

ソフトウェアによる精密ペーシング機構の提案と評価

高野 了成^{†,††} 工藤 知宏[†] 児玉 祐悦[†]
松田 元彦[†] 手塚 宏史[†] 石川 裕^{†,††}

パースト転送を行わないようにパケットの送信間隔を調整するペーシングにより、帯域遅延積の大きな TCP/IP 通信で発生する中間ルータのバッファあふれによるパケット破棄を防ぐことができることが知られている。しかし、従来は精密なペーシングには専用ハードウェアが必要であった。本稿では、オペレーティングシステムのパケットスケジューラにおいてギャップパケットを挿入することで、パケットの送信間隔を任意に調整するソフトウェアによる精密ペーシング手法を提案する。さらに、広域ネットワークを模擬した環境を用いた評価により、提案手法によって TCP/IP 通信性能が改善されることを示す。

Precise Software Pacing Method for Long Fat Pipe Communication

TAKANO RYOUSEI,^{†,††} KUDOH TOMOHIRO,[†] MATSUDA MOTOHIKO,[†]
KODAMA YUETSU,[†] TEZUKA HIROSHI[†] and ISHIKAWA YUTAKA^{†,††}

It is known that pacing can effectively reduce packet losses which are caused by buffer overflow in intermediate routers in a long fat pipe TCP/IP communication. However, special hardware is required to realize precise pacing.

In this report, we propose a precise software pacing method which realize virtual inter packet gaps(IPG) by inserting “Ethernet gap packets”. The gap packets are inserted by the operating system without using any additional hardware. By adjusting the size of gap packets, the pacing ratio can be precisely controlled. The effectiveness of the proposed method is shown by evaluating TCP/IP communication performance using a wide area network emulation environment.

1. はじめに

近年、グリッド技術が注目され、インターネット上に分散した計算機資源を利用した大規模計算、大規模データ通信のニーズが高まっている。

しかし、グリッド環境のように、ネットワークの帯域遅延積が大きい場合、TCP/IP 通信の実効帯域が低下することはよく知られている。これは、輻輳発生時に転送量が大きく低下することが原因であり、この問題に対して、HighSpeed TCP⁶⁾、Scalable TCP⁵⁾、FAST⁷⁾ など、新しい TCP 輻輳制御方式が提案、実装されている。

また、我々は、メトロポリタンネットワーククラスのグリッド環境を想定した並列アプリケーション実行の検討²⁾ や、SC2003 Bandwidth Challenge(以下、BWC と呼ぶ)で行った日米間高速ファイル転送実験³⁾

から、ネットワーク帯域使用率を上げるには、TCP における輻輳制御方式の改善だけではなく、データリンク層においてボトルネックリンクに合わせたペーシングが必要であるという知見を得た。

SC2003 BWC では、ギガビットイーサネットネットワークテストベッド GNET-1⁴⁾ を利用したパケット間ギャップ (IPG: Inter Packet Gap)^{*}制御によってペーシングを行った。その結果、日米間にパケットロスのない安定したネットワークフローを実現し、理論ピーク性能の 97%である 472.5MB/s のデータ転送レートを達成した。

APAN LA と東京間の OC-48 回線 (物理帯域は約 300MB/s) がボトルネックになるネットワークにおいて、6 対 6 ホスト間で対向にファイル転送を行った際の帯域変化を図 1、図 2 に示す。なお、RTT は 141ms、ホスト側のソケットバッファサイズは 8MB、TCP 輻輳制御方式には HighSpeed TCP を用いた。図 1 に示すように、ペーシングを行わない場合には、各ホスト

[†] 産業技術総合研究所グリッド研究センター (National Institute of Advanced Industrial Science and Technology)

^{††} 株式会社アックス (AXE, Inc.)

^{†††} 東京大学大学院情報理工学系研究科 (University of Tokyo)

^{*} イーサネットではパケットではなくフレームと呼ぶが、本稿ではパケットに表記を統一する

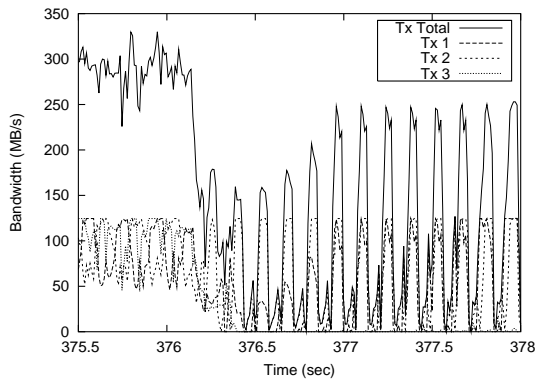


図1 GNET-1によるペーシング無効時:
APAN LA-Tokyo (300MB/s)

