割込みハンドラと高精度PCによるソフトウェアタイムスタンプの精度改善

町澤　朗彦[†a)]　　北口　善明[†]

Improvement of Software Timestamp Accuracy with Interrupt Handler and High Precision PC

Akihiko MACHIZAWA[†a)] and Yoshiaki KITAGUCHI[†]

あらまし　ネットワーク計測では，タイムスタンプが重要な物差しとなっており，PCの汎用性，経済性，扱いやすさなどから，ソフトウェアタイムスタンプには利点が多いが，専用ハードウェアによるタイムスタンプに比べて，精度が劣るといわれてきた．本論文では，PCのソフトウェア及びハードウェアを詳細に解析し，GPSを基準とする専用ハードウェアタイムスタンプよりも高精度なソフトウェアタイムスタンプを非RT Linux上に実現した．できる限りネットワークに近い部分でPCC（Processor Cycle Counter）をタイムスタンプとして取得し，リソース競合を防ぐためのシステムチューニング及びパケットフィルタを施している．これらの技術を組み合わせることにより，標準偏差34 nsで，ネットワークの片道遅延を測定することができた．長期間にわたる精度の維持も，高精度PCに高精度周波数源を外部接続することによって可能である．更に，ハードウェアコンポーネントのジッタ解析を行い，本手法がPCの能力をアーキテクチャの限界近くまで引き出していることを示した．

キーワード　タイムスタンプ，ネットワーク計測，PCアーキテクチャ，PC時計

1. まえがき

ネットワーク計測では，タイムスタンプが重要な物差しとなっているが，これまで，PCのソフトウェアによるタイムスタンプの精度は不十分として，DAG project [1] などの専用ハードウェアが開発されてきた．専用ハードウェアのタイムスタンプ精度は，GPSを接続することにより100 ns程度となっている．しかし，近年のPCの性能改善を背景に，PCによるソフトウェアタイムスタンプの精度を改善するための研究が進められている．ソフトウェアタイムスタンプの利用は，PCの汎用性により，パケットキャプチャやアクティブ計測でのパケット送出など様々な用途に広がっているが，帯域推定など，新たなアルゴリズムの開発も容易であり，多数のネットワーク計測ソフトウェアツールが使われている．また，特別なハードウェアを必要とせず，安価なため，手軽に使用することができるなど，ソフトウェアタイムスタンプには利点が多い．

Pásztorらは，ネットワークインタフェース（NIC）のデバイスドライバコードでPCC（Processor Cycle Counter）を使うことにより，PCでも高精度なタイムスタンプを実現できることを示し，スレッド切換に伴う突発的な遅延増大を，リアルタイムOS（RT-Linux）を用いて低減し，1マイクロ秒程度のタイムスタンプ精度を実現しているが [2]，専用ハードウェアによるタイムスタンプの精度よりも1けた以上性能が低い．これは，RT-Linuxがスケジューリング精度を高めるために割込みを保留しているが，PCがパケットを受信する最初のステップはNICからの割込みであり，割込み保留により精度が低下していると考えられる．しかも，PCに搭載されている水晶振動子の温度安定度が悪いため，長時間にわたる精度の維持も不可能である．

本論文では，パケット受信時に発生するCPU割込みの際のPCCをタイムスタンプとするとともに，カーネルとユーザランド間のプロセス切換時間に基づいた

† 情報通信研究機構，小金井市
National Institute of Information and Communications Technology, 4–2–1 Nukui-Kitamachi, Koganei-shi, 184–8795 Japan
a) E-mail: machi@nict.go.jp

パケットフィルタリングを用いることによって，ソフトウェアタイムスタンプの精度を改善する手法について提案する．割込みハンドラは，パケット到着後に最初に実行されるソフトウェアステップであり，「高精度タイムスタンプは，よりネットワークに近くで」の原則を実現している．また，水晶振動子の周波数変動の問題を解決するために，我々が開発を進めている高精度 PC [3] を用いている．高精度 PC は，マザーボード上の水晶を，より高精度な発振子に置き換えることによって，長期間にわたって CPU の動作周波数を一定に保つことができる．置き換える発振子の種類によって，得られる精度は異なり，ネットワーク計測では，OCXO あるいは Rb 基準信号発生器などが適している [4]．また，文献 [4] で用いている CPU の動作周波数評価では，2 台の PC のクロックずれを評価しているが，本論文では，同手法を用いてソフトウェアタイムスタンプの精度を評価する．まず，**2.** で，ソフトウェアタイムスタンプの精度を改善するための手法を提案し，**3.** において，提案手法の効果を実証する．更に，**4.** では，得られたソフトウェアタイムスタンプのジッタを，PC を構成している各ハードウェアコンポーネントごとに解析した．本手法を用いることにより，標準偏差 34 ns 精度のソフトウェアタイムスタンプを実現することができた．これは，GPS を用いたハードウェアタイムスタンプをしのぐ精度であり，今後，様々な応用が期待される．

