

RFC 10049 : Roughtime: A Protocol for Rough Time Synchronization

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 7 octobre 2026

Date de publication du RFC : Octobre 2026

<https://www.bortzmeyer.org/10049.html>

Ce nouveau protocole de synchronisation d'horloges, Roughtime est, officiellement, encore expérimental. Deux de ses buts essentiels sont la possibilité d'une synchronisation sécurisée (avec de la cryptographie, donc), même quand le client n'a aucune idée de l'heure et du jour, et un moyen de signaler les erreurs et problèmes rencontrés. Il concernera notamment les objets connectés qui, souvent, n'ont pas de batterie pour garder l'heure pendant un arrêt.

Si vous connaissez NTP (RFC 5905¹), vous savez qu'il n'offre pas de solution à ces deux problèmes. Roughtime est donc une alternative à NTP.

La synchronisation des horloges est un problème très ancien sur l'Internet (cf. RFC 738). Elle est à la fois cruciale, notamment en sécurité, et très difficile à réaliser. Pour comprendre son importance en sécurité, pensez par exemple à l'estampillage de journaux. Ou aux signatures DNSSEC. Ou à TOTP (RFC 6238). Ou encore à la vérification qu'un certificat n'est pas expiré. Mais c'est justement aussi cela qui rend difficile de la sécuriser. NTP a un mécanisme de sécurité, NTS, normalisé dans le RFC 8915, mais qui dépend d'un certificat <<https://mailarchive.ietf.org/arch/msg/last-call/tCWzj0vGowUeNfC6ioO8VUtU>> donc d'avoir l'heure correcte. C'est un intéressant problème d'œuf et de poule : pour vérifier le certificat du serveur de temps, vous avez besoin de connaître l'heure, justement ce que vous vouliez demander au serveur (section 6 du RFC).

Roughtime, spécifié dans ce RFC 10049, résoud le problème en partant d'une **liste** de serveurs et de leurs clés publiques. Celles-ci sont censées durer assez longtemps pour que, dans la liste des serveurs, il

1. Pour voir le RFC de numéro NNN, <https://www.ietf.org/rfc/rfcNNN.txt>, par exemple <https://www.ietf.org/rfc/rfc5905.txt>

y en ait plusieurs qui soient correctes. En prime, l'utilisation de réponses signées par les serveurs Roughtime permet à un client de prouver à un tiers qu'un serveur est malveillant (ou complètement déconnant ; on ne peut pas distinguer les deux cas). Le client obtient cette preuve en chaînant les réponses de plusieurs serveurs. J'insiste : si on n'utilise qu'un seul serveur Roughtime, le client ne peut pas être sûr que l'heure obtenue est correcte. La sécurité ne vient que lorsqu'on a « suffisamment » de serveurs. (Évidemment, s'ils sont tous malveillants ou déconnants, on est fichu. Comme dit lors d'une discussion IETF, « *"Ideally, you would use 3 servers run by 3 different organizations using 3 different software implementations running on 3 different operating systems in 3 different well guarded buildings in 3 different countries..."* ».) Cette possibilité de prouver à un tiers qu'un des serveurs ne servait pas l'heure correcte est une importante différence avec un autre concurrent de NTP, Khronos (RFC 9523).

Ce RFC décrit le format des paquets et le protocole mais pour que Roughtime sorte de son statut expérimental, il faudra aussi déployer un écosystème de serveurs et un mécanisme pour obtenir une liste (avec les clés, souvenez-vous). Aucune méthode n'est prévue pour l'instant pour changer les clés, il faudra donc vraiment qu'elles durent longtemps, ou que l'objet connecté puisse facilement mettre à jour de manière sécurisée sa liste de serveurs.

La section 3 du RFC décrit le protocole en termes généraux. Signer les réponses n'est pas suffisant car il faut aussi que la réponse dépende de la requête du client (pour qu'il s'assure qu'on ne lui serve pas une vieille réponse rassise) et qu'on puisse la chaîner avec celle des autres serveurs (via le classique arbre de Merkle), pour qu'un tiers puisse repérer le serveur qui ne répond pas correctement. S'il n'y a qu'un serveur utilisé, le client peut tout juste vérifier que le message est correctement signé, et correspond à la question (celle-ci inclut un numnique <<https://www.bortzmeyer.org/nonce.html>>) mais s'il y en a plusieurs, le client peut les comparer et repérer un serveur qui est trop à l'écart (que ce soit par erreur ou par malveillance, en général, on ne peut pas distinguer les deux). Une fois le serveur anormal repéré, on peut le signaler sur un forum, le faire virer des listes de serveurs qui circulent, etc.

La section 4 décrit le format des messages. Parmi les points à noter, la notion d'étiquette ("*tag*"). Les valeurs transportées sont toutes étiquetées, chaque étiquette étant sur quatre octets, en général choisies pour être une valeur en ASCII. NONC va ainsi indiquer que la valeur qui suit est le numnique ("*nonce*"), VER (avec un octet nul à la fin) indique la version (actuellement version 1 <<https://www.iana.org/assignments/roughtime#roughtime-versions>>), etc. Les étiquettes connues figurent dans un registre IANA <<https://www.iana.org/assignments/roughtime#roughtime-tags>> et on peut en ajouter selon la politique « Spécification nécessaire » du RFC 8126.