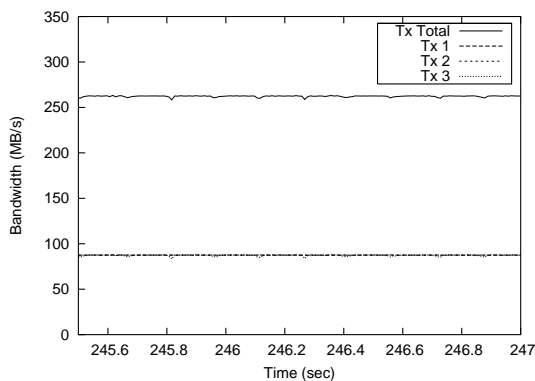


図2 GNET-1によるペーシング有効時 (87.5M/s x 3):
APAN LA-Tokyo (300MB/s)

がバースト的に送信を行うので、その合計が物理帯域を越え、ルータのバッファがあふれた時点で、パケットロスが発生しており、通信帯域は安定しない。一方、図2は2ホストの合計送信レートを87.5MB/sにペーシングし、6ホスト合計の送信レートが物理帯域を越えないように制御した場合であるが、送信帯域は一定で安定しており、パケットロスは発生していない。このように帯域遅延積の大きなネットワークにおける通信性能改善においてペーシングは有効な手段である。

本稿では、GNET-1のような特別なハードウェアを用いずに、ソフトウェアによって精密なペーシングを実現する手法を提案し、その実装と評価について述べる。提案手法では、パケットスケジューラによって本来送信すべきパケットの間に任意長のギャップパケットを挿入することによって、ボトルネックリンクにおけるIPGを調整し、ペーシングを行う。

以下、2節においてボトルネックリンクごとにペーシングが必要となる理由について述べる。続いて、3節で本方式の設計、Linux上への実装方法について述べ、4節で評価を行う。5節で得られた結果に対する議論を行う。6節で関連研究について述べる。最後に

7節でまとめを行う。

2. 背景

2.1 TCP 輻輳制御

TCPはセルフクロッキングプロトコルであり、確認応答(ACK)パケットを「クロック」として利用することで、通信経路中のボトルネックリンクに合わせてパケット送信間隔を調整する。つまり、ボトルネックリンクが存在する場合、受信側がACKを返信する間隔は、ボトルネックリンクを通過するパケット到着間隔となり、以降、送信側はACK受信に合わせて、パケットを送信する。

しかし、セルフクロッキングによるフィードバックが働くのは、ボトルネックリンクがパケットで満たされた以降であり、それまで送信側はNICの物理帯域による送信を続ける。つまり、通信開始時のバースト転送をセルフクロッキングで防ぐことはできない。また、キューイング遅延[☆]の影響によりACKが等間隔に返信されず、密着して送信側に届くことにより、バースト転送が発生するACK圧縮⁹⁾という現象も知られている。

さらに、TCPはストリームごとの輻輳ウィンドウによって輻輳制御しており、同じボトルネックリンクを通るストリームであっても、ストリーム間で転送量を協調して制御することはない。したがって、単一ストリームではボトルネックリンクルータのバッファで吸収できるバースト転送であっても、複数ストリームが同時にボトルネックリンクを通ると、パケットロスの原因となる。

2.2 ペーシング

TCPフローレベルでは送信レートを輻輳ウィンドウサイズ/RTTに制御しているが、パケットレベルでは、その送信レートにしたがって均一にパケット送信間隔が制御される保証はなく、バースト送信が発生する可能性がある。バースト送信の問題に対して、パケット送信間隔を均一に平滑化し、これを改善する手法としてペーシング¹⁰⁾が提案されている。

例えば、ギガビットイーサネット(以下、GbEと呼ぶ)では、1500バイトのパケットを送信するのにかかる時間は12usである。ここで送信レートを最大レートの半分である62.5MB/sにペーシングする場合、パケット送信間隔は24usに調整する必要がある。なお、パケットが送信されていない時間である12usで送信できるバイト数、すなわち1500バイトを、パケット間ギャップ(IPG: Inter Packet Gap)と呼ぶ。通常のNICではIPGを8~12バイトに設定している。

[☆] 通信遅延は物理的な伝送遅延とルータにおけるキューイング遅延に分類できる。光ファイバの伝送遅延は5ns/mであるが、キューイング遅延はホップ数、各ルータにおけるバッファリングにしたがって増加する。