なお，本論文で対象としているタイムスタンプは，UTS 等の標準時刻との間に，固定的なオフセットを有しているが，ネットワーク計測では，多くの場合，固定的なオフセットは問題とならず，変動成分が重要であることが多い．

2. ソフトウェアタイムスタンプ精度改善手法

まず，従来手法をまとめ，次に従来法の欠点を克服するための改善手法を提案する．

2.1 従来方式

PC では NIC にパケットが到着すると，割込み要求を発生する．CPU は割込み要求を検知すると，直ちに OS（オペレーティングシステム）内の割込みハンドラ関数を呼び出し，割込み番号が NIC に割り当てられている番号であることを確認した後，NIC のデバイスドライバへジャンプする．さて，2 台の PC をネットワークケーブルで直結したときのパケットの流れを図 1 に示す．時刻 T1 に送信側（TX）PC のユーザランドで生成されたパケットは，時刻 T2 にカーネル内の NIC デバイスドライバを経て送出される．一方，受信側（RX）PC では，NIC にパケットが到着し，割込みハンドラが起動された時刻を T3，割込み処理である NIC デバイスドライバを経た後，最終的に，時刻 T4 にユーザランドに達する．従来，高精度なタイムスタンプを得るためには，送受信ともに NIC のデバイスドライバでタイムスタンプを取得してきたが，パケット到着後に最初に実行されるソフトウェアステップは割込みハンドラであり，「高精度タイムスタンプは，よりネットワークに近くで」の原則を実行していない．

なお，図 1 において，$S()$ はパケット送受信に要する時間経過を区間（segment）に分けており，$S(U > K) = T2 - T1$ などを示し，$U > K$ は，「ユーザランドからカーネル」を意味している．

一方，Linux 等のマルチユーザマルチタスク OS は，システムビジー状態におけるプロセス切換で待ち時間が生じ，突発的にタイムスタンプ精度が劣化する．Pásztor らは，RT-Linux を用いて突発的な遅延時間の増大を防いでいる [2]．RT-Linux では，予測不能な割込みを保留してスケジューリング精度を高めているが，割込み要求を保留することにより，突発的な遅延時間増大を防ぐことができても，逆に，システムアイドル時の割込み応答精度は低下してしまう．

また，上記以外にも，タイムスタンプとしての 64 ビットのプロセッササイクルカウンタ（PCC），高精度 PC，不要プロセスの停止，ディスクキャッシュ頻度の低減，送信側では NIC デバイスドラバでのタイムスタンプ取得，などもソフトウェアタイムスタンプを高精度化するために用いられている．

PentiumII 以降のインテルプロセッサには，Time

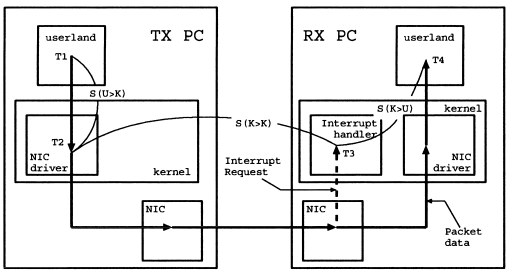


図 1　パケットの流れ
Fig. 1　Packets flow.

Stamp Counter と呼ばれる 64 ビットの PCC が設けられ，RDTSC 命令により PCC を読み出すことができるようになっている．この，PCC は，電源投入時に 0 にリセットされ，CPU の 1 サイクルごとにカウントアップされるため，CPU の動作周波数の解像度で時を刻むことができる．さて，PCC 値 c から時刻 t への変換は，時刻 t_0 での PCC 値 c_0 が与えられれば，次式で求めることができる．

$$t = t_0 + (c - c_0)/r \quad (1)$$

ただし，r は CPU の動作周波数である．しかし，式 (1) では，動作周波数を一定値としているが，実際の PC は数十 ppm 精度の水晶に駆動されており，時間経過とともに，周波数が変動してしまう．そこで，本論文では，長期にわたる精度を維持するために，PCC のもととなる 14.318 MHz の水晶を，高精度な振動子に置き換えた高精度 PC を用いる．

