un serveur a besoin d'être sûr de l'heure qu'ils indiquent. De même, le chercheur qui mesure le temps de propogation d'un paquet IP entre deux machines en soustrayant le temps de départ de celui d'arrivée (cf. RFC 7679 [1]) doit être sûr des horloges des deux machines, sinon la mesure sera fausse. Pour synchroniser des horloges, on pourrait munir chaque ordinateur connecté à Internet d'un récepteur GPS ou équivalent. Mais de tels récepteurs sont relativement chers (surtout pour les engins les moins puissants comme les PC bas de gamme ou les téléphones portables) et ne marchent pas dans toutes les circonstances (le GPS ne fonctionne bien qu'en plein air). Le mécanisme privilégié sur l'Internet est donc que seules certaines machines sont connectées à une horloge physique correcte et que les autres d'ajustent sur celles-ci, par le biais du protocole NTP, objet de ce RFC.

A priori, synchroniser une machine C sur une autre, nommée S, est simple : C demande l'heure à S et S lui donne. C'est ainsi que fonctionnait le protocole "*Time*" du RFC 868. Mais ce mécanisme a un gros défaut : il ne tient pas compte du temps de propagation dans le réseau. Lorsque C reçoit la réponse, elle n'est déjà plus correcte... La résolution d'horloge qu'on peut obtenir est donc limitée par ce temps de propagation. En outre, les algorithmes tels que celui du RFC 868 ne permettent pas de tirer profit de multiples serveurs de temps puisqu'ils ne fournissent aucun moyen de décider quel est le « meilleur » serveur.

Pour expliquer la synchronisation d'horloge, tout un vocabulaire spécifique est nécessaire. Il n'y a pas de glossaire dans le RFC mais des mots come "*delay*" et "*offset*" sont définis en section 2 : pour résumer, une horloge peut avoir un **écart** ("*offset*") avec le « vrai » temps, UTC (ou bien avec une autre horloge,

1. Pour voir le RFC de numéro NNN, https://www.ietf.org/rfc/rfcNNN.txt, par exemple https://www.ietf.org/rfc/rfc7679.txt

si on s'intéresse juste au fait qu'elles soient synchronisées, ce qui n'est pas le cas de NTP). Si cet écart est inférieur à une certaine valeur, on dit que l'horloge est **correcte** ("*accurate*"). Et l'horloge peut avoir un **décalage** ("*skew*"), elle peut avancer plus ou moins vite qu'une autre (ce qui entrainera l'apparition ou l'aggravation d'un écart). Pire, la dérive peut être variable, auquel cas on mesure la dérivée seconde du décalage sous le nom de **dérive** ("*drift*", mais NTP ignore ce facteur). Enfin, l'horloge a une certaine **résolution** ("*precision*"), la plus petite unité de temps qu'elle peut mesurer. Des problèmes de mesure de temps analogues sont présents dans bien d'autres RFC notamment le RFC 2330 mais aussi les RFC 4656 ou RFC 5481. Pour obtenir le temps d'un autre serveur, alors que le délai de propagation est non-nul, NTP utilise des **estampilles** temporelles, des valeurs de la mesure de l'horloge, mises dans le paquet. Après plusieurs paquets échangés, chaque serveur NTP connait le délai de propagation avec un autre serveur (ainsi que, au bout d'un temps un peu plus long, la **gigue**, la variation de ce délai, suite aux hoquets du réseau) et peut donc déduire ce délai des temps mesurés par son pair. Sur une machine Unix, voyons ce que cela donne avec la commande `ntpq -c peers` :

```
% ntpq -c peers
     remote           refid      st t when poll reach   delay   offset  jitter
==============================================================================
+relay1.nic.fr   192.134.4.11     3 u  998 1024  377    0.297   -1.554   0.163
 gw.prod-ext.pri .INIT.          16 u    - 1024    0    0.000    0.000   0.000
+obelix.gegeweb. 145.238.203.10   3 u  695 1024  377    5.226    0.586   1.768
-ntp.univ-poitie 224.22.30.230    3 u  498 1024  377    6.885   -4.565   0.267
*ns1.azuria.net  193.67.79.202    2 u   56 1024  377    2.739   -1.411   0.305
-rps.samk.be     193.190.198.10   3 u  984 1024  377    5.293    5.930   0.317
```

Le pair précédé d'un astérisque est le serveur actuellement sélectionné (ceux précédés d'un plus sont les candidats possibles à son éventuel remplacement). Ceux précédés d'un moins ont un écart trop important. "*jitter*" est la gigue et "*delay*" le RTT en milli-secondes.

