社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2009－ARC－182 (12)
2009－HPC－119 (12)
2009／2／27

# 精密な帯域共有とトラフィック隔離を実現するパケットスケジューリング方式

高野了成[†1] 工藤知宏[†1]
児玉祐悦[†1] 岡崎史裕[†1]

本稿では，精密なトラフィック隔離と公平な帯域共有を実現するパケットスケジューリング方式を提案する．提案方式は，パケットの送信予定時刻を送信済みバイト数を基に決定することで，ハードウェアタイマに依存しない精密なスケジューリングを可能にする．提案方式を Linux オペレーティングシステムで動作するパケットスケジューリングモジュールとして実装し，10 ギガビットイーサネット環境において評価した．その結果より，既存のトークンバケット方式よりも，精密なトラフィック隔離と公平な帯域共有が両立できることを示す．さらに本稿では，パケットスケジューラとハードウェアオフロード機能の協調動作の影響についても議論する．

# Elastic packet scheduling method for bandwidth sharing and traffic isolation

Ryousei Takano,[†1] Tomohiro Kudoh,[†1] Yuetsu Kodama[†1]
and Fumihiro Okazaki[†1]

In this paper, we propose an elastic packet scheduling method, which provides precise traffic isolation and fair bandwidth sharing. The proposed method determines transmission timing of packets by the number of bytes transferred to enable a precise packet scheduling without dependence on a hardware timer. We implemented the proposed method in the Linux operating system. Experimental results show that the proposed method achieves both precise traffic isolation and fair bandwidth sharing compared with a Token Bucket method on a 10 Gigabit Ethernet environment. We also discuss that effects of cooperation between a packet scheduler and hardware offloading mechanisms.

## 1. はじめに

大容量通信を伴うアプリケーションのサービス品質およびネットワーク利用効率を向上するには，各アプリケーションが要求する帯域を正確に保証し，競合する要求に対して優先度に基づいた調停機能を有するパケットスケジューリング機構を提供する必要がある．具体的には，(1) 帯域利用効率の向上，(2) 精密なトラフィック隔離，(3) 公平な帯域共有が要求される．

(1) については自明であるが，(2) はあるアプリケーションの通信が他のアプリケーションの通信に影響を受けて性能低下しないように，指定された送信レートを超過しない精密な帯域制御を実現すること，(3) はアプリケーションの要求帯域が利用可能帯域を超過する場合 (over subscribed) に，アプリケーションによって帯域の偏りができないように，送信レートに比例して帯域を共有すること，と本稿では定義する．

ギガビットクラスのネットワークを用いた予備実験の結果，既存の HTB (Hierarchical Token Bucket) では，上記の 3 つの要求を十分に満たせないことが明らかになった．そこで，本稿では，論文 1) で提案したバイトクロックの概念を利用したパケットスケジューリング方式に対する改良を提案する．既存提案方式は目標送信レートに基づいた精密な帯域制御を実現できるが，over subscribed 設定での動作は考慮していないなどの問題がある．そこで，(1) パケット選択アルゴリズムの見直しによる上限送信レートの引き上げ，(2) over subscribed 設定時のトラフィック隔離と帯域共有への対応を行った．提案方式を Linux オペレーティングシステム上に実装し，10 ギガビットイーサネット環境にて評価実験を行ったので，その結果について報告する．

以下，2 節で提案方式の設計について述べる．3 節で実装について述べ，評価実験の結果について 4 節で

†1 産業技術総合研究所 情報技術研究部門
Information Technology Research Institute, National Institute of Advanced Industrial Science and Technology (AIST), Japan

述べる．5節では，ネットワークインタフェースデバイスのハードウェアオフロード機能について，システムソフトウェアとハードウェアの協調動作の観点から議論する．最後に6節でまとめを行う．