Autre notion cruciale, les valeurs temporelles ("*timestamp*") sont un entier sur 64 bits, indiquant le nombre de secondes depuis le 1 janvier 1970. Le message complet est composé d'un nombre magique (0x4d49544847554f52, ce qui fait "ROUGHTIM"), du nombre de valeurs, d'une liste de décalages des valeurs, d'une liste d'étiquettes, puis des valeurs. Les deux listes sont triées selon l'ordre des étiquettes. La première valeur a pour décalage zéro.

Le message est ensuite transmis en UDP (avec du remplissage obligatoire pour éviter les attaques avec amplification, section 9.7 du RFC). L'algorithme cryptographique est forcément Ed25519, décrit dans le RFC 8032. Il n'y a pas d'agilité cryptographique - RFC 7696, il faudra changer de version du protocole pour, par exemple, passer à la cryptographie post-quantique (section 9.5). Dans une requête, les étiquettes VER (version du protocole), NONC (le numnique du client, qui doit évidemment être aussi imprédictible que possible, cf. RFC 4086) et TYPE (0 pour une requête) sont obligatoires. On peut y ajouter SRV, qui indique la clé publique que le client compte utiliser (cela permet au serveur, entre autres, de sélectionner la bonne clé si on en a plusieurs, mais aussi de voir qu'un client a toujours une vieille clé).

La réponse a les étiquettes NONC (celui qui avait été envoyé par le client), TYPE (1 pour une réponse), SIG (la signature du message), PATH (des valeurs de l'arbre de Merkle), CERT (un certificat)

et SREP (la partie utile de la réponse). La valeur associée à l'étiquette SREP est elle-même un message Roughtime avec notamment les étiquettes MIDP (l'heure sur le serveur, c'est-à-dire l'information principale pour le client), RADI (une estimation de la précision de l'heure, en secondes) et ROOT (la racine de l'arbre de Merkle, qui sera utilisé si on interroge plusieurs serveurs, cf. section 5.3).

On peut noter qu'il n'existe pas de réponse pour une requête erronée (pas de FORMERR comme dans le DNS ou de *"Kiss o' Death"* comme dans NTP). L'expérience de NTP (section 7.4 du RFC 5905, section 5.4 du RFC 8633 et section 8.3 et 8.7 du RFC 8915) est que cela facilite trop certaines attaques par déni de service en faisant croire qu'un serveur a un problème.

Pour éviter l'ossification du protocole, la section 7 du RFC demande de graisser (RFC 9170) les réponses en envoyant de temps en temps des réponses invalides (absence d'étiquettes obligatoires, par exemple, présence d'étiquettes non définies et surtout heures délibérément incorrectes mais avec des signatures invalides; un client bien fait ignorera ces réponses, les autres se feront tromper).

Côté client, maintenant, le protocole prévoit que le client vérifie les signatures et signale (par un moyen non normalisé) les serveurs qui s'écarteraient trop des autres. La section 8.4 décrit un format en JSON pour produire des rapports, au type `application/roughtime-malfeasance+json` (avec un exemple dans l'annexe B) mais, pour l'instant, il n'y a pas grand'chose de spécifié et encore moins de déployé.

Notez que, si les réponses sont signées, Roughtime ne fournit en revanche aucune confidentialité.

Sur ma machine au logiciel pas tout à fait récent, Wireshark ne connaît pas encore Roughtime mais, quand on analyse un paquet, on voit bien la liste des étiquettes :

Frame 10: 434 bytes on wire (3472 bits), 434 bytes captured (3472 bits)

...