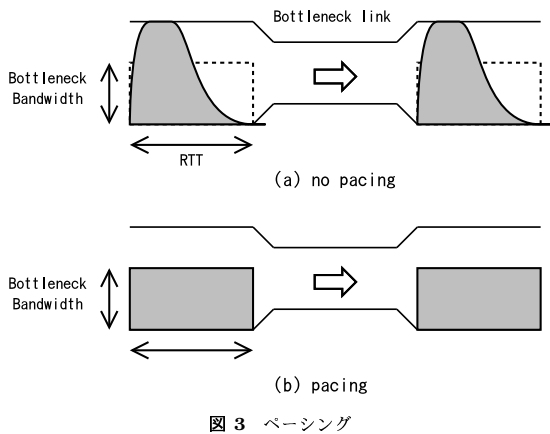


図 3 ペーシング

図 3(a) は、送信がペーシングされていない例であり、RTT あたりの送信レートはペーシングした場合 (図 3(b)) と同じであるが、ボトルネックリンクの帯域を越えるバースト送信が起きている。そして、バースト送信が続くと、ボトルネックルータのバッファがあふれてしまう。バッファ管理に TailDrop 方式を採用している一般的なルータは、キューイング量がバッファサイズを越えると、それ以降に到着したパケットをすべてドロップさせるので、バースト的なパケットロスが発生する。一方、図 3(b) のように、ボトルネックリンク帯域に合わせてペーシングすることで、ボトルネックリンクルータのバッファあふれ、パケットロスを防ぐことができる。このようにペーシングを利用することで、図 1 で発生するようなパケットロスを防ぎ、図 2 のように安定した転送レートを実現できる。

3. 設計と実装

3.1 ギャップパケットを用いたペーシング方式

ペーシングの実現方法は、GNET-1 のようなハードウェアによる IPG 制御と、ソフトウェアタイマによるパケット送信間隔制御に大別できる。前者は IPG を精密に制御できるが、特殊なハードウェアを用いる必要がある。一方、後者は汎用的なハードウェア上で実現できるが、時間的粒度や精密さ、負荷の増加が問題となる。

そこで、パケット間に、実通信とは無関係なダミーパケットを挿入することで、IPG を調整する方式を提案する。この方式は、ソフトウェア制御によって精密かつ任意長の IPG を制御できる利点がある。例えば、帯域を半分にする場合は、実パケット間に実パケットと同じサイズのダミーパケットを挿入すればよい。以降、このダミーパケットをギャップパケットと呼ぶ。また、ギャップパケット以外の本来送信すべきパケットを実パケットと呼ぶ。

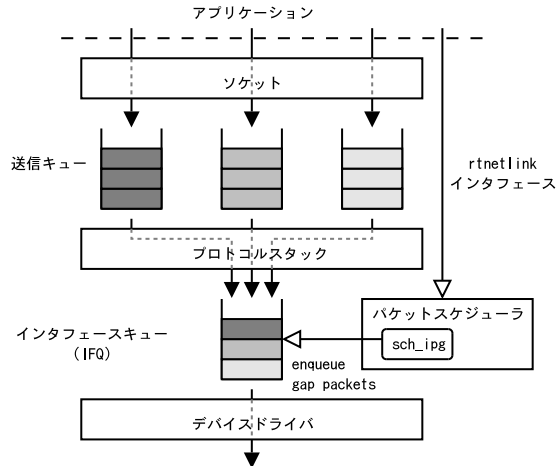


図 4 実装の概要

3.2 sch_ipg モジュール

本方式をデバイスインタフェース毎に存在するインタフェースキュー (以降、IFQ と呼ぶ) を制御するパケットスケジューリングポリシー (QDisc: Queue Discipline) として Linux 上に実装した。

実装の概要を図 4 に示す。アプリケーションからの送信データは、ソケットごとの送信キューにキューイングされ、TCP/IP プロトコルスタックにおいて IP パケット化される。そして、IP パケットはデバイスインタフェースごとのインタフェースキューにキューイングされる。Linux ではインタフェースキューに対するキューイングポリシーはカーネルモジュールとしてプラグイン可能であり、単純な FIFO から Class Based Queuing¹⁵⁾、Stochastic Fairness Queuing¹⁶⁾ など、多くのポリシーが標準で提供されている。

上記の仕組みを利用し、提案手法をカーネルモジュール sch_ipg として実装した。sch_ipg は、プロトコルスタックから IFQ の enqueue 関数が呼ばれたときに、IPG を計算し、実パケットと同時にギャップパケットも IFQ に挿入する。

