Équipe d'ingénierie de l'Internet (IETF)
Request for Comments : 5905
RFC rendues obsolètes : 1305, 4330
Catégorie : Sur la voie de la normalisation
ISSN : 2070-1721
Traduction Claude Brière de L'Isle

D. Mills, ISC
U. Delaware, ISC
J. Martin, éditeur, ISC
J. Burbank & W. Kasch, JHU/APL
juin 2010

Protocole de l'heure du réseau version 4 : spécification du protocole et des algorithmes

Résumé
Le protocole de l'heure du réseau (NTP, *Network Time Protocol*) est largement utilisé pour synchroniser les horloges des ordinateurs dans l'Internet. Le présent document décrit la version 4 de NTP (NTPv4) qui est rétro compatible avec NTP version 3 (NTPv3) décrit dans la [RFC1305], ainsi qu'avec les précédentes versions du protocole. NTPv4 inclut un en-tête de protocole modifié pour s'accommoder de la famille d'adresse du protocole Internet version 6. NTPv4 inclut des améliorations fondamentales dans les algorithmes d'atténuation et de discipline qui étendent la précision potentielle à la dizaine de microsecondes des stations de travail modernes et des LAN rapides. Il inclut un schéma de découverte dynamique de serveur, de sorte que dans de nombreux cas, une configuration spécifique de serveur n'est pas exigée. Il corrige certaines erreurs de la conception et de la mise en œuvre de NTPv3 et inclut un mécanisme d'extension facultatif.

Table des matières

1. Introduction

Le présent document définit la version 4 du protocole de l'heure du réseau (NTP4, *Network Time Protocol version 4*) qui est largement utilisée pour synchroniser les horloges système parmi un ensemble de serveurs et clients horaires répartis. Il décrit le cœur de l'architecture, le protocole, les automates à états, les structures de données, et les algorithmes. NTPv4 introduit une nouvelle fonctionnalité à NTPv3, comme décrit dans la [RFC1305], et une fonctionnalité étendue à partir de la version 4 du protocole simple de l'heure du réseau (SNTPv4, *Simple NTP version 4*) comme décrit dans la [RFC4330] (SNTPv4 est un sous ensemble de NTPv4). Le présent document rend obsolètes les [RFC1305] et [RFC4330]. Bien que certains changements mineurs aient été faits dans certains champs d'en-tête de protocole, ceux-ci n'affectent pas l'interopérabilité entre NTPv4 et les précédentes versions de NTP et SNTP.

Le modèle de sous réseau NTP comporte un certain nombre de serveurs horaires principaux largement accessibles synchronisés par le réseau filaire ou radio aux normes nationales. L'objet du protocole NTP est de convoyer les informations de conservation de l'heure à partir de ces serveurs principaux jusqu'aux serveurs secondaires et clients horaires via à la fois des réseaux privés et l'Internet public. Des réglages fins des algorithmes atténuent les erreurs qui peuvent résulter des interruptions du réseau, des défaillances de serveur, et de possibles actions hostiles. Les serveurs et clients sont configurés de telle sorte que les valeurs coulent vers les clients depuis les serveurs principaux à la racine via des embranchements de serveurs secondaires.

La conception de NTPv4 résout des insuffisances significatives de la conception de NTPv3, corrige certaines erreurs, et incorpore de nouvelles caractéristiques. En particulier, les définitions d'horodatage NTP étendu encouragent l'utilisation du type de données à doubles décimales flottantes tout au long de la mise en œuvre. Par suite, la résolution de l'heure est meilleure que une nanoseconde, et la résolution de fréquence est de moins d'une nanoseconde par seconde. Les autres améliorations incluent un nouvel algorithme de discipline d'horloge qui a une meilleure réponse aux fluctuations de fréquence des matériels d'horloge système. Les serveurs principaux normaux qui utilisent des machines modernes sont précis à quelques dixièmes de microsecondes près. Les serveurs secondaires et les clients normaux sur les LAN rapides qui sont précis à quelques centièmes de microsecondes avec les intervalles d'interrogation jusqu'à 1024 secondes, qui était le

maximum avec NTPv3. Avec NTPv4, les serveurs et clients sont précis à quelques dixièmes de millisecondes avec des intervalles d'interrogation jusqu'à 36 heures.

Le corps principal de ce document décrit le cœur du protocole et les structures de données nécessaires pour l'interopération entre les mises en œuvre conformes. L'Appendice A contient un exemple pleinement détaillé sur la forme d'un squelette de programme, incluant les structures de données et les segments de code pour les algorithmes principaux ainsi que les algorithmes d'atténuation utilisés pour améliorer la fiabilité et la précision. Bien que le squelette de programme et les autres descriptions de ce document s'appliquent à une mise en œuvre particulière, ils ne sont pas destinés à être le seul moyen par lequel les fonctions exigées peuvent être mises en œuvre. Le contenu de l'Appendice A sont des exemples non normatifs conçus pour illustrer le fonctionnement du protocole et ce n'est pas une exigence de conformité de mise en œuvre. Alors que le schéma d'authentification de clé symétrique NTPv3 décrit dans ce document a été importé de NTPv3, le schéma d'authentification de clé publique Autokey qui est nouveau pour NTPv4 est décrit dans la [RFC5906].

Le protocole NTP inclut des modes de fonctionnement décrits dans la Section 2 en utilisant des types de données décrits dans la Section 6 et des structures de données décrites dans la Section 7. Le modèle de mise en œuvre décrit à la Section 5 se fonde sur une architecture de tissu multi processus, bien que d'autres architectures puissent aussi être utilisées. Le protocole de réseau décrit à la Section 8 se fonde sur un concept de temps réversible qui dépend seulement des décalages d'horloge mesurés, mais n'exige pas de livraison fiable du message. La livraison fiable du message comme avec TCP [RFC0793] peut en fait rendre la livraison du paquet NTP moins fiable car de nouveaux essais vont augmenter la valeur du délai et d'autres erreurs. Le sous réseau de synchronisation est un réseau maître/esclave auto organisé, hiérarchique, avec des chemins de synchronisation déterminés par un arbre d'expansion de plus court chemin et une métrique définie. Bien que plusieurs maîtres (serveurs principaux) puissent exister, ce n'est pas exigé pour le choix d'un protocole.

Le présent document inclut des matériaux provenant de [ref9], qui contient des diagrammes et des équations qui ne conviennent pas pour le format de RFC. Il y a beaucoup d'informations supplémentaires dans [ref7], incluant une analyse technique extensive et des mesures de performances du protocole et des algorithmes de ce document. La mise en œuvre de référence est disponible à www.ntp.org.

Le reste de ce document contient de nombreuses variables et expressions mathématiques. Certaines variables prennent la forme de caractères grecs, qui sont épelés par leur nom complet sensible à la casse. Par exemple, DELTA se réfère au caractère grec en majuscule, tandis que delta se réfère au caractère minuscule. De plus, les soulignés sont notés avec '_' ; par exemple, theta_i se réfère au caractère grec minuscule thêta avec i souligné, ou phonétiquement thêta i souligné. Dans ce document, toutes les valeurs de temps sont en secondes (s), et toutes les fréquences vont être spécifiées comme des décalages de fréquence fractionnaires (FFO, *fractional frequency offset*) (nombre pur). Il est souvent pratique d'exprimer ces FFO en parties par million (ppm).

1.1 Notation des exigences

Les mots clés "DOIT", "NE DOIT PAS", "EXIGE", "DEVRA", "NE DEVRA PAS", "DEVRAIT", "NE DEVRAIT PAS", "RECOMMANDE", "PEUT", et "FACULTATIF" dans ce document sont à interpréter comme décrit dans la [RFC2119].

2. Modes de fonctionnement

Une mise en œuvre NTP fonctionne comme un serveur principal, un serveur secondaire , ou un client. Un serveur principal est synchronisé à une horloge de référence directement traçable à l'UTC (par exemple, GPS, Galileo, etc.). Un client se synchronise avec un ou plusieurs serveurs amont, mais ne fournit pas la synchronisation aux clients dépendants. Un serveur secondaire a un ou plusieurs serveurs amont et un ou plusieurs serveurs ou clients vers l'aval.
Tous les serveurs et clients qui sont pleinement conformes à NTPv4 DOIVENT mettre en œuvre la suite entière des algorithmes décrits dans le présent document. Afin de conserver la stabilité dans les grands sous réseaux NTP, les serveurs secondaires DEVRAIENT être pleinement conformes à NTPv4. Des algorithmes de remplacement PEUVENT être utilisés, mais leur résultat DOIT être identique à celui des algorithmes décrits dans la présente spécification.

3. Modes de protocoles

Il y a trois variantes du protocole NTP : symétrique, client/serveur, et diffusion. Chacune est associée à un mode d'association (une description de la relation entre deux locuteurs NTP) comme le montre la Figure 1. De plus, les associations persistantes sont mobilisées au démarrage et ne sont jamais démobilisées. Les associations éphémères sont mobilisées à l'arrivée d'un paquet et sont démobilisées sur une erreur ou une fin de temporisation.

Mode d'association	Valeur de mode d'assoc.	Valeur de mode de paquet
Symétrique actif	1	1 ou 2
Symétrique passif	2	1
Client	3	4
Serveur	4	3
Serveur de diffusion	5	5
Client de diffusion	6	N/A

Figure 1 : Modes d'association et de paquet

Dans la variante client/serveur, un client persistent envoie des paquets en mode de paquet 4 à un serveur, qui retourne des paquets en mode paquet 3. Les serveurs fournissent la synchronisation à un ou plusieurs clients, mais n'acceptent pas d'eux la synchronisation. Un serveur peut aussi être un pilote d'horloge de référence qui obtient l'heure directement d'une source standard comme un receveur GPS ou un service de modem téléphonique. Dans cette variante, les clients tirent la synchronisation des serveurs.

Dans la variante symétrique, un homologue fonctionne à la fois comme serveur et comme client en utilisant une association symétrique active ou passive. Une association persistante symétrique active envoie des paquets symétriques actifs (mode 1) à une association symétrique active homologue. Autrement, une association éphémère symétrique passive peut être mobilisée à l'arrivée d'un paquet symétrique actif sans association correspondante. Cette association envoie des paquets symétriques passifs (mode 2) et persiste jusqu'à une erreur ou une fin de temporisation. Les homologues poussent et tirent tous deux la synchronisation de et vers chacun d'eux. Pour les besoins du présent document, un homologue opère comme un client, de sorte que les références à un client impliquent aussi l'homologue.

Dans la variante diffusion, une association de serveur de diffusion persistante envoie des paquets périodiques de serveur de diffusion (mode 5) qui peuvent être reçus par plusieurs clients. À réception d'un paquet de serveur de diffusion sans une association correspondante, une association de client de diffusion éphémère (mode 6) est mobilisée et persiste jusqu'à une erreur ou une fin de temporisation. Il est utile de fournir une salve initiale où le client fonctionnant en mode client échange plusieurs paquets avec le serveur, afin de calibrer le délai de propagation et de lancer le protocole de sécurité Autokey, après quoi le client revient au mode client de diffusion. Un serveur de diffusion pousse la synchronisation jusqu'aux clients et aux autres serveurs.

En suivant à peu près les conventions établies par l'industrie du téléphone, le niveau de chaque serveur dans la hiérarchie est défini par un numéro de strate. Les serveurs principaux sont à la couche un, les serveurs secondaires à chaque niveau inférieur reçoivent un numéro de strate supérieur de un au niveau précédant. Lorsque le numéro de strate augmente, sa précision se dégrade selon le chemin réseau particulier et la stabilité de l'horloge système. Les erreurs moyennes, mesurées par les distances de synchronisation, augmentent approximativement en proportion du numéro de strate et du délai d'aller retour mesuré.

Dans la pratique, la topologie de l'heure du réseau devrait être organisée de façon à éviter des boucles de temps et minimiser la distance de synchronisation. Dans NTP, la topologie de sous réseau est déterminée en utilisant une variante de l'algorithme d'acheminement réparti de Bellman-Ford, qui calcule l'arbre d'expansion de plus court chemin dont les racines sont les serveurs principaux. Par suite de cette conception, l'algorithme réorganise automatiquement le sous réseau, de façon à produire l'heure la plus précise et fiable, même quand il y a des défaillances dans le réseau de l'heure.

3.1 Découverte dynamique de serveur

Il y a deux associations spéciales, le client multi envois et le serveur multi envoi,s qui assurent une fonction de découverte dynamique de serveur. Il y a deux types d'associations de client multi envois : persistent et éphémère. Le client multi envois persistent envoie des paquets de client (mode 3) à une adresse IPv4 ou IPv6 désignée de diffusion ou de diffusion groupée. Les routeurs désignés de multi envois dans la gamme du champ Durée de vie (TTL, *time-to-live*) dans l'en-tête de paquet écoutent pour les paquets qui ont cette adresse. Si un serveur convient pour la synchronisation, il retourne un paquet de serveur ordinaire (mode 4) en utilisant l'adresse d'envoi individuel du client. À réception de ce paquet, le client mobilise une association éphémère de client (mode 3). L'association éphémère de client persiste jusqu'à une erreur ou une fin de temporisation.

Un client de multi envois continue d'envoyer des paquets pour chercher un nombre minimum d'associations. Il commence avec un TTL égal à un et lui ajoute continûment un jusqu'à ce que le nombre minimum d'associations soit atteint ou que le TTL atteigne la valeur maximum. Si le TTL atteint sa valeur maximum et si il n'y a pas encore assez d'associations mobilisées, le client arrête la transmission pendant une période déterminée pour libérer toutes les associations, et répète

alors le cycle de recherche. Si un nombre minimum d'associations a été mobilisé, le client commence alors à transmettre un paquet par période de temporisation pour maintenir les associations. Les contraintes de champ limitent la valeur minimum à 1 et le maximum à 255. Ces limites peuvent être réglées en fonction des besoins des applications individuelles.

Les associations éphémères sont en compétition entre elles. Lorsque de nouvelles associations éphémères sont mobilisées, le client lance les algorithmes d'atténuation décrits à la Section 10 et au paragraphe 11.2 pour trouver les meilleurs candidats parmi la population, les associations éphémères restantes arrivent en fin de temporisation et sont démobilisées. De cette façon, la population inclut seulement les meilleurs candidats qui ont répondu le plus récemment avec un paquet NTP pour discipliner l'horloge système.

4. Définitions

Un certain nombre de termes techniques sont définis dans cette section. Une échelle de temps est un cadre de référence où le temps est exprimé comme la valeur d'un compteur binaire à accroissement monotone avec un nombre de bits indéfini. Il compte en secondes et fractions de seconde, quand une virgule décimale est employée. L'échelle de temps du temps universel coordonné (UTC, *Coordinated Universal Time*) est définie par la Recommandation UIT-R TF.460 [TF.460]. Sous les auspices de la Convention du mètre de 1865, en 1975, la CGPM [CGPM] a fortement recommandé l'utilisation de l'UTC comme base de l'heure civile.

L'échelle de temps du temps universel coordonné (UTC) représente un temps solaire moyen comme diffusé par les laboratoires des normes nationales. Le système horaire est représenté par l'horloge système maintenue par le matériel et le système d'exploitation. Le but des algorithmes NTP est de minimiser à la fois la différence horaire et la différence de fréquence entre l'UTC et l'horloge système. Quand ces différences ont été réduites en dessous des tolérances nominales, l'horloge système est dite être synchronisée à l'UTC.

La date d'un événement est le temps UTC auquel l'événement a lieu. Les dates sont des valeurs éphémères désignées par T en majuscule. Le temps courant est une autre échelle de temps qui est coïncidente à la fonction de synchronisation du programme de NTP.

Un horodatage T(t) représente la date UTC ou le décalage horaire par rapport à l'UTC à l'instant courant t. Ce qui est signifié devrait être clair d'après le contexte. Soit T(t) le décalage horaire, R(t) le décalage de fréquence, et D(t) le taux de vieillissement (dérivée première de R(t) par rapport à t). Alors, si T(t_0) est le décalage de temps UTC déterminé à t = t_0, le décalage de l'heure UTC à l'instant t est T(t) = T(t_0) + R(t_0)(t-t_0) + 1/2 * D(t_0)(t-t_0)^2 + e, où e est un terme d'erreur stochastique discuté plus loin dans le présent document. Alors que le terme D(t) est important pour caractériser des oscillateurs de précision, il est ordinairement négligé pour les oscillateurs des ordinateurs. Dans le présent document, toutes les valeurs de temps sont en secondes (s) et toutes les valeurs de fréquence sont en secondes par seconde (s/s). Il est parfois pratique d'exprimer les décalages de fréquence en parties par million (ppm), où 1 ppm est égal à 10^(-6) s/s.

Il est important dans les applications informatiques de conservation de l'heure de vérifier les performances de la fonction de conservation de l'heure. Le modèle de performances de NTP inclut quatre statistiques qui sont mises à jour chaque fois qu'un client fait une mesure avec un serveur. Le décalage (thêta) représente le décalage horaire vraisemblable maximum de l'horloge du serveur par rapport à l'horloge système. Le délai (delta) représente le délai d'aller-retour entre le client et le serveur. La dispersion (epsilon) représente l'erreur maximum inhérente à la mesure. Elle augmente à un taux égal à la tolérance maximum de la fréquence de l'horloge système disciplinée (PHI), normalement 15 ppm. La gigue (psi) est définie comme la moyenne de l'écart quadratique moyen (RMS, *root-mean-square*) des plus récentes différences de décalage, et elle représente l'erreur nominale de l'estimation du décalage.

Bien que les statistiques de thêta, delta, epsilon, et psi représentent des mesures de l'horloge système par rapport à chaque horloge de serveur séparément, le protocole NTP inclut des mécanismes pour combiner les statistiques de plusieurs serveurs pour discipliner et calibrer plus précisément l'horloge système. Le décalage du système (THETA) représente le décalage maximum vraisemblable estimé pour la population de serveurs. La gigue de système (PSI) représente l'erreur nominale de l'estimation du décalage du système. Les statistiques de delta et epsilon sont accumulées à chaque niveau de strate pour l'horloge de référence pour produire les statistiques de délai racine (DELTA) et de dispersion racine (EPSILON). La distance de synchronisation (LAMBDA) égale à EPSILON + DELTA / 2 représente l'erreur maximum due à toutes causes. La formulation détaillée de ces statistiques est donnée au paragraphe 11.2. Elles sont disponibles aux applications dépendantes afin de vérifier les performances de la fonction de synchronisation.

5. Modèle de mise en œuvre

La Figure 2 montre l'architecture d'une mise en œuvre multi trames typique. Elle inclut deux processus dédiés à chaque serveur, un processus d'homologue pour recevoir les messages provenant du serveur ou de l'horloge de référence, et un processus d'interrogation pour transmettre les messages au serveur ou à l'horloge de référence.

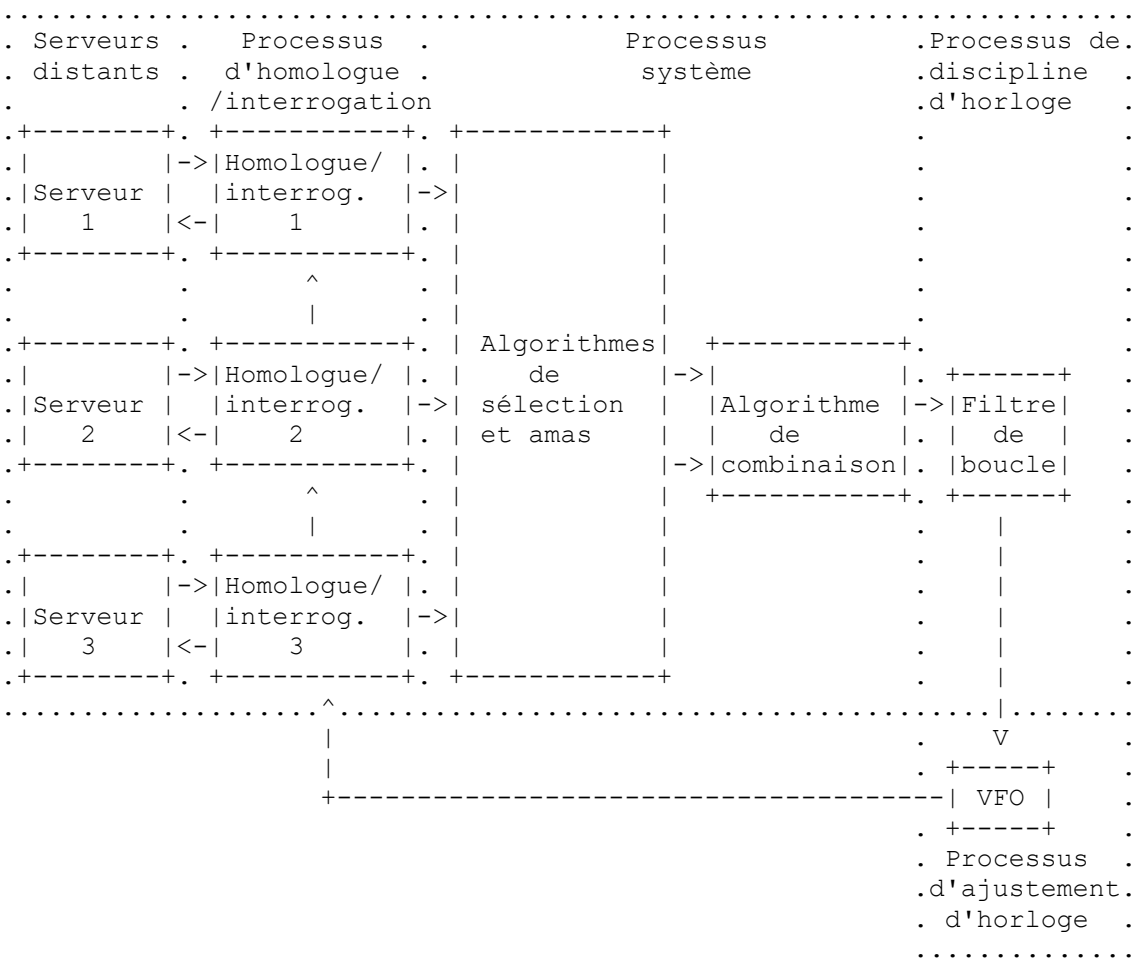


Figure 2 : Modèle de mise en œuvre

Ces processus opèrent sur une structure de données commune, appelée une association, qui contient les statistiques décrites plus haut avec diverses autres données décrites à la Section 9. Un client envoie des paquets à un ou plusieurs serveurs et ensuite traite les paquets retournés quand ils sont reçus. Le serveur échange les adresses et accès de source et de destination, écrase certain champs dans le paquet et le retourne immédiatement (dans le mode client/serveur) ou un moment après (dans les modes symétriques). Lorsque chaque message NTP est reçu, le décalage thêta entre l'horloge de l'homologue et l'horloge système est calculé ainsi que les statistiques de delta, epsilon, et psi associées.

Le processus système inclut les algorithmes de sélection, d'amas, et de combinaison qui font un amalgame des divers serveurs et horloges de référence pour déterminer les candidats les plus précis et fiables pour synchroniser l'horloge système. L'algorithme de sélection utilise les principes de détection de faute byzantins pour éliminer les candidats présumés incorrects appelés des "faux tiqueurs" *(falsetickers)* de la population incidente, laissant seulement les bons candidats appelés des "vrais sonneurs" *(truechimers)*. Un vrai sonneur est une horloge qui conserve la précision de l'heure selon une norme préalablement publiée et de confiance, tandis qu'un faux tiqueur est une horloge qui donne une heure trompeuse ou incohérente. L'algorithme d'amas utilise les principes statistiques pour trouver l'ensemble le plus précis de vrais sonneurs. L'algorithme de combinaison calcule le décalage d'horloge final en faisant la moyenne statistique des vrais sonneurs survivants.

Le processus de discipline d'horloge est un processus système qui contrôle l'heure et la fréquence de l'horloge système, représentée ici comme un oscillateur à fréquence variable (VFO, *variable frequency oscillator*). Les horodatages tirés du VFO terminent la boucle de retours qui maintient l'heure de l'horloge système. Associé au processus de discipline d'horloge

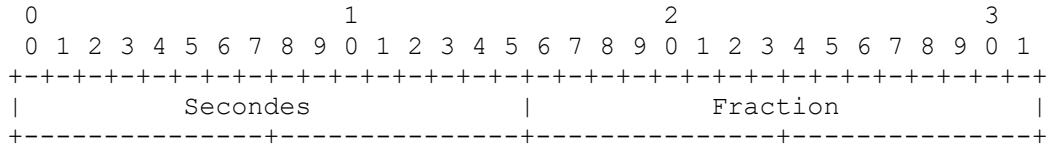
est le processus d'ajustement d'horloge, qui fonctionne une fois par seconde pour injecter un décalage de temps calculé et maintenir une fréquence constante. Le RMS moyen des différences de décalage de temps passé représente l'erreur nominale ou gigue de l'horloge système. L'écart quadratique moyen (RMS) des différences passées de décalage de fréquences représente la stabilité de fréquence ou le dérapage de fréquence de l'oscillateur. Ces termes reçoivent une interprétation précise au paragraphe 11.3.

Un client envoie des messages à chaque serveur avec un intervalle d'interrogation de 2^tau secondes, comme déterminé par l'exposant tau d'interrogation. Dans NTPv4, tau va de 4 (16 s) à 17 (36 h). La valeur de tau est déterminée par l'algorithme de discipline d'horloge comme correspondant à la constante de temps de boucle T_c = 2^tau. En mode client/serveur, le serveur répond immédiatement ; cependant, dans les modes symétriques, chacun des deux homologues gère tau comme une fonction du décalage et de la gigue du système actuel, de sorte qu'ils peuvent ne pas s'accorder sur la même valeur. Il est important que le comportement dynamique de l'algorithme de discipline d'horloge soit étroitement contrôlé afin de maintenir la stabilité dans le sous réseau NTP d'ensemble. Cela exige que les homologues s'accordent sur un tau commun égal à l'exposant d'interrogation minimal des deux homologues. Le protocole NTP inclut des dispositions pour négocier correctement cette valeur.

