NTP メモ

2007 年 4 月 13 日

本作品はクリエイティブ・コモンズ・ライセンスの下でラインセンスされています。
あなたは以下の条件に従う場合に限り、自由に

- 本作品を複製、頒布、展示、上演、演奏、口述、上映、公衆送信することができます。
- 二次的著作物を作成することができます。

あなたの従うべき条件は以下の通りです。

非営利. あなたはこの作品を営利目的で利用してはなりません。

同一条件許諾. もしあなたがこの作品を改変、変形または加工した場合、あなたはその結果生じた作品をこの作品と同一の許諾条件の下でのみ頒布することができます。

- 再利用や頒布にあたっては、この作品の使用許諾条件を他の人々に明らかにしなければなりません。
- この作品の権利者から許可を得ると、これらの条件は適用されません。

上記によって、著作権等に対する制限条項に基づく利用その他あなたの権利が影響を受けることはまったくありません。

目 次

はじめに

このメモは, インターネットの標準時刻同期プロトコルである NTP(Network Time Protocol) の参考資料になるべく書かれたものである. 本メモには記述が中途半端に終わっている部分や , 筆者の貧弱な知識で正しく理解していない部分が多々あることを御了承願いたい. 内容に間違いや修正すべき箇所があれば, 筆者まで御連絡頂きたい.

2007 年 4 月 13 日

thotta@gmail.com

第1章　時間について

1.1　時計の原理

まず, コンピュータで取り扱う時間について説明する前に簡単な時計の構造を説明する. 日時計を除くほとんどの時計は大きく分けて二つの部分から構成される (参考文献 [1]). 一つは発振器で周期的な振動を起こす部品である. 昔は振り子やバネの伸び縮みを利用していたが, 現在の主流は水晶や原子の振動を利用している. もう一つは計数器で, 発振器から規則正しい振動が得られれば, 振動している拍子の数を正確に数える事で振動数を時間に変換することができる (図 1.1).

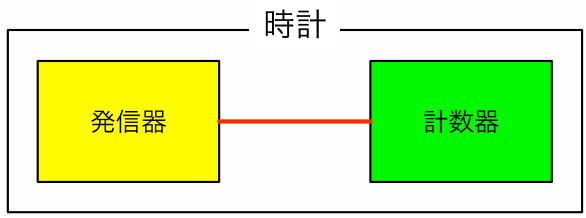

図 1.1: 時計の根本原理

従って, 正確な時刻を得るという事は, 精度の高い振動 (パルス信号や周波数) を安定して生成するという事に置き換えることができる. 現在, 精度の高いパルス信号 (クロック) を得る方法としては, 鉱物や原子の振動 (発振) を利用するものがほとんどで, 発振器は PLL/FLL 回路によって, 正確な標準周波数やパルス信号を生成できる.

1.2　時間の正確さについて

時計の正確さを表すためにいくつかの独特の用語があるので, それを織り交ぜながら説明していくことにする. 昔の時計は概ね 1 秒単位に時刻を刻んでいたが, 今のデジタル時計は 1 秒よりももっと細かく時を刻む事ができる. この刻む単位が小さければ小さいほど, 時間を細かく測定する事ができる. これを**分解能**(resolution) と呼んでいる.

分解能が高いというだけでは, 時間が正確とは言えない. 時間は一定間隔で時を刻む必要がある. それを数値として表したものが刻み精度 (precision) と呼ばれるもので, ある一定の期間でどの程度時間がずれるかを示す. ちなみに, NTP の精度は自動的に求められ, 2 の冪乗マイクロ秒で示される. “`ntpq -c rl`” と実行し, “`precision=`−16” と表示されれば, このサーバの刻み精度は約 15 マイクロ秒 (2^{-16} 秒) という事になる.

時間を繰り返し読んでいくと, 非常に正確な時計と比較して時間がずれている事がわかる. この差は通常ランダムに表れ, 差の量もまちまちである. 短い時間での時刻の揺れを**ジッタ**と呼んでいる. 一方, 長い期間での揺れもある. これを**ワンダ**と呼んでいる. 時計 (クロック) にはジッタは悪いがワンダは良いもの, その逆も有り得る. 当然, ジッタやワンダは小さければ小さい程良い.

正確な時計にするには, リファレンスとなる時刻 (UTC) に如何に近付けるかである. この事を**正確度**(accuracy) と呼んでいる. ところが残念な事に多くの時計はあまり正確ではない. 例えば, 1 日で 1 秒狂う時計があるとしたら, それは一日で 0.001%狂っている事になる. これは発振器の中心発振周波数のズレ (変動範囲) を示しており, 時計やクロックの正確さを表している. この単位を小数点で表すと見にくいので, **PPM**(Part Per Million: 百万分の 1) という単位が用いられる. 1PPM というのは 0.0001% (10^{-6}) となる. 例えば, 44.1kHz の発振周波数で ±50ppm の精度というのは, $44100 \times 50 \times 10^{-6} = 2.205$Hz で, 44097.795Hz 〜44102.205Hz の範囲に周波数が収まっていなければならない事を意味する.

以上, 時間の正確さを示す用語を説明してきたが, たとえシステムに内在するエラーが無くなったとしても, 時計が完璧に正確になるとは限らない. 現実には水晶や原子の振動が温度, 気圧, 磁場等の外部環境の影響を受けてしまうため, 振動は揺らいでしまう. なお, 指定された精度をある環境下でどの程度維持できるかを**信頼性** (reliability) と呼んでいる.

1.2.1 クォーツ時計

クォーツ[1]は, 地球で二番目に豊富な鉱物で多くの岩石や砂に含まれている. それを純粋化したものが水晶となる. 水晶には**圧電**と呼ばれる特性がある. クォーツの結晶に圧力をかけるとその両側に電圧が生じ, 逆に結晶の両側に電圧をかけると結晶は伸び縮みの振動を繰り返す. これを**発振**と呼んでいる.

水晶の発振は鐘の原理に似ていて, その形, 大きさや材質などによって, 固有の振動数を持っている. 水晶が時計に適している理由は, 資源の豊富さだけでなく, この振動が非常に安定しており, 結晶の薄片を適切な形にすると, 望む周波数を生成できることが大きい.

時計では毎秒数千回から数百万回という値が用いられ, この振動周波数を数えられるよう

[1]大部分は二酸化硅素である.

に周波数が選ばれている. 現在よく選ばれているのは, 32,768Hz (2^{15}Hz) という周波数で, 出力を 15 回半分にすれば 1 秒に 1 回の正確な拍子が得られる.

水晶が安定しているといっても, その精度は環境条件, 特に温度変化に敏感で, それが元で発振周波数に狂いが生じる. 精度の安定性はほぼ材質によって決まり, 温度の変化に強く精度の高い水晶はとても高価である (表 1.1, 本表では短期安定度は 1 秒を, 長期安定度では 1 日を, 確度は 1 年の範囲での値を示している).

	方式	短期安定度	長期安定度	確度	備考
↓	XO	—	—	10^{-5}	一般機器
高	TCXO	10^{-9}	10^{-8}	10^{-6}	一般機器
価	OCXO	10^{-12}	10^{-10}	10^{-8}	高短期安定
↓	ルビジウム	10^{-11}	10^{-12}	10^{-10}	二次標準器
↓	セシウム	$10^{-11\sim12}$	$10^{-13\sim14}$	$10^{-12\sim13}$	一次標準器
↓	水素メーザ	10^{-12}	$10^{-14\sim15}$	10^{-13}	超高安定

表 1.1: 高安定発振器の現状

1.2.2 原子時計

原子時計は高い精度が求められる標準時刻や標準周波数の提供に利用されている. 信号源としてルビジウム原子や**セシウム原子** が用いられ, 現在はセシウム原子時計が主流である. ほとんどのセシウム原子時計は, セシウム原子の吸収・放出する電磁波の周波数を利用しており, 30 万年から 170 万年に 1 秒誤差がある程度と非常に精度が高い.

後述する TAI と呼ばれる時間はセシウム原子時計を使った時刻を基準とし,「1 秒はセシウム 133 原子の基底状態の 2 つの超微細準位 (F=4, M=0 および F=3, M=0) 間の遷移に対応する放射の 9,192,631,770 周期の継続時間とする」と定義している.

また, セシウム原子時計は通信事業者や放送事業者などでも利用されており (非常に厳密な精度のクロックを必要とするデジタル関連機器でも利用されている), 民生用の工業製セシウム原子時計が 500 万円〜900 万円程度で販売されている. 工業製のセシウム原子時計はビームチューブ方式と呼ばれる構造でチューブの耐用年数が来たら, ビーム管を新しいものに交換する必要がある (寿命は概ね 3~7 年程度). 工業製といってもあなどってはいけない. 実は TAI に寄与している 260 余りの原子時計のうち大多数は工業製セシウム原子時計である.

現在, 工業製セシウム原子時計はアジレント社や Symmetricom 社から販売されているが,

アジレントテクノロジー社の 5071A が非常に人気があり, TAI の工業製原子時計のほとんどを占めている[2].

1.3 TAI と UTC

我々は日常生活の中で時間についてあまり深く考える事は無いが, 実は地球上では二種類の非常に似通った時間の定義が存在している. 一つは原子の振動数を元にしたもの, もう一つは地球の自転を元にしたものである.

まず, 原子の振動数を元にして決められている時間を **TAI**(Temps Atomique International)[3] と呼んでいる. TAI はセシウム 133 原子の出す特定の電磁波が 91 億 9263 万 1770 回出るのに要する時間を「原子時の 1 秒」と定義した時間で, TAI は国際度量衡局 (BIMP: Bureau International des Poids et Measures) によって管理されている. 原子時の決定方法は, 50ヵ国以上 270 以上の原子時計を平均して得た時系 (自由原子時: EAL) を 8 つの一次標準器[4]で較正して国際原子時を決めている (図 1.2, 1.3).

名称	所在地	型
NICT-O1	日本 (情報通信研究機構)	熱ビーム光励起型
PTB-CsF1	ドイツ	冷却原子泉型
NIST-F1	米国	冷却原子泉型
NMIJ-F1	日本 (産業技術総合研究所)	冷却原子泉型
SYRTE-FO2	フランス	冷却原子泉型
SYRTE-JPO	フランス	熱ビーム光励起型
IT-CsF1(IEN-CsF1)	イタリア	冷却原子泉型
NPL-CsF1	英国	冷却原子泉型

表 1.2: 世界の一次周波数標準器 (PFS)

一方, 我々は太陽の出没, すなわち地球の自転を基準に生活を営んでおり, 地球の一回転を一日とした時間で過ごしている. この地球の自転を基準にした時間を **UT**(Universal Time: 世界時)[5] と呼ぶ. ところが, 地球の自転は季節や場所によって変動するため, 面倒なことに世界時は UT0, UT1, UT2 の三つが定義されている.

[2] 2004 年で 68%.
[3] TAI はフランス語で, 日本語での意味は国際原子時 (International Atomic Time) となる.
[4] TAI が秒の定義と整合性が取れるよう, 自由原子時を較正するために用いられる原子時計で, 最近では, SYRTE-JPO (仏), NIST-F1 (米), PTB-CSF1(独), NICT-O1(日本) 等がその役目を務めている (表 1.2).
[5] 世界時 UT はグリニッジにおける平均太陽時角を元に決められる.

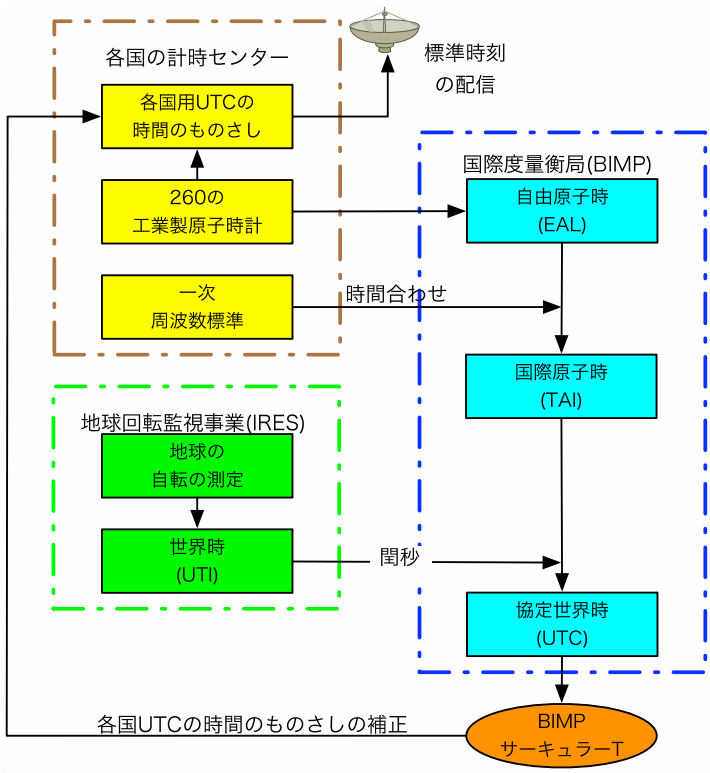

図 1.2: 世界時システム (http://www.npl.co.uk/time/workd_time_system.html)

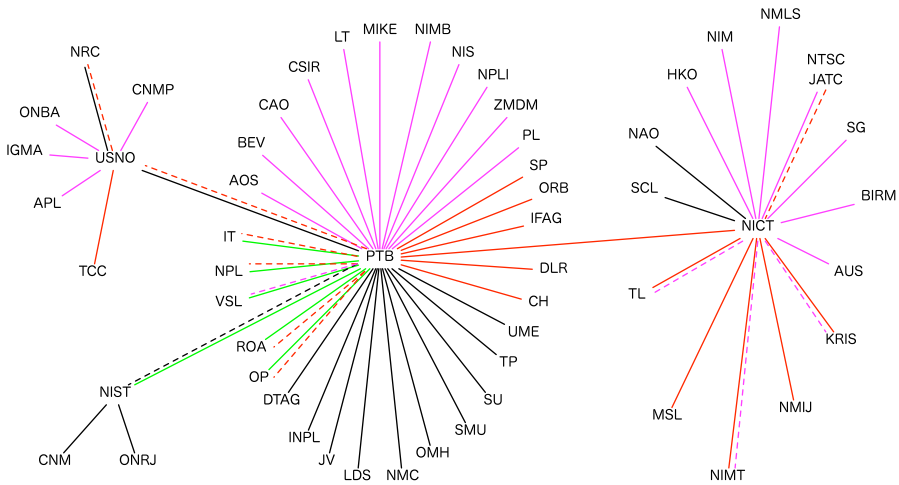

図 1.3: TAI のネットワーク (http://www.bipm.org/en/scientific/tai/tai.html)

- UT0 は, 平均太陽時で地球上の位置や季節により変動がある.
- UT1 は, 地球の自転軸が変化することにより, その場所の緯度経度が時間と共に移り変わる. その結果として生じる地球上の位置による太陽時の変動を補正したもの. 地球上の場所によらず一定であるため, 一般的に地球の自転に基づく時間とされる. しかし, 自転速度が一様ではないため, UT1 は 1 日あたり ±3 ミリ程度のずれがある.
- UT2 は, UT1 から更に季節変動を取り除いたものだが, 現在はほとんど使われていない.

$$UT2 = UT1 + 0.0220\sin(2\pi t) - 0.0120\cos(2\pi t) - 0.0060\sin(4\pi t) + 0.0070\cos(4\pi t) \text{ 秒} \tag{1.1}$$

t はベッセル年 (Besselian Year)[6]で表した時間.

しかも, 潮汐力などの影響で地球の自転は必ずしも一定ではないため[7], 地球の自転一回転を 1 日と定義してしまうと, 厳密な意味での 1 秒の長さがその時々で変わってしまい, 不便が生ずる (すなわち, TAI の 1 秒と UT の 1 秒の二つの定義が存在することになる). この矛盾を解消するため, 1972 年 1 月 1 日 0 時 0 分 0 秒から 1 秒の長さの定義は TAI を使う事に

[6]天文学的に便利なように定められた 1 年間.

[7]2003 年現在, 地球の自転を観測すると, 地球は TAI をベースにした 24 時間よりも約 1 ミリ秒長くかかって 1 回転している. すなわち, UTC の一日を TAI で表すと 86400.002 秒に近くなる.

なった. これを元にしてできた世界時が **UTC**(**協定世界時**) と呼ばれるもので, 我々は普段この時間 (UTC) を使っている. 従って, 1 秒以下の精度で良ければ, UTC は UT1 の近似として使うこともできる.

1.4 閏秒

世界時 (UT) の 1 秒の定義を TAI ベースにすることが決定されたが, TAI の 1 秒は地球の自転と無関係に時を刻むため, 一日は TAI 秒で表すと 86400.002 秒になってしまう. UT に対して何も修正を加えないと, 次第に TAI との誤差が大きくなり, (かなりの年数を要するが) 最終的には昼夜が逆転することも起こり得る. そこで, 国際地球回転観測事業 (IERS: International Earth Rotation Service) の天文観測によって得られるデータを元に, TAI と地球自転時 (UT1) の時刻差が ± 0.9 秒内に収まるよう**閏秒**という調整秒を加えたり, 取り除いて 1 日の補正を行っている. すなわち, 閏秒が 1 秒加わる場合には 23:59:59 → 23:59:60 → 00:00:00 と変化し, 閏秒が 1 秒減じられる場合には 23:59:58 → 00:00:00 → 00:00:01 となる.

TAI と UT の調整は, 1958 年 1 月 1 日 0 時 0 分 0 秒を起点に世界時 (正確には UT を補正した UT2) と合わせている. UTC がスタートしたは 1972 年 1 月 1 日であり, この時, 既に UT と TAI の差が 10 秒に近付いたため, 特別調整として 10 秒差に設定された. そして, 閏秒の調整時期は 1 月 1 日 0 時 0 分 0 秒または (及び)7 月 1 日 0 時 0 分 0 秒 (の直前) と決められ, 1972 年 7 月 1 日に第 1 回目の閏秒調整が実施された. 以後, 現在まで TAI と UTC には 33 秒の差がある[8]. なお, 閏秒の調整は事前に計測データを元に予測され, 少なくとも 8 週間前に告示することになっている.

表 1.3 の閏秒の実施状況によると, 一番新しい閏秒の挿入が 2006 年の元日で, その前は 1999 年元日である. 不思議なことに 5 年もの間, 閏秒は挿入されていなかった. これは地球の自転速度が一定であることを示しているのだが, なぜこのような現象が続いていたかについてはよくわからない[9].

1.5 閏時

1.5.1 閏秒廃止論

これまで, 地球の自転と TAI の時間とのずれの調整を行うために閏秒が導入されて来たが, コンピュータやインターネットの発展により閏秒に関する問題が浮かび上がって来た.

[8] http://jjy.nict.go.jp/QandA/data/leapsec.html
[9] 月は地球から 1 年に 3.5cm づつ遠ざかっているため, 公転周期や自転周期は遅くなっている筈である.

回目	実施年月日	閏秒	UTC−TAI	回目	実施年月日	閏秒	UTC−TAI
23	2006-01-01	+1	−33	11	1982-07-01	+1	−21
22	1999-01-01	+1	−32	10	1981-07-01	+1	−20
21	1997-07-01	+1	−31	9	1980-01-01	+1	−19
20	1996-01-01	+1	−30	8	1979-01-01	+1	−18
19	1994-07-01	+1	−29	7	1978-01-01	+1	−17
18	1993-07-01	+1	−28	6	1977-01-01	+1	−16
17	1992-07-01	+1	−27	5	1976-01-01	+1	−15
16	1991-01-01	+1	−26	4	1975-01-01	+1	−14
15	1990-01-01	+1	−25	3	1974-01-01	+1	−13
14	1988-01-01	+1	−24	2	1973-01-01	+1	−12
13	1985-07-01	+1	−23	1	1972-07-01	+1	−11
12	1983-07-01	+1	−22		1972-01-01		−10

表 1.3: 過去の閏秒の実施状況

- 閏秒の挿入頻度は今後増加する (表 1.4 参照): 月が地球から離れていくに従って, 月の朝夕力による自転への影響が弱まっていくため, 徐々にではあるが地球の自転は遅くなると予想されている.
- ソフトウェア上の問題: 閏秒は予測ができないためコンピュータ誤動作を引き起こす事故[10]が多い点, 1 日の秒数が一定ではない点, 時間表記の問題がある.
- 通信の問題: 閏秒中のイベントの調整が難しい
- 時間尺度に依存しないシステム (例: GPS) の発展

閏秒が大きな問題になる前に, アメリカを中心に閏秒の無い UTC を再定義し, 替わりに約 600 年 (予測では最初の閏時の挿入は 2505〜2063 年に行われる) に一度調整すればよい**閏時**を設ける, とする動きが 2003 年頃から出始めている[11].

なお, 国際電気通信連合無線通信部門 (ITU-R) は, 2005 年 12 月から翌年の 1 月にかけて, 閏秒に関する意見を募集していた.

[10]大きな事故として, 1996 年にはラジオ局のプログラムが誤動作して間違った番組を放送したり, 2003 年にはモトローラの GPS 受信機が閏秒のバグのため誤った時間 62:28:15 を表示等がある.

[11]詳しくは `http://www.ucolick.org/~sla/leapsecs/`.

leap rate	25.6 $s/century^2$ (since 1620)	31 $s/centry^2$ (observed over 2700 years)	42 $s/centry^2$ (long term)	挿入タイミング
1 s/yr	2015	[1981]	[1939]	毎年発生しない 6 月と 12 月のどちらか 1 あるいは 2 年に一度
2 s/yr	2211	2142	2058	毎 6 月と 12 月 1 年に 2〜4 回
4 s/yr	[2602]	2464	2296	毎 3, 6, 9, 12 月 1 年に 4〜12 回
12 s/yr	[4167]	3753	3248	毎月 ITU-R TF.460 では無理
1 s/week	—	[10223]	[8031]	毎週 (70 年で 1 閏時)
1 s/day	—	[60644]	[45298]	毎日 (10 年で 1 閏時)

表 1.4: 今後の閏秒挿入予想

1.5.2 閏時とは

閏時への移行プランは 2003 年トリノで開催された ITU WP 7A SRG で示されている.

1. まず, 現行の UTC から新しいシステムに移行するため, TI (Temps International: 国際時) と呼ばれる時間尺度を新たに設ける.
2. UTC の 50 周年である 2022 年に閏秒をリセットする.
3. TI はそのまま原子時 (今の TAI) とし, TI と UT1 の差が 30 分を超えた時に閏時を挿入する (いつどのようなタイミングかは?).

これで次の閏時が挿入されるまでの間 (約 600 年後), 原子時と同じ時間を示す TI が, 我々が日常的に使う時間となる.

ITR-R では 2007 年 9 月にも閏時に関する合意をし, 2010 年には移行プロセスの合意を目指すとのことである.

1.5.3 UTC-SLS (UTS)

UTC から閏秒を無くす案として, `adjtime()` と同じような考え方で 1 秒を伸び縮みさせてしまう方法も考えられた. UTC with Smoothed Leap Seconds (UTC-SLS あるいは UTS) と呼ばれる方式[12] で, 1000 秒の間に 1 秒の差分を伸び縮みさせることで, 閏秒の挿入をしなくてよいようにしている. ユニークな案ではあるが, IETF ドラフトは期限切れになっており, ITU-R でも採用されなかったようだ.

UTC	UTS	
23:43:20.000	23:43:20.000	
23:43:21.000	23:43:21.000	UTS=UTC の終わり
23:43:22.000	23:43:21.999	
23:43:23.000	23:43:22.998	
23:43:24.000	23:43:23.997	
995 秒後...		
23:59:59.000	23:59:58.002	
23:59:60.000	23:59:59.001	閏秒開始
00:00:00.000	00:00:00.000	再び UTS=UTC
00:00:01.000	00:00:01.000	

1.6 日本標準時 (JST)

日本標準時(**JST**: Japan Standard Time) は日本国内の標準時間を表すもので, UTC から 9 時間進んでいる (UTC +0900 となる). 日本標準時は情報通信研究機構 NiCT(旧通信総合研究所 CRL) から供給されており, そのために 18 台のセシウム原子時計と 3 台の水素メーザー原子時計が使われている (これらと世界各国の原子時計との時刻比較測定結果は BIMP に送られ, TAI の決定にも使われている).

そして, NiCT は日本国内に広く日本標準時の配布するため, 次な方法を用いている.

- 長波を使った電波 (JJY)
- 電話回線 (テレホン JJY)
- インターネット[13] (NTP)

[12] `http://www.cl.cam.ac.uk/ mgk25/time/utc-sls/draft-kuhn-leapsecond-00.txt`
[13] インターネットマルチフィード株式会社 (`http://www.jst.mfeed.ad.jp/`)

第2章　コンピュータと時間

2.1　コンピュータが持つ時計

コンピュータにとって, 時計はファイルの作成・編集日時, メールのタイムスタンプ, トランザクション処理, データの同期化など様々な用途で必要となる. また, 分散コンピューティング環境や電子商取引の発達により, ネットワーク内の全てのコンピュータに時刻の正確さが求められるようになった.

通常, デスクトップ PC から大型のコンピュータまで, あらゆるコンピュータは何らかの形でカレンダ機能を有しており, その内部機能を使ってコンピュータの時刻合わせを行っている.

Unix 系オペレーティングシステムを例にとると, ハードウェアクロックとシステムクロックの二つの時計を持っている. ハードウェアクロックは, マザーボード上の LSI に搭載された時計で, PC の電源を切った状態でもバッテリ (電池) によって時刻を刻み続ける. 時刻は CMOS に保存されるため, CMOS 時計, リアルタイムクロック (RTC) や BIOS 時刻等と呼ばれることもある. 一方, システムクロックの方はカーネルが管理している時計で, システム起動時に一度だけハードウェアクロックの時間を参照し, それ以降はカーネルが刻む割り込みを元に時間を算出する. 従って, オペレーティングシステムの時計はシステムクロックが基準となる (コンピュータが動作中はマザーボード上にあるハードウェア時計を使っていない).

通常, パソコンで使われている時計はあまり精度がよくないので (ビデオデッキに内蔵されている時計よりも劣るといわれている), 本題である NTP を使うことでシステムクロックの精度が増す必要がある. Linux の場合, ハードウェアクロックとシステムクロックを相互に反映させることができる. システムクロックをハードウェアクロックに合わせるには, `hwclock(8)` コマンドを使って “`hwclock --systohc`” と実行し, 逆にハードウェアクロックをシステムクロックに反映させるには, “`hwclock --hctosys`” と実行する.

システムクロックとハードウェアクロックとの整合性を持たせるため, コンピュータの終了時に NTP で正確さを増したシステムクロックの時刻をハードウェアクロックに反映させておいた方がよい.

2.2 ハードウェアクロックの設定方針

一般的に, ハードウェアクロックは UTC にしておくのが良いとされている. それぞれのオペーレーティングシステム別に合わせ方を説明する.

2.2.1 Linux

Red Hat 系のほとんどのディストリビューションは, `timeconfig` コマンドで設定できる. ハードウェアクロックを UTC に, タイムゾーンが日本 (東京) の場合, 次のように設定する.

```
# timeconfig --utc "Asia/Tokyo"
```

`timeconfig` コマンドは起動時に読み込まれる設定ファイル`/etc/sysconfig/clock` の編集を合わせて行っているので, このファイルを手動で変更することもできる.

```
ZONE="Asia/Tokyo"
UTC=true
ARC=false
```

Debian 系のディストリビューションの場合, `/etc/default/rcS` ファイルの中の UTC を yes に変更する.

また, `hwclock` コマンドについて追加情報として, `--utc` オプションをつけることで, システムクロックをハードウェアクロックに UTC で反映できる.

```
# hwclock --systohc --utc
```

2.2.2 FreeBSD

FreeBSD の場合, `/etc/wall_cmos_clock` がある場合, ハードウェアクロックをローカル時間として処理し, 無ければ UTC として扱う. これはインストール時に, BIOS クロックが UTC かどうかを尋ねられるので, ここで Yes を選択しておけば, ハードウェアクロックは UTC として扱われる. なお, ハードウェアクロックをローカル時間として扱うと, `adjkerntz` デーモンが常駐する.

2.2.3 NetBSD

NetBSD は Generic カーネルは最初からハードウェアクロックを UTC として扱うようになっている. ハードウェアクロックをローカル時間に合わせるには, カーネルの rtc_offset 変数の変更が必要である. 変更には, RTC_OFFSET をローカル時間に合うように設定して (−540 とする), カーネルの再構築を行うか, 下記の方法でカーネル内の変数を変更する.

```
# gdb --write /netbsd
(gdb) set rtc_offset=-540
(gdb) quit
```

2.2.4 OpenBSD

OpenBSD も Generic カーネルは最初からハードウェアクロックを UTC として扱うようになっている. ハードウェアクロックをローカル時間に合わせるには, カーネルの timezone 変数の変更が必要である. 変更には, TIMEZONE をローカル時間に合うように設定して (−540 とする), カーネルの再構築を行うか, 下記の方法でカーネル内の変数を変更する.

```
# config -ef /bsd
OpenBSD 3.9 (GENERIC) #617: Thu Mar  2 02:26:48 MST 2006
    deraadt@i386.openbsd.org:/usr/src/sys/arch/i386/compile/GENERIC
Enter 'help' for information
ukc> timezone -540
timezone = -540, dst = 0
ukc> quit
Saving modified kernel.
```

2.2.5 Solaris

Solaris も他の BSD Unix 同様, ハードウェアクロックを UTC として扱うようになっている. ハードウェアクロックをローカル時間に合わせるには, rtc コマンドを利用する. すると, /etc/rtc_config ファイルの zone_lang に UTC との時間差 (秒) が記録される.

```
# /usr/sbin/rtc -z Japan
```

2.2.6 Mac OS X

Mac OS X もまた他の Unix 同様, ハードウェアクロックを UTC として扱うようになっている. 従って, BootCamp を使って Windows XP をデュアルブートにする場合, XP の起動時に AppleTime というソフトウェアを走らせて, ソフトウェアクロックを強制的にローカル時間と合わせるようにしている.

Parallels Desktop という仮想化ソフトウェアは VM がローカル時間になっているようなので, Windows は問題無いが, 逆に Linux や BSD を使う場合はハードウェアクロックがローカル時間になるように設定を直す必要がある.

2.3 UNIX 時間

コンピュータの代表的なオペレーティングシステムの一つである Unix には独自の時間, Unix 時間を持っている. Unix 時間は, コンピュータが計算しやすいように, "**エポック**(Epoch): 1970-01-01-00:00:00 GMT" を基点とした経過秒数で表され, FreeBSD, Linux, Solaris 等, Unix から派生した全てのオペレーティングシステムで使われている.

エポック時間の基点となっている GMT(Greenwich Mean Time) というのは, グリニッジ[1] 標準時という意味でイギリスのタイムゾーンを示しているが, この表記は現在では使われていない. エポックの正しい時間は, "1970-01-01 00:00:10 UTC" と書くか, TAI で表すなら "1970-01-01 00:00:10 TAI" となる.

このエポック時間からの経過秒数を元に我々が日常的に使っている何年何月何日何時何分何秒に変換する. この変換を行うシステムコールが `ctime(3)`, `gmtime(3)`, `localtime(3)` などのライブラリルーチンである .

現在の Unix 時間を知りたければ, `date` コマンドを次のように使う.

```
% date +%s
1135861280
```

逆に Unix 時間を UTC に変換するには, ちょっとしたスクリプトが必要だが, Ruby の場合は, 非常に簡単に変換できる.

```
ruby -e 'p Time.at(1135861280)'
Thu Dec 29 22:01:20 JST 2005
```

[1] グリニッジ天文台は 1998 年 10 月で活動を停止し, 323 年に及ぶ歴史の幕を閉じた.

2.4 Unix における閏秒の取り扱い

Unix に実装されている時間関連のライブラリは長らく閏秒をサポートしていなかったために, TAI を UTC として取り扱って来た. これでは閏秒が起きるたびに UTC の時刻からずれていく事になる. しかし, 皆さんは NTP のようなプロトコルを使って正しい UTC を参照していれば何も問題無いのではとは思うかも知れない.

ところが, 過去に遡って, 時間を知る必要がある場合や時間の経過をスタンプするアプリケーションの場合, 閏秒を考慮に入れていなければ正しい (過去の) 時間を知る事はできなくなる. もしも, 1 秒でも狂ってはまずいアプリケーションプログラムがあった場合 (例えば, 商取引のコンピュータのような), それが重大な問題となるのは自明であろう.

D.J. Bernstein 氏が指摘しているように[2], `localtime(3)` ライブラリが閏秒をサポートしていないため, xntpd では閏秒が起こると時刻は次のように変わっていく.

```
1997-06-30 23:59:59.7 UTC -> 867715199.7 xntpd
1997-06-30 23:59:59.8 UTC -> 867715199.8 xntpd
1997-06-30 23:59:59.9 UTC -> 867715199.9 xntpd
1997-06-30 23:59:60.0 UTC -> 867715200.0 xntpd
1997-06-30 23:59:60.1 UTC -> 867715200.1 xntpd
1997-06-30 23:59:60.2 UTC -> 867715200.2 xntpd
1997-06-30 23:59:60.3 UTC -> 867715200.3 xntpd
1997-06-30 23:59:60.4 UTC -> 867715200.4 xntpd
1997-06-30 23:59:60.5 UTC -> 867715200.5 xntpd
1997-06-30 23:59:60.6 UTC -> 867715200.6 xntpd
1997-06-30 23:59:60.7 UTC -> 867715200.7 xntpd
1997-06-30 23:59:60.8 UTC -> 867715200.8 xntpd
1997-06-30 23:59:60.9 UTC -> 867715200.9 xntpd
1997-07-01 00:00:00.0 UTC -> 867715200.0 xntpd
1997-07-01 00:00:00.1 UTC -> 867715200.1 xntpd
1997-07-01 00:00:00.2 UTC -> 867715200.2 xntpd
```

右側の Unix 時間の秒数を見て欲しい. xntpd はタイムスケール (867715200) を繰り返していることがわかる (POSIX の仕様上は正しい). これでは, この間の時間を正しい UTC に変換することはできない[3]. もし, 閏秒を考慮するなら, Unix 時間の調整をする必要がある (挿入された場合は 1 秒戻し, 削除された場合は 1 秒進める).

[2] http://cr.yp.to/proto/utctai.html

[3] Arthur Olson 氏らが閏秒を取り扱うことができる時間 (tz) ライブラリを作っており, ftp://elsie.nci.nih.gov/pub/から入手できる.

2.4.1 BSD 系のオペレーティングシステム

BSD 系のオペレーティングシステム (FreeBSD, NetBSD, OpenBSD 等) は閏秒をサポートしている. デフォルトでは考慮されていないので, 閏秒を考慮するには, `zoneinfo` を次のように作り直す必要がある.

```
# cd /usr/src/share/zoneinfo/
# leapseconds ファイルを編集
# make clean
# make LEAPSECONDS=yes install
# cp /usr/share/zoneinfo/Asia/Tokyo /etc/localtime
```

`leapseconds` ファイルは次のように編集する. 例えば, 一番新しい閏秒の挿入を行う場合, 次の行を 1998 の下に加える. CORR(correction) は閏秒の挿入 (+) か削除 (−) を, R/S は閏秒の時刻が UTC なら S(Stationary) を, ローカル時刻なら R(Rolling) を与える.

```
# Leap  YEAR    MONTH   DAY     HH:MM:SS        CORR    R/S
Leap    2005    Dec     31      23:59:60        +       S
```

2.4.2 Linux 系のディストリビューション

Linux 系のディストリビューション は glibc のバージョンが 2.1 以降であれば閏秒に対応している. タイムゾーンもしくは環境変数 `TZ` が “right/Asia/Tokyo[4]” に設定されていれば, 閏秒に対応していることになる. 新しい閏秒が設定された際は, 対応した timezone あるいは zoneinfo パッケージ (tzdata) を導入すればよい.

次の手順で行うこともできる.

1. `ftp://elsie.nci.nih.gov/pub/`から最新の timezone 情報のデータを入手する (2007 年 4 月 13 日時点の最新版は `tzdata2006l.tar.gz` である).
2. ファイルを適当な場所で解凍する.
3. `asia` と `leapseconds` ファイルを使って zoneinfo ファイルを次のコマンドで作成する. すると, Asia ディレクトリの下に各地の zoneinfo ファイルが作成される.

[4] Asia/Tokyo とすると閏秒に対応していない.

```
# zic -d . -L leapseconds asia
```

4. Asia に作成された zoneinfo ファイルを/usr/share/zoneinfo/Asia にコピーし, Asia/Tokyo を/etc/localtime にコピーする.

2.4.3 NTP

NTP では閏秒の調整前に時刻パケットの Leap Indicator(LI) を 01(挿入), もしくは 10(削除) にセットすることになっている. 上位の NTP サーバが LI をセットすれば下位の NTP サーバはそれを順次伝えていくことになっている.

実際は, ストラタム 1 の NTP サーバが LI を設定しなければならないが, NTP のソフトウェアとしてはクロックドライバ任せになっているようだ (ほとんどのドライバはサポートしていない). もしも, ストラタム 1 が LI の設定を行わないとどうなるかというと,

1. 1/1 もしくは 7/1 の 08:59:60 までは正しい NTP 時間を返す.
2. 09:00:00 以降, NTP 時間は 1 秒大きな値を返す.
3. 一定時間経過すると, GPS に対するオフセットが −1 秒になる. NTP 時間は 1 秒大きい値を返す.
4. 更に一定時間経過すると, 時刻修正を行い 1 秒戻して, 正しい NTP 時間を返すようになり, Unix 時間も同時に調整される[5].

のようになる.

但し, Solaris の NTP ソフトウェアは独自に LI フラグを ntpq コマンドの writevar(148 ページ参照) を使ってマニュアルで付加することができる. 付加するには二つのどちらかの変数を使う. 一つは leapwarning 変数で閏秒の実行 1ヶ月前に設定できる. もう一つは, leapindicator 変数でこちらは閏秒実施日に設定する (日本の場合, 6 月 30 の 09:00:00 から 7 月 1 日 08:59:60 の間もしくは 12 月 31 の 09:00:00 から 1 月 1 日 08:59:60 の間). これは NTP 側の設定で, 実際のクロック源からの閏秒の調整を何らかの方法で行う必要がある.

[5] ストラタム 1 の NTP サーバであっても, リファレンスクロックとは擬似的な NTP パケットを交換する NTP のクライアントであることに変わりは無いため.

```
ntpq> readvar 0 leapwarning
status=c011 sync_alarm, sync_unspec, 1 event, event_restart
leapindicator=00
ntpq> writevar 0 leapwarning=1
done! (no data returned)
```

2.5 Unix の Y2K 問題

西暦 2000 年が訪れる際に Y2K 問題が世界を騒がせたが, Unix には深刻な時間の問題が内在している. 32 ビットをベースにした Unix 系のオペレーティングシステムは `ctime()` システムコールを使って現在の時刻を獲得するが, この戻り値 (`time_t`) の型は `long int` で定義されている. 符号付きの `long int` 型は $-2,147,483,648$〜2,147,483,648 の範囲の整数が扱えるが, この値を 1 年 (31,536,000 秒) で割るとエポック時間からのカウントは約 68 年となる. 従って, 2038 年中に値が溢れてしまい, Unix の時間関連のライブラリは破綻してしまう.

このことは次のプログラムを動かして確かめることができる.

```
#include <stdio.h>
#include <time.h>
int main()
{
    time_t t; t = 0;
    printf("0x%08x: %s", (unsigned)t, ctime(&t));
    t = time(NULL);
    printf("0x%08x: %s", (unsigned)t, ctime(&t));
    t = 0x7fffffff;
    printf("0x%08x: %s", (unsigned)t, ctime(&t));
    t = 0x80000000;
    printf("0x%08x: %s", (unsigned)t, ctime(&t));
    t = 0xffffffff;
    printf("0x%08x: %s", (unsigned)t, ctime(&t));
    return 0;
}
```

ほとんど全ての Unix 系オペレーティングシステムは次のような結果になる. 32 ビット符号付き整数の最大値 0x7fffffff では, 2038 年 1 月 19 日 12 時 14 分 7 秒[6]だが, 0x80000000 では負の値になり 1901 年に逆戻りしてしまう.

[6]GMT/UTC では 2038 年 1 月 19 日 3 時 14 分 7 秒

```
0x00000000: Thu Jan  1 09:00:00 1970
0x3fa64f91: Mon Nov  3 21:52:33 2003
0x7fffffff: Tue Jan 19 12:14:07 2038
0x80000000: Sat Dec 14 05:45:52 1901
0xffffffff: Thu Jan  1 08:59:59 1970
```

このような結果になる Unix 系オペレーティングシステムには 2038 年問題があるとわかる. 解決方法としては, この 32 ビット数を単純に符号無し整数 (`unsigned long int`) に変更すれば, 更に 68 年間延長できるが, 根本的な解決方法では無い. Unix 系のオペレーティングシステムを開発しているグループは, 時刻を 64 ビット (`long long int`) で扱うように修正する方向に動いている[7] . アプリケーションの再コンパイルが必要だが, 64 ビットになれば, 西暦 292,277,266,665 年まで扱えるようになる. いずれにしろ, 破綻までには 30 年以上あるので, その時までにはオペレーティングシステムの修正が施されるかも知れないが[8], 時刻を先んじて扱う必要がある場合に備えて早目の対策が求められる. 現在, `time_t` が 64 ビットになっているオペレーティングシステムは, 64 ビット版 Windows, Solaris, Debian GNU Linux などである.

なお, 同様の問題は NTP にも存在する (4.5 節, 36 ページ参照).

2.6 夏時間問題

夏時間 (Daylight saving time: DST) は時刻を 1 時間早めて, それに合わせた生活を送る制度で, 明るいうちに仕事をして, 夜は早く寝るようになるため, 結果的に省エネルギーにつながるとされており, 日本政府が導入を推進しようとしている.

ところが, 夏時間を導入しているのは高緯度で夏の日照時間が長い欧米諸国で, 日本のような夏の暑さが夜でも続くような国で夏時間が有効なのかという疑問や反対論が存在している.

いずれにしろ, 夏時間の制度の導入はコンピュータシステムに与える影響は非常に大きい (Y2K 問題以上かも知れない). 但し, 多くのコンピュータはどこのゾーンにいても夏時間を導入できるよう考慮されている. 先ほど説明した, `zoneinfo` のファイルの Asia/Tokyo を夏時間用に修正すれば良い. 夏時間は国によっては決まっていない場合もあるので, 定義の修正を毎年行う必要があるかも知れない.

[7] ANSI C の改訂も必要である.

[8] この問題は 1970 年代にその存在が知られていたが, それから 30 年近く経っている今になっても変更されていないという事実自体に問題があるのだが...

第3章　時刻配信サービス

今まで説明してきた通り, コンピュータ内蔵の時計は必ずしも精度が高いわけではない. ネットワーク経由で時刻を得る事ができない場合や, 更に正確な時刻が欲しい場合は, 何らかの手段で UTC と同期した時刻源の情報を直接コンピュータに取り込む必要がある.

ちなみに, NTP のストラタム 1 サーバは UTC と同期した時計 (ストラタム 0) から時刻の供給を受けているサーバである. 正確な時刻源を得る一つの方法としては, 原子時計とコンピュータを直接接続する方法が考えられるが, 工業製セシウム原子時計は一台 700 万前後と非常に高価なため, 簡単には導入できない. そこで, よく用いられる方法に, 多くの国で行われている標準時刻の配信サービス (表 3.1) を利用する方法がある. ほとんどのサービスで時刻源に高い精度を持つ原子時計を用いられているため, これらのサービスを利用すればコンピュータに正確な時刻を供給できる (また, NTP ストラタム 1 サーバを安価に構築する事もできる).

時刻配信サービス	提供者	コメント
標準電波放送	各国で実施	GPS と比較して精度がやや劣る
GPS	米国防総省	地球全体をカバー
CDMA	携帯電話キャリア	クアルコム社の方式
時報	NHK, NTT	
FM 文字放送	FM 放送局	

表 3.1: 標準時刻配信サービス

3.1　標準電波放送 (JJY)

標準電波放送 (JJY) は時間と周波数の標準, 並びに UTC に基づく各国のローカル標準時 (日本の場合は JST) を広く国内に知らせるために運用されている電波放送で, アメリカ, 日本, ドイツなど先進各国が行っている. 放送に使用される電波は主に短波や長波が使われており, 代表的なものに表 3.2 がある.