## 2. 設　　計

### 2.1 目　　的

本稿で提案するパケットスケジューリング方式の目的は，(1) 帯域利用効率の向上，(2) 精密なトラフィック隔離，(3) 公平な帯域共有を実現することである．

まず前提として，スケジューラは複数のキューを管理する．入力されたパケットはIPアドレスやプロトコルに基づいてクラス分けされ，クラスごとのキューにキューイングされる．クラスには目標送信レートが設定でき，その値にしたがって出力パケットが選択される．また，目標送信レートの設定方法には次の2種類が存在する．各クラスが要求する目標送信レートの合計が利用可能帯域を超過しない under subscribed 設定と，超過する over subscribed 設定である．前者の場合には，通信が発生しないアイドル時間が存在する．一方，後者の場合はすべてのクラスの要求を同時に満たせないので，何らかのポリシにしたがって，一時的に送信レートを要求以下に制限する必要がある．

次にそれぞれの目的について詳しく述べる．

(1) 帯域利用効率の向上
トークンバケットによる送信レート制御方式のように，単位時間あたりの平均で送信レートを制御する場合，一定のバースト送信は回避できない．その影響によるルータのバッファ溢れを防止するために，目標送信レートを低めに制限するトラフィックエンジニアリングが行われることがある．提案方式では，パケット単位で送信間隔を精密に制御し，目標送信レートを超えないぎりぎり最大限で通信を行う．

(2) 精密なトラフィック隔離
あるクラスが目標送信レート以上のトラフィックを発生することにより，他のクラスに影響を及ぼさないように，指定された送信レートを超過しない精密な帯域制御を実現する．

(3) 公平な帯域共有
over subscribed 時には，上記のように一時的に送信レートを要求以下に制限する必要がある．本稿では，クラス間の公平性を重視して，各クラスの目標送信レートに比例して帯域を傾斜配分する．

### 2.2 バイトクロックに基づくパケットスケジューリング方式

これらの目的を実現するため，論文1) で提案したバイトクロック (byte clock) の考えを導入する．バイトクロックは，ネットワークインタフェースがワイヤレートで1バイトの送信に要する時間の単位であり，スケジューラは通信開始からの送信バイト数をバイトクロックとして管理する．そして，パケット送信時刻を正確にスケジューリングするために，実時間の代わりにバイトクロックを用いる．その根拠は，例えばギガビットイーサネットであればバイトクロックが8ナノ秒と一定であり，パケットを隙間なく連続して送信できれば，送信時刻は正確に制御できるからである．

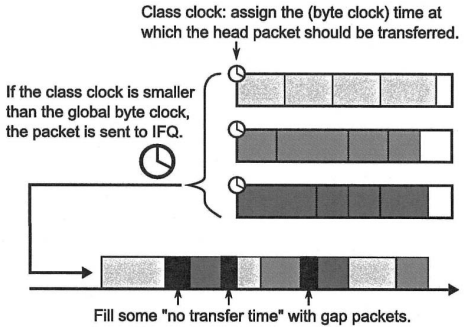



**図1** バイトクロックスケジューリング
Fig. 1 Byte clock scheduling.

パケットスケジューリング方式の概要を**図1**に示す．各クラスは，キューの先頭パケットの送信予定時刻を保持するために，クラスクロックを管理する．現在時刻はインタフェースごとのグローバルクロックに保持する．スケジューラは，グローバルクロックとクラスクロックを比較し，送信予定時刻に達し，かつクラスクロックが一番小さいクラスの先頭パケットから順に送信する．その際，グローバルクロックにはパケット送信に要する時間 $t(s)$ を加算する．同時にクラスクロックには $t(s)$ に加えて，次のパケット送信までの待ち時間 $w(s,r)$ も加算する．バイトクロックに基づくスケジューリングでは，$t(s)$ はパケットサイズ $s$ と等しくなる．

$$t(s) = s \tag{1}$$

$w(s,r)$ は式2から導出できる．なお，$r$ は目標送信レート，$r_{max}$ は最大送信レート，つまり一般的にはネットワークインタフェースの物理帯域である．

$$w(s,r) = \left(\frac{r_{max}}{r} - 1\right) \times s \tag{2}$$

すべてのクラスが送信予定時刻に達していない場合は，通信がアイドルであることを意味する．このアイドル時間を正確に保つために，このアイドル状態のパケット間隔に一致するサイズ（以下，ギャップサイズと呼ぶ）の擬似的なパケットを挿入する．したがって，インタフェースからは隙間なく連続してパケットが送出される．以下，この擬似的なパケットをギャップパケットと呼び，通常のパケットは実パケットと呼び区別する．

