PEPを用いたネットワークにおけるフロー制御の動作とその性能評価

鹿間　敏弘*

A Mechanism of a Flow Control in a Network using a PEP and Its Performance Evaluation

Toshihiro Shikama

In IP networks, where bandwidth-delay product is large, a PEP (Performance Enhancing Proxy) is effective for improving the performance of TCP. The PEP splits an end-to-end TCP connection into two parts; it forwards received data from one TCP connection to another. If the maximum throughput of one TCP connection is larger than the other, the PEP has to buffer forwarding data and perform a flow control to adjust the throughput. To confirm the mechanism of the flow control and to evaluate the required volume of buffers, we investigate the flow control by experiments using Linux and report their results.

Keywords: PEP, TCP, Split connection, Flow control, TCP spoofing

1. まえがき

インターネットなどのIPネットワークではエンドツーエンドでTCPを用いる通信が大きな部分を占めるが，衛星通信やパケットの伝送誤りが発生する高速の無線通信など帯域遅延時間積の大きな通信環境では，既存端末の TCP では性能が得られない問題がある．衛星通信の場合は，主に大きな伝搬遅延時間によりウィンドウサイズを大きくしなければならないが，このためにはTCPのパラメータ変更が必要となる．無線通信を含むIPネットワークでは，伝送誤りの発生によるパケット喪失の影響が大きい．TCPではパケットの喪失を検出すると，ルータにおけるバッファオーバーフローなどネットワーク内の輻輳により喪失が発生したものと見なし輻輳制御を行う．このため伝送誤りによりパケット喪失が発生した場合，不要な輻輳制御が行われ，TCPのスループットが低下する[1]．以上のような問題に対し既存端末の TCP をそのまま使用する解決策として，TCPプロキシによりTCPコネクションを分割し中継を使うTCPコネクション分割方式が考えられる．このような方式はTCPスプーフィングと呼ばれることもある．TCPプロキシは一般にPEP (Performance Enhancing Proxy) と呼ばれ，本稿においては以下このPEPの用語を用いる．PEPは衛星通信においてTCPの性能を向上させるために広く使用されており[2]，衛星自体にこの機能を持たせる方式も研究されている[3]．また地上無線によるネットワークにおいても効果が報告されている[4]．PEPには多くのバリエーションが存在し， TCPコネクションを終端しないPEPも研究されている[5]．PEPの方式についてはRFC3135にまとめられている[6]．

＊ 電気電子情報工学科

本稿が対象とするPEPは，図 1に示すエンドツーエンドのコネクションを二つの TCP コネクションに分割し，PEPが両方のコネクションを終端する方式を前提としている．PEPは一方のTCPコネクションから受信したデータを他方の TCP コネクションに中継する．PEPでは二つのTCPコネクションを連携させてエンドツーエンドのフロー制御を実現しなければならない．具体的には，スループットの大きい TCP コネクションで受信したデータをスループットの小さい TCP コネクションに中継する場合，送信元のデータ送信をスループットの小さいTCPコネクションに整合させる必要がある．この実現方法はTCPの告知ウィンドウ[7]を動的に制御して行うのが一般的と考えられるが，実現方式に依存し，一般には公開されていない．また，スループットの差がある場合には，PEPにおいて中継データをバッファリングする必要がある．PEPが多数の TCP コネクションをサポートする場合には，必要なバッファ量を予測しなければならないが，PEPにおけるバッファリングについて知見が必要である．