波	送信国	識別信号	周波数	出力 (kW)
長波	日本 (福島)	JJY	40 kHz	50
	日本 (佐賀)	JJY	60 kHz	50
	米国 (コロラド)	WWVB	60 kHz	50
	ドイツ	DCF77	77.5 kHz	50
	イギリス	MSF	60 kHz	25
短波	韓国	HLA	5 MHz	2
	米国 (ハワイ)	WWVH	2.5/5/10/15 MHz	2.5/10/10/10
	米国 (コロラド)	WWV	2.5/5/10/15/20 MHz	2.5/10/10/10/2.5
	カナダ	CHU	3.33/7.335/14.67 MHz	3/10/3

表 3.2: 世界の標準周波数時報局

日本では情報通信研究機構 NiCT がその任を負っており, 以前は 5, 8, 10MHz の短波帯周波数を使って送信されていたが, 現在は長波帯を使って, 福島県の大鷹鳥谷山 (おおたかどややま) と佐賀県のはがね山の 2ヵ所から日本全域をカバーできるタイムコードを送信している[1].

この放送は “**JJY**(ジェージェーワイ)” と呼ばれるが, この言葉自身に何か意味があるわけではなく, JJY という名前は無線局の識別信号 (コールサイン) である.

その他, 試験的に電話回線を使った時刻の配信も行っている (テレフォン JJY[2]). これは米国で行われている NIST のテレフォンサービス (ACTS[3]) に似ており, モデム信号を使って標準時刻を取得できる事が可能なサービスで, ほぼ ±1 ミリ秒以内の精度で同期をとることができる.

3.2 GPS

GPS(全地球測位システム: Global Posisioning System) はカーナビゲーションシステムや測量などで用いられ, 一般にも広く認知されているシステムである[4]. 元々は米国国防総省 (DoD) が軍事目的の他, 航空機・船舶等の航法支援用に開発したシステムである.

[1]詳しくは http://jjy.nict.go.jp/を参照すると良い.
[2]http://jjy.crl.go.jp/TelJJY/index.html
[3]http://www.boulder.nist.gov/timefreq/service/acts.htm
[4]同様のシステムとして, GLONASS(Global Navigation Satellite System) と呼ばれる GPS システムをロシアが持っているが, 2007 年 4 月時点で 8 機の衛星が稼働している. また, 米国依存から脱却するためにヨーロッパ (EU) においてガリレオプロジェクトという独自システムを計画中である.

このシステムは上空約 2 万 km を周回する 24 個の GPS 衛星 (6 軌道面に 4 基ずつ配置されている) を使い[5], GPS 衛星の追跡と管制を行う管制局, 測位を行うための利用者の受信機で構成されている. 衛星は約 12 時間で地球を一周し, 地球上のどこからでも常に 4 個以上見えるよう設定されている.

衛星から送信される電波には L1 バンドと L2 バンドの 2 種類がある. L1 バンドは 1575.42MHz の周波数を使用し, L2 バンドは 1227.6MHz の周波数を使用する. 送られてくる信号には, 米軍のみに使用許可が与えられている P(Y) コードと, 民生利用が可能な C/A コードがある. C/A コードは L1 バンドのみで送られるが, P(Y) コードは L1 と L2 バンドを使って送られてくる. P(Y) コードは 2 種類のバンドを使うことで, 電離層の補正が実測値で可能になるため誤差を小さく抑えることが可能となる. また, P(Y) コードと C/A コードには 32 種類あり, GPS 衛星それぞれに別々のコードが割り当てられており, 同時に複数の衛星の信号を受信しても混同しないようにできている.

バンド	L1	L2
搬送波周波数	1575.42MHz (10.23MHz×154)	1227.60MHz (10.23MHz×120)
波長	約 19cm	約 24cm
P コード	送信	送信
C/A コード	送信	なし

表 3.3: GPS の信号

C/A コードというのは, Coarse Acquisition Code の略で, L1 帯を使い 1023 の 0 と 1 で擬似ランダム変調が施され, チップレート 10.23MHz で送られてくる. C/A コードは敵対国が軍事目的に利用しないよう 1992 年 7 月にセレクティブ・アベイラビリティ (SA) の政策を課し, 測位精度を 100 メートル以内に制限していた. 一方, P(Y) コードは Precise Code の略で, より長い約 10^{14} ビットに 2 位相擬似ランダム変調を施し, n チップレート 10.23MHz で送られ, 更に軍事用として P コードの信号を暗号化した Y コードが使われている. P(Y) コードでの測定誤差は約 15m と高精度で, C/A コードでの測定誤差は 100m 程度と P(Y) コードに比べてかなり甘い. しかし, 1993 年に C/A コードの民生利用が許可され, カーナビなどの目的に広く利用されるようになった. そして, 2000 年 5 月 2 日 13:00(JST) に米大統領声明により GPS に施されていた SA が解除され, 測定誤差が 10m 程度にまで改善された (但し, 戦争等の不測事態が起きた場合に挿入される可能性は否定できない).

[5]衛星の寿命は約 7.5 年で, これまで 40 機以上の衛星が打ち上げられており, 2003 年 5 月時点で 29 機の GPS 衛星が稼働している.

3.2.1 GPS 受信機が位置を計算する方法

衛星は約 2 万 Km 上空の軌道上を周回しているので, 衛星から届く時間に違いが出る. そのため, GPS で位置を知るには, 4 つ以上の衛星からの同時に L1 もしくは L2 バンドの電波を捕らえ, 衛星の位置情報と送られるタイムパルスの到達時間差を測位演算することにより自身の位置を決定する. 正確なタイムパルスを作るために, 各衛星にセシウム原子時計がなんと 4 台も搭載されており, 全ての衛星の時刻が同期している (精度の維持は地上の監視局から行われる) [6].

GPS 受信機と GPS 衛星との距離 (擬似距離) を計算する方法は, GPS 受信機のタイプにより大きく分けるとコードベースと搬送波位相ベースの二つがある.

コードベースの受信機は, 衛星への距離を計算するために, 光速と衛星から受信機への信号の伝達にかかる時間差を利用する. 時間差はコード化された信号の特定の部分が衛星を発した時間とアンテナに到着した時間を比べることによって行われる. 時間差は光速の定数 (c=298,000Km/秒) を乗ずることによって距離に変換される. コードベース方式の場合, 最低 4 衛星からの距離が測位を行うことができる受信機が必要で, 測位はおおよそ毎秒受信機で行われる.

GPS で位置を算出するための簡単な原理を単独測位法を元に説明する. 図 3.1 で 4 つの衛星 (SV1, SV2, SV3, SV4) が既知の時, 各衛星と GPS 受信機との距離 (d1, d2, d3, d4) を知る事ができれば, 球の方程式の解として, GPS 受信機の位置 (X, Y, Z) が決まるというものである.

まず, i 番目の衛星と GPS 受信機の距離は, 送信時刻と受信時刻の差 (伝搬時間) に光速 (c) を掛けることで算出できる ($d_i = c\,t_i$). 次に受信地の座標を (x, y, z) とすると, 次の式 (3.1) が成り立つ.

$$d_i = \sqrt{(x - X_i)^2 + (y - Y_i)^2 + (z - Z_i)^2} + s \tag{3.1}$$

s は GPS 受信機の時計誤差による距離への影響を表す (この誤差を含むため, 疑似距離と呼ばれる). ここで 4 つの未知数 (x, y, z, s) を解くためには, 4 つの式が必要となり, そのため 4 つの衛星に対して同時に測定・計算し, 連立方程式を解いて位置を知る.

なお, 測位精度を向上させるために, ディファレンシャル GPS(DGPS) やキネマティック GPS(KGPS)[7] の技術が開発された. ディファレンシャル補正の技術は, 一般に数メートル以内の精度の測位を得ることができる (3.2.2 節).

[6] 地上で 1 マイクロ秒ずれると地上で約 300m の誤差となるので, 数 m の測位を得るには数ナノ秒の精度で時刻の同期をとる必要がある.

[7] DGPS を移動体に適用させたもの.

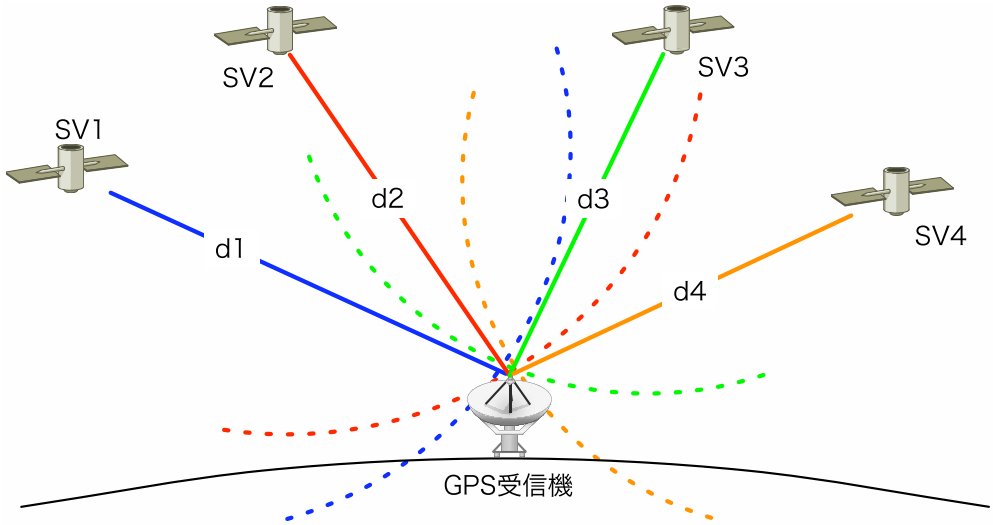

図 3.1: GPS の原理 (単独測位法)

一方, 搬送波位相受信機はコードベースよりも精度が高く, 測地基準点や精密測量のアプリケーションで広く使われ, サブセンチメートル (cm) のディファレンシャル精度の能力がある. この受信機のタイプは衛星と受信機間の全体の波長の数 (N) の決定と, 信号の波長の部分 (位相) の測定によって見えている衛星への距離 (擬似距離と呼ぶ) を計算する. 波長の値がわかったら, 擬似距離は搬送波信号の波長 (L1 と L2、それぞれは 19cm と 24.4cm) に N を乗じ, 波長の一部を加えることによって計算される. そして, 一つの受信機を正確にわかった緯度, 経度と高度のポイントに置いて, ベースライン (二点間の距離) の計算で, 未知の点の座標が決定される.

3.2.2 ディファレンシャル GPS

ディファレンシャル GPS(DGPS) は受信機や位置測定作業によって生じる誤差を除去するために開発された技術である. DGPS を利用するには, 緯度/経度と高度が既知の基準局が必要で, 基準局の位置と GPS によって計算された基準局との位置の差を求めることで, 位置を補正する事ができる. DGPS を使う事で, その精度は 5〜6m まで求める事ができる.

日本の場合, 基準局は海上保安庁[8], 国土地理院によって管理されており, 中波によるビーコンや FM 音声多重放送が使われる.

[8]船舶で使われるため.

また, 赤道上の静止衛星から GPS と同じ信号で補正データを送ることで測位精度を向上させる方法も利用されている. 米国ではインマルサットを利用した方式で WAAS(Wide Area Augmentation System) と呼ばれている (日本でも利用可能). 日本では運輸多目的衛星 MTSAT を利用する方式で MSAS(Multi-function Transport Satellite) と呼ばれている. 欧州ではインマルサット衛星を利用し, ENGNOS(European Geostationary-Satellite Navigation Overlay Service) と呼ばれている.

3.2.3 GPS と 1PPS

GPS では正確な UTC 時刻情報と共に, **1PPS**(1 Pulse Per Second) と呼ばれる正確に 1 秒を刻む時刻基準信号が送られてくる. この 1PPS 信号を使う事で非常に正確な基準周波数の発振器や原子時計 “並” の精度を持つ時計を作る事が可能である.

“並” という意味は, 電波は空間を伝搬して来る途中で空間状態 (電離層の電子密度変動など) やノイズによるジッター (ゆらぎ) があるため, GPS 衛星が搭載している原子時計と全く同じ精度にはならない所以である. それでも, 100 億分の 1(10^{-10}) から 1000 億分の 1(10^{-11}) の誤差精度が得られる. この精度はルビジウム発振器と同等である. そればかりか, クロックの較正が不要という点に大きなメリットがある.

3.3 CDMA

クアルコム社が開発した CDMA(Code Division Multiple Access) は国内で KDDI の携帯電話サービスで使われている通信方式である. CDMA という通信方式は, 基地局間の正確な時刻の同期 (基地局間同期方式) を必要とするため[9], CDMA の基地局には正確な時間と周波数基準を供給する高性能の GPS 受信機が置かれており, CDMA 基地局から時刻情報や 1PPS が携帯電話の電波に乗せて送られている.

この信号を受信すれば, CDMA を時刻源として使う事が可能である. CDMA は GPS に比べると信号を受信し易く, 受信機の設置場所をほとんど選ばないため (室内でも良い), 正確な時刻を必要とする可搬型のインターネット QoS 測定器や GPS アンテナを上げる事ができない場所で用いる事ができる.

[9]全ての基地局で下りに関しては同じ PN 信号 (Pseudo Noise, 擬似雑音. 拡散のために必要な一種の暗号のような信号) を使って, 各基地局間で位相をずらす事で制御しているため, 基地局間で精度の高い時刻の同期を取る必要があり, 全基地局の時刻同期に GPS を利用している.

3.4 その他

その他, NHK の時報 (時刻標準のみ), NTT の時報 (117), ISDN の網同期信号, FM 文字放送時刻情報 (±18ms), テレビの色同期信号 (カラーバースト信号: 3.579545MHz) などを使って, 時刻や標準周波数を取得する方法がある.

第4章　NTPの概要

4.1　NTPとは

コンピュータの内部時計はあまり信用できないため (家庭用ビデオレコーダよりも劣るといわれている), 正確な時刻を得てそれを維持するためにもっともよく使われる方法が, NTP を使ってインターネット上のタイムサーバから時刻を得る方法である. ここで使われる **NTP(Network Time Protocol)** は, コンピュータ同士が時刻情報を交換する方法を定めたプロトコルである.

今ではマイクロソフト社の Windows XP やアップル社の Macintosh でも標準的に NTP が使えるようになっており, インターネットに接続さえしていれば, 正確な時刻をネットワークから得られるようにあらかじめ設定されている.

NTP の代表的な実装にはデラウェア大学で開発された xntpd/ntpd と呼ばれるソフトウェアがある[1]. このソフトウェアは NTP を使ってサーバから正しい UTC 時刻を取得し, コンピュータの時刻を UTC に同期させる機能を持ち, Unix 系 OS や Windows NT 上で動作する.

xntpd/ntpd はインターネット上やキャンパス LAN 内にある NTP サーバから UDP(ポート番号 123) を使って, 時刻を取得する機能がメインだが, コンピュータのシステム時計の精度を維持する機能, 原子時計のような精度の高いクロック源や GPS のような時刻配信サービスを利用してネット上に時刻の供給が可能なリファレンス・タイムサーバを立ち上げる機能を持つ.

4.2　NTP以前の時刻調整手段

NTP バージョン 1 が登場したのは 1985 年である. しかし, NTP が開発される前にもコンピュータ同士が時刻を合わせる手段は存在していた. 最初に開発された手法は, Daytime(RFC867) や Time(RFC868) で, その瞬間の時刻を相手のコンピュータに合わせるプ

[1] 他に, Richard Curnow 氏 (rc@rc0.org.uk) が開発している chrony と呼ばれるソフトウェアもあるが本書では取り上げない.

ロトコルで, TCP/IP の標準プロトコルスタックとして, TCP/IP を搭載するあらゆるコンピュータに実装されている.

Daytime と Time は基本的に同じ機能だが, その違いは Daytime が人間にとって認識し易い形式で時間と日付を返すのに対し, Time は UTC からの経過秒数を 32 ビットバイナリ数で返す点である. TCP でも UDP でも利用できるが, 一般的に TCP 上で利用される. 例えば, Telnet クライアントを用いて, Daytime のポート番号である 13 にアクセスすると次のようになる.

```
# telnet localhost daytime
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
Sat Jul  5 15:08:46 2003
Connection closed by foreign host.
```

一方, Time プロトコルを利用するには, `rdate`[2] と呼ばれるクライアントが利用できる.

```
# rdate -p 10.1.1.1
[10.1.1.1]      Mon Jul  7 16:42:21 2003
```

但し, これら二つのプロトコルは実装されてはいるがセキュリティ上の都合で使われない状態に設定されている場合が多い.

その後, 4.3BSD Unix において, LAN 関連のアプリケーションが強化され, LAN 上の多数のコンピュータ間の時刻を合わせるためのアプリケーション `timed` が開発された. `timed` はマスタ・スレーブ方式で時刻を合わせるデーモンプログラムで, まずスレーブホストはマスターとなる正確な時刻を持つ Unix マシンにネットワーク時刻を問い合わせ, その時刻に合わせる (通常は起動時). 以後, スレーブホストはマスタが定期的に配信するブロードキャスト同期メッセージの時刻に修正し, その結果としてネットワーク内にある全 BSD UNIX マシンの時刻を一定に保つ事ができる. `timed` は UDP のブロードキャストを使用するため, 広域網では利用できないが, LAN の中では手軽に利用できるアプリケーションであった. しかし, NTP の登場と普及で, この `timed` が利用される事はほとんど無くなった.

[2] OpenBSD の `rdate` は SNTP や閏秒にも対応するよう拡張されてている.

4.3 NTP の歴史

NTP の開発は古く, 1985 年に開発の元になる仕様が規定された (バージョン 0, RFC 958). ここではプロトコルの大まかな設計 (サーバクライアントモードを基本とした基礎的な計算方法) が規定されただけで, 周波数エラーを補正するような機能は考えられていなかった.

最初の完全なプロトコル仕様は 1988 年に NTP バージョン 1(RFC1059) として発表された. このバージョンではクライアントサーバモードのみならず, シンメトリック・オペレーションモードが記載されている.

1 年後の 1989 年に出たバージョン 2(RFC1119) では対象鍵認証が追加された. ほぼ同じ時期に時刻同期プロトコルとして Digital Time Synchronization Service(DTSS) が DEC 社から発表されている. そして, この時期にトロント大学の Dennis Fergusson 氏によって, xntp と呼ばれるソフトウェアが開発され, 広く公開された.

1992 年には DTSS とマージされ, アルゴリズムの改良やブロードキャストモードの追加が行われ, NTP バージョン 3(**RFC1305**) が規定された. 現在, インターネット内で使われているほとんどの NTP サーバはバージョン 3 に基づいている.

1994 年には NTP だけではなく, オペレーティングシステムのクロック調整に関する実装やインタフェース仕様が発表され (RFC1589), コンピュータに 1 マイクロ秒の精度を持たせることができるようになった. そして, バージョン 3 の仕様を元に xntp の改良が行われ, xntp3 が公開された. 1996 年にはよりシンプルで IPv4 以外のプロトコルでも利用できる, **SNTP**(Simple Network Time Protocol) の仕様 (RFC2030) が発表されている (第 8 章 SNTP).

そして, 今までは原子時計や GPS のような正確なパルスを Unix が持つクロックにキャリブレートし安定して動して, OS の時計を正確にする方法が独自に開発されていたが, その標準的なデザインを API として 1999 年に発表している (RFC2783). 現在はバージョン 4 の開発が進められている.

以上が NTP の開発の歴史である (表 4.1).

NTP の特徴は列挙すると次のようになる.

- NTP は UTC を標準時刻とし, リファレンスクロックを持つサーバに時刻を同期させる事でネットワーク上の全てのコンピュータの時計を UTC に合わせることができる.
- NTP はフォールト・トレンラトなプロトコルで, 複数の時刻源から最適なものを自動的に選択して, 時刻を合わせていく.
- NTP はリファレンスクロックを頂点とする階層構造化する事で, 高いスケーラビリティを確保している.

年代	歴史
1980	数百 ms の精度で時間同期させられる最初実装
1985	RFC 958 (NTP バージョン 0)
1988	RFC 1059 (NTP バージョン 1)
1989	RFC 1119 (NTP バージョン 2)
1989	RFC 1128 (NTP の性能について)
1989	RFC 1129 (インターネットにおける時間同期)
1990	RFC 1165 (OSI 上での NTP)
1992	RFC 1305 (NTP バージョン 3)
1992	RFC 1361 (SNTP バージョン)
1994	RFC 1589 (精度計測のカーネルモデル)
1994	RFC 1708 (NTP の OSI プロトコル仕様)
1995	RFC 1769 (SNTP バージョン 3)
1996	RFC 2030 (SNTP バージョン 4)
1999	RFC 2783 (PPS API のデザイン)
2005	RFC 4075 (DHCPv6 の SNTP 設定について)
2006	RFC 4330 (SNTP バージョン 4 の改定)

表 4.1: NTP の開発の歴史 (RFC)

- NTP は様々な時刻源を利用可能にするため, 高い精密さ (accurate) とナノ秒クラス (2^{-32} 秒) の分解能をもっている.
- ネットワークが一時的に使用不可になっても, NTP は過去のデータから現在の時刻とエラーを推定できる.
- NTP はローカル時計の精度の計算管理を行う.

4.4 NTP バージョン 4

NTP はバージョン 1 から開発が始まり, 現在はバージョン 4 の開発が進行中である. 最新の NTP ソフトウェアはバージョン 4 対応を謳っているが, 現時点でバージョン 4 は RFC 化されていないためインターネット標準は未だバージョン 3 である[3]. なお, ベンダーによっては独自に NTP プロトコルを改良しているケースもある.

NTP バージョン 3(NTPv3) は 1992 年に開発されたもので, 当時の混雑したインターネット回線と数十メガヘルツ (MHz) の CPU クロックを持つ Unix ワークステーションを元にアルゴリズムがチューニングされている. ところが, 現在ではギガビットのスピードを持つネットワークと数ギガヘルツ (GHz) のクロック周波数の CPU を持つ PC が一般に利用される環境に進化しており, それに合わせる形で NTP のアーキテクチャ, プロトコル, そしてアルゴリズムが改良され, NTP バージョン 4(NTPv4) の開発が進められている.

NTP の研究やバージョン 4 の実装は主にデラウェア大学の David Mills 博士が中心となっている NTP プロジェクト[4]で進められており, NTP バージョン 4 の特徴は以下のとおりである.

- 64 ビット固定小数点形式から 64 ビット倍精度浮動小数点形式に変わった.
- ポーリング間隔, ジッタのハンドリングや精度を改善するため, 時刻規律 (dicipline) アルゴリズムの再設計を行った[5]
- ナノ秒精度が出せるナノカーネルをサポートした.
- *Autokey* として知られる公開鍵暗号技術をサポートした.
- 自動サーバ発見メカニズム (メニーキャストモード) をサポートした.

[3] Internet Draft, `draft-ietf-ntp-ntpv4-proto-04.txt` が出ている.
[4] DARPA, NSF, NWSC や NASA の援助で勧められている (`www.ntp.org`).
[5] 一例として, ダイアルアップ環境で使う場合ポーリング間隔を 1 日以上に設定できる (但し, 十分クロックが安定しているならば).

- ダイアルアップアクセスのようなネットワークで素早く時刻同期を行うメカニズムをサポートした (burst モード).
- リファレンスクロックドライバの新規追加及び一部更新を行った.
- 動作するオペレーティングシステムを増やした.
- `enable`/`disable` コマンドの引数が変更され, `authenticate` コマンドが削除された.

NTP は良く設計されたプロトコルで, バージョンの互換性の問題は古いバージョンのクライアントが新しいバージョンのサーバを使う限りはあまり問題になる事は無い.

4.5 NTP が扱うタイムスケール

NTPv4 では NTP 時間のフォーマットは 128 ビット長, 64 ビット長, 32 ビット長の三つが定義されている. いずれも共通するのは, 1900 年 1 月 1 日からの経過秒数で表される点と, ビット長を半分に区切って, 前を秒に後を小数点以下の端数を表す形式となっている. 128 ビット形式は NTP の処理の中では用いられてはいないが, ファイルシステムや専用ハードウェアなどで用いらることを念頭においているデータ表現形式である. 64 ビット形式は NTP の処理の中で日時を表すために用いられるタイムスタンプ形式で, 32 ビット形式は遅延時間などを表すのに用いられるショート形式である.

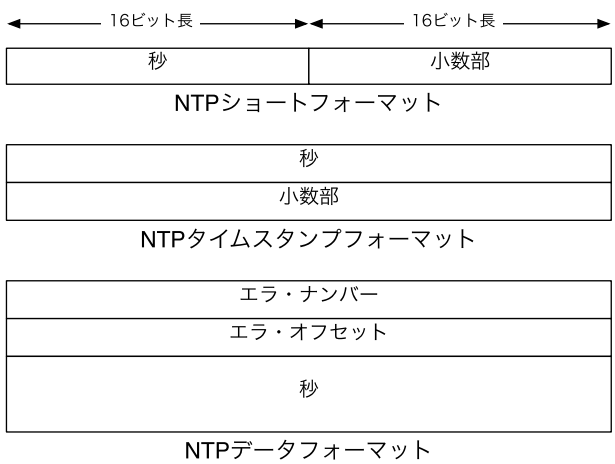

図 4.1: NTP タイムフォーマット

一般に NTP 時間は 64 ビットで表される形式で, 32 ビットの符号無し整数で表す経過秒数 (最大 136 年) と, 32 ビットの小数点以下の端数秒を意味し, この事から, NTP の分解能は $1/2^{32} = 232.8306 \times 10^{-12}$ 秒, すなわち 232 ピコ秒となる.

一方, Unix のエポック時間は 1970 年を起点としているので, 二つの値に違いが出てくる. 例えば, 2000-08-31 18:52:30.735861 の値は次のようになる.

Unix: 39aea96e.000b3a75
NTP: bd5927ee.bc616000

また, NTP の時間は 1900 年 1 月 1 日からの経過秒数を表されるため, 32 ビットの秒カウンタが 136 年サイクルで一回りする事になり, このままでは 2036 年 2 月 7 日 6 時 28 分 16 秒 (日本時間同日午後 3 時 28 分 16 秒) に桁が溢れてしまう. この問題は「**NTP の 2036 年問題**」として知られている[6].

そこで, RFC2030 では上位 32 ビットを 1 ビットの状態ビットと 31 ビットの経過秒数に分けてアプリケーション側で判別する事になった. 状態ビットが “1” の場合は現在の経過秒が 1968 年から 2036 年の間に有る事を示し, “0” の場合は 2036 年以降である事とした. この方法により, 2104 年までは対処が可能となる.

4.6 NTP の通信モード

NTP を実行しているコンピュータ間の通信は**アソシエーション**と呼ばれ, 2 つのピア間で NTP メッセージを交換し, 一方ないしは両方が NTP アソシエーションを形成し, 両者の間で正確な時刻が維持される.

NTP は小さなストラタム値を持つサーバからクライアントに対して時刻メッセージが送られるが, ネットワークやクロック源の環境が変わる場合があり, 必ずしも安定しているとは限らない. そのため, NTP は複数のサーバから一番良いもの, すなわちストラタム値が低く (一次時刻参照に近い), ネットワーク遅延が少なく, そして要求する時刻精度を持つクロック源やタイムサーバを選択する.

NTP サーバそれぞれが確立するアソシエーションには次のモードが用意されている.

- クライアント・サーバ
- 対称アクティブ/対称パッシブ
- ブロードキャスト
- メニーキャスト (第 9 章)

[6] 但し, この問題は Unix の 2038 年問題のようにプログラム中で時間計算を行うわけではないので, それほど深刻ではないと考えられる.

4.6.1 クライアント・サーバモード

クライアント・サーバモードは NTP の設定としても最も一般的である. このモードでは, クライアントはサーバのストラタムや到達性に関わらず, 定期的にサーバモードで動作するホストに時刻を要求するメッセージを送出し、応答された時刻に合わせる. サーバ側はクライアントからの要求メッセージに対して, 自分が持つ時刻を応答する.

4.6.2 対称アクティブ/対称パッシブモード

対称 (Symmetric) モードにはクライアントとサーバの区別は無く, ホストが能動的に NTP メッセージを送出するかしないかで, 二つのモードに分けられる.

対称アクティブモードのホストはピアへの到達性 (reachability) やストラタムに関係なく, 定期的に NTP メッセージを送出し, 時刻の同期を行う. 一方, 対称パッシブモードで動作するホストは対称アクティブモードで動作しているホストからのメッセージが到着すると, アソシエーションが形成される. 通常, ピアがそのホストと同等かそれより小さなストラタムで動作し, 到達性がある限りアソシエーションは継続する. パッシブモードで動作するサーバはピアに対して, 時刻のリファレンスサーバとして動作するよう使われる.

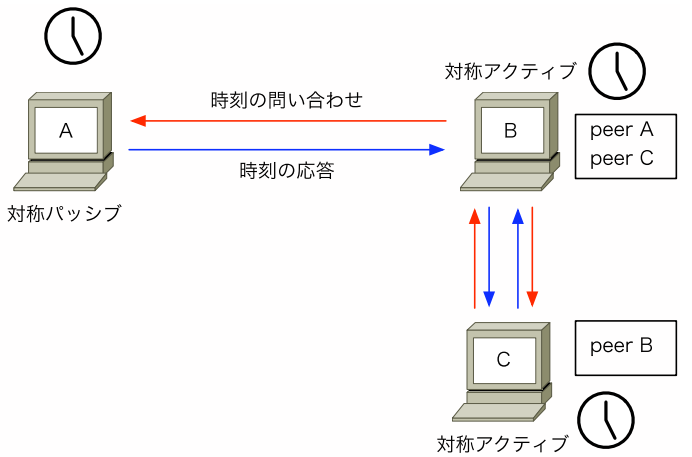

図 4.2: NTP 対称モード

4.6.3 ブロードキャスト/マルチキャストモード

ブロードキャストモードは主に LAN で動作させるためのモードで, ピアのストラタムや到達性に関係なく定期的に時刻メッセージを送出するホストと, 送出された時刻メッセージを受信して時刻同期するホストがある. 時刻の配信にマルチキャストが使われるモードがマルチキャストモードとなる.

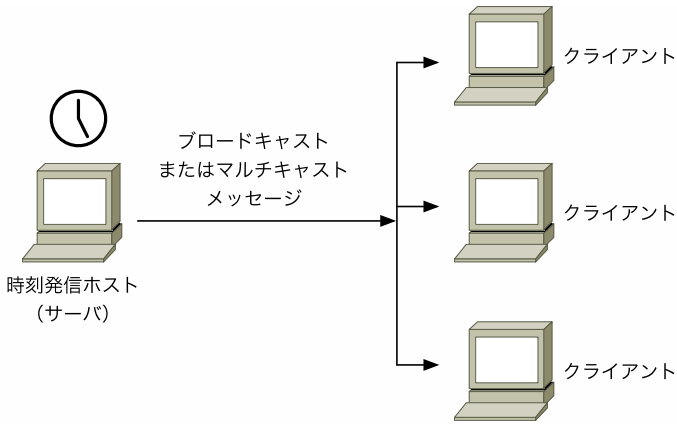

図 4.3: NTP ブロードキャストモード

4.7 NTP のアーキテクチャ

NTP が時刻を合わせる原理を簡単に説明する.
NTP の時刻の合わせ方はマシン同士が自身の持つ時刻情報を交換する事で, 相互に時刻をチェックし, 時刻を補正していくという方法を取っている. 時刻がほぼ同期した NTP サーバ同士は 1 分間につき 1 パケット送るだけで, LAN 内の 2 台のコンピュータをほぼ 1 ミリ秒以内に同期化する事が可能である.

NTP の最大の特長はサーバ間の接続が階層構成を持つ点にある (図 4.4). 時刻源からそのコンピュータがどれだけ離れているかを**ストラタム**と呼ばれるホップ数で示され, サーバは時刻源をルートとした論理的な階層型ネットワークを形成する. ストラタム 1 の NTP サーバは正確な時刻源から直接 UTC 時刻を受け, ストラタム 2 の NTP サーバはストラタム 1 の NTP サーバから時刻を受信する. 同様に, 以降のストラタムの NTP サーバも時刻を順次階層的に受信していく. この階層は 15 段階まで許されている. また, 複数のサーバの中からサーバを選択する場合にストラタムの数が低いサーバが優先されるようになって

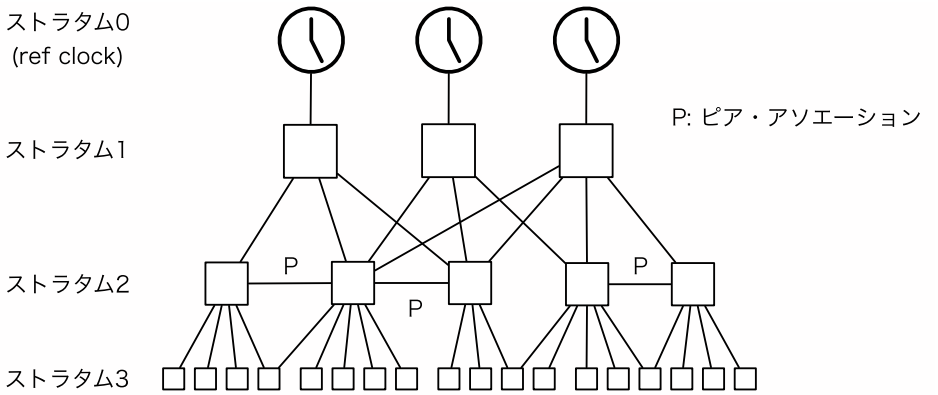

図 4.4: NTP ネットワークの階層

いるが, NTP は多数の NTP サーバとピアする事を許しており, 同一ストラタムのサーバから最適の時刻を選択するメカニズムを持っている.

ストラタム値は NTP サーバを起動する際に指定できるが, ストラタム 1 以外の NTP サーバは NTP サーバとの**アソシエーション**手続きの中で**自動的に決定される**. そして, NTP のネットワークがストラタム 1 のサーバ群を頂点にツリー上に形成されていく (図 4.5).

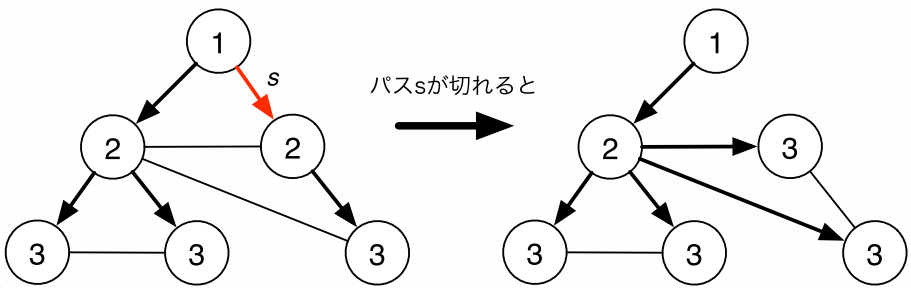

図 4.5: NTP に時刻同期の変化

NTP は時刻情報のループを避けるために, (1) 時刻ソースに 32 ビットの “リファレンス識別子 (reference identifier)” を振り, (2) ストラタム数の上限を 15 とし, (3) サーバは必ず低いストラタム値を持つサーバを優先するという約束がある. このようにする事で, NTP の論理ネットワークがループ構成にならないようにしている.

また, インターネットでは往復遅延, ジッタなどが時間とともに刻々と変化するため, NTP はクライアントとサーバの間で次の三つの変動する数値を利用して時刻同期の堅牢さを高めている.

- ネットワーク遅延 (往復遅延)
- パケット交換に要する分散 (dispersion)：参照時刻の揺れを数値化したもので, 二台のホスト間の最大クロックエラーの値
- クロックオフセット：参照した時刻と内部時刻の差

そのため, 2000km も離れた広域なネットワーク内でも 10 ミリ秒 (0.01 秒) レベルの同期が, LAN レベルでは 1 ミリ秒 (0.001 秒) の同期が可能となっている.

そして, NTP は次の二つの方法で時刻が不明確な装置との同期を回避している.

- 自動的に同期化されない装置とは決して同期化しない.
- 複数の装置から報告された時刻を比較し, ストラタム値が小さくても, 他よりも著しく異なる時間を持つ装置とは同期化しない. 従って, 複数台とアソシエーションを構成する事が望ましい.

4.8 時刻調整アルゴリズムの簡単な考え方

NTP の内部構造を説明する前にネットワークを使った簡単な時刻調整の考え方を説明する. 時刻調整の鍵となるのはネットワークの遅延とクロックオフセット (時間差) で, NTP ではマシン間で図 4.13 のような NTP メッセージを交換して調整を行う. このメッセージの中にはクライアントがメッセージを発した時間 (T_1), サーバがメッセージを受信した時間 (T_2), サーバがクライアントに応答メッセージを発した時間 (T_3), そしてクライアントがメッセージを受信した時間 (T_4) が入っている. これらの時間を使って, 時刻同期が行われる.

例えば, クライアントがサーバの持つ時刻に同期を行う場合, メッセージの流れは図 4.6 のようになる. 非常に単純に考えると, クライアントは時間差分 (θ_0) だけ時刻を遅らせたり進めたりすれば, リファレンスとなった NTP サーバの時刻と同期が取れる事になる.

どれだけサーバとクライアントに時間差があるかはオフセット時間を算出することでわかる (この時, 往復の遅延時間は対称であることが前提である). このオフセット時間は,

$$\theta = \frac{(T_2 - T_1) + (T_3 - T_4)}{2} \tag{4.1}$$

で算出できる. また, 往復遅延時間 δ は,

$$\delta = (T_4 - T_1) - (T_3 - T_2) \tag{4.2}$$

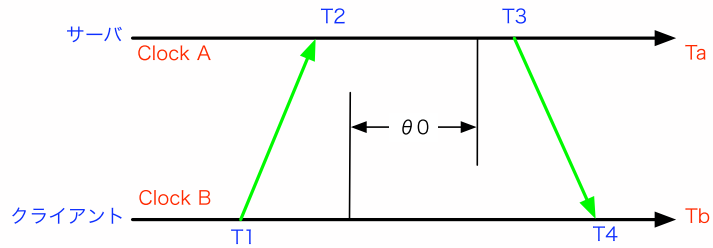

図 4.6: ネットワーク遅延と時間差

で算出できる. わかりやすいように, 具体的な数値でそれを確かめてみる.

まず, 計算がしやすいように 1 対 1 のクライアントサーバのモデルを使う事とし, リファレンスとなるサーバの時間を 0:10, クライアントの時間 0:00(すなわち, 10 分遅れていると考える), ネットワーク遅延は送受対象でそれぞれ 10 分, サーバの処理遅延を 5 分とする.

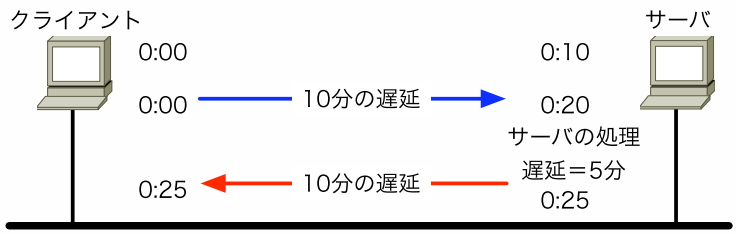

図 4.7: NTP の時間計算の例

クライアントが受け取るメッセージは次のようなタイムスタンプが書き込まれる.

- $T_1 = 0{:}00$ (クライアントがメッセージ発した時間 0:00)
- $T_2 = 0{:}20$ (サーバがメッセージを受信した時間 $0 : 10 + 10$)
- $T_3 = 0{:}25$ (サーバがクライアントにメッセージを送信した時間 $0 : 20 + 5$)
- $T_4 = 0{:}25$ (クライアントがメッセージを受信した時間 $0 : 00 + 10 + 5 + 10$)

すると, 遅延 δ は “$(0 : 25 - 0 : 00) - (0 : 20 - 0 : 25)$” で 30 分. クロックオフセットは, “$((0 : 20 - 0 : 00) + (0 : 25 - 0 : 25))/2$” で 10 分となる. クライアントは 0:00 から 10 分進めれば, サーバ側の時刻と同期ができる事になる.

4.9 揺らぎと分散

NTP では 64 台までのサーバと関係 (relationship) を持てるが, そのうち 10 台が時刻同期のソースになり得るサーバになる. 参照するサーバ数を多くすることで, より正確な時刻が得られる可能性が大きくなる上, 更に障害が起きた場合に時刻源の非参照時間を短くできるというメリットがある. これらの長所はクラスタリング・アルゴリズムとクロック・コンビネーション・アルゴリズムによって実現されている.

NTP は正確な時刻との同期に加え, 要求と応答を行った際のオフセット値の統計的な揺らぎ (ジッタ) も計算している. クライアントはジッタが小さいサーバほど, 正確な時刻同期が可能であると判断する. この最大のジッタ幅を別の言い方で分散 (dispersion)[7] とも呼んでいる. オフセットと遅延の相関関係の統計を取ると図 4.8 のようになる.

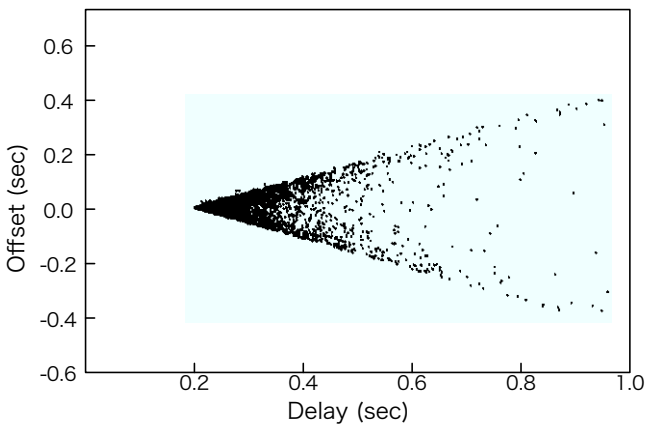

図 4.8: オフセットと遅延の散布図

従って, NTP の時刻選択の要素となるものは, クライアントとサーバの時間差であるオフセット値 (θ), 遅延 (δ), エラーの幅を示す分散 (ϵ) である.

オフセットと分散の関係をもう少し説明する. オフセット上の最大可能エラーが分散と表され, クライアントの時間をサーバの時間に合わせるための調整は, オフセットと分散を加えたものとオフセットから分散を引いたものの間になければならない (図 4.9).

$$\theta - \frac{\delta}{2} - error <= \text{調整時間} <= \theta + \frac{\delta}{2} + error \tag{4.3}$$

[7] 分散値の算出方法は RFC 1305 の Appendix H に詳しく記述されている.

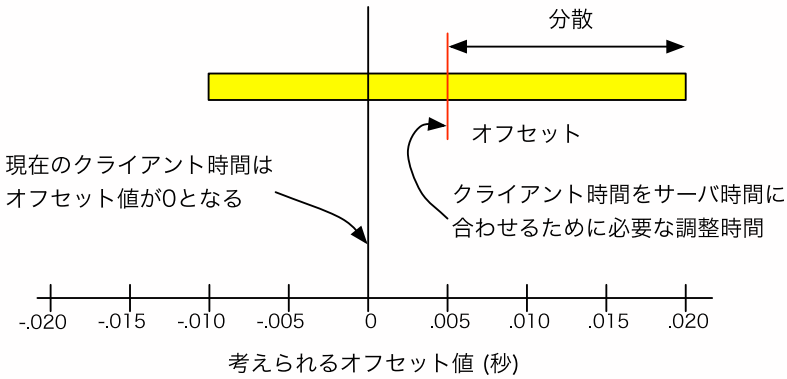

図 4.9: オフセットと分散の関係

4.10 NTP の内部構造

NTP のアーキテクチャは多数の NTP サーバと時刻同期を行う機能だけで無く, あまり正確ではないカーネル時計を修正する機能も合わせ持っている. 実際の時刻調整は 4.8 節の説明よりも複雑だが (RFC 1305, Mills 教授の NTP の論文やプレゼン資料が参考になる[8]), 大まかな流れは, 次のリストのあげた順番で行われる. そして, 精度を上げるために NTP のソフトウェア構造は図 4.10 のようなハイブリッドなフェーズロックループ (PLL/FLL) を構成している.

- サニティ・チェック (健全性のチェック)
- フィルタリング
- インターセクションアルゴリズム
- クラスタリングアルゴリズム
- クロックコンビネーション

4.10.1 サニティ・チェック

NTP はノイズデータや (ストラタム的に) ループされて届いたパケットを避けるよう設計されている. そのため, 送られてきたパケットを処理する前にそのパケットが有効かどうかの検査がサニティ・チェックで行われる. このチェックをパスしたパケットが次の手順に

[8] http://www.eecis.udel.edu/~mills/を参照.