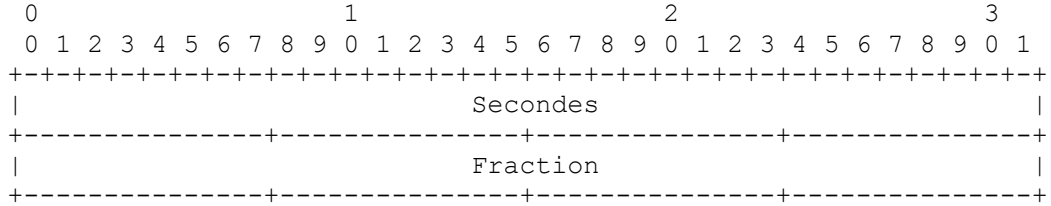
Le modèle de mise en œuvre inclut des moyens pour régler et ajuster l'horloge système. Le système d'exploitation est supposé assurer deux fonctions : une pour régler l'heure directement, par exemple, la fonction Unix settimeofday(), et une autre pour ajuster l'heure par petits incréments avançant ou retardant l'heure d'une quantité fixée, par exemple, la fonction Unix adjtime(). Ici et dans les références qui suivent, les parenthèses à la suite d'un nom indiquent une référence à une fonction plutôt qu'une simple variable. Dans la conception prévue, le processus de discipline d'horloge utilise la fonction adjtime() si l'ajustement est inférieur à un seuil fixé, et la fonction settimeofday() si il est au dessus du seuil. La manière de le faire et la valeur du seuil sont décrites à la Section 10.

6. Types de données

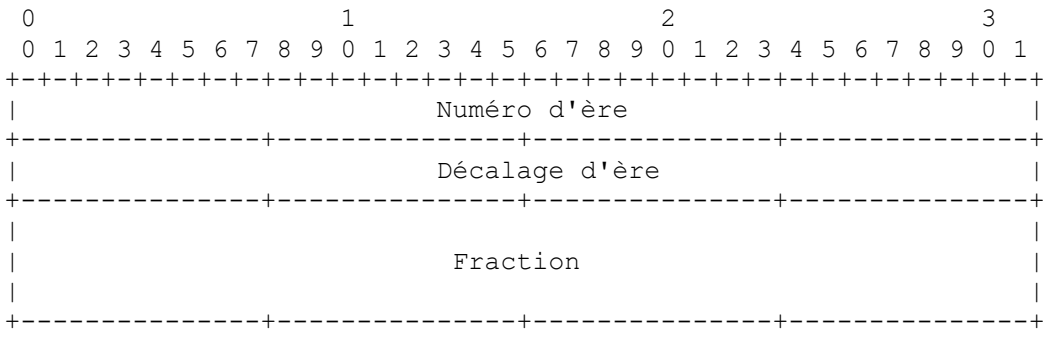
Toutes les valeurs de temps de NTP sont représentées dans un format de complément à deux, les bits étant numérotés dans l'ordre gros boutien (comme décrit dans l'Appendice A de la [RFC0791]) avec le zéro commençant à la position de gauche, ou position de poids fort. Il y a trois formats de temps NTP, un format de date de 128 bits, un format d'horodatage de 64 bits, et un format court de 32 bits, comme le montre la Figure 3. Le format de date de 128 bits est utilisé lorsque une mémorisation et une taille de mot suffisantes sont disponibles. Il inclut un champ de secondes de 64 bits signé s'étendant sur 584 milliards d'années et un champ de fraction de seconde de 64 bit d'une résolution de 0,05 attoseconde (c'est-à-dire, 0,5e-18). Pour faciliter la transposition entre les formats, le champ des secondes est divisé en un champ de 32 bits de numéro d'ère et un champ de 32 bits de décalage d'ère. Les ères ne peuvent pas être produites directement par NTP, et il n'est pas nécessaires qu'elles le soit. Lorsque nécessaire, elles peuvent être déduites de moyens externes, comme le système de fichiers ou d'un matériel dédié.



Format NTP court



Format d'horodatage NTP



Format de date NTP

Figure 3 : Formats du temps NTP

Le format d'horodatage de 64 bits est utilisé dans les en-têtes de paquet et autres lieux où la taille de mot est limitée. Il inclut un champ de 32 bits non signé de secondes s'étendant sur 136 ans et un champ de fractions de 32 bits d'une résolution de 232 picosecondes. Le format court de 32 bits est utilisé dans les champs d'en-tête de délai et de dispersion où la pleine résolution et gamme des autres formats ne sont pas justifiées. Il inclut un champ de 16 bits non signé de secondes et un champ de fraction de seconde de 16 bits.

Dans les formats de date et d'horodatage, la première époque, ou date de base de l'ère 0, est à 0 h le 1er janvier 1900 UTC, quand tous les bits sont à zéro. On devrait noter que strictement parlant, l'UTC n'existait pas avant le 1er janvier 1972, mais il est pratique de supposer qu'il a existé de toute éternité, même si toutes les connaissances de l'historique des sauts de secondes ont été perdues. Les dates se rapportent à l'époque première ; les valeurs supérieures à zéro représentent les temps après cette date ; les valeurs inférieures à zéro représentent les temps antérieurs. Noter que le champ Décalage d'ère du format de date et le champ Secondes du format d'horodatage ont la même interprétation.

Les horodatages sont des valeurs non signées, et les opérations sur eux produisent un résultat dans la même ère ou une ère adjacente. L'ère 0 inclut les dates à partir de la première époque jusqu'à environ 2036, quand l'horodatage revient à zéro et que la date de base pour l'ère 1 est établie. Dans tous les formats, une valeur de zéro est un cas particulier représentant un temps inconnu ou non synchronisé. La Figure 4 montre un certain nombre de dates NTP historiques avec leur date dans le calendrier Julien modifié (MJD, *Modified Julian Day*), l'ère NTP, et l'horodatage NTP correspondants.

Date	**MJD**	**Ère NTP**	**Décalage d'ère de l'horodatage NTP**	**Époque**
1 jan -4712	-2 400 001	-49	1 795 583 104	1er jour julien
1 jan -1	-679 306	-14	139 775 744	an 2 avant l'ère chrétienne
1 jan 0	-678 491	-14	171 311 744	an 1 avant l'ère chrétienne
1 jan 1	-678 575	-14	202 939 144	1er jour de l'ère chrétienne
4 oct 1582	-100 851	-3	2 873 647 488	dernier jour julien
15 oct 1582	-100 840	-3	2 874 597 888	premier jour grégorien
31 déc 1899	15 019	-1	4 294 880 896	dernier jour de l'ère NTP -1
1 jan 1900	15 020	0	0	premier jour de l'ère NTP 0
1 jan 1970	40 587	0	2 208 988 800	premier jour UNIX
1 jan 1972	41 317	0	2 272 060 800	premier jour UTC
31 déc 1999	51 543	0	3 155 587 200	dernier jour du 20ème siècle
8 fév 2036	64 731	1	63 1	premier jour de l'ère NTP 1

Figure 4 : Dates NTP intéressantes historiquement

Soit p le nombre de bits significatifs dans la fraction de seconde. La résolution d'horloge est définie comme 2^(-p), en secondes. Afin de minimiser les biais et aider à faire des horodatages imprévisibles pour un intrus, les bits non significatifs devraient être réglés à une chaîne binaire aléatoire sans biais. La précision d'horloge est définie comme le temps courant pour lire l'horloge système, en secondes. Noter que la précision définie de cette façon peut être supérieure ou inférieure à la résolution. Le terme rhô, qui représente la précision utilisée dans le protocole, est la plus grande des deux.

La seule opération arithmétique permise sur les dates et les horodatages est la soustraction de complément à deux, donnant un résultat signé de 127 ou 63 bits. Il est critique que les différences de premier ordre entre deux dates préservent la précision complète de 128 bits et que les différences de premier ordre entre deux horodatages préservent la précision complète de 64 bits. Cependant, les différences sont ordinairement petites par rapport à l'étendue des secondes, de sorte qu'elles peuvent être converties en format à double flottation pour la suite du traitement et sans compromettre la précision.

Il est important de noter que l'arithmétique de complément à deux ne distingue pas entre valeurs signées et non signées (bien que des comparaisons puissent prendre le signe en compte) ; seules les instructions de branche conditionnelle le font. Donc, bien que la distinction soit faite entre dates signées et horodatages non signés, elles sont traitées de la même façon. Un risque reconnu avec des calculs d'horodatage de 64 bits s'étendant sur une ère, comme il sera possible en 2036, peut résulter en un report. En fait, si le client est établi dans les 68 ans du serveur avant que débute le protocole, des valeurs correctes sont obtenues même si le client et le serveur sont dans des ères adjacentes.

Certaines valeurs de temps sont représentées en format d'exposant, incluant la précision, la constante de temps, et l'intervalle d'interrogation. Elles sont en format d'entier signé de 8 bits en log2 (logarithme de base 2) secondes. Les seules opérations arithmétiques permises sur elles sont l'incrémentation et la décrémentation. Pour les besoins du présent document et pour simplifier la présentation, une référence à une de ces variables par le nom signifie la valeur exponentielle, par exemple, l'intervalle d'interrogation est de 1024 s, tandis que la référence par le nom et l'exposant signifie la valeur réelle, par exemple, l'exposant d'interrogation est 10.

Pour convertir l'heure système de tout format en formats de date et horodatage NTP il faut que le nombre de secondes s depuis la première époque jusqu'à l'heure système soit déterminée. Pour déterminer l'entier de l'ère et de l'horodatage étant donné s, ère = s / 2^(32) et horodatage = s - ère * 2^(32), qui fonctionne pour les dates positives et négatives. Pour déterminer s étant donnés l'ère et l'horodatage, s = ère * 2^(32) + horodatage.

La conversion entre l'heure NTP et l'heure système peut être un peu embrouillée, et sort du domaine d'application de ce document. Noter que le nombre de jours dans l'ère 0 est supérieur au nombre de jours dans la plupart des autres ères, et cela ne se reproduira plus jusqu'à l'année 2400 de l'ère 3.

Dans la description des variables d'état qui suit, une référence explicite au type d'entier implique un entier non signé de 32 bits. Cela simplifie les vérifications de frontière, car seule la limite supérieure a besoin d'être définie. Sans référence explicite, le type par défaut est la double flottation à 64 bits. Les exceptions seront notées comme nécessaire.

7. Structures de données

Les automates à état NTP sont définis dans les paragraphes suivants. Les variables d'état sont séparées en classes selon leur fonction dans les en-têtes de paquet, les processus d'homologue et d'interrogation, le processus système, et le processus de discipline d'horloge. Les variables de paquet représentent les valeurs d'en-tête NTP dans les paquets émis et reçus. Les variables d'homologue et d'interrogation représentent le contenu de l'association séparément pour chaque serveur. Les variables de système représentent l'état du serveur tel que vu par ses clients dépendants. Les variables de discipline d'horloge représentent le travail interne de l'algorithme de discipline d'horloge. Un exemple est décrit à l'Appendice A.

7.1 Conventions de structure

Afin de distinguer entre les différentes variables du même nom mais utilisées dans des procès différents, on a adopté la convention de dénomination résumée dans la Figure 5. Une variable de réception de paquet v est un membre de la structure de paquet r avec le nom pleinement qualifié r.v. D'une façon similaire, x.v est une variable de paquet transmis, p.v est une variable d'homologue, s.v est une variable de système, et c.v est une variable de discipline d'horloge. Il y a un ensemble de variables d'homologue pour chaque association ; il y a seulement un ensemble de variables de système et d'horloge.

Nom	Description
r.	variable d'en-tête de paquet en réception
x.	variable d'en-tête de paquet en émission
p.	variable d'homologue/interrogation
s.	variable système
c.	variable de discipline d'horloge

Figure 5 : Conventions de préfixe

7.2 Paramètres globaux

En plus des classes de variables, un certain nombre de paramètres globaux sont définis ici, incluant ceux montrés avec les valeurs de la Figure 6.

Nom	Valeur	Description
PORT	123	numéro d'accès NTP
VERSION	4	numéro de version NTP
TOLERANCE	1,50E-005	tolérance de fréquence PHI (s/s)
MINPOLL	4	exposant minimum d'interrogation (16 s)
MAXPOLL	17	exposant maximum d'interrogation (36 h)
MAXDISP	16	dispersion maximum (16 s)
MINDISP	0,01	incrément minimum de dispersion (s)
MAXDIST	1	seuil de distance (1 s)
MAXSTRAT	16	nombre maximum de strates

Figure 6 : Paramètres globaux

Bien qu'ils soient les seuls paramètres globaux nécessaires pour l'interopérabilité, une plus grande collection est nécessaire dans toute mise en œuvre. L'Appendice A.1.1 contient ceux utilisés par le squelette des algorithmes d'atténuation, l'algorithme de discipline d'horloge, et les fonctions relatives qui dépendent de la mise en œuvre. Certaines de ces valeurs de paramètres sont gravées dans le marbre comme le numéro d'accès NTP alloué par l'IANA et le numéro de version alloué à NTPv4 lui-même. D'autres, comme la tolérance de fréquence (aussi appelée PHI) impliquent une hypothèse sur le pire cas de comportement d'une horloge système une fois synchronisée et ensuite amenée à dériver lorsque ses sources sont devenues injoignables. Les paramètres minimum et maximum définissent les limites des variables d'état comme décrit dans les paragraphes ultérieurs de ce document.

Bien que montrées avec des valeurs fixes dans ce document, certaines mises en œuvre peuvent les rendre ajustables par des commandes de configuration. Par exemple, la mise en œuvre de référence calcule la valeur de PRECISION comme le log2 du temps minimum dans plusieurs itérations pour lire l'horloge système.

7.3 Variables d'en-tête de paquet

D'un point de vue externe les plus importantes variables d'état sont les variables d'en-tête de paquet décrites dans la Figure 7 et ensuite. L'en-tête de paquet NTP consiste en un nombre entier de mots de 32 bits (4 octets) dans l'ordre des octets du réseau. Le format de paquet consiste en trois composants : l'en-tête lui-même, un ou plusieurs champs d'extension facultatifs, et un code facultatif d'authentification de message (MAC, *message autentification code*). L'en-tête est identique à celui de NTPv3 et des versions antérieures. Les champs d'extension facultatifs sont utilisés par les algorithmes de chiffrement de clé publique Autokey décrits dans la [RFC5906]. Le MAC facultatif est utilisé aussi bien par Autokey que l'algorithme de chiffrement à clé symétrique décrit dans la présente RFC.

Nom	Formule	Description
leap	leap	indicateur de saut (LI)
version	version	numéro de version (VN)
mode	mode	mode
stratum	stratum	strate
poll	poll	exposant d'interrogation
precision	rho	exposant de précision
rootdelay	delta_r	retard de racine
rootdisp	epsilon_r	dispersion de racine
refid	refid	identifiant de référence
reftime	reftime	horodatage de référence
org	T1	horodatage d'origine
rec	T2	horodatage de réception
xmt	T3	horodatage d'émission
dst	T4	horodatage de destination
keyid	keyid	identifiant de clé
dgst	dgst	résumé de message

Figure 7 : Variables d'en-tête de paquet

Le paquet NTP est un datagramme UDP [RFC0768]. Certains champs utilisent plusieurs mots et d'autres sont groupés dans de plus petits champs au sein d'un mot. L'en-tête de paquet NTP montré à la Figure 8 a 12 mots suivis par des champs d'extension facultatifs et finalement un code d'authentification de message (MAC) facultatif consistant en les champs Identifiant de clé et Résumé de message.

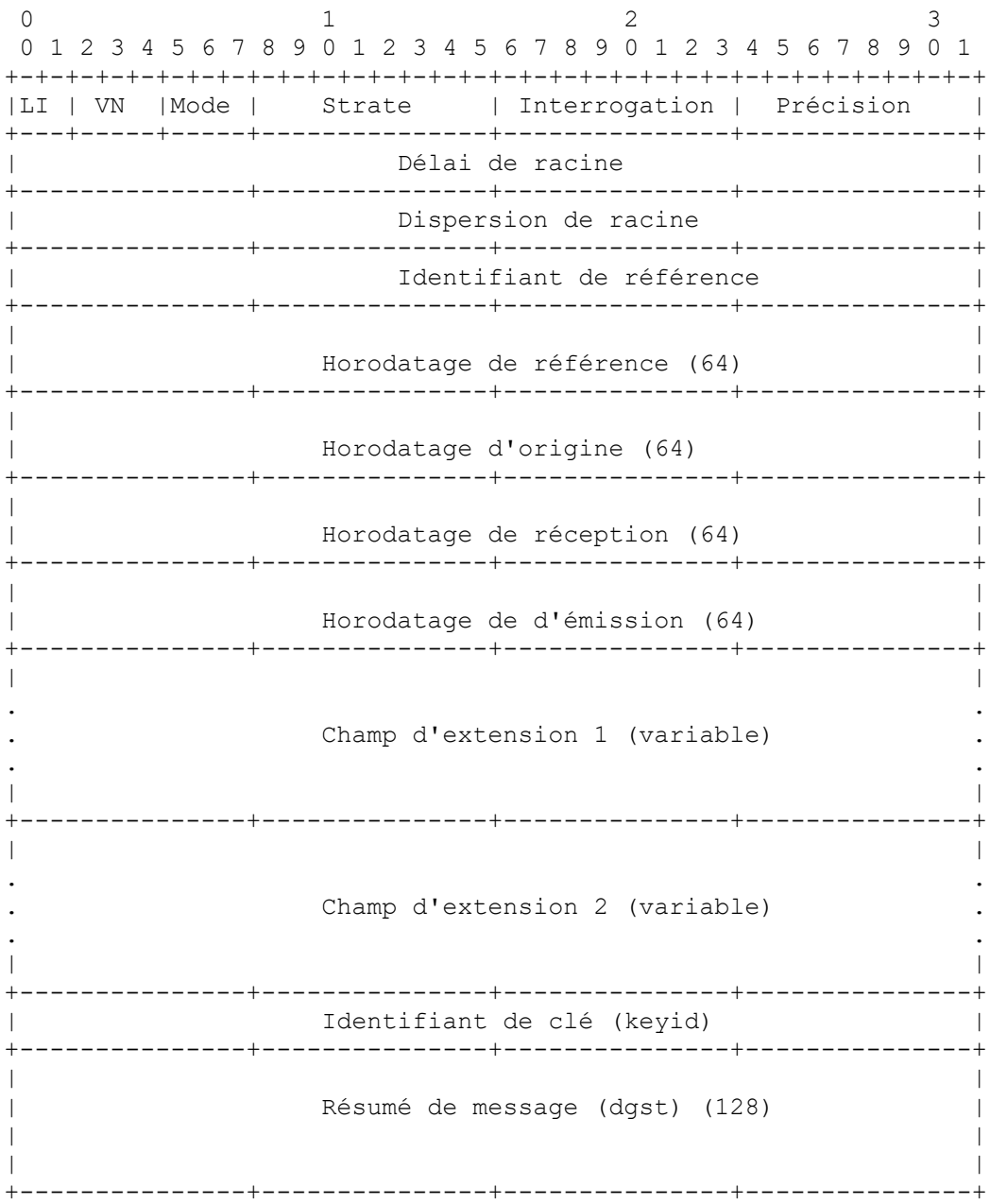


Figure 8 : Format d'en-tête de paquet

Les champs d'extension sont utilisés pour ajouter des capacités facultatives, par exemple, le protocole de sécurité Autokey [RFC5906]. Le format du champ d'extension est présenté afin que le paquet soit analysé sans la connaissance des fonctions du champ d'extension. Le MAC est utilisé aussi bien par Autokey que par le schéma d'authentification de clé symétrique.

Une liste des variables d'en-tête de paquet est montrée à la Figure 7 et elles sont décrites en détail ci-dessous. Sauf une variation mineure quand on utilise la famille d'adresses IPv6, ces champs sont rétro compatibles avec NTPv3. Les champs d'en-tête de paquet s'appliquent aussi bien aux paquets émis (préfixe x) qu'aux paquets reçus (préfixe r). Dans la Figure 8, la taille de certains champs de mots multiples est montrée en bits quand ce n'est pas le 32 bits par défaut. L'en-tête de base s'étend du début du paquet à la fin du champ Horodatage d'émission.

Les variables de champs et de paquet associées (entre parenthèses) sont interprétées comme suit :

LI (*Leap Indicator)* indicateur de saut : entier de 2 bits avertissant d'une seconde à insérer ou à supprimer dans la dernière minute du mois courant avec les valeurs définie dans la Figure 9.

Valeur	Signification
0	pas d'avertissement
1	la dernière minute du jour a 61 secondes
2	la dernière minute du jour a 59 secondes
3	inconnu (horloge non synchronisée)

Figure 9 : Indicateur de saut

VN *(Version Number)* numéro de version : entier de 3 bits représentant le numéro de version NTP, actuellement 4.

Mode : entier de 3 bits représentant le mode, avec les valeurs définies ci dessous.

Valeur	Signification
0	réservé
1	symétrique actif
2	symétrique passif
3	client
4	serveur
5	diffusion
6	message de contrôle NTP
7	réservé pour utilisation privée

Figure 10 : Modes d'association

Strate *(stratum)* : entier de 8 bit représentant la strate, avec les valeurs définies ci-dessous :

Valeur	Signification
0	non spécifié ou invalide
1	serveur principal (par exemple, équipé d'un récepteur GPS)
2 à 15	serveur secondaire (via NTP)
16	non synchronisé
17 à 255	réservé

Figure 11 : Strate de paquet

Il est de coutume de transposer la valeur de strate 0 dans les paquets reçus en MAXSTRAT (16) dans la variable d'homologue p.stratum et de transposer les valeurs de p.stratum de MAXSTRAT ou supérieures à 0 dans les paquets émis. Cela permet aux horloges de référence, qui apparaissent normalement à la strate 0, d'être mixées de façon convenable en utilisant les mêmes algorithmes de choix d'horloge que pour les sources externes (voir un exemple à l'Appendice A.5.5.1).

Interrogation : entier de 8 bits signé qui représente l'intervalle maximum entre les messages successifs, en log2 secondes. Les limites par défaut suggérées pour les intervalles minimum et maximum d'interrogation sont respectivement 6 et 10.

Précision : entier de 8 bits signé qui représente la précision de l'horloge système, en log2 secondes. Par exemple, une valeur de -18 correspond à une précision d'environ une microseconde. La précision peut être déterminée quand le service commence comme le temps minimum de plusieurs itérations pour lire l'horloge système.

Délai de racine (rootdelay) : délai total d'aller-retour à l'horloge de référence, en format NTP court.

Dispersion de racine (rootdisp) : dispersion totale de l'horloge de référence, en format NTP court..

Identifiant de référence (refid) : code de 32 bits qui identifie le serveur ou horloge de référence particulier. L'interprétation dépend de la valeur dans le champ Strate. Pour la strate de paquet 0 (non spécifié ou invalide) c'est une chaîne ASCII [RFC1345] de quatre caractères, appelé "kiss code", utilisée pour le débogage et la surveillance. Pour la strate 1 (horloge de référence) c'est une chaîne ASCII de quatre octets, justifiée à gauche, bourrée de zéros, allouée à l'horloge de référence. La liste d'autorité des identifiants de référence est tenue par l'IANA ; cependant, toute chaîne commençant par le caractère ASCII "X" est réservée pour les expérimentations et développement non enregistrés. Les identifiants de la Figure 12 ont été utilisés comme identifiants ASCII :

Identifiant	Source d'horloge
GOES	Satellite d'environnement d'orbite géosynchrone
GPS	Système mondial de positionnement
GAL	Système de positionnement Galileo
PPS	Impulsion générique par seconde
IRIG	Groupe d'instrumentation inter gammes
WWVB	LF Radio WWVB Ft. Collins, CO 60 kHz
DCF	LF Radio DCF77 Mainflingen, DE 77.5 kHz
HBG	LF Radio HBG Prangins, HB 75 kHz
MSF	LF Radio MSF Anthorn, UK 60 kHz
JJY	LF Radio JJY Fukushima, JP 40 kHz, Saga, JP 60 kHz
LORC	MF Radio LORAN C station, 100 kHz
TDF	MF Radio Allouis, FR 162 kHz
CHU	HF Radio CHU Ottawa, Ontario
WWV	HF Radio WWV Ft. Collins, CO
WWVH	HF Radio WWVH Kauai, HI
NIST	Modem téléphonique du NIST
ACTS	Modem téléphonique du NIST
USNO	Modem téléphonique de USNO
PTB	Modem téléphonique européen

Figure 12 : Identifiants de référence

Au dessus de la strate 1 (serveurs secondaires et clients) : c'est l'identifiant de référence du serveur et cela peut être utilisé pour détecter des boucles de l'heure. Si on utilise la famille d'adresses IPv4, l'identifiant est l'adresse IPv4 de quatre octets. Si on utilise la famille d'adresses IPv6, ce sont les quatre premiers octets du hachage MD5 de l'adresse IPv6. Noter que, quand on utilise la famille d'adresses IPv6 sur un serveur NTPv4 avec un client NTPv3, le champ Identifiant de référence apparaît comme une valeur aléatoire et une boucle du temps peut n'être pas détectée.

Horodatage de référence : heure à laquelle l'horloge système a été établie ou corrigée pour la dernière fois, en format d'horodatage NTP.

Horodatage d'origine (org) : heure chez le client à laquelle la demande est partie pour le serveur, en format d'horodatage NTP.

Horodatage de réception (rec) : heure chez le serveur à laquelle la demande du client est arrivée, en format d'horodatage NTP.

Horodatage d'émission (xmt) : heure chez le serveur à laquelle la réponse est partie pour le client, en format d'horodatage NTP.

Horodatage de destination (dst) : heure chez le client à laquelle la réponse du serveur est arrivée, en format d'horodatage NTP.

Note : Le champ Horodatage de destination n'est pas inclus dans les champs d'en-tête ; il est déterminé à l'arrivée du paquet et rendu disponible dans la structure de données de l'antémémoire de paquet.
Si le NTP a accès à la couche physique, les horodatages sont alors associés au commencement du symbole après le début de trame. Autrement, les mises en œuvre devraient tenter d'associer l'horodatage au premier point accessible dans la trame.