2.2 改善方式の提案

本論文では，まず，「高精度タイムスタンプは，よりネットワークに近くで」の原則を実行するために，パケット到着後，最初のソフトウェアステップである割込みハンドラでタイムスタンプ $T3$ を取得する手法を提案する．したがって，本論文では，ネットワークの遅延時間を，区間 $S(K > K) = T3 - T2$ となる．

更に，本論文では，マルチタスク OS によるリソース競合問題を回避するために，システムビジーを検知することによって，タイムスタンプの精度を推定し，精度が確保できない場合には，当該パケットを破棄する手法を提案する．システムアイドルの状態では，プロセス切換はスムーズに行われるため，パケットを送受信するユーザランドプロセスとカーネルプロセスとの間の時間 $S(U > K)$ 及び $S(K > U)$ は，ほぼ一定の短時間となる．このプロセス間時間を監視することにより，システムビジーを推定する．システムビジー状態で得られたタイムスタンプは不正確となるため，システムビジー状態に送受されたパケットは不適格パケットとして除外（フィルタリング）する．

なお，本論文では，提案手法に加えて，タイムスタンプとしての 64 ビットのプロセッササイクルカウンタ（PCC），高精度 PC，不要プロセスの停止，ディスクキャッシュ頻度の低減，送信側では NIC デバイスドラバでのタイムスタンプ取得，なども併せて利用する．

3. 性能評価

本章では，提案手法の効果について評価する．まず，評価する測定系について述べた後，個々の技術についてジッタの改善度を計測した．次に，パケットサイズ及びパケット送出間隔の影響を測定し，最後に，提案手法を使ったソフトウェアタイムスタンプの精度とハードウェアタイムスタンプの精度を比較する．

3.1 タイムスタンプ精度評価方法

本論文では，タイムスタンプの精度を評価するために，ネットワーク遅延が一定となるよう，2 台の PC をケーブルで直結し，一方のタイムスタンプをネットワークを介して他方に送って比較する．この構成は，図 1 の接続そのものである．この場合，ネットワーク上でジッタは生じないため，二つのタイムスタンプの時系列から最小二乗化法により求められた回帰直線からの残差が PC 内部で発生したジッタとなる [4]．

タイムスタンプは IPv4/UDP パケットのペイロード（データ）部にデータを上書きする形で挿入する．これは，**4.1** で，NIC の送受信処理時間特性を切り分けて測定する際に，PC 間に挿入したハードウェアタイムスタンパによるタイムスタンプ追加によって，パケットサイズが変化することを防ぐことを目的としている．例えば RFC781 [5] では，IP ヘッダにタイムスタンプを付与するオプションを規定しているが，タイムスタンプの追加により，パケットサイズが長くなってしまう．しかし，Prasad 等はパケットサイズが異なると，ルータやスイッチなどの store-and-forward 型ネットワークデバイスでは，バッファリング時間が異なることにより，遅延時間も変化してしまうことを指摘しているが [6]，Jin らは，PC のシステムパフォーマンスが帯域推定に及ぼす影響について解析しており，ネットワークデバイスと同様のバッファリング時間が遅延時間に影響を与えることを示している [7]．そこで，本論文では，ペイロード部のデータを上書きすることにより，タイムスタンプ挿入に伴うパケットサイズの変化を防いでいる．なお，ここでは IPv4/UDP を用いているが，実際の応用にあたっては，提案手法を他のプロトコル（IPv6, TCP, ICMP, etc.）に容易に適用することが可能である．また，パケットキャプチャなどに際しては，IP パケットに乗せて伝送する必要がないため，ペイロードデータを上書きすることなく，任意のパケットのキャプチャ時間のタイムスタンプを得ることができる．

計測に用いたパケットはペイロードサイズ 64 バイト，パケット間隔 10 ミリ秒で，100 秒間計測した．パケットサイズや送出間隔のタイムスタンプ精度への影響については，**3.4** で考察する．また，使用した PC の諸元を表 1 に示す．マザーボードに Supermicro P4DPR-iG2（CPU: Intel Xeon 2.4 GHz, Chip set: Intel E7500）に高精度化の改造を施し [3]，外部から Cs 原子時計により駆動し，オペレーティングシステムには Linux 2.4.18 を用いている．また，ネットワークインタフェースは，ジッタ特性（**4.1** 参照）を考慮し，送出側には，Intel Pro/1000 を用い，受信側には ADMtek AN983(B) を用いている．以後，特に断らない限り，同じ測定条件を用いる．