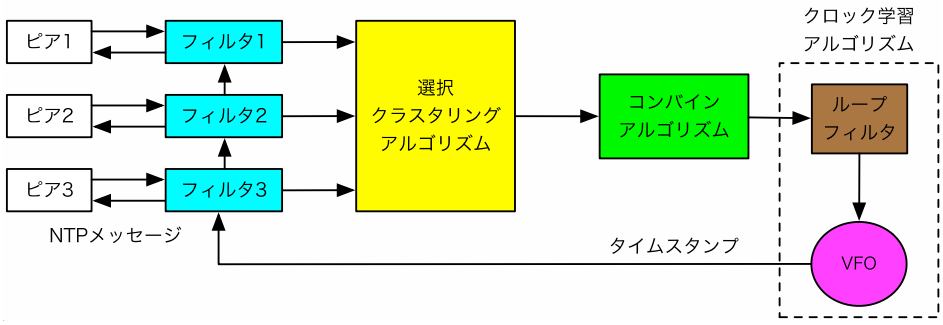

図 4.10: NTP アーキテクチャ

進む. そのサニティ・チェックは次のような項目の検査が行われる.

1. 認証 (Autokey や MAC) が成功するかどうか.
2. パケットが同一サーバから二重に届いていないかどうか.
3. パケットがクライアントによって送出された最後のパケットと同じクライアント・タイムスタンプを含んでいるかどうか.
4. サーバが到達可能かどうか.
5. サーバへの往復 (ラウンドトリップ) 遅延と分散を計算し, 16 秒より小さい値でなければならない.
6. サーバのクロックは, 1 日以内に更新された外部時刻源と同期されなければならない.
7. サーバはクライアントよりも同じあるいは低いストラタム値でなければならない.
8. 総遅延と最大エラーがルートクロックから 16 秒より小さい値でなければならない.

4.10.2 フィルタリングとインターセクションアルゴリズム

NTP では受信したパケットは一旦送信元別のキューに入り (キューが溜まるまで時刻同期は行われないことを意味する), このパケットをサンプルにして, オフセット, 遅延及び分散などから一番大きなエラー値を得る. これがフィルタリング機能である.

そして, フィルタから得られたデータを比較して, 正確な時計の判断基準点 (インターセクション) を決定し, 各ピアのエラー幅から判断基準点にかかるクロックは正確な時計 (true chimers), かからないクロックは不正確な時計 (falsetickers) と判断する (図 4.11). ほとんどの場合, 正確な時計になるが, falseticker があれば, 選択アルゴリズムから排除される (falseticker の多くは内部時計を使っているサーバである).

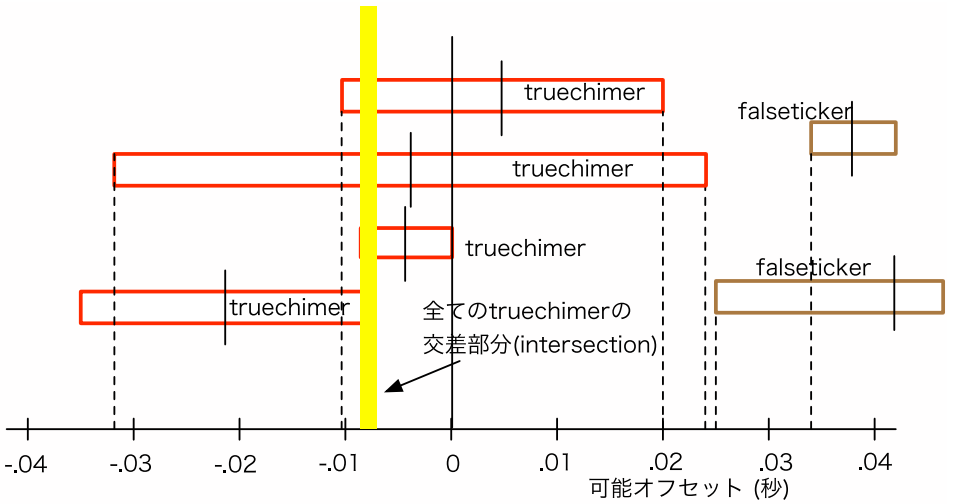

図 4.11: インターセクションアルゴリズム

なお, サーバが falseticker であれば, `ntpq -c peers` の出力結果でサーバ名の前にタリーコード “x” が付く (12.1.4 節, 152 ページ参照).

4.10.3 クラスタリングアルゴリズム

クラスタリングアルゴリズムはサーバの特性を決定する様々な要因 (ストラタム, ルートクロックからの分散値, クライアントへの分散値, ルートクロックからの遅延, クライアントへの遅延, 考えれるスキューエラーなど) の重み付けされた組合せに基づいて truechimer を並び替える (このうち, ストラタムは非常に重要な決定要素になる). そして, この中から truechimers ベスト 10 が選択される. truechimer が 10 以上あって, リストから捨てられたサーバには `ntpq -c peers` の結果でサーバ名の前に “.(ピリオド)” が付く.

4.10.4 コンバインアルゴリズム (クロックコンビネーション)

truechimer の中で検査に合格したサーバが, いつでも参照できるサーバの候補となるが, そのうち一番精度が良いサーバと同期をする. なお, クライアントの時計の補正は, クラスタリングアルゴリズムで選択された全てのサーバの影響を受け調整される.

4.10.5 ステップ調整とスリュー調整

NTP では, カーネル時計と NTP の時刻に大きな時間差があったり, ネットワーク遅延時間が瞬間的に大きくなった場合, いきなり正しい時刻に合わせる (ステップ調整) のではなく, 緩やかに時間が合う方法 (スリュー調整) を取る方法が採用されている.

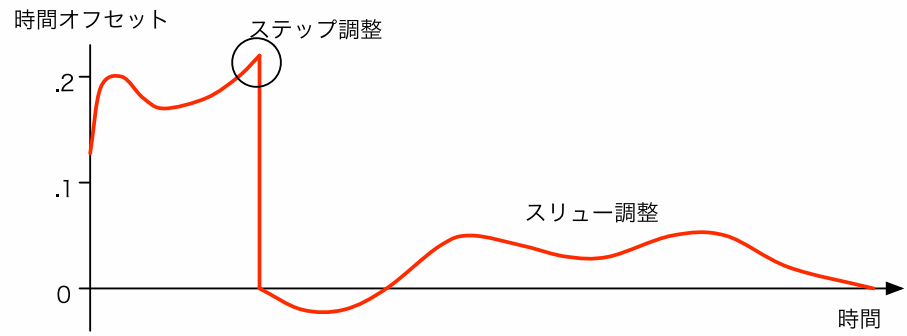

図 4.12: ステップ調整とスリュー調整

この緩やかに微調整しながら時刻を合わせるのに `adjtimex(2)` システムコールが使われる. `adjtimex(2)` の特徴は時間を逆行しない点にある. 時間が進んでいる場合は時刻をゆっくり進めて数時間かけて一致させ, 遅れている場合は時刻の進め方を速める事で追い付くようにする. 実際には, NTP デーモンは `settimeofday(2)` と `adjtimex(2)`(もしくは `ntp_adjtime(2)`) の二つのシステムコールを用いている. `adjtimex` は, 1 秒のずれに 2000 秒かけて補正するため, 大きくずれている (128 ミリ秒以上ずれている) 場合は, `settimeofday(2)` を用いる事になっている.

以上を含め, NTP はシステムクロックの調整に 4 つの基本的なシステムコールを利用する.

- `settimeofday(2)` は時間のステップ調整を行う.
- `adjtimex(2)` は時間のスリュー調整を行う.

- `ntp_adjtime(2)` は `adjtimex(2)` の機能に加え, 下記のソフトウェアクロックのパラメータを制御する.
  - ソフトウェアクロックのオフセットの調整
  - ソフトウェアクロックの仮想周波数 (frequency) を直接調整
  - PPS イベント処理の有効/無効
  - 閏秒の制御処理
  - クロックの特有な値を読み込みと設定
- `hardpps()` は, 外部の PPS シグナルにカーネル時計を同期させる際に用いられる (11.3 節, 133 ページ参照).

NTP は定期的にシステムクロックと細かな調整を行う事で正確な時刻を維持しようとする (スリュー調整) が, 実装によってはこの調整がうまくできないものがある. しかもシステムクロックの品質をモニターする標準的なインタフェースが存在していない.

従って, 正確な時刻維持のために必要となる機能を RFC1589(“A Kernel Model for Precision Timekeeping”) で提案している. 概要としては, クロックの問い合わせと制御を行う新たに改良を施したシステムコール, 1 マイクロ秒の精度を持つシステムクロック, クロックのモニター方法の提案である.

4.11 NTP の実力

ネットワーク上で NTP を使って時刻合わせを行う場合, クライアントとサーバの時刻差が 128 ミリ秒以下である事が求められている. これ以上ずれが起こると、時刻の同期調整を中断する. インターネット上での誤差はネットワーク遅延の変動を考えても 5 ミリ秒から 100 ミリ秒の間にほとんどが入るため, 通常の状態で中断される事はまずない. 以前の Nelson Minar 氏が行った調査 (A Survey of the NTP Network)[9] では, NTP サーバの 90%がネットワーク遅延 100 ミリ秒以下に収まっており, 99%が 1 秒以内に同期しているという結果が出ている. 実際にインターネットでどの程度の差なのかは大規模に調査されていないが, Terje Mathisen 氏によれば, 400 のサーバを見たが 2 ミリ秒以下であったと報告している.

一例として, 50 マイクロ秒の精度を持つような PPS による時刻同期では, Pentium クラスの CPU を使えば, 0.1PPM 以下の変動に抑えられるといわれている.

NTP サーバがどの程度のクライアントに時間を供給できるか, CPU の処理能力, ネットワークの帯域等さまざまな要因に影響される. 明確な数値は出されていないが, かなり多く

[9] `http://www.media.mit.edu/~nelson/research/ntp-survey99/`

(数十万台規模) のクライアントに時間を供給できると考えられている.

Terje Mathisen 氏によれば, NTP のために必要となるネットワーク帯域について次のような机上検討がなされている. 256 秒間隔で時刻のやり取りを 50 万台のクライアントが行うと想定すると, 下記のように 1 秒間に 4000 パケットが NTP のメッセージ交換でやり取りされる.

$$2/256 \times 500000 \doteq 4000 \text{ パケット} \tag{4.4}$$

パケットサイズに認証を考慮し, メッセージ長を 128 バイトとすると, $128 \times 4000 \doteq 512KB/s$ となり, 50 万台のクライアントを持つ通信としては 4Mbps 程度の帯域が必要となる.

4.12 NTP のメッセージ

4.12.1 ヘッダフォーマット

図 4.13 が NTP メッセージのヘッダフォーマットである.

4.12.2 各ヘッダフィールドの内容

閏秒インジケータ (Leap Indicator)

最初の 2 ビットは閏秒インジケータ (LI) で, 閏秒の警告用の 2 ビットコードである. このビットは閏秒の挿入日の 23:59 以前に設定され, 次の日の 00:00 でリセットされる (図 4.14 参照).

閏秒インジケータは, ストラタム 1 サーバでは手動によって設定されるが, それより下位のサーバは閏秒インジケータの値がそのまま NTP パケットに反映される. 閏秒インジケータの値は, 表 4.2 のような意味を持つ.

0	警告無し (通常の NTP メッセージ)
1	最後の 1 分が 61 秒になる
2	最後の 1 分が 59 秒になる
3	警告状態で, 時刻が同期していない

表 4.2: 閏秒インジケータの値

バージョン (Version)

次の 3 ビットは NTP のバージョンを示している (現在は 4).

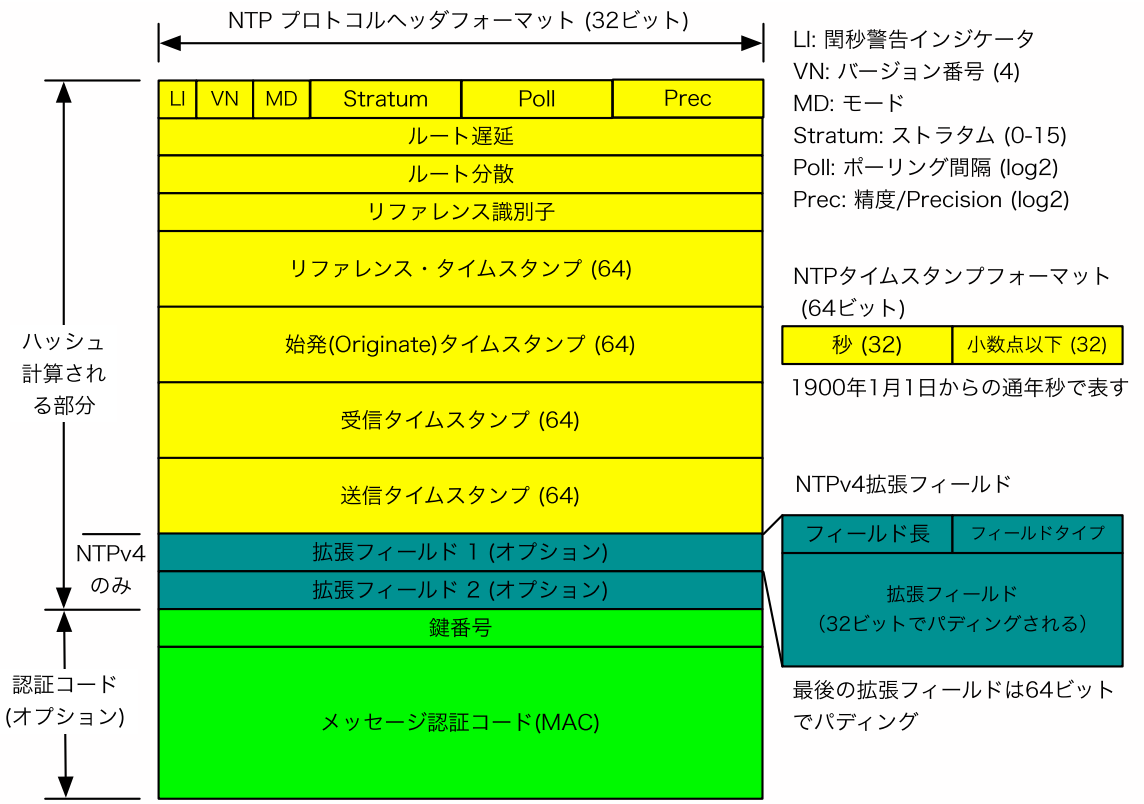

図 4.13: NTP のプロトコルヘッダフォーマット

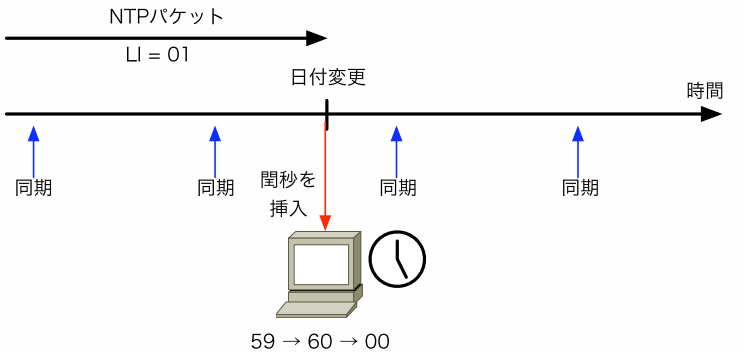

図 4.14: 閏秒インジケータの役割

モード (Mode)

次の 3 ビットは表 4.3 のような動作モードを示す.

0	予約
1	Symmetric Active
2	Symmetric Passive
3	クライアント
4	サーバ
5	ブロードキャスト/マルチキャスト
6	NTP 制御メッセージ (ntpq)
7	ntpdc が使う NTP 制御メッセージ (NTPv4 のドラフトではプライベートで利用)

表 4.3: モードの値

ストラタム (Stratum)

その次の 8 ビットは表 4.4 のように NTP サーバのストラタム階層値を示す. ストラタムフィールドで表される数値は 255 まであるが, ストラタム値は 0〜15 までしか許されていない.

階層	意味
0	不明または無効
1	一次参照 (原子時計や GPS などの時刻を参照している)
2〜15	二次参照 (NTP で時刻を参照している)
16	クライアント
17〜255	未定義

表 4.4: ストラタム値

ポール間隔 (Poll Interval)

これは, 転送されたメッセージ間の最大間隔を示す 8 ビットの符号付き整数で, 最も近い 2 の冪乗で表される (すなわち $log2$ の値). 単位は秒で, 例えば 6 であれば, 2^6 となり 64 秒の最大間隔を示す事になる. 現在このフィールドに現れる値は, 4(16 秒) から 14(16284 秒) の範囲である [RFC2030 より].

精度 (Precision)

これはローカル時計の精度を表す 8 ビット符号付き整数値で, ポール間隔と同様に 2 の冪乗で表され (すなわち $log2$ の値), 単位は秒である (ρ). 負の値になれば, 小数点以下の値となる.

ルート遅延 (Root Delay)

これは一次参照源までの往復遅延の合計を示しており, 32 ビットの NTP ショート形式で表される (δ). この値は, 時計の精度とスキュー (skew) に依存し, 相対時間と周波数変位によって, 正と負の値を取り得る.

ルート分散 (Root Dispersion)

これは一次参照源との名目的な相対誤差を示しており, 32 ビットの NTP ショート形式で表される (capepsilon).

リファレンスクロック識別子 (Reference Clock Identifier)

参照している時刻を見分けるための 32 ビットコードで, ストラタム 1(プライマリ NTP サーバ) の場合, リファレンスクロックを表す 4 文字のアスキー文字列が入る (4 文字未満の場合, 残りは 0 が入る). ストラタム 2 から 15 までは参照サーバが IPv4 アドレスであれば 32 ビットの IPv4 アドレスが, IPv6 の場合はアドレスを MD5 ハッシュした上位 32 ビット

の値が入る.

NTP の一次参照源は, 表 4.5 のようなコードが使われる.

コード	外部参照源	コード	外部参照源
GOES	静止実用環境衛星	GPS	全地球無線測位システム
PPS	原子時計または秒単位の発振器源	IRIG	Inter-Range Instrumentation Group
WWVB	米国の長波の標準電波	DCF	独の長波の標準電波
HBG	スイスの長波の標準電波	MSF	英国の長波の標準電波
JJY	日本の長波の標準電波	LORC	LORAN-C
TDF	仏の中波の標準電波	CHU	加の短波の標準電波
WWV	米国の短波の標準電波	WWVH	米国の短波の標準電波
NIST	米国標準技術局電話サービス	USNO	米国が海軍天文台の電話サービス
PTB	独の物理工学研究所の電話サービス	LOCL	外部の時計を参照しないローカル時計
CDMA	CDMA		

表 4.5: 参照クロック ID

リファレンス・タイムスタンプ (Reference Timestamp)

ローカル時計が最後に更新 (設定あるいは修正) された時刻で, 64 ビットの NTP タイムスタンプフォーマットで表される. 全く同期されていない場合は 0 となる.

発信タイムスタンプ (Originate Timestamp)

NTP メッセージが送られたピア側のローカル時刻で, 64 ビットの NTP タイムスタンプフォーマットで表される. ピアが Unreachable な場合は 0 となる.

受信タイムスタンプ (Receive Timestamp)

ピアから NTP メッセージが到着した時のローカル時刻で, をが送られたピア側のローカル時刻で, 64 ビットの NTP タイムスタンプフォーマットで表される. ピアが Unreachable な場合は 0 となる.

送信タイムスタンプ (Transmit Timestamp)

NTP メッセージが発信元から発信された時間で, 64 ビットの NTP タイムスタンプフォーマットで表される.

4.12.3 クライアントのメッセージ

クライアントからサーバに対して時刻の問い合わせがなされる NTP メッセージには, 次のような特徴がある.

- Leap Indicator は初期の NTP メッセージでは時刻が同期されていない意味の 11 が付くが時刻が同期し始めると, 00(通常の NTP メッセージ) に変わる.
- モードはクライアントを意味する 3.
- 初期のメッセージではストラタム値は不明を表す 0 だが, アソシエーションが確立されると自身のストラタム値を入れる.
- ポール間隔はデフォルトの 6.
- 初期の NTP メッセージでは, Reference, Originate, Receive の各タイムスタンプは NULL が入れられるが, サーバから応答があれば, 次からはタイムスタンプに時刻が入る.

4.12.4 サーバのメッセージ

サーバからクライアントに対して時刻情報が応答されるが, 次のような特徴がある.

- Leap Indicator は 00.
- モードはサーバの意味の 4.
- ストラタム値にはサーバのストラタム値が入る.
- ポール間隔はデフォルトの 6.
- Reference Clock ID には時刻源が, ストラタム 2 以降であれば上位のサーバの IP アドレスが入る.
- 各タイムスタンプにそれぞれの時間が入る.

4.13 Kiss-o'-Death

ストラタムフィールドが 0 の時は単に不明あるいは無効扱いだったが, NTPv4 からは Kiss-o'-Death(KoD) パケットで理由を示すことができるようになった. KoD パケットはクライアントに対して, アソシエーションやアクセス状態を表 4.6 のような ASCII メッセージを送り, サーバにパケットを送出しないよう伝える役目を担う.

コード	意味
ACST	アソシエーションがユニキャストサーバに属する
AUTH	サーバ認証が失敗
AUTO	Autokey シーケンスが失敗
BCST	アソシエーションがブロードキャストサーバに属する
CRYP	暗号認証または同定が失敗
DENY	リモートサーバによってアクセス拒否
DROP	Symmetric モードでピアが喪失
RSTR	ローカルポリシーのためにアクセス拒否
INIT	アソシエーションが初期段階で同期できなかった
MCST	アソシエーションが動的に見付かったサーバに属する
NKEY	鍵が見付からない, あるいは信用できない
RATE	混雑が原因でアクセス拒否
RMOT	`ntpdc` でリモートホストからアソシエーションの変更
STEP	システム時間のステップ変更が起きたが, アソシエーションが同期不全

表 4.6: NTP キッスコード

第5章　NTPソフトウェア

5.1　NTPソフトウェアの概要

NTPのプロトコルやソフトウェアの開発はデラウェア大学のDavid L. Mills博士が中心のNTPプロジェクトで行われており, Unixや Windows上への実装もここで行われている. このプロジェクトが開発したソースコードは唯一のNTPv4のフル実装ソフトウェアでインターネットを通じてソースコードあるいはバイナリ形式で配布され, NTPプロジェクトのホームページ[1]から HTTPもしくは FTPを使って誰でも[2]入手できる[3]. NTPのソフトウェアは表5.1のような複数のプログラム群からなるパッケージソフトウェアである.

プログラム	説明
ntpdate	ntpdの前に初期時刻を合わせるプログラム
ntpd	NTPデーモン (本尊とも言えるプログラム)
ntpq	標準NTP問い合わせプログラム
ntpdc	特殊NTP問い合わせプログラム (リモート制御用)
ntptrace	ストラタム1への繋がりをトレースする
tickadj	時刻関連のカーネル変数を調整する
ntptime	時刻を精密に扱うためのナノカーネルをデバッグする時に用いる. ntp_gettime() システムコールを使って時間関連のカーネル変数を表示させる.
ntp-keygen	NTPv4の Autokeyで使用する鍵を生成する
ntpdsim	NTPのシミュレータプログラム

表 5.1: ntpdパッケージののコンポーネント

[1] http://www.ntp.org/ or http://ntp.isc.org/
[2] 暗号ソフトウェアの輸出規制の関係で全てのコードを入手できない時期があった.
[3] http://ntp.isc.org/bin/view/Main/WebHome もしくは, ftp://ftp.udel.edu/pub/ntp/

5.2 NTP のバージョン番号

NTP プロジェクトは “4.2.2” からバージョン番号の表記方法を次のような定めた. 形式は,

```
A.B.C[Point][Special][ReleaseCandidate]
```

と表し, それぞれの意味は次の通りである.

- A はプロトコルバージョンを表す
- B はメジャーバージョンの番号を表す
- C はマイナーバージョン番号で, 偶数値は安定化版を, 奇数値は開発版を表す.
- Point は修正版 (スナップショット) を意味し, “p” の後に数値が続き, リリース版まで修正された数だけ数値が増加していく.
- Special は暫定的に使われるだけで一般には見ることは無い.
- ReleaseCandidate はリリース候補で “-RC” と表す.

2007 年 4 月 13 日時点でのバージョンは表 5.2 の通りである.

リリース	バージョン	日付
Stable	4.2.4	2006/12/28
Release Candidate	RC5	2007/02/24
Development	4.2.5p15	2007/03/03

表 5.2: リリースされている NTP のバージョン

5.3 NTP のインストール

ストラタム 2 や 3 といったリファレンス時計を直接参照しない NTP サーバとして利用する場合, オプションの設定無しに `configure` スクリプトを動かし, `make` でコンパイルし, `make install` でインストールすれば良い. 但し, Autokey(第 10 章で説明する) と呼ばれる NTPv4 の認証機能を利用する場合, OpenSSL ライブラリ (http://www.openssl.org/) をあらかじめインストールしておく必要がある. なお, NTP-4.2.4 から K&R C はサポートされなくなったので, 必ず ANSI C コンパイラでコンパイルする.

5.3.1 事前準備 (OpenSSL のインストール)

OpenSSL は, 多くのオペレーティングシステムにプリインストールされているが, プリインストールされていない場合は OpenSSL のホームページからソースコードをダウンロー

ドしてインストールする必要がある.

OpenSSL のインストールには Perl5(http://www.perl.org/) が必要となるので事前に確認しておく. Perl5 が使えることを確認したら, ソースコードは展開し, config スクリプトを実行し, make でコンパイル/テスト/インストールを行う.

```
# tar xfz openssl-0.9.8a.tar.gz
# cd openssl-0.9.8a
# ./config
# make
# make test
# make install
```

5.3.2 configure スクリプト

configure スクリプトはコンパイルを始める前に, スムースにコンパイルできるようオペレーティングシステムの開発環境に合わせ (ヘッダファイルやライブラリの調査), プログラムやファイルのインストール先, ntpd の諸機能を有効・無効を選択する事が主な役割である. configure スクリプトで指定できるオプションは “--help” を引数に実行する事で表示できる.

パス名の指定は表 5.3 ような引数が用意されている. 例えば, 基準となるパス名を--prefix で指定する (デフォルトは/usr/local になっている).

オプション	意味	デフォルトパス
--prefix=PREFIX	基準パス名	/usr/local
--exec-prefix	バイナリの基準パス名	EPREFIX=PREFIX
--bindir	ユーザ実行ファイル	EPREFIX/bin
--sbindir	システム管理者用実行ファイル	EPREFIX/sbin
--libexecdir	プログラム用実行ファイル	EPREFIX/libexec
--datadir	読み取りのみのアーキテクチャ依存のデータ	PREFIX/share
--sysconfdir	NTP の関連ファイル	PREFIX/etc
--sharedstatedir	アーキテクチャ依存のデータ	PREFIX/com
--localstatedir	マシン固有のデータ	PREFIX/var
--libdir	ライブラリ	EPREFIX/lib
--includedir	C ヘッダファイル	PREFIX/include
--infodir	info 形式のマニュアル	PREFIX/info
--mandir	オンラインマニュアル	PREFIX/man

表 5.3: パス名の指定

機能的なオプションやリファレンス時計ドライバの組み込みは--enable-*foo* と指定し, 無効にしたい場合は--disable-*foo* と指定する. 機能面の指定は表 5.4 のようなものがある. 一部の機能は ntp.conf ファイルの中で変更することもできる.

オプション	意味
--enable-accurate-adjtime	adjtime() システムコールが正確であるかどうか
--enable-clockctl	スーパーユーザ以外でもシステムクロックの制御を利用できるよう, /dev/clockctl を使う [enable]
--enable-linuxcaps	Linux のケーパビリティを利用する
--enable-debugging	デバッグコードを含む [no]
--enable-dst-minutes	夏時間での調整 (分) [60]
--enable-gst-surveying	GDT サーベイコードを含む [disable]
--enable-hourly-todr-sync	1 時間おきに TODR と同期を取る
--enable-kernel-fll-bug	Solaris カーネルの FLL バグを無視する
--enable-kmem	tick/tickadj は/dev/kmem を読む
--enable-ntpdate-step	ntpdate をステップ調整にする
--enable-simulator	NTPD シミュレータをインストールする
--enable-slew-always	時間の調整は常にスリュー調整にする
--enable-step-slew	時間調整をステップ調整してスリュー調整する
--enable-tick	'tick' の値
--enable-tickadj	tickadj の値
--enable-udp-wildcard	UDP ワイルドカード機能を使う
--enable-ipv6	IPv6 を利用する
--with-electricfence	ElectricFence malloc デバッガをコンパイルする
--with-sntp	SNTP をビルドする
--with-arlib	非同期リゾルバライブラリをコンパイルする
--with-kame	KAME の IPv6 ライブラリパス [/usr/local/v6]

表 5.4: configure スクリプトのオプション機能の指定

リファレンス時計のドライバは, それぞれ識別できるように番号が振られており, ntp-4.2.x で 44 の時計が登録されているが (ソースコードの README.refclocks を参照), 全ての時計が組み込まれるわけではない (configure --help で "s" が付いているドライバがデフォルトで組み込まれる). 全ての時計を組み込む場合は, configure 時に--enable-all-clocks を指定する. 組み込む時計ドライバをある特定のものに固定したければ, --disable-all-clocks とし, 利用するドライバを指定すればよい. 例えば, NMEA 対応の GPS レシーバを使う場合, 次のように実行する.

```
# ./configure --disable-all-clocks --enable-NMEA
```

オプションとして, Autokey と呼ばれる公開鍵の認証機能を有効にする場合は先ほど説明した OpenSSL を使用しなければならない. configure スクリプトが自動検索できないディレクトリにライブラリがインストールされている場合は表 5.5 のオプションでパス名等を指定する必要がある.

オプション	意味
with-openssl-libdir	OpenSSL のライブラリ
with-openssl-incdir	OpenSSL のヘッダファイル
with-crypto	認証機能の有無

表 5.5: 認証関係の指定

例えば, 次のように configure スクリプトを実行する.

```
# ./configure --prefix=/usr/local/ntp --with-openssl-libdir=/usr/local/lib
--with-openssl-incdir=/usr/local/include --with-crypto
```

5.3.3 コンパイルとインストール

configure スクリプトが正常に終われば, make でコンパイルを行う. コンパイルが正常に終了したら, 次にセルフ検査を行うため “make check” を実行する. これも正常に終われば, スーパーユーザになって “make install” でインストールを実行し完了となる.

コンパイルが失敗したり, 再度設定を変更してコンパイルする場合は, バイナリファイル, オブジェクトファイルや一時ファイルを削除する必要がある. “make clean” はバイナリ, オブジェクト, 一時ファイルを削除し, コンパイル設定作業 (configure スクリプトの実行) 後の状態である. ソースコードを展開した時点の状態に戻す場合は “make distclean” を実行する.

5.4 NTP の開発コード

5.4.1 入手とコンパイル方法

NTP プロジェクトが開発している NTP ソフトウェアは誰でも開発に参加できるオープンソースのスタイルを採っている.

NTP のソースコードは BitKeeper と呼ばれるソースコードのバージョン管理ツールで管理されており[4], BitKeeper を使うことで NTP の開発コードのリポジトリーを下記の方法で丸ごとダウンロードする事ができる.

```
# bk clone bk://www.ntp.org/home/bk/ntp-stable ntp-stable
# bk clone bk://www.ntp.org/home/bk/ntp-dev ntp-dev
```

一度, ダウンロードをすれば, 後は下記の方法でリポジトリを更新できる.

```
# cd REPO
# bk pull
```

開発中のソースコードを使ってみたい場合は, 公開されているバージョンをインストールする時と異なり, コンフィギュレーションファイルを生成する必要がある. 生成するには `autoconf` バージョン 2.53 以降, `automake` バージョン 1.5 以降及び最新の `lynx` が必要となる. これらのツールをインストールし, "`autoreconf -f -i`" を実行するとコンフィギュレーションファイルが作られる. その後は, 通常のインストールと同じように, `configure` スクリプトを実行し, `make` コマンドでコンパイルすれば良い.

```
# autoreconf -f -i
# ./configure
# make
```

なお, NTP プロジェクトは開発中のソースコードに関するバグの登録や追跡を行うために, Bugzilla を利用しているので, 開発中のソースコードを使っていて不審に思う点があれば, Bugzilla を参照すると良い. なお, 開発中のコードに関しては, NTP Software Development のページを参照すると良い.

5.4.2 NTP の情報を得る方法

NTP の情報を得るにはメーリングリストやネットニュースがある. 積極的に情報を入手し, 関係者と議論を深めたい人は, NTP プロジェクトが用意している表 5.6 のメーリングリストに参加する方法がある. メーリングリストは Mailman で運用されており, http://lists.ntp.isc.org/から申し込むことができる.

[4] `http://www.bitkeeper.com/`, Linux カーネルのコードも BitKeeper で管理されていることで有名. bk のヘルプは "`bk help` *command*'' で表示できる (例: `bk help pull`).

リスト名	内容
announce questions	NTP ソフトのリリースやセキュリティ関連の情報 NTP に関する疑問, その他が議論される場で活発に情報交換が行われている (comp.protocols.time.ntp にも配布されている)
bugs hackers bk-ntp-stable-send bk-ntp-dev-send repologs commitlogs	NTP ソフトのバグやパッチ報告 NTP ソフトのコードやプロトコルを開発している人達の技術討議の場で活発に議論されている ntp-stable リポジトリの変更 ntp-dev リポジトリの変更 BitKeeper のリポジトリログ NTP FAQ や Web サイトの変更ログ
ntp-legal ntpwg	NTP の法的/ポリシー/ライセンス問題の討議の場 IETF WG の討議の場

表 5.6: NTP.ORG のメーリングリスト

このメーリングリストのうち, ntp-questions はネットニュースの comp.protocols.time.ntp というニュースグループとリンクしている. 近くにニュースサーバがあれば, このグループの記事を読む事で色々な情報が手に入る.

その他, NTP プロジェクトでは NTP に関する Wiki(NTP TWiki)[5] が運用されており, ここを起点として, Bugzilla, NTP のドキュメント, FAQ などにアクセスできる.

5.5 最新の NTP について

5.5.1 NTP 4.2.2

2006 年 6 月 8 日に NTP Public Services Project からリリースされた NTP 4.2.2 について若干説明する. 新しい機能は, 次のとおりである.

- 完全動作の SNTPv4 クライアント (sntp コマンド)
- NTPv4 の現在の目標 (goal state) の実装
- Autokey の改良

[5] http://twiki.ntp.org/

- 改良された IPv6 のサポート
- マルチキャストモードがより多くのプラットフォームで利用可能[6]
- 新しいバージョン番号の考え

5.5.2 NTP 4.2.4

2006 年 12 月 28 日に NTP Public Services Project からリリースされた NTP 4.2.4 について若干説明する. 主な変更点は, 次のとおりである.

- DHCP のようにアドレスが動的に変更される場合への対応
- コマンド引数の処理に GNU AutoGen を使用
- リファレンスクロックのドライバの変更
- バグフィックス
- K&R C の非サポート (ANSI C のみ)

[6] http://ntp.isc.org/bin/view/Dev/MulticastStatus

第6章　ntpdateを使った時刻合わせ

NTP ソフトウェアを利用した時刻合わせには, ntpdate と ntpd プログラムがある. ntpdate はその時点の時刻を, 参照する NTP サーバの時刻に合わせるクライアントプログラムで, cron や at 等を用いて間欠的に使用される. 一方, ntpd は常駐プログラム (デーモン) で正確な時刻を維持し続ける場合に使用される.

一般的には, 初期に ntpdate で合わせ, その後は ntpd を用いる使い方が多い. Linux のようにシステムクロックとハードウェアクロックを相互に反映できる OS では, システム終了時には “hwclock --hctosys” を実行して, NTP の時刻をハードウェアクロックに反映させておくとよい.

6.1　ntpdate コマンドによる時刻合わせ

ntpdate コマンドは NTP サーバにポーリングして, 正しい時刻を判別し, ローカル時計をその時刻に設定するプログラムである. NTP サーバは複数指定する事ができ, その場合は NTP のクロックフィルタから複数のサンプルを取り, サーバ選択アルゴリズムを適用し, 最適なサーバを選択する. 単一のサーバを選択するより, 複数のサーバ (可能であれば 3 台程度) を選択した方が信頼性の向上につながる.

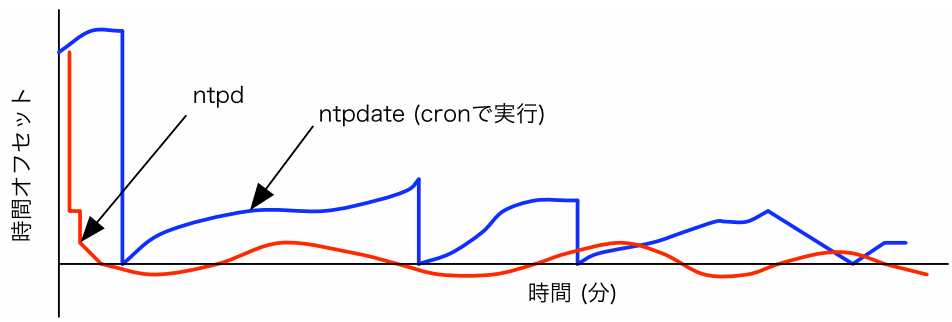

図 6.1: cron を使った ntpdate と ntpd の比較

ntpdate の簡単な使い方は, コマンドの後に 1 つ以上の NTP サーバ (IP アドレスもしくは FQDN) を引数にして実行する (6.6 節). 複数のサーバを指定する場合は空白を空けてサーバ記述し, 指定できる NTP サーバの最大数はコマンド行の制限に従うようだ (トークンが 20, 1024 文字).

下記は 3 台の NTP サーバを使って時刻を合わせた場合の例である.

```
# ntpdate ntp1.jst.mfeed.ad.jp ntp2.jst.mfeed.ad.jp ntp3.jst.mfeed.ad.jp
05 Nov 14:27:02 ntpdate[2439]: adjust time server 210.173.160.87 offset
-0.000084 sec
```

コマンドは 3 台のうちどのサーバを選択して時刻を合わせ, そのサーバとのオフセットが何秒であったかを示して終了する. *adjust* と表示されるのはデフォルトがスリュー調整を行ったためである. オフセット値が –(マイナス) であれば, こちらのコンピュータの時間が進んでいる事を示している.

6.2 二つの時刻調整モード

ntpdate の時刻合わせには二つのモードがある. 誤差 (オフセット) が 0.5 秒以上離れている場合, settimeofday() ルーチンを呼び出して, ステップ調整でサーバが持つ時刻に合わせる. 0.5 秒未満の場合は adjtime() ルーチンを呼び出し, スリュー調整を行う[1]. ntpdate でのスリュー調整は一度合わせるとコマンドは終了してしまうため, 参照した NTP サーバの時間が合う事は無い. オフセットが大きい場合は, サーバの時刻に近付けるために何度も ntpdate を実行しなければならず無駄である. 但し, スリュー調整は時間の進み遅れを微妙に調整し, 決して時刻が逆方向に進む事はないため, オフセットが小さい場合はスリュー調整の方が混乱は少ない.

先ほどの例をステップ調整にしてみる. ステップ調整を行うには, -b オプションを使用する.

```
# ntpdate -b ntp1.jst.mfeed.ad.jp ntp2.jst.mfeed.ad.jp ntp3.jst.mfeed.ad.jp
5 Nov 14:29:40 ntpdate[2443]: step time server 210.173.160.27 offset
0.000181 sec
```

adjust だった部分が *step* に変わっている事がわかる. この時, 参照した時刻サーバの IP アドレスも変化しているが, これは 2 台のサーバが微妙な時間しか差違が無いため, ネットワークの状況等でたまたま選択アルゴリズムはこのサーバを選択したのだと思われる.

[1] *ntpdate.h* の NTPDATE_THRESHOLD で定義されている.

もしも, 誤差が 0.5 秒[2]以上狂っていてもスリュー調整を行いたい場合は, -B オプションを付けて実行する事ができる. 但し, スリュー調整はその時点の時刻を合わせるものでは無く, 少しの時間差を合わせるにも数時間要するので[3], かなり非現実的といえる.

下記の例では 1 分近く時計が進んでいるにも関わらず, スリュー調整を試みた例である. ちょっとづつ近付いているのかも知れないが, 全く意味を為していない (結局, ステップ調整で時間を合わせた).

```
# ntpdate -B ntp1.jst.mfeed.ad.jp ntp2.jst.mfeed.ad.jp ntp3.jst.mfeed.ad.jp
10 Nov 13:46:54 ntpdate[2795]: adjust time server 210.173.160.87 offset
-53.416960 sec
# ntpdate -B ntp1.jst.mfeed.ad.jp ntp2.jst.mfeed.ad.jp ntp3.jst.mfeed.ad.jp
10 Nov 13:46:57 ntpdate[2796]: adjust time server 210.173.160.27 offset
-53.415802 sec
# ntpdate ntp1.jst.mfeed.ad.jp ntp2.jst.mfeed.ad.jp ntp3.jst.mfeed.ad.jp
10 Nov 13:46:10 ntpdate[2797]: step time server 210.173.160.87 offset
-53.412164 sec
# date
2003 年 11 月 10 日 月曜日 13:46:16 JST
```

従って, システムの起動時はステップ調整 (-b オプション) で合わせ, その後は ntpd もしくは ntpdate のスリュー調整を行う方が望ましい事がいえる.

6.3 ログの出力

一般的に, ntpdate はコンピュータの起動時や定期的に cron などを使って, バックグランドで実行する場合が多い. そのため, ログを標準出力に情報を表示させるのではなく, syslog に書き込むオプション-s が用意されている. より多くの情報を表示させたい場合は-v オプションを付けて実行する. cron デーモンで動かすような場合, ステップ調整でログは syslog に書き込む方法がよく取られるが, "ntpdate -bsv *ntpserver* ..." とすればよい.

6.4 時刻サンプルの調整

ntpdate は時刻合わせに際し, サーバから時刻サンプルを入手して, 評価を行っている. この数はデフォルトでは 4 つになっているが, 1 から 8 までの値を設定できる. 例えば, サー

[2]マニュアルでは ±128ms となっているが, ソースコード (ntpdate.c) を見ると 0.5 秒に見える. 一方, ntpd の閾値は ntpd/ntp_loopfilter.c を見ると 128ms になっている.

[3]標準的な Linux マシンでは, 0.5ms/秒と換算されるため, 1 秒差があれば 20 秒, 1 分差であれば 20 分かかる.

バ数を少なくする代わりに 8 つのサンプルを使って確実性を上げたければ, 次のようにコマンドを実行する.

```
# ntpdate -s -b -p 8 ntp1.jst.mfeed.ad.jp
```

6.5 ファイアウォールへの対応

NTP サーバからの応答はクライアントの UDP ポート 123 に投げられるが (NTP が使用するポート番号は送信も受信も UDP ポート 123 を使用する), LAN 内に設置している NTP サーバを保護する関係で UDP ポート 123 に対する通信を閉じている場合は, 時刻の取得ができなくなってしまうため, -u オプションを使って 1024 以上のランダムなポート番号を割り当てるようにできる.

NAT 環境化においても Well-Known なポートをデフォルトで閉じてしまうルータが多いので, -u オプションを用いて実行した方が良い.

6.6 サーバ指定の省略

ntpd デーモンを起動する前に ntpdate を動作させるケースでは, /etc/ntp.conf に記述されている NTP サーバを使った方が良いのは当然である. そこで, ntpdate にサーバを指定するのではなく, ntp.conf に記述されているサーバを参照するため, 次のような方法で ntpdate のサーバ指定を省略する事ができる. 一つは sed, awk などのスクリプトでサーバを抜き出す方法, もう一つは ntptime_config.c に含まれている loadservers() を使う方法である.

Apple 社の Mac OS X(Darwin) に標準搭載されている ntpdate コマンドには, サーバの選択に loadservers() が使われている.

```
# /usr/sbin/ntpdate -bsv
18 Nov 10:29:40 ntpdate[1873]: step time server 210.173.160.27 offset
0.000181 sec
```

標準ではコンパイルされないが, NTP のソースコードの ntpdate ディレクトリ中に ntptimeset コマンドがある. これは, Darwin が改良したように, ntp.conf に記述されたサーバを参照して同期する.

6.7 ntpdate のコマンドリファレンス

下表がコマンドオプションの一覧である.