Le MAC consiste en l'Identifiant de clé suivi du Résumé de message. Le résumé de message, ou cryptosomme, est calculé comme dans la [RFC1321] sur tout l'en-tête NTP et les champs d'extension facultatifs, mais pas sur le MAC lui-même.

Champ d'extension n : voir au paragraphe 7.5 la description du format de ce champ.

Identifiant de clé (keyid) : entier non signé de 32 bits utilisé par le client et le serveur pour désigner une clé secrète MD5 de 128 bits.

Résumé de message (dgst) : hachage MD5 de 128 bits calculé sur la clé, suivi par l'en-tête de paquet NTP et les champs d'extensions (mais pas les champs Identifiant de clé ou Résumé de message). On notera que le calcul de MAC utilisé ici diffère de celui défini dans les [RFC1305] et [RFC4330] mais il est cohérent avec la façon dont les mises en œuvre existantes génèrent un MAC.

7.4. Paquet "baiser de la mort"

Si le champ Strate est 0, ce qui implique "non spécifié" ou "invalide", le champ Identifiant de référence peut être utilisé pour porter des messages utiles au rapport de l'état et du contrôle d'accès. Ils sont appelés des paquets "baiser de la mort" (KoD, *Kiss-o'-Death*) et les messages ASCII qu'ils portent sont appelés "kiss codes". Les paquets KoD sont appelés ainsi parce qu'une première utilisation était de dire aux clients d'arrêter d'envoyer des paquets qui violent les contrôles d'accès du serveur. Les kiss codes peuvent fournir des informations utiles pour un client intelligent, NTPv4 ou SNTPv4. Les kiss codes sont codés sur une chaîne de quatre caractères ASCII qui sont justifiés à gauche et bourrés avec des zéros. Les chaînes sont conçues pour l'affichage des caractères et les fichiers d'enregistrement. Une liste de kiss codes actuellement définis est donnée à la Figure 13. Les receveurs des kiss codes DOIVENT les inspecter et, dans les cas ci-dessous, effectuer les actions suivantes :

a. Pour les kiss codes DENY et RSTR, le client DOIT démobiliser toutes les associations à ce serveur et arrêter d'envoyer des paquets à ce serveur ;
b. Pour le kiss code RATE, le client DOIT immédiatement réduire son intervalle d'interrogation à ce serveur et continuer de le réduire chaque fois qu'il reçoit un kiss code RATE.
c. Les kiss codes qui commencent par le caractère ASCII "X" sont pour des expérimentations et développements non enregistrés et DOIVENT être ignorés si ils ne sont pas reconnus.
d. À part les conditions ci-dessus, les paquets KoD n'ont pas de signification pour le protocole et sont éliminés après inspection.

Code	Signification
ACST	L'association appartient à un serveur d'envoi individuel
AUTH	Échec de l'authentification du serveur
AUTO	Échec de la séquence Autokey
BCST	L'association appartient à un serveur de diffusion
CRYP	Échec de l'authentification cryptographique ou de l'identification
DENY	Accès refusé par le serveur distant
DROP	Perte de l'homologue en mode symétrique
RSTR	Accès refusé par la politique locale
INIT	L'association n'a pas encore été synchronisée pour la première fois
MCST	L'association appartient à un serveur découvert de façon dynamique
NKEY	Pas de clé trouvée, qu'elle n'ait jamais été installée ou qu'elle ne soit pas de confiance
RATE	Dépassement de taux. Le serveur a temporairement refusé l'accès parce que le client excède le seuil de taux
RMOT	Altération de l'association de la part d'un hôte distant qui utilise ntpdc
STEP	Un changement d'étape dans l'heure système est intervenu, mais l'association n'a pas encore été resynchronisée

Figure 13 : Kiss codes

L'horodatage de réception et l'horodatage d'émission (réglés par le serveur) sont indéfinis dans un paquet KoD et NE DOIVENT PAS être utilisés pour avoir des valeurs valides et DOIVENT être éliminés.

7.5 Format de champ d'extension NTP

Dans NTPv4, un ou plusieurs champs d'extension peuvent être insérés après l'en-tête et avant le MAC, qui est toujours présent quand un champ d'extension est présent.À part définir le format de champ, ce document n'utilise pas le contenu du champ. Un champ d'extension contient un message de demande ou réponse dans le format de la Figure 14.

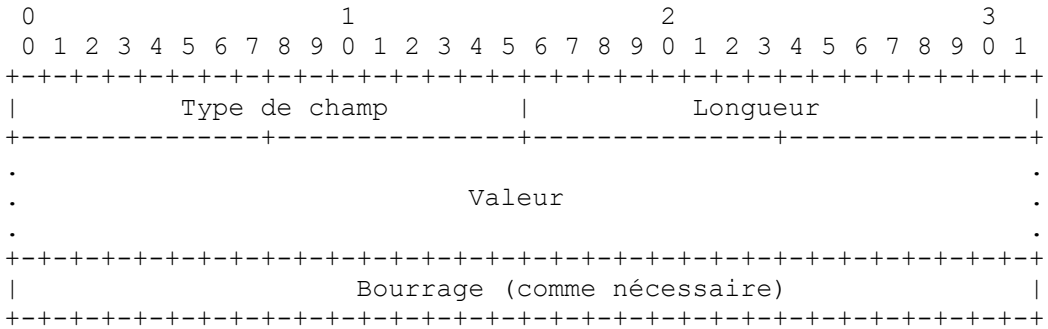


Figure 14 : Format de champ d'extension

Tous les champs d'extension sont bourrés de zéros jusqu'à une limite de mot (quatre octets). Le champ Type de champ est spécifique de la fonction définie et n'est pas plus détaillé ici. Bien que la longueur de champ minimum exigée soit quatre mots (16 octets) une longueur de champ maximum reste à établir.

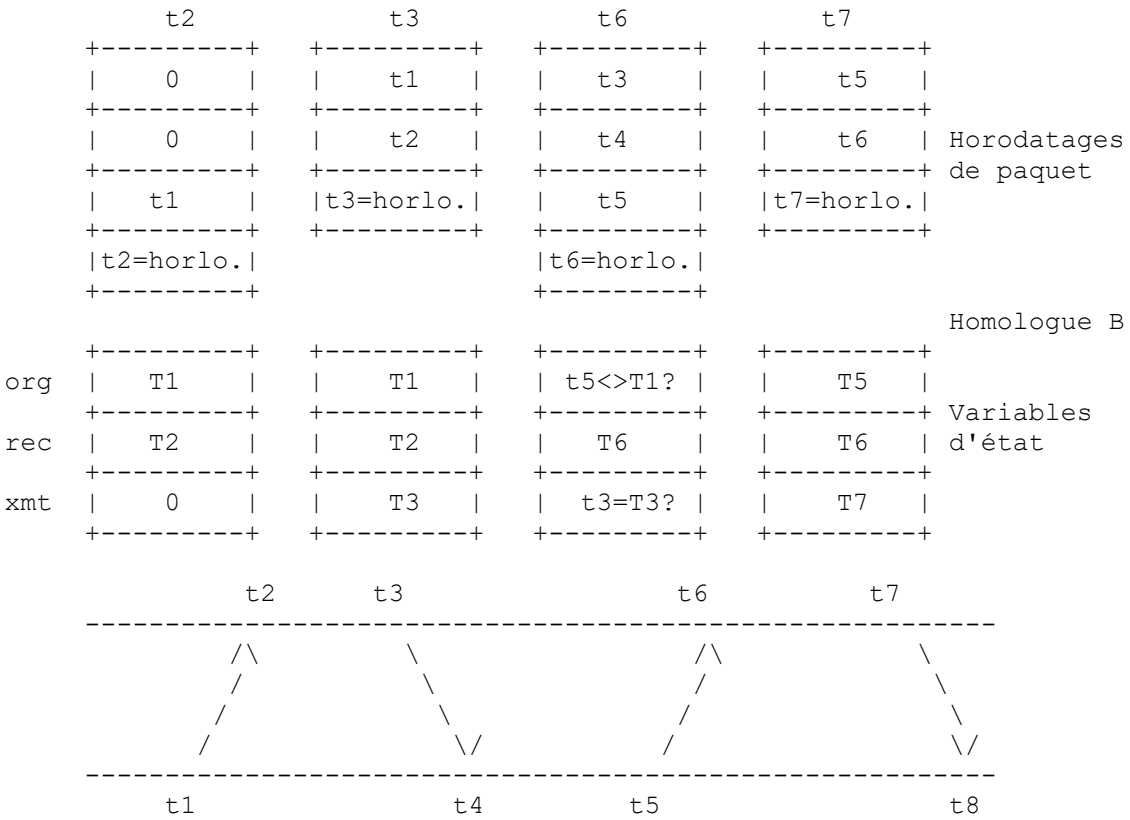
Le champ Longueur est un entier non signé de 16 bits qui indique la longueur du champ d'extension entier en octets, incluant le champ Bourrage.

8. Protocole de réseau

Le cœur du protocole en ligne NTP est le mécanisme central qui échange les valeurs de temps entre les serveurs, les homologues, et les clients. Il est par nature résistant à la perte ou la duplication de paquets. La protection de l'intégrité des données est fournie par les sommes de contrôle IP et UDP. Aucune facilité de contrôle de flux ou de retransmission n'est fournie ni nécessaire. Le protocole utilise des horodatages, qui sont soit extraits des en-têtes de paquet soit tirés de l'horloge système à l'arrivée ou au départ d'un paquet. Les horodatages sont des données de précision et elles devraient être redonnées en cas de retransmission au niveau de la liaison et corrigées du temps nécessaire au calcul d'un MAC en émission.

Les messages NTP utilisent des modes de communication différents, un à un, et un à plusieurs, auxquels on se réfère couramment sous les noms de "envoi individuel" et "envoi en diffusion". Pour les besoins du présent document, le terme de diffusion est interprété comme tout mécanisme disponible de un à plusieurs. Pour IPv4, cela équivaut à la diffusion IPv4 ou à la diffusion groupée IPv4. Pour IPv6, cela équivaut à la diffusion groupée IPv6. À cette fin, l'IANA a alloué l'adresse de diffusion groupée IPv4 224.0.1.1 et l'adresse de diffusion groupée IPv6 se terminant par :101, avec le préfixe déterminé par les règles de portée. Toute autre adresse de diffusion groupée non allouée peut aussi être utilisée en plus de ces adresses de diffusion groupée allouées.

Le protocole en ligne utilise quatre horodatages numérotés de t1 à t4 et trois variables d'état, org, rec, et xmt, comme le montre la Figure 15. Cette figure montre le cas le plus général où chacun des deux homologues, A et B, mesure indépendamment le décalage et le délai par rapport à l'autre. Aux fins d'illustration, les horodatages de paquet sont montrés en minuscules, tandis que les variables d'état sont en majuscules. Les variables d'état sont copiées de l'horodatage du paquet à l'arrivée ou au départ d'un paquet.



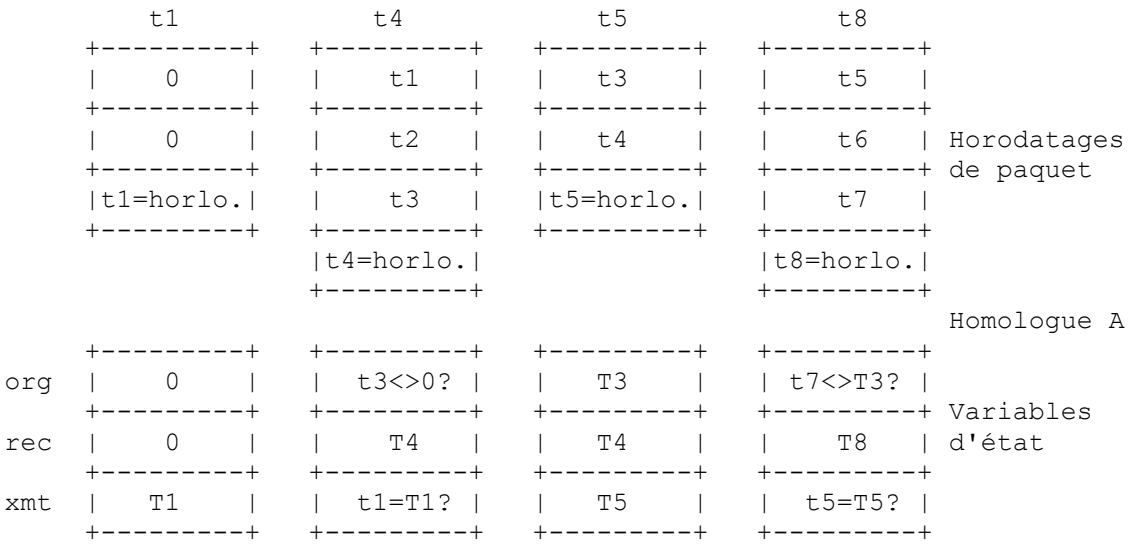


Figure 15 : Protocole en ligne

Dans la figure, le premier paquet transmis par A contient seulement l'horodatage d'origine t1, qui est alors copié à T1. B reçoit le paquet à t2 et copie t1 à T1 et l'horodatage de réception t2 à T2. À ce moment ou un peu plus tard en t3, B envoie un paquet à A contenant t1 et t2 et l'horodatage d'émission t3. Les horodatages sont tous trois copiés aux variables d'état correspondantes. A reçoit le paquet à t4 contenant les trois horodatages t1, t2, et t3 et l'horodatage de destination t4. Ces quatre horodatages sont utilisés pour calculer le décalage et le retard de B par rapport à A, comme décrit ci-dessous.

Avant que les variables d'état xmt et org soient mises à jour, deux vérifications de bonne santé sont effectuées afin de protéger contre les duplications, les paquets erronés ou répétés. Dans l'échange ci-dessus, un paquet est dupliqué ou répété si l'horodatage d'émission t3 dans le paquet correspond à la variable d'état org de T3. Un paquet est bogué si l'horodatage d'origine t1 dans le paquet ne correspond pas à la variable d'état xmt de T1. Dans l'un et l'autre de ces cas, les variables d'état sont mises à jour, puis le paquet est éliminé. Pour protéger contre la répétition du dernier paquet transmis, la variable d'état xmt est réglée à zéro immédiatement après une vérification d'erreur réussie.

Les quatre horodatages les plus récents, de T1 à T4, sont utilisés pour calculer le décalage de B par rapport à A :

theta = T(B) - T(A) = 1/2 * [(T2-T1) + (T3-T4)]

et le délai d'aller-retour : delta = T(ABA) = (T4-T1) - (T3-T2).

Noter que les quantités entre parenthèses sont calculées à partir d'horodatages de 64 bits non signés et résultent en valeurs signées avec 63 bits significatifs plus le signe. Ces valeurs peuvent représenter des dates de 68 ans dans le passé à 68 ans dans le futur. Cependant, le décalage et le délai sont calculés comme des sommes et des différences de ces valeurs, qui contiennent 62 bits significatifs et deux bits de signe, de sorte qu'ils peuvent représenter des valeurs non ambiguës de 34 ans dans le passé à 34 ans dans le futur. En d'autres termes, l'heure du client doit être réglée dans les 34 ans du serveur avant que le service commence. C'est une limitation fondamentale de l'arithmétique d'entiers de 64 bits.

Dans les mises en œuvre où l'arithmétique à double flottement est disponible, les différences de premier ordre peuvent être converties en double flottement et les sommes et différences de second ordre être calculées dans cette arithmétique. Comme les termes de second ordre sont normalement très petits par rapport à la magnitude de l'horodatage, il n'y a pas de perte de signification, et la gamme non ambiguë est restaurée de 34 ans à 68 ans.

Dans certains scénarios où le décalage initial de fréquence du client est relativement grand et où le temps réel de propagation est faible, il est possible que le calcul du délai devienne négatif. Par exemple, si la différence de fréquence est de 100 ppm et si l'intervalle T4-T1 est de 64 s, le délai apparent sera -6,4 ms. Comme les valeurs négatives sont trompeuses dans la suite des calculs, la valeur de delta devrait être calée à pas moins de s.rho, où s.rho est la précision du système décrite au paragraphe 11.1, exprimée en secondes.

La discussion précédente suppose le cas le plus général où deux homologues symétriques mesurent indépendamment les décalages et les délais entre eux. Dans le cas d'un serveur sans état, le protocole peut être simplifié. Un serveur sans états copie T3 et T4 du paquet client à T1 et T2 du paquet de serveur et joint l'horodatage d'émission T3 avant de l'envoyer au

client. Des détails supplémentaires pour remplir les champs de protocole restants sont donnés dans la Section 9 et les paragraphes suivants et dans l'Appendice.

Noter que le protocole en ligne tel qu'il est décrit résiste à la répétition d'un paquet de réponse du serveur. Cependant, il ne résiste pas à la répétition d'un paquet de demande de client, qui résulterait en un paquet de réponse du serveur avec de nouvelles valeurs de T2 et T3 et un décalage et délai incorrects. Cette vulnérabilité peut être évitée en réglant la variable d'état xmt à zéro après le calcul du décalage et du délai.

9. Processus d'homologues

Les descriptions de processus qui suivent incluent une liste des variables d'état importantes, suivie par une vue d'ensemble des opérations mises en œuvre, comme des sous programmes. Une fréquente référence est faite au squelette de l'Appendice. Le squelette inclut des fragments en langage C qui décrivent les fonctions plus en détail. Cela inclut les paramètres, variables, et déclarations nécessaires pour une mise en œuvre conforme à NTPv4. Cependant, de nombreuses variables et sous programmes supplémentaires peuvent être nécessaire dans une mise en œuvre réelle.

Le processus d'homologue est invoqué à l'arrivée d'un paquet d'un serveur ou d'un homologue. Il fait fonctionner le protocole en ligne pour déterminer le décalage et le délai d'aller retour de l'horloge et calcule les statistiques utilisées par le système et le processus d'interrogation. Les variables d'homologue sont instanciées dans la structure d'association des données quand la structure est initialisée et mise à jour par l'arrivée de paquets. Il y a un processus d'homologue, un processus d'interrogation, et un processus d'association pour chaque serveur.

9.1 Variables de processus d'homologues

Les Figures 16, 17, 18, et 19 résument les noms communs, les noms de formules, et une brève description des variables de l'homologue. Les noms communs et les noms de formules sont interchangeables ; les noms de formules sont destinés à améliorer la lisibilité des équations de cette spécification. Sauf notation contraire, toutes les variables d'homologue supposent le préfixe p.

Nom	Formule	Description
srcaddr	srcaddr	adresse de source
srcport	srcport	accès de source|
dstaddr	dstaddr	adresse de destination
dstport	destport	accès de destination
keyid	keyid	identifiant de clé

Figure 16 : Variables de configuration de processus d'homologue

Nom	Formule	Description
leap	leap	indicateur de saut
version	version	numéro de version
mode	mode	mode
stratum	stratum	strate
ppoll	ppoll	exposant d'interrogation d'homologue
rootdelay	delta_r	délai de racine
rootdisp	epsilon_r	dispersion de racine
refid	refid	identifiant de référence
reftime	reftime	horodatage de référence

Figure 17 : Variables de paquet du processus d'homologue

Nom	Formule	Description
org	T1	horodatage d'origine
rec	T2	horodatage de réception
xmt	T3	horodatage émission
t	t	heure du paquet

Figure 18 : Variables d'horodatages du processus d'homologue

Nom	Formule	Description
offset	theta	décalage d'horloge
delay	delta	délai d'aller-retour
disp	epsilon	dispersion
jitter	psi	gigue
filter	filter	filtre d'horloge
tp	t_p	heure de filtrage

Figure 19 : Variables de statistiques de processus d'homologue

Les variables de configuration suivantes sont normalement initialisées quand l'association est mobilisée, soit à partir d'un fichier de configuration, soit à l'arrivée du premier paquet pour une association inconnue.

srcaddr : adresse IP du serveur ou horloge de référence distant. Cela devient l'adresse IP de destination dans les paquets envoyés à partir de cette association.

srcport : numéro d'accès UDP du serveur ou horloge de référence. Cela devient le numéro d'accès de destination dans les paquets envoyés à partir de cette association. Dans les modes symétriques (1 et 2) ce champ doit contenir le numéro d'accès NTP PORT (123) alloué par l'IANA. Dans les autres modes, il peut contenir tout nombre cohérent avec la politique locale.

dstaddr : adresse IP du client. Cela devient l'adresse IP de source dans les paquets envoyés à partir de cette association.

dstport : numéro d'accès UDP du client, ordinairement le numéro d'accès NTP PORT (123) alloué par l'IANA. Cela devient le numéro d'accès de source dans les paquets envoyés depuis cette association.

keyid : identifiant de clé symétrique pour la clé MD5 de 128 bits utilisée pour générer et vérifier le MAC. Le client et le serveur ou l'homologue peuvent utiliser des valeurs différentes, mais elles doivent se transposer en la même clé.

Les variables définies à la Figure 17 sont mises à jour à partir de l'en-tête de paquet lorsque chaque paquet arrive. Elles sont interprétées de la même façon que les variables de paquet de même nom. Il est pratique pour la suite du traitement de convertir les valeurs de paquet de format NTP court r.rootdelay et r.rootdisp en variables d'homologues à double flottement.

Les variables définies à la Figure 18 incluent les horodatages échangés par le protocole en ligne à la Section 8. La variable t est le compteur de secondes c.t associé à ces valeurs. La variable c.t est tenue par le processus d'ajustement d'horloge décrit à la Section 12. Il compte les secondes depuis que le service a commencé. Les variables définies à la Figure 19 incluent les statistiques calculées par le sous programme clock_filter() décrit à la Section 10. La variable tp est le compteur de secondes associé à ces valeurs.

9.2 Opérations du processus d'homologues

Le sous programme de réception définit le flux de traitement à l'arrivée d'un paquet. Un exemple est décrit par le sous programme receive() dans l'Appendice A.5.1. Il n'y a pas de méthode spécifique exigée pour le contrôle d'accès, bien qu'il soit recommandé que les mises en œuvre incluent un tel schéma, qui est similaire à de nombreux autres, d'une utilisation maintenant très répandue. Le sous programme access() de l'Appendice A.5.4 décrit une méthode qui met en œuvre des restrictions d'accès en utilisant une liste de contrôle d'accès (ACL, *access control list*). Les vérifications de format exigent une longueur de champ et un alignement corrects, un numéro de version acceptable (1 à 4) et une syntaxe correcte de champ d'extension, si il y en a.

Il n'y a pas d'exigences spécifique pour l'authentification ; cependant, si l'authentification est mise en œuvre, l'algorithme de hachage chiffré MD5 décrit dans la [RFC1321] doit être pris en charge.

Ensuite, une recherche dans le tableau d'association est effectuée pour trouver l'adresse et l'accès de source qui correspondent, par exemple, en utilisant le sous programme find_assoc() dans l'Appendice A.5.1. La Figure 20 est un tableau de répartition où les colonnes correspondent au mode de paquet et les rangées correspondent au mode d'association. L'intersection des modes d'association et de paquet répartit le traitement à une des étapes suivantes.

		Mode paquet				
Mode d'association		1	2	3	4	5
Pas d'association	0	NEWPS	DSCRD	FXMIT	MANY	NEWBC
Sym. active	1	PROC	PROC	DSCRD	DSCRD	DSCRD
Sym. passive	2	PROC	ERR	DSCRD	DSCRD	DSCRD
Client	3	DSCRD	DSCRD	DSCRD	PROC	DSCRD
Serveur	4	DSCRD	DSCRD	DSCRD	DSCRD	DSCRD
Diffusion	5	DSCRD	DSCRD	DSCRD	DSCRD	DSCRD
Client de diffusion	6	DSCRD	DSCRD	DSCRD	DSCRD	PROC

Figure 20 : Tableau de répartition d'homologue

DSCRD : ceci indique une violation non fatale du protocole par suite d'une erreur de programmation, un paquet trop en retard, ou répété. Le processus d'homologue élimine le paquet et se termine.

ERR : ceci indique une violation fatale du protocole par suite d'une erreur de programmation, un paquet trop en retard ou répété. Le processus d'homologue élimine le paquet, démobilise l'association passive symétrique, et se termine.

FXMIT : ceci indique un paquet en mode client (mode 3) qui ne correspond à aucune association (mode 0). Si l'adresse de destination n'est pas une adresse de diffusion, le serveur construit un paquet de serveur (mode 4) et le retourne au client sans conserver l'état. L'en-tête de paquet de serveur est construit. Un exemple est décrit pas le dernier sous programme fast_xmit() à l'Appendice A.5.3. L'en-tête de paquet est assemblé à partir du paquet reçu et des variables système comme le montre la Figure 21. Si les variables système s.rootdelay et s.rootdisp sont mémorisées en double flottement, elles doivent être converties d'abord en format NTP court.

Variable de paquet	**-->**	**Variable**
r.leap	-->	p.leap
r.mode	-->	p.mode
r.stratum	-->	p.stratum
r.poll	-->	p.ppoll
r.rootdelay	-->	p.rootdelay
r.rootdisp	-->	p.rootdisp
r.refid	-->	p.refid
r.reftime	-->	p.reftime
r.keyid	-->	p.keyid

Figure 21 : En-tête de paquet en réception

Noter que, si l'authentification échoue, le serveur retourne un message spécial appelé un crypto-NAK. Ce message inclut les données d'en-tête NTP normal montré à la Figure 8, mais avec un MAC consistant en quatre octets de zéros. Le client PEUT accepter ou rejeter les données du message. Après cette action, le processus d'homologue se termine.

Si l'adresse de destination est une adresse de diffusion groupée, l'envoyeur fonctionne en mode client multi envois. Si le paquet est valide et si la strate du serveur est inférieure à la strate du client, le serveur envoie un paquet de serveur ordinaire (mode 4) mais qui utilise son adresse de destination d'envoi individuel. Un crypto-NAK n'est pas envoyé si l'authentification échoue. Après ces actions, le processus d'homologue se termine.

MANY : cela indique un paquet de serveur (mode 4) ne correspondant à aucune association. Ordinairement, cela ne peut arriver que par suite d'une réponse de serveur multi envois à un paquet de client de diffusion groupé envoyé précédemment. Si le paquet est valide, une association ordinaire de client (mode 3) est mobilisée et l'opération continue comme si l'association était mobilisée par un fichier de configuration.