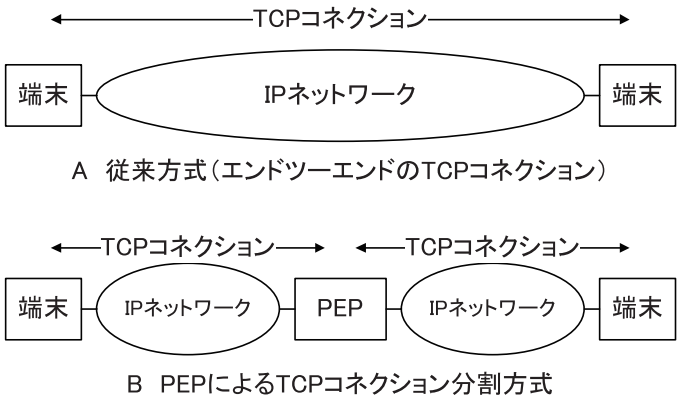


図 1　TCP コネクション分割方式

我々はこれまで伝送誤りが発生する無線通信の場合について， PEP の効果を評価してきた[8][9]．PEPにより無線区間でパケットの喪失が発生する場合において，TCPのスループットの低下が抑えられることを確認した．また，無線区間での伝送誤りを再送により回復する場合，TCPの受信側におけるリアセンブリング処理により PEP の出力においてバースト的なトラヒックが発生することを明らかにした．このような評価は，主にネットワークシミュレータ ns-2 を利用して行ってきた[10]．しかし，ns-2 の TCP そのままでは動的に告知ウィンドウを制御することができず，スループットが大きい側から小さい側への中継は正確な評価を行うことができなかった．これを解決するために，ns-2 に機能追加を行う場合，実システムにおけるフロー制御の動作をモデル化するための知見が必要となる．以上の背景から本稿では，Linuxにより簡単な処理でPEPの動作を実現し，TCP コネクション間のフロー制御の動作を観測して，PEP においてどの程度のバッファが使用されているのか，およびどのようにモデル化するのが適当か調査検討することを目的としている．

本稿では，2章でPEPにおけるフロー制御の動作について説明し，3章でLinuxを用いた実験方法と実験結果及び考察を述べ，最後に4章で全体をまとめる．

2. PEPにおけるフロー制御

先に述べたように，PEPがスループットの大きなTCPコネクションからスループットの小さいTCPコネクションに中継する場合，スループットの大きなTCPコネクションの送信速度をスループットの小さな TCP コネクションに整合させるようフロー制御を行う必要がある．具体的にはスループットの大きなコネクションにおいて，送信側にPEPが送信するACKの告知ウィン

ドウを制御して調整することになる．すなわち，スループットの大きいコネクションから小さいコネクションへの中継データが PEP のバッファに滞留するので，滞留する中継データ量が一定値を超えたらTCPのコネクションにおけるACKの告知ウィンドウを小さくする．または，滞留する中継データ量に応じて告知ウィンドウの調整を行う．このような TCP 告知ウィンドウの値を調整する方法としては種々の方式が考えられるが，実装に依存する．PEPをLinuxなどのOSのカーネル空間で実現する方法も考えられるが，最も容易な方法はアプリケーションとしてPEPを実現する方法である．ここで，アプリケーションは通常のソケットインタフェースを利用してTCPの送受信を行う．具体的には，以下のような動作になると考えられる．

LinuxにおいてTCPのコネクションを確立する度に中継用のプロセスまたはスレッド(以後はスレッドを仮定する）を二つ用意し，双方向あるTCPの通信方向ごとに一つの中継スレッドで，一方のTCPコネクションからの受信と他方のTCPコネクションへの送信をソケットインタフェースを介して行う．スレッドの動作及びフロー制御は次のように行われる．

① Linuxのカーネルは TCP コネクションごとに送信側と受信側でバッファを用意する．中継スレッドがスループットの大きな中継元 TCP コネクションでデータを受信すると，直ちにスループットの小さい中継先TCPコネクションへ送信要求を行う．

② 送信要求された中継データは中継先TCPコネクションの送信バッファに書き込まれ，書き込みが完了すると，中継スレッドは次の受信要求を中継元のTCPコネクションに対し行う．

③ スループットの小さい中継先の送信バッファが一杯になると中継スレッドは送信要求を行ってもバッファに書き込めないため，書き込み処理が完了するまで停止する．

④ この間にスループットの大きい中継元 TCP コネクションで受信したデータは中継スレッドが読み出さないため，中継元TCPコネクションの受信バッファに滞留する．

⑤ 受信バッファの大きさまで中継データが滞留すると，これ以上受信できなくなるので，これを防ぐために中継元 TCP コネクションの告知ウィンドウの値を小さくして，受信バッファ溢れを防ぐ．