論文1) では，目標送信レートに基づいてパケット

送信間隔を均一に平滑化するためにバイトクロックを用いたが，(1) 目標送信レートの上限値に制限がある，(2) over subscribed 設定に対応していないという問題があった．それぞれの問題の解決について，続く 2.3 節と 2.4 節にて述べる．

### 2.3 ギャップパケット挿入のアルゴリズム

ギャップパケットをイーサネットフレームとして実現する場合，最小サイズは 64 バイト，最大サイズは MTU（例えば 1500 バイト）に制限される．ギャップサイズが MTU 以上になる場合は，複数のギャップパケットに分割して送信することで簡単に対応できる．一方，最小サイズを *minsize* とすると，ギャップサイズを *minsize* 以下に制御できないので，目標送信レートの上限は $(MTU/(MTU + minsize)) \times max_rate$ に制限される．例えばギガビットイーサネットで MTU が 1500 バイトの場合では，935.2 Mbps が目標送信レートの上限になっていた．

この問題に対しては，最小サイズのギャップパケットを挿入するかどうかを制御することで対応可能である．クラスクロックとパケット間ギャップの関係は**図 2** に示す 3 パターンとなり，次に述べるようにパケット送信をスケジューリングする．図中，各パターンの上段はクラスの先頭パケットとクラスクロックを，下段はそのパケットが実際にスケジューリングされる時刻を示す．また，X 軸はグローバルクロックに対する相対値である．

(1) $gapsize <= minsize/2$
即時に実パケットを送信する．実パケットは省略した *gapsize* 分だけ早く送信される．

(2) $minsize/2 < gapsize < minsize$
即時にギャップパケットを送信する．実パケットの送信は $minsize - gapsize$ 分だけ遅延する．

(3) $minsize <= gapsize$
即時にギャップパケットを送信する．実パケットの送信は正確に制御される．

上記の制御により，パケット送信を $\pm 1/2\ minsize$ のずれでスケジュールし，かつ平均的に実効送信レートを目標送信レートに近づけることが可能になる．

### 2.4 over subscribed 設定への対応

各クラスの目標送信レートの合計がインタフェースの物理帯域を超える over subscribed 時には，特別なパケット優先度制御がなければ，クラスクロックが最小値を示すクラスから順にパケットを送信するので，目標送信レートに比例した傾斜配分になる．ただし，次の 2 つの問題を解決する必要がある．つまり (1) 新規クラスの追加時など，クラスがアクティブになったときにクラスクロックの初期値をどう設定するか，(2) over subscribed 時には，クラスクロックがグローバルクロックより遅れてしまいパケットがキューに溜まる，という問題である．なお，アクティブとはキューにパケットが存在する状態のことを指す．

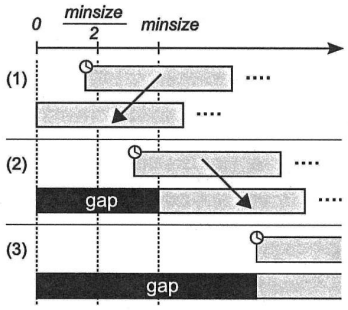



**図 2** ギャップパケット挿入のアルゴリズム
Fig. 2 Algorithm of gap packet injection

(1) については，初期値をグローバルクロックに設定すると，すべてのクラスクロックがグローバルクロックに追いつくまで当該パケットの送信が遅延される．そこで，グローバルクロックとすべてのクラスクロックの最小値を初期値とした．

(2) については，あるクラスが通信を停止するなどの理由で非アクティブに遷移したとき，溜まったパケットが出力可能となり，目標送信レートを超えるパケットがバースト的に送信されてしまう．それを解決するために，アクティブかつクラスクロックが最小なクラスのクラスクロックをグローバルクロックに一致するように，その差分をすべてのアクティブなクラスに加算する．