QDisc として実装することで、ユーザーレベルからルーティングソケット (rtnetlink) インタフェース経由で、パラメータ変更などの操作が実行できる。rtnetlink はルーティング用のインタフェースであるため、ボトルネックリンクごとに設定が可能であり、デバイスに依存しない機構として実現できる利点がある。

sch_ipg を利用するためにカーネルの再コンパイルは不要であり、モジュールをインストールするだけで利用可能になる。そして、プログラミングレベルだけではなく、既存のコマンドを流用することで、コマンドレベルで送信レートなどのパラメータを設定できる。例えば、次のように tc コマンドを実行し、送信レートを設定できる。

```
# /sbin/tc qdisc add dev eth0 root ipg rate 62.5mbps
```

3.3 IPG の計算

GbE の場合、目的の送信レート (*TargetRate*) に設定するために必要な IPG は、次式から求めることができる。なお、*Hd1* はイーサネットのヘッダに FCS(Frame Check Sequence) を加えた 18 バイト、*Hd2* は *Hd1* にさらにプリアンプルを加えた 26 バイトである。

$$IPG' = \frac{(MTU + Hd1) \times 125}{TargetRate} - (MTU + Hd2)$$

$$IPG = \max(IPG', 12)$$

3.4 ギャップパケットフォーマット

ギャップパケットは IPG 以外に他通信への副作用を与えないことが必要である。このためにはスイッチの入力ポートでギャップパケットが破棄される必要がある。そこで、現在の実装では、ギャップパケットとして IEEE 802.3x フロー制御で定義されている PAUSE パケットを使用している。ホストとスイッチ間のフロー制御が無効の場合、PAUSE パケットはスイッチで破棄される。一方、フロー制御が有効の場合は、PAUSE 時間 0 とすることにより、副作用を回避している。

また、GbE では MTU が 1500 バイト以上のジャンボフレームを送信することが可能であり、一般的にジャンボフレーム対応の NIC は 9000 バイトのパケットを送信することができる。そこで、IPG が 9000 バイトより小さい場合は、単一のギャップパケットを送出し、それ以上の場合は、複数個のギャップパケットを送出することで任意長の IPG に対応する。

4. 実験

4.1 実験環境

提案手法を評価するために、ボトルネックリンクが存在する広域ネットワークを模擬した実験環境を用意した。ネットワーク構成を図 5 に示す。ホストとスイッチ (Dell PowerConnect 5224) 間、スイッチ間に、それぞれ GNET-1 を配置した。GNET-1 は FPGA と GbE ポートを四つ、そしてポートごとに 18MB の SRAM を持ち、ハードウェアロジックによる正確で細粒度な通信遅延の模擬や、送信レート制御などを実現している。

図 5 における GNET-1(C) では 62.5MB/s の帯域、200ms の往復通信遅延、そして TailDrop ルータのパケットロス挙動を模擬する。さらに、GNET-1 は通信挙動に影響をあたえることなく、ワイヤレートでパケットをキャプチャすることが可能である。そこで、GNET-1(A), (B) はパケットキャプチャおよび 32ns の解像度による詳細バンド幅測定に利用する。

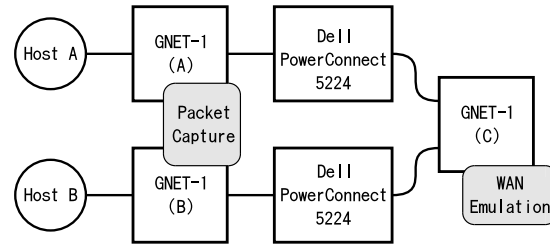


図 5 実験ネットワーク構成

表 1 ホストのハードウェア仕様

Processor	Pentium4 2.8GHz
Mother Board	Intel D865GLC
Main Memory	1GB (DDR400)
NIC	Intel 82547EI (CSA)

表 2 OS 設定

OS	RedHat Linux 9
Kernel	Linux 2.4.26 + Web100 2.3.8
Intel PRO/1000 Driver	5.2.30.1-k1
TCP Protocol	Scalable TCP WAD_IFQ
Max Socket Buffer	32MB

各ホストにおけるハードウェアの諸元を表 1 に、ソフトウェア構成を表 2 に示す。

カーネルは Linux 2.4.26 をベースに Web100¹⁴⁾ パッチを適用し、TCP 輻輳制御方式として Scalable TCP を用いた。Scalable TCP の実装は、Web100 パッチに同梱されているものを使用した。なお、Web100 パッチは、プロトコルスタックにフックを挿入することで、TCP コネクション単位での輻輳ウィンドウ、スロースタート閾値、広告ウィンドウなどの統計情報を取得し、proc ファイルシステムを介してユーザアプリケーションに提供する。