オプション	説明
`-4`	ホスト名を IPv4 アドレスで解決する
`-6`	ホスト名を IPv6 アドレスで解決する
`-a` *key*	`/etc/ntp.keys` ファイルに書かれている認証鍵の番号を指定する
`-b`	強制的に `settimeofday` システムコールで時刻合わせる (デフォルトはスリュー調整)
`-B`	128 ミリ秒以上狂っていても強制的に `adjtimex` システムコールでスリュー調整を行う
`-d`	デバッグモードを有効にする (時間合わせは行われない)
`-e` *authdelay*	認証処理を秒単位で遅延させる
`-k` *keyfile*	デフォルト以外の認証鍵ファイルを指定する
`-o` *version*	問い合わせの NTP パケットのバージョンを指定する (デフォルトはバージョン 3)
`-p` *samples*	各サーバから入手するサンプルの数はを 1 から 8 までの値に設定する (デフォルトは 4)
`-q`	問い合わせのみを行い, 時刻合わせはしない
`-s`	`ntpdate` の出力を標準出力ではなく, `syslog` に記録する
`-t` *timeout*	応答を待つ時間を設定する, この時間は 0.2 秒の倍数に四捨五入される (デフォルトは LAN で使用するのに適した 1 秒)
`-u`	ファイアウォールで入りの ntp ポート (123) が塞がっている場合に使用する (ランダムに発信ポート番号が選ばれる)
`-v`	`ntpdate` コマンドが出力するメッセージがより詳細なものになる

第7章　ntpdの基本的な設定

インターネットを通じてコンピュータの時刻を正確に合わせたい場合や, 会社やキャンパス内でコンピュータに時刻を配信するために時刻サーバを動かしたい場合は, 常駐型のデーモンプログラム ntpd を使用する.

ntpd の主な役割は (1) リファレンスクロックやストラタム数の低い複数の NTP サーバから正確な時刻を得て, (2) もっとも正確な時刻を持つサーバを選択し, (3) 取得した時刻を他のコンピュータからの参照を許すあるいは配信する事にある.

ntpd は NTP サーバ機能を司るプログラムで, ntpdate と比較してより高機能で, アクセス制限やセキュリティ機構などを備えているため, 設定が複雑である. 本章では, ntpd の基本的な設定方法を説明する.

7.1　ntpdデーモンの動き

ntpd は最初に時刻同期を図る NTP サーバとのネットワーク的な遅延を測定し (この測定には 4, 5 分程度要する), それから時刻の同期を始める。

起動された ntpd デーモンは, まずドリフトファイル (ntp.drift; 7.10.1 節) と呼ばれるローカルクロックの周期的な狂い具合 (数値) が記述されているファイルを探しにいく. もし, このファイルが無ければ, マシンが持つローカルクロック (クロックオシレータ) の周期的な狂い具合を測定し, 約 15 分かけて周期誤差の内部値を初期化し, ドリフトファイルにその値を書き込んでから, 通常のモードに入る. 通常のモードに入っても, 狂い具合は常時監視され続け, 1 時間に 1 度この値を更新する.

ntpd の時刻合わせはサーバとの時刻差が 128 ミリ秒以上狂っていなければ, スリューモードでゆっくり時刻を合わせていく. もし, 何らかの理由で 128ms 以上狂ってしまった場合, ステップ調整で時刻を合わせる. 128ms 以上狂った場合でもスリュー調整を行いたい人のために -x オプションが用意されているが, スリュー調整の時刻合わせは 500PPM[1] に抑えられているため, 時間の狂いが大きいと補正にはかなりの時間を要すると覚悟しておくべきである (1 秒合わせるのに約 2000 秒要する).

[1] 1 日に 43 秒ずれるレベル (1 日に 1 秒ずれるのは 12PPM と覚えておくと良い)

7.2 ntpd コマンドと関連ファイル

7.2.1 ntpd のコマンドオプション

ntpd はデーモンとして動作するプログラムで, 細かな設定はコンフィグファイル (/etc/ntp.conf) で行われるが, ntpd 自身にも様々な引数を持っている. 下記がコマンド行の一覧である.

```
ntpd [ -46aAbdDgLmnNqx ] [ -c conffile ] [ -f driftfile ] [ -i jaildir ] [ -k keyfile ]
     [ -l logfile ] [ -p pidfile ] [ -P priority ] [ -r broadcastdelay ] [ -s statsdir ]
     [ -t key ] [ -u user[:group] ] [ -U interface_update_interval ] [ -v variable ]
     [ -V variable ]
```

オプション	説明
-4	ホスト名を IPv4 アドレスで解決する
-6	ホスト名を IPv6 アドレスで解決する
-a	クライアント及び対象パッシブモードで暗号認証を要求する (デフォルトで有効)
-A	クライアント及び対象パッシブモードで暗号認証を要求しない
-b	ブロードキャストサーバに時刻同期するためのクライアント機能を有効にする
-c *conffile*	コンフィグファイルのパス名を指定する
-d	デバッグモードを有効にする
-D *level*	デバッグレベルを指定する. 大きな数字ほど多くの情報が表示される
-f *driftfile*	ドリフトファイル (ntp.drift) のパス名を指定する
-g	普通は時間差 (オフセット) がパニック閾値 (1000 秒) に達すると, システムログにメッセージを残して ntpd は終了する. このオプションを付ける事で制限を無くす働きをする. -q と -x と一緒に用いる
-i *jaildir*	Chroot のディレクトリを指定する.
-k *keyfile*	鍵ファイル (ntp.keys) のパス名を指定する
-l *logfile*	ログファイルのパス名を指定する
-L	仮想 IP アドレスでは受信しない

オプション	説明
-m	IPv4 マルチキャストサーバに時刻同期をするためのクライアント機能を有効にする
-n	フォークしない
-N	高プライオリティで実行される (OS に依存)
-p *pidfile*	ntpd のプロセス ID が記録されたファイル (ntpd.pid) のパス名を指定する
-P *priority*	ntpd プロセスのプライオリティを指定する
-q	時間が設定されたら ntpd を終了する. ちょうど ntpdate のような動きになる. 注意点として, このオプションを用いるとカーネル時間 discipline は無効になる
-r *broadcastdelay*	マルチキャスト/ブロードキャストサーバから配送遅延時間を指定する
-s *statsdir*	統計ファイルの作成に使用するディレクトリを指定する
-t *key*	トラスティッド鍵リストに指定の鍵番号を追加する
-u *user[:group]*	ntpd デーモンが動作するユーザ権限を指定する. root 以外のユーザでシステムクロックを変更できる権限を持つ OS で有効なオプションで現在は NetBSD と Linux のみに対応する.
-U *interface_update_interval*	インタフェースの状態をスキャンする間隔で, 0 は無効でデフォルトは 5 分.
-v *variable*	指定のシステム変数を追加する
-V *variable*	デフォルトでリストされた指定のシステム変数を追加する
-x	普通は時間差の閾値が 128ms より小さければスリューモードが働き, この値以上に差があればステップモードが働く. 閾値を 600 秒まで拡大して働かせる

7.2.2 関連ファイルの種類

ntpd の大部分の設定は ntp.conf ファイルに記述される. その他の関連ファイルには, 表 7.2 のファイルがある. これらのファイルのパス名は起動時に指定したり, ntpd.conf の中で指定する. パス名が指定されなければ, デフォルトのパス名でファイルを読み込んだり生成する.

ファイル名	説明	デフォルトパス	指定	コマンド
ntpd.conf	設定ファイル (本尊)	/etc	-c	—
ntp.keys	鍵ファイル	/etc	-k	keysdir
ntp.drift	ドリフトファイル	/etc	-t	driftfile
ntpd.pid	プロセス ID が入るファイル	/var/run	-p	pidfile
ログファイル	ログファイル	/var/log	-l	logfile
分割ファイル	別ファイルにする	—	—	includefile
統計ファイル	統計ファイル	/var/NTP/	-s	statsdir

表 7.2: ntpd の関連ファイル指定オプション

例えば, 設定関連ファイルを/etc/ntp の下に置きたければ, configure のオプションに --sysconfdir=/etc/ntp と指定するか, 次のように起動する.

```
ntpd -c /etc/ntp/ntp.conf -k /etc/ntp/ntp.keys -t /etc/ntp/ntp.drift
```

7.3 ntpd.conf の概要

ntpd がどのようなモードで, どのサーバを参照するのか等, NTP の設定を行うファイルが ntp.conf である[2].

ここでは機能別に次のように分けて, 各種設定コマンドを説明する.

- **サーバオプション**は, 参照する NTP サーバに関する情報を指定する.
- **認証オプション**は, 認証機能や NTPv4 から採用されている公開鍵ベースの認証機能に関する設定を行う[3].
- **監視オプション**
- **アクセス制御オプション**

[2]通常は, /etc/ntp.conf に置かれている.
[3]認証及びオートキーに関わる部分は後で説明するので, 本章では割愛する.

- **リファレンスクロックオプション**は, 利用するリファレンスクロックの設定を行う.
- その他のオプション

なお, ntp.conf の中で, “#” はコメントを意味するサインで, #から行末までがコメントになる.

7.4 ntpd.conf のもっとも単純な例

ntpd のもっとも単純な設定は外部の NTP サーバを数台参照する設定である. インターネットの内で公開されている NTP サーバのリストは次の Web ページで公開されているので, このリストから 3 台程度のサーバを選択して, ntp.conf を設定する.

- ストラタム 1：http://www.eecis.udel.edu/~mills/ntp/clock1a.html
- ストラタム 2：http://www.eecis.udel.edu/~mills/ntp/clock2a.html

このリストの中に日本国内の NTP サーバはあまり掲載されていないが, 実験的に公開されている公開 NTP サーバは 10〜20 ホスト存在している (表 7.3). 利用ポリシーが明示されていないサーバについては無断で利用せず, 管理者に参照して良いかを尋ねた方が無用なトラブルを招かずに済む (くれぐれもネチケットを忘れずに).

後で詳しく説明するが, サーバを指定するには, server コマンドを使用する. 例えば, 情報通信研究機構が提供しているストラタム 1 の NTP サーバ (ntp.nict.jp) を利用する場合は, 次のように指定する (サーバが複数の IP アドレスを持つ場合はその数だけ記述してもよい).

```
server ntp.nict.jp
server ntp.nict.jp
server ntp.nict.jp
```

もっとも単純な設定例としてはこれだけでも良い. 上記例ではサーバを FQDN で指定しているが, IP アドレスで記述しても構わない (IPv4/IPv6 通信や DNS の問題を避けたければ, むしろ IP アドレスを指定した方が良いかも知れない[5]).

サーバの指定数は ntpdate の時よりも多くのサーバを指定できるが, 結局サーバは NTP の選択アルゴリズムで一つに絞られるので、あまり多く指定しても意味は無い. それでも, なるべく近くにある安定した NTP サーバを指定し, 冗長性を考慮して最低 3 台程度は指定しておきたい.

[4]利用方法や注意点等は, “http://www2.nict.go.jp/w/w114/stsi/PubNtp/qa.html” を参照.
[5]公開 NTP サーバの IP アドレスが変更されるのは稀な事である.

ホスト名	管理組織	ストラタム
ntp.nict.jp	情報通信研究機構[4]	1
gpsntp.miz.nao.ac.jp	国立天文台	1
clock.nc.fukuoka-u.ac.jp	福岡大学鶴岡研	1
drake.nc.fukuoka-u.ac.jp	〃	1
eagle.center.osakafu-u.ac.jp	大阪府立大学	1
ntp1.tohoku.ac.jp	東北大学	1
ntp2.tohoku.ac.jp	〃	1
ntp.tut.ac.jp	豊橋技術科学大学	1
ntp.nc.u-tokyo.ac.jp	東京大学	1
ntp.hiroshima-u.ac.jp	広島大学	1
ntp1.jst.mfeed.ad.jp	マルチフィード (株)	2
ntp2.jst.mfeed.ad.jp	〃	2
ntp3.jst.mfeed.ad.jp	〃	2
sutns.sut.ac.jp	東京理科大学	2
ntp.wakayama-u.ac.jp	和歌山大学	2

表 7.3: 国内のパブリックな NTP サーバリスト

7.5 サーバオプション

サーバオプションでは, どの NTP サーバからどのように時刻を参照するかを指定する部分で, (1) サーバもしくはピアの IPv4/IPv6 アドレス, (2) サブネットのブロードキャストアドレス, (3) マルチキャストアドレス, (4) リファレンスクロックのアドレス (127.127.x.x) を指定する.

参照するサーバは FQDN(Full Qualified Domain Name) で指定する事もできるが, OS によってはソケットがアドレスを解決する際に IPv6 アドレスが優先されるため, IPv4 を優先したい場合は-4 を付けて実行する必要がある.

NiCT の NTP サーバは IPv4/IPv6 をサポートしており, IPv6 をサポートするオペレーティングシステムを利用している場合, IPv6 を優先する場合が多く, IPv6 で通信できない環境の場合, -4 を付けた方が良い.

```
server -4 ntp.nict.jp
```

7.5.1 サーバタイプの指定

参照する時刻源の指定や時刻配信方法にブロードキャストを使うかどうか等は, 表 7.4 のコマンドで指定する.

一番迷うのは, server と peer の使い分けだろう. server は指定した NTP サーバを時刻リファレンスにするということを意味し, そのサーバから時刻をもらえば (選択すれば), 自分のストラタム値は一つ上がる事になる. そしてサーバ側から自分の時刻を参照される事はない. peer は指定したサーバとは対等の立場である事を意味し, どちらがクライアントでどちらがサーバという区別は無い. クライアントとサーバの関係では, サーバ側からクライアントの時刻を参照する事はないが, peer の場合は自身の時刻が相手から参照される可能性があるという特徴がある. サーバは相互に peer を指定する必要があり, peer の場合は他のサーバを参照した状態にならない限り, 自分のストラタム値は上がらない.

7.5.2 サーバへのポーリングの指定

ntpd は時刻同期が完了して暫く経過すると, NTP サーバへの時刻同期の問い合わせポーリング間隔をだんだん広げていく (時間とともにソフトウェアクロックが安定してくるため). この間隔の指定は起動オプションや ntp.conf の中で指定でき, 間隔は 2 の冪乗 (秒) で表される. minpoll が最小間隔を maxpoll が最大間隔を表し, デフォルトでは minpoll は 6(64 秒), maxpoll は 10(1024 秒) である.

コマンド	サーバタイプ
server	リモートの NTP サーバもしくはリファレンスクロックを指定する. このモードでは, NTP サーバに対してクライアント的アソシエーションとなり, 指定したリモート NTP サーバからこのクロックを参照されることは決して無い
peer	リモート NTP サーバを指定する. このモードでは指定したサーバとは symmetric-active モードで動作する
broadcast	ブロードキャストまたはマルチキャストアドレスを指定し, NTP の時刻情報をブロードキャストまたはマルチキャストで定期的に送信する事を指定する. ブロードキャストの場合, 指定したサブネットアドレスを持つインタフェースで送信するが, マルチキャストの場合はマルチキャストが有効になったインタフェースで送信する. マルチキャストアドレスを指定しない場合は IANA で予約済の 224.0.1.1(IPv4) もしくは [Prefix]:101 が使われる
manycastclient	メニーキャスト機能を使う際のマルチキャストアドレスを指定する

表 7.4: サーバタイプの指定

指定できるポーリング値の最小と最大はソフトウェアの中で決められており, xntp3-5.93e では最小 4(16 秒) で最大が 14(4.5 時間), ntp-4.2.x では最小 4(16 秒) で最大が 17(1.5 日) になっている.

どのくらいの間隔でポーリングを行えばよいかは, なかなか難しく一概に言えない. ポーリング間隔を短くすると, ローカルクロックが信頼できない場合に有利だが, ジッタやランダムエラーの影響を受けやすくなる. 長い間隔にすると, ローカルクロックの精度が要求される. 始めはデフォルトの値で運用し, 問題あれば修正していく形にならざるを得ないが, 回線に十分な帯域と安定性があり, サーバが信頼できる時刻源になりうるなら, ポーリングを 1 分に固定しても良いだろう.

minpoll/maxpoll の指定は, server/peer コマンドのサーバアドレスの後に指定すれば良い. 例えば, ポーリング間隔を 1 分固定にしたければ,

```
server ntp.nict.jp minpoll 6 maxpoll 6
```

と指定する.

7.5.3 サーバの優先と非使用

複数のサーバを指定する際に, 安定性が優れているサーバが予めわかっているならば, prefer オプションを付けて, そのサーバを優先的に参照するようにできる[6]. 例えば, NiCT の NTP サーバを優先したければ, 先ほどの例は次のようになる.

```
server ntp1.jst.mfeed.ad.jp
server ntp.nict.jp prefer
server clock.nc.fukuoka-u.ac.jp
```

prefer は複数付けても良いが, 付いているものは同等の重み付けになる. 逆にそのサーバを参照したくない場合, noselect を付ければ良い. サーバは選択アルゴリズムから無視される. 例えば, 福岡大学の NTP サーバを使わなければ, 次のようになる.

```
server ntp1.jst.mfeed.ad.jp
server ntp.nict.jp
server clock.nc.fukuoka-u.ac.jp noselect
```

7.5.4 ブロードキャスト/マルチキャストを使った時刻配信

broadcast コマンドは指定されたブロードキャスト/マルチキャストアドレスで, 自身が持つ時刻をネットワークに乗せて配信する機能である. 通常, ブロードキャストの場合は NTP サーバのネットワークインタフェースのブロードキャストアドレスを, マルチキャストの場合は IANA で予約済のマルチキャストアドレス 224.0.1.1(NTP.MCAST.NET) もしくは [Prefix]:101 を使用する.

マルチキャストの場合, サイトローカルマルチキャストアドレスや伝搬範囲を指定する TTL 値を明示的に指定した方が良い. ローカルなマルチキャストアドレスを使えば, 自ネットワーク以外に伝搬しないようルータでのパケットフィルタが簡単になるし, TTL 値が指定されていればルータで TTL boundary を設定することで, 外部にパケットが流れないようにできる. TTL 値が何も指定されなければ, 127 に設定されるが, broadcast コマンドでアドレスを指定した後に ttl オプションで値を指定できる. 例えば, TTL 値を 31, 239.100.100.123 で時刻を配信するには次のように指定する.

```
broadcast 239.100.100.123 ttl 31
```

[6] もちろん, クロック選択アルゴリズムで正常なホストと認定された場合に優先される.

もし, 複数のブロードキャスト/マルチキャストメッセージを送出したければ, 複数の broadcast コマンドを記述する. 例えば, サブネット LAN 上ではブロードキャストを, キャンパス LAN 全体ではマルチキャストを使用する場合は次のように指定する (7.6 節, 81 ページ参照).

```
broadcast 10.1.2.255 key 5      # broadcast on subnet LAN shared key
broadcast 224.0.0.1 key 3       # multicast on subnet LAN shared key
broadcast 239.1.1.2             # multicast on campus LAN
```

なお, ブロードキャストでもマルチキャストでも, ユニキャストの時と同様にセキュリティを高めるために対象鍵または公開鍵を使った認証機能を使うことができる.

7.5.5 ダイアルアップネットワーク (バーストモード)

ntpd はインターネットに常時接続されていることを前提としているため, NTP サーバの妥当性チェック, カーネルクロックの調整などに時間を要する. ダイアルアップネットワーク環境ではすぐに時刻同期を開始する必要があるため, バーストモードと呼ばれるアソシエーションモードが用意されている.

バーストモードには二つのモード (burst, iburst) があり, どちらのモードも一度のポーリングで一つのパケットが送られるところを, 2 秒間隔で断続的に 8 つのパケットを送出して, 素早く時刻を合わせる. burst モードはポーリングの度にバーストパケットを送って時刻合わせを行い, iburst モードはコネクションが確立された初期の時点でバーストパケットを送出する.

burst モードは通常の 8 倍のパケットが送出されるため、ポーリング間隔の頻度が少ない事を念頭におく必要があり, minpoll を 10(1024 秒) に設定しておくと良い. 一方, iburst モードは, ダイアルアップスクリプトに ntpdate コマンドを書き加えて, 時刻合わせを行うような仕組みを ntpd だけで働くようにしたと考えて良いだろう. そのため, コネクションの確立に要する時間を考慮して, バーストパケットの最初のパケットと二番目のパケットの間隔を calldelay コマンドで伸ばす事ができる. 例えば, 30 秒に伸ばしたければ, 下記のように設定する.

```
server 1.1.1.1 iburst
server 2.2.2.2 iburst
calldelay 30
```

7.5.6 NTP パケットのバージョン指定

version オプションでサーバから出ていく NTP パケットのバージョンフィールドの値を指定できる. ntp-4.2.x ではデフォルトでは 4 になるが, バージョン 3 で出したければ, 下記のように指定する.

```
server 10.1.20.50 version 3
```

NTP パケットのバージョンフィールドの値で処理が拒否されることまずは無いので, 特別意識する必要は無いと思われる.

7.6 対象鍵方式による認証

7.6.1 NTP の認証機能の概要

NTPv3 から NTP クライアントの不正なアクセスを防止するために対象鍵方式を使った認証機能を有している. 当初, RFC-1305 の仕様では, NTP パケットは DES-CBC で暗号認証されていたが, 今は秘密鍵を使った一方向ハッシュの方式に置き換わっており, そのアルゴリズムには RSA 社開発の Message Digest 5 (MD5) が用いられている.

対象鍵方式は, サーバとクライアントが同一の秘密鍵を共有しており, NTP パケットはこの鍵を使ってハッシュ化されたデータが付加されて送られ, 正しい鍵を持っているマシンだけがこのパケットが正しいかどうかを認証できる (図 7.1).

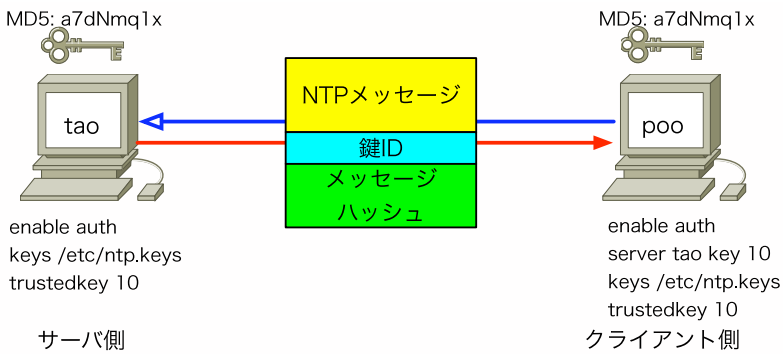

図 7.1: 対象鍵を使った NTP の認証

NTPv4 では NTPv3 の対象鍵方式に加え, よりセキュリティを強化するために, 公開鍵方式の **Autokey** と呼ばれる方式を採り入れた. 対象鍵方式の最大の問題点は鍵の伝達 (配布)

方法にあるが, 公開鍵方式は各サーバが鍵を生成するため, 秘密鍵の安全性は保たれる. また, Autokey の鍵管理は X.509 の認証方式をベースとしており, インターネットの PKI サービスとの親和性も良いと考えられている.

Autokey については第 10 章で説明し, 本章では対象鍵方式の認証方法を説明する.

7.6.2 鍵の設定方法

対象鍵方式では 32 ビットの鍵 ID のうち 65,534 個の鍵が識別でき, 認証機能を使いたいサーバとクライアントの間で同一の鍵を使用しなければならない. 鍵情報は `ntp.keys` ファイルで指定される. この鍵ファイルは安全のために root もしくは ntpd のオーナー以外は読めないようにファイルへのアクセス許可を変更しておく.

鍵ファイルの形式は 1 行に一つの鍵を記述し, 第一カラムに鍵 ID(1 から 65534) を, 第二カラムに鍵タイプを, 第三カラムに鍵を指定する. 現在有効な鍵タイプは MD5 鍵を表す M のみで, 1〜31 文字の ASCII(0x21〜0x7f の印字可能な文字と'#').

例えば, `ntp.keys` を次のように記述する.

```
1   M   68767ce625aef123
2   M   df2ab658da62237a
498 M   NTPv4.98
```

7.6.3 鍵の利用方法

認証機能を有効にするコマンドとして, `enable` コマンドの `auth` オプションがあるが, デフォルトで認証機能は有効になっているので明示的に指定する必要は無い.

認証機能が有効になっていると, `ntpd` は起動と同時に, `ntp.conf` の中の `keys` コマンドや`-k` オプションで指定されている鍵ファイル (`ntp.keys`) を読み込んでメモリ内にキャッシュする. キャッシュされた鍵が信頼された鍵となるが, ファイルの中の特定の鍵のみを使う場合は, `trustedkey` コマンドを使って鍵 ID を指定することもできる. 対称鍵認証を使うサーバ側の指定は認証に利用する鍵を指定するだけでよい.

下記の例では, 鍵 ID が 1, 2 と 15 の鍵でハッシュ化されたメッセージを信頼する事になる. この場合, 制御メッセージを含めて信頼してしまうため, `ntpq` や `ntpdc` にメッセージ認証とは別の鍵を使う場合は, `ntpdc` のパスワードとして使われる鍵を `requestkey` コマンド, `ntpq` のパスワードとして使われる鍵を `controlkey` コマンドで別に指定できる (但し, これらの鍵は `trustedkey` の中で指定されなければならない).

```
keys /etc/ntp.keys
trustedkey 1 2 15
requestkey 15                   # ntpdc
controlkey 15                   # ntpq
```

一方, クライアント側では上記に加えてサーバコマンド (server, peer, broadcast, manycastclient) に key というオプションを付けて鍵 ID(1〜65534) を指定する事で, サーバまたはピアとの間でメッセージ認証が行われる.

```
server 1.2.3.4 key 1
peer 10.1.1.103 key 2
```

7.6.4 ntpdate, ntpq, ntpdc コマンドの認証機能

ntpdate コマンドの場合 (6.7 節, 69 ページ参照), -k オプションで鍵ファイルを, -a オプションで鍵 ID を指定する.

```
# ntpdate -k /etc/ntp.keys -a 1 1.2.3.4
```

ntpd は ntpdc/xntpdc コマンドを使うことで, リモートから動的に ntpd の設定を変更できるが, ntpq や ntpdc コマンドはコマンド行の中で鍵を指定できないので, 内部コマンドでパスワードを入力するようになっている. このように, requestkey コマンドを設定しておくことで, リモート設定用のパスワードの役目を果たす.

```
ntpdc> keytype md5
ntpdc> unconfig 127.127.8.0
Keyid: 15
MD5 Password:
```

7.6.5 MD5 の問題

NTP が利用している認証は MD5 アルゴリズムを利用しているが, CRYPTO 2004 カンファレンスにて, Xiaoyun Wang, Dengguo Feng, Xuejia Lai, Hongbo Yu らによって MD5

のコリジョンが発見され, 衝突攻撃の影響を受ける可能性があると発表され, MD5 の信頼性に疑義が生じている [18]. SHA-1 についても同様の問題が起こる可能性が示唆されており, SHA-2 への移行が必要となっている.

広く使われている NTP の暗号アルゴリズムをいきなり変更するのは簡単ではないため, 暗号以外の IP アドレス, ポート番号, タイムスタンプ, 遅延や分散値からも不正な NTP メッセージであることを見破る仕組が必要となっている.

7.7 ブロードキャスト/マルチキャストのクライアント機能

ブロードキャスト/マルチキャスト機能を有効にするには, enable コマンドで bclient オプションを指定する必要がある. そして, broadcastclient コマンドでサブネットブロードキャストの受信を指定し, multicastclient コマンドでマルチキャストの受信を指定する. multicastclient コマンドではマルチキャストグループアドレスの指定が可能で, アドレスを指定しないと, NTP のデフォルトグループアドレス 224.0.1.1 が使われる. また, 受信できるグループアドレスは次の例のように複数指定することもできる.

```
enable bclient
broadcastclient
multicastclient 224.0.0.1 239.1.1.123
```

ブロードキャストやマルチキャストモードでは, サーバ間のネットワーク遅延がどの程度あるかを実測できないので, ネットワーク遅延を決定するため特別なキャリブレーションとして, ネットワークメディアを想定したネットワーク遅延を broadcastdelay コマンドで指定する. イーサネットの場合, 0.003 から 0.007 秒が当てられるが, コマンドを指定しない場合は 0.004 秒の遅延が設定される.

```
broadcastdeley 0.00153
```

ブロードキャスト, マルチキャストでもサーバ・クライアントやピア関係と同様に, セキュリティを高めるため対象鍵または公開鍵を使うことができる. 例えば, 対象鍵を使う場合はクライアントを次のように指定する. ここではサーバで指定される鍵 ID と鍵を ntp.keys ファイルの中に指定しておく.

```
enable auth bclient
multicastclient 224.0.0.1
keys /etc/ntp.keys
```

7.8 アクセス制御

7.8.1 アクセス制御機能の概要

ntpd にはアドレス/マスクの形式によるアクセス制御機能を有している．アドレス/マスクの表現方法はマスクの bitwise AND 形式で, **パケットのソースアドレスでアクセス制御を行う**．例えば, アドレスを “192.168.1.0 mask 255.255.255.0” とすれば, 192.168.1.0〜192.168.1.255 の範囲のアドレスを持つソースアドレスという事になる．リストは複数記述できるが設定ファイルの上から順番に評価し, 一致したリストがあればそれが適用される．

リストの記述は restrict コマンドにより行い, サーバへのアクセスを禁止したり, 問い合わせを禁止したり色々な制限が可能である．

7.8.2 restict コマンド

restrict コマンドは指定した IP アドレスに対し, パケットの通信制限を設定するコマンドで, *flag* サブコマンドを使って細かく制限方法を指定できる．*flag* サブコマンドを使わなければ, そのアドレスからのあらゆる NTP パケットを許可する．*flag* サブコマンドは複数記述でき, 複数のフラグを適用したければ, 空白をあけてそのままま列挙する．

マスクを指定しない場合は 255.255.255.255, すなわち一つのホストアドレスを意味する．また, 全てのソースアドレスを示すアドレス 0.0.0.0, マスク 0.0.0.0 を default と表し (default は IPv4 アドレスのみ使われる), リストの最初に置かれる．なお, マルチキャストアドレスがソースアドレスの場合はその行は無効になる．

flag サブコマンドは次の通りである．

フラグ	説明
ignore	ntpq や ntpdc からのクエリを含む全てのパケットを拒否する
kod	kill-o’-daeth(KoD) パケットが送られた時にアクセス拒否を行う．なお, KoD パケットは 1 秒に 1 パケットを制限を上限とし, 1 秒以内に KoD パケットが届いた場合はパケットをドロップする．
limited	discard コマンドで指定された流量を越えるパケットが来た場合にサービス拒否を行う．

フラグ	説明
lowpriotrap	アドレスと一致したホストは低プライオリティにセットされる.
nomodify	時刻問い合わせや情報へのクエリは許されるが, ntpq や ntpdc からのサーバの状態を変更するためのクエリ (再設定) を拒否する.
noquery	ntpq や ntpdc からのクエリを拒否する. 但し, サービスには影響されない.
nopeer	時刻問い合わせには応答するが, 新しいアソシエーションを確立するパケットを拒否し, 相互に時刻同期を行わない.
noserve	ntpq や ntpdc からのクエリ以外のパケットを拒否する.
notrap	このフラグが付いたホストはモード 6 制御メッセージのトラップサービスの提供を拒否する. トラップサービスは ntpq 制御メッセージプロトコルのサブシステムである.
notrust[7]	暗号によって認証されたパケットでなければサービスを拒否する.
ntpport	ソースポートが標準の NTP ポート (123) であるパケットのみに制限を加えたい場合に使用する.
version	現在動作中の NTP バージョンと一致しないパケットは拒否する.

IPv6 の場合は “-6” を付けて区別する. それ以外は IPv4 の時と同じように記述する.

```
restrict -6 default ignore
restrict -6 ::1
restrict -6 v:u:t:s::1
restrcit -6 2001:838:0:1:: mask ffff:ffff:ffff:ffff:: nomodify notrap nopeer
```

7.8.3 discard コマンド

discard コマンドはクライアントからサーバへのパケットの流量制限を行うコマンドである. 最小平均パケット間隔 (average *avg*), 最小パケット間隔 (minimum *min*) を指定して, “Kiss-o’-death” パケットからの防御を行う. デフォルトは 5 と 2 である.

discard コマンドには monitor サブコマンドが用意されている. ここではレート制御ウィンドウをオーバフローするパケットの破棄率を指定できる[8].

[7] 公開鍵認証が実装されていないバージョンでは, 時刻問い合わせには応答するが, 指定したホストを時刻合わせには使用しない (時刻同期のソースにしない), という意味だった.

[8] 4.2.0 ではバグで monitor サブコマンドが使えない. このバグは 4.2.0 以降のバージョンでフィックスされている.

```
discard average 5 minimum 2 monitor 0.5
```

7.8.4 アクセス制御を設定例

いくつかの例をあげて説明する. 下記の例はもっとも簡単な例である. デフォルトは NTP サービスを拒否し, ホスト自身と特定のネットワークからの NTP クライアントからのみ応答する設定である.

```
restrict default ignore                    # default deny
restrict -6 default ignore                 # IPv6
restrict 127.0.0.1                         # allow local host
restrict -6 ::1                            # allow local host (IPv6)
restrict 10.0.0.0 mask 255.0.0.0
restrict 172.17.0.0 mask 255.255.0.0
```

次は, 管理ネットワークからのクエリは受け付け, 他は時刻サービスだけを提供する場合の設定例である. これは, 10.1.2.0/24 から全ての NTP メッセージを受付け, 10.0.0.0/24 や 172.17.0.0/16 からは時刻問い合わせのみを受け付ける設定となる.

```
restrict default ignore                    # default deny
restrict 127.0.0.1                         # allow local host
restrict 10.1.2.0 mask 255.255.255.0
restrict 10.0.0.0 mask 255.255.255.0 kod nomodify notrap nopeer noquery
restrict 172.17.0.0 mask 255.255.0.0 kod nomodify notrap nopeer noquery
```

7.9 状態監視に関する設定

7.9.1 ntpd の状態監視のコマンド

ntpd のクロック, ループフィルタ等の状態監視は, statistics コマンドで監視する項目を指定し, filegen コマンドで統計情報を記録するファイル名を指定する. そして, 記録された統計情報ファイルを使って, フィルタプログラムを利用して統計情報の集計やグラフ化を行う. ソースコード中に用意されているスクリプト (./scripts/stats) は AWK や S 言語で書かれているため, 今のスタイルには馴染にくい. これらを参考に Perl, Python, Ruby

などのスクリプト言語使って, 新たにフィルタプログラムを自作し, MRTG や RRDtools を使ってグラフ化処理した方がよいだろう.

次節から各状態監視コマンドを説明する.

7.9.2 クロックの状態監視コマンド

クロックドライバの統計情報を収集するには, statistics コマンドのオプションに clockstats を指定する. クロックドライバからの情報が更新される度に, clockstats: という名前でファイルが生成され, 次のような情報が書き込まれる.

```
49213 525.624 127.127.4.1 93 226 00:08:29.606 D
```

49213	ユリウス歴の日付
525.624	ユリウス歴の時間 (UTC)
127.127.4.1	リファレンスクロックのアドレス (Spectracom 8170/Netclock-2)
93 226 00:08:29.606 D	タイムコード (クロックドライバのマニュアルを参照)

7.9.3 暗号状態監視コマンド

公開鍵認証を使った場合のプロトコル情報を記録するには, statistics コマンドのオプションに cryptostats を指定する. cryptostats: という名前でファイルが生成され, 次のような情報が書き込まれる.

```
49213 525.624 127.127.4.1 message
```

49213	ユリウス歴の日付
525.624	ユリウス歴の時間 (UTC)
127.127.4.1	リファレンスクロックのアドレス
message	メッセージタイプやプロトコル情報

7.9.4 ループフィルタ状態監視コマンド

ループフィルタの統計情報を記録するには, statistics コマンドのオプションに loopstats を指定する. ローカルクロックが更新される度に, loopstats: という名前でファイルが生成され, 次のような情報が書き込まれる.

```
50935 75440.031 0.000006019 13.778190 0.000351733 0.0133806 6
```

50935	ユリウス歴の日付
75440.031	ユリウス歴の時間 (UTC)
0.000006019	タイムオフセット (秒)
13.778190	周波数オフセット (PPM)
0.000351733	RMS ジッタ (秒)
0.0133806	Allan deviation 安定性 (PPM)
6	システムポール間隔

7.9.5 ピア状態監視コマンド

ピアの統計情報を記録するには, statistics コマンドのオプションに peerstats を指定する. ピア情報が更新される度に, peerstats: という名前でファイルが生成され, 次のような情報が書き込まれる.

```
48773 10847.650 127.127.4.1 9714 -0.001605376 0.000000000
0.001424877 0.000958674
```

48773	ユリウス歴の日付
10847.650	ユリウス歴の時間 (UTC)
127.127.4.1	ピアの IP アドレス
9714	ピアの状態 (Hex), RFC 1305 の付録 B.2 参照
-0.001605376	オフセット (秒)
0.000000000	遅延 (秒)
0.001424877	分散 (秒)
0.000958674	RMS ジッタ (秒)

7.9.6 raw 状態監視コマンド

raw タイムスタンプの統計情報を記録するには, statistics コマンドのオプションに rawstats を指定する. ピアやクロックドライバからの NTP メッセージを受信する度に, rawstats: という名前でファイルが生成され, ほぼ NTP メッセージに準じた情報が書き込まれる.

```
50928 2132.543 10.4.1.1 10.4.1.20 3102453281.584327000
3102453281.586228000 3102453332.540806000 3102453332.541458000
```

50928	ユリウス歴の日付
2132.543	ユリウス歴の時間 (UTC)
10.4.1.1	リモートピアの IP アドレス
10.4.1.20	ローカル IP アドレス
3102453281.584327000	リファレンス時間 (NTP 時間)
3102453281.586228000	送信時間 (NTP 時間)
3102453332.540806000	受信時間 (NTP 時間)
3102453332.541458000	転送時間 (NTP 時間)

7.9.7 システム状態監視コマンド

システムの状態監視を記録するには, `statistics` コマンドのオプションに `sysstats` を指定する. `sysstats:` という名前でファイルが生成され, 定期的に `ntpd` の次の統計情報が記録される.

```
50928 2132.543 36000 81965 0 9546 56 71793 512 540 10 147
```

最初の二つのカラムはユリウス暦による日付と時間を, 残りの 10 個のフィールドは順番に次の情報を示す.

50928	ユリウス歴の日付
2132.543	ユリウス歴の時間 (UTC)
36000	システムが再起動してからの経過時間 (hours)
81965	受信した総パケット数
0	パケットを送信してから受信したパケット数
9546	現 NTP バージョン番号を持つパケット数
56	以前の NTP バージョン番号を持つパケット数
71793	それ以外の NTP バージョンを持つパケット数
512	アクセスを拒否したパケット数
540	パケット長やフォーマットが不正なパケット数
10	認証が無効なパケット数
147	レート制限により破棄されたパケット数

7.9.8 ファイルの生成コマンド

今まで説明した統計情報を記録するファイルをどのような生成するかを filegen というコマンドで設定する. filegen コマンドの後に生成したい統計情報名を指定し, その後に次のオプションを指定する.

- ファイル名の指定
- ファイルタイプの指定
- リンクの有無
- ファイル生成の可否

なお, filegen コマンドで指定できる内容は, ntpdc でリモートから制御が可能である.

ファイル名の指定

file というオプションでファイル名 *filename* を指定できる. ファイル名は, 三つの名前を合成して出来上がる. 一つはパス名で statsdir コマンドで指定される (パス名で指定するディレクトリ名の最後は"/(スラッシュ)" で終わる必要がある). その次はファイル名でこれは, *filename* として file オプションの後で指定する. 3 つ目はサフィックスで type オプションで異なる. 例えば, type オプションで毎日ファイルを更新すると指定すると, ファイル名のサフィックスには年月日を表す数字 (例: 20031201) が付けられる. 従って, ファイル名は "*statdir/filename.suffix*" となる.

まず, 統計情報ファイルを置くパス名 (ディレクトリ名) はコンパイル時に決定されるが, ntpd.conf の中で設定したい場合は, statsdir コマンドで指定する. 例えば, /var/log/ntp の下に置きたければ, 次のようになる.

```
statsdir /var/log/ntp/
```

そして, loopstats を *looplog* というファイル名で保存したければ, 次のように指定する. ファイル名は/var/log/ntp/looplog となる.

```
statistics loopstats
statsdir /var/log/ntp/
filegen loopstats file looplog type none enable
```

タイプの指定

type というオプションを使うと生成されるファイルの機能 (例えば毎日ファイルを更新するなど) を設定できる. 機能は表 7.6 の通りである.

タイプ	説明
none	プレーンファイルでサフィックスは付かない.
pid	ntpd が起動すると新しいファイルが生成される. サフィックスにはプロセス ID が付く.
day	一日 (00:00 から 24:00) 単位にファイルが生成される. サフィックスには YYYYMMdd が付く. この意味は YYYY が西暦, MM が月, dd が日である. 例えば, 2003 年 12 月 1 日であれば “20031201” となる.
week	1 週間単位にファイルが生成される. サフィックスは西暦の後に文字 “W” を付け, その後に 1 年の通算の日付を 7 で除算した数を付ける. 例えば, 2003 年の第 1 週目であれば, “2003W1” となる.
month	月単位にファイルが生成される. サフィックスは西暦と月が付く. 2003 年 1 月であれば “200301” となる.
year	1 年単位にファイルが生成される. サフィックスは西暦が付く.
age	ntpd が起動してから, 24 時間ごとにファイルが生成される. サフィックスは文字 “a” を付け, その後に 8 桁でサーバの起動秒数を付ける.

表 7.6: filegen で指定できるファイルタイプ

例えば, 毎日ファイルを更新したい場合, filegen コマンドを次のように指定する.

```
filegen loopstats file looplog type day enable
```

ハードリンクの有無

link を指定する事で, サフィックスの無いファイル名へのハードリンクが生成される. 例えば, 前の例の場合 “looplog” というファイル名のハードリンクが生成され, 日付が変わってファイル名が更新された場合も最新のファイル名がハードリンクされる.

デフォルトはハードリンクが生成されるが, ハードリンク機能をやめたければ, nolink を指定する.

ファイル生成の可否

ファイル生成は enable を指定する事で有効になる. デフォルトで enable は設定されているので, ファイル生成を一時的にやめたい場合は, ntpdc を使って disable に設定すれば良い.

統計情報の記録例

下記はピア情報, 自サーバのクロック情報, クロックドライバの情報を 1 日単位で記録する場合の設定例である.

```
statsdir /var/spool/ntp/
statistics peerstats loopstats clockstats
filegen peerstats file peerstats type day enable
filegen loopstats file loopstats type day enable
filegen clockstats file clockstats type day enable
```

7.10 ntpd.conf で指定するファイル

7.10.1 ドリフトファイル

通常, どんな時計であっても正確にクロックを刻むわけではなく, クロックの性能や環境によっても正しいクロックパターンから外れていく (はずれ方も緩やかだったり急激だったり色々なパターンがある). この外れていくことを**ドリフト**と呼び, どのくらいの時間でどの程度外れるかを示すのに PPM(Part Per Million) という単位が用いられる. このドリフトレートが予めわかっていれば, クロックを補正させることができる.

ntpd もソフトウェア的なクロックを構成しているが, ネットワーク環境によってずれが起こる. ずれは一定では無く, 常に監視する必要があり, 特に初期動作時にどのようなドリフト値があるかを調べなければ, クロックを補正させる事ができない.

そこで, ntpd が計算した周波数誤差の値を記録するファイルを用意している. 起動時にこのドリフトファイルが存在すると, それが読み取られ, ntpd の周期誤差の内部値を初期化するのに使われ (その分, ノーマル運転までの時間が早くなる), その後は 1 時間に 1 回ずつ更新される. 値の更新はファイル自身を rename(2) ライブラリを使って更新するため, ドリフトファイルが置かれるディレクトリのパーミッションが ntpd デーモンにとって書き込み不可になっていると更新ができなくなってしまうため注意が必要である.

ドリフトファイルは ntpd.conf の中で dirftfile コマンドでパス名を変更できる. 例えば, /var/run/ntp.drift というパス名にしたければ, 次のように設定する.

```
driftfile /var/run/ntp.drift
```

7.10.2 ファイルの分割

ntpd.conf を分割したり, 付加的に加えるファイルを指定する場合, includefile コマンドが使われる. なお, ファイル名は絶対パスで指定する。

```
includefile /etc/ntp/server-1.conf
includefile /etc/ntp/server-2.conf
includefile /etc/ntp/ntplog.conf
```

7.10.3 ログファイル

syslog によってログが書き込まれるファイル名を指定するのが logfile コマンドである. そして, 何の情報を書き込むかを指定するのが logconfig コマンドで, 何も指定しなければ, 全ての情報が syslog に送られる.

logconfig コマンドはログキーワードに =(一致), +(追加), −(削除) を付ける事で, メッセージの出力を細かく制御できる. メッセージのクラスには 4 つのクラス (clock, peer, sys, sync) があり, それぞれのクラスに次の 4 つの制御が可能である. このクラスに加え, 全てのクラスを表す all が用意されている.