## 3. 実　　装

PSPacer[1] はバイトクロックを用いて，パケット送信間隔を均一に調整することでバースト送信の平滑化と精密な帯域制御を実現している．本稿では，PSPacer のスケジューリング方式を拡張することで，提案方式を実装した．PSPacer は Linux オペレーティングシステム上で動作する Qdisc (Queueing discipline) カーネルモジュールである．そのためインストール時にはカーネルの再構築が不要であり，トランスポートプロトコルやデバイスドライバに依存せずに利用できる．以下，本実装を PSPacer+ と呼ぶ．

ギャップパケットは，PC から送信する必要があるが，他の通信に副作用を与えることなく，また動作環境を選ばないために標準に準拠することを考慮すべきである．我々はイーサネット規格 IEEE 802.3x で規定される PAUSE フレームがこの目的に利用できることを明らかにした[1]．本実装でも，任意サイズの PAUSE フレームをギャップパケットして利用する．ギャップパケットは送信計算機の直近のルータまたはスイッチの入力ポートで必ず破棄されるため，他のネットワーク機器に影響を与えることはない．

## 4. 実　　験

### 4.1 実験環境

実験に用いた計算機は，CPU に Quad-core Xeon E5430/2.66GHz x2，4 GB メモリ (DRR2-667)，そして 10 ギガビットイーサネットインタフェースとして PCI-Express x8 バスに接続した Myricom Myri-10G を搭載する．Myri-10G の MTU は 9000 バイトとした．計算機上では Ubuntu server 8.10 (Linux カーネル 2.6.27.10) が動作し，十分なネットワーク性能を得るために，**表 1** に示すネットワーク関連の sysctl パラメータを変更した．また，提案方式と比較するため，Linux 標準の HTB に対しても同様に実験した．

上記の計算機を 2 台，ハードウェアネットワークテストベッド GtrcNET-10[2),3)] を介して接続した．GtrcNET-10 は，10 ギガビットイーサネット XENPAK と 1 ギガバイト DDR-SDRAM を各 3 ポートづつ，大規模 FPGA (XilinxXC2VP100) に接続した構成になっている．GtrcNET-10 は，FPGA のプログラムを変更することによってネットワークのエミュレーション，パケットキャプチャリング，トラフィックモニタリングなどの様々機能をワイヤレートで実現できる．本実験では，100 ミリ秒間隔に正確な通信帯域を測定するために使用した．

### 4.2 帯域制御

単一 TCP 通信に対する帯域制御の正確さを評価するために，目標送信レートを 100 Mbps から 10 Gbps まで，100 Mbps 刻みに設定した場合の，グッドプット（アプリケーションレベルでのスループット）を測定した．実験には Iperf を用い，各目標レートに対して 5 秒間通信を行った．PSPacer+の目標送信レートはイーサネットレベルの値であるため，IP/TCP ヘッダのオーバヘッド分だけグッドプットは低くなることに注意されたい．

実験結果を**図 3** に示す．X 軸が設定した目標送信レートであり，Y 軸が測定結果のグッドプットである．PSPacer+では目標送信レートと実験結果がきわめて一致しているのに対して，HTB は目標送信レートが高くなるにしたがい，正確な帯域制御に失敗している．

### 4.3 トラフィック隔離

#### 4.3.1 VServer 仮想環境

あるクラスに対するトラフィックが他のクラスの帯域制御に影響を及ぼさないように，トラフィックが隔離されているか評価するために，オペレーティングシステム仮想化技術である VServer[4)] 環境を用意して実験を行った．VServer は仮想環境（VE）に対する資源の可視性を制御することで，単一のカーネル上に複数の VE を実現する．VServer は，Xen などの仮想マシン型の仮想化技術と比較してオーバヘッドが小さいという特長があり，VE 上のバルク転送性能はホスト OS と同等であることを確認している[5)]．なお，本実験で用いた VServer のバージョンは 2.3.0.36.2 である．