⑥ 告知ウィンドウの値が小さくなると，中継元TCPコネクションの送信側で送信データ量が絞られる．

以上のようなメカニズムにより，中継スレッドは告知ウィンドウを意識することなくデータの中継とフロー制御を行うことができる．ここで，中継スレッドは中継データの受信と送信を繰り返すので同じバッファを再利用することができ，受信 1 回分のバッファのみを用意すれば良い．スループット差にともなうバッファリングは中継スレッドではなく，送信側の TCP と受信側のTCPに設けられたバッファで行われる．ソケットインタフェースを介したTCPの送受信においては TCP コネクションごとに図 2 に示す送信及び受信バッファがカーネル内で用いられる[11][12]．受信したセグメントのデータはカーネル空間内の受信待ちデータ用のバッファに蓄積され，中継スレッドが読み出し(RECV)を行うと，中継スレッドのバッファにコピーされる．

中継スレッドがこのバッファの送信要求(SEND)を行うと，バッファのデータは送信待ちデータのバッファにコピーされる．送信バッファはこの送信待ちデータを保持するとともに送信は完了して送達確認を待つデータ（再送の可能性があるデータ)を保持するためにも使用する．コピーが完了すると中継スレッドのバッファを次の読み出し(RECV)に使用できる．中継先 TCP のスループットが中継元 TCP のスループットより小さい場合，中継先 TCP の送信待ちデータ量が増加し，送信待ちデータのバッファが一杯となる．この場合に中継スレッドが送信要求(SEND)を行うと，中継データを送信待ちデータのバッファに書き込めないため，送信要求が完了しなくなる．送信要求が完了しないと中継元 TCP の受信待ちデータが読み出されず，受信待ちデータ量が増加し，受信バッファの空きが減少する．TCP の告知ウィンドウはこの受信バッファの空きの量を送信側に通知しており，送信側は告知ウィンドウが絞られたことにより，送信するデータ量を減少させる．実際は，TCP の送信は輻輳ウィンドウと告知ウィンドウの値の小さい値を使用する．以上のようなメカニズムにより，中継元と中継先の二つの TCP コネクション間の速度の整合が図られるものと考えられる．PEP による中継でバッファに滞留するデータ量は，中継先 TCP の送信バッファにおける送信待ちデータ量と中継元 TCP の受信バッファにおける受信待ちデータ量の和になる．

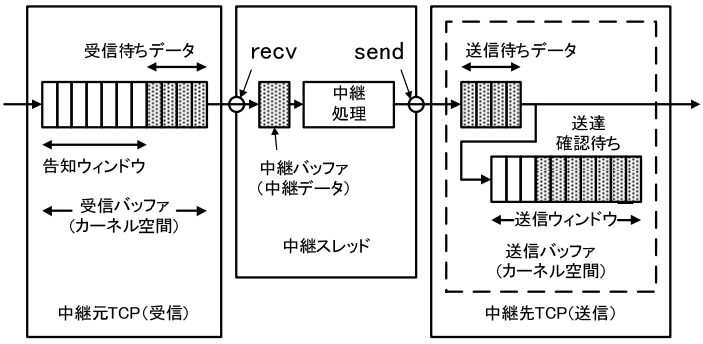


図 2　中継に使用する TCP のバッファ

3. Linux を用いた実験

3.1.　実験構成

PEP におけるフロー制御の動作を確認するため，Linux を用いて実際のプロトコルを用いた環境で評価を行った．図 3 は，評価モデルと実際の実験構成を示している．Linux を搭載した PC 3 台と FreeBSD を搭載した PC 1 台, 及び回線遅延シミュレータを用いて構成した．図 3 で左端の Linux PC を送信側, 右端の Linux PC を受信側としてデータを転送し，中央の Linux PC を PEP として左側の中継元ネットワークから右側の中継先ネットワークにデータを転送した. 中継元ネットワークの帯域を 50 Mbit/s, 遅延時間を 5 ms とし，この部分は FreeBSD 上で動作する Dummynet [13]を用いて実現した．PEP は Linux のアプリケーションとして実現し，先に説明したように 1 本の TCP コネクションの片方向の中継を Linux の一つのスレッドで行った．TCP コネクションの双方向通信は独立なスレッドを二つ用いて実現している．右側の中継先のネットワークの遅延時間は 40ms とし，帯域を 2，5 Mbit/s と変えて中継元