Lorsque plusieurs serveurs NTP sont accessibles, NTP sélectionne le meilleur (en tenant compte de divers paramètres comme justement la gigue). Il n'y a pas de vote entre serveurs, NTP est une dictature où le meilleur serveur a toujours raison.

Comme norme, NTP est très stable, et n'avait pas bougé entre cette version 3 et la version 4, apparue dix-huit ans plus tard dans le RFC 5905. La version 2 était décrite dans le RFC 1119 et les différences (aucune n'est essentielle) entre les deux versions sont décrites dans l'annexe D. La version 3 a surtout amélioré les divers algorithmes de traitement des données, sans changer le modèle, et en changeant à peine le protocole. (La première version de NTP était dans le RFC 958.) NTP n'était pas le premier protocole de synchronisation d'horloge, loin de là. Il a été précédé par "*daytime*" (RFC 867) ou "*time*" (RFC 868), ainsi que par des options d'ICMP comme celle du RFC 781. Il y a également eu des mises en œuvre non normalisées comme le démon `timed` sur Berkeley Unix. NTP a aussi été inspiré par le protocole DTS, partie du système DCE mais, contrairement à DTS, il n'impose pas que toutes les machines participantes dépendent du même administrateur. Enfin, il y a eu d'innombrables projets de recherche sur des thèmes bien plus ambitieux comme la détermination de la justesse ou de la fausseté d'une horloge, problèmes que NTP n'essaie pas de traiter. Voir la section 1.1 pour plus de détails sur ces autres techniques. Enfin, la version 4 de NTP, on l'a dit, a été publiée en juin 2010 dans le RFC 5905, sans qu'elle change fondamentalement les principes du protocole.

Le problème de la synchronisation d'horloges est très complexe et plein de détails difficiles. Le RFC 1305 est donc un des plus gros RFC qui soit (plus de cent pages). En outre, l'abondance des formules mathématiques, rares dans les RFC mais nécessaires ici à cause des calculs d'erreur, fait que notre RFC n'est pas uniquement disponible en texte brut <`https://www.bortzmeyer.org/rfc-en-texte-brut.html`>. S'il y a bien une version dans ce format, l'auteur prévient qu'il est recommandé de lire la version PDF, disponible au même endroit <`http://www.rfc-editor.org/rfc/rfc1305.pdf`>. Elle

est la seule à disposer des équations mais aussi de la pagination. Enfin, le caractère plus « scientifique » de ce RFC fait que la section 7, les références, est spécialement longue et évoque plus un article de physique qu'une norme TCP/IP. Le lecteur pressé a sans doute intérêt à lire plutôt l'article de David Mills (l'auteur du RFC), "*Internet Time Synchronization : the Network Time Protocol*" <http://www.cis.udel.edu/~mills/database/papers/trans.ps> qui reprend le plan du RFC mais en omettant les détails (attention, cet article ne décrit que la version 2 de NTP mais les différences ne sont pas vitales). Autrement, la page Web dudit auteur, propose plein d'informations sur NTP <http://www.cis.udel.edu/~mills/ntp.html>.

La section 2 décrit le modèle utilisé. NTP considère que certaines machines ont l'information correcte (par exemple parce qu'elles l'ont obtenu d'une horloge sûre) et que le seul problème est de transmettre cette information aux autres machines. NTP ne s'essaie pas à déterminer si les serveurs ont raison ou tort, il leur fait confiance, il n'existe pas de système de vote, NTP n'est pas une démocratie. Certaines machines ont raison et les autres s'alignent.

Communiquant avec un serveur, un client NTP détermine l'**écart** ("*clock offset*") avec ce serveur, le RTT ("*roundtrip delay*") et la dispersion, qui est l'écart maximal avec l'autre horloge.

