Journal of the Faculty of Management and Information Systems,
Prefectural University of Hiroshima
2011 No.3 pp. 183 － 194

論　　文

広帯域ネットワーク環境における TCP 性能の一考察

陳　　春　祥・佐々木　宣　介

Perfomanace investigation of TCP in large BDP networks

Chun-Xiang CHEN and Nobusuke SASAKI

要　約

インターネットの TCP 通信において，フロー制御，ふくそう（輻輳）制御と誤り制御はいまだに 1 つの大きな課題である。特に長距離・広帯域のネットワーク（つまり，帯域と遅延の積 BDP（Bandwidth-Delay Product）が大きいネットワーク）においては，通信路に帯域の余裕があるにもかかわらず帯域を使い切れない問題が顕在化している。本稿では BDP の大きいネットワーク環境において，Opnet Modeler を用いて，TCP のウィンドウサイズ，受信バッファーサイズおよび往復遅延が TCP 性能に対する影響を検証すると共に，高速ネットワーク環境を取り巻く新しい技術と実装について考察する。

キーワード：広帯域ネットワーク，TCP 輻輳制御，TCP ウィンドウサイズ

Abstract

In this paper , we analyze the TCP performances in the network with large bandwidth-delay product (PBD) by the network simulator: OPNET Modeler 16.0. Especially, the influences of the receive buffer, window scaling, MSS and different congestion control schemes, are investigated. It is clarified that TCP receive window size may hamper the TCP performance in large BDP networks. By comparing with some new congestion control schemes, we give some comments on the implementation and management of the network with large bandwidth.

Key Words: TCP congestion control, TCP window size, High speed networks

1　はじめに

パケット交換方式のIP（Internet Protocol）ネットワークにおいて，TCP（Transmission Control Protocol）を用いて信頼性を必要とする通信を提供する仕組みになっているが，通信帯域を確保して品質を保証する回線交換方式と異なり，TCPは帯域を保証しないベストエフォート型通信である。通信に先立ち，TCPコネクションの確立（コネクション型通信と言われる）を行うが，送信側と受信側は正確にネットワークの状況を知る手段が実装されていない。代わりにスロースタートというメカニズムを採用してふくそう（輻輳）制御を行う。つまり最初に少量のデータ（パケット単位で1つか2つ程度）を送出し，受信側からの応答確認を待つ。応答確認がスムーズに戻れば，ネットワークは混雑していないと予測し，次に連続して送出するデータ量を増す。この方式はTCP Tahoeと呼ばれ，TCP Tahoeを改良したTCP Renoがいまもっとも普及している［1, 2］。

インターネットの普及につれ，トラフィック量が飛躍的に増加し，それに応じてハードウェアの性能も向上してきて，バックボーンネットワークの帯域は飛躍的に拡大してきた。現在はインターネット接続事業者間のバックボーンネットワークの帯域が数10Gbps以上になりつつある。しかし，このような長距離（つまり，遅延が大きい）・広帯域のネットワーク，すなわち，帯域と遅延の積（以下，Bandwidth-Delay Product : BDPで記する）が大きなネットワーク環境では，TCPを利用するエンドツーエンド間の通信では，回線の帯域に余裕があるにもかかわらず，スループットが上がらないという問題が顕著になっている［5］。

本稿では，BDPの大きいネットワーク環境において[1]，OPNET Modeler［11］を利用して，TCPのウィンドウサイズ，受信バッファーそして往復遅延がTCPのパフォーマンスに対して与える影響を検証する。更に高速ネットワークにおける新しい技術について概説し，TCP通信の現状とクライアントOSへの実装状況について考察する。本論文の構成は以下のようになっている。

第2節で，BDPの大きいネットワークにおいてTCPの問題点について述べる。第3節では，シミュレータを用いて高速ネットワークにおけるTCPのパフォーマンス，特にウィンドウサイズ，受信バッファー容量などの影響を検証する。第4節で，高速ネットワークにおける技術動向について述べる。第5節でTCPの現実とその実装についてエンドユーザとネットワーク構築の観点から議論する。第6節でまとめる。