3.2 割込みハンドラでのタイムスタンプの利用

まず，図 2 で，ネットワーク遅延計測に関し，ユーザランド内で PCC 値を取得した場合と，ネットワークインタフェースのデバイスドライバコード内で PCC 値を取得した場合のジッタをヒストグラムにより比較する．ただし，ヒストグラムのビン幅は 16 ns としている．ユーザランドではジッタが大きいだけではなく，800 ns から 1000 ns 付近にかけて，セカンドピークが存在している．しかし，ジッタはたかだか 1000 ns 程度であり，後述するリソース競合に伴う処理待ち時間に比べると小さい．なお，図 2 では，分布の広がりを見るために，それぞれのヒストグラムの立上りを x 軸の原点にそろえている．

次に，図 3 で，ネットワーク遅延計測に関し，受信時に NIC のデバイスドライバでタイムスタンプを取得した場合と，割込みハンドラでタイムスタンプを取得した場合のジッタをヒストグラムにより比較する．ただし，ヒストグラムのビン幅は 60 ns としている．ジッタの標準偏差で比較すると，NIC のデバイスドライバ間では 53 ns であるのに対し，受信時に割込みハンドラでタイムスタンプを取得することにより 34 ns まで低減することが可能である．

3.3 システムチューニングとパケットフィルタリング

Linux はマルチユーザマルチタスク OS であるため，プロセス間で，リソースの競合が生じる．本測定では，単に不要プロセスを止めるだけではなく，ディスクキャッシュのフラッシュ頻度を最大限低減させている．しかし，他のプロセスを完全に停止させることはできないため，極少数のパケットではあるが，他のプロセスの影響を受けたパケットは，見かけ上の大きな遅延が生じてしまう．このようなパケットは，不適格サンプルとして，フィルタリング処理により破棄する．

表 1　測定に用いた PC の諸元
Table 1　Specification of the PC.

Mother board	Supermicro P4DPR-iG2
CPU	Intel Xeon 2.4 GHz
Network controller #1	Intel 82546EB drv e1000-5.2.16
Network controller #2	ADMtek AN983(B) drv tulip-0.9.14
Network controller #3	Intel 82550EY drv e100-2.1.19
OS	linux 2.4.18

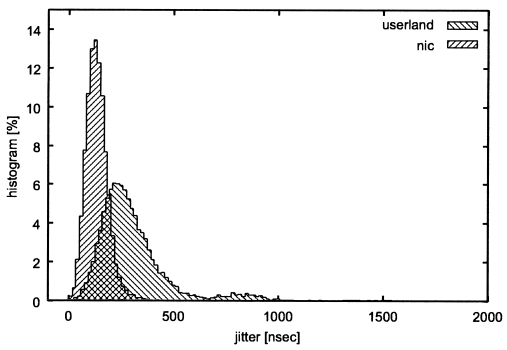


図 2　ユーザランドと NIC デバイスドライバコードでのタイムスタンプの比較
Fig. 2　Comparison of timestamping in userland and NIC device driver code.

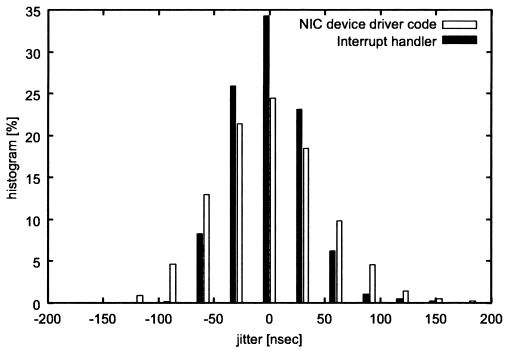


図 3　提案方式によるソフトウェアタイムスタンプのジッタ分布
Fig. 3　Jitter distribution of applied software timestamping.

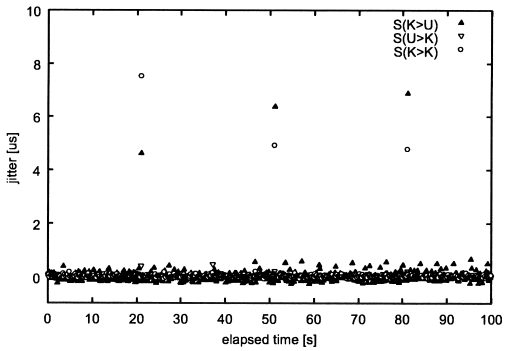


図 4　不適格データフィルタ処理を施さない時系列データ
Fig. 4　Raw data without nonapproved data filtering.