User Datagram Protocol, Src Port: 2002, Dst Port: 47639

Data (392 bytes)

```

0000  52 4f 55 47 48 54 49 4d 7c 01 00 00 07 00 00 00  ROUGHTIM|.....
0010  40 00 00 00 44 00 00 00 64 00 00 00 64 00 00 00  @...D...d...d...
0020  a8 00 00 00 40 01 00 00 53 49 47 00 56 45 52 00  ...@...SIG.VER.
0030  4e 4f 4e 43 50 41 54 48 53 52 45 50 43 45 52 54  NONCPATHSREPCERT
0040  49 4e 44 58 73 d4 d3 56 32 f7 d1 03 28 31 35 bf  INDXs..V2...(15.
0050  18 14 ea 6e 62 74 99 58 ca b2 85 bf 1f 0b ca a1  ...nbt.X.....
0060  4e 06 1c 4a ea 0b e0 9c 93 12 5b 7d 79 23 16 c3  N..J.....[y#..
0070  32 d9 41 d9 b0 94 5b 68 c0 e3 f1 c3 61 ab 00 6f  2.A...[h....a..o
0080  58 a8 d9 0c 0b 00 00 80 ac 90 f3 d2 69 b9 f8 d6  X.....i...
0090  7a 0d b7 37 42 e3 27 ab ed e3 fb d2 e8 41 f9 25  z..7B.'.....A.%
00a0  75 de be 60 0a d6 59 80 03 00 00 00 04 00 00 00  u..'..Y.....
00b0  0c 00 00 00 52 41 44 49 4d 49 44 50 52 4f 4f 54  ....RADIMIDPROOT
00c0  03 00 00 00 3c 2c c6 6a 00 00 00 00 28 68 0c 3e  ....<,.j....(h.>
00d0  99 9c 8d c3 1c 86 9b 1f b9 05 4b 3d a5 92 0d 4c  .....K=...L
00e0  48 4f a0 7c 16 ae e4 1e 0b 24 2e d1 02 00 00 00  HO.|.....$.
00f0  40 00 00 00 53 49 47 00 44 45 4c 45 cc 05 f4 6c  @...SIG.DELE...l
0100  56 46 12 eb 4d 45 47 c9 77 21 f4 9b 78 8f 46 2c  VF..MEG.w!..x.F,
0110  fa e1 08 8b 49 40 2e 70 e2 c3 15 f1 fe 2e 50 ab  ....I@.p.....P.
0120  00 ab 77 18 3e f2 e0 56 7b 41 9a 39 78 de 4a e4  ..w.>..V{A.9x.J.
0130  aa 8d bf c8 79 b3 f9 86 75 8a 18 0f 03 00 00 00  ....y...u.....
0140  20 00 00 00 28 00 00 00 50 55 42 4b 4d 49 4e 54  ...(...PUBKMINT
0150  4d 41 58 54 b4 f2 07 3c f9 a5 f4 f0 d2 e5 03 51  MAXT...<.....Q
0160  1a cb b0 31 2f e3 cd 83 b7 14 ef 64 eb bb 9c 3b  ...l/.....d...;
0170  4c 69 3a ea dc 8e c5 6a 00 00 00 00 5c e0 c6 6a  Li:....j....\..j
0180  00 00 00 00 00 00 00 00  ....
Data [truncated]: 524f55474854494d7c01000007000000400000006400000064000000a8000000400100005349470056
2455043455254494e445873d4d35632f7d103283135bf1814ea6e62749958cab285bf1f0bcaa14e061c4aea0be09c93125b7d79231
[Length: 392]

```

Voyons maintenant les mises en œuvre concrètes de ce protocole, côté client et côté serveur. D’abord, un avertissement : le RFC vient de sortir, le projet de norme a eu plusieurs itérations et vous trouverez donc en ligne divers programmes qui n’interagissent pas toujours entre eux (par exemple, le logiciel client de Cloudflare ne fonctionne pas avec le serveur public de Cloudflare <<https://github.com/cloudflare/roughtime/issues/72>>). Il faudra donc attendre un peu que tout soit d’équerre. En attendant, jouons un peu. D’abord, le logiciel de Cloudflare <<https://github.com/cloudflare/roughtime>>, écrit en Go (vous pouvez aussi lire leur documentation <<https://developers.cloudflare.com/time-services/roughtime/>>) :

```
% git clone https://github.com/cloudflare/roughtime.git
% cd roughtime
% cd cmd/getroughtime
% go build main.go
% ./main
Either provide a configuration via -config or an address via -ping
```

Ah, c’est logique, rappelez-vous ce que j’ai écrit sur la liste de serveurs (et leurs clés, cf. sections 8.1 et 8.3 du RFC). getroughtime sait utiliser une liste écrite en JSON. Le RFC décrit le format dans sa section 8.3, et on utilise le type `application/roughtime-server+json` si on la distribue en ligne. Un exemple figure dans l’annexe A. Il y a une liste fournie dans le logiciel, utilisons-la :

```
% ./main -config ../../ecosystem.json
skipped Cloudflare-Roughtime-2: protocol: response is missing NONC tag
skipped int08h-Roughtime: no reply
skipped roughtime.se: no reply
time.txryan.com: 2026-10-06 09:36:47 +0000 UTC ±3s (in 96ms)
Delta: -741ms
```

Bon, plusieurs des serveurs ne répondent pas, ou pas correctement selon les clients (ma remarque ci-dessus à propos des différentes versions qui traînent). Mais le dernier marche, et on a bien l’heure. Le même logiciel contient aussi un serveur de test :

```
% cd ../testserver
% go build main.go
% ./main
main.go:64: Root public key: zFVzPprRDxDOMdsrifp+/XPkVkDEhllv6QDJxOWWYd0=
```

Le serveur tourne (par défaut, sur 127.0.0.1:2002). Notez que le port officiellement enregistré à l’IANA <<https://www.iana.org/assignments/service-names-port-numbers?search=1&page=44>> est désormais 5319. Et le serveur nous affiche sa clé. Essayons en indiquant cette clé :

```
% cd ../getroughtime
% ./main -ping localhost:2002 -pubkey zFVzPprRDxDOMdsrifp+/XPkVkDEhllv6QDJxOWWYd0=
Ping response: 2026-10-06 11:37:57.754243 +0200 CEST ±1s (in 1ms)
```

Parfait, tout va bien. Si on avait indiqué une mauvaise clé :

<https://www.bortzmeyer.org/10049.html>