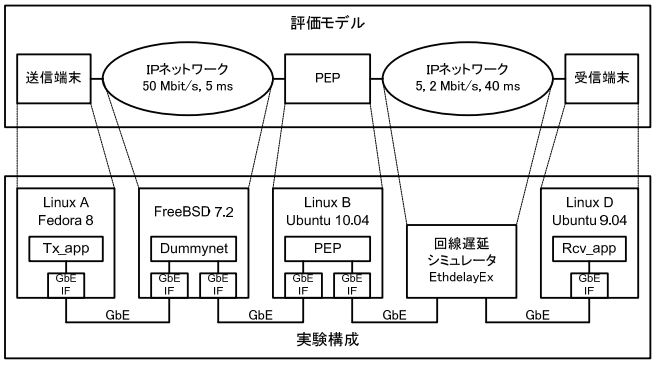


図 3　実験システムの構成

のネットワークに対し帯域が十分に小さい条件を基本とした．中継先ネットワークは回線遅延シミュレータ EthdelayEx [14]を使用して実現した．実験は 20 MB (バイト)のデータ転送を行い，PEP の中継スレッドにおける中継バッファの大きさは 3000 B とした．PEP において，パケット・アナライザ Wireshark [15]を用いて送受信する TCP のセグメントを取得し，告知ウィンドウの時間変化や PEP に滞留する中継データ量を求めた．

3.2. 実験結果

(1) 中継先ネットワーク帯域が 5 Mbit/s の場合

図 4 は中継元の TCP コネクションにおいて PEP が送信する ACK に含まれる告知ウィンドウ値の時間変化を示している．ここで，中継先ネットワークの帯域は 5 Mbit/s である．TCP コネクション確立時にウィンドウスケールオプションが設定されており，値は 6 となっている．図から，パルス的な変化を除くと，大部分の時間は告知ウィンドウが小さな値となっているが，データ転送が終了すると 1.14 MB 程度の値に増加している．図 5 は時刻 4 sec から 8 sec の間，及び時刻 14 sec から 17 sec の間の告知ウィンドウの値の変化を拡大したものである．時刻 4 sec から 8 sec の間，告知ウィンドウは定常的に小さな値を取る．これは中継元 TCP コネクションにおいて，10 ms の RTT (Round Trip Time)の間にスループットの平均が 5 Mbit に近い値となるよう告知ウィンドウの大きさが調整されているためと考えられる．時刻 14 sec から 17 sec の間では，告知ウィンドウの値が 0 となる状態が 1.5 sec 程度継続しており，送信側からウィンドウ探索[7]が行われるシーケンスが発生している．図 4 においてデータ転送開始直後及び時刻 14.5 sec などで告知ウィンドウがピーク状に大きくなる現象が見られる．

時刻 14.5 sec では告知ウィンドウのピーク値は 1.8 MB 程度となっている．図 6 は PEP に滞留する中継データ量の時間変化を示している．このデータ量には中継データの他に中継先 TCP における送達確認待ちのデータ量も含まれている．最大で 4 MB を超える中継データが 1 本の TCP コネクションで滞留していることが分かる．中継データ量は実験前に予想していた値より大きい数値となっており，必要以上に大量の中継データが滞留している．図 4 に見られる告知ウィンドウのパルス的な増加の原因解明は今後の課題であるが，Linux の TCP において送信バッファや受信バッファの大きさが動的に自動調整されることによる影響と考えられる[15]. 図 4 及び図 6 から，中継元 TCP コネクションから PEP へのデータ転送は時刻 25 sec で終了し，時刻 25 sec から時刻 33 sec の間に PEP に