2　TCPの輻輳制御と問題

2．1　TCP Renoの輻輳制御

TCPの輻輳制御では，ウィンドウ制御方式のTCP Reno［1, 2］（あるいはNew Reno［13, 14］）が最も普及し，広く実装されている。そのメカニズムは，ウィンドウサイズを用いて送信量を調整することにある。ウィンドウサイズとは送信側が受信側からの確認応答を待たずに，連続して送信可能なデータ量のことである。TCP Renoでは，3つのフェーズに応じて，ウィンドウサイズ（w_c：現在のウィンドウサイズ。単位はパケットである。）を調整する（図1）。まず，スロースタートフェーズという初期段階で，w_cは初期ウィンドウサイズw_Iから開始する。応答確認が戻ってきて，なおかつw_cはある閾値w_{th}以下であれば，w_cを1つ増やす。w_c分のパケットを送信し，正常に応答確認が戻ってきたら，$2w_c$まで増加することになる。第2のフェーズでは，輻輳回避フェーズと呼ばれ，w_cが閾値w_{th}を超えても，パケットロスが起きなかったら，w_cを増加させつ

つも，w_c の増加ペースを減らす。応答確認を受け取るたびに，w_c を $1/w_c$ 増加させる。結果として現在の w_c 分のパケットを送った後に，1 パケット分を増やすことになる。そして，第 3 のフェーズはパケットロス（輻輳発生）フェーズである。パケットロスが発生したら，w_c を半分に減少して，輻輳回避フェーズのウィンドウサイズの調整にしたがって動作する。まとめると以下のアルゴリズムになる。ただし，$w_c(t')$ は次の時刻 t' のときのウィンドウサイズである。

$$w_c(t') = \begin{cases} w_c(t) + 1, & if \quad w_c(t) \leq w_{th}, \quad \text{スロースタートフェーズ} \\ w_c(t) + 1/w_c(t), & if \quad w_c(t) > w_{th}, \quad \text{輻輳回避フェーズ} \\ w_c(t)/2, & if \quad \text{パケットロス発生}, \end{cases} \tag{1}$$

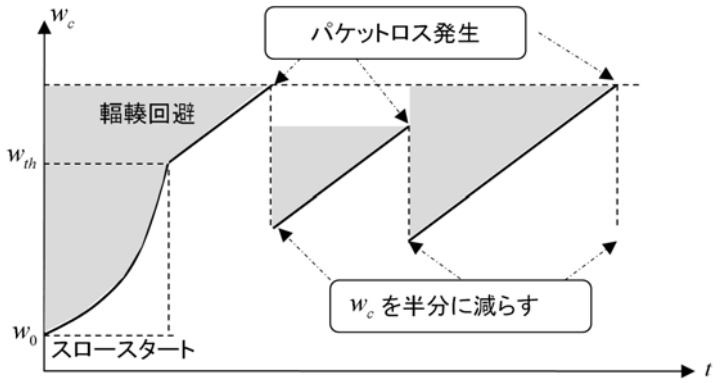

図 1 ：TCP Reno スロースタートと輻輳回避メカニズム

2. 2　高速通信における輻輳制御の問題

TCP において送信側と受信側に対してネットワークの状況を伝える仕組みはないが，ネットワーク内で発生した誤りの制御，輻輳制御（パケットロスの回避）そしてエンドツーエンド間のフロー制御は以下のような簡単な仕組みで行っている。

輻 輳 制 御　ウィンドウ制御アルゴリズム（式 (1)）を用いて，ネットワークの状況に応じて送信量 w_c を増減させ，輻輳を回避する。

フロー制御　受信側の受信バッファーサイズ（R_{win}）を利用して，受信側の処理能力を超えないように，送信側の送信量を調整する。

誤 り 制 御　受信側からの応答確認で，再送要求があれば再送信によって誤り制御を行う。

エンドツーエンド間のフロー制御においては，TCP の応答確認パケットによって受信ウィンドウサイズを送信側に通知することが可能であるが，再送要求があったとき，その再送要求は，ネットワーク輻輳で生じたパケットロスによるものか，パケット誤りによるものか，見分けがつかず，一律に輻輳によるものと見なされ，ウィンドウサイズを減少させる。現在広く利用されている TCP Reno では，いったんパケットロスが発生すると，一気にウィンドウサイズを半分に減