また、遅延が大きなネットワークにおいて、IFQ あふれをパケットロスと同様に輻輳として扱おうと、必要以上に輻輳ウィンドウを制限することになる。そこで、Web100 拡張の WAD_IFQ を有効にし、IFQ あふれを輻輳と扱わない設定とした¹⁾。これは IFQ 長パラメータである txqueuelen を大きくすることと同等の効果がある。さらに、ソケットバッファ管理に関しては、デフォルトカーネルの挙動と一致させるために、Web100 バッファオートチューニングは無効とした。最大ソケットバッファサイズは遅延が 200ms でも十分な性能が得られるように 32MB とした。

4.2 パケット送信間隔

3.3 節で示した式のように、IPG の変更による送信レートの制御、さらにパケット送信間隔の平滑化について確認した。

まず、ギャップパケットサイズを変更した場合の帯

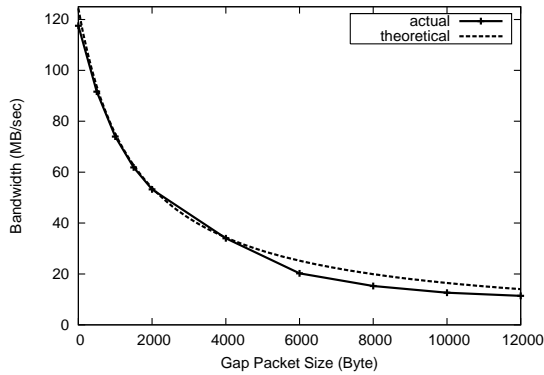


図 6 ギャップパケットサイズと帯域

域の変化を図 6 に示す。実線が実測値、点線が理論値である。パケットサイズが 6000 バイト以上の場合に、理論値と 2.5MB/s 程度の開きが見られるが、ほぼ理論値通りの結果を示している[☆]。なお、遅延がない場合も、同様な結果となる。

続いて、ペーシングによるパケット送信間隔の平滑化を確認するために、iperf 実行時のパケットを GNET-1(A) によってキャプチャし、パケット送信時間とシーケンス番号の対応を調べた。なお、結果はスロースタート後の定常状態のものであり、シーケンス番号は両方式を比較しやすいように正規化している。以降、ペーシングを有効にした場合を pacing 方式、無効にした場合を stock 方式と呼ぶ。

両方式におけるシーケンス番号の遷移を図 7 に示す。これより pacing 方式は約 60MB/s と均一の傾きでシーケンス番号が増加しているのに対して、stock 方式は約 120MB/s と、pacing 方式の倍の傾きで通信と停止を繰り返すという挙動を示している。さらに、ms 以下の解像度でパケット送信間隔を調べたところ、pacing 方式では 24us、stock 方式では 12us になっていることがわかった。2.2 節で述べたように、送信レート 62.5MB/s でペーシングする場合はパケット送信間隔が 24us になればよいので、提案手法が精密にペーシングできていることがわかる。

4.3 TailDrop 環境におけるペーシングの効果

4.2 節の実験では、バースト送信が発生しても十分吸収できるバッファを GNET-1 が持っているため、パケットロスが発生しなかった。そこで、GNET-1 で TailDrop ルータの挙動を模擬することで、バースト送信によるパケットロスとペーシングの効果について実験した。具体的には TailDrop サイズを変更しながら、iperf による通信を 5 分間行った場合の、平均帯域、TailDrop パケット数を測定した。TailDrop サイズが

[☆] 予備実験として、デバイスドライバ内に本方式を実装したが、このように明らかな差異は見られなかった²⁾。今後の調査が必要である。

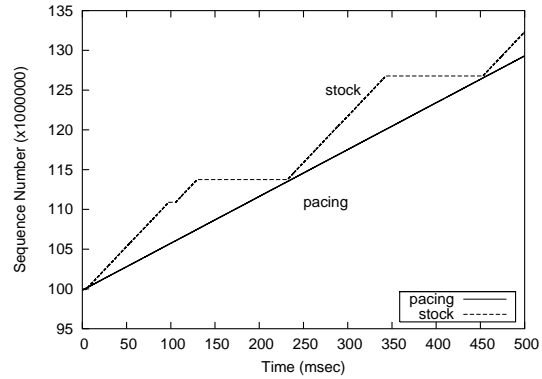


図 7 送信シーケンス番号の遷移

表 3 TailDrop 環境における帯域 (Scalable + Pacing)

TailDrop size (KB)	Bandwidth (MB/s)	TailDrops
512	56.5	0
1024	56.6	0
2048	56.8	0
4096	57.3	0

表 4 TailDrop 環境における帯域 (Scalable)

TailDrop size (KB)	Bandwidth (MB/s)	TailDrops
512	5.0	2263
1024	1.7	5473
2048	9.3	18635
4096	49.3	1146