2 台の物理計算機上にそれぞれ 5 つの VE を作成し，異なる計算機上で動作する VE 間で netperf を 60 秒間実行した．このとき，通信開始時間は 5 秒づつずらした．ホスト OS では各 VE の送信元 IP アドレスを基にトラフィックをクラス分けし，VE に対応するクラスごとに帯域制御を行った．

#### 4.3.2 under subscribed 設定

まず，5 つの VE の目標送信レートを 4，2，2，1，1 Gbps に設定し，合計帯域が物理帯域に収まる under subscribed 設定で実験した．

**図 4** に GtrcNET-10 で観測したトラフィックを示す．さらに実験開始後 20 秒から 30 秒間の平均帯域を**表 2** にまとめる．括弧内の数値は合計帯域に占めるパーセンテージである．PSPacer+では，各 VE に割り当てられた帯域を超過することなく正確に守っている．一方，HTB はでは．4.2 節の実験からわかるように，HTB の出力は目標レートを超過する．しかし，20 秒から 60 秒区間の合計帯域は，表 2 からわかるように，PSPacer の方が 160 Mbps 以上高く，トラフィック隔離を実現しつつ，帯域利用効率も高いことがわかる．

#### 4.3.3 over subscribed 設定

続いて，目標送信レートの設定を 4，4，4，2，2 Gbps に変更し，合計帯域が 16 Gbps となる over subscribed 設定で同様な実験を実行した．

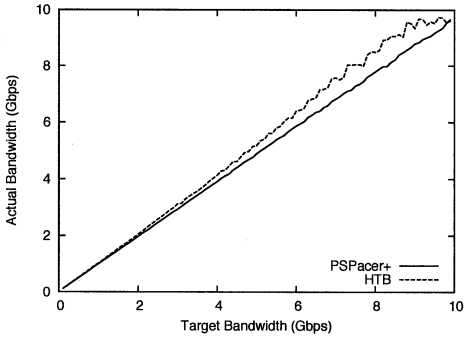
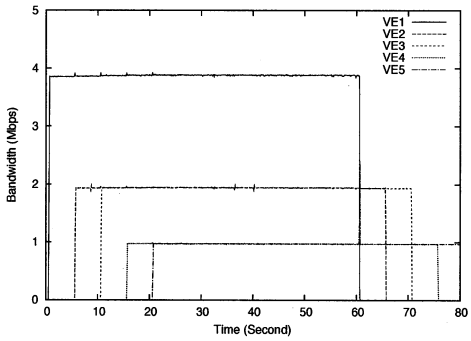


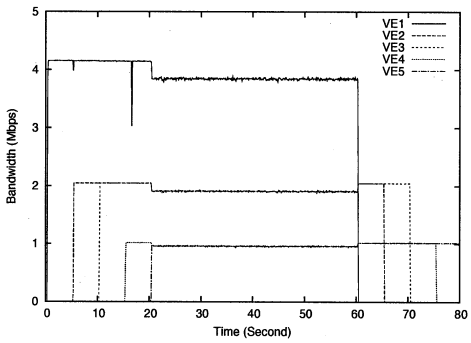

**図 3** PSPacer+と HTB の帯域制御の比較
Fig. 3 Bandwidth control comparison between PSPacer+ and HTB.

**表 1** sysctl パラメータ
Table 1 sysctl parameters.

| | |
|---|---|
| net.core.netdev_max_backlog | 250000 |
| net.core.wmem_max | 16777216 |
| net.core.rmem_max | 16777216 |
| net.ipv4.tcp_rmem | 4096 87380 16777216 |
| net.ipv4.tcp_wmem | 4096 65536 16777216 |
| net.ipv4.tcp_no_metrics_save | 1 |



(1) PSPacer+



(2) HTB

**図 4** under subscribed 設定でのトラフィック (100 ミリ秒間隔)
Fig. 4 Traffic monitoring with under subscribed setting (100 msec interval).