Pour cette communication, il existe plusieurs mécanismes (section 2.1), le plus courant étant un mécanisme client/serveur où le client NTP demande au serveur le temps. Celui-ci répond et l'information contenue dans le paquet permettra au client de déterminer les valeurs ci-dessus, notamment l'écart. En fait, NTP est plus compliqué que cela car il existe plusieurs niveaux de serveurs et chaque client utilise plusieurs serveurs, pour se prémunir contre une panne. La proximité d'un serveur avec une « vraie » horloge est mesurée par un nombre, la **strate** (section 2.2), qui vaut 1 lorsque le serveur est directement connecté à l'horloge et N+1 lorsque le serveur est coordonné avec un serveur de strate N.

Le serveur calcule ensuite (par une variante de l'algorithme de Bellman-Ford), le chemin le plus court (en utilisant comme métrique le nombre d'étapes et la strate), et c'est par celui-ci que sera transmise la valeur correcte de l'horloge (je le répète, NTP ne fait pas de pondération entre les serveurs possibles, sauf dans le cas particulier décrit dans l'annexe F, une nouveauté de la version 3 de NTP).

La section 3 décrit le protocole. La plus importante information transportée par NTP est évidemment le temps, exprimé sous forme d'un doublet. La première partie du doublet est un nombre sur 32 bits qui indique le nombre entier de secondes depuis le 1er janvier 1900 (section 3.1). La seconde est la partie fractionnaire de cette même valeur. Cela assure donc une précision d'environ 200 picosecondes. Comme tout le fonctionnement de NTP dépend de la précision de ces estampilles temporelles, le RFC recommande que leur écriture dans les paquets soit, autant que possible, fait dans les couches les plus basses, par exemple juste au dessus du pilote de la carte réseau.

32 bits n'est pas énorme et la valeur maximale sera atteinte en 2036. Les programmes devront donc s'adapter et faire preuve d'intelligence en considérant qu'un nombre de secondes très faible signifie « un peu après 2036 » et pas « un peu après 1900 ». Plus drôle, comme la valeur zéro est spéciale (elle signifie que le temps n'est pas connu), pendant 200 picosecondes, au moment de la transition, NTP ne marchera plus.

Pour son bon fonctionnement, NTP dépend de certaines variables, décrites en section 3.2. Il y par exemple l'adresse IP du pair mais aussi :
— "*leap indicator*" qui indique si une seconde intercalaire est imminente,
— la strate (1 si le serveur a une horloge, de 2 à 255 autrement),
— la précision de son horloge,

— un identifiant sur quatre octets, typiquement une chaîne de caractères pour un serveur de strate un, par exemple GPS pour le GPS, ATOM pour une horloge atomique ou LORC pour le bon vieux LORAN,
— temps d'aller-retour lors de la communication avec le pair,
— etc.

L'identifiant du type d'horloge est indiqué dans la sortie de `ntpq -c peers` si le pair est de strate 1, ou bien par la commande `ntptrace` sous le nom de `refid`. Par exemple, ici, `clock.xmission.com` est synchronisé au GPS :

```
% ntptrace
localhost: stratum 3, offset 0.000175, synch distance 0.072555
tock.slicehost.net: stratum 2, offset 0.000658, synch distance 0.057252
clock.xmission.com: stratum 1, offset 0.000010, synch distance 0.000274, refid 'GPS'
```

NTP peut fonctionner selon plusieurs modes (section 3.3), en client ou en serveur mais aussi en mode diffusion (un serveur qui envoie ses données que quelqu'un écoute ou pas). Ainsi, avec le programme ntpd (voir <`http://support.ntp.org/`>) d'Unix, si on configure dans `/etc/ntp.conf` :

```
broadcast 224.0.0.1
```

(où l'adresse est celle de diffusion) ntpd va diffuser ses informations à tout le réseau local. Si on met :

```
broadcastclient
```

il écoutera passivement les annonces des serveurs qui diffusent. Si enfin, on écrit :

```
server ntp1.example.net
server ntp2.example.net
```

il cherchera activement à se connecter aux serveurs `ntp1.example.net` et `ntp2.example.net` pour obtenir d'eux des informations sur l'heure qu'il est. Les deux serveurs établiront alors une **association** entre eux.