大きいことはボトルネックリンクルータのバッファサイズが大きいことを意味し、バースト転送に対する耐性がある。逆に、TailDrop サイズが小さい場合にも送信レートが低下しないことは、バースト転送が起きていないことを意味する。

pacing 方式の結果を表 3 に、stock 方式の結果を表 4 に示す。なお、iperf の結果は TCP のスループットであるが、データリンク層でペーシングしているため、そのヘッダサイズを考慮する必要がある^{☆☆}。

pacing 方式では、ペーシングによって指定した帯域に対して精密にレート制御できており、TailDrop ルータにおけるパケットロスは発生していない。一方、stock 方式ではバースト送信によりパケットロスが頻発し、TailDrop サイズが 2048KB 以下の場合では、ほとんど期待される性能が出なかった。

参考として標準 TCP での結果を表 5 に示すが、遅延が大きいにもかかわらず、Scalable TCP の平均帯域を上回っている。両者の TailDrop 回数がほぼ同じ場合でも標準 TCP の帯域が上回っていることは、標

^{☆☆} iperf による帯域が 59.4 MB/s を示しているとき、図 5 中の GNET-1(C) で測定したイーサネットレベルの帯域は 61.8 MB/s であった。

表 5 TailDrop 環境における帯域 (Standard)

TailDrop size (KB)	Bandwidth (MB/s)	TailDrops
512	24.4	521
1024	27.1	3012
2048	45.6	2063
4096	53.5	1278

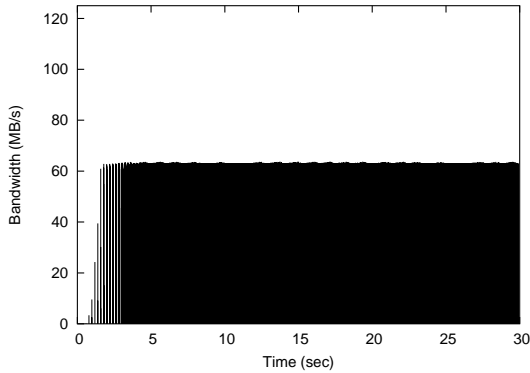


図 8 ペーシング有効時の送信帯域の変化

準 TCP の方がより多くのパケットロスを高速再送によって救えていることを意味する。Scalable TCP では、パケットロス後に輻輳ウィンドウの回復を早めるために、ACK 受信時の輻輳ウィンドウ増加量を大きくしている。したがって、Scalable TCP のバースト性は高くなり、パケットロス数だけでなく、バースト的なパケットロスが増えたと考える。

より細粒度での帯域の挙動を調べるために、図 5 中の GNET-1(A) において 500us の解像度で送信帯域を測定した。なお、TailDrop サイズは 4096KB である。

pacing 方式における結果を図 8 に、stock 方式における結果を図 9 に示す。これより pacing 方式では、ボトルネックリンクの帯域以上の送信は行われず、送信が平滑化されているのに対して、stock 方式では、ボトルネックリンクの帯域を越えるバースト送信によるパケットロスが発生している。

この結果より、提案手法がバースト送信を抑え、Tail-Drop 環境における TCP 通信性能の向上に効果があることが推測できる。特に ScalableTCP など、輻輳検出後に輻輳ウィンドウ回復を積極的に行う輻輳制御方式では、バースト送信の危険性が高いため、有効であると考えられる。

4.4 メモリ帯域へ与える影響

NIC はメインメモリから DMA することによりパケットを送出している。また、メインメモリへのアクセスは CPU からのアクセスと競合するため、DMA が増えると CPU からのアクセスが遅くなり、アプリケーションの実効性能に影響を与える可能性がある。

ギャップパケットも DMA を使って送出しており、実際の転送量が同じならば、ギャップパケット分の DMA

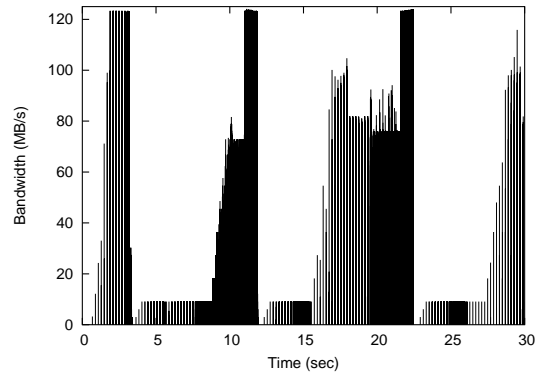


図 9 ペーシング無効時の送信帯域の変化

が増える。そこで、提案手法適用時におけるメモリ帯域への影響を評価した。

評価には、iperf によるバルク転送をバックグラウンドで実行し、1MB のバッファを memcpy 関数によって繰り返しコピーするベンチマークプログラムを使用した。測定条件は次に示す四つであり、ベンチマーク実行の有無による iperf の結果に違いはなかった。