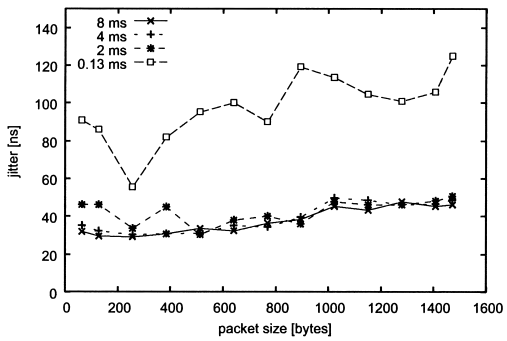


図 5　パケットサイズ及び送出間隔に対するジッタ特性
Fig. 5　Jitter characteristics for packet size and interval.

不適格サンプルも含めた，遅延ジッタの例を図 4 に示す．不適格サンプルの生起頻度は小さく，ヒストグラムでは表現困難なため，全サンプルを時系列に表示する．この場合，不適格サンプルの割合は，全 10000 サンプルに対して，3 サンプル（0.03%）のみである．

他プロセスとのリソース競合は，カーネルとユーザランド間の切換時間にも影響するため，カーネルとユーザランドにおけるタイムスタンプの変動からリソース競合をある程度，検出可能である．図 4 に，カーネルとユーザランド間のジッタも加えて表示している．図 4 の例では，$S(K > K)$ で，大きなジッタが生じたときには，$S(K > U)$ でも，大きなジッタが生じていることが分かる．この特性を用いることにより，パケット受信時に，そのパケットの割込みハンドラでのタイムスタンプとユーザランドでのタイムスタンプの差が，一定値以上の場合，そのパケットのカーネルタイムスタンプを不適格として取り除くフィルタ処理を施す．今回は，図 2 により，平均遅延時間より 2 マイクロ秒以上遅延時間の長いパケットを不適格とした．

3.4　パケットサイズ及び送出間隔の影響

ここで，用いるパケットのサイズ及びパケット間隔の影響について検討する．パケット間隔を 0.13 ms から 8 ms とした場合について，パケットサイズに対するジッタの変化を図 5 に示す．横軸にパケットサイズ，縦軸にジッタの標準偏差を表す．図 5 より，パケットサイズが 1500 バイトに近く，パケット間隔が 2 ms 以下で短くなるほど，精度が劣化することが分かる．しかし，1500 バイトサイズパケットのポート占有時間である 0.13 ms 間隔としても，ジッタの標準偏差は 130 ns 程度の精度を保っている．

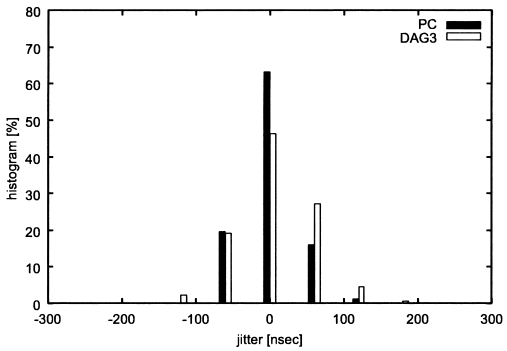


図 6　提案手法によるソフトウェアタイムスタンプと DAG3 システムの 24 時間継続計測時のジッタ分散比較（参考文献 [1] Fig. 1 を改編）
Fig. 6　Jitter distribution for 24-hour period: PC vs. DAG3 (modified reference [1] Fig. 1).

3.5　ハードウェアタイムスタンパとの精度比較

本章で用いたすべての手法を利用することにより，タイムスタンプのジッタは，図 6 のような分布となる．外部周波数源にはコストを考慮して Cs 原子時計ではなく，Rb 基準信号発生器を用いている．また，GPS を基準とする専用ハードウェア DAG3 のタイムスタンプ精度（文献 [1] Fig. 1）と比較する．データは，毎秒 1 パケットの頻度で 24 時間にわたり測定し，ヒストグラムのビン幅を 60 ns 幅としている．PC によるタイムスタンプの精度は，GPS を利用したハードウェアタイムスタンプよりも分散が小さく，高精度であることが分かる．

4.　ハードウェアコンポーネントのジッタ解析

本章では，PC を構成している各コンポーネントに

起因するジッタを解析し，前章で得られた精度が，ハードウェアアーキテクチャの限界近くまで達していることを示す．

4.1 NIC 依存性

送受信におけるジッタには NIC 依存性がある．本節では，2 台の PC 間に，精度 11 ns のハードウェアタイムスタンパ(注1)を挟み込み，送信側 NIC デバイスドライバコードとハードウェアタイムスタンパの間及び，ハードウェアタイムスタンパと受信側割込みハンドラ間の遅延ジッタを測定した．精度 11 ns は，100Base-T のクロック 25 MHz に起因している．Intel Pro/1000, Intel Pro/100 及び ADMtek AN983(B) について，図 7 に送信特性，図 8 に受信特性を示す．上記 3 種の NIC では，送信は Intel Pro/1000，受信は ADMtek AN983(B) がそれぞれ最も分散が小さいことが分かる．なお，NIC の「まとめ割込み機能」は利用していない．このように，NIC によってジッタ特性が異なるため，ソフトウェアタイムスタンプを用いる場合には，事前に使用する NIC の特性を測定するとともに，必要に応じて，NIC を選択する必要がある．