NTP peut se défendre contre un pair qui enverrait des informations aberrantes mais il ne contient pas de système de vote qui pourrait protéger contre des pairs malveillants. Même avec un tel système, si tous les pairs participaient d'un commun accord à l'attaque, NTP serait alors trompé. Il faut donc s'assurer que les pairs sont bien ceux qu'on croit. La section 3.6 (et l'annexe C) décrit les contrôles d'accès qu'on peu utiliser. En pratique, cette sécurité est peu pratiquée ; bien que fausser l'horloge d'une machine puisse avoir de sérieuses conséquences pour la sécurité (journaux avec une heure fausse, par exemple), les attaques sur NTP semblent rares. La principale protection reste le filtrage au niveau du pare-feu.

La section 4 décrit les algorithmes de correction qui permettent de réparer dans une certaine mesure les erreurs des horloges, ou bien celles introduites par la propagation dans le réseau. NTP peut aussi bien filtrer les mesures d'une horloge donnée (en ne gardant que les meilleures) que garder les bonnes horloges parmi l'ensemble des pairs accessibles. C'est une des parties les plus mathématiques du RFC, et qui justifie de lire la version PDF.

La précision obtenue avec NTP dépend évidemment de la précision de l'horloge locale. La section 5 décrit la conception d'une horloge telle qu'utilisée dans le système Fuzzball et permettant d'atteindre les objectifs de NTP.

Le format effectif des paquets apparait dans l'annexe A. Les paquets NTP sont transportés sur UDP, port 123. Chaque message indique le mode de l'émetteur (par exemple client, serveur ou autre), les différentes estampilles temporelles (heure de réception du paquet précédent "*Receive Timestamp*", heure d'émission de celui-ci "*Transmit Timestamp*", etc), précision de l'horloge locale, etc. Les estampilles temporelles sont indiquées dans le format à 64 bits décrit plus haut. Leur usage est décrit en section 3.4. Du fait que chaque paquet contienne toutes les estampilles nécessaires, les paquets sont auto-suffisants. Il n'est pas nécessaire qu'ils arrivent dans l'ordre, ni qu'ils soient tous délivrés (d'où le choix d'UDP comme protocole de transport).

Pour observer NTP en action, on peut utiliser tcpdump :

```
# tcpdump -vvv udp and port 123
tcpdump: listening on eth0, link-type EN10MB (Ethernet), capture size 96 bytes
17:11:46.950459 IP (tos 0x0, ttl 64, id 0, offset 0, flags [DF], proto UDP (17), length 76) munzer.bortzmeyer.or
        Client, Leap indicator:  (0), Stratum 3, poll 10s, precision -20
        Root Delay: 0.037567, Root dispersion: 0.082321, Reference-ID: tock.slicehost.net
          Reference Timestamp:  3448017458.952901899 (2009/04/06 16:37:38)
          Originator Timestamp: 3448018483.951783001 (2009/04/06 16:54:43)
          Receive Timestamp:    3448018483.951863646 (2009/04/06 16:54:43)
          Transmit Timestamp:   3448019506.950407564 (2009/04/06 17:11:46)
            Originator - Receive Timestamp:  +0.000080633
            Originator - Transmit Timestamp: +1022.998624563
17:11:46.950946 IP (tos 0x10, ttl 63, id 0, offset 0, flags [DF], proto UDP (17), length 76) tock.slicehost.net.
        Server, Leap indicator:  (0), Stratum 2, poll 10s, precision -20
        Root Delay: 0.036941, Root dispersion: 0.012893, Reference-ID: clock.xmission.com
          Reference Timestamp:  3448019415.234667003 (2009/04/06 17:10:15)
          Originator Timestamp: 3448019506.950407564 (2009/04/06 17:11:46)
          Receive Timestamp:    3448019506.951188027 (2009/04/06 17:11:46)
          Transmit Timestamp:   3448019506.951214015 (2009/04/06 17:11:46)
            Originator - Receive Timestamp:  +0.000780425
            Originator - Transmit Timestamp: +0.000806425
```