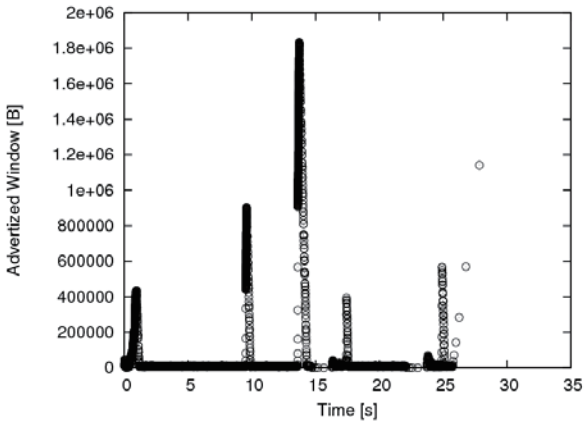
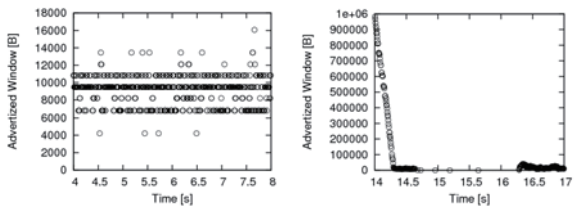


図 4 告知ウィンドウの時間変化（中継先の帯域が 5 Mbit/s の場合）



(a) 4 sec から 8 sec の間 (b) 14 sec から 17 sec の間

図 5 告知ウィンドウの時間変化（拡大）

滞留している約 4.5 MB のデータが中継先 TCP コネクションにおいて 5 Mbit/s の速度で送信されている. 4 MB のデータを 5 Mbit/s で転送するには 7.2 sec を要するので, 実験結果はこれと整合している．また，20 MB の全データの転送に要した時間は約 33 sec で，5 Mbit/s に近いスループットが得られている.

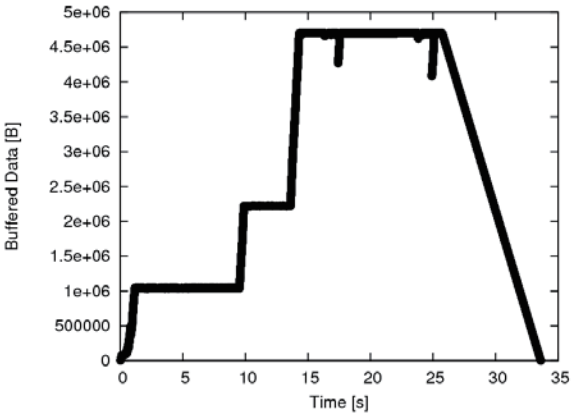


図 6　PEP に滞留する中継データ量の時間変化（中継先の帯域が 5 Mbit/s の場合）

(2) 中継先の帯域を 2 Mbit/s にした場合

図 7 は中継先ネットワークの帯域を 2 Mbit/s に減少させた場合で, 中継元 TCP コネクションにおいて PEP が送信する ACK に含まれる告知ウィンドウの値の時間変化を示している．帯域が小さくなったことにより，中継元 TCP コネクションでのデータ転送に要する時間は，25 sec から 68 sec と増加している．図 8 は図　5 と同様に時刻 30 sec から 35 sec の間，及び時刻 37 sec から 45 sec の間の告知ウィンドウの値の変化を拡大したものである．時刻 30 sec から 35 sec の間の告知ウィンドウの値は平均的に図 5 の時刻 4 sec から 8 sec の値に比べ半分以下になっていることが分かる．これは中継先の速度が小さくなったので，スループットを抑えるために告知ウィンドウの平均的な大きさが減少したためと考えられる．以上から，PEP におけるフロー制御をシミュレーションする上では，TCP 間の中継用のバッファとそこに滞留するデータ量とを考慮し，中継用バッファの空きの大きさを告知ウィンドウとするようなモデル化を行えば良いと考えられる．図 7 においても，転送の開始直後から時刻 20 sec 付近など告知ウィンドウの値がパルス的に大きくなる現象が見られる．Linux の TCP において送信バッファや受信バッファの大きさが動的に自動調整されていることによる影響と考えられる．図 9 は PEP に滞留する中継データ量の時間変化を示している．この場合も，PEP において最大で 4.5 MB を超える中継データが 1 本の TCP コネクションで滞留している．実験は，中継先及び中継元ネットワークの帯域時間積が 64 KB 以下の条件で行っており，実験で観測された PEP に滞留する中継データ量や告知ウィンドウの大きさは必要以上に大きいと考えられる.