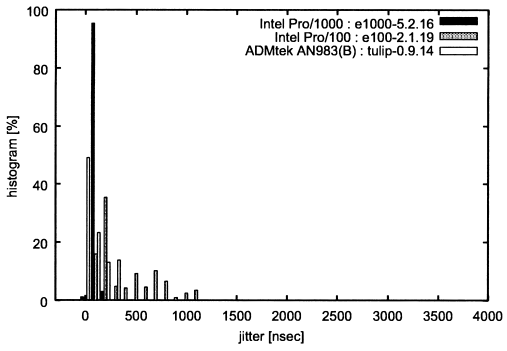


図 7　各種 NIC の送信時ジッタ分散特性
Fig. 7　Jitter distribution for each NIC at Tx.

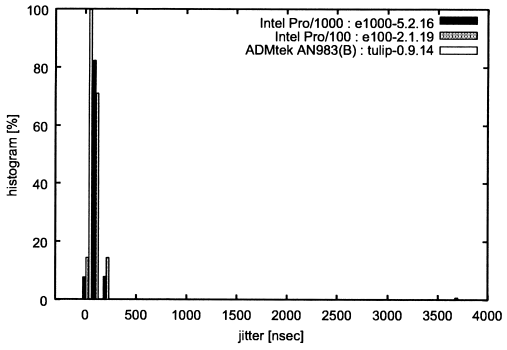


図 8　各種 NIC の受信時ジッタ分散特性
Fig. 8　Jitter distribution for each NIC at Rx.

4.2 CPU 動作周波数

同一マザーボードで，BIOS の設定により，CPU 動作周波数のみを 800 MHz から 2400 MHz まで変化させた場合のジッタの変化を図 9 に示す．横軸には周波数の逆数をとり，1 クロック当りの時間としているため，原点に近いほど高速 CPU となる．CPU での処理時間は，動作周波数に反比例するため，高速な CPU ほどジッタが小さくなると，予想される．

図 9 より，送信時では右上りの直線にほぼ乗っているが，受信時では CPU 動作周波数の効果はほとんど見られないことが分かる．これは，送信時では，PCC を読み出してから，パケットを送出するまでに，多数のソフトウェアステップを要するのに対し，受信時には，ハードウェア割込みが生じた直後に PCC を読み出すことができるため，ほとんどソフトウェアステップを必要としないためであろう．

さて，回帰直線を求め，図 9 に書き加えると，y 切片は，CPU の動作周波数を無限大まで高速化し，CPU 内での処理が瞬時に完了したとしても残る，CPU 以外のコンポーネントで発生するジッタで，送信時で 23.8 ns，受信時で 16.8 ns と考えることができる．

4.3 CPU 動作周波数のスプレッドスペクトル

近年の PC では，EMI 対策のため，CPU や PCI バスなどの周波数を変動させ，漏れ電磁波のピークを分散させており，CPU 動作周波数の変動はタイムスタンプ精度の劣化を伴う．本節では，本論文で用いている PC に使用されている，ICS 932S200 クロックジェネレータによるスプレッドスペクトルの影響を解析する．

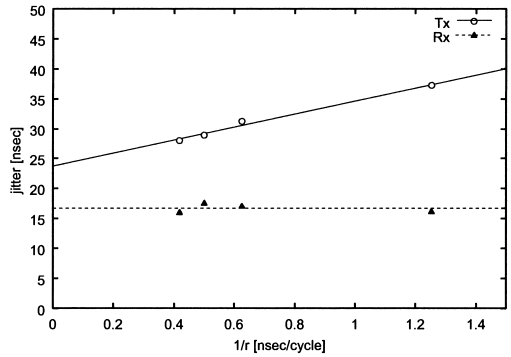


図 9　CPU 動作周波数の影響
Fig. 9　Effects of CPU frequency.