On voit ici que `munzer.bortzmeyer.org`, machine chez Slicehost <`https://www.bortzmeyer.org/slicehost-debut.html`> a contacté le serveur de temps, `tock.slicehost.net` (le second serveur NTP de cet hébergeur se nomme `tick.slicehost.net`) en indiquant que le paquet était émis ("*Transmit Timestamp*" à 3448019506.950407564 (soit le six avril 2009 vers 17 :11 :46). `tock.slicehost.net` répond, renvoie cette estampille dans le champ "*Originator Timestamp*", indique qu'il l'a reçue à 3448019506.951188027, soit 780 microsecondes plus tard et qu'il a répondu à 3448019506.951214015, 26 microsecondes après la réception. `munzer.bortzmeyer.org`, en regardant à quelle heure il a reçu le paquet, peut en déduire le délai d'acheminement et le décalage des deux horloges. Si le serveur venait de démarrer, toutes les estampilles sont à zéro, sauf celle de transmission :

```
22:39:26.203121 IP (tos 0x0, ttl 64, id 0, offset 0, flags [DF], proto UDP (17), length 76) munzer.bortzmeyer.or
        Client, Leap indicator: clock unsynchronized (192), Stratum 0, poll 6s, precision -20
        Root Delay: 0.000000, Root dispersion: 0.000000, Reference-ID: (unspec)
          Reference Timestamp:  0.000000000
          Originator Timestamp: 0.000000000
          Receive Timestamp:    0.000000000
          Transmit Timestamp:   3448384766.203066617 (2009/04/10 22:39:26)
            Originator - Receive Timestamp:  0.000000000
            Originator - Transmit Timestamp: 3448384766.203066617 (2009/04/10 22:39:26)
```

L'annexe E contient des passionnants détails sur les calculs de temps. Par exemple, l'annexe E.3 est consacrée à un exposé plus détaillé des divers services de synchronisation d'horloges publics, par exemples ceux du NIST, le LORAN ou le DCF77 allemand. Notez que, à l'époque de la publication du RFC, le GPS était encore en cours de déploiement.

L'annexe E.4 parle de l'organisation du temps, les calculs calendaires. (Je me permets de placer ici une petite publicité pour le livre de Reingold & Dershowitz "*Calendrical calculations*" <https://www.bortzmeyer.org/calendrical-calculations.html>.)

L'annexe G, chargée en mathématiques et en physique, décrit la modélisation des horloges. En G.1.2, l'informaticien intéressé par Unix trouvera un exposé très détaillé de l'application de ce modèle aux horloges Unix. Les appels système `settimeofday()` et `adjtime()` permettent de modifier l'horloge système, brutalement dans le premier cas et progressivement dans le second (celui utilisé par NTP). `adjtime()` prend bien soit que l'horloge ne recule jamais (si elle est en avance, on la ralentit, mais on ne la fait pas aller en arrière).

L'annexe H est une analyse fouillée des causes d'erreurs dans la synchronisation d'horloges, aussi bien celles dues à l'horloge elle-même que celles dues à la propagation des messages dans le réseau. Ces dernières erreurs sont difficiles à caractériser car elles sont tout sauf aléatoires. (En effet, elles dépendent surtout de la taille des files d'attente dans les routeurs, puisque le temps de propagation, lui, est peu variable, cf. H.4. Les variations de taille de ces files sont assez chaotiques.)

L'annexe I plaira aux programmeurs car elle est la seule qui contient du code source, en C, mettant en œuvre certains des algorithmes décrits dans le RFC.