NEWBC : cela indique un paquet en diffusion (mode 5) ne correspondant à aucune association. Le client mobilise une association de client (mode 3) ou une association de client de diffusion (mode 6). Des exemples sont présentés dans les sous programmes mobilize() et clear() à l'Appendice A.2. Ensuite, le paquet est validé et les variables d'homologue sont initialisées. Un exemple en est fourni par le sous programme packet() à l'Appendice A.5.1.1.

Si la mise en œuvre ne prend pas en charge de fonctions de sécurité ou de calibrage supplémentaires, le mode d'association est réglé à client de diffusion (mode 6) et le processus d'homologue se termine. Les mises en œuvre qui prennent en charge l'authentification de clé publique PEUVENT utiliser le protocole de sécurité Autokey ou un

protocole équivalent. Les mises en œuvre DEVRAIENT régler le mode d'association à 3 et faire un échange client/serveur court pour déterminer le délai de propagation. Suite à l'échange, le mode d'association est réglé à 6 et le processus d'homologue continue en mode d'écoute seule. Noter la distinction entre un paquet de mode 6, qui est réservé pour les fonctions de surveillance et contrôle de NTP, et une association de mode 6.

NEWPS : cela indique un paquet symétrique actif (mode 1) ne correspondant à aucune association. Le client mobilise une association symétrique passive (mode 2). Un exemple en est montré dans les sous programmes mobilize() et clear() à l'Appendice A.2. Le traitement se poursuit dans la section PROC ci-dessous.

PROC : cela indique un paquet qui correspond à une association existante. Les horodatages du paquet sont vérifiés avec soin pour éviter les paquets invalides, dupliqués, ou erronés. Des vérifications supplémentaires sont récapitulées à la Figure 22. Noter que tous les paquets, y compris crypto-NAK, ne sont considérés valides que si ils passent ces tests.

Type	Description	Commentaire
1	paquet dupliqué	Le paquet est au mieux un vieux dupliqué ou au pire une répétition par un pirate. Cela peut arriver dans des modes symétriques si les intervalles d'interrogation sont irréguliers.
2	paquet erroné	
3	invalide	Un ou plusieurs champs d'horodatage sont invalides. Cela apparaît normalement dans les modes symétriques quand un homologue envoie le premier paquet à l'autre et avant que l'autre ait reçu sa première réponse.
4	accès refusé	Le contrôle d'accès a mis cette source en liste noire.
5	échec d’authentification	Le résumé de message cryptographique ne correspond pas au MAC.
6	non synchronisé	Le serveur n'est pas synchronisé à une source valide.
7	mauvais en-tête de données	Un ou plusieurs champs d'en-tête sont invalides.

Figure 22 : Vérification des erreurs de paquet

Le traitement se continue par la copie des variables de paquet aux variables d'homologue, comme le montre la Figure 21. Un exemple est décrit par le sous programme packet() à l'Appendice A.5.1.1. Le sous programme receive() met en œuvre les essais 1 à 5 de la Figure 22 ; le sous programme packet() met en œuvre les essais 6 et 7. Si on trouve des erreurs, le paquet est éliminé et le processus d'homologue se termine.

Le protocole en ligne calcule le décalage d'horloge théta et le délai d'aller-retour delta à partir des quatre plus récents horodatages, comme décrit dans Section 8. Alors qu'il est, en principe, possible de faire tous les calculs excepté les différences d'horodatage de premier ordre en arithmétique à virgule fixe, il est beaucoup plus facile de convertir les différences de premier ordre en double flottement et de faire le reste des calculs dans cette arithmétique, et c'est cela qui sera supposé dans la suite de la description.

Ensuite, on utilise le registre glissant de 8 bits p.reach du processus d'interrogation décrit à la Section 13 pour déterminer si le serveur est accessible et si les données sont fraîches. Le registre est glissé à gauche de un bit quand un paquet est envoyé et le bit de droite est réglé à zéro. Lorsque des paquets valides arrivent, le bit de droite est réglé à un. Si le registre contient des bits non zéro, le serveur est considéré accessible ; autrement, il ne l'est pas. Comme l'intervalle d'interrogation de l'homologue peut avoir changé depuis le dernier paquet, l'intervalle d'interrogation de l'hôte est revu. Un exemple est fourni par le sous programme poll_update() à l'Appendice A.5.7.2.

La statistique de dispersion epsilon(t) représente l'erreur maximum due à la tolérance de fréquence et au temps écoulé depuis l'envoi du dernier paquet. Elle est initialisée à epsilon(t_0) = r.rho + s.rho + PHI * (T4-T1) quand la mesure est faite à t_0 selon le compteur de secondes. Ici, r.rho est la précision de paquet décrite au paragraphe 7.3 et s.rho est la précision du système décrite au paragraphe 11.1, toutes deux exprimées en secondes. Ces termes sont nécessaires pour tenir compte de l'incertitude de lecture de horloge système chez le serveur et chez le client.

La dispersion croît alors au taux constant PHI ; en d'autres termes, à l'instant t, epsilon(t) = epsilon(t_0) + PHI * (t-t_0). Avec la valeur par défaut PHI = 15 ppm, cela fait environ 1,3 s par jour. Cela étant, l'argument t va être abandonné et la dispersion représentée simplement par epsilon. Les statistiques restantes sont calculées par l’algorithme de filtre d'horloge décrit à la Section suivante.

10. Algorithme de filtre d'horloge

L'algorithme de filtre d'horloge fait partie du processus d'homologue. Il gère le flux des données sur le réseau pour choisir les échantillons les plus aptes à représenter l'heure précise. L'algorithme produit les variables montrées à la Figure 19, incluant le décalage (theta), le délai (delta), la dispersion (epsilon), la gigue (psi), et l'heure d'arrivée (t). Ces données sont utilisées par les algorithmes d'atténuation pour déterminer le meilleur décalage et le décalage final utilisé pour discipliner le système d'horloge. Elles sont aussi utilisées pour déterminer la santé du serveur et si il convient pour la synchronisation.

L'algorithme de filtre d'horloge sauvegarde les plus récents quartets d'échantillons (theta, delta, epsilon, t) dans la structure de filtre, qui fonctionne comme un registre glissant de 8 étapes. Les quartets sont sauvegardés dans l'ordre d'arrivée des paquets. Ici, t est l'heure d'arrivée du paquet conformément au compteur de secondes et ne devrait pas être confondu avec la variable d'homologue tp.

Le schéma suivant est utilisé pour s'assurer que des échantillons suffisants sont dans le filtre et que les vieilles données périmées sont éliminées. Initialement, les quartets de toutes les étapes sont réglés au quartet factice (0, MAXDISP, MAXDISP, 0). Lorsque des paquets valides arrivent, les quartets sont glissés dans le filtre, causant l'élimination des vieux quartets, afin que finalement seuls les quartets valides restent.

Si les trois bits de moindre poids de chaque registre sont à zéro, indiquant que trois intervalles d'interrogation sont expirés sans paquet valide reçu, le processus d'interrogation invoque l'algorithme de filtre d'horloge avec un quartet factice tout comme si le quartet était arrivé du réseau. Si cela persiste pendant huit intervalles d'interrogation, le registre retourne à la condition initiale.

Dans l'étape suivante, les étapes du registre glissant sont copiées dans une liste temporaire et la liste est triée par delta croissant. Soit i l'indice des étapes commençant pas le plus bas delta. Si la première époque du quartet t_0 n'est pas plus tard que le dernier échantillon valide d'époque tp, le sous programme se termine sans affecter les variables d'homologue actuelles. Autrement, soit epsilon_i la dispersion de la ième entrée, alors

$$\text{epsilon} = \sum_{i=0}^{i=n-1} \frac{\text{epsilon\_i}}{2^{(i+1)}}$$

est la dispersion d'homologue p.disp. Noter que la surcharge de epsilon, en entrée ou en sortie du filtre d'horloge, devrait être claire d'après le contexte.

L'observateur devrait noter que (a) si toutes les étapes contiennent le quartet factice avec la dispersion MAXDISP, la dispersion calculée est un peu moins que 16 s, (b) chaque fois qu'un quartet valide est glissé dans le registre, la dispersion diminue d'un peu moins de la moitié, selon la dispersion des quartets valides, et (c) après le quatrième paquet valide, la dispersion est généralement d'un peu moins de 1 s, qui est la valeur supposée du paramètre MAXDIST utilisé par l'algorithme de sélection pour déterminer si les variables d'homologue sont acceptables ou non.

Soit theta_0 le premier décalage d'étape dans la liste triée ; alors, pour les autres étapes dans tout ordre, la gigue est le RMS moyen

$$\text{psi} = \frac{1}{(n-1)} * \sqrt{\sum_{j=1}^{n-1} (\text{theta\_0} - \text{theta\_j})^2}$$

où n est le nombre de quartets valides dans le filtre (n > 1). Afin d'assurer la cohérence et éviter de diviser des exceptions dans d'autres calculs, le psi est bordé par en dessous par la précision du système s.rho exprimée en secondes. Bien qu'en général elle ne soit pas considérée comme un facteur majeur dans le classement de la qualité du serveur, la gigue est un indicateur valable des performances fondamentales de la conservation de l'heure et de l'état d'encombrement du réseau.

D'une importance particulière pour les algorithmes d'atténuation est la distance de synchronisation de l'homologue, qui est calculée à partir du délai et de la dispersion.

lambda = (delta / 2) + epsilon.

Noter que epsilon et donc lambda augmentent au taux PHI. Le lambda n'est pas une variable d'état, car lambda est recalculé à chaque utilisation. C'est un composant de la distance de synchronisation de la racine utilisé par les algorithmes d'atténuation comme métrique pour évaluer la qualité de l'heure disponible provenant de chaque serveur.

Il est important de noter que, à la différence de NTPv3, les associations NTPv4 ne présentent pas une condition de fin de temporisation en réglant la strate à 16 et l'indicateur de saut à 3. Les variables d'association conservent les valeurs déterminées à l'arrivée du dernier paquet. Dans NTPv4, lambda augmente avec le temps, de sorte que finalement la distance de synchronisation excède le seuil de distance MAXDIST, et dans ce cas, l'association est considérée comme impropre à la synchronisation.

Un exemple de mise en œuvre de l'algorithme de filtre d'horloge est montré dans le sous programme clock_filter() à l'Appendice A.5.2.

11. Processus du système

Comme chaque nouvel échantillon (theta, delta, epsilon, jitter, t) est produit par l'algorithme de filtre d'horloge, tous les processus d'homologue sont examinés par les algorithmes d'atténuation consistant en les algorithmes de sélection, d'amas, de combinaison, et de discipline d'horloge dans le processus système. L'algorithme sélection examine toutes les associations et élimine les "faux tiqueurs", qui ont une heure incorrecte démontrée, laissant seulement les "vrais sonneurs". Dans une série de tours, l'algorithme d'amas élimine les associations qui sont statistiquement les plus éloignées du centre jusqu'à ce qu'un nombre minimum spécifié de survivants restent. L'algorithme de combinaison produit les meilleures statistiques finales sur la base d'une moyenne pondérée. Le décalage final est passé à l'algorithme de discipline d'horloge pour amener l'horloge système à l'heure correcte.

L'algorithme d'amas choisit un des survivants comme homologue système. Les statistiques associées (theta, delta, epsilon, jitter, t) sont utilisées pour construire les variables de système héritées par les serveurs et clients dépendants et rendues disponibles aux autres applications qui fonctionnent sur la même machine.

11.1 Variables du processus du système

La Figure 23 résume les noms communs, les noms de formules, et une brève description de chaque variable système. Sauf mention contraire, toutes les variables supposent le préfixe s.

Nom	Formule	Description
t	t	heure de mise à jour
p	p	identifiant d'homologue système
saut	leap	indicateur de saut
strate	stratum	strate
précision	rho	précision
décalage	THETA	décalage combiné
gigue	PSI	gigue combinée
rootdelay	DELTA	délai de racine
rootdisp	EPSILON	dispersion de racine
v	v	liste des survivants
refid	refid	Identifiant de référence
reftime	reftime	heure de référence
NMIN	3	nombre minimum de survivants
CMIN	1	minimum de candidats

Figure 23 : Variables du processus du système

Sauf pour les variables t, p, décalage, et gigue et les constantes NMIN et CMIN, les variables ont le même format et interprétation que les variables d'homologue du même nom. Les paramètres NMIN et CMIN sont utilisés par les algorithmes de sélection et d'amas décrits au paragraphe suivant.

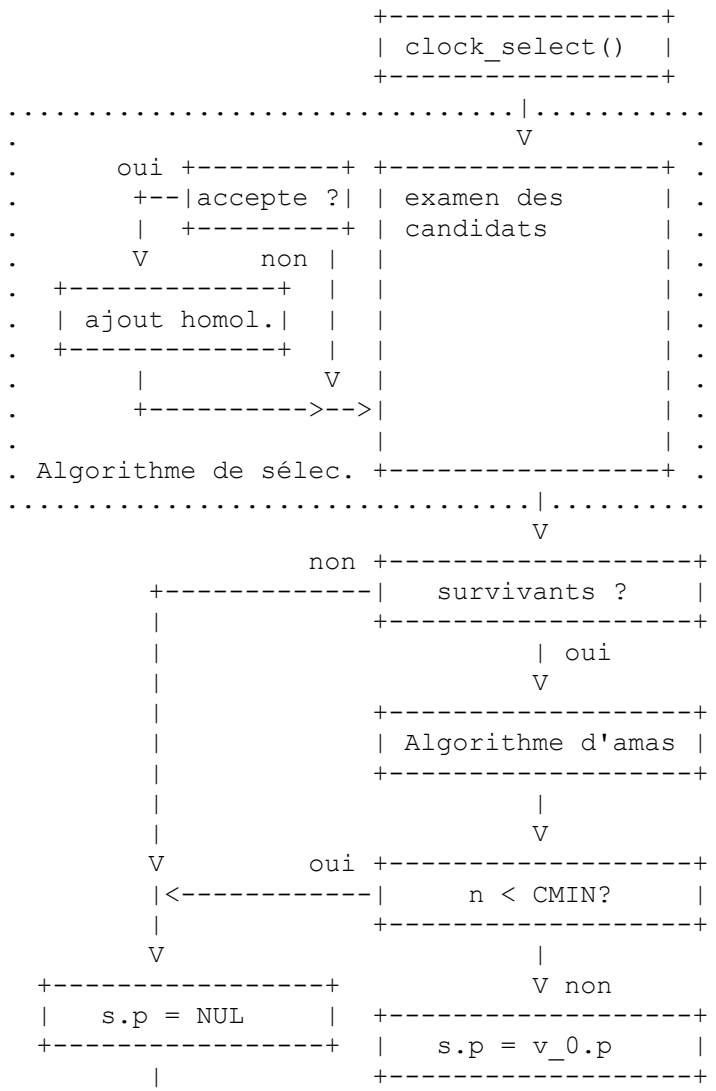
La variable t est le compteur de secondes au moment de la dernière mise à jour. Un exemple est montré par le sous programme clock_update() à l'Appendice A.5.5.4. La variable p est l'identifiant de l'homologue système déterminé par le sous programme cluster() au paragraphe 11.2.2. La variable précision a le même format que la variable paquet du même nom. La précision est définie comme le plus grand de la résolution et de l'heure pour lire l'horloge, en unités log2. Par exemple, la précision d'une horloge à la fréquence du réseau d'alimentation électrique qui s'incrémente à 60 Hz est 16 ms, même quand la représentation du matériel de l'horloge est à la nanoseconde.

Les variables décalage et gigue sont déterminées par l'algorithme de combinaison au paragraphe 11.2.3. Ces valeurs représentent le meilleur et final décalage et gigue utilisés pour discipliner l'horloge système.

Initialement, toutes les variables sont ramenées à zéro, puis le saut est réglé à 3 (non synchronisé) et la strate est réglée à MAXSTRAT (16). On se rappelle que MAXSTRAT est transposé en zéro dans le paquet transmis.

11.2 Opérations du processus du système

La Figure 24 résume les opérations du processus système effectuées par le sous programme de choix d'horloge. L'algorithme de sélection décrit au paragraphe 11.2.1 produit une clique majoritaire de candidats présumés corrects (vrai sonneurs) sur la base des principes d'accord. L'algorithme d'amas décrit au paragraphe 11.2.2 élimine les écarts pour produire la liste des survivants les plus précis. L'algorithme de combinaison décrit au paragraphe 11.2.3 fournit le meilleur et final décalage pour l'algorithme de discipline d'horloge. Un exemple est décrit à l'Appendice A.5.5.6. Si l'algorithme de sélection ne peut pas produire une clique majoritaire, ou si il ne peut pas produire au moins CMIN survivants, le processus système se termine sans avoir discipliné l'horloge système. Si il réussit, l'algorithme d'amas choisit le candidat statistiquement le meilleur comme homologue système et ses variables sont héritées comme variables de système.



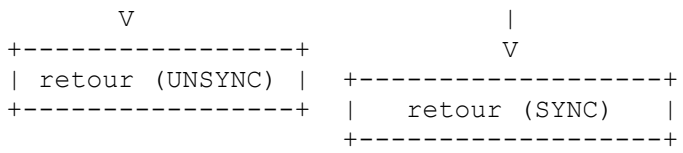


Figure 24 : Sous programme de choix d'horloge

11.2.1 Algorithme de choix

Noter que les algorithmes de sélection et d'amas sont décrits séparément, mais combinés dans le squelette de code. L'algorithme de sélection opère pour trouver un intervalle d'intersection contenant une clique majoritaire de vrais sonneurs en utilisant les principes d'accord byzantin proposés à l'origine par Marzullo [ref6], mais modifiés pour améliorer la précision. une vue d'ensemble de l'algorithme est donnée ci-après et il est décrit dans la première moitié du sous programme clock_select() à l'Appendice A.5.5.1.

D'abord, les serveurs qui ne sont pas utilisables selon les règles du protocole sont détectés et éliminés comme le montre le sous programme accept() à l'Appendice A.5.5.3. Ensuite, un ensemble de triplets (p, type, edge) est généré pour les candidats restants. Ici, p est l'identifiant d'association et type identifie les points d'extrémité supérieur (+1), moyen (0), et inférieur (-1) d'un intervalle de correction centré sur theta pour ce candidat. Il en résulte trois triplets, point bas (p, -1, theta - lambda), point moyen (p, 0, theta), et point haut (p, +1, theta + lambda) où lambda est la distance de synchronisation de la racine. Un exemple de ce calcul est montré par le sous programme rootdist() à l'Appendice A.5.1.1. Les étapes de l'algorithme sont :

1. Pour chacune des m associations, placer trois triplets comme défini ci-dessus sur la liste des candidats.

2. Trier les triplets de la liste par composant edge. Ordonner le point bas, le point moyen, et point haut de ces intervalles du plus bas au plus haut. Régler le nombre de faux tiqueurs f = 0.

3. Régler le nombre de points moyens d = 0. Régler c = 0. Examiner du point d'extrémité le plus bas au plus haut. Ajouter un à c pour chaque point bas, soustraire un pour chaque point haut, ajouter un à d pour chaque point moyen. Si c ≥ m - f, arrêter ; régler l = point bas actuel.

4. Régler c = 0. Examiner du point d'extrémité supérieur au point d'extrémité inférieur. Ajouter un à d pour chaque point moyen. Si c ≥ m - f, arrêter ; régler u = point haut actuel.

5. Si d = f et l < u, suivre l'étape 5A ; sinon, suivre l'étape 5B.

5A. Réussite : l'intervalle d'intersection est [l, u].

5B. Ajouter un à f. Si f < (m / 2) passer à nouveau à l'étape 3. Sinon, passer à l'étape 6.

6. Échec ; on n'a pas pu trouver une clique majoritaire. Il n'y a pas de candidat convenable pour discipliner l'horloge système.

L'algorithme est décrit en détails à l'Appendice A.5.5.1. Noter qu'il commence par l'hypothèse qu'il n'y a pas de faux tiqueur (f = 0) et qu'il tente de trouver un intervalle d'intersection non vide contenant les points médians de tous les serveurs corrects, c'est-à-dire, les vrais sonneurs. Si on ne peut pas trouver un intervalle non vide, on augmente de un le nombre de faux tiqueurs supposés et on essaye à nouveau. Si on trouve un intervalle non vide et si le nombre de faux tiqueurs est inférieur au nombre de vrais sonneurs, une clique majoritaire a été trouvée et le point médian de chaque vrai sonneur (theta) représente les candidats disponibles pour l'algorithme d'amas.

Si on ne trouve pas une clique majoritaire, ou si le nombre de vrais sonneurs est inférieur à CMIN, les candidats sont insuffisants pour discipliner l'horloge système. CMIN définit le nombre minimum de serveurs cohérent avec les exigences de correction. Un opérateur soupçonneux va régler CMIN de façon à s'assurer que plusieurs serveurs redondants sont disponibles pour que les algorithmes fassent un mélange approprié. Cependant, pour des raisons historiques la valeur par défaut pour CMIN est un.

11.2.2 Algorithme d'amas

Les candidats de la clique majoritaire sont placés dans la liste des survivants sous la forme de quartets (p, theta_p, psi_p, lambda_p) où p est un identifiant d'association, theta_p, psi_p, et stratum_p sont respectivement le décalage actuel, la gigue et la strate de l'association p, et lambda_p est un facteur de mérite égal à stratum_p * MAXDIST + lambda, où lambda est la distance de synchronisation de la racine pour l'association p. La liste est traitée par l'algorithme d'amas ci-dessous. Un exemple est présenté par la seconde moitié de l'algorithme clock_select() à l'Appendice A.5.5.1.

1. Soit (p, theta_p, psi_p, lambda_p) qui représente un candidat survivant.

2. Trier les candidats par lambda_p croissant. Soit n le nombre de candidats et NMIN le nombre minimum requis de survivants.

3. Pour chaque candidat, calculer la gigue de sélection psi_s :

$$psi\_s = \left(\frac{1}{n-1} \sum_{j=1}^{n-1} (theta\_s - theta\_j)^2 \right)^{1/2}$$

4. Choisir psi_max comme le candidat qui a le psi_s maximum.

5. Choisir psi_min comme le candidat qui a le psi_s minimum.

6. Si psi_max < psi_min ou n ≤ NMIN, suivre l'étape 6A ; sinon, suivre l'étape 6B.

6A. Terminé. Les candidats restants sur la liste des survivant sont rangés dans l'ordre de préférence. La première entrée sur la liste représente l'homologue système ; ses variables seront utilisé ultérieurement pour mettre à jour les variables de système.

6B. Supprimer le candidat marginal qui a psi_max ; réduire n de un et retourner à l'étape 3.

L'algorithme opère par une série de tours où chaque tour élimine le marginal statistique qui a la gigue de sélection maximum psi_s. Cependant, si psi_s est inférieur à la gigue minimum d'homologue psi_p, aucune amélioration n'est possible par l'élimination des marginaux. Cela et le nombre minimum de survivants représentent les conditions terminales de l'algorithme. À la fin, la valeur de psi_max est sauvegardée comme gigue de sélection de système PSI_s pour être utilisée ultérieurement.

11.2.3 Algorithme de combinaison

Le chemin de combinaison d'horloge traite les survivants restants pour produire les données meilleures et finales pour l'algorithme de discipline d'horloge. Le sous programme traite les statistiques de décalage et de gigue d'homologue pour produire le décalage système combiné THETA et la gigue d'homologue système PSI_p, où chaque statistique de serveur est pondérée par la réciproque de la distance de synchronisation de la racine et le résultat est normalisé. Un exemple est présenté par le sous programme clock_combine() à l'Appendice A.5.5.5.

Le THETA combiné est passé au sous programme de mise à jour d'horloge. Le premier candidat sur la liste des survivants est désigné comme homologue du système avec l'identifiant p. La gigue de l'homologue système PSI_p est une composante de la gigue de système PSI. Elle est utilisée avec la gigue de sélection PSI_s pour produire la gigue de système :

PSI = [(PSI_s)^2 + (PSI_p)^2]^1/2

Chaque fois qu'une mise à jour est reçue de l'homologue système, le sous programme de mise à jour de l'horloge est invoqué. La règle est qu'une mise à jour est éliminée si son heure d'arrivée p.t n'est pas strictement plus tardive que la dernière mise à jour utilisée s.t. Les étiquettes IGNOR, PANIC, ADJ, et STEP se réfèrent au codes de retour provenant du sous programme d'horloge locale décrit au paragraphe suivant.

IGNORE signifie que la mise à jour a été ignorée comme marginale. PANIC signifie que le décalage est supérieur au seuil de panique PANICT (1000 s) et DEVRAIT causer la fin du programme avec un message de diagnostic au journal du système. STEP signifie que le décalage est inférieur au seuil de panique, mais supérieur au seuil d'étape STEPT (125 ms). Dans ce cas, l'horloge est réglée au décalage correct, mais comme cela signifie que toutes les données d'homologue ont été invalidées, toutes les associations DOIVENT être réinitialisées et le client recommence au départ initial. ADJ signifie que le décalage est inférieur au seuil d'étape et est donc une mise à jour valide. Dans ce cas, les variables de système sont mises à jour à partir des variables de l'homologue comme le montre la Figure 25.

Variable système	**<--**	**Variable d'homologue système**
s.leap	<--	p.leap
s.stratum	<--	p.stratum + 1
s.décalage	<--	THETA
s.jitter	<--	PSI
s.rootdelay	<--	p.delta_r + delta
s.rootdisp	<--	p.epsilon_r + p.epsilon + p.psi + PHI * (s.t - p.t) + |THETA
s.refid	<--	p.refid
s.reftime	<--	p.reftime
s.t	<--	p.t

Figure 25 : Mise à jour des variables système

Un important détail n'est pas montré. L'incrément de dispersion (p.epsilon + p.psi + PHI * (s.t - p.t) + |THETA|) a une limite inférieure de MINDISP. Dans les sous réseaux et réseaux qui ont des processeurs très rapides et un très faible délai et dispersion cela force à une augmentation monotone définie de s.rootdisp (EPSILON), qui évite des boucles entre les homologues qui opèrent à la même strate.