(注1)：現在，情報通信研究機構において開発中．

ICS 932S200 では，図 10 に示すように，50 kHz，0.5%の三角波による変調をかけているが，PCC の値は，CPU 動作周波数の累積であるため，周波数変動に伴って誤差 $\Delta c(t)$ は周期 20 マイクロ秒で変動する．

$$\Delta c(t) = \int_0^t 4rf_s\Delta r\xi d\xi \tag{2}$$

$$= 2rf_s\Delta rt^2 \tag{3}$$

誤差の最大 Δc_{max} は，図 10 の斜線部の面積に相当する．この誤差を秒の単位に変換するには，式 (1) より，r で除すればよい．平均周波数を r [Hz]，spread spectrum による振幅を $\Delta r = 0.25$ [%]，変調周波数を $f_s = 5 \times 10^4$ [Hz] とすれば，

$$\Delta c_{max} = \frac{2}{r}\int_0^{1/4f_s} 4rf_s\Delta r t dt \tag{4}$$

$$= \Delta r/4f_s \tag{5}$$

$$= 12.5[\text{ns}] \tag{6}$$

一方，分散は，以下で与えられる．

$$\sigma^2 = \frac{4f_s}{r^2}\int_0^{1/4f_s} \Delta c(t)^2 dt \tag{7}$$

$$= 16f_s^3\Delta r^2\int_0^{1/4f_s} t^4 dt \tag{8}$$

$$= 16f_s^3\Delta r^2\frac{1}{5}\frac{1}{4^5 f_s^5} \tag{9}$$

$$= \frac{\Delta r^2}{54^3 f_s^2} \tag{10}$$

したがって，標準偏差は，以下のとおりとなる．

$$\sigma = \frac{\sqrt{5}\Delta r}{40f_s} = 2.80[\text{ns}] \tag{11}$$

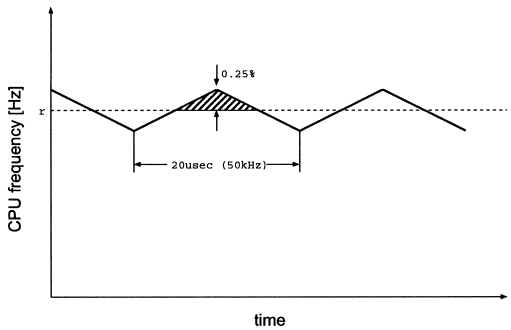


図 10　スプレッドスペクトルの影響
Fig. 10　Effects of spread spectrum.

4.4　内部バスの影響

PC は，CPU やメモリ等，多数のモジュールにより構成されており，本論文で用いた PC の各モジュールは，図 11 に示すように，チップセットを中心として，クロックスピードの異なるバスによって相互に接続している．クロックスピードの異なるバスをまたがってデータが伝送されるときには，クロックの立上り（または，立下り）を待つために，受け側のバスのクロックスピードの逆数を幅とする一様分布の待ち時間が生じる．

ここで，バスのクロックスピードを f_b [Hz] とすれば，平均待ち時間 μ は，

$$\mu = \frac{1}{2f_b} \tag{12}$$

分散 σ_b^2 は，

$$\sigma_b^2 = 2f_b\int_0^{1/2f_b} t^2 dt \tag{13}$$

$$= \frac{1}{12f_b^2} \tag{14}$$

標準偏差 σ_b は，

$$\sigma_b = \frac{\sqrt{3}}{6f_b} \tag{15}$$

で，与えられる．各バスでの待ち時間の標準偏差を計算した結果を表 2 に示す．また，PC を接続している通信回線のデータキャリヤ周波数もバス周波数と同様に，一様分布の待ち時間を生じる．本論文では 100Base-TX を用いているため，キャリヤ周波数は 25 MHz である．この通信回線キャリヤ周波数と前節のスプレッドスペクトルにより生じる遅延ジッタも表 2 に加えて表示する．これらの遅延ジッタを加え合わせたトータルの遅延ジッタは，図 9 において，CPU

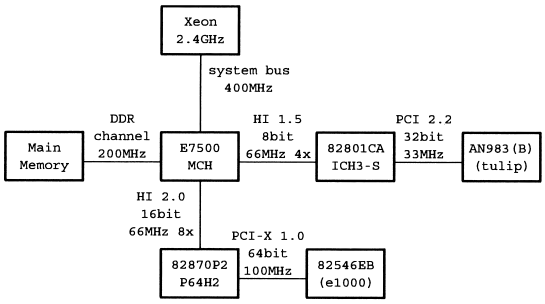


図 11　PC アーキテクチャとバス構成
Fig. 11　PC architecture and BUS.

表 2　各バスにおけるクロック待ち時間による生じるジッタの標準偏差
Table 2　Standard deviation of waiting time at BUS.