**表 2** under subscribed 設定での平均帯域 (20〜50 秒)
Table 2 Average bandwidth (20-50 seconds) with under subscribed setting.

| | PSPacer+ | HTB |
|---|---|---|
| VE1 | 3896.39 (40.0%) | 3843.99 (40.1%) |
| VE2 | 1948.39 (20.0%) | 1909.39 (19.9%) |
| VE3 | 1948.40 (20.0%) | 1907.29 (19.9%) |
| VE4 | 974.224 (10.0%) | 957.546 (10.0%) |
| VE5 | 979.079 (10.0%) | 956.921 (10.0%) |
| Total | 9737.483 (100%) | 9575.137 (100%) |

**表 3** over subscribed 設定での平均帯域 (20〜50 秒)
Table 3 Average bandwidth (20-50 seconds) with over subscribed setting.

| | PSPacer+ | HTB |
|---|---|---|
| VE1 | 2427.18 (25.0%) | 2429.81 (25.1%) |
| VE2 | 2427.19 (25.0%) | 2406.42 (24.9%) |
| VE3 | 2427.18 (25.0%) | 2389.59 (24.7%) |
| VE4 | 1213.66 (12.5%) | 1233.68 (12.7%) |
| VE5 | 1213.64 (12.5%) | 1221.78 (12.6%) |
| Total | 9708.85 (100%) | 9681.28 (100%) |

トラフィックの観測結果を**図 5** に，通信開始 20 秒から 30 秒間の平均帯域を**表 3** に示す．PSPacer+，HTB 共に要求通りの帯域制御を実現している．ただし，帯域割当ての精度を比較すると，PSPacer+が 2:2:2:1:1 の割合に正確に制御できているのに対して，HTB では 0.1 から 0.2%の誤差が生じている．また合計帯域も PSPacer+の方が 27.57 Mbsp 高い．

これら 2 つの実験結果より，提案方式では under subscribed と over subscribed のどちらの設定においても，精密な帯域共有とトラフィック隔離を実現できることを確認した．

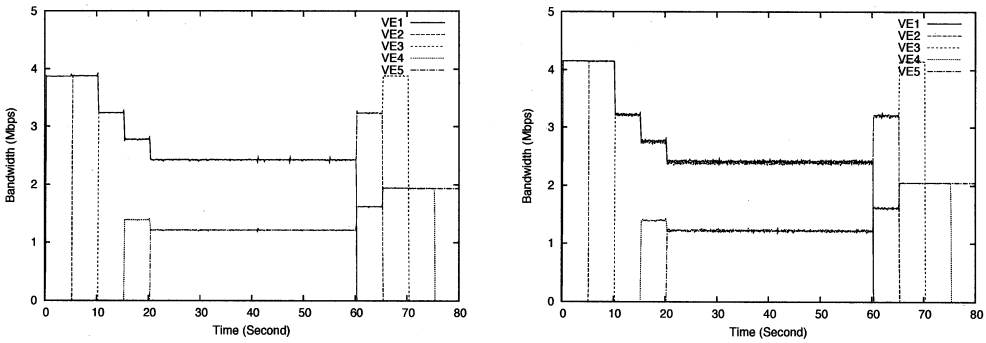
## 5. 議　　論

最近のネットワークインタフェースは，チェックサムオフロードや，TSO (TCP Segmentation Offload)，UFO (UDP Fragmentation Offload)，LRO (Large Receive Offload) など，プロトコルスタック処理の一部をハードウェアが担うことで CPU 負荷を軽減する仕組みを備えている．プロセッサのプリフェッチ機構によるシーケンシャルアクセスの高速化などの効果により，データコピーといった per-byte 処理の負荷は軽減されているが，一方でヘッダ処理やバッファ管理といった per-packet 処理が相対的に大きな割合を占める傾向にある[6]．TSO や LRO は per-packet 処理の負荷を軽減する仕組みとして有効である．

TSO に対応したデバイスは，ドライバから数パケット分のデータを格納したバッファを受け取り，パケットの分割と送信を一括してハードウェアで処理する．一般的にこのバッファサイズは最大 64 KB である．さらに，Linux では，TSO の概念をプロトコルスタックからドライバまでの処理全体に汎用化した GSO (Generic Segmentation Offload) を実装している．