La mise en œuvre la plus courante de NTP, sur Unix, est celle du serveur `ntpd`. Il se configure dans le fichier `/etc/ntp.conf` et la documentation complète figure en ligne sur <http://doc.ntp.org/>. La commande `ntpdate` permet une mise à jour sommaire, sans faire tourner de serveur :

```
# ntpdate pool.ntp.org
 23 Feb 01:34:10 ntpdate[3335]: step time server 192.83.249.31 offset 2.155783 sec
```

Mais pour une meilleure précision, il faut un serveur tournant en permanence (autrement, il existe une version simplifiée de NTP, SNTP, décrite dans le RFC 4330). Voici par exemple une station de travail ordinaire, synchronisée au serveur NTP de son réseau, `ntp.example.org` :

```
server ntp.example.org
server 127.127.1.0
fudge 127.127.1.0 stratum 13
```

Les deux dernières lignes sont là pour dire à ntpd que l'horloge locale est raisonnablement stable et qu'on peut la considérer comme de strate 13. Comme on ne veut pas forcément que tout l'Internet aille ensuite noyer cette machine sous le trafic NTP et, pire, changer son horloge, on restreint souvent les possibilités de NTP à certaines actions. Par exemple :

```
# Exchange time with everybody, but don't allow configuration.
restrict default kod notrap nomodify nopeer noquery
# Local users may interrogate the ntp server more closely.
restrict 127.0.0.1 nomodify
```

Une fois cette machine synchronisée, les commandes `ntpq` et `ntptrace` permettront de regarder l'état de NTP :

```
% ntptrace
localhost: stratum 3, offset 0.000116, synch distance 0.015000
fuzzer.example.org: stratum 2, offset 0.000149, synch distance 0.009868
192.0.2.77: timed out, nothing received
***Request timed out
```

Ici, le serveur de strate 1, `192.0.2.77` n'accepte pas des interrogation directes de la part des stations, il ne répond donc pas à `ntptrace`. On peut avoir plus de détails sur les pairs avec `ntpq`, par exemple :

```
ntpq> peers
     remote           refid      st t when poll reach   delay   offset  jitter
==============================================================================
*fuzzer.ex       192.0.2.176      2 u   50   64  377    0.304    0.107   0.064
 LOCAL(0)        .LOCL.          13 l    1   64  377    0.000    0.000   0.001

ntpq> clocklist
assID=0 status=0000 clk_okay, last_clk_okay,
device="Undisciplined local clock", timecode=, poll=33834, noreply=0,
badformat=0, baddata=0, fudgetime1=0.000, stratum=13, refid=76.79.67.76,
flags=0
```

qui permet de voir le délai de la communication avec le serveur de strate 2 (ce délai est logiquement de zéro avec l'horloge locale, de strate 13). On peut aussi voir qu'il y a eu association :

```
ntpq> associations

ind assID status  conf reach auth condition  last_event cnt
===========================================================
  1 16199  9614   yes   yes  none  sys.peer   reachable  1
  2 16200  9014   yes   yes  none    reject   reachable  1

ntpq> pstatus 16199
assID=16199 status=9614 reach, conf, sel_sys.peer, 1 event, event_reach,
srcadr=fuzzer.example.org, srcport=123, dstadr=192.0.2.1, dstport=123,
leap=00, stratum=2, precision=-20, rootdelay=1.999,
rootdispersion=7.858, refid=192.0.2.176, reach=377, unreach=0,
hmode=3, pmode=4, hpoll=6, ppoll=6, flash=00 ok, keyid=0, ttl=0,
offset=0.116, delay=0.305, dispersion=3.077, jitter=0.015,
reftime=cd848978.4d74dea3  Mon, Apr  6 2009 16:00:24.302,
org=cd848a03.2ce4faea  Mon, Apr  6 2009 16:02:43.175,
rec=cd848a03.2ce6b9ee  Mon, Apr  6 2009 16:02:43.175,
xmt=cd848a03.2cd129e9  Mon, Apr  6 2009 16:02:43.175,
filtdelay=     0.31    0.32    0.32    0.32    0.32    0.31    0.37    0.34,
filtoffset=    0.13    0.13    0.13    0.13    0.13    0.12    0.15    0.12,
filtdisp=      0.00    0.99    1.94    2.93    3.90    4.88    5.85    6.80
```

Ici, il n'y avait qu'une seule vraie association, de numéro 16199, avec le serveur NTP de strate 2.