Bus	Clock [MHz]	σ [ns]
system bus	400	0.72
DDR memory bus	200 × 2	0.72
HI 2.0 bus	66	4.37
PCI-X bus	100	2.89
FastEthernet	25	11.55
Spread Spectrum		2.80
Total		23.77

を無限大まで高速化した際の送信時のジッタに一致し，現在の PC ハードウェアアーキテクチャの限界と考えることができる．

しかし，信頼性を高めるために，ほかのチップセットを用いた PC でも確認する必要がある．

5. むすび

PC システムをソフトウェア及びハードウェアの両面から詳細に解析し，割込みハンドラでプロセッササイクルカウンタを取得し，非リアルタイムの通常の Linux システムに標準のチューニングを施すことにより，GPS を利用したハードウェアタイムスタンプを超える，標準偏差 34 ns 精度のソフトウェアタイムスタンプを実現し，ネットワークの片道遅延の変動を精密に測定することが可能となった．また，マルチタスク OS で障害となる，リソース競合による影響を排除するフィルタ手法も提案した．ただし，PC は不要プロセスを停止し，タイムスタンプ処理に専念しており，多数プロセス実行下での提案フィルタ手法の効果は検討していない．更に，長期間の運用に対しても，高精度 PC を用いることによって，精度を維持することが可能である．コスト的にも，Rb 基準信号発生器及び PC の水晶置換え改造は，GPS 受信機より安価であり，GPS アンテナの設置問題も生じない．また，ハードウェアコンポーネントのジッタ解析を行い，本手法が PC の能力をアーキテクチャの限界近くまで引き出していることを示した．なお，PC の性能向上は今後も続くと考えられるが，本論文で提案した手法は，「できるだけネットワークの近くで」の原則に沿っており，高速化した PC 性能を十分に引き出すために，将来にわたり有効と考えられ，より高精度なタイムスタンプにより，新なアプリケーションの開発が期待される．

また，本論文で改善したタイムスタンプは座標系が UTS 等の時刻座標にはのっていない．ネットワーク計測など遅延時間を測る場合には，時刻座標とのオフセットが一定であれば，その大きさは問題とはならない場合が多いが，用途によっては，時刻座標系とのオフセットの精度（時刻精度）が重要となる場合もあり，時刻精度の向上も今後の課題である．

文　献

[1] J. Micheel, I. Graham, and S. Donnelly, “Precision timestamping of network packets,” Proc. ACM IMW, 2001.

[2] A. Pásztor and D. Veitch, “PC based precision timing without GPS,” Proc. ACM SIGMETRICS 2002, pp.1–10, 2002.

[3] H. Okazawa, A. Machizawa, S. Nakagawa, Y. Kitaguchi, T. Asami, and A. Ito, “Advanced NTP synchronization device for internet monitoring tools,” Proc. INET2001, Stockholm, 2001.

[4] 町澤朗彦，北口善明，岡沢治夫，中川晋一，“ネットワークを用いた CPU 動作周波数期間安定度の精密評価,” DICOMO 2002 予稿集，pp.563–566, July 2002.

[5] Z.-S. Su, “A specification of the internet protocol (IP) timestamp option,” RFC781, 1981.

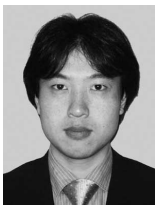
[6] R.S. Prasad, C. Dovrolis, and B.A. Mah, “The effect of layer-2 switches on pathchar-like tools,” Proc. ACM IMW 2002, pp.321–322, 2002.

[7] G. Jin and B.L. Tierney, “System capability effects on algorithms for network bandwidth measurement,” Proc. Internet Measurement Conference 2003 (IMC 2003), pp.27–38, Miami, Oct. 2003.

（平成 16 年 1 月 9 日受付，5 月 7 日再受付）

町澤　朗彦　（正員）

昭 59 上智大・理工・電気電子卒．同年郵政省電波研究所（現情報通信研究機構）入所．平 6 科学技術庁に出向し，IMnet 立上げに参与．平 8〜11 Univ. Canterbury 客員研究員．平 15 JGN2 立上げに参与．画像の高能率符号化，視覚情報処理，計算機ネットワークの研究に従事．日本認知科学会会員．



北口　善明　（正員）

平 7 新潟大・理・物理卒．平 9 同大大学院自然科学研究科博士前期課程了．同年（株）インテック入社．平 12 インテック・ウェブ・アンド・ゲノム・インフォマティクス株式会社に転籍，現在に至る．平 12〜16 通信・放送機構（現情報通信研究機構）研究員．主として，ネットワーク運用と計測技術，IPv6 ネットワークの研究開発に従事．情報処理学会会員．