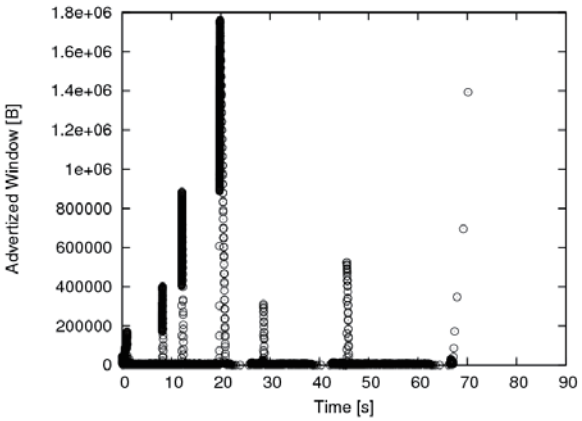


図 7　告知ウィンドウの時間変化（中継先の帯域が 2 Mbit/s の場合）

(3) 中継先の帯域が大きい場合

比較のため逆の場合についても測定を行った．図 10 と図 11 は中継元の帯域が 5 Mbit/s，中継先の帯域が 50 Mbit/s の場合について, それぞれ中継

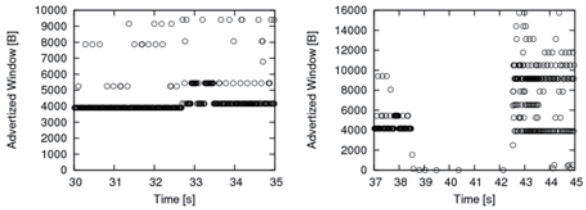


図 8　告知ウィンドウの時間変化（拡大）

元の TCP コネクションにおける告知ウィンドウの時間変化及び PEP でバッファされた中継データ量の変化を示している．この場合，基本的にはフロー制御の必要がないため，告知ウィンドウは大きな値を取ることが分かる．また，PEP において滞留する中継データ量も少ないが，パルス的に中継データ量が大きくなる現象が見られた．これは，中継スレッドによる受信データの読み出しが遅れることによるものと考えられる．

(4) 使用するバッファサイズを変更した場合

以上では TCP のパラメータに関して特に設定を行わずデフォルト値を用いて実験を行った．告知ウィンドウの値は 1.8 MB を超える場合が見られ，必要以上に大きな値と考えられる．ソケット API を利用すると TCP の送受信バッファサイズなどを変更できるので，送受信とも 64 KB に設定して実験を行った．図 12 は中継先ネットワーク帯域が 5 Mbit/s の場合の告知ウィンドウの時間変化を示している．告知ウィンドウの大きさが 100 KB 以下に抑えられているが，設定した 64 KB の大きさを超える場合が存在する．調査した結果，受信バッファサイズは指定した値の 2 倍に設定されるとの記述があり[16]，受信バッファサイズは 128 KB と考えられるので，不合理な結果とは言えない．図 13 は PEP に滞留する中継データ量の時間変化を示している．図 6 と比較すると中継データ量は大幅に減少しているが，それでも最大で 2.5MB を超えるデータが滞留している．これは，指定した送受信バッファサイズよりかなり大きく，詳細を調査する必要がある．