Et sur un routeur Cisco ? Configurer NTP en client est simplement :

```
Router#config terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#ntp server 129.237.32.2
Router(config)#^Z
```

Le configurer en serveur pour d'autres machines, ici en strate 10 par défaut :

```
Router#config terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#ntp master 10
Router(config)#^Z
```

On peut alors vérifier l'état de NTP :

```
Router#show ntp status
Clock is synchronized, stratum 3, reference is 128.249.2.2
nominal freq is 250.0000 Hz, actual freq is 249.9961 Hz, precision is 2**16
reference time is BF454660.7CCA9683 (22:37:36.487 EDT Sat Sep 8 2001)
clock offset is 4.3323 msec, root delay is 136.28 msec
root dispersion is 37.69 msec, peer dispersion is 1.14 msec

Router#show ntp associations

      address         ref clock     st  when  poll reach  delay  offset    disp
  *~128.249.2.2      192.5.41.40     2    4    64   377    76.9    5.49     0.4
  -~130.218.100.5    198.72.72.10    3   33   128   377     7.1   13.13     0.6
  +~129.237.32.2     192.43.244.18   2   16    64   377    44.8    3.05     0.9
  +~128.118.25.3     128.118.25.12   2   48    64   377    39.7    5.50     1.4
 * master (synced), # master (unsynced), + selected, - candidate, ~ configured
```

Tout le monde n'a pas forcément un serveur NTP chez lui, ni un fournisseur qui lui en procure un de qualité. Il est donc pratique de faire appel à des serveurs publics. Pour cela, le projet `pool.ntp.org` enregistre dans le DNS l'adresse IP des volontaires qui participent au service `ntp.pool.org`. Il suffit au client de mettre dans sa configuration :

```
server pool.ntp.org
server pool.ntp.org
server pool.ntp.org
server pool.ntp.org
```

pour se synchroniser à quatre serveurs, pris au hasard parmi les volontaires. Notez bien que le fait de répéter la même ligne quatre fois n'est pas une erreur. Chacune de ces lignes va déclencher une requête DNS qui donnera à chaque fois une liste d'adresses IP dans un ordre différent :

```
% dig +short A  pool.ntp.org
91.208.102.2
195.83.66.158
81.19.16.225
87.98.147.31
88.191.77.246

% dig +short A  pool.ntp.org
88.191.77.246
91.208.102.2
195.83.66.158
81.19.16.225
87.98.147.31

% dig +short A  pool.ntp.org
87.98.147.31
88.191.77.246
91.208.102.2
195.83.66.158
81.19.16.225
```

Pour être sûr d'avoir des adresses différentes, il suffit de préfixer les noms par un chiffre :

```
server 1.pool.ntp.org
server 2.pool.ntp.org
server 3.pool.ntp.org
server 4.pool.ntp.org
```

Vu avec `ntpq`, on pourra avoir, par exemple :

```
% ntpq
ntpq> peers
     remote           refid      st t when poll reach   delay   offset  jitter
==============================================================================
 ks34176.kimsufi 193.52.184.106   2 u  25d 1024    0    1.116   -4.039   0.000
*ns1.azuria.net  193.67.79.202    2 u  116  256  377    1.412   -1.931   0.647
+mail1.vetienne. 129.240.64.3     3 u  208  256  377    1.657  -18.063   3.348
+ntp1.adviseo.ne 209.81.9.7       2 u  114  256  377    1.001   -3.440   1.622
ntpq>
```

où des machines d'hébergeurs très différents sont utilisées, NTP choisissant le meilleur (atteignable, proche, faible gigue, etc). Bien sûr, `pool.ntp.org` n'offre aucune sécurité, puisque rien ne dit qu'un méchant ne s'est pas glissé dans le lot dans le but de perturber vos horloges. Mais c'est un service gratuit, fourni par la communauté <http://www.pool.ntp.org/join.html> et globalement de très bonne qualité. À noter que ceux qui réalisent un système d'xploitation peuvent demander un sous-domaine <http://www.pool.ntp.org/en/vendors.html>, pour y mettre par défaut leur machine. C'est ce que fait, par exemple, Debian et une machine Debian sera configurée par défaut avec quelque chose du genre :

```
server 0.debian.pool.ntp.org
server 1.debian.pool.ntp.org
server 2.debian.pool.ntp.org
server 3.debian.pool.ntp.org
```