- info: 情報メッセージ
- events: イベントメッセージ
- statistics: 統計メッセージ
- status: 状態メッセージ

クラスとメッセージをつなぎ合わせてログキーワードを作る. 例えば, sys クラスののイベントメッセージ (events) は “sys+events” で sysevents となる.

logconfig コマンドに話を戻すと, ntpd の同期状態情報とメジャーなシステムイベントを syslog に吐き出す場合は次のように指定する.

```
logconfig=syncstatus +sysevents
```

全てのクロック情報と同期情報を syslog に記録する場合は次のように指定する.

```
logconfig=syncall +clockall
```

7.11 その他

7.11.1 システムフラグ

暗号化やモニタ等, いくつかのシステムオプションを有効にするかしないかは, enable(有効)/disable(無効) コマンドで設定することができる. 現在, 表 7.7 のように 8 つの機能が指定できるが, デフォルトで有効になっていないものは, 明示的に設定しなければならない.

フラグ	説明	デフォルト
auth	認証機能 (公開鍵, 対象秘密鍵)	有効
bclient	ブロード/マルチキャストのクライアント機能	無効
calibrate	リファレンスクロックのキャリブレート機能	無効
kernel	カーネル時間学習機能	サポート
monitor	モニタ機能	有効
ntp	時間と周波数の学習機能	有効
pps	PPS 信号を利用する機能	無効
stats	統計情報収集機能	無効

表 7.7: システムフラグ

自分のシステムがどのシステムフラグが立っているかは, ntpdc コマンドの sysinfo サブコマンドで知る事ができる.

```
ntpdc> sysinfo
system peer:          GPS_NMEA(1)
system peer mode:     client
leap indicator:       00
stratum:              1
precision:            -16
root distance:        0.00000 s
root dispersion:      0.00002 s
reference ID:         [GPS]
reference time:       c39227d7.ffffde12  Tue, Dec 23 2003 11:32:55.999
system flags:         auth monitor ntp kernel stats calibrate
jitter:               0.000000 s
stability:            2.595 ppm
broadcastdelay:       0.003906 s
authdelay:            0.000122 s
```

7.11.2 setvar コマンド

setvar コマンドは, 付加的システム変数を新たに加える際に用いる. “setvar 変数名 = 変数値” で変更でき, ntpq -c rv で確認できる. 変数名は既にあるシステム変数と重ならなければ良い. 例えば, access_policy という変数を作成するには, ntp.conf で次のように指定する. default というキーワードを付けることで, ntpq -c rv で表示させることができる.

```
setvar access_policy="open access" default
```

7.11.3 tinker コマンド

tinker コマンドは非常に例外的な状況下でシステム変数を変更するために使用される. ntpd は一般的な環境下では十分最適化されているが複雑な動きをするため, 非常に例外的な環境下でうまく動作しない場合に (滅多にない), デバッグなどを行いやすくする目的で以下のシステム変数を変更するのがこのコマンドの役割である.

オプション	説明
allan	PLL/FLL クロック規律アルゴリズムのパラメータの一つである最小アラン・インターセプトを設定する. デフォルト値は 7 (1024 秒).

オプション	説明
freq	周波数オフセットを設定する (ppm).
huffpuff	huff-n'-puff と呼ばれる新しいフィルタ (NTPv4 で実装) のスパンを設定する (デフォルトは 900 秒).
panic	パニック閾値を設定する (デフォルトは 1000 秒). この値を 0 にするとパニックサニティーチェックは行われず, どんなクロックオフセットでも受け付けるようになる.
step	ステップ調整の閾値を設定する (デフォルトは 128 ミリ秒). 0 にするとステップ調整は行われない. この値を 0 あるいはデフォルトよりも大きな値にすると, カーネル時間規律は行われない.
stepout	ステップタイムアウトの時間を調整する (デフォルトは 900 秒). 0 に設定すると, ステップアウトパルスは抑えられない.

第8章　SNTP

8.1　SNTP とは

SNTP(Simple Network Time Protocol) は, NTP(RFC 1305) の簡易版といえるもので, 組み込み機器のような複雑な処理ができない機器やクライアントコンピューが正確な時刻を問い合わせることができるよう NTP の仕様の中から複雑な部分 (PLL/FLL に基づくソフトウェア時計, ストラタムの概念, 長時間に渡る統計的な誤差の処理など) を省略して軽量化したプロトコルである. RFC 4330[19](RFC 2030 の改定版) で定義されている.

NTP と SNTP のパケットフォーマットや時間, オフセット, 往復遅延の算出方法は同じである. そのため, サーバが NTP なのか SNTP なのか, クライアントが NTP なのか SNTP なのかはどちらからも判別できない. SNTP は, NTP が行う複雑な部分を省いているため, サーバは非常に多くの NTP クライアントに時刻を供給できる. しかし, NTP が持つ高機能なソフトウェア時計を持たないため, 時刻源と直結していることが求められており, 通常ストラタム 1 や 2 に同期させて動作させる.

オペレーションモードとしては, NTP と全く同じモードで動作する. 従って, ユニキャスト, ブロードキャスト, マルチキャスト, メニーキャストのどのモードでも動作する. 但し, OSI にはブロードキャスト/マルチキャストが定義されていない.

8.2　SNTP の実装

完全な SNTP はデラウェア大学の NTP パッケージが実装している. `sntp` コマンドは, SNTP クライアントに加え, サーバモードも実装している.

クライアントとして動かすには, 引数に SNTP サーバを付けて動作させればよい.

```
# sntp -4 ntp.nict.jp
2006 Jun 15 17:47:38.481 + 5.477 +/- 0.018 secs
```

`cron` や `at` のように定期的に実行するのではなく, デーモンとして動かすことで, 永続的に時刻同期が可能である. その時, `-x` を付けることで, クロックドリフトが作成され, クラ

イアントとサーバの間に大幅な時刻のずれが生じた場合に時刻合わせを中止させることもできる (デフォルトでは 300 秒/日). `sntp` はデフォルトで adjtime されないので, スムースに時刻同期を行うよう, `-a` を付けた方がよい.

```
sntp -a -x -4 ntp.nict.jp > output 2>1
```

SNTP サーバとして動作させるには, 引数にサーバを指定せずに動作させるとデーモンとして動作する.

8.3 アクセスマナーについて

RFC 4330 には, クライアントがサーバにアクセスする際にネットワークやサーバの資源に負担をかけないためのベストプラクティスが示されている.

1. クライアントはどんな条件の元でも, 15 秒より小さなポーリング間隔にしてはならない (参考: 7.5.2 節, `ntpd` は `minpoll` 値の最小が 4).
2. クライアントは, サーバが妥当な時間内に応答がなければ, 指数的なバックオフを使ってポーリング間隔を増やしていくべきである.
3. クライアントはバックボーンネットワークに不必要なトラヒックの増加を避けるために, ローカルな NTP サーバを使うべきである.
4. クライアントはファームウェアで指定されたデフォルト IP アドレスの代わりに, あるいはそれに加えて, 一次または更に別のサーバが設定できるようにしておかなければならない.
5. ファームウェアでデフォルトサーバの IP アドレスが設定するなら, そのサーバは製造会社またはデバイスの販売者によって運用されたサーバでなければならない, もしくは運用者の許可が得られた場合のみ別のサーバが設定できる.
6. クライアントはサーバ運用者が NTP サーバのロードバランスや IP アドレスの変更した場合にも対応できるように, サーバの IP アドレスの解決に DNS を使うべきである.
7. クライアントは定期的な間隔でサーバの IP アドレスの再解決すべきである. しかし, DNS 応答の中の TTL より小さな値で再解決を行ってはいけない.

8. クライアントは NTP アクセス拒否メカニズムをサポートすべきで, クライアントの要求に対してサーバの Kiss-o'-Death が応答された場合, クライアントはサーバへの要求を中止する.

第9章　メニーキャスト(自律型コンフィギュレーション)

9.1　自律型コンフィギュレーションとは

NTP は正確な時刻を持つ NTP サーバと同期することで自システムの時刻を合わせるプロトコルで, 同期する NTP サーバの候補を複数持たせることで, 信頼性と冗長性を高めている.

サーバの設定は同期した NTP サーバをマニュアルで指定する必要があるため, 管理者が上位ストラタムのサーバを何らかの手段で知るか, 公のサーバリストが必要となる[1].

このような煩わしさを軽減するため, NTPv4 では自律型コンフィギュレーション (NTP Autonomous Configuraion) 法が検討された. 自律という意味は, どこかで仲介者が介在する形 (例えば, 集中的に NTP サーバのデータベースの管理やリストの配布を行う人や組織がある) ではなく, それぞれのシステムが自律的に動いている状態でサーバの諸設定が行われることを意味し, この機能の実現に必要となるサーバの発見, セキュリティの設定 (具体的には鍵の交換) が行え, ネットワークトポロジーやサーバの変更があっても, なんら不都合なく動作するものでなければならない. また, NTP では精度の面も考慮に入れたものでなければならない — 近くにサーバが存在するにもかかわらず, 遠くにあるサーバと時刻同期をしては意味がない.

検討の結果, NTPv4 から自律型コンフィギュレーションシステムとして, **メニーキャスト**と呼ばれる方式が実装されている.

9.2　メニーキャストとは

メニーキャストは NTPv4 から新しく採り入れられた機能の一つで, 個別の NTP サーバを設定することなしに, クライアントがネットワーク的に近い NTP サーバを発見し, セキュリティ機能を含め, 個々のホストが自律的に NTP の設定を行うための手法である.

[1] 例えば, 誰か (例えばプロバイダ) がサーバリストを広く伝搬させるか, NTP ソフトウェアの中に NTP サーバリストを組み込んだ形で配布することが必要だ.

最小の設定だけで良いという意味では, ブロードキャスト, マルチキャスト, エニーキャストや DNS を使っても可能である. マルチキャストの場合, NTP サーバのあるネットワークが Mbone のようなインターネットの広域マルチキャストネットワークに接続されていれば, `ntpd.conf` に `multicastclient` コマンドを一つ書き加えるだけの設定で良い. しかし, マルチキャストはサーバから時刻情報がマルチキャストメッセージとしてタレ流し的に送られて来るため, パケットのジッタやワンダの影響が大きく, 時刻精度がユニキャストに比べて安定していない.

次に, RFC 1546[9] にあるようなエニーキャストを利用する方法も考えられる. エニーキャストは確かにネットワーク的に近くにあるサーバとの間でサービスが行えるが, あくまで 1 対 1 であり, 冗長構成を持たせにくく Point of Failur に弱い.

その他, DNS のラウンドロビン名前解決を利用する方法もある. `pool.ntp.org` プロジェクト (13.3 節, 171 ページ参照) では, DNS のラウンドロビン機能を利用して, 設定を簡易にしている. DNS の場合は集中的な管理が必要であるという点で自律的なシステムかは少し疑問がある. また, DNS の名前解決はメッセージ長の制約があるため, 一度に送れるサーバの IP アドレス数が限定されてしまう. `pool.ntp.org` では, サーバリストからランダムにサーバを抽出して DNS を更新するプログラムを作って, この制限を越えるホスト数を登録できるようにしているが, 一度の名前解決で表示されるホスト数には制限がある.

そこで, クライアントが簡単な設定を行うだけで NTP の時刻同期が可能となるよう, 新たな方式が考えられた. それがメニーキャスト・アソシエーションで, クライアントが複数のサーバを自動発見し冗長化を持たせる事ができ, しかも, そのサーバと安全に時刻情報をやり取りするために対象鍵・公開鍵 (Autokey) とも利用できる点など, マルチキャストやエニーキャストと比べると表 9.1 のように優れている.

項目	メニーキャスト	マルチキャスト	エニーキャスト
設定自動化	○	○	○
セキュリティ	○	○	○
時刻同期の精度	○	×	○
サーバの冗長化	○	○	×

表 9.1: 各方式の比較

9.3 メニーキャストの動き

まず, メニーキャストクライアントはサーバの存在を発見するためにマルチキャストを使ってマルチキャストクライアントメッセージをブロードキャストする. マルチキャストグループアドレスは IPv4 の場合はクラス D アドレス 224.0.0.0/4 が, IPv6 の場合は FFx::が使われる[2]. メッセージの伝搬範囲を限定する場合は, TTL を小さな値に設定する.

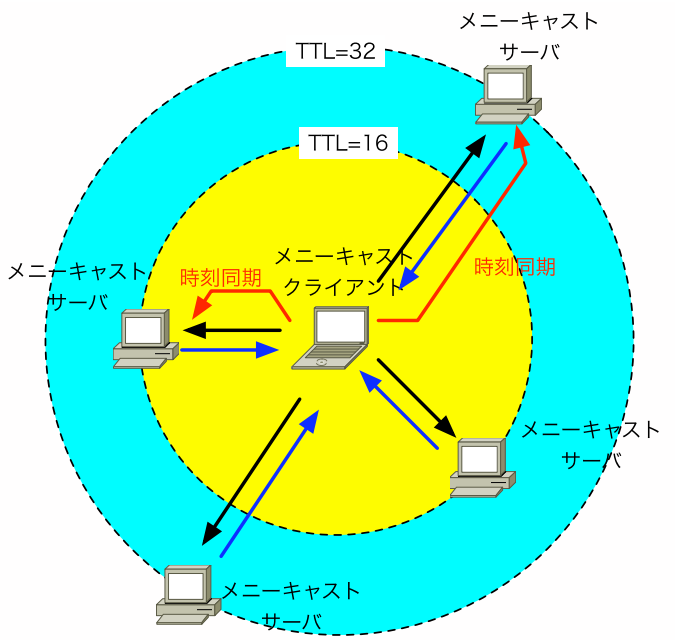

図 9.1: メニーキャストの動き

次に, NTP クライアントは NTP サーバから応答メッセージを待ち, 応答メッセージがあれば, NTP サーバが発見された事になる. 発見された NTP サーバから自分のストラタムより小さな値で認証が有効であるサーバを選択し, 選択した NTP サーバに対してユニキャストでポーリングを行い, 時刻を同期させる.

[2]NTP 専用に IANA では, IPv4 アドレス 224.0.1.1(`NTP.MCAST.NET`) と IPv6 アドレス [Prefix]:101 が予約されている.

9.4 メニーキャストの設定

9.4.1 クライアント側の設定

メニーキャストクライアント側の設定は manycastclient コマンドが使われる．コマンドオプションはメッセージを送信するマルチキャストグループアドレスを指定する．グループアドレスを指定しない場合, デフォルトのマルチキャストグループアドレスが使われる．

なるべく近くのサーバを発見した方がクロックの精度が良くなるので, メニーキャストではマルチキャストメッセージの送信に徐々に遠くのノードを探索する手法 (expanding ring search) が使われる．この手法は, メッセージの送信の際に少しづつ TTL 値を増やし, 徐々に遠くのサーバを検索する手法で, 望みのサーバ数が得られれば, メッセージの伝搬をやめて検索が広範囲にならないよう防ぐ役割を持っている．manycastclient コマンドでは, 最大 TTL 値を ttl オプションで指定し, TTL の増加分を ttl コマンドで指定する．デフォルトの最大 TTL は 127 で, 31 で開始され 32 の 8 づつ増えていく．16 づつ増やしていく場合, 次のように指定する．

```
manycastclient ttl 127
ttl 16
```

その他, ストラタム値の制限や応答サーバの最低確保数などは 9.4.3 節で説明する tos コマンドで指定する．メッセージの送信間隔は server や peer コマンド同様, minpoll, maxpoll を用いる．また, 対象秘密鍵 (key) や Autokey(autokey) の設定も同じように行うことができる．また, NTP メッセージのバージョンを指定する場合は, version を指定する．

例えば, 1024 秒間隔で送信するには, 下記のように指定する．

```
manycastclient 224.100.100.1 minpoll 12 maxpoll 12
```

9.4.2 サーバ側の設定

サーバ側は manycastserver でグループアドレスを指定する．アドレスを指定しなければ, IANA が割り当てた NTP マルチキャストアドレスが使われる．これで, サーバはメニーキャストクライアントメッセージを受信し, 応答することができる．なお, 対象秘密鍵や Autokey の設定は, クライアントサーバの設定時と全く同じである．

```
manycastserver 224.100.100.1
```

9.4.3 メニーキャストのオプション

メニーキャストをインターネットで使えるようにするには, マルチキャスト環境がなければならない (ここが最大の難点である). もし, マルチキャスト環境がインターネットでふつうに利用できるようになると, 多数のサーバからマルチキャストクライアントメッセージが返ってくることも考えられる. その場合, 品質に影響が出るし, NTP ピア数の増大を招く可能性がある. そこで, 品質とピア数を抑えるためにクロック選択法とクラスタリングアルゴリズムを制御する tos コマンドが用意されている.

まず, クラスタリングアルゴリズムを働かせる最低のピア数を minclock オプションで設定する事ができる. 1 から設定することができるが, デフォルト値は 3 である. また, minclock の数のピアがあれば, ceiling で指定したストラタム数以上のピアを無視させることができる. 例えば, ceiling を 5 に設定すると, 発見されたサーバの中にストラタム 6 のサーバがあったとするとそのサーバとはピアを行わない. 加え, floor で指定したストラタム数以下のピアを無視させることもできる. 例えば, floor を 3 とすると, ストラタム 1 や 2 のサーバとはピアを張らないことになる. なお, ceiling のデフォルト値は 15 で, floor のデフォルト値は 1 なので, 全てのピアから時刻同期を受けられる状態になっている.

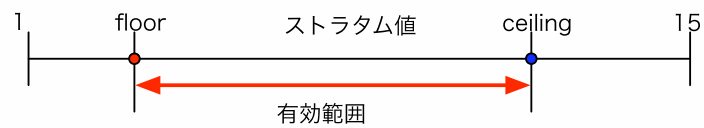

図 9.2: ceiling と floor の関係

最初にメニーキャストサーバが自システムのストラタムより大きい場合は時刻同期を行わないと説明したが, 同一ストラタムの場合に同期をするかしないかを cohort オプションで指定する. 指定は enable(0) と disable(1) を指定でき, デフォルトは enable(1) になっている.

次にクラスタリングアルゴリズムをうまく機能させるため, クロック選択アルゴリズムで使用する候補となるサーバの最小数を指定するのが, minsane オプションである. この数が小さければ, クロックは規律は取れなくなり, 自由に走らせてしまうことになる. デフォルトでは古いシステムとの互換性のため 1 に設定されているが, ビザンティン規約の原理によると少なくとも 4 に設定すべきである[3].

例えば, ストラタム 4 を超えるサーバとはピアせずに, クロック選択アルゴリズムで使用

[3]ビザンティン故障に対する耐性として, 全プロセス数 $t(< n/3)$ 個の停止故障に耐性がある?

する候補を 4 とする場合は次のように設定する.

```
tos ceiling 3 minsane 4
```

第10章　Autokey (公開鍵認証)

10.1　NTP におけるセキュリティのねらい

NTP サービスはあまねく広く利用されるべきインフラサービスで, 時刻情報は既知の情報であるため, 通信路上に流れる時刻情報を秘匿する必要は無い. むしろ, 時刻情報の改竄やサーバのなりすましによる偽の時刻情報の配信がもっとも懸念される事項と云える. そこで, NTP が考慮すべきセキュリティは, メッセージ認証機能の強化にある. NTPv3 からサポートされた対象鍵 (秘密鍵) を使ったメッセージ認証機能はメッセージの認証性を高めるための有効な手段だが, 送られて来たメッセージの身元をはっきりさせる機能はない. また, 鍵の配布方法が定義されていないため, クローズなグループでの時刻配信サービスには有効だが, インターネットのような大きなネットワークで利用するには安全な鍵の配布方法を考える必要がある.

送られて来たメッセージが適切なサーバから送られたものかを検証するため, 公開鍵暗号技術を採り入れる方法が考えられる. NTP で必要となるセキュリティは, ホスト認証とメッセージの認証である. また, NTP のホストはクライアントでもあり, サーバでもあるという特徴や, 自律型の設定機能 (第 9 章, メニーキャスト) を持っており, これらの機能と親和性が求められる.

10.2　対象鍵による認証方式

まず, NTPv3 と NTPv4 が使っている対象鍵暗号による認証方法をもう一度おさらいする.

NTP ホスト間のメッセージ認証に対象鍵 (共有秘密鍵) を用いる方式は, RFC 1305 がオリジナルで, 最初は暗号アルゴリズムに DES-CBC[1]が用いられていた. その後, DES の安全性が不安視されるようになり, 現在は MD5 のみが用いられる.

対称鍵方式は, NTP メッセージを交換したいホスト間であらかじめ共通の秘密鍵 (128 ビットのプライベート鍵と 32 ビットの鍵 ID) を設定しておいて, 送信側が NTP メッセー

[1]共通秘密鍵を使ったブロック方式の暗号化を CBC モードで行い, 途中の暗号ブロックを捨て, 最後の暗号ブロックをメッセージダイジェストとみなす方式. DES は 64 ビットで 1 ブロックであるため, MAC 長は 64 ビットになる. MAC にするには短過ぎると言われて, 今は用いられていない.

ジを所定のプライベート鍵と暗号アルゴリズムでメッセージハッシュを計算し, 使用した鍵 ID とハッシュ値を NTP メッセージに添付して送信する (鍵 ID は通信路上では公知になる). 受信側は受信した NTP メッセージを同じ鍵とアルゴリズムでハッシュの計算を行い, 送信されてきたハッシュ値との比較を行い, 両者のハッシュ値が一致していればメッセージ認証が成功したことになる (図 10.1). 鍵付きの MD5 を使うことで認証に加えてパケットの完全性も保証される. この方式は一般的に**メッセージ認証コード** (MAC) あるいは IPsec の ICV(Integrity Check Value) の考え方と同じである[2].

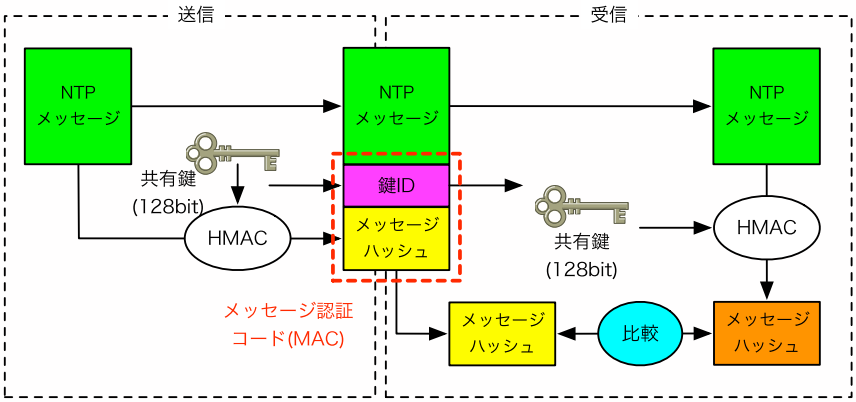

図 10.1: 対象鍵による NTP の認証

双方の鍵の識別には先ほど述べたように 32 ビットの鍵 ID が用いられ, この鍵 ID は対象鍵と Autokey の鍵が混同しないよう, 65,536 を境に二つのサブスペースに分けて使われる. 対象鍵が使う鍵 ID は 1 から 65,535 までで, これより大きな鍵 ID は Autokey で使われる.

対称鍵方式の問題点はどのようにして鍵を同意し, 全サーバに配布するかという点にあると説明した. NTPv3 ではこの点は全く言及されていない. 例えば, 1 週間に一度鍵を交換するという約束にして, 3ヵ月分なり半年分の鍵をあらかじめ安全な手段で配っておくという考え方がある (エニグマ鍵のようだが). そして, 期限が来たら, `ntpdc` コマンドで鍵を無効 (revoke) にする. ただ, 鍵の管理を定期的にマニュアルで行う必要があり, 人為的ミスも発生しやすい. やはり, 鍵の管理は自動化が望まれる.

[2]鍵を使わなければ単なるハッシュで, MAC は鍵付きハッシュ(HMAC: Keyed-Hashing for Message Authentication[14]) である.

10.3 公開鍵認証を採り入れるには

NTPv3 の対象鍵方式による認証スキームはメッセージ認証コード (MAC) と呼ばれる方式で[3], 事前に鍵を設定し, しかもそれを NTP のピア同士で共有しておかなければならない. そのため, 鍵の配布に人手が介在することになり, 自律的 (autonomous) な仕組みであるとは言えない. また, 対称鍵方式はメッセージの認証は可能だが, 送られて来たメッセージが本当に正当なホストから送られて来たものなのかの判断ができない. 攻撃は下記のように多様化しており, 様々な攻撃から身を守るには新たに強固なセキュリティ方式を考える必要が出てきたといえる.

ここで, 一般的なネットワークの攻撃をおさらいする. ネットワーク上の攻撃には主に次の 4 つの方法を挙げることができる.

- **再生攻撃 (reply attack)**：この方法は正規のユーザがサーバとやり取りしたデータを盗聴記録しておき, 同じデータをサーバに送りつける事で正規ユーザになりすまそうとする攻撃である. 単純なユーザ名とパスワードによるリモートログインは簡単にこの方法でログインされてしまう.

- **メッセージ (パケット) の改変 (modification)**：メッセージの改変は攻撃者が通信途中に媒介し, メッセージの中身を攻撃者にとって都合の良いように変更してしまう攻撃をいう. メッセージの偽造・改竄 (falsification) とも呼ばれる.

- **なりすまし (spoofing)**：攻撃者が第三者になりすます事で攻撃をし, 犯人を特定されにくくする手段である. 仮装攻撃 (masquerade attack), スプーフィング攻撃 (spoofing attack) とも呼ばれる. 発信先を隠す目的で, 偽の送信元 IP アドレスを持つパケットを使って, DoS 攻撃にも利用される.

- **clogging 攻撃 (clogging attack)**：サービスを妨害する攻撃をいう. 大量のデータや不正パケットをサーバに送り, 回線を詰まらせたり (clogging), サーバの応答を遅くする攻撃である. 一般的には DoS(Denial of Service: サービス不能) 攻撃と呼ばれる. 攻撃元を複数箇所から行う DoS 攻撃を DDoS(Distributed Denial of Service) と呼ぶ.

再生攻撃やメッセージの改変の検出には, これまでに説明したメッセージ認証コードを使った方式によって可能だが, 送信元の検証を行うためにはデジタル署名や X.509 などの身

[3] この方式が考えられた当時はコンピュータの処理性能も低く, DES や MD5 と比較して大幅に処理時間がかかる公開鍵暗号の採用は時刻の正確を失わせる原因にもなりかねなかったと思われる (MD5 や DES-CBC と比較して, RSA は 100〜1000 倍処理時間がかかる). 現在では, 当時のコンピュータと比較して, 100 倍以上の処理性能があり, 公開鍵アルゴリズムを使っても十分実用に耐え得るようになった.

元検証技術を使う必要がある．メッセージ認証など全てを公開鍵で行う方法も考えられるが, 公開鍵は処理に時間がかかるため, NTP のような時間にクリティカルなプロトコルではそれは致命的な欠点である．そこで, メッセージ認証は対称鍵に任せ, 鍵の同意・交換や身元確認の部分を公開鍵技術で補った方が両者の良い点を採り入れることができる．このように, NTP では対称鍵方式と公開鍵方式を組み合わせたハイブリッド方式を採用することになった．

- デジタル署名を使って, 相手の身元認証を行いつつ, 公開鍵を配布する．
- その公開鍵を使って鍵の同意と交換を行う．
- 鍵の同意ができたら, 対称鍵でメッセージ認証を行う．

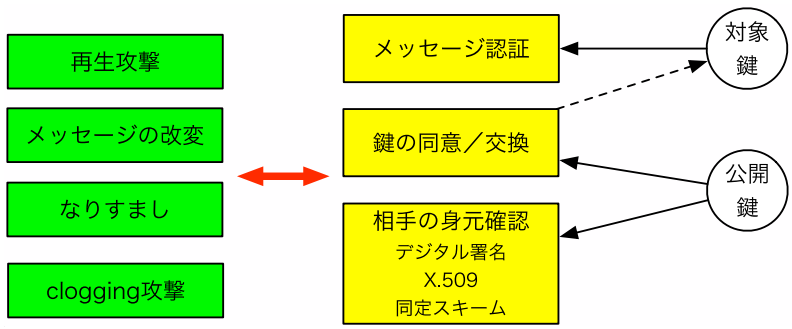

図 10.2: NTP の公開鍵の考え方

NTPv4 では公開鍵をベースとした認証方式, メッセージダイジェストを使ったパケットの完全性の検証, デジタル署名を使った送信元の身元検証, そしてオプションだがチャレンジ・レスポンスアルゴリズムをベースにした同定スキームをサポートする．特に同定スキームは, ゼロ知識証明 (Zero Knowledge Interactive Proof) と呼ばれるパスワードを通信路上に流さないで, 相手の身元を確認するアルゴリズムが用いられており, 安全性が増すと考えられている．これらの方式を全て使用すると, 再生攻撃, パケット改竄, なりすまし, clogging 攻撃などから保護でき, とても強力なセキュリティを提供できる．

10.4 Autokey とは

NTPv4 では, 対象鍵方式との互換は維持したままセキュリティの強化を行い, 公開鍵暗号をベースにした **Autokey** と呼ばれるセキュリティプロトコルを開発した．

Autokey には二つの役割がある．一つは NTP の各アソシエーションモードに応じてホスト間で自律的に鍵の設定や交換を行って, メッセージの認証や完全性を検証する (メッセー

ジの認証や完全性の検証は MAC を用いる). もう一つは身元証明の機能である. ここで自動的に設定される鍵は MAC 鍵のことで, 鍵は疑似乱数機構を使って自動生成され, メッセージごとに変更される. なお, この機構はアソシエーションモードに応じてやり取りの仕方が多少変わってくる.

Autokey プロトコルは NTPv4 で実装されており, OpenSSL ライブラリ[4]の暗号アルゴリズム機能をベースにして, メッセージダイジェスト, デジタル署名, X.509 証明書機能をサポートする. 従って, Autokey を使う場合は必ず OpenSSL ライブラリが必要となる.

10.5 Autokey の動作

Autokey が使用している暗号アルゴリズムは難解だが, 動作はそれほど複雑ではない. しかし, 仕様となるべき IETF Draft[23] が古くなっており, 執筆時点では Autokey がどのように動作するのか不明な点が多かった. そこで, Autokey の動作や設定などは ntp-4.2.n の実装と照し合せて説明をしていきたい. なお, 先に述べたように, Autokey の仕様は確定しておらず, 将来ここで説明した動作と違う動きをする可能性があることを了解していただきたい.

10.5.1 Autokey のメッセージ

Autokey は Autokey 専用のメッセージを交換することで鍵や証明書等の情報交換が行われる. Autokey メッセージは NTPv4 で追加された Autokey の拡張フィールドが用いられ, 表 10.1 に示される 9 つのタイプが用意されている. これらのメッセージを利用して, 相手の認証, 鍵の交換そしてクッキーを元に MAC 用の鍵を生成する.

10.5.2 Autokey の大まかな動き

最初は, クライアントがサーバに ASSOC メッセージを送ることから始まる. ASSOC メッセージを交換し合い, Autokey のパラメータを交換することになる. 次に CERT メッセージで証明書の交換を行う. PC スキームと TC スキームはここで認証が終わる. もし, オプションとして同定認証を使う設定になっていれば次に IFF, GQ, MV メッセージのいずれかを使って相手認証が行われる. そして, 問題なく認証が行われれば, COOKIE メッセージでサーバ側のクッキーを受信し, クッキーを元に MAC 鍵を生成し, 以後は対称鍵認証が行われる. その後は使い捨てパスワードのようにパケットごとに鍵が変わっていく. これが Autokey の大まかな動きである.

[4] http://www.openssl.org/

タイプ	Opcode	内容
Association	ASSOC(1)	アソシエーションパラメータとホスト名の要求と応答で, 署名されていない.
Certificate	CERT(2)	署名者と有効期限の付いた X.509 証明書の要求と応答.
Cookie	COOKIE(3)	公開鍵を送信し, 暗号化されたクッキーを受信する.
Autokey	AUTO(4)	Autokey のデータを要求し, Autokey データを応答する.
Leapseconds	LEAP(5)	閏秒テーブルを交換し, 最新版を管理する.
Sign	SIGN(6)	クライアント証明書を送り, 署名されたクライアント証明書を受信する.
IFF Message	IFF(7)	IFF 同定スキームのデータを交換する.
GQ Message	GQ(8)	GQ 同定スキームのデータを交換する.
MV Message	MV(9)	MV 同定スキームのデータを交換する.

表 10.1: Autokey のメッセージ (`ntp_crypto.h`)

10.5.3 Autokey パラメータの交換

まず, ASSOC メッセージを使って, クライアントとサーバ間で Autokey パラメータが交換される. ASSOC メッセージの中で交換されるデータはホスト・ステータス・ワード[5]とホスト名である. ホスト名は `gethostbyname()` によって得られる文字列が入る. クライアントからサーバに対しては自身のホスト名を付けてメッセージが送られ, サーバ側からはステータスワードに Autokey を実装しているかどうか, 使用可能な認証スキームは何かを示した情報を入れて, ホスト名と共に送られる.

10.5.4 証明書の交換

次に CERT メッセージを使って, 証明書を交換し, 証明書の検証や署名が行われる. この時, クライアントからサーバの証明書を要求する時は, メッセージは署名されていない. そして, サーバからクライアントに自己署名された証明書が送られ, クライアントはそれを保時する. なお, 証明書にはサーバの公開鍵やグループ鍵等が埋め込まれており, フォーマッ

[5]ホスト・ステータス・ワードは 32 ビットで構成され, 様々な Autokey に関する情報が含まれる. 大きく二つに分けられ, ローオーダの 16 ビットは Autokey プロトコルの状態を定義し, ハイオーダの 16 ビットはメッセージダイジェストや署名の暗号スキームの方法が入る.

トには X.509 が使われている.

10.5.5 同定スキームによる認証

オプションとして, 同定スキームを使用するよう設定されていると, 証明書の交換の後に同定スキームを実施する. 現在, 同定スキームには IFF, GQ, MV の 3 つの方式があり, それぞれの同定スキーム用のメッセージを用いて, 相手の認証を行う. 認証が成功すれば, クッキーの同意状態に移る.

10.5.6 クッキーの生成と同意

証明書の交換あるいは同定スキームの後に, クッキーの同意を行う. クライアントは署名の付いた公開鍵をサーバに送って, クッキーを送ってもらうようリクエストを送る. サーバ側は署名の検証を行い, 確認が取れれば, クライアントアドレス, サーバアドレス, 鍵 ID(0), 疑似乱数で生成されたプライベート値 (4 バイトの乱数[6]) からハッシュの計算を行い, クッキーを作る. そして, サーバは生成したクッキーを, 送られて来た公開鍵で暗号化して署名し, クライアントに送信する. クライアントはサーバの署名を検証し, 確認が取れればクッキーを復号して初期のクッキー値を得る. 以上がクッキーの獲得までのやり取りである.

但し, クッキーはアソシエーションモードによって異なる値を持つ. クライアント/サーバモードのクッキーは IP アドレスやプライベート値のハッシュを取り生成される. シンメトリモードでは, Diffie-Hellman 同意鍵を使って, サーバ間でクッキーを共有する. そして, マルチキャスト/ブロードキャストモードではクッキーの値は 0 である (すなわちクッキーの交換は行われない).

10.5.7 セッション鍵の生成

クッキーが得られたら, このクッキー値を使ってセッション鍵を生成する. セッション鍵は図 10.3 のように, IP アドレス (送信, 受信), 鍵 ID, クッキーの 4 つから構成され, このセッション鍵からメッセージ認証コード用の秘密鍵を得て, メッセージ認証に使われる.

ここで, セッション鍵を構成する IP アドレスは IPv4 と IPv6 の場合でパケット長が異なるため, IPv4 のセッション鍵は 4 ワード (1 ワード =32 ビット) で構成され, IPv6 のセッション鍵は 10 ワードで構成されることになる.

セッション鍵も鍵の識別用に鍵 ID を持っている点で NTPv3 と何ら変わりない. 以前述べたように, NTPv3 の鍵 ID と下位互換を持たせるために, 対象鍵とは重ならないようにセッション鍵の鍵 ID は 65,536 以上の値が取ると説明した[7]. 鍵 ID の生成には疑似乱数 (ノ

[6] OpenSSL の RAND_bytes() 関数を使って生成する.
[7] `ntp_crypto.c` 参照.

図 10.3: セッション鍵のフォーマット

ンス) [8]が使われ, 65,535 より大きな値且つキャッシュにその鍵 ID がなければ, それが当該セッションの初期鍵 ID(Autokey シード) となる. なお, 特別に鍵 ID が 0 のセッション鍵が用意されており, これは NAK リプライとして利用される.

```
/* ntp_crypto.c の中より */
while (1) {
        keyid = (u_long)RANDOM & 0xffffffff;
        if (keyid <= NTP_MAXKEY)
                continue;
        if (authhavekey(keyid))
                continue;
        break;
}
```

NTPv3 の場合, 対象鍵は設定されたら変更されない限りそのまま使用されるため, 事実上無期限なのに対し, セッション鍵はメッセージが交換されるたび (ポーリングごと) に消去される (デフォルトでは最大 1 時間の生存時間).

セッション鍵の生成で見落としてならない事は, IP アドレスを Autokey の要素とするため, アソシエーションが確立している間に IP アドレスが変更されてしまう可能性がある NAT(Network Address Translate) やモバイル環境下で Autokey を使う事ができない点である. モバイル IP で Autokey を利用するには, IP アドレスに代わるユニークな識別 ID が必要となるが, Mills 教授らを中心とした NTP プロジェクトで検討中のようである[9] .

10.5.8 メッセージ認証の開始

図 10.4 のように, IP アドレス (Dst/Src), 鍵 ID 及びクッキー値から生成されたセッション鍵を元に MD5 メッセージダイジェストアルゴリズムでハッシュ計算が行われ, 128 ビットの鍵データ値が得られる. 鍵のネットワークバイトオーダの最初の 32 ビットが次の鍵 ID となり, セッションで必要になる鍵を 100 個生成する (この 100 個という値は, `ntp_crypto.h` の `NTP_MAXSESSION` で定義されている). そして, 鍵は鍵 ID と共にメモリにキャッシュされ

[8]mrand() もしくは random() 関数が用いられる.
[9]`http://www.eecis.udel.edu/~mills/autokey.html`

る. 100 個の鍵を使いきったら, もう一度最初からセッション鍵を作り直す (セッション鍵の生存時間は ntp.conf の中で, automax コマンドで設定できるようだ).

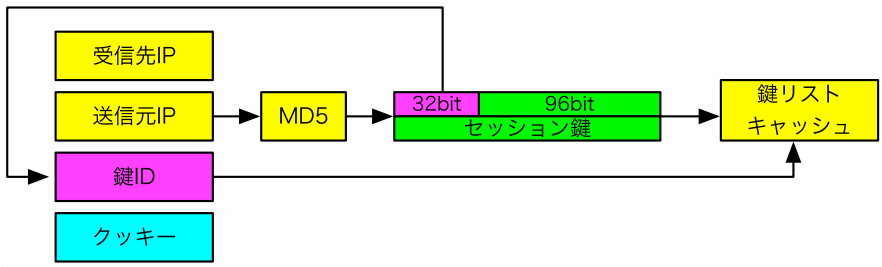

図 10.4: セッション鍵リストの構築

メッセージの認証には先ほどの手順で得られた MAC 鍵を使って, 図 10.5 のように手順で行われる.

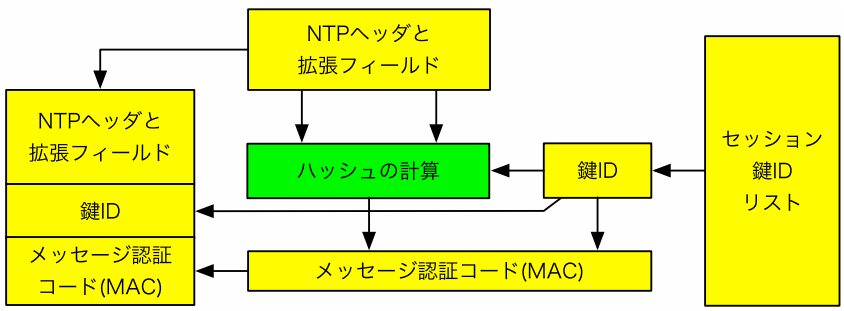

図 10.5: Autokey でのメッセージの転送

10.6 Autokey の拡張フィールド

Autokey で使われる鍵, 証明書や署名の交換をするために, NTP メッセージと MAC の間にセキュリティの拡張フィールドを設け, Autokey の様々な情報の交換に使われる. 拡張フィールドは一つの NTP メッセージに複数持つことができるが, 頭から順番に処理される.

一つの拡張フィールドは図 10.6 のようなフォーマットで構成される[10].

[10] Autokey バージョン 1 では異なるいくつかの拡張フィールドが作られたが, バージョン 2 ではひとつに統一された.

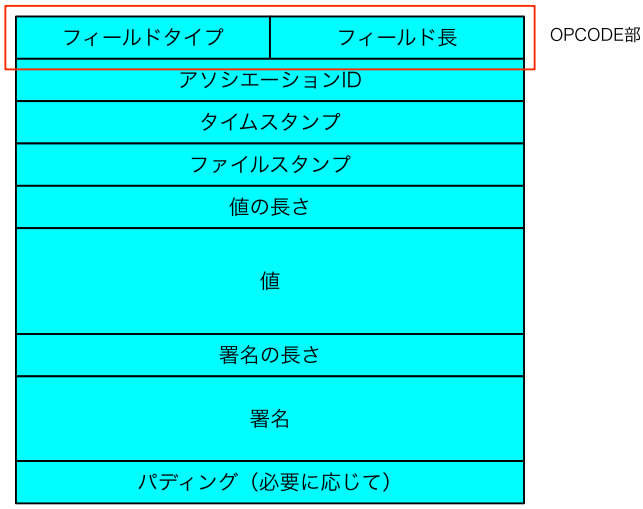

図 10.6: NTPv4 の拡張フィールド

10.6.1 拡張フィールドの特徴

拡張フィールドは各フィールドが 32 ビット単位に収まるよう, パディング (連続したゼロ) が挿入される. また, フィールドの最後は 64 ビットで収まるよう, パディングが挿入される (従って, 一つの拡張フィールドの場合は 64 ビットで収まるようにパディングが挿入されることになる). 拡張フィールド中のフィールド長はパディングを含んだ値である. なお, リファレンスの実装では, 1024 バイト[11]以上のフィールド長を持つパケットは破棄されるようになっている.

ここで注意すべき点がある. MAC は拡張フィールドを含んだ形で計算される. MAC と拡張フィールドがあるかどうか, ヘッダの最後からパケットの最後までの残りのエリアのパケット長から決定できる. `ntpd` に実装では, パーザは残りのパケット長を計算し, 4 の倍数でなければフォーマットエラーでパケットを破棄している. そして残りのワード (32 ビット) で拡張フィールドがあるかどうかが決定される.

[11] `include/ntp.h` の中の `NTP_MAXEXTEN` で定義されている.

0	パケットは認証されていない.
4	前のパケットの結果がエラーレポートか crypto-NAK で,
2, 3, 4	パケットがフォーマットエラーで破棄される.
5	パケットはメッセージ認証コードである (鍵 ID+MAC).
5 より大きい	一つ以上の拡張フィールドがある.

10.6.2 拡張フィールドの構成

OPCODE 部は次のように構成されている.

ビット数	名称	説明
0	R	レスポンスフラグでサーバからの応答メッセージであれば, このビットが 1 になる
1	E	エラーがあれば, このビットが 1 になる
2∼7	Version	Autokey のバージョン番号 (2)
8∼15	Code	オペレーションコードで Autokey のメッセージが何であるかを示す
16∼31	Length	拡張フィールド長

表 10.2: OPCODE の構成

OPCODE 部の次に来るのがアソシエーション ID(32 ビット長) で, ここはふつう 0 である. 以降は Autokey メッセージによってフィールドの内容は異なり, 詳しくは IETF Draft[23] を読んで頂きたい.

10.7 相手認証と同定スキーム

Autokey では, 大まかに 5 つの認証機能をベースにしている. ここでは 5 つの認証について簡単に説明する.

1 つ目は, グループ鍵にプライベート証明書 (Private Certificate: PC) を使う認証で, 証明書は安全な手段でグループ内の全てのメンバーにコピーされる. この方法は, プライベート証明書が安全である限り, 暗号としては強力であるが, 鍵や証明書をたくさんのユーザに配布しなければならないため, 配布や更新が面倒である (小規模向け).

2 つ目の認証方法としては, 証明書の信頼の連鎖を利用した認証方法で, 証明書は信頼されるホスト (通常, ストラタム 1 サーバや公的な認証局) により近い発行者によって署名される (信頼されるホストは自己署名). この方法を信頼される証明書 (Trusted Certificate: TC)

による認証と呼ばれる. この方法が Autokey のデフォルトの認証である.