- mbench only
ベンチマーク単体での実行
- half rate (sockbuf)
ソケットバッファの制限により 62.5MB/s で送信
- half rate (pacing)
提案手法により 62.5MB/s で送信
- full rate
制限なしに iperf を実行

実パケットの転送レートは half rate (pacing) と half rate (sockbuf) が等しく、実パケットにギャップパケットを加えた転送レートは half rate (pacing) と full rate が等しい。なお、half rate (sockbuf) はソケットバッファを 10MB に制限することによって、送信レートを半分にした。理論的には $62.5\text{MB/s} \times 0.2\text{s}$ から、12.5MB に制限すればよいが、今回は実測値から経験的に最大ソケットバッファサイズを決定した。

結果を表 6 に示す。mbench only, half rate (sockbuf), full rate から送信レートの上昇に比例して、メモリ帯域が低下している。これは NIC における DMA 転送やプロトコルスタック内の処理が増加するからである。また、half rate (pacing) のメモリ帯域は、half rate (sockbuf) を下回ったが、full rate は上回った。これはギャップパケットのペイロード本体は初期化時に確保し、IFQ にキューイングする場合に skb_clone 関数によって管理ヘッダだけを複製しているため、ペイロード部分に対するメモリ確保、解放処理が不用であるからと考える。

これより、ギャップパケットの送信分、メモリ帯域は低下は避けられないが、skb_clone 関数を使った処

表 6 メモリ帯域への影響

条件	メモリ帯域 (MB/s)
mbench only	1935.5
half rate (sockbuf)	1236.3
half rate (pacing)	1043.0
full rate	923.0

理削減により低下幅を 25 % から 15 % に改善できることがわかった。

5. 議 論

本稿では単一ストリーム、単一ボトルネックリンクの条件下で評価を行ったが、利用可能帯域や遅延が異なるリンクが複数存在する場合は、単純にギャップパケットを挿入するだけの方式では制御できない。例えば、複数ホストが一つのボトルネックリンクを共有する場合は、各ホストでペーシングすることで、ボトルネックルータへの負荷を軽減できると考えるが、単一ホストから複数のボトルネックリンクへ通信を行う場合は、現時点で対応できない。また、ギャップパケットとして PAUSE パケットを利用しているために、直近のスイッチまではペーシングされるが、それより先のネットワークに対する効果はない。

前者の問題に対しては、複数ストリームが存在する場合は、ギャップパケットの代わりに他ストリームに対するパケットをスケジューリングすることで、ボトルネックごとに IPG を調整する方法が考えられる。実装方法としては、QDisc を階層 (クラスフル) 化することで対応できる。つまり、ボトルネックリンクごとにサブ QDisc を用意し、パケットをフィルタリングする。サブ QDisc はそれぞれ転送レートを保持しており、ルート QDisc はそれらの転送レートにしたがって、パケットを選択、送出する。この際、メイン QDisc は、IPG の必要性に応じてギャップパケットを挿入する。

後者の問題に対しては、MPLS ネットワークによる QoS 制御など、ネットワーク側の帯域保証機構と組み合わせることで解決できると考える。提案手法はあくまで送信側のエンドホストでバースト転送を抑えるための手法である。

また、本稿では TCP レベルでの有効帯域の推定に対する議論を行わなかったが、トランスポートレイヤとデータリンクレイヤでの連携が必要であると考えられる。この点に関しては、FAST や TCP/Vegas⁸⁾ のようなキューイング遅延ベースの輻輳制御方式とペーシングによるバースト転送の抑制を組み合わせる方法が有効であると考えられる。

6. 関連研究

提案手法ではギャップパケットの挿入によってペーシ

ングを実現したが、3.1 節で述べたように、ソフトウェアタイマを利用した実装方法も考えられ、D.Lacamera による実装¹¹⁾ や、中村らによる実装¹²⁾¹³⁾ が存在する。

ソフトウェアタイマを利用して、パケット送信間隔を「RTT / 輻輳ウィンドウ」時間毎になるように制御するためには、その実装に高解像度のタイマが必要になる。Linux 2.6 カーネルでは、ソフトウェアタイマの解像度が 10ms から 1ms となり、より高精度なタイマ制御が可能となった。しかし、GbE において IPG を 12 バイトに設定する場合、送信間隔を 96ns に制御するためには、1ms の解像度でも十分ではない。