Les variables de système sont disponibles aux programmes d'application dépendants comme statistiques de performances nominales. Le décalage de système THETA est le décalage d'horloge par rapport aux sources de synchronisation disponibles. La gigue de système PSI est une estimation de l'erreur de détermination de cette valeur, qu'on appelle ailleurs l'erreur attendue. Le délai de racine DELTA est le délai total d'aller-retour par rapport au serveur principal. La dispersion de racine EPSILON est la dispersion accumulée sur le réseau à partir du serveur principal. Finalement, la distance de synchronisation de racine est définie comme : LAMBDA = EPSILON + DELTA / 2, qui représente l'erreur maximum due à toutes les causes et est désignée comme distance de synchronisation de la racine.

Un exemple de sous programme de mise à jour d'horloge est présenté à l'Appendice A.5.5.4.

11.3 Algorithme de discipline d'horloge

L'algorithme de discipline d'horloge NTPv4, abrégé en "discipline" dans ce qui suit, fonctionne comme une combinaison de deux systèmes de contrôle des retours de philosophies assez différentes. Dans la conception de la boucle à verrouillage de phase (PLL, *phase-locked loop*) les mises à jour périodiques de phase aux intervalles de mise à jour de mu secondes sont utilisées directement pour minimiser l'erreur de temps et indirectement l'erreur de fréquence. Dans la conception de la boucle à verrouillage de fréquence (FLL, *frequency-locked loop*) la fréquence périodique de mise à jour aux intervalles mu est utilisée directement pour minimiser l'erreur de fréquence et indirectement l'erreur de temps. Comme le montre [ref7], une PLL fonctionne généralement mieux quand la gigue de réseau domine, tandis qu'une FLL fonctionne mieux quand la fluctuation de l'oscillateur domine. Ce paragraphe contient une présentation de la façon dont fonctionne la conception de NTPv4. Une discussion en profondeur des principes de conception se trouve dans [ref7], qui comporte aussi une analyse des performances.

La discipline est mise en œuvre comme le système de contrôle de retour montré à la Figure 26. La variable theta_r représente le décalage de l'algorithme de combinaison (phase de référence) et theta_c le décalage de VFO (phase de contrôle). Chaque mise à jour produit un signal V_d qui représente la différence instantanée de phase theta_r - theta_c. Le filtre d'horloge de chaque serveur fonctionne comme une ligne de retard en dérivation, avec le résultat sélectionné par l'algorithme de filtre d'horloge pris à la prise. Les algorithmes de sélection, d'amas, et de combinaison combinent les données provenant de multiples filtres pour produire le signal V_s. Le filtre de boucle, avec la réponse d'impulsion F(t), produit le signal V_c, qui contrôle la fréquence de VFO omega_c et donc l'intégralité de la phase theta_c qui clôt la boucle. Le signal V_c est généré par le processus d'ajustement de l'horloge de la Section 12. Les équations détaillées qui mettent en œuvre ces fonctions sont mieux présentées dans les sous programmes des Appendices A.5.5.6 et A.5.6.1.

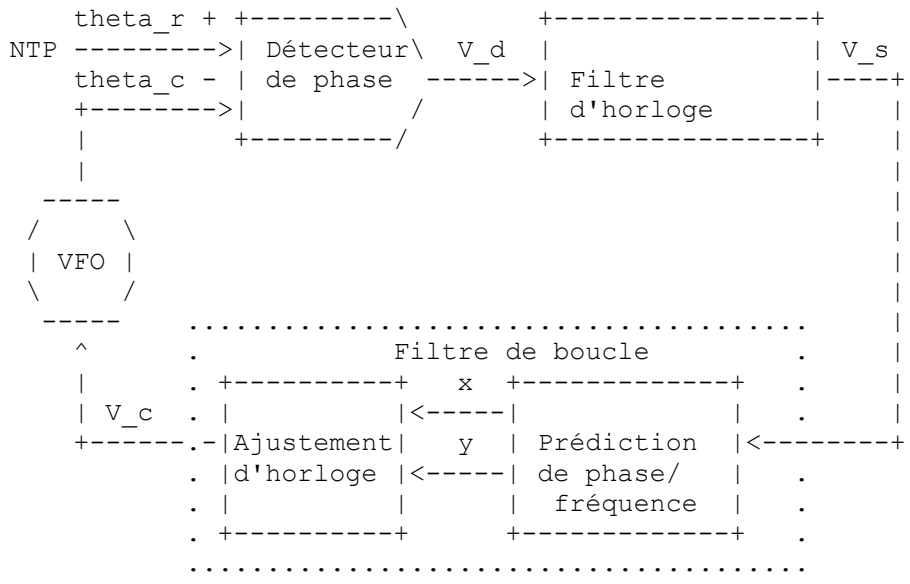


Figure 26 : Boucle de retour de discipline d'horloge

Ordinairement, la boucle de retour pseudo linéaire décrite ci-dessus opère pour discipliner l'horloge système. Cependant, il y a des cas où un algorithme non linéaire offre des améliorations considérables. Un cas est quand la discipline commence sans connaissance de la fréquence d'horloge intrinsèque. La boucle pseudo linéaire prend plusieurs heures pour développer une mesure précise et durant la plus grande partie de ce temps, l'intervalle d'interrogation ne peut pas être augmenté. La boucle non linéaire décrite ci après fait cela en 15 minutes. Un autre cas est quand des salves occasionnelles de grosse gigue sont présentes à cause de l'encombrement des liaisons du réseau. L'automate à états décrit ci-dessous résiste à des salves d'erreur durant moins de 15 minutes.

La Figure 27 contient un résumé des variables et paramètres incluant le nom de la variable (en minuscules) ou du paramètre (en majuscules) le nom de la formule, et une brève description. Sauf notation contraire, toutes les variables ont le préfixe c supposé. Les variables t, tc, état, hyster, et compte sont des entiers ; les variables restantes sont à double flottement. La fonction de chacune va être expliquée ensuite dans les descriptions d'algorithme.

Nom	**Formule**	**Description**
t	timer	compteur de secondes
décalage	theta	décalage combiné
resid	theta_r	décalage résiduel
freq	phi	fréquence d'horloge
gigue	psi	gigue de décalage d'horloge
wander	omega	dérapage de fréquence d'horloge
tc	tau	constante de temps (log2)
état	state	état
adj	adj	ajustement de fréquence
hyster	hyster	compteur d’hystérèse
STEPT	125	seuil de pas (0,125 s)
WATCH	900	seuil de sortie (s)
PANICT	1000	seuil de panique (1000 s)
LIMIT	30	limite d'hystérèse
PGATE	4	portail d'hystérèse
TC	16	échelle de constant de temps
AVG	8	constante de moyenne

Figure 27 : Variables et paramètres de discipline d'horloge

Le processus se termine immédiatement si le décalage est supérieur au seuil de panique PANICT (1000 s). La fonction de transition d'état est décrite par la fonction rstclock() à l'Appendice A.5.5.7. La Figure 28 montre la fonction de transition d'état utilisée par ce sous programme. Elle a quatre colonnes qui montrent, respectivement, le nom de l'état, le prédicat et l'action si le décalage theta est inférieur au seuil d'étape, le prédicat et les actions autrement, et finalement des commentaires.

État	theta < STEP	theta > STEP	Commentaire
NSET	->FREQ	->FREQ	pas de fichier de fréquence
	ajuster l'heure	pas d'heure	
FSET	->SYNC	->SYNC	fichier de fréquence
	ajuster l'heure	pas d'heure	
SPIK	->SYNC	si < 900 s ->SPIK	marginal détecté
	ajuster la fréquence	sinon ->SYNC	
	ajuster l'heure	pas de fréqu., pas d'heure	
FREQ	si < 900 s ->FREQ	si < 900 s ->FREQ	fréquence initiale
	sinon ->SYNC	sinon ->SYNC	
	step freq	pas de fréqu.	
	ajuster l'heure	ajuster l'heure	
SYNC	->SYNC	si < 900 s ->SPIK	fonctionnement normal
	ajuster la fréquence	sinon ->SYNC	
	ajuster l'heure	pas de fréqu.	
		pas d'heure	

Figure 28 : Fonction de transition d'état

Dans les entrées du tableau, l'état suivant est identifié par la flèche -> avec les actions mentionnées en dessous. Les actions comme ajuster l'heure et ajuster la fréquence sont mises en œuvre par la boucle de retour PLL/FLL dans le sous programme local_clock(). Une action "step clock" est mise en œuvre en réglant l'horloge directement, mais ceci n'est fait qu'après le seuil de sortie WATCH (900 s) quand le décalage est supérieur au seuil de pas STEPT (0,125 s). Ceci résiste aux pas d'horloge dans des conditions d'encombrement extrême du réseau.

Les statistiques de gigue (psi) et de dérapage (omega) sont calculées en utilisant une moyenne exponentielle avec le facteur de pondération AVG. L'exposant de constante de temps (tau) est déterminé en comparant psi avec la magnitude du décalage courant theta. Si le décalage est supérieur à PGATE (4) fois la gigue d'horloge, le compteur d'hystérèse hyster est réduit de deux ; autrement, il est augmenté de un. Si hyster augmente jusqu'à la limite supérieure LIMIT (30), tau est augmenté de un ; si il diminue à la limite inférieure -LIMIT (-30), tau est diminué de un. Normalement, tau oscille autour de MAXPOLL, mais diminue rapidement si un pic de température cause une accélération de la fréquence.

12. Processus d'ajustement d'horloge

Le processus réel d'ajustement d'horloge fonctionne à des intervalles de une seconde pour ajouter la correction de fréquence et un pourcentage fixe du décalage résiduel theta_r. theta_r est, en effet, la diminution exponentielle de la valeur theta produite par le filtre de boucle à chaque mise à jour. Le paramètre TC étalonne la constante de temps pour qu'elle corresponde à l'intervalle d'interrogation par convention. Noter que la dispersion EPSILON augmente de PHI à chaque seconde.

Le processus d'ajustement d'horloge inclut une facilité d'interruption de temporisateur qui pilote le compteur de secondes c.t. Il commence à zéro quand commence le service et s'incrémente d'un à chaque seconde. À chaque interruption, le sous programme clock_adjust() est invoqué pour incorporer l'heure de discipline d'horloge et les ajustements de fréquence, puis les associations sont examinées pour déterminer si le compteur de secondes est égal à la variable d'état p.next définie au paragraphe suivant, ou l'excède. Si c'est le cas, le processus d'interrogation est invoqué pour envoyer un paquet et calculer la prochaine valeur de p.next.

Un exemple du processus clock-adjust est donné par le sous programme clock_adjust() à l'Appendice A.5.6.1.

13. Processus d'interrogation

Chaque association prend en charge un processus d'interrogation qui fonctionne à des intervalles réguliers pour construire et envoyer des paquets dans les associations symétriques, de client, et de serveur de diffusion. Il fonctionne en continu, que les serveurs soient ou non joignables afin de gérer le filtre d'horloge et atteindre le registre.

13.1 Variables du processus d'interrogation

La Figure 29 résume les noms communs, les noms de formule, et donne une brève description des variables du processus d'interrogation (en minuscules) et des paramètres (en majuscules). Sauf mention contraire, toutes les variables supposent le préfixe p.

Nom	Formule	Description
hpoll	hpoll	exposant d'interrogation d'hôte
last	last	heure de la dernière interrogation
next	next	prochain instant d'interrogation
reach	reach	registre d'accès
unreach	unreach	compteur de non accès
UNREACH	24	limite de non accès
BCOUNT	8	compteur de salves
BURST	flag	salve activé
IBURST	flag	iburst activé

Figure 29 :Variables et paramètres du processus d'interrogation

Les variables du processus d'interrogation sont allouées dans la structure de données d'association avec les variables de processus d'homologue. Leur description est donnée ci-dessous. Les paramètres vont être invoqués dans le texte qui suit.

hpoll : entier signé qui représente l'exposant d'interrogation, en log2 secondes.

last : entier qui représente le compteur de secondes quand le paquet le plus récent a été envoyé.

next : entier qui représente le compteur de secondes quand le prochain paquet sera envoyé.

reach : registre glissant d'entiers de 8 bits partagé par les processus d'homologue et d'interrogation.

unreach : entier qui représente le nombre de secondes pendant lesquelles le serveur a été injoignable.

13.2 Opérations du processus d'interrogation

Comme décrit précédemment, le processus clock-adjust est invoqué une fois par seconde. Ce sous programme invoque le sous programme d'interrogation pour chaque association tour à tour. Si l'heure du prochain message d'interrogation est supérieure au compteur de secondes, le sous programme revient immédiatement. Les associations symétriques (modes 1, 2), de client (mode 3), et de serveur de diffusion (mode 5) ont une routine d'envoi de paquets. Une association de client de diffusion (mode 6) fait fonctionner le sous programme pour mettre à jour les variables reach et unreach, mais n'envoie pas de paquets. Le processus d'interrogation invoque le processus d'émission pour envoyer un paquet. Si on a une salve (burst > 0), rien d'autre n'est fait excepté d'invoquer le sous programme de mise à jour d'interrogation pour régler l'intervalle de la prochaine interrogation.

Si on n'est pas dans une salve, la variable reach est glissée à gauche d'un bit, un zéro remplaçant le bit de droite. Si le serveur n'a pas été entendu pendant les trois derniers intervalles d'interrogation, le sous programme de filtre d'horloge est invoqué pour augmenter la dispersion. Un exemple est donné à l'Appendice A.5.7.3.

Si le fanion BURST est mis et si le serveur est joignable et si une source de synchronisation valide est disponible, le client envoie une salve de paquets BCOUNT (8) à chaque intervalle d'interrogation. L'intervalle entre les paquets dans la salve est de deux secondes. C'est utile pour mesurer de façon précise la gigue avec de longs intervalles d'interrogation. Si le fanion IBURST est mis et si c'est le premier paquet envoyé quand le serveur a été injoignable, le client envoie une salve. C'est utile pour réduire rapidement la distance de synchronisation en dessous du seuil de distance et synchroniser l'horloge.

Si le fanion P_MANY est allumé dans les mots p.flag de l'association, c'est une association de client de multi envois. Les associations de client de multi envois envoient des paquets en mode client aux adresses de groupes de diffusion groupée désignées à l'intervalle MINPOLL. L'association commence avec un TTL de 1. Si au moment de la prochaine interrogation il y a moins de MINCLOCK serveurs mobilisés, le TTL est augmenté de un. Si le TTL atteint la limite TTLMAX, sans avoir trouvé MINCLOCK serveurs, l'intervalle d'interrogation augmente jusqu'à atteindre BEACON, et il recommence alors depuis le début.

Le sous programme poll() comporte une caractéristique qui diminue l'intervalle d'interrogation si le serveur devient injoignable. Si reach est différent de zéro, le serveur est joignable et unreach est réglé à zéro ; autrement, unreach est incrémenté de un pour chaque interrogation jusqu'au maximum UNREACH. À partir de là, pour chaque interrogation, hpoll est augmenté de un, ce qui double l'intervalle d'interrogation jusqu'au maximum MAXPOLL déterminé par le sous programme poll_update(). Quand le serveur redevient joignable, unreach est réglé à zéro, hpoll est remis à la variable système tc, et le fonctionnement reprend normalement.

Un paquet est envoyé par le processus d'émission. Certaines valeurs d'en-tête sont copiées des variables d'homologue laissées par un paquet précédent et d'autres par les variables de système. La Figure 30 montre quelles valeurs sont copiées dans chaque champ d'en-tête. Dans les mises en œuvre qui utilisent les types de données à double flottement pour le retard et la dispersion de racine, celles-ci doivent être converties en format court NTP. Tous les autres champs sont copiés intacts des variables d'homologue et de système ou collés comme un horodatage provenant de l'horloge système.

Variable de paquet	**<--**	**Variable**
x.leap	<--	s.leap
x.version	<--	s.version
x.mode	<--	s.mode
x.stratum	<--	s.stratum
x.poll	<--	s.poll
x.precision	<--	s.precision
x.rootdelay	<--	s.rootdelay
x.rootdisp	<--	s.rootdisp
x.refid	<--	s.refid
x.reftime	<--	s.reftime
x.org	<--	p.xmt
x.rec	<--	p.dst
x.xmt	<--	clock
x.keyid	<--	p.keyid
x.digest	<--	md5 digest

Figure 30 : En-tête de paquet xmit_packet

Le sous programme de mise à jour d'interrogation est invoqué quand un paquet valide est reçu et qu'immédiatement après a été envoyé un message d'interrogation. Dans une salve, l'intervalle est fixé à 2 s ; autrement, l'exposant d'interrogation d'hôte hpoll est réglé au minimum du ppoll provenant du dernier paquet reçu et du hpoll provenant du sous programme poll, mais pas à moins que MINPOLL ou à plus que MAXPOLL. Donc, la discipline d'horloge peut être sur échantillonnée, mais pas sous échantillonnée. C'est nécessaire pour préserver un comportement dynamique du sous réseau et protéger contre les erreurs de protocole.

L'exposant d'interrogation est converti en un intervalle, qui, ajouté à la variable d'heure de la dernière interrogation, détermine la valeur de la variable prochain instant d'interrogation. Finalement, la variable heure de la dernière interrogation est réglée au compteur de secondes courant.

14. Protocole simple de l'heure du réseau (SNTP)

Les serveurs principaux et clients qui se conforment à un sous ensemble de NTP, appelé le protocole simple d'heure du réseau (SNTPv4, *Simple Network Time Protocol*) [RFC4330], n'ont pas besoin de mettre en œuvre les algorithmes d'atténuation décrits à la Section 9 et aux paragraphes suivants. SNTP est destiné aux serveurs principaux équipés d'une seule horloge de référence, ainsi qu'aux clients qui ont un seul serveur amont et pas de client dépendant. La mise en œuvre pleinement développée de NTPv4 est destinée aux serveurs secondaires avec plusieurs serveurs amont et pluseirurs serveurs ou clients aval. En dehors de ces considérations, les serveurs et clients NTP et SNTP sont complètement interopérables et peuvent être mélangés dans les sous réseaux NTP.

Un serveur principal SNTP qui met en œuvre le protocole en ligne décrit dans la Section 8 n'a pas de serveur amont excepté une seule horloge de référence. En principe, on ne peut pas le distinguer d'un serveur principal NTP qui a les algorithmes d'atténuation et est donc capable de mitiger plusieurs horloges de référence.

À réception d'une demande de client, un serveur principal SNTP construit et envoie le paquet de réponse comme décrit à la Figure 31. Noter que le champ Dispersion dans l'en-tête de paquet doit être mis à jour comme décrit à la Section 5.

Variable de paquet	<--	Variable
x.leap	<--	s.leap
x.version	<--	r.version
x.mode	<--	4
x.stratum	<--	s.stratum
x.poll	<--	r.poll
x.precision	<--	s.precision
x.rootdelay	<--	s.rootdelay
x.rootdisp	<--	s.rootdisp
x.refid	<--	s.refid
x.reftime	<--	s.reftime
x.org	<--	r.xmt
x.rec	<--	r.dst
x.xmt	<--	clock
x.keyid	<--	r.keyid
x.digest	<--	md5 digest

Figure 31 : En-tête de paquet fast_xmit

Un client SNTP qui met en œuvre le protocole en ligne a un seul serveur et pas de clients dépendants. Il peut opérer avec tout sous ensemble du protocole en ligne NTP, la plus simple approche utilisant seulement l'horodatage d'émission du paquet du serveur et ignorant tous les autres champs. Cependant, la complexité supplémentaire pour mettre en œuvre le protocole en ligne complet est minimale de sorte que la mise en œuvre complète est encouragée.

15. Considérations sur la sécurité

Les exigences sur la sécurité de NTP sont encore plus strictes que celles de la plupart des autres services répartis. D'abord, le fonctionnement du mécanisme d'autentification et le mécanisme de synchronisation de l'heure sont inextricablement imbriqués. Une synchronisation fiable de l'heure exige des clés de chiffrement qui ne soient valides que sur un intervalle de temps désigné ; mais, les intervalles de temps ne peuvent être mis en application que quand les serveurs et clients participants sont fiablement synchronisés à l'UTC. De plus, le sous réseau NTP est hiérarchique par nature, de sorte que l'heure et la confiance s'écoulent des serveurs principaux à la racine vers les serveurs secondaires aux clients aux extrémités.

Un client NTP ne peut prétendre avoir l'heure authentique envers les applications dépendantes que si tous les serveurs sur le chemin jusqu'au serveurs principaux sont authentifiés. Dans NTP, chaque serveur authentifie les serveurs de la strate inférieure suivante et authentifie par induction les plus basses strates de serveurs (primaires). Il est important de noter que l'autentification dans le contexte de NTP n'implique pas nécessairement que l'heure est correcte. Un client NTP mobilise un certain nombre d'associations concurrentes avec différents serveurs et utilise un algorithme d'accord ouvré pour tirer les vrais sonneurs d'une population qui peut inclure des faux tiqueurs.

La spécification NTP suppose que le but de l'intrus est d'injecter de fausses valeurs de l'heure, de perturber le protocole, ou géner le réseau, les serveurs, ou les clients avec des paquets parasites qui épuisent les ressources et dénient le service aux applications légitimes. Il y a un certain nombre de mécanismes de défense déjà construits dans l'architecture, le protocole, et les algorithmes de NTP. Le schéma d'échange d'horodatages sur le réseau est par nature résistant à l'usurpation, à la perte de paquet, et aux attaques en répétition. Le filtre d'horloge élaboré, les algorithmes de sélection et d'amas, sont conçus pour défendre contre les cliques malfaisantes de traitres byzantins. Bien que pas nécessairement conçus pour vaincre des intrus déterminés, ces algorithmes et les vérifications de bonne santé qui les accompagnent ont bien fonctionné au fil des ans pour écarter les scénarios au fonctionnement impropre mais présumé amical. Cependant, ces mécanismes n'identifient pas et n'authentifient pas de façon sûre les serveurs aux clients. Sans autre protection spécifique, un intrus peut injecter une des attaques suivantes, ou toutes :

1. Un intrus peut intercepter et archiver des paquets pour toujours, ainsi que toutes les valeurs publiques générées et transmises sur le réseau.
2. Un intrus peut générer des paquets plus vite que le serveur, le réseau ou le client ne peuvent les traiter, en particulier si ils exigent de coûteux calculs cryptographiques.
3. Dans une attaque d'espionnage, l'intrus peut intercepter, modifier, et répéter un paquet. Cependant, il ne peut pas empêcher de façon permanente la transmission vers l'avant du paquet original ; c'est-à-dire que il ne peut pas cassser le réseau, mais seulement mentir et l'encombrer. Généralement, le paquet modifié ne peut pas arriver à la victime avant le paquet original, et il n'a pas les clés privées du serveur ou ses paramètres d'identité.

4. Dans une attaque par interposition ou sous un déguisement, l'intrus se positionne entre le serveur et le client, de sorte qu'il peut intercepter, modifier et répéter un paquet et empêcher la transmission plus avant du paquet original. Cependant, l'interposé n'a pas les clés privées du serveur.

Le modèle de sécurité NTP suppose les limitations possibles suivantes :
1. Les temps d'activité des algorithmes de clé publique sont relativement longs et très variables. En général, les performances de la fonction de synchronisation de l'heure sont très dégradées si ces algorithmes doivent être utilisés pour chaque paquet NTP.
2. Dans certains modes de fonctionnement, il n'est pas possible à un serveur de conserver les variables d'état pour chaque client. Il est cependant faisable de les régénérer pour un client à l'arrivée d'un paquet provenant de ce client.
3. La durée de vie des valeurs cryptographiques doit être mise en application, ce qui exige une horloge système fiable. Cependant, les sources qui synchronisent l'horloge système doivent être de confiance. Cette interdépendance circulaire des fonctions de conservation de l'heure et d'autentification exige un taitement particulier.
4. Les fonctions de sécurité de client doivent impliquer seulement des valeurs publiques transmises sur le réseau. Les valeurs privées ne doiventt jamais être divulguées au delà de la machine sur laquelle elles ont été créées, sauf dans le cas d'un agent de confiance (TA, *trusted agent*) spécial assigné à cette fin.

À la différence du modèle de sécurité Secure Shell (SSH) où le client doit être authentifié en toute sécurité auprès du serveur, dans NTP le serveur doit être authentifié en toute sécurité auprès du client. Dans SSH, chaque adresse d'interface différente peut être liée à un nom différent, comme retourné par une interrogation inverse au DNS. Dans cette conception, des paires de clé séparées publique/privée peuvent être requises pour chaque adresse d'interface avec un nom distinct. Un avantage perceptible de cette conception est que le comportement de sécurité peut être différent pour chaque interface. Cela permet qu'un pare-feu, par exemple, exige que certaines interfaces s'authentifient auprès du client et pas d'autres.

Dans le cas de NTP comme spécifié ici, les clients de diffusion NTP sont vulnérables à des perturbations par des serveurs de diffusion SNTP ou NTP qui se comportent mal ou sont hostiles ailleurs dans l'Internet. Une telle perturbation peut être minimisée par plusieurs approches. Le filtrage peut être employé pour limiter l'accès aux clients NTP connus ou de confiance aux serveurs de diffusion NTP. Un tel filtrage empêchera le trafic malveillant d'atteindre les clients NTP. L'autentification cryptographique chez le client va permettre seulement à des information horaires provenant de messages NTP signés de façon appropriée d'être utilisées pour synchroniser son horloge. De plus hauts niveaux d'autentification peuvent être obtenus par l'utilisation du mécanisme Autokey [RFC5906].

La Section 8 décrit un souci de sécurité potentiel avec la répétition des demandes de client. Suivre les recommandations de cette section protège contre de telles attaques.

On devrait noter que la présente spécification décrit une mise en œuvre existante. Bien que les insuffisances de sécurité de l'algorithme MD5 soient bien connues, son utilisation dans la spécification NTP est cohérente avec son large déploiement dans la communauté de l'Internet.