残りの 3 つの手法はオプションとしてゼロ知識証明をベースにした同定スキーム (Identity Schemes) を使って相手の認証が行われる. ゼロ知識証明の特徴は通信路上にパスワードが流れないという特徴がある (簡単にいえば, 合言葉のようなもので通信相手が正しい相手かを決定する). この同定スキームには IFF (Identity Friendly or Foe) と呼ばれる Schnorr アルゴリズムを改良したアルゴリズム, Guillou-Quisquater(GQ) アルゴリズム, そして改良版 Mu-Varadharajan(MV) アルゴリズムが使われる.

10.8 ntp-keygen コマンド

10.8.1 コマンド実行前の準備

NTPv4 での認証や同定で使われる鍵 (秘密鍵, 公開鍵) の生成は `ntp-keygen` コマンドにより行うことができる. また, MD5 の対称鍵も生成する機能が付加されている. `ntp-keygen` コマンドを利用できるようにするには, インストール時に OpenSSL ライブラリを NTP をコンパイルする前にインストールしておく必要がある (5.3 節, 58 ページ参照).

次に, `ntp-keygen` コマンドを実行するには, 暗号鍵作成の種となる疑似乱数ファイルが必要となるので, `openssl` コマンドでファイルを作成しておく必要がある. 疑似乱数ファイルが無いと, 下記のように表示され実行されない.

```
# ntp-keygen
Using OpenSSL version 90702f
RAND_load_file /root/.rnd not found or empty
```

疑似乱数ファイルは鍵の生成に重要な役割を担うため, 鍵を更新する度に変更することが望ましい. n バイトの乱数ファイルを作成したければ, コマンドの後に `rand` サブコマンドを付け, "`openssl rand` n" と実行する. ファイルはバイナリ形式で標準出力される. 例えば, 2048 バイトのファイルであれば, 下記のように実行する.

```
# openssl rand 2048 > .rnd
```

なお, `ntp-keygen` コマンドはデフォルトでホームディレクトリの下にある `.rnd` 名の疑似乱数ファイルを読みにいく. このファイル名を変更したければ, 環境変数`$RANDFILE` を変更しておく.

10.8.2 対称鍵の作成

ntp-keygen コマンドは MD5 対称鍵の作成もサポートされている. -M オプションを付けて実行すると, ntpkey_MD5key_*hostname*.*filestamp* というファイルに, ntp.keys ファイル形式 (7.6.2 節, 82 ページ参照) で 16 個の MD5 鍵が入ったファイルが作成される.

10.8.3 ntp-keygen の基本的な使用法

ntp-keygen が作成されるファイルは PEM エンコード化された印字可能な ASCII 形式で, 電子メールで証明書の送付が行える. デフォルトではファイルは暗号化されない. "-p *password*" オプションを指定することで, パスワードで暗号化される. "-q *password*" でパスワードを使って読むことができる (パスワードを入力しないと読めない).

ntp-keygen が作成するファイル名は一定のルールがあり, "ntpkey_" を頭に, 最後には " _*hostname*.*filestamp*" が付く. *hostname* は鍵の作成が行われたホスト名 (FQDN) で, gethostbyname() が返す文字列が入る. *filestamp* はファイルが生成された NTP 秒 (数字) が入る. このようにしておくことで, それぞれのファイルがユニークな名前を持つことができる. "*rm ntpkey**" で全ての鍵を削除できるし, "*rm *filestamp*" で特定の時間に生成されたファイルを削除できる.

鍵ファイルは全てインストール時に指定した鍵のディレクトリにインストールしなければならないが (デフォルトでは/usr/local/etc), ntp-keygen が生成するファイルはカレントディレクトリに作られるので, root になって/usr/local/etc に移って, コマンドを実行する. もちろん, 鍵の管理は十分注意を払うべきで, 特に秘密鍵はシステム管理者以外は読み書きできないよう設定しておく必要がある.

10.8.4 ntp-keygen コマンドのオプション

ntp-keygen コマンドのコマンド行オプションを説明する.

オプション	説明
-c *algorithm*	メッセージダイジェストと署名のアルゴリズム (*algorithm*) を指定する. 指定できるアルゴリズム名は, RSA-MD2, RSA-MD5, RSA-SHA, RSA-SHA1, RSA-MDC2, RSA-RIPEMD160, DSA-SHA, DSA-SHA1 である (デフォルトは RSA-MD5).
-d	デバッグモードで実行する.
-e	IFF のクライアント鍵を標準出力に書き込む.
-G	GQ スキームのパラメータと鍵を生成する.

オプション	説明
-g	既存のパラメータを使って鍵を生成する. パラメータが無いとこのオプションでは生成できない.
-H	既存のホスト鍵を破棄して, 新しい鍵を生成する.
-I	IFF スキームのパラメータを生成する.
-i *subject*	サブジェクト名を *subject* に設定する. サブジェクトフィールドはホスト鍵と署名ファイルの名前に使われる.
-m	証明書の鍵サイズを指定する (512~2048)
-M	MD5 鍵ファイルを生成する.
-P	プライベート証明書を生成する. デフォルトではパブリック証明書になる.
-p *password*	パスワード (*password*) と DES-CBC アルゴリズムを使ってプライベートファイルを暗号化する.
-q *password*	*password* を使ってファイルを読む.
-S RSA or DSA	新しい署名鍵を指定されたアルゴリズム (RSA あるいは DSA) で生成する.
-s *name*	署名者 (issue) の名前を設定する. 証明書の issuer フィールドと同定スキームのファイル名に使われる.
-T	信頼できる証明書を生成する. デフォルトは信頼できない証明を生成するようになっている.
-V *nkeys*	MV スキームのパラメータと鍵を生成する.

10.9 Autokey の設定

NTP ソフトウェアでの Autokey の設定を説明する. Autokey の設定はとても簡単である. 鍵や証明書の生成も NTP パッケージの中にある `ntp-keygen` コマンドを使用することで簡単に行うことができる.

10.9.1 ntp.conf の設定

`ntp.conf` の設定は共通することとして, `crypto` コマンドを追加するだけで良い. これで Autokey が使えるようになる. Autokey を使いたい場合, クライアント側で `key` の代わりに `autokey` を使えば良い.

サーバ側では,

```
server 127.127.22.1 minpoll 4
fudge  127.127.22.1
crypto randfile /var/ntp/.rnd
```

のように設定する. crypto コマンドにはオプションとして, シード乱数ファイル (randfile) や鍵・証明書ファイルを指定できる.

クライアント側では, server あるいは peer コマンドでサーバを指定する際に autokey を加えるだけである.

```
server 10.10.1.20 autokey
crypto
```

ブロードキャスト・マルチキャストやメニーキャストも同様に autokey を加えるだけである.

```
broadcast 224.0.1.1 autokey
manycastclient 239.1.1.1 autokey maxpoll 12 ttl 10
```

驚くほど簡単である.

10.9.2 PC スキーム

PC/TC スキームの場合を説明する. まず, サーバ側で, ntp-keygen コマンドを実行し, ホスト鍵と自己署名の証明書を作成する. デフォルトでは RSA を使ったホスト鍵と RSA-MD5 を使った証明書が生成される. このアルゴリズムを変更することも可能で, RSA の代わりに DSA を, MD5 の代わりに SHA を使う.

鍵と証明書の生成方法は, NTP ストラタム 1 サーバで, ntp-keygen に -P オプションを付けて, 鍵と証明書を作成する. この時, -p オプションを使って, ファイルにパスワードを付けておくことが可能である.

```
# ntp-keygen -P -p 1234
Using OpenSSL version 90607f
Random seed file /root/.rnd 1024 bytes
Generating RSA keys (512 bits)...
RSA 0 10 20     1 11 24                          3 1 2
Generating new host file and link
ntpkey_host_alice->ntpkey_RSAkey_alice.3287978014
Using host key as sign key
Generating certificate RSA-MD5
X509v3 Basic Constraints: critical,CA:TRUE
X509v3 Key Usage: digitalSignature,keyCertSign
X509v3 Extended Key Usage: private
Generating new cert file and link
ntpkey_cert_alice->ntpkey_RSA-MD5cert_alice.3287978014
```

この時, “X509v3 Extended Key Usage” に注意して欲しい. ここが private になっていれば, PC スキームのプライベートな証明書を作成したことになる. -P オプションがないとパブリックな証明書が作成される.

デフォルトで作成される鍵や証明書のサイズは 512 ビットである. このサイズを変更したければ, -m オプションを使用する. 但し, サイズが大きくなると処理に時間がかかるので, 非力なマシンの場合は注意が必要である. また, ホスト鍵を更新する場合は-H オプションを付ければ, ホスト鍵が更新できる.

鍵と証明書を作成したら, 安全な方法で NTP のグループ内のホストに送り, 自サイトの鍵名や証明書名にシンボリックリンクする.

```
# ln -s ntpkey_RSA-MD5cert_alice.3288000340 ntpkey_cert_bob
# ln -s ntpkey_RSAkey_alice.3288000340 ntpkey_host_bob
```

10.9.3 TC スキーム

TC スキームは PC スキームの証明書がプライベートではなく, パブリックであるというだけの違いである. 何もオプションを付けなければ, パブリックな証明書が作成される.

```
# ntp-keygen -p 1234
Using OpenSSL version 90607f
Random seed file /root/.rnd 1024 bytes
Generating RSA keys (512 bits)...
RSA 0 10 20     1 11 24                          3 1 2
Generating new host file and link
ntpkey_host_alice->ntpkey_RSAkey_alice.3287982294
Using host key as sign key
Generating certificate RSA-MD5
X509v3 Basic Constraints: critical,CA:TRUE
X509v3 Key Usage: digitalSignature,keyCertSign
Generating new cert file and link
ntpkey_cert_alice->ntpkey_RSA-MD5cert_alice.3287982294
```

作成された鍵と証明書はPCスキームと同様に, 安全な方法でNTPのグループ内のホストに送り, 自サイトの鍵名や証明書名にシンボリックリンクする.

10.9.4 IFFスキーム

IFFスキームでは, ホスト鍵, 証明書ファイルに加えてIFF用のパラメータファイルと鍵を作成する (鍵とパラメータファイルはひとまとめのPEM形式のファイルに収められる). そのため, まずNTPストラタム1サーバで, -Iオプションを加えて各ファイルを生成する. IFF, GQ及びMVスキームは信頼のある認証局 (第三者) を前提としているので, -Tオプションを付け, Trustedな証明書であることを示す必要がある.

```
# ntp-keygen -T -I -p 1234
```

このようにすると, ホスト鍵と証明書に加え, IFFパラメータファイルが生成される. ここで, -eオプションを付けてntp-keygenを実行すると, IFFのクライアント鍵を取り出すことができ, 鍵は標準出力に書き込まれる. クライアント機能のみを使うようなホストはこの鍵があれば良い (パラメータファイルは不要である).

```
# ntp-keygen -e
```

同定スキームを使う場合は, 相手の認証はIFFスキームが行うため, ホスト鍵や証明書は個々のサーバで作り (但し, -Tオプションを付けない), IFFパラメータファイルを共有するだけで良い. ストラタム1で作成したIFFのパラメータファイルを下位のストラタムに安

全な手段を用いてコピーをし, 参照するサーバの名前を付けておく. 実際には, クライアントでは次のファイルを作成する.

```
# ntp-keygen -p 1234
# ln -s ntpkey_IFFpar_alice.3287982294 ntpkey_iff_alice
```

10.9.5 GQ スキーム

GQ スキームの設定は, IFF と良く似ているが, クライアントでの扱いが少し変わっている. まず, ストラタム 1 サーバで, GQ 用のパラメータファイルを作成する. そのため, -G オプションを加えてホスト鍵と証明書を生成する.

```
# ntp-keygen -T -G -p 1234
```

これでホスト鍵と証明書に加え, GQ パラメータファイルが生成される. 既存のパラメータファイルを使って, GQ 同定スキームの鍵を生成するには, -g オプションを付けて ntp-keygen を実行する.

クライアントにはパラメータファイルを安全な手段で送り, そのファイルにサーバと自分の名前を付けておく. 実際には, クライアントでは次のファイルを作成する.

```
# ntp-keygen -p 1234
# ln -s ntpkey_GQpar_alice.3287818394 ntpkey_gq_alice
# ln -s ntpkey_GQpar_alice.3287818394 ntpkey_gq_bob
```

10.9.6 MV スキーム

MV スキームの設定は, パラメータファイルと鍵の生成を信頼できる認証局に任せる点で, IFF や GQ スキームとは異なっている. まず, ホスト鍵と証明書は各ホストで作成し, MV スキーム用のパラメータと鍵を認証局で作成する. ファイルの作成は-V オプションを付けて, ntp-keygen を実行する. この時, 無効にできるクライアントの数 n を指定する. n の値は, 通常 5 で実行し, その数より少ないクライアント鍵 $(0 < d < n)$ が生成される.

```
# ntp-keygen -V 5 -p 1234
```

そして, パラメータファイルをストラタム 1 のサーバに, クライアント鍵をストラタム 1 を参照するサーバに安全な手段を用いてコピーする. サーバでは,

```
# ntp-keygen -p 1234
# ln -s ntpkey_MVpar_trish.3287833122 ntpkey_mv_alice
```

クライアントでは,

```
# ntp-keygen -p 1234
# ln -s ntpkey_MVpar_trish.3287833122 ntpkey_mv_alice
# ln -s ntpkey_MVkey1_trish.3287833122 ntpkey_mv_bob
```

のように設定する.

注意： ntp-keygen のマニュアルでは, クライアントは MV のクライアント鍵を ntpkey_mvkey_bob にリンクする, と記述されているが, ソースコードを見る限り, ntpkey_mvkey_という記述は無いので, 実装途中なのか, マニュアルが違っているのか不明である.

第11章　リファレンス時計を使う

11.1　リファレンス時計

11.1.1　リファレンス時計について

ストラタム 1 の NTP サーバを立ち上げるには, UTC と高い精度で同期する時計 (リファレンス時計, 第 3 章参照) から時刻情報を直接獲得する必要がある.

リファレンス時計として使えるものには工業用原子時計, GPS 受信機, 電波時計 (標準電波受信機), CDMA 受信機, モデムアクセスによる時刻配信サービス (NIST や JJY) などがあるが, この中でもっともポピュラーなものは GPS 受信機と電波時計[1]である. GPS 受信機はナビゲーションシステムや建築などでパソコンと一緒に利用されるため, コンピュータとの接続が容易である. 電波時計は長波ラジオのようなもので, コンピュータに接続可能な受信機が安価に手に入る.

リファレンス時計の多くはコンピュータとの接続にシリアルないしはパラレルインタフェースを使っているが, 最近の GPS 受信機は小型化が進み, ノートパソコンや PDA との接続が容易な PCMCIA やメモリカード IO タイプのものもある.

NTP で利用する場合はシリアル/パラレルインタフェースの受信機, あるいはドライバが提供されている PC カードの製品を選んだ方が良い.

11.1.2　リファレンス時計の選び方

ntpd にはリファレンス時計の装置から時刻を読むためのドライバソフトが NTP ソース内にタイプ別に `refclock_`*FOO*`.c` というファイル名で用意されている. もし, ドライバが用意されていなくても, ドライバの書き方が公表されており, それほど難しくはない[2].

リファレンス時計選びは NTP サーバを設置しようとする環境に依存する. GPS の場合は屋外の, しかも全天が見渡せる場所にアンテナを上げなければならない. また, 電波時計でも障害物が少なく長波がきっちり受信できる場所でなければならない.

---

[1] 長波ラジオのようなものと考えれば良い.

[2] `http://www.eecis.udel.edu/ mills/ntp/html/howto.html`

自分でドライバソフトを書きたく無い人は, NTP ソフトウェアの内蔵ドライバ, もしくはベンダがドライバを提供している受信機を選択する必要がある. 最近は NTP で使う事を目的とした受信機がかなりの数売られているので, その中から選択するのが確実だろう. そして, NTP サーバとなるコンピュータも受信機との接続用にシリアルまたはパラレルインタフェースが搭載されているものを選択する必要がある. もし, USB しか搭載していないコンピュータを選択してしまったら, USB シリアルアダプタを使用する.

精度面でいえば, GPS や CDMA 受信機には 1PPS と呼ばれる正確なパルスを発振する機能を有するものがある. 通常, シリアルまたはパラレルインタフェースを通じて, 時刻情報を読み取るが, ビット列を読んで何時何分何秒かを解釈する処理にある程度の時間を要するため, より精度の高い時刻を得たい場合は正確なパルス信号を基にコンピュータの時刻を合わせる方がよい。GPS 受信機が搭載する 1PPS インタフェースは同軸ケーブルインタフェースが多いが, 同軸の場合はコンピュータのシリアルインタフェースに接続するための変換器 (Gadget box) が必要になる[3]. しかし, 最近は最初から NTP で使用する目的のために, 1PPS をシリアルインタフェースのキャリアディテクト (CD) 信号として出力する受信機も売られているので, こういった製品を選択した方が扱い易い (図 11.1).

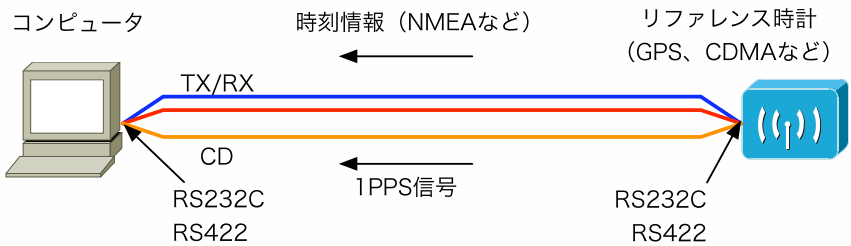

図 11.1: コンピュータとリファレンス時計の接続

11.1.3 ntpd はどのように時刻情報を読み取るか

シリアルまたはパラレルインタフェースで接続されたリファレンス時計から出力される時刻情報へのアクセス方法は工夫されている. ntpd は, リファレンス時計へのアクセスを 127.127.x.x のループバックアドレスを使って, 普通の NTP サーバと同じように `server` コマンドを使って設定することで, 他のサーバと同じように扱うことができる. すなわち, リファレンス時計は NTP のサーバアーキテクチャの中ではストラタム 0 という考え方になる.

[3] `http://www.eecis.udel.edu/ mills/ntp/html/pps.html`

リファレンス時計はタイプ別に “127.127.*t*.*u*” というループバックアドレスを使用する. *t* はクロックのタイプを表し, *u* はユニット番号を表す. ユニット番号は 0〜3 までの数字で同じタイプの受信機を区別するための固有番号で, 後でわかりやすいようにシリアルポート番号やバスアダプタ番号を付ける場合が多い. 例えば, Trimble Acutime2000 と呼ばれる GPS 受信機をシリアルポート COM 0 に接続して使用する場合, これはタイプが 29 でユニット番号が 0 となり, 127.127.29.0 でアクセスする. COM 2 に同じ受信機を追加したら, 127.127.29.2 になるという具合である.

11.2 ntpd の設定

リファレンス時計へのアクセスは `server` コマンドで可能と説明したが, ネットワーク上の NTP サーバを指定する場合とは少し設定が異なる (ネットワーク部分やセキュリティ機能を設定する項目が不要). また, リファレンス時計から得た時刻のオフセット等 (ファッジ係数) を設定するため, `fudge` と呼ばれるコマンドが用意されている.

11.2.1 server コマンド

`server` コマンドで指定するのは, リファレンス時計のループバックアドレス (127.127.*t*.*u*) と, 以下のオプションを指定できる.

オプション	説明
`prefer`	リファレンス時計を優先することを示す.
`mode` *int*	ドライバで固有に解釈されるモード番号を指定する.
`minpoll` *int*	
`maxpoll` *int*	リファレンス時計へのポーリング間隔の最小値と最大値を 2 の冪乗秒数で指定する. デフォルトでは `minpoll` と `maxpoll` は 6(64 秒) である. モデムを使ってアクセスするリファレンス時計の場合はこれがダイアル間隔になる. 指定できる範囲は 4(16 秒) から 17(36.4 時間) までとなっている.

11.2.2 fudge コマンド

`fudge` コマンドはリファレンス時計のファッジ係数 (Fudge Factor) を設定するために使われるコマンドで, ファッジ係数を指定しなくても必ずリファレンス時計の場合, `server` コマンドの直後に指定しなければならない (オプションを指定しないということはデフォルトの PLL/FLL ループを使うという意味でもある).

```
server 127.127.20.0
fudge 127.127.20.0
```

指定できるファッジ係数は以下のとおりである.

オプション	説明
`time1` *sec*	ドライバが得た時刻に対して, 時間オフセットを増減する (固定小数点方式で表される秒数で指定).
`time2` *secs*	ドライバが持っている時間オフセットで, ドライバによっては使われない場合もある.
`stratum` *int*	ドライバに割り当てるストラタム数を指定する. ドライバの中で指定されているが, 特別な理由が無い限り変更しない (ふつうは 0 である).
`refid` *string*	ドライバのリファレンス識別子を 1〜4 つの ASCII 文字で指定する. ふつうはドライバの中で指定されている.
`mode` *int*	ドライバで固有に解釈されるモード番号を指定する (`server` コマンドの `mode` オプションと同じ).
`flag1`〜`flag4`	クロックドライバをカスタマイズするために使われる 4 つのフラグがあり, 0 か 1 で指定する.

11.2.3 補正値の算出

リファレンス時計が刻む時刻情報はコンピュータがその時間と認識するまでに時間差が生ずる. 時間差の原因は,

- シリアルポートまたはオペレーティングシステムの処理待ち時間
- シリアルケーブルが長い
- 受信機の内部遅延

などが考えられる. `time1` で挿入する時間オフセット値は定常的に挿入されるオフセットなので, システム導入時の試験運転の段階でオフセット値をモニターして, 定常的に現れる時間差を測ることが重要である.

二つ以上のラジオクロックまたは PPS 信号があれば, `ntpd` のキャリブレーション機能を使って補正値を出すことができる. まず, `enable calibrate` コマンドあるいは `ntpdc` でこの機能を有効にする. キャリブレーション機能は `time1` の値を調整する機能で, 一旦有効に

すると収束するまでに 1 時間程度走らせる必要がある. そして, 自動キャリブレーションで得られた補正値は ntpq コマンドの clockvar コマンドで表示されるので, この fudgetime1 値をファッジ係数とする. 補正値が得られたら, 必ずキャリブレーション機能は無効化しておく.

11.3 PPS (Pulse-Per-Second)

ntpd は単純にリファレンス時計から送られて来る時刻情報 (文字列) を得るタイミングで時刻調整を行うが, 文字列は必ずしも正確なタイミングで出力されず, “ずれ” が生じる (数ミリ秒の誤差があるといわれている). ストラタム 1 として運用するには, この誤差は無視できない. そこで, このずれを小さくするため, GPS の場合は衛星から, CDMA の場合は基地局から発せられる PPS(Pulse-Per-Second) 信号に同期させる方法が考えられた.

PPS 信号は非常に正確に 1 秒を刻むパルスで, 基準周波数や正確なパルスの生成に使われるものだが, コンピュータにとっては時刻情報よりもパルス信号をハンドリングする方が扱い易く, かつ正確である. そのため, PPS を用いる事で数マイクロ秒の精度で時刻を合わせる事が可能となる.

11.3.1 PPS API

Linux や BSD といった Unix 系のオペレーティングシステムが PPS 信号を扱うための API(Application Program Interface) が, RFC 2783[15] として仕様化されている. また, RFC 1589[10] では Unix オペレーティングシステムのカーネルが高い精度で時刻をハンドリングする方法が論じられており[4], この中に外部クロック (PPS や IRIG) を用いる方法が説明されている. この仕様をベースに PPS のハンドリング機能を具体的に実装したと考えれば良い.

PPS のパルス信号をコンピュータに取り込むには, PC やワークステーションが標準的に搭載しているシリアルインタフェースの DCD(Data Carrier Detect) ピンが利用される.

PPS API の機能としては, PPS の検出すると hardpps() と呼ばれるカーネルルーチンが使われ, hardpps() ルーチンは PPS 信号のパルスが検出される度に, CPU クロックオシレータを PPS 信号に合わせる機能を持つ. この PPS API を実装しているオペレーティングシステムとしては, 次のようなものがある.

- Linux (PPSkit パッチが必要)
- FreeBSD
- NetBSD (まじめに実装されていないらしい)

[4] これらは SunOS 4 のカーネルに実装された.

11.3.2 NTP ナノカーネル

PPS の精度がいくら高くても, コンピュータ側がそれと同じ精度でパルスをハンドリングできなければ, 宝の持ち腐れになってしまう. 普通のコンピュータが外部の信号をハンドリングできる精度は, コンピュータが搭載しているオシレータの周波数にもよるが, 約 1 マイクロ秒 (0.838 マイクロ秒) である[5]. この精度を 1 ナノ秒まで高めたのが NTP ナノカーネルである.

NTP ナノカーネルはナノ秒を取り扱うために, カーネルのレイテンシ (応答時間) の低減や, カーネルの時間関係の構造体の拡張などを行っている. ふつうナノカーネルは PPS と一緒に用いられるため, カーネルで PPS をハンドリングできるように再構築しておけば, ナノカーネルが使えるようになる[6].

11.3.3 PPS をハンドリングできるオペレーティングシステム

LinuxPPS

ミリ秒単位での時刻合わせは通常の Linux カーネルで問題無いが, PPS 信号を用いてマイクロ秒単位で時刻を合わせる場合, PPS API の機能がカーネルに必要となる. 2.4 系のカーネルの場合, PPSKit と呼ばれる PPS API と NTP ナノカーネルパッチが同梱されたキットが下記から入手できる. なお, 最新版 PPSKit-2.1.7 はカーネル 2.4.33.2 に対応している.

```
ftp://ftp.kernel.org/pub/linux/daemons/ntp/PPS/
```

PPSKit は, カーネルにパッチを適用後, カーネルの設定で PPS を有効にするため, 以下の 3 項目を設定し, カーネルの再構築を行う.

```
CONFIG_EXPERIMENTAL=y
CONFIG_NTP=y (NTP kernel support)
CONFIG_NTP_PPS=y (NTP PPS support)
CONFIG_NTP_PPS_SERIAL=y (NTP PPS support on serial port)
CONFIG_SERIAL=y
```

[5] コンピュータは 14.318MHz 水晶発信子を使って, この基準クロックを PLL 回路で変換して使っている. 実際にタイマ割り込みは 12 で割った 1.193MHz に固定されており, この割り込みタイミングがコンピュータの最小時間粒度となる.

[6] もちろん, NTP ナノカーネルで無くても PPS 機能を利用できる.

カーネル 2.6 系の LinuxPPS は Rodolfo Giometti 氏によって開発されており, 下記から入手できる.

```
http://ftp.enneenne.com/pub/misc/linuxpps/patches
```

また, PPS カーネルでなくても PPS による時刻源を許容するための SHM PPS ドライバが, David J. Schwartz 氏によって開発されている. このドライバは下記から入手できる.

FreeBSD

FreeBSD では, カーネル 3.5 あたりから PPS API はサポートされた. デフォルトでは使えるようになっていないため, 以下のオプションを有効にして, カーネルを再構築する必要がある.

```
options PPS_SYNC
```

OpenSolaris

OpenSolaris に NTP カーネルと PPS API の実装をしようとするプロジェクト[7]がある.

- Solaris に ntp 4.2.2 を組み込む.
- Solaris に NTP ナノカーネルコードを組み込む.
- PPS API(RFC 2783) を実装する.
- PARSE STREAMS モジュールを組み込む.
- 大きな FIFO を使うことでシリアルインタフェースの低遅延モードを実装する.
- TOD 同期モデルの調査と調整.

11.3.4 NTP の設定

新しい NTP であれば, PPSKit に同梱されているパッチは不要で, PPS 機能を使えるように設定してコンパイルする. PPS を発信するデバイス (GPS 受信機) をシリアル接続 (`/dev/gps0`) し, `/etc/ntp.conf` を次のように設定する.

[7] `http://www.opensolaris.org/os/project/ntp/`

```
pps /dev/gps0 assert hardpps
server 127.127.20.1 prefer
fudge 127.127.20.1 flag3 1 refid GPS
```

11.4 GPS

11.4.1 GPS 受信機

NTP で利用できる GPS 受信機はシリアルインタフェースを備え, NTP ソフトウェアが解釈できるフォーマットで時刻情報を送り出し, 可能であればシリアルインタフェースで PPS 信号を出力できるものが良い. また, GPS はふつう位置測位を行うために 4 つの GPS 衛星を捉えなければ情報が出力されないが, NTP では位置情報は使わないので, 1 つの衛星を捕捉するだけで時刻情報が出力されるタイプが便利である.

11.4.2 GPS アンテナの設置

GPS の一番厄介な点は見晴らしの良い場所にアンテナを設置しなければならない点にある. GPS 受信機は位置を測位するのが仕事であるため, 衛星を 4 つ捕捉した状態でないと情報が出力されないものが多い. 最近の GPS 受信機は 1 つの衛星を捕捉すれば, 時刻情報や PPS 信号を出力するタイプもあるが, 見晴らしが良い場所でなければ, 衛星が視界から消えてしまい, 情報が出力されなくなる場合も考えらる.

通常, GPS アンテナはテレビのアンテナのように建物屋上に上げることが一般的である. 受信機とアンテナの間は RG-58 もしくは RG-59 と呼ばれる同軸ケーブルが使われる. なかにはイーサネットで使われるツイストペアケーブルを使うタイプもある. 通常, 20~30m 程度のケーブルが受信機と一緒に同梱されているが, 大きなビルでは, ケーブル長が 100m 以上になる場合がある. この場合, 信号を増幅するためのブースタや UP/DOWN コンバータなどが用いる必要がある (図 11.2).

GPS アンテナの多くは “お椀型” の形状をしており, アンテナの中に受信機を内臓したタイプのものもある (例: Garmin, Trimble Acutime 2000). アンテナは支柱をねじ込めるための穴が空いており, その穴に固定された支柱にねじ込めるか, GPS 受信機セットと一緒に提供される短い支柱にねじ込め, その支柱ごと別の支柱に固定する方法がとられる.

これまで屋外用のアンテナを説明したが, GPS アンテナには窓ガラスに吸着させる室内用のアンテナもある. 室内アンテナを使う場合はなるべく南に, 無理であれば東か西の空を広く見渡せる窓に取り付ける. 電波を透過しないガラスや金属膜を持つ熱線遮蔽フィルム

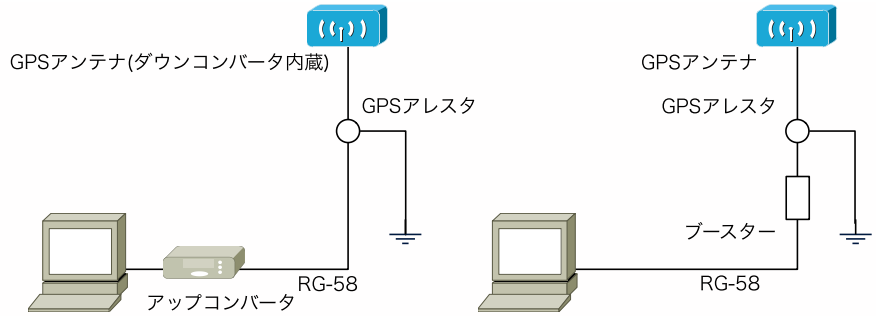

図 11.2: GPS アンテナの上げ方

を貼ったガラスの内側では使用できないので, 注意が必要である. 衛星をキャッチできなくなっても内蔵のオシレータで自走可能な NTP サーバであれば, 室内アンテナでも良いかもしれないが, 確実に GPS 信号を受信したい場合は屋外アンテナになる.

11.4.3 タイムコードとプロトコル

受信機がコンピュータに時刻を出力するためのタイムコード[8]やプロトコルには様々な種類がある.

NTP で良く使われるものに, NMEA-0183(単に NMEA と呼ぶ場合が多い) がある. NMEA は米国海洋電子機器協会 (National Marine Electronics Association)[9]が定めた規格で, 受信機とナビゲーション機器の通信に使用されるプロトコルである. その中でも NMEA-0183 は GPS 受信とナビゲーション機器間のシリアルポートで利用される通信規格である. NMEA-0183 は全ての文字が ASCII テキストで送られ, 出力できるデータは受信機の種類によって異なる. 一方, ベンダ独自のもので有名なものには Trimble 社の二つのプロトコル, ASCII 形式の TAIP(Trimble ASCII Interface Protocol) とバイナリ形式の TSIP(Trimble Standard Interface Protocol) がある.

参考までに, シリアルタイムコードとして有名なものに IRIG-B がある. IRIG-B 信号[10]

[8] タイムコードは 1950 年頃から地球物理, 宇宙, 軍事等の分野で, 観測データに日時を付けて記録するために考案されたもので, それぞれのグループごとに独自の形式が使用されていた。バラバラに考案されたため, 1960 年初めには約 40 種類にも増え, 米国内ではこれらグループ間でコード形式の標準化が進められた. その結果, SD(Serial Decimal) タイムコード, BCD(Binary Cord Decimal) タイムコード及び PB(Parallel Grouped Binary) タイムコードの 3 系統に概ね整理された. これらの代表的なものとして, ミサイル開発グループの IRIG (Interrange Instrumentation Group) コード, アメリカ航空宇宙局の NASA コード等の直列 BCD コードと, 両グループで標準化を提案している PB コードがよく知られている.

[9] http://www.nmea.org/

[10] http://www.phys.washington.edu/ berns/SUPERK/GPS/irigcode.html

は信号線の中に異なる時間幅のパルスが 1 秒間に 100 パルス伝送され (これが 1 フレームになる). このパルス幅の変化を捉えることで, 時刻符号を検出する.

11.4.4 その他の機材

GPS 信号 (L バンド信号) の分配器

屋外にアンテナを上げるのは大変な作業となるので, 一つのアンテナに複数の GPS 受信機を共有して使用したい場合や, 後で GPS 受信機を追加する場合のために, GPS 信号 (L バンド信号) の分配器が使われる. GPS 信号の分配器は電源が不要のものから, 電源を必要とするものまで様々あるが, インピーダンスの関係で分配器を挟むと GPS 受信機がうまく動かなくなる場合があるので, よくベンダーに確認してから購入することをお薦めする.

GPS アレスタ (避雷器)

GPS アレスタは GPS アンテナ用の避雷器で, GPS 受信機を外部からの雷サージや電磁パルス (EMP) から守る役割を負う. 誘導雷に対して有効だが, アンテナが直接雷を受けた場合はアンテナ自体が故障するため, 避雷針を併用する方ことをお薦めする. また, アレスタは良好な接地があって, 本来の働きをするので, 接地工事をしっかり行っておく必要がある.

但し, アンテナ側に受信機やブースターを内蔵しているタイプのものは, ケーブルから DC 電気を供給すると思われるため, 避雷器の挿入は難しいかも知れない. よく購入先と相談して頂きたい.

11.4.5 HP 社製 GPS 受信機 Z3801A

クアルコム社の CDMA 方式による携帯電話や WiMAX は基地局間の時刻を同期させる必要があるため, 基地局に高精度の GPS 受信機が設置されている. 米国の CDMA キャリアの間でよく使われていたのが HP 社製の Z3801A(図 11.3) で, 設備の交換時期にこの受信機が大量に放出され, かなりの台数が中古市場に流通している.

Z3801A の仕様上の精度は 10^{-9} だが, 短期の安定度は 10^{-11} ともいわれ, 中古市場ではこの受信機が数万程度で手に入る事もあり, NTP の時刻源として使う人もいる.

1. 標準では RS232C に 1PPS が出力されないので, RS232C の DCD ピンに 1PPS が出力されるよう少し改造を行う.

2. Z3801A は同じ HP 社の 58503A と似ているが, 下記の点で少し異なるため, Jeff Mock 氏 (`jeff@mock.com`) 作成のパッチをあて, `ntpd` を作り直す.

図 11.3: HP Z3801A

- RS232C が 7 ビットの odd パリティである.
- 通信速度が 19200bps である.

詳しくは以下の URL を参考にしていただきたい.

- RS232C の DCD ピンに 1PPS を出力させる改造方法
  - http://www.febo.com/time-freq/gps/z3801a/mods/
  - http://www.shoshin.co.jp/computer/lantro/html/z3801a.htm
- http://www.realhamradio.com/GPS_Frequency_Standard.htm
- Z3801A 用のパッチ: http://www.mock.com/ntp/ntp-4.1.72-patch.tgz

11.4.6 GPS と太陽フレア

太陽は 11 年周期で活動が活発化することが知られている. 現在, 11 年ぶりの極大期に向けて活発化しているといわれるが, 大規模な太陽フレアが起きると衛星に多大な影響を及ぼす. GPS も例外ではなく, 太陽フレアの後に発生する磁気嵐 (爆発に伴って強力なエックス線や高エネルギー粒子が放出される現象) によって衛星の故障や GPS 信号に誤差が発生する.

なお, 太陽活動が通信や放送といった生活に密着している今, NiCT では宇宙天気情報センターを 2006 年 5 月に開設し, 宇宙天気情報と一緒に GPS ケア情報[11]も提供している.

[11] http://swc.nict.go.jp/gps/

11.5 JJY

11.5.1 JJY の受信機

欧米では短波や長波を使った電波時計を時刻源にするケースが多い (ヨーロッパの CHU や, 米国の WWV など). 日本でも長波による電波時計の放送が 1999 年から始まり, これを時刻源にするための受信機がいくつか販売されている.

腕時計や目覚し時計に内臓されている電波時計は一日に数回〜24 回 (1 時間に 1 度) しか, 時間を合わせないため, ほとんどの時間帯は時計に内蔵されている温度補償型の水晶発振子を使用している. JJY に合わせた時点では時間は正しいのだが, 水晶時計化していまった段階で既に狂い始めていると考えて良い.

従って, NTP で使用する JJY 受信機は, このような電波時計ではなく毎分 JJY の時刻信号と合わせるタイプでなければならない。表 11.3 が NTP の使用に耐え得る市販されている JJY 受信機である. これらは数万から数十万円と一般の電波時計に比べかなり高価である. シーデックスの電波時計はアンテナと受信機が一緒になっており (屋外用は防雨型), 本体とはツイストペアケーブルで接続する (仕様では最大 500m). 日本通信機の JJY 受信機はルビジウムで自走する二次周波数標準機で JJY を標準周波数の較正用に使用している装置である.

メーカー	モデル	精度	PPS
セイコー[12]	LFR200	100ms 以下	無し
シーデックス[13]	JST2000 シリーズ	10ms 以下	無し
日本通信機[14]	7572B[15]	150μs rms	有り
エコー計測器[16]	LT-2000	2.5ms	無し
システムアーツ[17]	SA-320LW	記述無し	無し

表 11.3: 市販されている JJY の受信機

[11] http://www.seiko-p.co.jp/system_clock/sysc_lfr.html
[12] http://www.c-dex.co.jp/
[13] http://www.nitsuki.com/ja/products/jjy-gps/jjy-gps.html
[14] CRL と共同開発したもの.
[15] http://www.clock.co.jp/pdct/p_lt2000.html
[16] http://www.hi-ho.ne.jp/systemarts/clock/sa320.html

11.5.2 JJY のアンテナ

長波受信アンテナはループ型, ホイップ型, ロングワイヤなどがあるが, 小型であること, 設置が容易なことから一般にはホイップ型のアンテナが用いられる場合が多い. ホイップ型は受信信号の波長に対してアンテナエレメントの長さが短く, アースが必要になる.

アンテナにはプリアンプを内臓し, 長波をゲインするだけでなく中波や電源ノイズを減衰させ, 受信感度を補うタイプもある.

11.6 CDMA

CDMA のレシーバは, 米国の EndRun Technologies 社[18]が販売している. CDMA 受信機がビルトインされた NTP 専用サーバもあるが, CDMA 受信機のみのタイプも販売されている. Praecis Ct/Cf は, 受信機のみのタイプで, RS232C を通してコンピュータと接続でき, PPS 信号を出すこともできる. また, Praecis Cf はオシレータを内蔵することもできる.

但し, 日本では上下の信号周波数が他の国と異なるため, 海外の受信機をそのまま持って来ただけでは信号を受信できないので, 日本国内向けの受信機を購入して頂きたい.

[18] http://www.endruntechnologies.com/

第12章　NTPの管理(モニタリングとリモート管理)

本章では, ntpq と ntpdc コマンドを中心に ntpd デーモンのモニターとリモート管理の方法を説明する.

12.1　ntpq

ntpq コマンドは NTP デーモン ntpd をモニターするために使われるユーティリティプログラムで, NTPv3 の仕様 RFC1305 の Appendix B で定義されている NTP モード 6 制御メッセージ (図 12.1) を使用している.

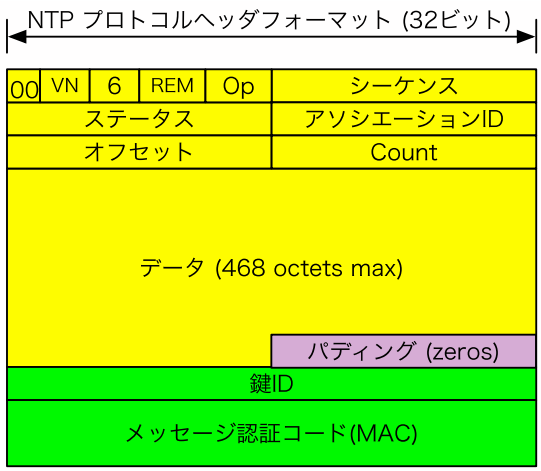

図 12.1: モード 6 制御メッセージ

ntpq コマンドは対話形式とコマンド行形式で利用し, コマンド行形式は次のとおりである.

```
ntpq [ -46inq ] [ -c command ] [ host ] [ ... ]
```

オプション	説明
-4	ホスト名を IPv4 アドレスで解決する
-6	ホスト名を IPv6 アドレスで解決する
-c	対話形式のコマンドを指定して実行する. 例えば, ntpq -c "hostname no"
-d	デバッグモードを有効にする
-i	対話形式モードで操作を行う
-n	ホスト名ではなく, IP アドレス形式で表示する
-p	NTP サーバのサマリ情報を表示する (これは対話形式の peers サブコマンドを実行したものと同じである)

12.1.1 ntpq の対話コマンド

引数無しに ntpq を実行すると対話モードになる (プロンプト ntp> と表示される). ここで, いくつかのコマンドが実行できるので, コマンドの全リストを得るため “help” とタイプしてみる.

```
hotta% ntpq
ntpq> help
Commands available:
addvars         associations    authenticate    cl              clearvars
clocklist       clockvar        cooked          cv              debug
delay           exit            help            host            hostnames
keyid           keytype         lassociations   lopeers         lpassociations
lpeers          mreadlist       mreadvar        mrl             mrv
ntpversion      opeers          passociations   passwd          peers
poll            pstatus         quit            raw             readlist
readvar         rl              rmvars          rv              showvars
timeout         version         writelist       writevar
```

対話モードでは, 内部コマンドや制御メッセージコマンドが用意されており, システム変数の追加・変更・削除ができる.

12.1.2 内部コマンド

ntpq で使用できる内部コマンドは不完全であっても, コマンド名が補完される. 例えば, help というコマンドは, he でも hel でも “help” と解釈される. 次の表が内部コマンドの一覧である.

コマンド	説明
? [*command_keyword*] help [*command_keyword*]	コマンドキーワードを入力せずに?あるいは help とタイプすると ntpq で利用できるコマンドの一覧を表示する. コマンドキーワードを入力すると, そのコマンドに関する情報を表示する
addvars *variable_name* [= *values*] [...] rmvars *variable_name* [...] clearvars	NTP モード 6 メッセージによって運ばれるシステム変数の追加・削除・クリアを行う. 現在設定されている内部システム変数は readlist コマンドで表示する事ができる
cooked	問い合わせコマンドの出力を人間が読解可能な形式で出力するようにする
debug more | less | off	デバッグモードの入切を行う
delay *milliseconds*	認証を必要とする要求に組み込まれるタイムスタンプに追加する時間間隔を指定する. 長遅延ネットワークまたは時間が同期化されていないコンピュータ間で信頼性のないサーバを再構成することができる. 引き数を指定しないでこのサブコマンドを入力した場合, 現行設定値が表示される.
host *hostname*	問い合わせを送るホスト (FQDN あるいは IP アドレス) を指定する
hostnames [yes | no]	yes と指定すると表示される情報はホスト名が表示され, no とするとホスト名で表示できても IP アドレスで表示される
keyid *keyid*	問い合わせ要求を認証する時に使用する鍵番号を指定する (サーバ側と一致する必要がある)
ntpversion 1 | 2 | 3 | 4	パケットに含まれる NTP バージョンを設定する (デフォルトは 3 である)[1]

[1]注意点として, モード 6 メッセージは NTP バージョン 1 には存在しない.