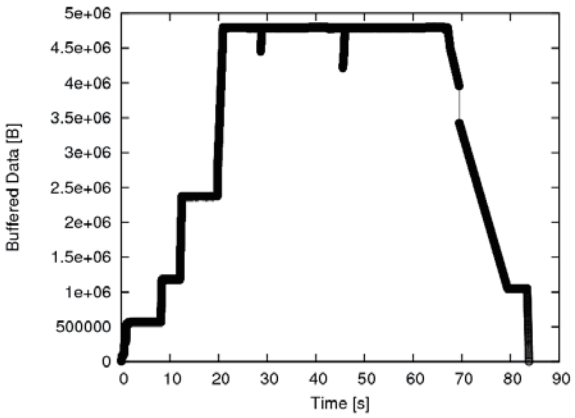


図 9　PEP に滞留する中継データ量の時間変化（中継先の帯域が 2 Mbit/s の場合）

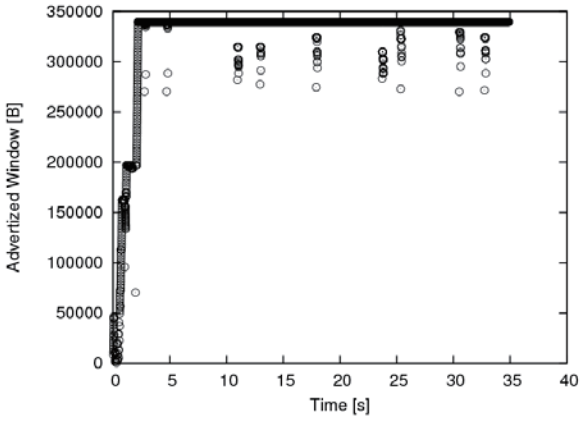


図 10　告知ウィンドウの時間変化（中継元の帯域が 5 Mbit/s，中継先の帯域が 50 Mbit/s の場合）

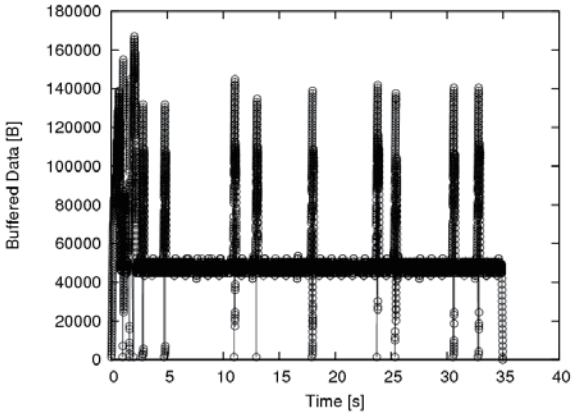


図 11　PEP でバッファされた中継データ量の時間変化（中継元の帯域が 5 Mbit/s，中継先の帯域が 50 Mbit/s の場合）

4. まとめ

TCP コネクション分割方式において，PEP におけるフロー制御の動作について Linux を用いた実験により動作を確認した．アプリケーションでソケットインタフェースを介してデータの中継を行ったが，カーネル空間において送信側及び受信側で必要以上に大量の中継データが滞留することが分かった．必要なバッファ量の削減は可能