16. Considérations relatives à l'IANA

L'accès UDP/TCP 123 était précédemment alloué par l'IANA pour ce protocole. L'IANA a alloué l'adresse de groupe de diffusion groupée IPv4 224.0.1.1 et l'adresse IPv6 de diffusion groupée se terminant par :101 pour NTP. Le présent document introduit les champs d'extension NTP qui permettent le développement de futures extensions au protocole, où une extension particulière va être identifiée par le sous champ Type de champ au sein du champ d'extension. L'IANA a établi et va tenir un registre des Types de champ d'extension associés à ce protocole, sans entrées initiales à ce registre. Lorsque de futurs besoins se feront jour, de nouveaus types de champ d'extension pourront être définis. Suivant les politiques décrites dans la [RFC5226], les nouvelles valeurs seront définies par revue de l'IETF.

L'IANA a créé un nouveau registre pour les codes d'identifiant de référence NTP. Cela inclut les codes actuels définis au paragraphe 7.3, et peut être étendu sur la base du premier arrivé premier servi. Le format de ce registre est :

ID	**Source d'horloge**
GOES	Geosynchronous Orbit Environment Satellite
GPS	Global Position System
...	...

Figure 32 : Codes d'identifiant de référence

L'IANA a créé un nouveau registre pour les codes de baiser de la mort NTP. Cela inclut les codes actuels définis au paragraphe 7.4, et peut être étendu sur la base du premier arrivé, premier servi. Le format du registre est :

Code	**Signification**
ACST	L'association apprtient à un serveur d'envoi individuel
AUTH	Échec de l'autentification du serveur
...	..

Figure 33 : Codes de baiser de la mort

Pour les codes d'identifiant de référence et de baiser de la mort, il a été demandé à l'IANA de ne jamais allouer un code commençant par le caractère "X", car c'est réservé pour les expérimentations et le dévelopepment.

17. Remerciements

Les éditeurs tiennent à remercier Karen O'Donoghue, Brian Haberman, Greg Dowd, Mark Elliot, Harlan Stenn, Yaakov Stein, Stewart Bryant, et Danny Mayer de leur révisions techniques et de leur textes de contribution au présent document.

18. Références

18.1 Références normatives

[RFC0768] J. Postel, "Protocole de datagramme d'utilisateur (UDP)", (STD 6), 28 août 1980.

[RFC0791] J. Postel, éd., "Protocole Internet - Spécification du protocole du programme Internet", STD 5, septembre 1981.

[RFC0793] J. Postel (éd.), "Protocole de commande de transmission – Spécification du protocole du programme Internet DARPA", STD 7, septembre 1981.

[RFC1321] R. Rivest, "Algorithme de résumé de message MD5", avril 1992. *(Information)*

[RFC2119] S. Bradner, "Mots clés à utiliser dans les RFC pour indiquer les niveaux d'exigence", BCP 14, mars 1997. *(MàJ par* RFC8174)

18.2 Références pour information

[CGPM] Bureau International des Poids et Mesures, "Comptes Rendus de la 15e CGPM", 1976.

[RFC1305] D. Mills, "Protocole de l'heure du réseau, version 3, spécification, mise en œuvre et analyse", STD 12, mars992. *(Remplacée par* RFC5905)

[RFC1345] K. Simonsen, "Mnémoniques de caractères & et jeux de caractères", juin 1992. *(Information)*

[RFC4330] D. Mills, "Version 4 du protocole simple de l'heure du réseau (SNTP) pour IPv4, IPv6 et OSI", janvier 2006. (*Remplace* RFC2030, RFC1769) *(Information) (Remplacée par* RFC5905)

[RFC5226] T. Narten et H. Alvestrand, "Lignes directrices pour la rédaction d'une section Considérations relatives à l'IANA dans les RFC", BCP 26, mai 2008. (*Remplace* RFC2434 *; remplacée par* RFC8126)

[RFC5906] B. Haberman, D. Mills, "Protocole de l'heure du réseau version 4 (NTPv4) : Spécification du modèle Autokey", juin 2010. *(Information)*

[ref6] Marzullo et S. Owicki, "Maintaining the time in a distributed system", ACM Operating Systems Review 19, July 1985.

[ref7] Mills, D.L., "Computer Network Time Synchronization - the Network Time Protocol", CRC Press, 304 pp, 2006.

[ref9] Mills, D.L., Electrical et Computer Engineering Technical Report 06-6-1, NDSS, June 2006, "Network Time Protocol Version 4 Reference et Implementation Guide", 2006.

[TF.460] Recommandation UIT-R TF.460, "Fréquence standard et émission de signaux horaires", Union Internationale des Télécommunications, février 2002.

Appendice A. Squelette de code

Cet appendice est destiné à décrire le protocole et les algorithmes d'une mise en œuvre d'une façon générale en utilisant ce qu'on appelle un programme squelette de code. Cela consiste en un ensemble de définitions, structures, et fragments de code qui illustrent les opérations du protocole sans les complexités d'une mise en œuvre réelle du protocole. Ce programme n'est pas un exécutable et n'est pas conçu pour fonctionner au sens ordinaire du terme.

La plupart des caractéristiques de la mise en œuvre de référence sont incluses ici, avec les exceptions suivantes : il n'y a pas de disposition sur le chiffrement des horloges de référence ou des clés publiques (Autokey). Il n'y a pas de filtre "huff-n'-puff", d'hystérèse anti saut d'horloge, ou de dispositions de survellance. De nombreuses valeurs qui peuvent être fluctuantes dans la mise en œuvre de référence sont supposées constantes ici. Il y a seulement des dispositions minimales sur le paquet "kiss-o'-death" et pas de code de réponse.

Le programme n'est pas destiné à être rapide ou compact, juste pour montrer les algorithmes avec une fidélité suffisante pour comprendre comment ils fonctionnent. Le squelette de code consiste en huit segments, un segment d'en-tête qu'incluent tous les autres segments, plus un segment de code pour le programme principal, le noyau d'entrée/sortie et les interfaces d'horloge système, et les processus d'homologue, de système, d'ajustement d'horloge, et d'interrogation. Ils sont présentés dans l'ordre ci-dessous avec les définitions et variables spécifiques de chaque processus.

A.1. Définitions globales

A.1.1 Définitions, constantes, paramètres

```
#include <math.h>          /* évite les plaintes sur sqrt() */
#include <sys/time.h>      /* pour gettimeofday() et les amis */
#include <stdlib.h>        /* pour malloc() et les amis */
#include <string.h>        /* pour memset() */

/*  Types de données
    Ce programme suppose que le type de données int est de 32 bits et que le type de données long est de 64 bits. Le type de
    données natives utilisé dans la plupart des calculs est le flottement double. Les types de données utilisés dans certains
    des champs d'en-tête de paquet exigent la conversion de et vers cette représentation. Certains champs d'en-tête
    impliquent de partitionner un octet, ici représenté par des octets individuels.

    Le format d'horodatage NTP de 64 bits utilisé dans les calculs d'horodatage est de secondes non signées et de fraction
    de seconde avec la virgule décimale à gauche du bit 32. La seule opération permise avec ces valeurs est la soustraction,
    donnant une différence signée de 32 bits. Le format court NTP de 32 bits utilisé dans les calculs de délai et de
    dispersion est en secondes et fraction avec la virgule de décimales à la gauche du bit 16. Les seules opérations permises
    avec ces valeurs sont l'addition et la multiplication par une constante.

    L'adresse IPv4 est de 32 bits, tandis que l'adresse IPv6 est de 128 bits. Le champ résumé de message fait 128 bits tel que
    construit par l'algorithme MD5. Les champ Précision et Intervalle d'interrogation sont en secondes log2 signées. */

typedef unsigned long long tstamp;      /* format d'horodatage NTP */
typedef unsigned int tdist;             /* format NTP court */
typedef unsigned long ipaddr;           /* adresse IPv4 ou IPv6 */
typedef unsigned long digest;           /* résumé md5 */
typedef signed char s_char;             /* précision et intervalle d'interrogation (log2) */

/* Conversion macroni d'horodatage */
```

```
#define FRIC        65536.                  /* 2^16 comme un double */
#define D2FP(r)     ((tdist)((r) * FRIC))   /* NTP court */
#define FP2D(r)     ((double)(r) / FRIC)

#define FRAC        4294967296.             /* 2^32 comme un double */
#define D2LFP(a)    ((tstamp)((a) * FRAC))  /* horodatage NTP */
#define LFP2D(a)    ((double)(a) / FRAC)
#define U2LFP(a)    (((unsigned long long) \ ((a).tv_sec + JAN_1970) << 32) + \ (unsigned long long) \ ((a).tv_usec / 1e6 *
            FRAC))

/* Conversions arithmétiques */

#define LOG2D(a)        ((a) < 0 ? 1. / (1L << -(a)) : \ 1L << (a))         /* interrogation, etc. */
#define SQUARE(x)       (x * x)
#define SQRT(x)         (sqrt(x))

/* Constantes globales. Certaines d'entre elles peuvent être converties en variables qui peuvent être modifiées par
   configuration ou calculées au vol. Par exemple, la mise en œuvre de référence calcule PRECISION au vol et fournit des
   réglages de performances pour les "define" marqués d'un % ci-dessous. */

#define VERSION         4       /* numéro de version */
#define MINDISP         .01     /* % minimum de dispersion (s) */
#define MAXDISP         16      /* maximum de dispersion (s) */
#define MAXDIST         1       /* % de seuil de distance (s) */
#define NOSYNC          0x3     /* saut non synchrone */
#define MAXSTRAT        16      /* strate maximum (métrique infinie) */
#define MINPOLL         6       /* % minimum d'intervalle d'interrogation (64 s)*/
#define MAXPOLL         17      /* % maximum d'intervalle d'interrogation (36,4 h) */
#define MINCLOCK        3       /* minimum de survivants de multi envois */
#define MAXCLOCK        10      /* maximum de candidats de multi envois */
#define TTLMAX          8       /* TTL maximum de multi envois */
#define BEACON          15      /* intervalle maximum entre les balises */
#define PHI             15e-6   /* % de tolérance de fréquence (15 ppm) */
#define NSTAGE          8       /* étapes d'enregistrement d'horloge */
#define NMAX            50      /* nombre maximum d'homologues */
#define NSANE           1       /* % minimum d'intersection des survivants */
#define NMIN            3       /* % minimum de survivants dans l'amas */

/* Valeurs de retour globales */

#define TRUE        1           /* "vrai" booléen */
#define FALSE       0           /* "faux" booléen */

/* Codes de retour du processus d'horloge locale */

#define IGNORE      0           /* ignorer */
#define SLEW        1           /* ajustement de stabilisation */
#define STEP        2           /* ajustement de pas */
#define PANIC       3           /* panique - pas d'ajustement */

/* Fanions système */

#define S_FLAGS         0       /* fanions de tout système */
#define S_BCSTENAB      0x1     /* active le client de diffusion */

/* Fanions d'homologue */

#define P_FLAGS         0           /* tous fanions d'homologue */
#define P_EPHEM         0x01        /* l'association est éphémère */
#define P_BURST         0x02        /* salve activée */
#define P_IBURST        0x04        /* salve intiale activée */
```

```
#define P_NOTRUST    0x08          /* accès authentifié */
#define P_NOPEER     0x10          /* mobilisation authentifiée  */
#define P_MANY       0x20          /* client de multi envois */

/* Codes d'authentification */

#define A_NONE       0             /* pas d'autentification */
#define A_OK         1             /* autentification réussie */
#define A_ERROR      2             /* erreur d'autentification */
#define A_CRYPTO     3             /* crypto-NAK */

/* Codes d'état d'association */

#define X_INIT       0             /* initialisation */
#define X_STALE      1             /* fin de temporisation */
#define X_STEP       2             /* pas temporel */
#define X_ERROR      3             /* erreur d'autentification  */
#define X_CRYPTO     4             /* crypto-NAK reçu */
#define X_NKEY       5             /* clé douteuse */

/* Définitions de mode de protocole */

#define M_RSVD       0             /* réservé */
#define M_SACT       1             /* symétrique active */
#define M_PASV       2             /* symétrique passive */
#define M_CLNT       3             /* client */
#define M_SERV       4             /* serveur */
#define M_BCST       5             /* serveur de diffusion */
#define M_BCLN       6             /* client de diffusion */

/* Définitions d'état d'horloge */

#define NSET         0             /* l'horloge n'a jamais été réglée */
#define FSET         1             /* fréquence réglé à partir d'un fichier */
#define SPIK         2             /* pointe détectée */
#define FREQ         3             /* mode de fréquence */
#define SYNC         4             /* horloge synchronisée */

#define min(a, b)   ((a) < (b) ? (a) : (b))
#define max(a, b)   ((a) < (b) ? (b) : (a))
```

A.1.2 Structures de données de paquet

```
/* Les paquets émis et reçus peuvent contenir un code d'authentification de message (MAC, message autentification code)
   facultatif consistant en un identifiant de clé (keyid) et un résumé de message (mac dans la structure de réception et dgst
   dans la structure émise). NTPv4 prend en charge les champs d'extension facultatifs qui sont insérés après l'en-tête et
   avant le MAC, mais ils ne sont pas décrits ici. */

*/ Paquet reçu (Noter que l'horodatage dst ne fait pas partie du paquet lui-même. Il est capturé à l'arrivée et retourné dans la
   mémoire tampon de réception avec la longueur et les données de la mémoire tampon. Noter que cerains des champs
   char sont empaquetés dans l'en-tête réel, mais les détails sont omis ici. */

struct r {
    ipaddr  srcaddr;              /* adresse de source (distante) */
    ipaddr  dstaddr;              /* adresse de destination (locale) */
    char    version;              /* numéro de version */
    char    leap;                 /* indicateur de saut */
    char    mode;                 /* mode */
    char    stratum;              /* strate */
    char    poll;                 /* intervalle d'interrogation */
    s_char  precision;            /* précision */
```

```
        tdist   rootdelay;              /* délai de racine */
        tdist   rootdisp;               /* dispersion de racine */
        char    refid;                  /* identifiant de référence */
        tstamp  reftime;                /* heure de référence  */
        tstamp  org;                    /* horodatage d'origine */
        tstamp  rec;                    /* horodatage de réception */
        tstamp  xmt;                    /* horodatage d'émission */
        int     keyid;                  /* identifiant de clé */
        digest  mac;                    /* résumé de message */
        tstamp  dst;                    /* horodatage de destination */
} r;

/* Paquet émis */

struct x {
        ipaddr  dstaddr;                /* adresse de source (locale) */
        ipaddr  srcaddr;                /* adresse de destination (distante) */
        char    version;                /* numéro de version */
        char    leap;                   /* indicateur de saut */
        char    mode;                   /* mode */
        char    stratum;                /* strate */
        char    poll;                   /* intervalle d'interrogation */
        s_char  precision;              /* précision */
        tdist   rootdelay;              /* délai de racine */
        tdist   rootdisp;               /* dispersion de racine */
        char    refid;                  /* identifiant de référence */
        tstamp  reftime;                /* heure de référence */
        tstamp  org;                    /* horodatage d'origine */
        tstamp  rec;                    /* horodatage de réception */
        tstamp  xmt;                    /* horodatage d'émission */
        int     keyid;                  /* identifiant de clé */
        digest  dgst;                   /* résumé de message */
} x;
```

A.1.3 Structures de données d'association

```
/* Structure d'étape de filtrage : noter que le membre t dans cette structure et les autres se réfère à l'heure de traitement, pas
     à l'heure actuelle. L'heure de traitement s'incrémente d'une seconde à chaque seconde écoulée en temps réel. */

struct f {
        tstamp t;                       /* heure de mise à jour */
        double offset ;                 /* décalage d'horloge */
        double delay;                   /* délai d'aller-retour */
        double disp;                    /* dispersion */
} f;

/* Structure d'association. Ceci est partagé entre le processus d'homologue et le processus d'interrogation. */

struct p {                              /* Variables réglées par configuration */
        ipaddr srcaddr;                 /* adresse de source (distante) */
        ipaddr dstaddr;                 /* adresse de destination (locale) */
        char version;                   /* numéro de version */
        char hmode;                     /* mode d'hôte */
        int keyid;                      /* identifiant de clé */
        int flags;                      /* fanions d'option */
        /*
                                        /* Variables réglées par le paquet reçu */
        char leap;                      /* indicateur de saut */
        char pmode;                     /* mode d'homologue */
        char stratum;                   /* strate */
```

```
        char ppoll;                 /* intervalle d'interrogation d'homologue */
        double rootdelay;           /* délai de racine */
        double rootdisp;            /* dispersion de racine */
        char refid;                 /* identifiant de référence */
        tstamp reftime;             /* heure de référence */
    #define begin_clear org         /* début de la zone en clair */
        tstamp org;                 /* horodatage d'origine */
        tstamp rec;                 /* horodatage de réception */
        tstamp xmt;                 /* horodatage d'émission */

                                    /* Données calculées */
        double  t;                  /* heure de mise à jour */
        struct f f[NSTAGE];         /* filtre d'horloge */
        double offset;              /* décalage de l'homologue*/
        double delay;               /* délai de l'homologue */
        double disp;                /* dispersion de l'homologue */
        double jitter;              /* RMS de la gigue */

                                    /* Variables du processus d'interrogation */
        char hpoll;                 /* intervalle d'interrogation de l'hôte */
        int burst;                  /* compteur de salves */
        int reach;                  /* registre d'accès */
        int ttl;                    /* durée de vie (en multi envois) */
    #define end_clear unreach       /* fin de zone en clair */
        int unreach;                /* compteur d'échec d'accès */
        int outdate;                /* heure de la dernière interrogation */
        int nextdate;               /* prochain instant d'interrogation */
} p;
```

A.1.4 Structures des données système

```
/* Liste d'accords. Ceci est utilisé par l'algorithme d'intersection. */

struct m {                          /* m est pour Marzullo */
        struct p *p;                /* pointeur de structure homologue */
        int type;                   /* haut : +1, moyen : 0, bas : -1 */
        double edge;                /* bord d'intervalle de correction */
} m;

/* Liste de survivants. C'est utilisé par l'algorithme d'amas. */

struct v {
        struct p *p;                /* pointeur de structure homologue */
        double  metric;             /* métrique de tri */
} v;

                                    /* Structure système */
struct s {
        tstamp  t;                  /* heure de mise à jour */
        char leap;                  /* indicateur de saut */
        char stratum;               /* strate */
        char poll;                  /* intervalle d'interrogation*/
        char precision;             /* précision */
        double rootdelay;           /* délai de racine */
        double rootdisp;            /* dispersion de racine */
        char refid;                 /* identifiant de référence */
        tstamp reftime;             /* heure de référence */
        struct m m[NMAX];           /* liste d'accords */
        struct v v[NMAX];           /* liste des survivants */
        struct p *p;                /* identifiant d'association */
        double offset;              /* décalage combiné */
```

```
        double jitter;                  /* gigue combinée */
        int flags;                      /* fanions d'option */
        int  n;                         /* nombre de survivants */
} s;
```

A.1.5 Structures de données d'horloge locale

```
                                        /* Structure de l'horloge locale */
struct c {
        tstamp t;                       /* heure de mise à jour */
        int state;                      /* état actuel */
        double offset;                  /* décalage actuel */
        double last;                    /* décalage antérieur */
        int count;                      /* compteur de sautillements */
        double freq;                    /* fréquence */
        double jitter;                  /* RMS de gigue  */
        double wander;                  /* RMS de dérapage */
} c;
```

A.1.6 Prototypes de fonction

```
                                                /* Processus d'homologue */
oid  receive(struct r *);                       /* paquet en réception */
oid  packet(struct p *, struct r *);            /* traite le paquet */
oid  clock_filter(struct p *, double, double, double);  /* filtre */
ouble root_dist(struct p *);                    /* calcule la distance de la racine */
nt   fit(struct p *);                           /* détermine l'aptitude du serveur */
oid  clear(struct p *, int);                    /* élimine l'association */
nt  access(struct r *);                         /* détermine les restrictions d'accès */

                                               /* Processus système */
nt  main();                                     /* programme principal */
oid clock_select();                             /* trouve les meilleures horloges */
oid clock_update(struct p *);                   /* met à jour l'horloge système */
oid clock_combine();                            /* combine les décalages */

                                               /* Processus d'horloge locale */
nt  local_clock(struct p *, double);            /* discipline d'horloge */
oid  rstclock(int, double, double);             /* transition d'état d'horloge */

                                                /* Processus d'ajustement d'horloge */
oid  clock_adjust();                            /* processus de temporisateur d'une seconde */

                                                /* Processus d'interrogation */
void poll(struct p *);                          /* processus d'interrogation */
void poll_update(struct p *, int);              /* mise à jour de l'intervalle d'interrogation */
void peer_xmit(struct p *);                     /* transmet un paquet */
void fast_xmit(struct r *, int, int);           /* transmet un paquet de réponse */

                                                /* Sous programmes d'utilitaires */
digest  md5(int);                                       /* génére un résumé de message */
struct p *mobilize(ipaddr, ipaddr, int, int, int, int); /* mobilisr */
struct p *find_assoc(struct r *);                       /* fait une recherche dans le tableau d'association */

                                                /* Interface noyau */
struct r *recv_packet();                        /* attente du paquet */
void    xmit_packet(struct x *);                /* envoi du paquet */
void    step_time(double);                      /* pas de temps */
void    adjust_time(double);                    /* ajuste (remplace) l'heure */
tstamp  get_time();                             /* lit l'heure */
```

A.2 Programme principal et sous programmes utilitaires

```
/* Définitions */

#define PRECISION      -18             /* précision (log2 s)  */
#define IPADDR           0             /* toute adresse IP */
#define MODE             0             /* tout mode NTP */
#define KEYID            0             /* tout identifiant de clé */

                                       /* main() - programme principal */
int
main()
{
    struct p *p;                       /* pointeur de structure d'homologue */
    struct r *r;                       /* pointeur de paquet en réception */

/* Lit les options de ligne de commande et initialise les variables de système. La mise en œuvre de référence mesure la
   précision spécifique de chaque machine en mesurant les incréments de l'horloge pour lire l'horloge système. */

    memset(&s, sizeof(s), 0);
    s.leap = NOSYNC;
    s.stratum = MAXSTRAT;
    s.poll = MINPOLL;
    s.precision = PRECISION;
    s.p = NULL;

/* * Initialise les variables de l'horloge locale */

    memset(&c, sizeof(c), 0);
    si (/* frequency file */ 0) {
          c.freq = /* freq */ 0;
          rstclock(FSET, 0, 0);
    } sinon {
          rstclock(NSET, 0, 0);
    }
    c.jitter = LOG2D(s.precision);

/* Lit le fichier de configuration et mobilise les associations persistentes avec les adresses, la version, le mode, l'identifiant
   de clé, et les fanions spécifiés. */

   lorsque (/* mobilize configurated associations */ 0) {
          p = mobilize(IPADDR, IPADDR, VERSION, MODE, KEYID, P_FLAGS);
    }

/* Lance le temporisateur système, qui bat la seconde. Puis, lit les paquets lorsque ils arrivent, donne l'horodatage de
   réception, et invoque le sous programme receive() . */

   lorsque (0) {
          r = recv_packet();
          r->dst = get_time();
          receive(r);
    }

    retour(0);
}

/* mobilize() - mobilise et initialise une association */

struct p
*mobilize(
      ipaddr  srcaddr,                 /* adresse IP de source */
      ipaddr  dstaddr,                 /* adresse IP de destination */
```

```
        int     version,                /* version */
        int     mode,                   /* mode de l'hôte */
        int     keyid,                  /* identifiant de clé */
        int     flags                   /* fanions de l'homologue */
        )
{
        struct p *p;                    /* pointeur sur le processus d'homologue */

/* Alloue et initialise la mémoire d'association */

        p = malloc(sizeof(struct p));
        p->srcaddr = srcaddr;
        p->dstaddr = dstaddr;
        p->version = version;
        p->hmode = mode;
        p->keyid = keyid;
        p->hpoll = MINPOLL;
        clear(p, X_INIT);
        p->flags = flags;
        retour (p);
}

/* find_assoc() - trouve une association corrrespondante */

struct p                                /* pointeur sur la structure d'homologue ou NUL */
*find_assoc(
        struct r *r                     /* pointeur de paquet en réception */
        )
{
        struct p *p;                    /* pointeur de structure d'homologue factice */

/* Parcourt le tableau d'association à la recherche d'une adresse de source, accès de source et mode correspondants. */

        lorsque (/* toutes associations */ 0) {
                si (r->srcaddr == p->srcaddr && r->mode == p->hmode)
                        retour(p);
        }

    retour (NULL);
}

/* md5() - calcule le résumé de message */

digest
md5(
      int     keyid                     /* identifiant de clé */
      )
{
/*    Calcule un résumé de messagea cryptographique chiffré. L'identifiant de clé est associé à une clé qui est dans
      l'antémémoire de clé locale. La clé est ajoutée devant l'en-tête de paquet et les champs d'extension et le résultat est
      haché par l'algorithme MD5 comme décrit dans la RFC 1321. Cela retourne un MAC consistant en l'identifiant de clé
      de 32 bits enchaîné avec le résumé de 128 bits. */

    retour (/* résumé MD5 */ 0);
}
```

A.3 Interface entrée/sortie de noyau

/* L'interface du noyau pour transmettre et recevoir les paquets. Les détails sont délibérement vagues et dépendent du système d'exploitation. recv_packet - reçoit le paquet provenant du réseau */

```
  struct r                              /* pointeur de paquet en réception */
  *recv_packet() {
        retour (/* receive packet r */ 0);
  }

/* xmit_packet - paquet transmis au réseau */

  void
  xmit_packet(
        struct x *x                     /* pointeur de paquet en émission */
        )
  {                                     /* envoie le paquet x */
}
```