コマンド	説明
passwd	認証コンフィグ要求で使われるパスワードを入力する (エコーされない)
quit	ntpq を終了する
raw	問い合わせコマンドからの全出力をサーバから受信したままの形式で出力する. 非アスキー形式のデータはプリント可能な文字に変換される (読めるとか限らない).
timeout *milliseconds*	サーバへの問い合わせのタイムアウト値を指定する (デフォルトは約 5000 ミリ秒)

この中で, cooked と raw の違いは説明するよりも一見した方が早い. 例えば, pstatus コマンドでは, 次の違いがある.

- フォーマットが成形される
- IP アドレスを可能なものは DNS の逆引きを行って, FQDN で表示する
- ステータスコードの意味を表示する
- 時間を可読な方式で表示する (org, rec, xmit)

具体的には, cooked の場合, 次のように表示される.

```
ntpq> pstatus 5079
status=9424 reach, conf, sel_candidat, 2 events, event_reach,
srcadr=ntp3.jst.mfeed.ad.jp, srcport=123, dstadr=1.2.3.4,
dstport=123, leap=00, stratum=2, precision=-18, rootdelay=0.305,
rootdispersion=0.381, refid=mf-isdn4.mfeed.ad.jp, reach=377, unreach=0,
hmode=3, pmode=4, hpoll=10, ppoll=10, flash=00 ok, keyid=0,
offset=-0.779, delay=1.165, dispersion=14.520, jitter=0.059,
reftime=c3473657.25700000  Mon, Oct 27 2003 15:14:47.146,
org=c3473657.c148c000  Mon, Oct 27 2003 15:14:47.755,
rec=c3473657.c1a1fd15  Mon, Oct 27 2003 15:14:47.756,
xmt=c3473657.c1531550  Mon, Oct 27 2003 15:14:47.755,
filtdelay=     1.16    1.18    1.05    1.40    1.26    1.27    1.04    1.30,
filtoffset=   -0.78   -0.72   -0.71   -0.56   -0.39   -0.46   -0.48   -0.38,
filtdisp=      0.01   15.40   30.76   46.15   61.51   69.19   76.89   84.58
```

一方, raw にすると次のように表示される.

```
ntpq> pstatus 5079
status=0x9424,
srcadr=210.173.160.87, srcport=123, dstadr=1.2.3.4, dstport=123,
leap=0, stratum=2, precision=-18, rootdelay=0.305, rootdispersion=0.381,
refid=210.173.176.4, reach=0xff, unreach=0, hmode=3, pmode=4, hpoll=10,
ppoll=10, flash=0x0, keyid=0, offset=-0.779, delay=1.165,
dispersion=14.520, jitter=0.059, reftime=0xc3473657.25700000,
org=0xc3473657.c148c000, rec=0xc3473657.c1a1fd15,
xmt=0xc3473657.c1531550,
filtdelay= 1.16 1.18 1.05 1.40 1.26 1.27 1.04 1.30,
filtoffset= -0.78 -0.72 -0.71 -0.56 -0.39 -0.46 -0.48 -0.38,
filtdisp= 0.01 15.40 30.76 46.15 61.51 69.19 76.89 84.58
```

12.1.3 制御メッセージコマンド

各NTP サーバと各ピアとの間には16 ビットのアソシエーション識別子 (association identifier) が割り当てられる. NTP 制御メッセージを送る際に, どのピアかを判別するためにこの識別子を含めて送られる. 0 という ID は特殊な番号で, 変数がシステム変数である事を示す.

制御メッセージコマンドを実行すると, サーバに一つ以上のモード 6 メッセージが送信され, 応答が戻って来る.

associations

問い合わせたサーバのアソシエーション識別子とピアステータスのリストを取得して表示する.

```
ntpq> assoc
ind assID status  conf reach auth condition  last_event cnt
===========================================================
  1 25580  9024   yes   yes  none    reject   reachable  2
  2 25581  9424   yes   yes  none  candidat   reachable  2
  3 25582  9624   yes   yes  none  sys.peer   reachable  2
  4 25583  9424   yes   yes  none  candidat   reachable  2
  5 25584  9324   yes   yes  none   outlyer   reachable  2
  6 25585  9324   yes   yes  none   outlyer   reachable  2
  7 25586  9524   yes   yes  none  selected   reachable  2
  8 25587  9324   yes   yes  none   outlyer   reachable  2
  9 25588  9124   yes   yes  none falsetick   reachable  2
 10 25589  9324   yes   yes  none   outlyer   reachable  2
```

passociations

新しく問い合わせを行わず, キャッシュされているアソシエーションリストから情報を取り出して表示する.

lassociations

サーバが状態を保持している全てのアソシエーションについて、そのアソシエーション識別子とピア・ステータスのリストを取得して表示する. associations コマンドとの違いは, 単にこのコマンドはスペック外のクライアントのアソシエーションの状態を保持するサーバを対象としているという点だけである.

lpassociations

passociations 同様, キャッシュされている曖昧な情報を含むアソシエーションリストから取り出して表示する.

readvar assocID variable_name [= value] [...]

変数読み取り要求を送信するこで, 指定された変数の値を戻すようにサーバに要求する. アソシエーション ID を省略した場合, または 0 の場合は, 変数はシステム変数となる. それ以外の場合はピア変数となり, 戻される値は対応するピアの値になる. 変数リストを省略するとデータ無しの要求が送信され, サーバからはデフォルトの表示が戻される. なお, readvar は rv と短縮できる.

```
ntpq> readvar 25582 delay
status=9624 reach, conf, sel_sys.peer, 2 events, event_reach,
delay=1.419
```

writevar assocID variable_name [= value] [...]

readvar 要求と似ているが, 指定された変数を読み取るのではなく書き込む.

readlist [assocID]

内部的な変数リスト内の変数の値を戻すようにサーバに要求する. アソシエーション ID を省略した場合, または 0 の場合は変数はシステム変数であると見なされる. それ以外の場合はピア変数として扱われる. 内部の変数リストが空の場合はデータなしで要求が送信され, リモート・サーバからはデフォルトの表示が戻される. なお, readlist は rl と短縮できる.

```
ntpq> rl 25582
status=9624 reach, conf, sel_sys.peer, 2 events, event_reach,
srcadr=210.173.160.57, srcport=123, dstadr=1.2.3.4, dstport=123,
leap=00, stratum=2, precision=-18, rootdelay=0.641,
rootdispersion=0.610, refid=210.173.176.4, reach=377, unreach=0,
hmode=3, pmode=4, hpoll=10, ppoll=10, flash=00 ok, keyid=0,
offset=3.405, delay=1.419, dispersion=13.103, jitter=0.668,
reftime=c4134f6d.8c469000  Tue, Mar 30 2004 10:44:13.547,
org=c4134f81.e588c000  Tue, Mar 30 2004 10:44:33.896,
rec=c4134f81.e4d8127b  Tue, Mar 30 2004 10:44:33.893,
xmt=c4134f81.e478a6da  Tue, Mar 30 2004 10:44:33.892,
filtdelay=     1.42    1.59    2.03    1.35    1.32    1.66    1.40    1.30,
filtoffset=    3.41    2.74    1.86    1.89    1.76    1.35    1.12    0.97,
filtdisp=      0.01   15.35   30.74   38.42   46.10   53.80   61.48   65.32
```

writelist [assocID]

readlist 要求と似ているが, 読み取る代わりに内部変数リスト以外を書き込む.

mreadvar assocID assocID [variable_name [= value[...]

readvar コマンドと似ているが, 問い合わせが複数の (ゼロでない) アソシエーション ID のそれぞれについて実行される点が異なる。この範囲は直前の associations コマンドによってキャッシュされたアソシエーション・リストから決定される。なお, mreadvar は mrv と短縮できる.

mreadlist assocID assocID

readlist コマンドと似ているが, 問い合わせ複数の (ゼロでない) アソシエーション ID のそれぞれについて実行される点が異なる. この範囲は直前の associations コマンドによってキャッシュされたアソシエーション・リストから決定される。なお, mreadlist は mrl と短縮できる.

clockvar [assocID] [variable_name [= value [...]] [...]

サーバのクロック変数のリストを送るように要求する. この要求に対して, ラジオ・クロックなどの外部同期機構を備えたサーバから有効な応答が戻される. アソシエーション識別子を省略した場合, または 0 の場合の要求は system clock の変数に関する要求になり, 通常はクロックを備えたすべてのサーバから有効な応答が戻される. もし, サーバがクロックを疑似ピアとして扱っており, そのためにサーバに複数のクロックが同時に接続されてい

る可能性がある場合, 該当するピアのアソシエーション ID を指定することにより, そのクロックの変数が表示される. 変数リストの指定を省略すると, サーバからはデフォルトの変数の表示が返される.

```
ntpq> clockvar 25580
status=0000 clk_okay, last_clk_okay,
device="Undisciplined local clock", timecode=, poll=112, noreply=0,
badformat=0, baddata=0, fudgetime1=0.000, stratum=2, refid=76.67.76.0,
flags=0
```

peers

サーバのピアのリストが各ピアの状態の要約情報が表示される (“ntpq -p” と同じ). 要約情報にはピアのアドレス, 参照 ID(参照 ID が不明の場合は 0.0.0.0), ピアのストラタム, ポーリング間隔 (秒単位), ピアのタイプ (**l**ocal, **u**nicast, **m**ulticast or **b**roadcast) 及び到達可能性レジスタ (8 進数) が含まれる他、ピアの現在の推定遅延, 偏差, 及び分散が全てミリ秒単位で表示される. 更に, 左端の文字はタリーコード (12.1.4 節) と呼ばれる文字が付く.

```
ntpq> peer
     remote           refid      st t when poll reach   delay   offset  jitter
==============================================================================
 127.127.1.0     127.127.1.0      2 l    8   64  377    0.000    0.000   0.004
-210.173.160.27  210.173.160.56   2 u 1896 1024  376    1.311    2.884   0.779
*210.173.160.57  210.173.176.4    2 u  813 1024  377    1.419    3.405   0.668
+210.173.160.87  210.173.176.4    2 u  295 1024  377    1.187    3.583   0.279
x133.42.48.3     130.34.11.117    2 u  292 1024  377  258.271  120.219 113.751
+133.31.30.8     133.31.180.6     2 u   32 1024  377    1.435    3.202   0.437
#193.2.4.6       193.2.4.2        2 u  299 1024  377  316.125    8.037   0.306
-216.27.190.202  216.152.68.20    2 u   27 1024  377  134.024   -5.699   0.992
x130.88.202.49   193.63.105.18    2 u  875 1024  377  250.542  293.142  63.277
-132.246.168.148 132.246.168.3    2 u  876 1024  377  218.601   -4.303   1.860
 194.137.39.67   0.0.0.0         16 u    -  256    0    0.000    0.000 4000.00
```

ポーリング間隔は ntpd 起動時は minpoll で指定された間隔で送られるが, 時間が経つに従って次第に maxpoll へと長くなっていく. これは, ポーリング間隔の変化を見れば一目瞭然である.

次に到達可能性レジスタについて説明する. このレジスタは 8 進数の数値で表され, 最大の数値は 377, すなわち 2 進数の 11111111 である. この値はパケットが到達したかしないかをビットで示している. 成功していれば 1 を, 失敗すれば 0 となる. すなわち, 8 つの統計

バッファを持つ事になる. 例えば, パケットが途中で落ちてしまった場合, 値は表 12.3 のように数値が変化する. また, ntpd 起動時は, 1 → 3 → 7 → 17 → 37 → 77 → 177 → 377 と増えていく.

パケットが落ちなければ, ずっと 377 を表示する.

バッファ	8 進数	16 進数	バッファ	8 進数	16 進数
11111111	377	255	11101111	357	239
11111110	376	254	11011111	337	223
11111101	375	253	10111111	277	191
11111011	373	251	01111111	177	127
11110111	367	247	11111111	377	255

表 12.3: 到達可能レジスタの変化

lpeers

peers と同じ動きをするが, 曖昧なものも含めてサーバが状態を保持する全アソシエーションの要約情報を表示する.

opeers

古い形式の peers コマンドで, 参照 ID の代わりにローカルインタフェースアドレスが使われる.

```
ntpq> opeers
     remote           local      st t when poll reach   delay   offset    disp
                                     (s)  (s)          (ms)     (ms)     (ms)
==============================================================================
 127.127.1.0     127.0.0.1        2 l   34   64  377    0.000    0.000   0.004
-210.173.160.27  1.2.3.4          2 u 1922 1024  376    1.311    2.884   0.779
*210.173.160.57  1.2.3.4          2 u  839 1024  377    1.419    3.405   0.668
+210.173.160.87  1.2.3.4          2 u  321 1024  377    1.187    3.583   0.279
```

pstatus assocID

指定されたアソシエーションのステータス読み取り要求をサーバに送信し, 戻されたステータス値, 名前, 及びピア変数の値を表示する.

12.1.4 タリーコード

peers コマンドの出力 (ntpq -p) の左端に表示される文字は, タリーコードと呼ばれるもので, クロック選択アルゴリズムにおけるこのピアの状態を表す. ピアの一番左端に表示される文字あるいは空白は表 12.4 の意味がある.

文字	意味	説明
空白	reject	このサーバは, (1) IP 的に到達不能 (2) このサーバに同期されている (sync loop 状態) (3) 距離距離が遠い, の理由で捨てられたサーバ
x	falsetick	intersection 検査で捨てられたサーバ
.(ピリオド)	excess	参照サーバが多いため (10 以上ある) に捨てられたサーバ
-	outlyer	クラスタリング検査で捨てられたピア
+	candidat	検査に合格し, いつでも参照可能なサーバ
#	selected	参照可能なサーバだが, 距離が遠い
*	sys.peer	同期中のサーバ. このマーク付くためには, 後述の reach のエントリに少なくとも 5 回以上連続して受信に成功していることを表す 37, 77, 177, 377 のいずれかが表示されている必要がある.
o	pps.peer	同期中の PPS 信号サーバ

表 12.4: タリーコード

2 カラム目以降の意味は表 12.5 の通りである.

名称	意味
remote	リモートサーバのホスト名または IP アドレス
refid	サーバが参照している NTP サーバのマシン ID (不明の場合は 0.0.0.0)
st	サーバのストラタム (サーバの階層番号)
t	ピアのタイプ (l:local, u:unicast, m:multicast, b:broadcast)
when	最後のパケットを受け取ってからの経過時間 (秒)
poll	ポーリング間隔 (秒)
reach	到達可能に関するレジスタ (8 進数表現で到達実績が多いと大)
delay	remote への推定遅延 (ミリ秒)
offset	remote との時間のずれ (ミリ秒)
disp	remote の時間のばらつき, 分散 (ミリ秒)

表 12.5: 2 カラム目以降のラベルの意味

12.2 ntpdc

ntpdc コマンドは ntpd デーモンの状態表示, 状態の変更や設定の変更を行うプログラムで, ntpq と似てコマンドラインからの実行と対話モードがある. ntpq との違いは ntpdc は NTP モード 7 メッセージ (図 12.2) を使っている点で, ntpd の設定変更をダイナミックに行うことができる点にある.

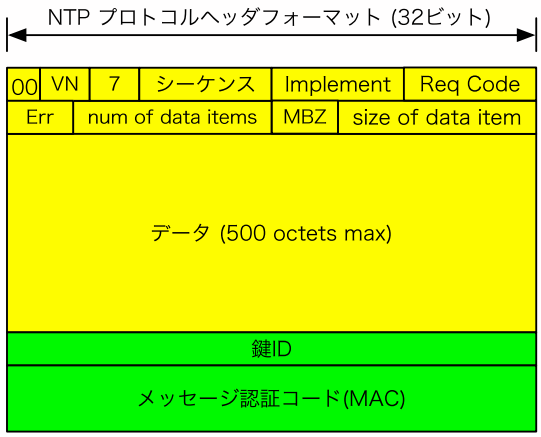

図 12.2: モード 7 制御メッセージ (include/ntp_request.h)

コマンド行の形式とオプションは次のとおりである.

```
ntpdc [ -46ilnps ] [ -c command ] [ host ] [ ... ]
```

オプション	説明
-4	ホスト名を IPv4 アドレスで解決する
-6	ホスト名を IPv6 アドレスで解決する
-c	対話形式のコマンドを指定して実行する
-i	対話形式モードで操作を行う
-l	ピアのリストを表示する (これは対話形式の listpeers サブコマンドを実行したものと同じである)
-n	ホスト名ではなく, IP アドレス形式で表示する
-p	NTP サーバのサマリ情報を表示する (これは対話形式の peers サブコマンドを実行したものと同じである)

オプション	説明
-s	状態のサマリ付きのピアリストを表示する. ntpq -p とは若干異なり, 対話形式の dmpeers サブコマンドを実行したものと同じである.

12.2.1 内部コマンド

コマンド	説明
? [*command_keyword*]	
help [*command_keyword*]	コマンドキーワードを入力せずに?あるいは help とタイプすると ntpq で利用できるコマンドの一覧を表示する. コマンドキーワードを入力すると, そのコマンドに関する情報を表示する
delay *milliseconds*	認証を必要とする要求に組み込まれるタイムスタンプに追加する時間間隔を指定する. 長遅延ネットワークまたは時間が同期化されていないコンピュータ間で信頼性のないサーバを再構成することができる. 引き数を指定しないでこのサブコマンドを入力した場合, 現行設定値が表示される.
host *hostname*	問い合わせを送るホスト (FQDN あるいは IP アドレス) を指定する
hostnames [yes | no]	yes と指定すると表示される情報はホスト名が表示され, no とするとホスト名で表示できても IP アドレスで表示される
keyid *keyid*	問い合わせ要求を認証する時に使用する鍵番号を指定する (サーバ側と一致する必要がある)
passwd	認証コンフィグ要求で使われるパスワードを入力する (エコーされない)
quit	ntpq を終了する
timeout *milliseconds*	サーバへの問い合わせのタイムアウト値を指定する (デフォルトは約 5000 ミリ秒)

12.2.2 制御メッセージコマンド

NTP モード 7 メッセージを使って ntpd デーモンの情報 (読み取りのみ) を得るための制御メッセージで, ntpq コマンドよりも多くの情報を表示する.

listpeers

サーバがピアを維持している状態の簡易なピアリストを表示する. このリストには時刻同期する可能性があると考えられるピアも含まれる.

```
ntpdc> listpeers
client    127.127.1.0
client    210.173.160.87
client    210.173.160.57
client    210.173.160.27
```

peers

サーバがピアを維持している状態のピアリストを表示する. 表示する内容はリモートピアの IP アドレス, ローカルインタフェースアドレス, ストラタム (16 の場合は非同期である), ポーリング間隔 (秒), 到達可能性レジスタ (8 進数), ピアの推定遅延・オフセット・分散 (全て秒) である. 左の記号はサーバのモードを表している (表 12.8, タリーコードとは違うので注意).

```
ntpdc> peers
     remote           local      st poll reach  delay   offset    disp
==============================================================================
=127.127.1.0     127.0.0.1       10   64  377 0.00000  0.000000 0.00093
=210.173.160.87  1.2.3.4          2 1024  377 0.00119  0.000603 0.01117
*210.173.160.57  1.2.3.4          2 1024  377 0.00137  0.000616 0.01117
=210.173.160.27  1.2.3.4          2 1024  377 0.00142  0.000608 0.01117
```

dmpeers

peers サブコマンドと左の文字が違うだけである. 左の文字の意味は, クロック選択アルゴリズムの最終段階にあるピアを表示していて, ".(ピリオド)" は falseticker の検出で捨てられたピアを意味し, falseticker 検出を通って選択リストに載ったものが, "+" で表示される. そして, "*" は現在時刻同期していることを意味する.

文字	意味
+	symmetric active モード
−	symmetric passive モード
=	このサーバに対し, クライアントモードでポーリングしている
^	このアドレスにブロードキャストしているサーバ
~	リモートピアがブロードキャストを送っている
*	現在時刻同期しているサーバ

表 12.8: ntpdc のモード情報

```
ntpdc> dmpeers
     remote           local      st poll reach  delay   offset    disp
==============================================================================
 127.127.1.0     127.0.0.1       10   64  377 0.00000  0.000000 0.00093
.210.173.160.87  210.171.226.186  2 1024  377 0.00139  0.000673 0.01405
*210.173.160.57  210.171.226.186  2 1024  377 0.00139  0.000761 0.01404
.210.173.160.27  210.171.226.186  2 1024  377 0.00174  0.000603 0.01404
```

showpeer

指定されたピア (複数指定できる) の詳細な情報を表示する. これらの情報は NTP バージョン 2 の仕様に出ているものである.

```
ntpdc> showpeer 210.173.160.57
remote 210.173.160.57, local 1.2.3.4
hmode client, pmode mode#255, stratum 2, precision -18
leap 00, refid [210.173.160.86], rootdistance 0.00047, rootdispersion 0.00514
ppoll 10, hpoll 10, keyid 0, version 4, association 15550
reach 377, unreach 0, flash 0x0000, boffset 0.00400, ttl/mode 0
timer 8061450s, flags system_peer, config, bclient
reference time:      c3f138c8.8cb35000  Thu, Mar  4 2004 14:10:32.549
originate timestamp: c3f13a6c.ade9e000  Thu, Mar  4 2004 14:17:32.679
receive timestamp:   c3f13a6c.ade58e64  Thu, Mar  4 2004 14:17:32.679
transmit timestamp:  c3f13a6c.ad88a04d  Thu, Mar  4 2004 14:17:32.677
filter delay:  0.00139  0.00131  0.00137  0.00130
               0.00162  0.00148  0.00146  0.00186
filter offset: 0.000761 0.000669 0.000616 0.000478
               0.000424 0.000416 0.000299 0.000220
filter order:  0        1        2        3
               4        5        6        7
offset 0.000761, delay 0.00139, error bound 0.11848, filter error 0.00009
```

pstats

ピア毎の統計情報を表示する.

```
ntpdc> pstats 210.173.160.57
remote host:          210.173.160.57
local interface:      1.2.3.4
time last received:   650s
time until next send: 376s
reachability change:  20062s
packets sent:         50
packets received:     50
bad authentication:   0
bogus origin:         0
duplicate:            0
bad dispersion:       0
bad reference time:   0
candidate order:      6
```

clockstat

ローカルのクロック (127.127.X.X) のファッジ係数やパフォーマンスに関する情報が表示される.

```
ntpdc> clockstat 127.127.20.1
clock address:        127.127.20.1
clock type:           Generic NMEA GPS Receiver (20)
last event:           0
current status:       0
number of polls:      1157339
no response to poll:  589
bad format responses: 0
bad data responses:   7933
running time:         18517420
fudge time 1:         0.000000
fudge time 2:         0.000000
stratum:              0
reference ID:
fudge flags:          0x0
```

kerninfo

カーネルのフェーズロックループの操作パラメータを表示する.

```
ntpdc> kerninfo
pll offset:           0.000656 s
pll frequency:        -13.018 ppm
maximum error:        0.084836 s
estimated error:      0.000647 s
status:               0001  pll
pll time constant:    6
precision:            1e-06 s
frequency tolerance:  512 ppm
pps frequency:        0.000 ppm
pps stability:        512.000 ppm
pps jitter:           0.0002 s
calibration interval: 4 s
calibration cycles:   0
jitter exceeded:      0
stability exceeded:   0
calibration errors:   0
```

loopinfo

選択したループフィルタの変数を表示する. ループフィルタは NTP の中でローカルシステムのクロック調整を扱う部分で, ここで表されるオフセット (offset) はパケット処理コー

ドがループフィルタにあたえた最後のオフセットを意味する. 周波数 (frequency) はローカルクロックの周波数エラー (ppm) を表す. ウォッチドッグタイマ (watchdog timer) は最後のサンプルオフセットがループフィルタに与えられてから経過した秒数を表す。

`oneline` とすれば, この情報が 1 行で表され, `multiline`(デフォルト) と指定すれば, 次のように複数行で表示される.

```
ntpdc> loopinfo
offset:               0.000671 s
frequency:            -13.018 ppm
poll adjust:          30
watchdog timer:       604 s
ntpdc> loopinfo oneline
offset 0.000671, frequency -13.018, time_const 30, watchdog 604
```

`sysinfo`

表示された項目のうち, 種々のローカルサーバに関する状態変数を表示する. “system flags” は様々なシステムフラグを表示し, そのうちのいくつかは `enable` や `disable` コマンドで設定や解除ができる. 安定性 (“stability”) は, システム周波数の修正が適用された後に, どうしてもなくせない周波数エラーで, メンテナンスおよびデバッグを目的としている. 多くのアーキテクチャでは, この値は最初の 500ppm から, 0.01∼0.1ppm の範囲内のほんのわずかな値にまで減少する. デーモンが起動してから暫く経っても高い値が続く場合は, ローカルクロックに何らかの問題があるか, カーネルの変数が正しくないと考えられる. “broadcastdelay” は, `broadcastdelay` コマンドで設定したブロードキャストの遅延を表示し (何も設定されていなくても, デフォルト値が設定されている), “authdelay” は `authdelay` コマンドで設定したデフォルトの認証遅延を表示する.

```
ntpdc> sysinfo
system peer:          210.173.160.57
system peer mode:     client
leap indicator:       00
stratum:              3
precision:            -18
root distance:        0.00185 s
root dispersion:      0.02907 s
reference ID:         [210.173.160.57]
reference time:       c3f14a70.a05da6a4  Thu, Mar  4 2004 15:25:52.626
system flags:         auth monitor ntp kernel stats
jitter:               0.000641 s
stability:            0.004 ppm
broadcastdelay:       0.003998 s
authdelay:            0.000000 s
```

sysstat

プロトコルモジュールにある統計情報を表示する.

```
ntpdc> sysstat
system uptime:          25191
time since reset:       25191
bad stratum in packet:  0
old version packets:    55
new version packets:    160
unknown version number: 0
bad packet format:      0
packets processed:      160
bad authentication:     0
packets rejected:       0
```

memstat

メモリの割り当てコードに関連する統計情報を表示する.

```
ntpdc> memstat
time since reset:     25239
total peer memory:    15
free peer memory:     11
calls to findpeer:    160
new peer allocations: 0
peer demobilizations: 0
hash table counts:      0   1   0   0   0   0   0   0
                        0   0   0   0   0   0   0   0
                        0   0   0   0   0   0   0   1
                        0   1   0   1   0   0   0   0
```

iostat

input-output モジュールにある統計情報を表示する.

```
ntpdc> iostat
time since reset:     25338
receive buffers:      9
free receive buffers: 9
used receive buffers: 9
low water refills:    0
dropped packets:      0
ignored packets:      0
received packets:     217
packets sent:         226
packets not sent:     0
interrupts handled:   213
received by int:      217
```

timerstat

タイマー/イベントキューのサポートコードにある統計情報を表示する.

```
ntpdc> timerstat
time since reset:  25373
alarms handled:    0
alarm overruns:    0
calls to transmit: 0
```

reslist

restrict コマンドで指定したサーバのアクセス制御リストを, 適用される順番に表示する.

```
ntpdc> reslist
   address           mask             count        flags
==================================================================
0.0.0.0         0.0.0.0                  0  ignore
127.0.0.1       255.255.255.255         60  none
210.171.226.186 255.255.255.255          0  ntpport, interface, ignore
210.173.160.27  255.255.255.255         54  noquery, nomodify
210.173.160.57  255.255.255.255         55  noquery, nomodify
210.173.160.87  255.255.255.255         51  noquery, nomodify
```

ifstats

ifstats コマンドで ntpd が利用しているネットワークインタフェースの統計情報を表示する.

```
ntpdc> ifstats
Keyid: 1
MD5 Password:
 # A Address/Mask/Broadcast T E IF name Flg TL #M recv sent drop S IX  PC uptime
==============================================================================
 0 . 0.0.0.0       A D wildcard 089   0  0     0     0     0  0  0   0      90
                                     255.255.255.255 M
 1 . 127.0.0.1     A E lo0      015   0  0     0     0     0  0  0   0      90
                                          255.0.0.0 M
 2 . 192.168.1.100 A E en0      019   0  0     6     6     0  0  0   3      90
                                       255.255.255.0 M
```

ifreload

システムインタフェースを再スキャンを実行することで, 統計情報の出力を更新できる. 変化があったインタフェースには “.”, 新たに加えられたインタフェースは “+”, 削除されたインタフェースは “–” がマークされる.

monlist

モニタ機能が集めたトラフィック情報を表示する.

```
ntpdc> monlist
remote address          port local address      count m ver drop   last   first
===============================================================================
127.0.0.1               32802 127.0.0.1            50 7 2      0      0   25463
210.173.160.57            123 210.171.226.186      55 4 4      0    717   25455
210.173.160.27            123 210.171.226.186      54 4 4      0    721   25451
210.173.160.87            123 210.171.226.186      51 4 4      0    774   25450
```

clkbug

基準クロックドライバーのデバッグ情報を表示する. この機能は, クロックドライバーが提供するもので, ドライバのソースコードがなければデコードできない.

```
ntpdc> clkbug 127.127.20.1
clock address:        127.127.20.1
values: 8
           4          64           6          57
          18           0  3281904000     1164802
times: 6
3287372238.000000 04:063:06:57:18.000  3287372238.000002 04:063:06:57:18.000
         0.000000                              -0.000002
         0.000000                               0.000001
```

12.2.3 ランタイム設定要求コマンド

ntpdc は ntpd デーモンが動作中に設定変更することができる. 但し, 設定変更の要求はサーバが認証する必要があり, サーバ側で認証機能を有効にしておかなければならない (7.6.4 節, 83 ページ参照). ランタイム設定要求コマンドを入力すると, Keyid が尋ねられる.

```
ntpdc> addserver 192.168.1.1
Keyid:
```

認証はパスワードによって行われるが, クラッカの盗聴や再生攻撃による侵入をより難しいものにするため, 認証を受けた要求にはタイムスタンプが含まれており, サーバ側はタイムスタンプを要求を受理したタイムスタンプと比較して, この差が大きな場合は要求を拒否するようにできている. また, 時間を比較することで離れたホストからの変更が難しくなるという利点もある.

addpeer

指定したアドレスに対するピアアソシエーションを追加し, peer active モードで動作するようになる. これは ntp.conf に peer コマンドで追加するのと同義である.

addserver

加えるのがクライアントモードであるということを除いて, addpeer コマンドと同義である.

broadcast

加えるのがブロードキャストモードであるということを除いて, addpeer コマンドと同義である.

unconfig

このコマンドは, 指定したピアを削除するコマンドである. 多くの場合, このコマンドによってピアのアソシエーションが削除される.

fudge

このコマンドは, リファレンスクロックの設定を追加するコマンドである。

restrict/unrestrict

restrict コマンドは, ntp.conf の restrict コマンドと同じようにアクセス制御を行うコマンドを追加する. また, unrestrict コマンドは対象となるアクセス制御リストを無効にする.

delrestrict

一致するアクセス制御リストのエントリを削除する.

readkeys

現在の認証キーの設定を消去し, ntp.conf ファイルで指定されている鍵ファイル (ntp.keys) を再読み取りし, 新しく設定し直す. これによって, デーモンを再起動せずに, 暗号鍵を変更できる.

readkeys/untrustkey

ntp.conf の trustedkey 及び untrustkey 設定ファイルコマンドと同じ機能を持つ.

authinfo

既知の鍵及び実行した暗号化および復号化の統計情報を含む認証モジュールに関する情報を表示する.

traps

サーバに設定したトラップを表示する.

addtrap

非同期メッセージのトラップを設定する.

clrtrap

非同期メッセージのトラップを解除する.

reset

サーバの様々なモジュールの統計カウンターをリセットする.

12.3 ntptrace

Network Time Protocol(NTP) サーバのチェーンをマスタ時刻源に戻ってトレースするコマンド (Perl スクリプト) である. 下記がコマンド行の一覧である. 引数の指定が無ければ, localhost からトレースを始める.

```
ntptrace [ -vdn ] [ -r retries ] [ -t timeout ] [ server ]
```

オプション	説明
-d	デバッグ機能を有効にする
-n	ホスト名ではなくホストの IP アドレスを出力する
-r *retries*	ホスト毎の再試行回数を指定する (デフォルトは 5)
-t *timeout*	再送タイムアウトを秒数で指定する (デフォルトは 2 秒)
-v	トレースした NTP サーバの詳細情報を出力する

ntptrace を実行すると次のようになる.

```
# /usr/sbin/ntptrace
localhost: stratum 3, offset 0.000023, synch distance 0.03494
ntp1.jst.mfeed.ad.jp: stratum 2, offset -0.000427, synch distance 0.01280
mf-isdn4.mfeed.ad.jp:   *Timeout*
```

それぞれの意味は左から右に：ホスト名, ホストのストラタム, ホストとローカルホスト間の遅延時間 (秒), ホストの同期距離 (秒), 最後にそのホストがストラタム 1 であれば参照クロック ID が表示される. ntptrace の実体は Perl スクリプトで, ntpq を動かしてトレースしているだけなので, NTP モード 6 制御メッセージが届かなければ, “*Timeout*” と表示される.

12.4 Web 監視の方法

NTP デーモンから得られるデータをグラフ化することで, 時間変動の傾向を一目で把握できるので, とても便利である. NTP の統計情報は syslog や ntpdc コマンドを使って取ることができるので, そのデータを MRTG や RRDtools を使ってグラフ化すれば良い.

取得しておきたいデータとしては,

- サーバ間の往復遅延：この時間が一定であるほど, 時刻同期の精度は高くなる.
- サーバとのオフセット (時間差)：UTC 時間を持つサーバとの時間差を示し, 値が 0 に近いほど精度が高い.
- 遅延時間の分散値：値が小さければ遅延時間のばらつきが少ないことを示し, 時刻が安定している.

などである. ローカルにリファレンスクロックを持っていれば,

- PLL オフセット
- PPS の安定度
- GPS 情報 (同期衛星の情報)

なども有効と思われる.

データの取得や表示 には色々な方法があると思われるが, 以下に公開されている Web 監視情報の URL を紹介するので, 参考にして欲しい.

- http://www.ipver6.jp/howtontp.ja.sjis.html

第13章　NTPサーバの運用

13.1　セキュリティ

NTP サーバの運用者は, 他のサーバプログラムと同様に ntpd デーモンが必ずしも安全であると過信すべきではなく, ntpd が何らかの手段で破られても良いという前提でサーバを運用すべきである.

サーバプログラムのセキュリティ破りで一番恐ろしいのが, root 権限やシェルの奪取である. そのため, 重要な事は,

- ファイルのアクセス権限を限定する
- ntpd デーモンは root 権限で動かさない
- ntpd は chroot や jail 環境の元で動かす

13.1.1　ファイルのアクセス権限

非常に基本的なことだが, NTP の設定ファイル (ntp.conf) や鍵ファイル (ntp.keys, Autokey の鍵や証明書) のアクセス権限を root もしくは, ntpd が起動するユーザのみが読めるよう設定しておくべきである.

```
-rw-------    1 root     root         2188 Apr 24 15:42 ntp.conf
-rw-------    1 root     root         2188 Apr 24 15:45 ntp.keys
```

また, ntp.conf にはコメントが記述できるので, 設定の意味を詳しく記述しておくことをお薦めする. 特に restrict にはポリシーを記述しておくと, なぜこの設定をしたのかがわかり, 設定の見直しや管理者の交替などで引継ぎが発生してもスムースに進む.

13.1.2　デーモンの権限 (CAP_SYS_TIME)

ntpd はシステムの時計を取り扱うため, Unix 系 OS では root 権限が必要となり, ふつうは ntpd を root 権限で動作させている. ところが, ntpd の何らかのセキュリティホールを

利用された場合, 侵入者に root 権限を渡してしまうため, できれば ntpd を root 権限で動かしたくない.

このようなセキュリティ上の問題を解決するため, オペレーティングシステムのセキュリティ機構の改良が検討された. セキュア機能を搭載した OS は, 従来スーパーユーザ (root) が持っていた権限を細かく分割し, そのプロセスに必要ではない権限を与えないようにできる. ntpd の場合, システム時刻を変更するための権限のみを持てば良いので, CAP_SYS_TIME という資格 (ケーパビリティ) を ntpd プロセスに与えれば良い.

Linux を例に説明すると, ntpd は起動後に cap_set_proc(3) や Linux 固有の capsetp(3) ルーチンを呼び出して, CAP_SYS_TIME という一つだけのケーパビリティを持つようにしておけば, その後はそのプログラムの機能はシステム時計の変更のみにでき, ntpd が乗っ取られても他の機能にアクセスされる恐れはなくなる. 但し, ntp-stable-4.2.0a-20040225 から対応しているが, ntpd-4.2.0 以前のバージョンはケーパビリティに対応していないため, パッチ[1]が必要となる. Linux のケーパビリティを有効にするには, configure 時に --enable-linuxcaps を付けてコンパイルする.

単純なアクセス権限構造は Unix の問題の一つとして知られていたが, 改善するための標準規格としては, “POSIX.1e”[2] が知られている (但し, この規格は取り消されてしまった). 結局, 現時点で規格は存在しないのだが, ほかっておくわけにもいかず, POSIX.1e の最終ドラフトを元に OS の開発者が独自にアクセス権限機能を実装しているのが現状である (表 13.1).

種類	OS 名, ツール名
Linux	Kernel 2.4.20 以降, 2.6, SELinux, PitBul, LIDS, LSM
BSD	Trusted BSD, FreeBSD 5.0
Sun	Trusted Solaris, Solaris 10
HP	HP virtualvault

表 13.1: アクセス権限が拡張されているオペレーティングシステム

13.1.3 chroot jail への閉じ込め

OS がアクセス権限を拡張していない場合は, chroot jail 環境を作って, ntpd をその中に閉じ込める方法がある. chroot(チェンジ・ルート) というのは, あるプログラムを閉じ込め

[1] Linux 用のパッチは “http://bugzilla.ntp.org/show_bug.cgi?id=251” にある.
[2] http://wt.xpilot.org/publications/posix.1e/

られた小さなディレクトリツリーの中に置き, そこから外にあるファイルをアクセスできないようにする方法である.

ntpd を chroot jail の中に置く事で, 侵入者が仮にセキュリティホールを使ってシステムにアクセスできたとしても, アクセスできる範囲を最小限に制限することができる.

BSD の場合

BSD 系の OS では ntpd を chroot jail 環境下に置くと, 下記のメッセージが表示され, 時刻を合わせることができない.

```
Feb 22 15:04:34 foo ntpd[501]: Can't set time of day: Device not configured
Feb 22 15:04:34 foo ntpd[501]: time reset -45.242243 s
Feb 22 15:04:34 foo ntpd[501]: synchronisation lost
```

ntpd の chroot jail を構築するには, まず ntpd のコンパイル前の configure スクリプトの実行時に, "--enable-clockctl" を付ける (表 5.4 参照, デフォルトは有効). そして, /dev/clockctl という仮想デバイスを作成し ("MAKEDEV clockctl"), 仮想デバイスをカーネルに登録する必要がある. このデバイスができたために, ntpd は chroot jail 環境で動くようになったと云える.

```
pseudo-device   clockctl
```

カーネルの再構築が完了したら, ntp という名前のユーザ/グループを作り, 下記のように起動する.

```
ntpd -u ntp:ntp -i /var/chroot/ntp
```

Linux の場合

ケーパビリティ機能を有効 (--enable-linuxcaps にして ntpd をコンパイルし, BSD の時と同様に ntp という名前のユーザ/グループを作り, 下記のように起動する.

```
ntpd -u ntp:ntp -i /var/chroot/ntp
```

また, 主要な Linux distro の ntpd は chroot 化されているケースが多い. Fedora Core の場合, ntp-4.2.0-droproot.patch というパッチが適用されており, 下記ように起動すると

chroot 環境で動作する (ケーパビリティを利用するために libcap ライブラリのインストールが必要).

```
ntpd -U ntp -T /var/chroot/ntp
```

13.2 NTP のトラブルシューティング法

ntpd を設定しても, うまく時刻同期ができない場合のトラブルシューティング方法を少し紹介する.

13.2.1 起動しない場合のトラブルの解決法

- まず, ntpd は起動時に ntp.drift と呼ばれるファイルが無いと, システムクロックの周波数誤差を修正するためのドリフト係数を算出するため, 特別なモードに入る. ファイルを生成した後に, 時刻同期を行うため, それまで 20 分程度時間がかかる.
- -d オプションを付けて起動すると, どのようなファイルが読み込まれるといった詳細情報が表示される. 各種メッセージが標準出力されるので, このログを記録して後で解析すると, どこに問題があるか見付けやすい. 例えば, Autokey で使われるファイルが設定されていなければ, エラーが表示されるのでとても参考になる.
- NTP は UDP のポート 123 が送信ポートであり受信ポートでもある. ルータやファイアウォールでこのポートが閉じられているか確認する.
- システムログやエラーログを確認する.
- ntpd は起動時にサンプルの時刻を得て, 有効な時間を持つサーバかどうかを判別する. クライアント時間とサーバ時間が極端に離れている場合 (1000 秒以上離れている[3]), システムログを出力し, 時間同期を行わずに終了する. ntpd を走らせる前に, 適度に時間があっている時計を見て, システムの時計が狂っていないかを確認し, もし狂っていれば date コマンドを使ってその時刻に合わせるか, ntpdate や ntpd -q コマンドで近くの NTP サーバから時刻を取得する.

[3] この閾値 (1000 秒) は “tinker panic” コマンドで修正できる.

- ローカル時間とサーバ時間の間でステップ閾値が 128 ミリ秒[4]よりも大きい場合, スリュー調整が行われないため, 時刻同期ができない状態になる. 従って, 時刻を他の NTP サーバと合わせてから, ntpd を起動する必要がある.

13.2.2 ピアキッスコード

ntpq プログラムでピア情報を表示させる次のように表示される. refid にはふつう参照しているクロック源あるいは NTP サーバが表示される.

```
ntpq> pe
     remote           refid      st t when poll reach   delay   offset  jitter
==============================================================================
 LOCAL(0)        LOCAL(0)        10 l   48   64  377    0.000    0.000   0.004
*ntp1.jst.mfeed. mf-isdn2.mfeed.  2 u  679 1024  377    1.331    0.624   0.216
+ntp2.jst.mfeed. mf-isdn1.mfeed.  2 u  741 1024  377    1.738    0.450   0.129
+ntp3.jst.mfeed. mf-isdn4.mfeed.  2 u  728 1024  377    1.178    0.462   0.166
```

アソシエーションが確立されていない状態の時に, この refid 部分には**ピアキッスコード**と呼ばれるコードでピアとの状態を表示する (表 4.6 参照). このコードがピアとのトラブルシューティングにも利用できる.

13.3 pool.ntp.org プロジェクト

NTP サーバを利用する場合, サービスではなく NTP サーバのホスト名 (FQDN) もしくは IP アドレスを知る必要がある. パブリックな NTP サーバはリストとして公開されているが, それらが正しく動いているかは, 実際に動かすか ntptrace を使って調べる必要がある.

そこで, 2003 年 1 月に Adrian von Bidder 氏が中心となり, DNS のラウンドロビン機能を使って多くの NTP サーバを一つのサーバ名でアクセスできるようにするためのプロジェクトを立ち上げている (2005 年 7 月からは Ask Bjøn Hansen 氏が引き継いで管理している). このプロジェクト名はアクセスするサーバ名を冠している. 立ち上げ当初, *time.fortytwo.ch* プロジェクトという名前で始まったが, David Mills 氏の協力により, プロジェクト名は *pool.ntp.org* に変わった.

[4] この値 (0.128 秒) は “tinker step” コマンドで修正できる.

13.3.1 pool.ntp.org の利用方法

pool.ntp.org の利用方法は非常に簡単で, サーバ名に *n*.pool.ntp.org を指定するだけである. サーバを複数指定しなければ意味が無いので, 典型的な例としては次のように設定する.

```
driftfile /var/lib/ntp/ntp.drift
server 0.pool.ntp.org
server 1.pool.ntp.org
server 2.pool.ntp.org
```

サーバは DNS ラウンドロビンでランダムに選ばれる. アルゴリズムによって一番良い NTP サーバが選択されるが, NTP は遅延の影響を受けやすいため, 国別 (日本の場合, jp.pool.ntp.org) や地域別 (アジアなら asia.pool.ntp.org) にゾーンを分ける試みが行われている.

13.3.2 pool.ntp.org への参加方法

インターネットに公開可能な NTP サーバを立ち上げたなら, pool.ntp.org への参加もできる. 固定の IP アドレスと十分な帯域幅を持っていれば, 参加が可能である. 参加方法もいたって簡単で, 管理ページ[5]からリクエストを送るだけである.