PSPacer+は TSO を考慮して帯域制御しており，前節の実験では TSO および GSO を有効にして行った．この際，帯域制御の正確さと CPU 負荷トレードオフに注意が必要である．TSO と GSO の設定を変えて，netperf を実行した結果を**表 4** に示す．この結果から TSO と GSO が CPU 負荷の軽減に効果があることがわかる．さらに両者を無効にした場合は，ある CPU の利用率が 99%に達しボトルネックとなったため，グッドプットは 7867.7 Mbps まで落ち込んだ．

TSO を有効にすると，バッファに格納されたパケット間にギャップを挿入することはできない．例えば，バッファサイズが 64 KB で MTU が 9000 バイトの場合，最大 7 パケットがバースト送信される．この挙動を確認するために，目標送信レートを 5 Gbps に設定し，GtrcNET-10 を用いてパケットをキャプチャした．TSO/GSO 有効時には 7 パケットがバースト送信され，続いて約 50 マイクロ秒のアイドルが発生し




(1) PSPacer+ (2) HTB

**図 5** over subscribed 設定でのトラフィック
Fig. 5 Traffic monitoring with over subscribed setting.

**表 4** TSO/GSO 設定と性能の関係
Table 4 TSO/GSO setting and performance.

| TSO | GSO | CPU util. | Bandwidht |
|---|---|---|---|
| on | on | 8.6 % | 9514.0 Mbps |
| off | on | 12.9 % | 9533.8 Mbps |
| off | off | 13.8 % | 7867.7 Mbps |

ていた．一方，TSO/GSO 無効時にはすべてのパケット間隔が約 7.2 マイクロ秒に平滑化されていた．

また，送信処理の並列化や MAC レベルでの QoS を実現するために，複数の送信キューを内部に備えたデバイスが普及しはじめた．例えば，仮想環境ごとに送信キューを割り当てることで，head-of-line ブロッキング回避のためにも有効であると考える．現在の実装は，単一の送信キューを前提にしているが，送信マルチキューの検討は今後の課題の一つである．

## 6. ま と め

本稿では，精密な帯域共有とトラフィック隔離を実現するパケットスケジューリング方式を提案し，10 ギガビットイーサネット環境にて評価実験を行った．提案方式と仮想化技術 VServer を組み合わせた実験では，under subscribed および over subscribed 設定に対して，精密なトラフィック隔離と公平な帯域共有の実現を確認した．

現在の実装では，over subscribed 時に，各クラスの帯域を単純に目標送信レートに比例して傾斜配分している．しかし，リアルタイムが要求されるストリーミング通信では送信を遅延するよりも破棄する方が適している可能性が高い．また，送信レートに関しても，最低保証値と上限値を指定できると便利である．今後は，そのようなトラフィックの性質に適した柔軟なパケット優先度制御の設計を進め，実装する予定である．

なお，PSPacer は GNU GPL ライセンスによるオープンソースソフトウェアとして公開しており，`http://www.gridmpi.org/pspacer.jsp` から入手可能である．

## 謝　辞

本研究の一部は，文部科学省科学研究費補助金 (20800083) による．

## 参 考 文 献

1) 高野了成，工藤知宏，児玉祐悦，松田元彦，石川　裕，岡崎史裕：ギャップパケットを用いたソフトウェアによる精密ペーシング方式，情報処理学会論文誌，Vol.47, No.SIG 7 (ACS 14), pp. 194–206 (2006).
2) GtrcNET-10: `http://projects.itri.aist.go.jp/gnet/`.
3) 児玉祐悦，工藤知宏，清水敏行：10GbE 対応ネットワークテストベッド GtrcNET-10 の構成と評価，情報処理学会研究会報告 2005-HPC-103, pp.109–114 (2005).
4) VServer: `http://linux-vserver.org/`.
5) 中田秀基，高野了成，竹房あつ子，工藤知宏：ネットワーク帯域予約と OS 仮想化機構を用いた分散アプリケーション実行環境に向けて，情報処理学会研究会報告 2009-HPC-119 (2009).
6) Menon, A. and Zwaenepoel, W.: Optimizing TCP Receive Performance, *USENIX 2008*, pp. 85–98 (2008).