A.4 Interface d'horloge système de noyau

```
/* Fonctions d'utilitaire du système d'horloge. Il y a trois formats d'heure : natif (Unix), NTP, et double flottement. Le sous
     programe get_time() retourne l'heure en format NTP long. Les sous programmes Unix attendent les arguments comme
     une structure de deux mots signés de 32 bits en secondes et microsecondes (timeval) ou nanosecondes (timespec). Les
     sous programmes step_time() et adjust_time() attendent des arguments signés en double flottement. Le code simplifié
     qu'on donne ici est seurment à des fins d'illustration et n'a pas été vérifié. */

#define JAN_1970  2208988800UL          /* 1970 - 1900 en secondes */

/*
 * get_time - lit l'heure système et la convertit au format NTP */

tstamp
get_time()
{
      struct timeval unix_time;

/* Il y a seulement deux appels sur des sous programme dans un programme. Un quand un paquet arrive du réseau et l'autre
    quand un paquet est placé dans la file d'attente d'envoi. Invoque l'heure du noyau (comme gettimeofday()) et la convertit
    au format NTP . */

      gettimeofday(&unix_time, NULL);
      retour (U2LFP(unix_time));
}

/*
 * step_time() - règle l'heure système à la valeur de décalage donnée */

void
step_time(
      double offset                     /* décalage d'horloge */
      )
{
      struct timeval unix_time;
      tstamp  ntp_time;

/* Convertit du format au format natif (signé) et l'ajoute à l'heure actuelle. Noter que l'addition est faite en format natif pour
     éviter une surcharge ou une perte de précision. */

      gettimeofday(&unix_time, NULL);
      ntp_time = D2LFP(offset) + U2LFP(unix_time);
      unix_time.tv_sec = ntp_time >> 32;
      unix_time.tv_usec = (long)(((ntp_time - unix_time.tv_sec) << 32) / FRAC * 1e6);
      settimeofday(&unix_time, NULL);
}
```

```
/*
 * adjust_time() - règle l'horloge système avec la valeur de décalage donnée */

void
adjust_time(
      double offset                     /* décalage d'horloge */
      )
{
      struct timeval unix_time;
      tstamp  ntp_time;

/* Convertit du format double au format natif (signé) et l'ajoute à l'heure actuelle. */

      ntp_time = D2LFP(offset);
      unix_time.tv_sec = ntp_time >> 32;
      unix_time.tv_usec = (long)(((ntp_time - unix_time.tv_sec) << 32) / FRAC * 1e6);
      adjtime(&unix_time, NULL);
}
```

A.5 Processus d'homologue

```
/* Un paquet crypto-NAK comporte l'en-tête NTP suivi par un MAC consistant en seulement l'identifiant de clé de valeur
     zéro. Cela dit au receveur qu'une demande précédente n'a pas pu être correctement authentifiée, mais les champs d'en-
     tête NTP sont corrects. *
* Un paquet "baiser de la mort" est un en-tête NTP avec un saut 0x3 (NOSYNC) et la strate 16 (MAXSTRAT). Il dit au
     receveur que quelque chose de drastique est survenu, comme l'indique le kiss code dans le champ refid. Les champs
     d'en-tête NTP peuvent ou non être corrects. */

/* Paramètres et constantes de processus d'homologue */

  #define SGATE       3             /* spike gate (filtre d'horloge) */
  #define BDELAY      0,004         /* délai de diffusion (s) */

/* Codes de répartition */

#define ERR          -1             /* erreur */
#define DSCRD        0              /* éliminer le paquet */
#define PROC         1              /* traiter le paquet */
#define BCST         2              /* diffuser le paquet */
#define FXMIT        3              /* paquet client */
#define MANY         4              /* paquet en multi envoi */
#define NEWPS        5              /* nouveau client passeif symétrique */
#define NEWBC        6              /* nouveau client de diffusion */

/* Matrice de répartition active  passv  client server bcast */
  int table[7][5] = {
  /* nopeer  */    { NEWPS, DSCRD, FXMIT,  MANY, NEWBC },
  /* active  */    { PROC,    PROC,    DSCRD, DSCRD, DSCRD },
  /* passv   */    { PROC,    ERR,     DSCRD, DSCRD, DSCRD },
  /* client  */    { DSCRD, DSCRD, DSCRD, PROC,   DSCRD },
  /* server  */    { DSCRD, DSCRD, DSCRD, DSCRD, DSCRD },
  /* bcast   */    { DSCRD, DSCRD,  DSCRD, DSCRD, DSCRD },
  /* bclient */    { DSCRD, DSCRD, DSCRD, DSCRD, PROC    }
  };

/* Macros diverses. Cette macro définit l'état d'autentification. Si x est 0, l'autentification est facultative ; autrement elle est
     exigée.  */

#define AUTH(x, y)    ((x) ? (y) == A_OK : (y) == A_OK || \ (y) == A_NONE)
```

```
/* Ceci est utilisé par le sous programme clear() */

#define BEGIN_CLEAR(p)  ((char *)&((p)->begin_clear))
#define END_CLEAR(p)    ((char *)&((p)->end_clear))
#define LEN_CLEAR       (END_CLEAR((struct p *)0) - \ BEGIN_CLEAR((struct p *)0))
```

A.5.1 receive()

```
/* receive() - reçoit le paquet et décode les modes */

void
receive(
        struct r *r                     /* reçoit le pointeur de paquet */
        )
{
struct p *p;                            /* pointeur de structure d'homologue */
int     auth;                           /* code d'authentification */
int     has_mac;                        /* taille du MAC */
int     synch;                          /* commutateur synchronisé */

/* Vérification des listes de contrôle d'accès. L'intention est ici de mettre en œuvre une liste des adresses IP spécifiquement
       acceptées et/ou une liste noire de celles qui sont spécifiquement rejetées. Ce pourrait être des listes différentes pour les
       clients authentifiés et les clients non authentifiés. */

    si (!access(r))
            retour;                     /* accès refusé */

/* La version ne doit pas être dans le futur. Les vérifications de format incluent la longueur de paquet, la longueur de MAC
       et les longueurs des champs d'extension, si il en est de présents. */

    si (r->version > VERSION            /* ou erreur de format */)
            retour;                     /* erreur de format */

/* L'authentification est conditionnée par deux commutateurs qui peuvent être spécifiés client par client :
* P_NOPEER : ne pas mobiliser une association sauf si authentifiée.
* P_NOTRUST : ne pas permettre l'accès sauf si authentifié (implique P_NOPEER).
* Il y a quatre résultats :
* A_NONE : le paquet n'a pas de MAC.
* A_OK : le paquet a un MAC et l'autentification réussit.
* A_ERROR : le paquet a un MAC et l'autentification échoue.
* A_CRYPTO : crypto-NAK. Le MAC a seulement quatre octets.
* Note : La macro AUTH (x, y) est utilisée pour filtrer les résultats. Si x est zéro, les résultats acceptables de y sont NONE
       et OK. Si x est un, le seul résultat acceptable de y est OK. */

    has_mac = /* longueur du champ MAC */ 0;
    si (has_mac == 0) {
            auth = A_NONE;              /* non exigé */
      } sinon si (has_mac == 4) {
            auth = A_CRYPTO;            /* crypto-NAK */
      } sinon {
            si (r->mac != md5(r->keyid))
                  auth = A_ERROR;       /* erreur d'authentification */
            sinon
                  auth = A_OK;          /* authentification OK */
      }

/*   Trouve l'association et le code de répartition. Si il n'y a pas d'association qui corresponde, on suppose que la valeur de
       p->hmode est NULL. */

p = find_assoc(r);
       switch(table[(unsigned int)(p->hmode)][(unsigned int)(r->mode)])
```

```
    {

/* Paquet client et pas d'association. Envoie la réponse du serveur sans sauvegarder l'état. */

cas FXMIT :

/* Si l'adresse de destination est en envoi individuel, envoi du paquet serveur. Si l'autentification échoue, envoi d'un paquet
    crypto-NAK. */

/* pas d'adresse de destination de diffusion groupée */

    si (0) {
        si (AUTH(p->flags & P_NOTRUST, auth)) fast_xmit(r, M_SERV, auth);
            sinon si (auth == A_ERROR)
                fast_xmit(r, M_SERV, A_CRYPTO);
            retour;                         /* paquet M_SERV envoyé */
    }

/* Ceci doit être en multi envois. Ne pas répondre si on n'est pas synchronisé ou si la strate est au dessus du multi envoyeur.
*/

    si (s.leap == NOSYNC || s.stratum > r->stratum)
        retour;

/* Ne répond que si l'autentification est réussie. Noter que c'est l'adresse d'envoi individuel qui est utilisée, pas celle de
    diffusion groupée. */

    si (AUTH(p->flags & P_NOTRUST, auth)) fast_xmit(r, M_SERV, auth);
        retour;

/* Nouvelle association éphémère de client de multi envois. Elle est mobilisée dans la même version que dans le paquet. Si
    l'autentification échoue, ignorer le paquet. Vérifier le paquet serveur en comparant l'horodatage r->org dans le paquet
    avec l'horodatage p->xmt dans l'association de client de diffusion groupée. Si ils correspondent, le paquet serveur est
    authentique. On omet les détails. */

cas MANY :
    si (!AUTH(p->flags & (P_NOTRUST | P_NOPEER), auth))
        retour;                                         /* erreur d'autentification */

    p = mobilize(r->srcaddr, r->dstaddr, r->version, M_CLNT,
      r->keyid, P_EPHEM);
    break;

/*  Nouvelle association symétrique passive. Elle est mobilisée dans la même version que dans le paquet. Si
    l'authentification échoue, envoyer un paquet crypto-NAK. Si il y a la restriction "no-moblize", envoyer à la place un
    paquet symétrique actif. */

cas NEWPS :
    si (!AUTH(p->flags & P_NOTRUST, auth)) {
        si (auth == A_ERROR)
            fast_xmit(r, M_SACT, A_CRYPTO);
        retour;                         /* paquet crypto-NAK envoyé */
    }
    si (!AUTH(p->flags & P_NOPEER, auth)) {
        fast_xmit(r, M_SACT, auth);
        retour;                         /* paquet M_SACT envoyé */
    }
    p = mobilize(r->srcaddr, r->dstaddr, r->version, M_PASV,
      r->keyid, P_EPHEM);
    break;
```

```
/* Nouvelle association de client de diffusion. Elle est mobilisée dans la même version que dans le paquet. Si
   l'autentification échoue, ignorer le paquet. Noter que ce code ne prend pas en charge la caractéristique initiale de salve
   dans la mise en œuvre de référence. */

cas NEWBC :
   si (!AUTH(p->flags & (P_NOTRUST | P_NOPEER), auth))
      retour;                          /* erreur d'autentification */

   si (!(s.flags & S_BCSTENAB))
      retour;                          /* diffusion non activée */

   p = mobilize(r->srcaddr, r->dstaddr, r->version, M_BCLN,
     r->keyid, P_EPHEM);
   break;                              /* le traitement continue */

/* Processus de paquet. Seulement bouche trou. */

cas PROC :
      break;                           /* le traitement continue */

/* Combinaison de modes invalide. Cela ne se produit que dans le cas d'associations éphémères, de sorte que l'action
   correcte est simplement la jeter. */

cas ERR :
      clear(p, X_ERROR);
      retour;                          /* combinaison de modes invalide */

/*Pas de correspondance ; juste éliminer le paquet. */

case DSCRD :
      retour;                          /* orphelin abandonné */
   }

/*Ensuite vient une suite rigoureuse de vérifications d'horodatage. Si l'horodatage d'émission est zéro, le serveur est en très
   mauvais état. */
   si (r->xmt == 0)
      retour;                          /* horodatage invalide */

/* Si l'horodatage d'émission duplique un horodatage antérieur, le paquet est une répétition. */

   si (r->xmt == p->xmt)
      retour;                          /* paquet dupliqué */

/* Si c'est un paquet en mode diffusion, sauter les autres vérifications. Si l'horodatage d'origine est zéro, l'envoyeur n'a pas
   encore entendu parler de nous. Autrement, si l'horodatage d'origine ne correspond pas à l'horodatage d'émission, le
   paquet est bogué. */

   synch = TRUE;
   si (r->mode != M_BCST) {
      si (r->org == 0)
         synch = FALSE;                /* non synchronisé */

      sinon si (r->org != p->xmt)
         synch = FALSE;  /             * paquet bogué */
   }

/* Mettre à jour les  horodatages d'origine et de destination. Si ils ne sont pas synchronisés ou si ils sont erronés,
   abandonner le navire. */

   p->org = r->xmt;
   p->rec = r->dst;
```

```
        si (!synch)
            retour;                         /* non synchronisé */

/* Les horodatages sont valides et le paquet reçu correspond au dernier envoyé. Si le paquet est un crypto-NAK, le serveur
   pourrait juste avoir changé les clés. On démobilise les associations et on attend des temps meilleurs. */

        si (auth == A_CRYPTO) {
            clear(p, X_CRYPTO);
            retour;                         /* crypto-NAK */
        }

/* Si l'association est authentifiée, l'identifiant de clé est non zéro et les paquets reçus doivent être authentifiés. C'est conçu
   pour éviter une attaque d'appâtage et commutation (bait-and-switch), qui était possible dans les anciennes versions. */

        si (!AUTH(p->keyid || (p->flags & P_NOTRUST), auth))
            retour;                         /* mauvaise authentification */

/* Tout ce qui est possible a été fait pour valider les horodatages et empêcher des malveillants de perturber le protocole ou
   d'injecter des données boguées. On en tire profit. */

        packet(p, r);
}
```

A.5.1.1 packet()

```
/* packet() - traite le paquet et calcule le décalage, le délai, et la dispersion. */

void
packet(
        struct p *p,                        /* pointeur sur la structure d'homologue */
        struct r *r                         /* pointeur sur le paquet reçu */
        )
{
        double  offset;                     /* décalage de l'échantillont */
        double  delay;                      /* délai de l'échantillon */
        double  disp;                       /* dispersion de l'échantillon */

/* Par chance, le paquet est valide. On s'occupe alors des champs d'en-tête restants. Noter qu'on transpose la strate 0 (non
   spécifié) en MAXSTRAT pour rendre plus simples les comparaisons de strates et fournir une interface naturelle pour
   les pilotes d'horloge radio qui opèrent par facilité à la strate 0. */

        p->leap = r->leap;
        si (r->stratum == 0)
            p->stratum = MAXSTRAT;
        sinon
            p->stratum = r->stratum;
        p->pmode = r->mode;
        p->ppoll = r->poll;
        p->rootdelay = FP2D(r->rootdelay);
        p->rootdisp = FP2D(r->rootdisp);
        p->refid = r->refid;
        p->reftime = r->reftime;

/* Vérifie que le serveur est synchronisé avec une strate valide et que l'heure de référence n'est pas plus tardive que l'heure
   d'émission. */

        si (p->leap == NOSYNC || p->stratum >= MAXSTRAT)
            retour;                         /* non synchronisée */

/* Vérifie que la distance de la racine est valide. */
```

```
        si (r->rootdelay / 2 + r->rootdisp >= MAXDISP || p->reftime >
           r->xmt)
              retour;                           /* valeurs d'en-tête invalides */

        poll_update(p, p->hpoll);
        p->reach |= 1;

/* Calcule le décalage, le délai et la dispersion, puis les passe au filtre d'horloge. Noter avec soin le traitement que cela
   implique. La différence de premier ordre est faite directement en arithmétique de 64 bitsc, puis le résultat est converti
   en double flottement. Tout le reste du traitement est en arithmétique de double flottement avec les arrondis faits par le
   matériel. Ceci est nécessaire afin d'éviter le débordement et préserver la précision.
   Le calcul du délai est un cas particulier. Dans les cas où les horloges de serveur et de client fonctionnent à des
   fréquences différentes et avec des réseaux très rapides, le délai peut apparaître négatif. Afin d'éviter de violer le
   principe du moindre étonnement, le délai est fixé à pas moins que la précision du système. */

        si (p->pmode == M_BCST) {
              offset = LFP2D(r->xmt - r->dst);
              delay = BDELAY;
              disp = LOG2D(r->precision) + LOG2D(s.precision) + PHI *
                 2 * BDELAY;
        } sinon {
              offset = (LFP2D(r->rec - r->org) + LFP2D(r->dst -
                 r->xmt)) / 2;
              delay = max(LFP2D(r->dst - r->org) - LFP2D(r->rec -
                 r->xmt), LOG2D(s.precision));
              disp = LOG2D(r->precision) + LOG2D(s.precision) + PHI *
                 LFP2D(r->dst - r->org);
        }
        clock_filter(p, offset, delay, disp);
}
```

A.5.2 clock_filter()

```
/* clock_filter(p, offset, delay, dispersion) - choisit le meilleur des huit derniers échantillons de délai/décalage. */

void
clock_filter(
        struct p *p,                        /* pointeur de structure d'homologue */
        double offset,                      /* décalage d'horloge */
        double  delay,                      /* délai d'aller-retour */
        double  disp                        /* dispersion */
        )
{
        struct f f[NSTAGE];                  /* liste triée */
        double  dtemp;
        int     i;

/* Le contenu du filtre d'horloge consiste en huit quartets (décalage, délai, dispersion, heure). Glisser chaque quartet vers la
   gauche, et éliminer le plus à gauche. Lorsque chaque quartet est déplacé, augmenter la dispersion depuis la dernière
   mise à jour du filtre. En même temps, copier chaque quartet dans une liste temporaire. Après cela, placer le quartet
   (décalage, délai, dispersion, heure) dans le quartet vacant le plus à droite. */

        pour (i = 1; i < NSTAGE; i++) {
              p->f[i] = p->f[i - 1];
              p->f[i].disp += PHI * (c.t - p->t);
              f[i] = p->f[i];
        }
        p->f[0].t = c.t;
        p->f[0]. offset = offset;
        p->f[0].delay = delay;
        p->f[0].disp = disp;
```

```
        f[0] = p->f[0];

/* Trier la liste temporaire des quartets en augmentant f[].delay. La première entrée de la liste triée représente le meilleur
      échantillon, mais il pourrait être vieux.*/

        dtemp = p->offset;
        p->décalage = f[0].offset ;
        p->delay = f[0].delay;
        for (i = 0; i < NSTAGE; i++) {
                p->disp += f[i].disp / (2 ^ (i + 1));
                p->jitter += SQUARE(f[i]. offset - f[0].offset );
        }
        p->jitter = max(SQRT(p->jitter), LOG2D(s.precision));

/* Principale directive : utiliser un échantillon une seule fois et jamais un échantillon plus vieux que le dernier, mais tout
      passe avant le premier synchronisé. */

        si (f[0].t - p->t <= 0 && s.leap != NOSYNC)
                retour;

/* Suppresseur de pointes. Compare la différence entre le dernier et le décalage courant de la gigue actuelle. Si elle est
      supérieure à SGATE (3) et si l'intervalle depuis le dernier décalage est inférieur à deux fois l'intervalle d'interrogation
      du système, écrase la pointe. Autrement, et si on n'est pas dans une salve, sort les vrais sonneurs. */

        si (fabs(p-> offset - dtemp) > SGATE * p->jitter && (f[0].t -
           p->t) < 2 * s.poll)
                retour;
        p->t = f[0].t;
        si (p->burst == 0)
                clock_select();
        retour;
}

/* fit() - vérifie si l'association p est acceptable pour la synchronisation */

int
fit(
        struct p *p                             /* pointeur de structure d'homologue */
        )
{
/* Une erreur de strate se produit si (1) le serveur n'a jamais été synchronisé, (2) la strate du serveur est invalide. */

        si (p->leap == NOSYNC || p->stratum >= MAXSTRAT)
                retour (FAUX);

/* Une erreur de distance se produit si la distance de la racine excède le seuil de distance plus un incrément égal à un
      intervalle d'interrogation. */

        si (root_dist(p) > MAXDIST + PHI * LOG2D(s.poll))
                retour (FAUX);

/* Une erreur de boucle se produit si l'homologue distant est synchronisé à l'homologue local ou si l'homologue local est
      synchronisé à l'homologue système actuel. Noter que ceci est le comportement pour IPv4 ; pour IPv6 le hachage MD5
      est utilisé à la place. */

        si (p->refid == p->dstaddr || p->refid == s.refid)
                retour (FAUX);

/* Une erreur injoignable se produit si le serveur est injoignable. */

        si (p->reach == 0)
```

```
            retour (FAUX);

        retour (VRAI);
}

/* clear() - réinitialise pour l'association persistente, démobilise pour l'association éphémère. */

void
clear(
        struct p *p,                    /* pointeur de structure d'homologue */
        int     kiss                    /* kiss code */
        )
{
        int i;

/* La première chose à faire est de retourner toutes les ressources à la banque. Les ressources normales ne sont pas
   détaillées ici, mais elles incluent des strutures allouées de façon dynamique pour les clés, les certificats, etc. Si c'est
   une association et non une initialisation, retourner aussi la mémoire d'association. */

/* Retour des ressources */
        si (s.p == p)
                s.p = NULL;
        si (kiss != X_INIT && (p->flags & P_EPHEM)) {
                free(p);
                retour;
        }

/* Initialiser les champs d'association pour la réinitialisation générale. */

        memset(BEGIN_CLEAR(p), LEN_CLEAR, 0);
        p->leap = NOSYNC;
        p->stratum = MAXSTRAT;
        p->ppoll = MAXPOLL;
        p->hpoll = MINPOLL;
        p->disp = MAXDISP;
        p->jitter = LOG2D(s.precision);
        p->refid = kiss;
        pour (i = 0; i < NSTAGE; i++)
                p->f[i].disp = MAXDISP;

/* Rend aléatoire la première interrogation pour le cas où des milliers de clients de diffusion se réveilleraient juste après
   une longue absence auprès du serveur de diffusion. */

        p->outdate = p->t = c.t;
        p->nextdate = p->outdate + (random() & ((1 << MINPOLL) - 1));
}
```

A.5.3 fast_xmit()

```
/* fast_xmit() - transmet un paquet de réponse pour un paquet reçu r */

void
fast_xmit(
        struct r *r,                    /* pointeur de paquet reçu */
        int  mode,                      /* mode d'association */
        int  auth                       /* code d'autentification */
        )
{
        struct x x;
```

```
/* Initialise l'en-tête et l'horodatage d'émission. Noter que la version transmise est copiée de la version reçue. C'est pour la
      rétro compatibilité. */

       x.version = r->version;
       x.srcaddr = r->dstaddr;
       x.dstaddr = r->srcaddr;
       x.leap = s.leap;
       x.mode = mode;
       si (s.stratum == MAXSTRAT)
             x.stratum = 0;
       sinon
             x.stratum = s.stratum;
       x.poll = r->poll;
       x.precision = s.precision;
       x.rootdelay = D2FP(s.rootdelay);
       x.rootdisp = D2FP(s.rootdisp);
       x.refid = s.refid;
       x.reftime = s.reftime;
       x.org = r->xmt;
       x.rec = r->dst;
       x.xmt = get_time();

/* Si le code d'autentification est A.NONE, inclut seulement l'en-tête ; si A.CRYPTO, envoie un crypto-NAK ; si A.OK,
      envoie un MAC valide. Utilise l'identifiant de clé dans le paquet reçu et la clé dans l'antémémoire locale de clé. */

       si (auth != A_NONE) {
             si (auth == A_CRYPTO) {
                   x.keyid = 0;
             } sinon {
                   x.keyid = r->keyid;
                   x.dgst = md5(x.keyid);
             }
       }
       xmit_packet(&x);
}
```

A.5.4 access()

```
/* access() - détermine les restrictions d'accès */

 int
 access(
       struct r *r                          /* pointeur de paquet en réception */
       )

 {
/* La liste de contrôle d'accès est un ensemble ordonné de tuplets consistant en une adresse, un gabarit, et un mot restreint
      contenant des bits définis. La liste est parcourue à la recherche d'une première correspondance sur l'adresse de source
      (r->srcaddr) et le mot restreint associé est retourné. */

       retour (/* bits d'accès */ 0);
 }
```

A.5.5 Processus système

A.5.5.1 clock_select()

```
/* clock_select() - trouve les meilleures horloges */

void
clock_select() {
```

```
    struct p *p, *osys;             /* pointeurs de structure homologue */
    double  low, high;              /* correction de l'extension d'intervalle */
    int     allow, found, chime;    /* utilisé par l'algorithme d'intersection */
    int     n, i, j;
```

/* On retire d'abord les faux tiqueurs de la population de serveurs, laissant seulement les vrais sonneurs. L'intervalle de correction pour l'association p est l'intervalle du décalage - root_dist() au décalage + root_dist(). Le but du jeu est de trouver une clique majoritaire ; c'est-à-dire, une intersection d'intervalles de correction comptant plus de la moitié de la population de serveurs.
D'abord, on construit la liste de triplets (p, type, edge) comme montré ci-dessous, puis on trie la liste par bord de bas en haut. */

```
osys = s.p;
s.p = NULL;
n = 0;
lorsque (fit(p)) {
    s.m[n].p = p;
    s.m[n].type = +1;
    s.m[n].edge = p->offset + root_dist(p);
    n++;
    s.m[n].p = p;
    s.m[n].type = 0;
    s.m[n].edge = p->offset;
    n++;
    s.m[n].p = p;
    s.m[n].type = -1;
    s.m[n].edge = p-> offset - root_dist(p);
    n++;
}
```

/* Trouve la plus grande intersection contiguë d'intervalles de correction. "Allow" est le nombre de faux tiqueurs permis ; "found" est le nombre de points moyens. Noter que les valeurs bordures sont limitées à la gamme +-(2 ^ 30) < +-2e9 par les calculs d'horodatage. */

```
    low = 2e9; high = -2e9;
    pour (allow = 0; 2 * allow < n; allow++) {
```

/* Examine la liste des sonneurs du plus bas au plus haut pour trouver le point d'extrémité inférieur. */

```
    found = 0;
    chime = 0;
    pour (i = 0; i < n; i++) {
        chime -= s.m[i].type;
        si (chime >= n - found) {
            low = s.m[i].edge;
            break;
        }
        si (s.m[i].type == 0)
            found++;
    }
```

/* Examine la liste des sonneurs du haut en bas pour trouver le point d'extrémité supérieur. */

```
    chime = 0;
    pour (i = n - 1; i >= 0; i--) {
        chime += s.m[i].type;
        si (chime >= n - found) {
            high = s.m[i].edge;
            break;
        }
        si (s.m[i].type == 0)
```

```
                found++;
        }
```

/*Si le nombre de points moyens est supérieur au nombre permis de faux tiqueurs, l'intersection contient au moins un vrai sonneur sans point moyen. Si c'est vrai, incrémente le nombre de faux tiqueurs admis et fait un nouveau tour. Sinon et si l'intersection n'est pas vide, déclare le succès. */

```
        si (found > allow)
                continue;
        si (high > low)
                break;
 }
```