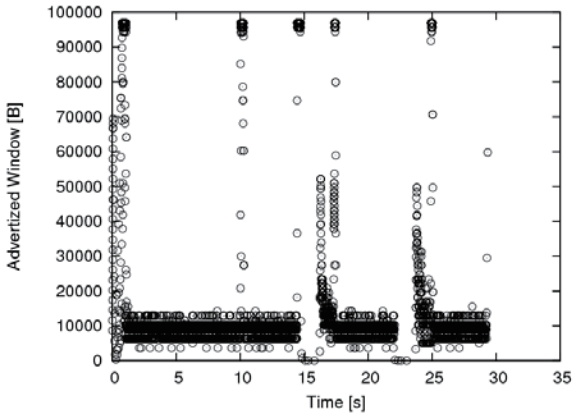


図 12　告知ウィンドウの時間変化（中継先の帯域が 5 Mbit/s の場合）

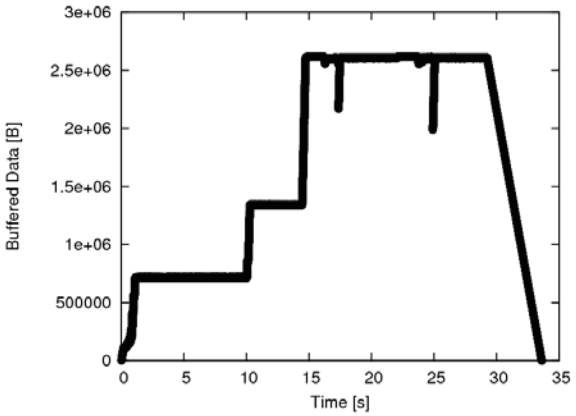


図 13　PEP に滞留する中継データ量の時間変化（中継先の帯域が 5 Mbit/s の場合）

と考えられ，具体的な方式を今後検討する．また，告知ウィンドウによるフロー制御は，受信バッファの空きの値を通知することにより行われていること，告知ウィンドウの大きさが 0 となる場合も存在することを確認した．今後，ns-2 などのネットワークシミュレータでモデル化する場合にはこのようなメカニズムを考慮する必要がある．実験において告知ウィンドウの大きさがパルス的に大きく変化する現象が見られた．Linux が TCP のバッファサイズを自動調整することによる影響と考えられる．また，中継先の帯域が大きい場合でも，パルス的に中継データが PEP に滞留する現象も見られた．今後の調査が必要である．

謝辞

本研究の一部は，科研費（20500079）の助成を受けたものである．ここに記して謝意を表す．

参考文献

[1] 松田崇弘，山本　幹，“無線ネットワークにおける TCP の研究動向と今後の課題,” 信学誌，vol.87，no.7，pp.589-594，July 2004.

[2] HUGHES, http://www.hughes.com/Pages/Default.aspx.

[3] M. Luglio, M. Y. Sanadidi, M. Gerla, and J. Stepank, “On-Board Satellite Split TCP Proxy,” IEEE JSAIC, vol.22, no.2, pp.362-370, Feb. 2004.

[4] M. Meyer, J. Sachs, and M. Holzke, “Performance Evaluation of a TCP Proxy in WCDMA Networks,” IEEE Wireless Communications, vol.10, no.5, pp.70-79, Oct. 2003.

[5] 和泉芳規，長田繁幸，横平徳美，“PEP による TCP の性能改善　～ 早期 ACK パケットの返送タイミングがスループットに及ぼす影響 ～,” 信学技報，NS2004-264，pp.103-106，Feb. 2005.

[6] J. Border, M. Kojo, J. Griner, G. Montenegro, and Z. Shelby, “Performance Enhancing Proxies Intended to Mitigate Link-Related Degradations,” RFC3135, June 2001.

[7] 村山公保，西田佳史，尾家祐二，“トランスポートプロトコル,” 岩波書店，2001.

[8] 鹿間敏弘，“TCP コネクション分割方式のバースト出力軽減方式の提案と性能評価,” 信学技報，NS2009-181，pp.113-118，March 2010.

[9] 鹿間敏弘，“無線リンクを含むネットワークにおける TCP プロキシのバースト出力軽減方式の評価,” DICOMO2010，pp.1019-1028，July 2010.

[10] Network Simulator – ns (version 2), http://www.isi.edu/nsnam/ns/.

[11] 高野了成，“HPC ユーザが知っておきたい TCP/IP の話,” SACSIS2009 チュートリアル，May 2009.

[12] H. Bhuiyan, M. McGinly, T. Li, M. Veeraraghavan, “TCP Implementation in Linux: A Brief Tutorial,” http://www.ece.virginia.edu/cheetah/documents/papers/TCPlinux.pdf.

[13] Dummynet, http://info.iet.unipi.it/~luigi/dummynet/.

[14] 回線遅延シミュレータ EthdealyEx, http://www.ncad.co.jp/contents/products/it/EthdelayEx/.

[15] WIRESHRAK, http://www.wireshark.org/.

[16] Linux におけるソケット機能の向上, http://www.ibm.com/developerworks/jp/linux/library/l-hisock/.

（平成 23 年 3 月 31 日受理）