サーバの制限事項は, pool.ntp.org で登録されている NTP サーバ群を参照していてはいけない点とグローバル IP アドレスがスタティックに設定されている必要がある. また, pool.ntp.org は WWW サーバも立ち上げているため, NTP サーバで Apache を動かせるようにしておく事が望まれている. 次のように本家サーバに Redirect するよう設定しておけば良い. Apache サーバの設定は以下のように設定する.

```
<VirtualHost *:80>
    ServerName pool.ntp.org
    ServerAlias *.pool.ntp.org
    Redirect permanent / http://www.pool.ntp.org/
</VirtualHost>
```

[5] http://www.pool.ntp.org/manager

13.3.3 Anycast による方法

NTP サーバは UDP を使った通信であるため, DNS 同様 Anycast を使う方法が考えられなくは無い. (1) NTP は時刻同期を時間をかけて計算するためサーバの選択が安定して行われる必要がある, (2) サーバを複数選択する必要がある, ことから, サーバ同士の NTP 接続には Anycast は馴染まないと考えられる.

但し, PC のようなクライアントが時刻問い合わせを間欠的に行う場合, 同じサーバである必要はないため, Anycast は有効である (Anycast を使う NTP サーバは NTP で同期されていることが前提).

第14章　パソコン系OSの時刻調整

本章ではパソコン系 OS の代表としてマイクロソフト社の Windows と Apple 社の Mac OS X の時刻調整方法について解説する.

14.1　Microsoft Windows の時刻調整

14.1.1　Windows の時刻の取り扱い

NT Time Epoch

Windows も Unix と同様に内部時間を持ち, NT Time Epoch(NTTE) と呼ばれている. NTTE は, 64 ビットカウンタで 1601 年 1 月 1 日 00:00:00 UTC からの経過秒数で表され, その分解能は 10^{-7} 秒 (100 ナノ秒), すなわち 100 ナノ秒ごとにカウントアップする.

ハードウェアクロック

Windows は Linux や BSD 系オペレーティングシステムに比べて, 時刻の扱い方が古く, ハードウェアクロックとソフトウェアクロックが独立していないため, ハードウェアクロックをローカル時間と同じ時間に合わせなければならない. 従って, Linux や BSD などとマルチブート環境を使って共存させる場合にとても不都合である. Windows はこの挙動を変えることができないので, Linux もしくは BSD 側でハードウェアクロックがローカル時間であるような設定に行わなければならない (2.2 節, 14 ページ参照).

夏時間の取り扱い

Windows はタイムゾーンに応じて, 夏時間が自動調整できるかどうかが決まっているため, 現在のところ日本にはそのオプションが無い. 日本で夏時間が採用された場合, タイムゾーンのレジストリの修正が生じる (マイクロソフト社からパッチが出ると思うが)[1].

[1] この問題が実際にオーストラリアで起きた.

閏秒の取り扱い

Microsoft Windows は閏秒をサポートしない. しかも, Windows で実装されている SNTP/NTP (W32Time: Windows Time Synchronization サービス) クライアントは, 閏秒インジケータ (LI) を閏秒の挿入警告を関知しないため, その日に閏秒が挿入された場合, 次の時刻同期まで 1 秒進んでいることになる[2]. もしも, 閏秒インジケータが正しく解釈できていれば, 時刻同期に関係なく閏秒をその日に挿入することが可能になる筈である (図 4.14).

更に具合の悪い事に, Windows の SNTP クライアントの同期間隔は表 14.2 のようになっており, 成功さえしていれば 8 時間おきに同期するようになる. 従って, 正常に同期できている場合, 閏秒発生から 8 時間の間は時間が 1 秒進んだり遅れたりした状態のままになってしまう. 同期間隔を短くするとか, Windows にポーティングされたデラウェア版 NTP を使うことをお勧めする.

14.1.2 NTP のサポート状況

Windows は各バージョンによって, NTP のサポート状況が異なる. Windows 2000 から Kerberos と呼ばれる認証プロトコルが追加されたが, Kerberos はチケット生成のために時間同期を必要とするため, Windows Time サービス (W32Time) という時刻同期サービスが導入された. Windows 2000 の W32Time は SNTP を実装したもので, XP 以降は NTP が実装されている (表 14.1).

Windows	プロトコル
Windows 95, 98, Me, NT	マイクロソフト独自
Windows 2000	SNTP クライアント/サーバ
Windows XP, Server 2003, Vista	NTP クライアント/サーバ

表 14.1: 各 Windows の時刻同期プロトコル

14.1.3 Windows 2000

Windows の時刻同期は W32Time というサービスで行われるが, Windows 2000 とそれ以降ではプロトコルも設定方法も異なっている.

[2] http://support.microsoft.com/kb/909614?sd=tech&ln=ja

クライアント機能

外部の NTP/SNTP サーバ (*sntpserver*) に時刻を同期させるには, 下記のようにコマンドを実行する.

```
net time /setsntp:sntpserver
```

複数台の NTP/SNTP サーバを指定する場合, *sntpserver* 部を引用符で括ってスペースを空けて以下のように記述する.

```
net time /setsntp:"sntpserver1 sntpserver2"
```

サーバ機能

SNTP サーバとして動作せるには, サービスの GUI 画面で Windows Time サービス (w32time) を起動させるか, 下記のコマンドを実行する. これで (S)NTP サービスが実行されたことになる. 但し, このサーバ機能は閏秒インジケータの閏秒挿入警告を無視する点に注意する.

```
net start w32time（開始）
net stop w32time（停止）
```

同期間隔の変更

Windows 2000 の W32Time はデフォルトでは最初は 45 分おき, 同期が成功すると 8 時間に 1 度しか NTP サーバと時刻同期を行わない. この間隔を短くするには W32Time の下記のレジストリを変更する必要がある (Windows XP ではレジストリが異なるので注意する).

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\W32Time\Parameters
```

このレジストリで指定できるのは表 14.2 の値である.
例えば, 10 分置きに時刻同期を行うには, この値を 144 にすれば良い.

値	説明
65531	同期に成功するまで 45 分おき, 成功すると 1 日 1 回
65532	同期に成功するまで 45 分おき, 成功すると 1 日 3 回 (8 時間)
65533	1 週間に 1 回
65534	3 日に 1 回
65535	2 日に 1 回
0	1 日に 1 回
freq	1 日 *freq* 回

表 14.2: W32Time レジストリキー

14.1.4 Windows XP/2003

Windows XP や Windows Server 2003 の W32Time は NTP を使っているため, Windows 2000 よりも細かく調整ができる (後方互換のため SNTP も残っている). 調整・時刻監視・トラブルシュートためのツールとして, `w32tm` コマンドがある[3].

設定変更

W32Time の設定変更はレジストリの値を変更するのだが, 基本的な設定であれば, `w32tm /config` コマンドを使って行うことができる. 例えば,

```
w32tm /config /update /manualpeerlist:ntpservers
```

と実行すると, 同期先を変更できる.

手動による時刻同期

以下のコマンドで, 手動で時刻同期を行うことができる.

```
w32tm /config /syncfromflags:MANUAL /manualpeerlist:ntpserver
```

NTP サーバが設定されていれば, 次のコマンドでも良い.

[3]詳しくは, `http://technet2.microsoft.com/WindowsServer/en/library/b43a025f-cce2-4c82-b3ea-3b95d482db3a1033.mspx?mfr=true`

```
w32tm /resync /computer:ntpserver
```

時刻同期の適用

デフォルトでは, ワークグループ環境の場合は 15 時間以上参照先の NTP サーバと時刻がずれていると時刻の同期は行われない. この動作を変更するには, レジストリ "...\W32Time\Config" の MaxNegPhaseCorrection(時刻を後戻りさせる) と MaxPosPhaseCorrection(時刻を進ませる) の二つを変更したり無効にする.

slew モードの有効範囲

W32Time にも slew モードを持っており, 何秒間時刻が異なる状態まで適用するかを変更できる. デフォルトの値を変更したい場合, レジストリ "...\W32Time\Config" の MaxAllowedPhaseOffset の値を変更する.

時刻監視

w32tm /monitor コマンドを使うと, 時刻の監視ができる. 例えば, /computers で NTP サーバを指定すると, その NTP サーバとの ICMP 遅延時間やローカルクロックがどのくらいの時間差があるかを表示する.

```
w32tm /monitor /computers:ntp.nict.jp,ntp1.jst.mfeed.ad.jp
```

w32tm /stripchart コマンドを使うと, 定期的に監視を行い結果をアスキー文字を使ったグラフに表示することができる.

```
w32tm /stripchart /computer:ntp.nict.jp /period:60
```

時間の変換

Windows NT の内部時間 (NTTE) や NTP 時間を人間が可読可能な日時表示にすることができる. /ntte が NT の内部時間, /ntpte が NTP 時間を指定する.

```
C:\>w32tm /ntte 127900000000000000
148032 09:46:40.0000000 - 2006/04/20 18:46:40
```

サーバ機能

最後に, NTP サーバとして動作せるには, サービスの GUI 画面で Windows Time サービス (w32time) を起動させるか, 下記のコマンドを実行する.

```
net start w32time（開始）
net stop w32time（停止）
```

なお, ActiveDirectory 環境下にあり, 時刻サーバとして動作させたい場合は, 権威のある NTP サーバとして動作させる必要がある.

14.1.5 NTP プロジェクト版 NTP のポーティング

Windows NT 4.0, 2000, XP, .NET Server 2003 で NTP プロジェクト版 NTP を動かす場合, ソースコードからコンパイルして利用する事もできるが, 誰でもすぐに使えるよう下記のサイトでバイナリが配布されている.

- http://norloff.org/ntp/
- http://www.meinberg.de/english/sw/ntp.htm

どちらのパッケージも ZIP ファイルを解凍し, `Setup.exe` をクリックしインストールするだけである. 但し, ファイルの置き場所がそれぞれ異なっている. norloff の場合, `%SystemRoot%`の下に `ntp.conf` と `ntp.drift` が, `%SystemRoot%\system32` の下に NTP の実行ファイルが置かれる. Meinberg の場合, `Program Files` の下に `NTP` というフォルダが作られ, そこにすべて置かれる.

また, どちらのポーティングも NTP を完全にポーティングしたわけではなく, OpenSSL を使った認証機能は組み込まれているが, IPv6 は利用できない等いくつか制限があるようだ.

なお, ソースコードをコンパイルする場合, Windows 上に C コンパイラ (推奨コンパイラは Visual C++ 6.0) が必要となる. また, Autokey による認証機能を使う場合は, OpenSSL があらかじめインストールされていなければならない.

14.2 Mac OS X の時刻調整方法

14.2.1 Mac OS X の時刻の取り扱い

Mac OS X というオペレーティングシステムは Mach 3.0 カーネル (モノリシック版) を基盤に BSD の機能が組み込まれた構造になっている (図 14.1). そのため, Mac OS X の時刻同期は標準で NTP を使っている. また, ハードウェア時計は UTC として扱うようになっている (2.2.6 節, 16 ページ参照).

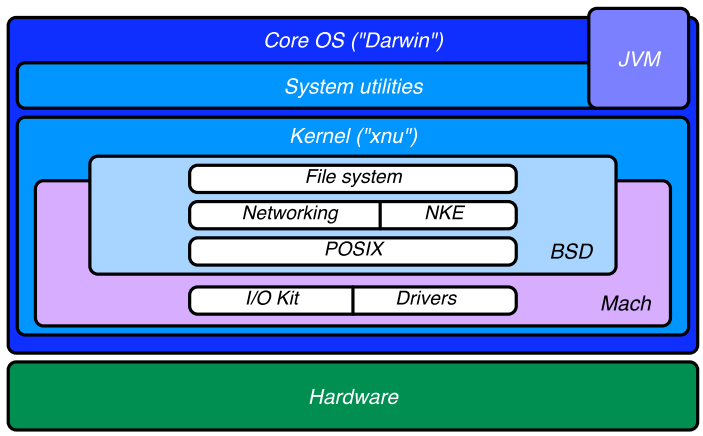

図 14.1: Mac OS X カーネル

通常, システム環境設定の “日付と時刻” で “日付と時刻を自動的に設定” を選択することで (図 14.2), Apple 社が設置している NTP サーバ[4]を参照するようになる.

14.2.2 NTP の設定

GUI で設定してしまうと細かな設定ができない. そこで, Mac OS X の NTP の設定方法を簡単に説明する. デフォルトでは `/etc/ntp.conf` が使われるが, これは NTP の起動ファイルで決められているので, この起動ファイルで参照している NTP の設定ファイルを別ファイルにしておけば, システム環境設定に関係なく NTP の設定ができる.

1. まず, 新しい NTP 設定ファイル (`/etc/yantp.conf`) を用意する.
2. NTP の設定ファイルを編集する.

[4] `time.apple.com`, 日本の場合 `time.asia.apple.com`.

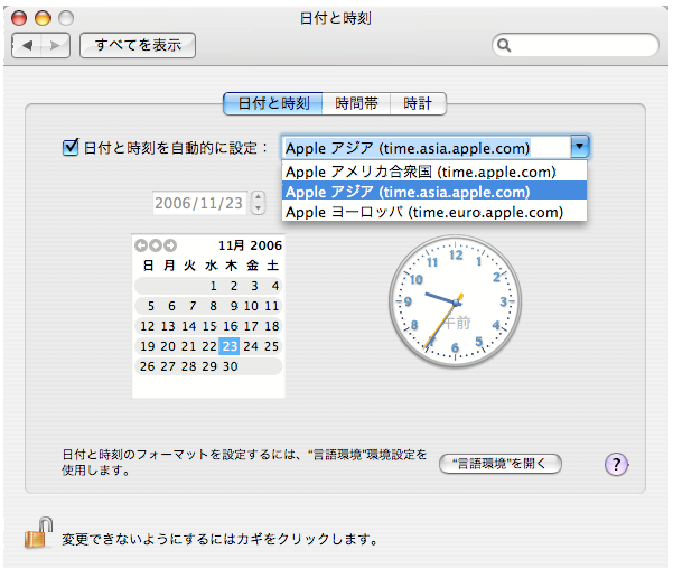

図 14.2: Mac OS X の NTP の設定 (システム環境設定)

3. NTP 起動ファイル “/System/Library/StartupItems/NetworkTime/NetworkTime” をの ntpd 起動部分を以下のように編集する.

```
ntpd -f /var/run/ntp.drift -p /var/run/ntpd.pid -c /etc/yantp.conf
```

4. Network Time サービスを再起動する.

```
sudo SystemStarter -d restart "Network Time"
```

14.2.3 スリープから復帰時の問題

Macintosh は未使用時やスリープなどによってネットワークの接続が失われると時刻同期が失われる問題がある. 実際には ntpd はネットワークの接続性が回復しても NTP サーバの妥当性のチェックやカーネルクロックの調整を完了させる必要があるため, 時刻同期が完了するには時間がかかる (クロックのずれが大きければ更に時間がかかってしまう). 一

時的な解決としてシステム環境設定の日付と時刻のパネルでチェックボックスの選択を解除し再度選択する方法があるが, いつもこれをやっていたのでは面倒である. そこで恒久的な解決方法として, クロックの調整を省いてサーバとの時刻同期を高速化できる iburst モード (7.5.5 節, 80 ページ参照) を使用することを推奨している (Apple Tech Infor Library 303731[5]).

GUI を使った設定では iburst モードにはならないので, 先ほどの NTP の設定法を利用して, サーバの指定オプションに `iburst` を付ける.

```
server ntp.nict.jp iburst
```

[5] http://docs.info.apple.com/article.html?artnum=303731

第15章　その他の時刻調整ソフトウェア

本章ではNTPプロジェクトのNTPD以外の時刻調整ソフトウェアについて解説する.

15.1　clockspeedとtaiclock

ネットワーク型では無いが, D.J Bernstain氏が作成した clockspeed というユニークなユーティリティプログラムもよくコンピュータの時刻調整に使われるプログラムである. clockspeed はハードウェアカウンタやオペレーティングシステムが持つ高分解能な時間測定機能を使って, システム時計の定常的な進みや遅れを補正するプログラムである. clockspeed はNTPのように常時通信を行う事はせず, 時刻調整時以外はサーバに依存せずに動作する.

clockspeedの動作としては, まずSNTPを使って信頼のおける時刻源を参照し (sntpclockを使う), 以後, clockspeed は自身のCPUクロックを使いながら, 時刻のずれ (スキュー) を計算して取り除く. clockspeed の原理をもう少し詳しく説明すると, IntelのCPUに用意されているRDTSC命令または, Solarisの gethrtime() のナノ秒カウンタを利用して[1], 起動時からの実行マシンサイクル数を元にシステムタイマーを調整する方法をとっており, この方法はシステムの基準発振子の安定度に依存している.

通常のPCに使われている発振子は, −10度から70度の温度範囲では10ppmから100ppmの変動を持つ水晶発振子 (TCXO) が使われ, これが PLL† 入力に使われている. そのためケース内温度の変化がシステムクロックの速度の変化として現われてしまう.

一日は $3600 \times 24 = 86400$ 秒なので, 1ppm(百万分の一) では一日あたりのズレが8.6msとなる. もし, 温度変化の影響を1ppm以下に抑えることができるなら, 時間の刻みに関しては実用には十分な精度を得られる事となる.

更に, Bernstain氏は clockspeed の他に taiclock という小さなクライアントサーバプログラムも作っている. taiclock はサーバから自身の時刻をクライアントに通知するプログラムである. taiclock 自身は正しいシステム時刻を保つ機能は無いので, clockspeed やNTPなどを使って, taiclock サーバを正確な時刻に合わせておく必要がある.

[1]従って, Intel CPUを積んだマシンかSolaris以外では動作しないという事になる.

なお, これら clockspeed や taiclock のプログラムは TAI をベースとしているため, システムのオペレーティングシステムを閏秒に対応させる必要がある.

15.1.1 clockspeed の利用方法

clockspeed の最大の特徴はハードウェアカウンタを使ってシステム時計の補正を行うプログラムであるため — カーネル時計のみを変更するプログラムではない, hwclock コマンドを使う必要はない. clockspeed は信頼できるクロック源を数回参照するだけで, スキューを計算し取り除き, 100 年に数秒狂うだけのハードウェアクロックを簡単に得ることができる.

15.1.2 clockspeed のインストール

clockspeed[2]は開発後かなり期間を経ているため, 新しいオペレーティングシステムではコンパイルエラーが発生する場合がある. 筆者が動かした Vine Linux 2.6r4 上では clockview.c にコンパイルエラーが発生したため, 下記の修正を施した.

- include ファイルを <sys/time.h> から <time.h> に変更
- void main() を int main() に変更

これらの修正を行えば, "make" でコンパイルが問題なく実行できる. コンパイルが正常に終わったら, 次に "make setup check" でインストールを行う. デフォルトでは, "/usr/local/clockspeed/bin" 以下に置かれる[3].

15.1.3 clockspeed の使い方

clockspeed を動かす前にまず信頼のできる NTP サーバと自分のローカル時計にどのくらいの差があるかを確認する. 下記の例では 3 秒ほど遅れているのがわかる. before がローカルのシステム時刻で, after が NTP の時刻 (基準時刻) である.

```
# sntpclock 1.2.3.4 | clockview
before: 2004-03-19 13:59:04.004407000000000000
after:  2004-03-19 13:59:07.209423480987548828
```

次に, clockadd を使って, 時刻を NTP サーバと合わせ確認する. 数ミリ秒に収まっているのがわかる.

[2] http://cr.yp.to/clockspeed/clockspeed-0.62.tar.gz から入手できる.
[3] インストール先を変えたければ, conf-home ファイルを変更すればよい.

```
# sntpclock 1.2.3.4 | clockadd
# sntpclock 1.2.3.4 | clockview
before: 2004-03-19 14:00:37.731074000000000000
after:  2004-03-19 14:00:37.731378183258056640
```

この状態で clockspeed を起動し, NTP サーバの時刻を clockspeed に通知する (adjust ファイルは clockspeed へのパイプファイルである).

```
# clockspeed &
# sntpclock 1.2.3.4 > /usr/local/clockspeed/adjust &
```

後は, 日に一度程度 clockspeed に時刻を通知する. これで一日数ミリ秒程度ずれる程度の精度が得られる. "/usr/local/clockspeed/etc/atto"(clockspeed が作った補正ファイル) を下記のように確認してみればズレがわかる.

```
# clockview < /usr/local/clockspeed/etc/atto
```

15.2 OpenNTPd

15.2.1 OpenNTPd とは

OpenNTPd は, OpenBSD の一部のプロジェクトとして, 主に Henning Brauer 氏によって開発が進められている NTP のサーバ/クライアントソフトウェアである. Udel 版 NTP とは異なり, ネットワークで時刻合わせを行うためだけに使われることを目的としている. リファレンスクロック, アクセス制御, 認証機能などソフトウェアを複雑にするものは一切実装されていない.

また, OpenBSD のソフトウェア群の特徴であるセキュリティに注意深く実装されている. OpenNTPd は二つのプロセスで構成されており, 親はカーネル時間を変えるために root 権限で動作するが, NTP サーバとの時刻同期は ntp engine が行い, このプロセスは _ntp 権限で動作し, 更に/var/empty に chroot 化されている. IPv6 もサポートしている点は嬉しい.

15.2.2 OpenNTPd の設定

機能を限定しているため, 設定が非常に簡単で, /etc/ntpd.conf を下記のように記述するだけで動作させられる. listen on をコメントアウトすれば, サーバモードとして動作しない.

```
listen on *
servers pool.ntp.org
```

もしも, 個別のサーバを指定したければ, servers の代わりに server を用いて, 複数行記述する.

```
server ntp1.example.org
server ntp2.example.org
```

15.2.3 Timedelta センサー

OpenBSD 4.0 からカーネルの時間精度を高めるために, Theo de Raadt 氏から Timedelta センサーというコンセプトが発表された. Timedelta センサーはローカルクロックを電波時計や GPS のようなリファレンスクロックを使って補正するためのセンサー機能を提供する. 手始めにスイス HBG とドイツの DCF77 の電波時計をハンドルできる udcf(4) というドライバが作られている.

また, 2006 OpenBSD ハッカソン (Hackathon) にて, Timedelta センサーで得られた時刻を OpenNTPd が利用できるようにする発表が行われている.

15.3 dntpd

dntpd は, DragonFly BSD で開発が進んでいる NTP のクライアントデーモンプログラムである. OpenNTPd よりも機能を絞り, サーバ機能をもっていない特徴がある. デフォルトでは, dntpd はオフになっているので, /etc/rc.conf に次のように書き加え,

```
dntpd_enable="YES"
```

/etc/dntpd.conf に NTP サーバを記述する.

```
server 1.1.1.1
server 2.2.2.2
server 3.3.3.3
```

15.4 chrony

その他に, Richard Curnow 氏 (rc@rc0.org.uk) が開発している chrony と呼ばれるソフトウェアがある. chrony はダイアルアップでインターネットにアクセスする人向けの NTP プログラムで "http://chrony.sunsite.dk/" から入手できる. ntpd ではダイアルアップ向けに, burst モードを搭載しているが, chrony は長時間電源をオフにするマシン用にリアルタイムクロック (RTC) の規則的なズレを解消する機能も含まれている. 本メモでは, chrony については以上に止めておくが, 詳しく知りたい方は chrony のホームページを参照して頂きたい.

15.5 Cisco ルータでの NTP 設定

15.5.1 Cisco ルータにおける時計の合わせ方

Cisco ルータもコンピュータと同じようにソフトウェア時計とハードウェア時計を持っている[4]. ソフトウェア時計をマニュアルで設定するには, clock set コマンドを使用する. コンフィグモードに入って, 次のように設定する. 順番は, 時間:分:秒, 日, 月, 年である.

```
Router# clock set 9:20:40 15 March 2006
```

一方, ハードウェア時計を設定するには calendar set コマンドを使用する時間の設定は clock set コマンドと同じである.

```
Router# calendar set 9:21:20 15 March 2006
```

外部からの時刻の供給を受けるのではなく, ハードウエアクロックを信頼できるクロック源とする場合, clock calendar-valid コマンドを設定する (通常, ntp master コマンドと一緒に用いる).

```
Router(config)# clock calendar-valid
```

次にタイムゾーンを合わせる. タイムゾーンは clock timezone コマンドを使用して, 次のように合わせる (UTC から 9 時間進んでいる).

[4] clock がソフトウェア, calendar がハードウェアと区別しているようだ.

```
Router(config)# clock timezone JST 9
```

Cisco ルータは NTP ソフトウェアと同じようにドリフトの計算も行っているが, 正確なドリフト値が得られるまでの間, clock time-period コマンドで指定した値を使ってソフトウェア時刻を計算する. この値はデフォルトで 171798692^{-32}(約 4 ミリ秒) で, ユーザがマニュアルに指定できるようになっているが, 設定せずに十分補正値が計算された後 (1 週間くらい), 設定値を書き込む (copy running-config startup-config) 事で正確な補正値が初期ファイルに設定され, 再起動をしても正確な時計として利用できる.

15.5.2 基本設定

Cisco ルータにおける NTP の基本設定を説明する. クライアントモードを使って時刻を合わせる場合, ntp server コマンドを使用する. key オプションで対称鍵の鍵 ID を source オプションで送信する IP アドレスを指定する (ルータはネットワークインタフェースが複数あるため). NTP パケットの発信 IP アドレスは ntp source コマンドで設定しても良い.

```
Router(config)# ntp server 1.1.1.1 key 10 source Ethernet 1/1
```

Symmetric モードでの使用も可能で, その場合は ntp peer コマンドを使用する. オプションは ntp server コマンドと同じである.

Cisco ルータはブロードキャストモードもサポートしている. サーバにもクライアントにも設定できるが, ブロードキャストモードを指定する場合, ブロードキャストを送受するインタフェースで設定を行う. サーバの場合,

```
Router(config)# interface Ethernet 1/1
Router(config-if)# ntp broadcast
```

クライアントの場合,

```
Router(config)# interface Ethernet 2/1
Router(config-if)# ntp broadcast client
```

のように設定する. なお, ルータで NTP を利用したくない場合, NTP パケット受け取りたくないインタフェースで ntp disable コマンドを設定する.

```
Router(config)# interface Ethernet 3/1
Router(config-if)# ntp disable
```

NTP の時計とルータのハードウェアクロックを同期させる場合, 次のように設定する.

```
Router(config)# ntp update-calendar
```

15.5.3 アクセス制御

NTP のアクセス制御はアクセスリストベースで可能である. 設定には `ntp access-group` コマンドを用いる. 制限内容として以下の 4 つを指定できる.

- `peer` — 時刻要求, NTP 制御クエリ, ルータが他の NTP サーバと時刻同期すること, いずれも許可する.
- `serve` — 時刻要求と NTP 制御クエリは許可するが, ルータが他の NTP サーバと時刻同期することは許可しない.
- `serve-only` — NTP の時刻要求のみを許可する.
- `query-only` — NTP の制御クエリのみを許可する.

```
Router(config)# ntp access-group serve 10
```

15.5.4 認証機能

Cisco は対称鍵による認証機能のみをサポートする.

```
Router(config)# ntp authenticate
Router(config)# ntp authenticate-key 10 md5 abcdefg
Router(config)# ntp authenticate-key 11 md5 zyxwvut
Router(config)# ntp trusted-key 11
```

15.5.5 ハードウェアクロックを時刻源にする

Cisco ルータは NTP を有効にし, リファレンス時計とのアソシエーションが確立され, 時刻が安定すれば NTP サーバとして利用できる. リファレンス時計が参照できない環境だっ

たり, リファレンス時計から時刻が供給されなくなってもサーバとして時刻を供給できるようにしたい場合, 自身のハードウェアクロックを使って NTP サーバとして動作させされるよう, `ntp master` コマンドが用意されている[5].

コマンドの後に数字を指定することで自走した場合のストラタム値を指定することもできる. ストラタム 3 で動作させる場合, 次のように設定する (デフォルトは 8).

```
Router(config)# clock calendar-valid
Router(config)# ntp master 3
```

15.5.6 Cisco ルータの NTP 状態表示コマンド

Cisco ルータでの NTP 状態表示コマンドには次のようなものがある.

コマンド	目的
`show calendar`	ハードウェアクロックを表示する
`show clock [detail]`	ソフトウェアクロックを表示する
`show ntp associations [detail]`	アソシエーションを表示する
`show ntp status`	NTP の状態を表示する

15.5.7 Cisco 7200 と Trimble Palisade を使ったストラタム 1 サーバ

Cisco 7200 シリーズの Cisco IOS 12.1(1)T で, Trimble Palisade NTP キットがサポートされており, Cisco ルータに直接時刻情報を取り込んでストラタム 1 として動作させることができる.

設定方法は, Paliside のシリアルポートと Cisco 7200 の AUX ポートを接続し, 次のコマンドを入力するだけである. うまく動かなければ, `debug ntp refclock` でデバッグを行うことができる.

[5] Cisco 7200 で時刻源 (GPS) を直接接続する場合にも利用する (15.5.7 節参照).

```
Router(config)# ntp master
Router(config)# ntp update-calendar
Router(config)# line aux 0
Router(config-line)# ntp refclock trimble pps none
Router(config-line)# exit
Router(config)# exit
Router# show ntp assoc

      address      ref clock   st  when  poll reach  delay  offset   disp
*~127.127.8.1    .GPS.         0      5    32  377    0.0    -0.01    0.0
 * master (synced), # master (unsynced), + selected, - candidate, ~ configured
```

`ntp refclock` コマンドには, PPS の入力をサポートしているが (CTS と RI ピンのどちらかの入力を受け付ける), Symmetriccom 社の Telecom Solutions シリーズであれば, 次のような設定で PPS 信号を受けられるようである.

```
Router(config)# ntp master
Router(config)# ntp update-calendar
Router(config)# line aux 0
Router(config-line)# ntp refclock telecom-solutions pps cts stratum 1
```

より詳細な情報は下記からダウンロードできる.

`http://www.cisco.com/en/US/products/sw/iosswrel/ps1834/products_feature_guide09186a008007fef8.html`

第16章　FAQ

16.1　時間

Q: UTC とは何ですか?

A: UTC(英 Universal Time Coordinated, 仏 Temps Universel Coordonné) は協定世界時と呼ばれる世界共通の標準時で, 世界中のどこでも使うことができる時間である.

1967 年に開催された第 13 回国際度量衡会議において, 1 秒の定義をセシウム 133 が 91 億 9263 万 1770 回振動する時間と定義され, これは UTC の 1 秒と同じである.

Q: UTC と GMT の違いは何ですか?

A: GMT(Greenwich Mean Time) グリニッジ標準時は, 伝統的に経度が 0 度と定められているイギリスのロンドにあったグリニッジ天文台での平均太陽時で, UTC が定められる以前は世界共通の標準時として広く使われていた.

太陽時 (Solar Time) は太陽が天球上で最も高い位置に達した時刻を正午とする時間である. ところが地球は楕円軌道を描くため, 正午に太陽が子午線を通過する時間は一定ではない. 一定ではない時間を平均し, 正午に常に子午線を通過するよう仮想的な太陽を設定しこれに基づく太陽時を平均太陽時と呼び, GMT は平均太陽時に基づいている.

GMT が世界共通の標準時として使われるようになったのは歴史的経緯による. イギリスの船員は航海中グリニッジ子午線からの経度差を計算するため, 自分達の時計を GMT に合わせていた. この習慣に加え, 他の九人ではネヴィル・マスケリンによる月距法 (グリニッジでの天体観測データと船上での月の位置の観測から経度を求める) が結び付き, GMT は世界中でどこでも用いられる基準時刻として使われるようになった.

地球の自転が不規則であるため, 安定した時間を得るため原子時計を基準にするようになり, 世界基準時は 1972 年 1 月 1 日に GMT から UTC に置き換えられた. UTC 以前の慣習により標準時刻を今でも GMT と呼ぶ場合がある.

なお, グリニッジ天文台は 1998 年 10 月に天文台としての活動を終えている.

Q: 閏秒とは何ですか?

A: 地球の自転周期は一定していないため, 原子時計が刻む国際原子時 (TAI) と地球の回転を元に決められている世界時 (UT1) との差が 1 秒未満となるよう, 必要に応じて閏秒を削除あるいは挿入して維持されている.

16.2 コンピュータの時間

Q: どのように時間を合わせているのすか?

A: コンピュータは起動時にハードウェアの時計を参照し, オペレーティングシステムが起動後はシステム時計 (ソフトウェア時計あるいはカーネル時計とも呼ばれる) を使って時を刻んでいる.

コンピュータが持つ時計は内部的にはエポックと呼ばれるカウンタを使っていて, ある時刻をエポック 0 している. そして, エポックのカウンタを一定間隔で増加させることで, エポック値から時刻をを割り出している. カウンタを一定間隔で増加させるのに用いるのが, 周期タイマーの割り込みから生成されている. この割り込みが拾えないと内部の時計は狂ってくる. また, 割り込み間隔が時計の分解能となる.

Q: コンピュータが持つ時計の精度は?

時計の精度は色々な要素から判断できる. まず, 正確に時を刻むのが時計と考えると, カウンタの間隔にずれが無いことが重要である. このずれは PPM(Part Per Million) で示すことができる. 大まかに 1 日に 1 秒ずれる場合, 12PPM となる. 逆に 500PPM と示されていれば, 1 日に 43 秒ずれることになる.

また, 時間をどのくらい細かく刻めるかも重要でこれが分解能になる. 分解能を高めるためには細かく一定間隔でカウンタを刻めるもの (周波数が高いと置き換えても良い) が必要である. 現在は原子の振動を使っている.

Q: CMOS クロックの精度

CMOS クロックの精度を正確に測った人はいないが, ある人によると PC の CMOS クロックは 12PPM 程度といわれている. これは 1 日に 1 秒ずれる計算になる. 一般的な水晶時計は月差 30 秒程度であるので, ほぼ水晶時計と同じと考えてよいだろう.

Q: ノート PC の時刻のずれが気になる

A: ノートパソコンのように CPU クロックが変動したり, 頻繁にサスペンドする環境では, タイマーの割り込みをカウントすることで時刻を管理している多くの Unix 系オペレー

ティングシステムでは, 時刻が大幅にずれてしまう. また, ntpd を走らせていたとしても, ntpd 自身時刻同期を常に行うことを前提としているため, NTP のソフトウェア時計が狂ってくる.

ntpd ではバーストモード特に iburst を使うことで, ネットワークが接続されてすぐにサーバと時刻同期を行うことができる.

時刻のずれは VMware などの仮想マシンソフトウェアでも起こる. これは仮想マシンでは時刻会わせを行うためのタイマー割り込み処理の頻度が少ないため, 時刻のずれが生じてくる.

16.3 NTP

Q: NTP とは何ですか?

A: NTP は Network Time Protocol の略で, コンピュータの時刻をリファレンス時計に同期させるために使われるインターネットプロトコルである. NTP はインターネットの標準プロトコルで, 元はデラウェア大学の David L. Mills 教授が開発した.

Q: SNTP とは何ですか?

A: SNTP (Simple Network Time Protocol) はプロトコル手順としては, NTP と同じだが, 様々な機能で必要となる内部アルゴリズムを持っていない. 一般的にクライアント向けのプロトコルである.

Q: なぜ時間を同期させる必要があるのですか?

A: 時間の同期は様々な理由で必要である.

Q: NTP の基本的な特徴を教えて下さい

A: NTP の特徴を下記に示す.

- NTP は “true time” を規定するリファレンス時計が必要で, 全ての時計はこの “true time” に合わせられる. NTP はリファレンス時間に UTC を使用する.
- NTP はフォールトトレラントなプロトコルで自動的に最適な時刻源を選択し同期を行う. 複数の候補があれば, 累積エラーを最小化するため, コンバインすることができる.
- NTP は高い拡張性を持つ.

- 様々な時刻源を利用できる. プロトコルとしてはナノ秒よりも小さい分解能 (2^{-32} 秒) を使ってより精度を高められる (RFC 868 で規定されている `rdate` の分解能は 1 秒しかない).
- 一時的にネットワークが利用不可になっても, 過去のデータを使って現在時刻とエラーを見積もることができる.
- NTP はローカル時間の精度を見積もることもできる.

Q: どんなオペレーティングシステムでサポートされていますか?

A: ほとんどの Unix 系オペレーティングシステム, VMS, Windows で動作する.

Q: インターネットにどのくらいの NTP サーバがありますか?

1999 年に調査された報告書によると, 175,000 台のホストで NTP が動作していたとの事である. 現在はその頃よりもインターネットが発達し, NTP がインターネットの標準プロトコルとなった今, その数は調査時よりも何百倍も増えていると考えてよいだろう.

Q: どのバージョンの NTP を使うべきですか?

現在, NTP はバージョン 3 と 4 があり, 最新の NTP ソフトウェアはバージョン 4 を使っているが, 公式なインターネットプロトコルは未だにバージョン 3 のままである.
TBD

Q: xntp と ntp の違いを教えて下さい

NTP バージョン 4 を実装した **ntp** が出る前にバージョン 3 を実装したソフトウェアの名称が `xntp` だったという点がある.
それから, Dennis Fergusson によると “x” は実験 (experimental) という意味があるそうだ.

Q: バージョン 4 はどう変わったのですか?

詳しくは NTP Version 4 Release Notes を見て頂きたいが, 次の点が変更されている.

- 固定小数点演算の代わりに浮動小数点演算を利用している.
- クロックディシプリンアルゴリズムを再設計している.
- ナノカーネルをサポートしている.
- Autokey と呼ばれる公開鍵暗号機能がある.

- 自動的にサーバを発見するメカニズム (メニーキャスト) がある.
- 起動時やネットワーク障害後に高速に同期する機能がある (バーストモード).
- リファレンス時計のドライバを多数用意した.
- 多数のオペレーティングシステムをサポートしている.

NTP の仕様はどこにありますか?

NTP バージョン 3 については RFC 1305, SNTP バージョン 4 については RFC 4330. NTP バージョン 4 については現在ドラフト段階で, これが RFC 1305 と 4330 を包含する予定である.

Q: NTP の歴史を教えて下さい

4.3 節 (33 ページ) を参照.

Q: 実装状況を教えて下さい

TBD

16.4 どのように動いているのか

Q: リファレンスクロックとは何ですか?

NTP サーバの時刻源となる時計で, 原子時計, 電波時計, GPS などを指す.

Q: ストラタム 1 とは何ですか?

正確なリファレンスクロックから時刻情報を得た NTP サーバを言う.

Q: 時間の同期方法を教えて下さい

TBD

Q: NTP 時間とは何ですか? (他の時間への変換方法)

TBD

Q: いつサーバに問い合わせるのですか?

TBD

Q: どのくらいの精度があるのですか?

TBD

Q: How frequently will the System Clock be updated?

TBD

Q: How frequently are Correction Values updated?

TBD

Q: How reliable are those Error-Estimates?

TBD

Q: クライアント数の制限はありますか?

TBD

Q: ストラタムとは何ですか?

TBD

Q: 同期ループ状態を避けるにはどうしていますか?

TBD

Q: minpoll/maxpoll の設定範囲は?

TBD

Q: 最善のポーリング間隔は?

TBD

Q: How will NTP discipline my Clock?

TBD

16.5 NTP の設定

Q: ntpdate だけでもいいですか?

TBD

Q: 最小の設定を教えて下さい

TBD

Q: リファレンスクロックを指す擬似アドレスは何ですか?

TBD

Q: ドリフトファイルとは何ですか?

TBD

Q: ホスト名, IP アドレスどちらにすべきでしょうか?

TBD

Q: ログ情報を記録したいのですが, どのように設定すれば良いですか?

TBD

Q: 時計の初期同期を早くしたいのですが, どのように設定すれば良いですか?

TBD

Q: 遠隔管理をしたいのですが、どのように設定すれば良いですか?

TBD

Q: 認証鍵はどのように使うべきですか?

TBD

Q: autokey はどのように使うべきですか?

TBD

Q: 公開サーバを使う場合のエチケットを教えて下さい

TBD

Q: どのように公開サーバを見つければよいですか?

TBD

Q: サーバの数にルールはありますか?

TBD

Q: サーバは primary と secondary を混ぜるべきですか?

TBD

Q: 大きなネットワークで NTP サービスをどうやって提供すべきでしょうか?

TBD

Q: 認証を使うべきでしょうか?

TBD

Q: autokey はどのように使うのですか?

TBD

Q: ブロードキャスト/マルチキャストはどのように使うのですか?

TBD

Q: メニーキャストはどのように使うのですか?

TBD

Q: PPS を使って精度を上げるには何が必要ですか?

TBD

Q: 公開 NTP サーバを立ち上げたいのですがどうすればよいですか?

TBD

16.6 リファレンスクロック

Q: ローカルクロックとは何ですか?

TBD

Q: GPS の PDOP, TDOP, GDOP とは何ですか?

TBD

Q: NMEA とは何ですか?

TBD

Q: TSIP とは何ですか?

TBD

Q: NTP で使えるリファレンスクロックはどのように見つければいいですか?

TBD

16.7 トラブルシューティング

Q: NTP がちゃんと動いているかを知る方法を教えて下さい

TBD

Q: peerstats と loopstats はどのように使いますか?

TBD

Q: クライアントとサーバの時間差はどのようにわかりますか?

TBD

Q: ステータスのコードを教えて下さい

TBD

Q: statistics files はどのように使いますか?

TBD

関連図書

[1] トニー・ジョーンズ著, 松浦俊輔訳: “原子時間を計る〜300 億分の 1 秒物語〜”, 青土社, 2003

[2] David L. Mills: “Network Time Protocol (NTP)”, RFC 958 ,September 1985.

[3] David L. Mills: “Network Time Protocol (Version 2) Specification and Implementation”, RFC 1119, September 1989.

[4] David L. Mills: “Measured Performance of the Network Time Protocol in the Internet System”, RFC 1128, October 1989.

[5] David L. Mills: “Internet Time Synchronizaation: the Network Time Protocol”, RFC 1129, October 1989.

[6] J. Crowcroft, J. Onions: “Network Time Protocol (NTP) over the OSI Remote Operations Service”, RFC 1165, June 1990.

[7] David L. Mills: “Network Time Protocol (Version 3) Specification, Implementation and Analysis”, RFC 1305, March 1992.

[8] David L. Mills: “Simple Network Time Protocol (SNTP)”, RFC 1361, August 1992.

[9] C. Partridge, T. Mendez and W. Milliken: “Host Anycasting Service”, RFC 1546, November 1993

[10] David L. Mills: “A Kernel Model for Precision Timekeeping”, RFC 1589, March 1994.

[11] D. Gowin: “NTP PICS PERFORMA For the Network Time Protocol Version 3”, RFC 1708, October 1994.

[12] David L. Mills: "Simple Network Time Protocol (SNTP)", RFC 1769, March 1995.

[13] David L. Mills: "Simple Network Time Protocol (SNTP) Version 4 for IPv4, IPv6 and OSI", RFC 2030, October 1996.

[14] H. Krawczyk, M. Bellare and R. Canetti: "HMAC: Keyed-Hashing for Message Authentication", RFC 2104, February 1997.

[15] J. Mogul, D. Mills, J. Brittenson, J. Stone, and U. Windl: "Pulse-Per-Second API for UNIX-like Operating Systems, Version 1.0", RFC 2783, March 2000.

[16] H. Prafullchandra and J. Schaad: "Diffie-Hellman Proof-of-Possession Algorithms", RFC 2875, July 2000.

[17] V. Kalusivalingam: "Simple Network Time Protocol (SNTP) Configuration Option for DHCPv6", RFC 4075, May 2005.

[18] P. Hoffman, B. Schneie: "Attacks on Cryptographic Hashes in Internet Protocols", RFC 4270, November 2005.

[19] D. Mills: "Simple Network Time Protocol (SNTP) Version 4 for IPv4, IPv6 and OSI", RFC 4330, January 2006

[20] http://www.rfc-editor.org/cgi-bin/errata.pl

[21] David L. Mills: "Proposed authentication enhancements for the Network Time Protocol version 4", Electrical Engineering Report 96-10-3, University of Delaware, October 1996, 36 pp.

[22] David L. Mills: "Public key cryptography for the Network Time Protocol", Electrical Engineering Report 00-5-1, University of Delaware, May 2000. 23 pp

[23] David L. Mills: Public Key Cryptography for the Network Time Protocol Version 2, IETF Draft `draft-ietf-stime-ntpauth-04`, November 2002.

[24] J. Burbank, W. Kasch, J. Martin, D. Mills: " Network Time Protocol Version 4 Protocol And Algorithms Specification", `draft-ietf-ntp-ntpv4-proto-04`, January 17 2007.

[25] Trimble: “Acutime 2000 Synchronization Kit User Guide”, May 2002.

[26] Philip M. White: Using a Garmin GPS 18 LVC as NTP stratum-0 on Linux 2.6, http://time.qnan.org/, Mar 11 2007.

[27] ブルース・シュナイア著, 山形浩生訳: “暗号技術大全”, ソフトバンクパブリッシング, 2003.

[28] ふなばただよし: プログラミングと暦,

http://www.funaba.org/programming-and-calendar.html

[29] 株式会社ユニゾン: 図解雑学 GPS のしくみ, ナツメ社, August 2003.

索 引