/* Algorithme d'amas. Construit une liste des survivants (p, * métrique) à partir de la liste des sonneurs, où la métrique est dominée d'abord par la strate et ensuite par la distance de la racine. Toutes choses égales par ailleurs, c'est l'ordre de préférence. */

```
        s.n = 0;
        pour (i = 0; i < n; i++) {
                si (s.m[i].edge < low || s.m[i].edge > high)
                        continue;
                p = s.m[i].p;
                s.v[n].p = p;
                s.v[n].metric = MAXDIST * p->stratum + root_dist(p);
                s.n++;
 }
```

/* Il doit y avoir au moins NSANE survivants pour satisfaire les assertions de correction. Ordinairement, le critère byzantin exige quatre survivants, mais pour notre démonstration, un est accepttable. */

```
        si (s.n < NSANE)
                retour;
```

/* Pour chaque association p tour à tour, calcule la gigue de sélection p->sjitter comme la racine carrée de la somme des carrés de (p->offset - q->offset) sur toutes les q associations. L'idée est d'éliminer répétivement le survivant avec la gigue de sélection maximum jusq'à ce qu'une condition de terminaison soit satisfaite. */

```
        lorsque (1) {
                struct p *p, *q, *qmax;         /* pointeurs de structure d'homologue */
                double  max, min, dtemp;
                max = -2e9; min = 2e9;
                pour (i = 0; i < s.n; i++) {
                        p = s.v[i].p;
                        si (p->jitter < min)
                                min = p->jitter;
                        dtemp = 0;
                        pour (j = 0; j < n; j++) {
                                q = s.v[j].p;
                                dtemp += SQUARE(p-> offset - q->offset);
                        }
                        dtemp = SQRT(dtemp);
                        si (dtemp > max) {
                                max = dtemp;
                                qmax = q;
                        }
                }
```

/*Si la gigue maximum de sélection est inférieure à la gigue minimum d'homologue, éliminer plus de survivants ne va pas diminuer la gigue minimum d'homologue, de sorte qu'on peut aussi bien arrêter. Pour s'assurer que quelques survivants restent pour l'algorithme d'amas, on arrête aussi si le nombre de survivants est inférieur ou égal à NMIN (3). */

```
        si (max < min || n <= NMIN)
            break;

/*Supprime qmax survivants de la liste et fait un nouveau tour. */

        s.n--;
    }

/* Prend la meilleure horloge. Si le vieil homologue système est sur la liste et à la même strate que le premier survivant sur
   la liste, ne pas faire de saut d'horloge. Autrement, choisir le premier survivant de la liste comme nouvel homologue
   système. */

    si (osys->stratum == s.v[0].p->stratum)
        s.p = osys;
    sinon
        s.p = s.v[0].p;
    clock_update(s.p);
}
```

A.5.5. root_dist()

```
/* root_dist() - calcule la distance à la racine. */

double
root_dist(
    struct p *p                         /* pointeur de structure d'homologue */
    )
{

/* La distance de synchronisation à la racine est l'erreur maximum due à toutes les causes de l'horloge locale par rapport au
   serveur principal. Elle est définie comme la moitié du délai total plus la dispersion totale plus la gigue d'homologue. */

    retour (max(MINDISP, p->rootdelay + p->delay) / 2 + p->rootdisp + p->disp + PHI * (c.t - p->t) + p->jitter);
}
```

A.5.5.3 accept()

```
/* accept() - vérifie si l'association p est acceptable pour la synchronisation. */

int
accept(
    struct p *p                         /* pointeur de structure d'homologue */
    )
{
/* Une erreur de strate se produit si (1) le serveur n'a jamais été synchronisé, (2) la strate du serveur est invalide. */

    si (p->leap == NOSYNC || p->stratum >= MAXSTRAT)
        retour (FAUX);

/* Une erreur de distance se produit si la distance racine excède le seuil de distance plus un incrément égal à un intervalle
   d'interrogation. */

    si (root_dist(p) > MAXDIST + PHI * LOG2D(s.poll))
        retour (FAUX);

/* Une erreur de boucle se produit si l'homologue distant est synchronisé à l'homologue local ou si l'homologue distant est
   synchronisé à l'homologue système actuel. Noter que c'est le comportement pour IPv4 ; on utilise le hachage MD5
   pour IPv6. */

    si (p->refid == p->dstaddr || p->refid == s.refid)
        retour (FAUX);
```

```
/* Une erreur injoignable se produit si le serveur est injoignable. */

        si (p->reach == 0)
                retour (FAUX);
        retour (VRAI);
}
```

A.5.5.4 clock_update()

```
/* clock_update() - met à jour l'horloge système. */

void
clock_update(
        struct p *p                             /* pointeur de structure d'homologue */
        )
{
        double dtemp;

/*  Si c'est une vieille mise à jour, par exemple, résultant d'un changement d'homologue système, éviter la. On utilise jamais
    un vieil échantillon ou deux fois le même échantillon. */

        si (s.t >= p->t)
                retour;

/* Combine les décalages de survivant et met à jour l'horloge système ; le sous programme local_clock() va donner les
    bonnes ou mauvaises nouvelles. */

        s.t = p->t;
        clock_combine();
        switch (local_clock(p, s.offset )) {

/* Le décalage est trop grand et probablement bogué. Enregistrer une plainte dans le journal système et ordonner à
    l'opérateur de régler l'horloge manuellement dans la gamme PANIC. La mise en œuvre de référence inclut une option
    de ligne de commande pour désactiver cette vérification et changer le seuil de panique de la valeur par défaut de 1000 s
    à celle requise. */

cas PANIC :
                exit (0);

/* Le décalage est supérieur au seuil de pas (0,125 s par défaut). Après un pas, toutes les associations ont maintenant des
    valeurs d'heure incohérentes, de sorte qu'elles sont réinitialisées et repartent. Le seuil de pas peut être changé dans la
    mise en œuvre de référence afin de diminuer les chances que l'horloge puisse aller à reculons. Cependant, cela peut
    avoir de sérieuses conséquences, comme noté dans les papiers du site du projet NTP. */

cas STEP :
                lorsque (/* toutes les associations */ 0)
                        clear(p, X_STEP);
                s.stratum = MAXSTRAT;
                s.poll = MINPOLL;
                break;

/* Le décalage était inférieur au seuil de pas, ce qui est le cas normal. Mettre à jour les variables de système à partir des
    variables de l'homologue. L'augmentation de la pince inférieure de la dispersion est pour éviter les boucles horaires et
    le saut d'horloge lorsque des sources très précises sont en jeu. La pince peut être changée de la valeur par défaut de
    0,01 s dans la mise en œuvre de référence. */

cas SLEW :
                s.leap = p->leap;
                s.stratum = p->stratum + 1;
                s.refid = p->refid;
```

```
            s.reftime = p->reftime;
            s.rootdelay = p->rootdelay + p->delay;
            dtemp = SQRT(SQUARE(p->jitter) + SQUARE(s.jitter));
            dtemp += max(p->disp + PHI * (c.t - p->t) +
               fabs(p->offset), MINDISP);
            s.rootdisp = p->rootdisp + dtemp;
            break;

/* Certains échantillons sont éliminés lorsque, par exemple, une mesure de fréquence directe est faite. */

cas IGNORE :
            break;
      }
}
```

A.5.5.5 clock_combine()

```
/* clock_combine() - combine les décalages. /*
void
clock_combine()
{
      struct p *p;                        /* pointeur de structure d'homologue */
      double x, y, z, w;
      int     i;

/* Combine les décalages des survivants de l'algorithme d'amas en utilisant une moyenne pondérée avec la pondération
      déterminée par la distance à la racine. Calcule la gigue de sélection comme la différence d'écart quadratique moyen
      pondéré entre le premier survivant et les survivants restants. Dans certains cas, la gigue d'horloge inhérente peut être
      réduite en n'utilisant pas cet algorithme, en particulier quand de fréquents sauts d'horloge sont impliqués. La mise en
      œuvre de référence peut être configurée pour éviter cet algorithme en désignant un homologue préféré. */

      y = z = w = 0;
      pour (i = 0; s.v[i].p != NULL; i++) {
            p = s.v[i].p;
            x = root_dist(p);
            y += 1 / x;
            z += p-> offset / x;
            w += SQUARE(p-> offset - s.v[0].p->offset) / x;
      }
      s. offset = z / y;
      s.jitter = SQRT(w / y);
}
```

A.5.5.6 local_clock()

```
/* Paramètres et constantes de discipline d'horloge. */
#define STEPT          0,128        /* seuil de pas (s) */
#define WATCH          900          /* seuil de sortie (s) */
#define PANICT         1000         /* seuil de panique (s) */
#define PLL            65536        /* gain de boucle en PLL */
#define FLL        AXPOLL + 1       /* gain de boucle en FLL */
#define AVG            4            /* constante de moyenne de paramètre */
#define ALLAN          1500         /* compromis d'interception d'Allan (s) */
#define LIMIT          30           /* seuil d'ajustement d'interrogation */
#define MAXFREQ        500e-6       /* tolérance de fréquence (500 ppm) */
#define PGATE          4            /* portail d'ajustement d'interrogation */

/* local_clock() - discipline l'horloge locale */
int                                 /* code de retour */
local_clock(
      struct p *p,                  /* pointeur de structure d'homologue */
```

```
        double  offset                  /* décalage d'horloge d'après combine() */
        )
{
        int     state;                  /* état de discipline d'horloge */
        double  freq;                   /* fréquence */
        double  mu;                     /* intervalle depuis la dernière mise à jour */
        int     rval;
        double  etemp, dtemp;           /* Si le décalage est trop grand, abandonner. */
        si (fabs(offset) > PANICT)
                retour (PANIC);

/* Fonction de transition de l'automate à états de l'horloge. C'est là qu'est l'action et qu'est défini comment le système réagit
   aux grosses erreurs d'heure et de fréquence. Il y a deux régimes principaux : quand le décalage excède le seuil de pas et
   quand il ne le fait pas. */

        rval = SLEW;
        mu = p->t - s.t;
        freq = 0;
        si (fabs(offset) > STEPT) {
                switch (c.state) {

/* Dans l'état S_SYNC on ignore le premier point aberrant et on passe à l'état S_SPIK. */
cas SYNC :
        state = SPIK;
          retour (rval);

/* Dans l'état S_FREQ, on ignore les aberrants et les enclavés. Au premier aberrant après le seuil de sortie, calcule la
   correction de fréquence apparente et règle l'heure. */
cas FREQ :
          si (mu < WATCH)
                retour (IGNORE);
          freq = ( offset - c.offset ) / mu;          /* retombe à S_SPIK */

/* Dans l'état S_SPIK, on ignore les aberrants réussis jusqu'à ce qu'on trouve un enclavé ou que le seuil de sortie soit
   excédé. */
cas  SPIK :
        si (mu < WATCH)
                retour (IGNORE);                /* revient à la valeur par défaut. */

 /* On passe ici par défaut dans les états S_NSET et S_FSET et à partir de ci-dessus dans l'état S_FREQ. Arrêter l'heure et
    diminuer l'intervalle d'interrogation.*
    Dans l'état S_NSET, une correction de fréquence initiale n'est pas disponible, généralement parce que le fichier de
    fréquence n'a pas encore été écrit. Comme l'heure est hors de la gamme de capture, l'horloge est arrêtée. La fréquence
    va être réglée diirectement à la suite de l'intervalle de sortie. *
    Dans l'état S_FSET, la fréquence initiale a été règlée d'après le fichier de fréquence. Comme l'heure est en dehors de la
    gamme de capture, l'horloge est arrêtee immédiatement, plutôt qu'après l'intervalle de sortie. Les gens s'énervent si il
    faut 17 minutes pour régler l'horloge pour la première fois.
    Dans l'état S_SPIK, le seuil de sortie a expiré et la phase est toujours au dessus du seuil de sortie. Noter qu'une seule
    pointe supérieure au seuil de sortie est toujours supprimée, même aux plus long intervalles d'interrogation. */

         default:

/* C'est la fonction d'établissement de l'heure du noyau, généralement mise en œuvre par l'invocation du système Unix
   settimeofday(). */

          step_time(offset);
          c.count = 0;
          s.poll = MINPOLL;
          rval = STEP;
          si (state == NSET) {
          rstclock(FREQ, p->t, 0);
```

```
        retour (rval);
        }
        break;
       }
       rstclock(SYNC, p->t, 0);
  } sinon {
```

/* Calcule la gigue d'horloge comme le RMS des différences de décalage à pondération exponentielle. C'est utilisé par le code d'ajustement d'interrogation. */

```
       etemp = SQUARE(c.jitter);
       dtemp = SQUARE(max(fabs(offset - c.last),
          LOG2D(s.precision)));
       c.jitter = SQRT(etemp + (dtemp - etemp) / AVG);
       switch (c.state) {
```

/* Dans l'état S_NSET, c'est la première mise à jour reçue et la fréquence n'a pas encore été initialisée. La première chose à faire est de mesurer directement la fréquence de l'oscillateur. */

```
cas NSET :
                    rstclock(FREQ, p->t, offset);
                    retour (IGNORE);
```

/* Dans l'état S_FSET, c'est la première mise à jour et la fréquence a été initialisée. Ajuste la phase, mais n'ajuste pas la fréquence jusqu'à la prochaine mise à jour. */

```
cas FSET :
                    rstclock(SYNC, p->t, offset);
                    break;
```

/* Dans l'état S_FREQ, ignore les mises à jour jusqu'au seuil de sortie. Après cela, corrige la phase et la fréquence et passe à l'état S_SYNC. */

```
cas FREQ :
                    si (c.t - s.t < WATCH)
                          retour (IGNORE);
                    freq = ( offset - c.offset ) / mu;
                    break;
```

/* On passe ici par défaut dans les états S_SYNC et S_SPIK. Ici, on calcule la mise à jour de fréquence due aux contributions PLL et FLL. */

```
       default:
```

/* Les constantes de gain de fréquence FLL et PLL selon l'intervalle d'interrogation et l'interception d'Allan. Le FLL n'est pas utilisé en-dessous de la moitié de l'interception d'Allan. Au dessus de cela le gain de boucle augmente par pas de 1 / AVG. */

```
        si (LOG2D(s.poll) > ALLAN / 2) {
             etemp = FLL - s.poll;
             si (etemp < AVG)
                  etemp = AVG;
              freq += (d offset - c.offset ) / (max(mu, ALLAN) * etemp);
       }
```

/* Pour le PLL l'intervalle d'intégration (numérateur) est le minimum de l'intervalle de mise à jour et de l'intervalle d'interrogation. Cela permet le sur échantillonnage, mais pas le sous échantillonnage. */

```
        etemp = min(mu, LOG2D(s.poll));
        dtemp = 4 * PLL * LOG2D(s.poll);
        freq += offset * etemp / (dtemp * dtemp);
```

```
        rstclock(SYNC, p->t, offset);
        break;
    }
  }
```

/* Calcule la nouvelle fréquence et la stabilité de fréquence (dérapage). Calcule le dérapage d'horloge comme le RMS des différences de fréquences à pondération exponentielle. Ce n'est pas utilisé directement, mais peut, avec la gigue, être un outil extrêment utile de surveillance et de débogage. */

```
    freq += c.freq;
    c.freq = max(min(MAXFREQ, freq), -MAXFREQ);
    etemp = SQUARE(c.wander);
    dtemp = SQUARE(freq);
    c.wander = SQRT(etemp + (dtemp - etemp) / AVG);
```

/* Ici, on ajuste l'intervalle d'interrogation en comparant le décalage actuel à la gigue d'horloge. Si le décalage est inférieur à la gigue d'horloge fois une constante, l'intervalle de moyenne est augmenté ; sinon, il est diminué. Un peu d'hystérèse calme la danse. Cela fonctionne mieux en utilisant le mode salve. */

```
    si (fabs(c.offset ) < PGATE * c.jitter) {
        c.count += s.poll;
        si (c.count > LIMIT) {
            c.count = LIMIT;
            si (s.poll < MAXPOLL) {
                c.count = 0;
                s.poll++;
            }
        }
    } sinon {
        c.count -= s.poll << 1;
        si (c.count < -LIMIT) {
            c.count = -LIMIT;
            si (s.poll > MINPOLL) {
                c.count = 0;
                s.poll--;
            }
        }
    }
    retour (rval);
}
```

A.5.5.7 rstclock()

```
/*  rstclock() - automate à états d'horloge.  */
 void
 rstclock(
      int     state,                    /* nouvel état */
      double  offset,                   /* nouveau décalage */
      double  t                         /* nouvelle heure de mise à jour */
      )
 {
```

/* Entre dans le nouvel état et règle les variables d'état. Noter qu'on utilise l'heure du dernier échantillon de filtre d'horloge, qui doit être plus tôt que l'heure actuelle. */

```
      c.state = state;
      c.last = c. offset = offset;
      s.t = t;
 }
```

A.5.6 Processus d'ajustement d'horloge

A.5.6.1 clock_adjust()

```
/*  clock_adjust() - fonctionne à des intervalles de une seconde. */
 void
 clock_adjust() {
        double  dtemp;

/* Met à jour l'heure de traitement c.t. Augmente aussi la dispersion depuis la dernière mise à jour. À la différence de
      NTPv3, NTPv4 ne déclare pas la non synchronisation après un jour, car le seuil de dispersion assure cette fonction.
      Quand la dispersion excède MAXDIST (1 s) le serveur est considéré comme inapte à la synchronisation. */

        c.t++;
        s.rootdisp += PHI;

/* Met en œuvre les ajustements de phase et de fréquence. Le facteur de gain (dénominateur) n'est pas autorisé à augmenter
      au delà de l'interception de Allan. Il n'y a pas de sens à faire la moyenne du bruit de phase au delà de ce point et cela
      aide à atténuer le décalage résiduel aux plus longs intervalles d'interrogation. */

        dtemp = c. offset / (PLL * min(LOG2D(s.poll), ALLAN));
        c.offset -= dtemp;

/* C'est la fonction de temps d'ajustement du noyau, généralement mise en œuvre par l'invocation du système Unix
      adjtime(). */

        adjust_time(c.freq + dtemp);

/* Temporisateur d'homologue. Invoque le sous programme poll() quand le temporisateur d'interogation arrive à expiration.
      */

        lorsque (/* toutes les associations */ 0) {
              struct p *p;                      /* pointeur factice de structure d'homologue */
              si (c.t >= p->nextdate)
                    poll(p);
        }

/* Une fois par heure, écrit la fréquence d'horloge dans un fichier. */
/* si (c.t % 3600 == 3599) écrit c.freq dans le fichier */
 }
```

A.5.7 Processus d'interrogation

```
/* Paramètres et constantes du processus d'interrogation*/

#define UNREACH      12          /* seuil dn compteur de non joints */
#define BCOUNT        8          /* paquets dans une salve */
#define BTIME         2          /* intervalle de salve (s) */
```

A.5.7.1 poll()

```
/* poll() - détermine quand envoyer un paquet pour l'association p-> */

void
poll(
      struct p *p                       /* pointeur de structure d'homologue */
      )
{
      int    hpoll;
      int    oreach;
```

```
/* Ce sous programme est invoqué quand l'heure actuelle c.t attrape le prochain instant d'interrogation p->nextdate. La
   valeur p->outdate est le dernier moment où ce sous programme a été exécuté. Le sous programme poll_update()
   détermine le prochain instant d'exécution p->nextdate. En diffusion, il n'est fait que si il y a synchronisation. */

hpoll = p->hpoll;
si (p->hmode == M_BCST) {
      p->outdate = c.t;
      si (s.p != NULL)
            peer_xmit(p);
      poll_update(p, hpoll);
      retour;
}

/* En multi envois, commence avec ttl = 1. Le ttl est augmenté de un pour chaque interrogation jusqu'à ce que
   MAXCLOCK serveurs aient été trouvés ou que ttl atteigne TTLMAX. Si MAXCLOCK, est atteint, on arrête
   d'interroger jusqu'à ce que le nombre de serveurs tombe en dessous de MINCLOCK, puis on recommence tout. */

si (p->hmode == M_CLNT && p->flags & P_MANY) {
      p->outdate = c.t;
      si (p->unreach > BEACON) {
            p->unreach = 0;
            p->ttl = 1;
            peer_xmit(p);
      } sinon si (s.n < MINCLOCK) {
            si (p->ttl < TTLMAX)
                  p->ttl++;
            peer_xmit(p);
      }

     p->unreach++;
      poll_update(p, hpoll);
      retour;
}
si (p->burst == 0) {

/* On n'est pas dans une salve. On glisse le registre d'accessibilité à gauche. Heureusement, un peu avant la prochaine
   interrogation un paquet va arriver et établir à un le bit de droite. */

      oreach = p->reach;
      p->outdate = c.t;
      p->reach = p->reach << 1;
      si (!(p->reach & 0x7))
            clock_filter(p, 0, 0, MAXDISP);
      si (!p->reach) {

/* Le serveur est injoignable, donc cela incrémente le compteur d'inaccessibilité. Si le seuil d'inacessibilité a été atteint,
   doubler l'intervalle d'interrogation pour minimiser la consommation de trafic sur le réseau. N'envoyer une salve qui si
   c'est activé et si le seuil d'inaccessibilité n'a pas été atteint. */

      si (p->flags & P_IBURST && p->unreach == 0) {
            p->burst = BCOUNT;
         } sinon si (p->unreach < UNREACH)
            p->unreach++;
           sinon
                hpoll++;
            p->unreach++;
   } sinon {

/* Le serveur est joignable. Régler l'intervalle d'interrogation à l'intervalle d'interrogation du système. N'envoyer une salve
   que si c'est activé et si l'homologue convient. */
```

```
            p->unreach = 0;
            hpoll = s.poll;
              si (p->flags & P_BURST && fit(p))
                  p->burst = BCOUNT;
            }
      } sinon {

/*Si dans une salve, la décompter. Quand la réponse revient le sous programme clock_filter() va invoquer clock_select()
      pour traiter les résultats de la salve. */
        p->burst--;
      }
/* Ne pas transmettre si en mode client de diffusion. */
       si (p->hmode != M_BCLN)
             peer_xmit(p);
       poll_update(p, hpoll);
}
```

A.5.7.2. poll_update()

```
/* poll_update() - met à jour l'intervalle d'interrogation pour l'association p.
Note : Ce sous programme est invoqué par les deux sous programmes packet() et poll(). Comme le sous programme
      packet() est exécuté quand un paquet réseau arrive et que le sous programme poll() est exécuté en résultat de la fin de
      temporisation, il peut en résulter une potentielle compétition, causant éventuellement un intervalle incorrect pour la
      prochaine interrogation. Ceci est considéré tellement peu probable que c'est négligeable. */
void
poll_update(
    struct p *p,                        /* pointeur de structure d'homologue */
    int     poll                        /* intervalle d'interrogation (log2 s) */
    )
{
/* Ce sous programme est invoqué par les deux sous programmes poll() et packet() pour déterminer le prochain instant
      d'interrogation. Si c'est au sein d'une salve, l'intervalle d'interrogation est de deux secondes. Autrement, c'est le
      minimum de l'intervalle d'interrogation de l'hôte et de l'intervalle d'interrogation de l'homologue, mais pas supérieur à
      MAXPOLL et pas inférieur à MINPOLL. Le concept assure qu'un intervalle plus long peut être préempté par un plus
      court si c'est necessaire pour une réponse rapide. */
  p->hpoll = max(min(MAXPOLL, poll), MINPOLL);
    si (p->burst > 0) {
          si (p->nextdate != c.t)
                retour;
          sinon
                p->nextdate += BTIME;
    } sinon {

/* Bien que ce ne soit pas montré ici, la mise en œuvre de référence rend aléatoire l'intervalle d'interrogation par un petit
      facteur. */

          p->nextdate = p->outdate + (1 << max(min(p->ppoll, p->hpoll), MINPOLL));
    }

/* Il se pourrait que le bon moment soit passé. Si c'est le cas, on le place à une seconde dans le futur. */
    si (p->nextdate <= c.t)
          p->nextdate = c.t + 1;
}
```

A.5.7.3. peer_xmit()

```
/* transmit() - émet un paquet pour l'association p */

void
peer_xmit(
      struct p *p                       /* pointeur de structure d'homologue */
```

```
        )
{
        struct x x;                     /* paquet transmis */

/* Initialise l'en-tête et l'horodatage d'émission. */
    x.srcaddr = p->dstaddr;
    x.dstaddr = p->srcaddr;
    x.leap = s.leap;
    x.version = p->version;
    x.mode = p->hmode;
    si (s.stratum == MAXSTRAT)
            x.stratum = 0;
    sinon
            x.stratum = s.stratum;
    x.poll = p->hpoll;
    x.precision = s.precision;
    x.rootdelay = D2FP(s.rootdelay);
    x.rootdisp = D2FP(s.rootdisp);
    x.refid = s.refid;
    x.reftime = s.reftime;
    x.org = p->org;
    x.rec = p->rec;
    x.xmt = get_time();
    p->xmt = x.xmt;

/*Si l'identifiant de clé est non zéro, envoie un MAC valide en utilisant l'identifiant de clé de l'association et la clé dans
     l'antémémoire de clé locale. Si quelque chose ne va pas, comme l'absence d'une clé de confiance, ne pas envoyer le
     paquet ; réinitialiser juste l'association et arrêter jusqu'à ce que le problème soit résolu. */

    si (p->keyid)
            si (/* p->keyid invalid */ 0) {
                    clear(p, X_NKEY);
                    retour;
            }
            x.dgst = md5(p->keyid);
    xmit_packet(&x);
}
```

Adresse des auteurs

Dr. David L. Mills
University of Delaware
Newark, DE 19716
US
téléphone : +1 302 831 8247
mél : mills@udel.edu

Jim Martin (editor)
Internet Systems Consortium
950 Charter Street
Redwood City, CA 94063
téléphone : +1 650 423 1378
mél : jrmii@isc.org

Jack Burbank
Johns Hopkins University Applied Physics Lab
11100 Johns Hopkins Road
Laurel, MD 20723-6099
téléphone : +1 443 778 7127
mél : jack.burbank@jhuapl.edu

William Kasch
Johns Hopkins University Applied Physics Lab
11100 Johns Hopkins Road
Laurel, MD 20723-6099
US
téléphone : +1 443 778 7463
mél : william.kasch@jhuapl.edu