中村らは、タイマ解像度がパケット送信間隔より粗い場合に、低負荷で実現可能な Clustered Packet Spacing を提案し、スロースタートに対する効果を評価している。Clustered Packet Spacing では、ソフトウェアタイマを用いて、パケットの送信間隔を粗く均一化することで、バースト送信を分割している。しかし、副作用として、他ホストの通信とボトルネックを共有した場合、ボトルネックリンクルータに対する負荷の増大が問題と指摘されている。

このようなソフトウェアタイマを用いた実装に対して、提案手法ではハードウェアによる IPG 制御と同様な精密なペーシングを実現しており、複数ホスト間でボトルネックリンクを共有した場合でも、ボトルネックリンクルータへの負荷を軽減できると考える。

7. ま と め

本稿では、送信レートに合わせて任意長のギャップパケットを挿入するパケットスケジューラによる、ソフトウェア精密ペーシング機構を提案し、その評価について述べた。その結果、本方式によって精密なペーシングが可能であること、TailDrop ルータ環境において TCP/IP 通信の性能を改善できること、またメモリ帯域への影響を明らかにした。

今後は、複数のストリーム、複数のボトルネックリンクが存在する環境において、ギャップパケット以外に、他ストリームのパケットも利用しながら、IPG を調整するようにスケジューリング方式を改良する予定である。

謝辞 なお、本研究の一部は文部科学省「経済活性化のための重点技術開発プロジェクト」の一環として実施している超高速コンピュータ網形成プロジェクト (NAREGI: National Research Grid Initiative) による。

参 考 文 献

- 1) 高野了成, 石川裕, 工藤知宏, 松田元彦, 児玉祐悦, 手塚宏史, “並列アプリケーション実行における TCP/IP 通信挙動の解析”, インターネットコンファレンス 2003, October 2003.
- 2) 高野了成, 工藤知宏, 児玉祐悦, 松田元彦, 手

- 塚宏史, 石川裕, “GridMPIのためのTCP/IP輻輳制御実装方式の検討”, 情報処理学会, SWoPP 2004, August 2004.
- 3) 建部修見, 小川宏高, 児玉祐悦, 工藤知宏, 高野了成, 関口智嗣, “グリッドデータファームとGNET-1による日米間高速ファイル複製”, 情報処理学会, HOKKE 2004, March 2004.
- 4) Y. Kodama, T. Kudoh, R. Takano, H. Sato, O. Tatebe and S. Sekiguchi, “GNET-1: Gigabit Ethernet Network Testbed”, IEEE, Cluster 2004, September 2004.
- 5) T. Kelly, “Scalable TCP: Improving Performance in Highspeed Wide Area Networks”, Computer Communication Review, Vol.32, No.2, April 2003.
- 6) S. Floyd, “HighSpeed TCP for Large Congestion Windows”, Internet Draft, <http://www.ietf.org/internet-drafts/draft-ietf-tsvwg-highspeed-00.txt>, Work in progress, July 2003.
- 7) C.Jin, D.X.Wei and S.H.Low, “FAST TCP: Motivation, Architecture, Algorithms, Performance”, IEEE INFOCOM, March 2004.
- 8) L.S.Brakmo and L.L.Peterson, “TCP Vegas: End to End Congestion Avoidance on a Global Internet”, IEEE Journal on Selected Areas in Communication, Vol.13, No.8, pp.1465-1480, October 1995.
- 9) J.C.Mogul, “Observing TCP Dynamics in Real Networks”, ACM SIGCOMM'92, pp.305-317, April 1992.
- 10) A.Aggarwal, S.Savage and T.Anderson, “Understanding the performance of TCP pacing”, IEEE INFOCOM, pp.1157-1165, March 2000.
- 11) D.Lacamera, “TCP Spacing Linux Implementation”, http://www.danielinux.net/projects/tcp_spacing.php
- 12) 中村誠, 稲葉真理, 平木敬, “ギガビットイーサネット上での遠距離TCP通信におけるPacket Spacing”, 電子情報通信学会, IA2003, pp.13-18, October 2003.
- 13) 中村誠, 亀澤寛之, 稲葉真理, 平木敬, “協調動作する並列TCPストリームへのPacket Spacingの適用とその評価”, 情報処理学会, SWoPP 2004, August 2004.
- 14) M.Mathis, J.Heffner and R.Reddy, “Web100: Extended TCP Instrumentation for Research, Education and Diagnosis”, ACM Computer Communications Review, Vol.33, No.3, July 2003. <http://www.web100.org/>.
- 15) S.Floyd and V.Jacobson, “Link-sharing and Resource Management Models for Packet Networks”, IEEE/ACM Transactions on Networking, Vol.3, No.4, pp.365-386, August 1995.
- 16) P.McKenney, “Stochastic Fairness Queuing”, IEEE INFOCOM, pp.733-740, June 1990.