---

[1] エンドツーエンド間の遅延の大小は，必ずしもエンドツーエンド間の地理的な距離の遠近を意味しない。ネットワークのトポロジーによって地理的に近くても通信遅延が大きい場合がある。

少させてから，輻輳回避フェーズにしたがって，ウィンドウサイズをゆっくり増加する（増加幅は $1/w_c$ である）。結果として，TCP の実効スループット（通信速度） ρ の上限は，以下のように制限される。

$$\rho = \min\left\{\text{通信路の帯域}, \frac{w_m}{rtt}\right\}. \tag{2}$$

但し，$w_m = \min(w_c, R_{win})$。R_{win} は受信バッファーサイズである。受信バッファーサイズについては，TCP ヘッダのウィンドウサイズで送信側に伝えるが，16 ビットで表現されているので，伝えられる受信バッファーサイズの最大値は 65,535 バイトである。 1 Gbps のスループットを実現するためには，R_{win} が最大の 65,535 バイト（これは Windows XP のデフォルトサイズである）であっても，エンドツーエンド間の往復遅延は 0.525 ミリ秒以下でなくてはならない。オフィス内或いは建屋内のギガビットイーサネット LAN 上での往復遅延は大体この条件を満たすが，企業の本社・支社間や大学の離れたキャンパス間だと，数ミリ秒以上になることも珍しくない。BDP（通信パスの帯域と遅延の積，以下 P_{bd} であらわす）の小さいネットワーク環境ではあまり問題にならないが，P_{bd} の大きい環境においては，通信路の帯域に余裕があるにもかかわらず，TCP のスループットが頭打ちで上がらない問題が起きている。つまり，長距離・高速ネットワーク環境においては，ネットワークの帯域を使いきれない場合がある（図 1 のグレーの部分がネットワークの帯域が使い切れていない状態を表している）。

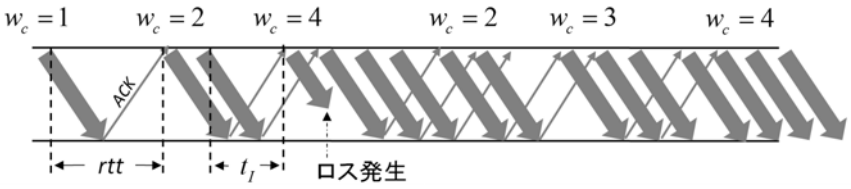

図 2 ：TCP Reno スロースタートと再送（低速の場合）

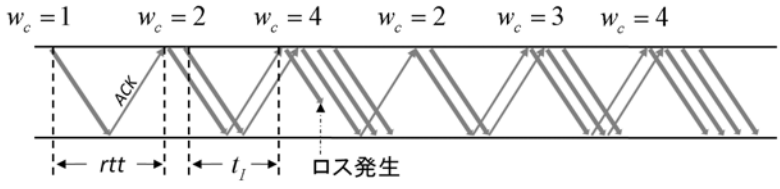

図 3 ：TCP Reno スロースタートと再送（高速の場合）

図 2 と 3 に，往復遅延（*rtt*）が同じであるが，通信速度の違いによる送信の様子の違いを示している。通信路の速度を *S* Mbps，往復遅延を *R* ミリ秒，パケットサイズを *M* バイトとすると，往復遅延中に送出可能なパケット数 *N* は

$$N = \frac{1000SR}{8M} = \frac{125SR}{M} \tag{3}$$

になる。つまり，通信帯域を使い切るためには[2]，TCP ウィンドウサイズ w_c と受信バッファーサイズ R_{win} ともに，*N* 以上でなくてはならない。また，TCP スロースタート（$w_c = w_1 = 1$）から開始して，ウィンドウサイズ w_c が *N* 以上に達するのに要する往復回数を *K* とすると，$K = \log_2^N +$

1 となり，送信したパケットの累積は $2N - 1$ となる。たとえば，文献［5］の例で，通信帯域が 10Gbps で，往復遅延が 100 ミリ秒で[3]，パケットサイズ（M）が 1,500 バイトである場合，N は約 83,333 パケットとなる。一本の TCP コネクションで 10Gpbs の帯域を使い切るためには，パケットのロス率（パケット誤りによる廃棄か，輻輳によるパケットロス）が 2×10^{-10} 以下である必要がある。これは現在の光ファイバーのネットワークにおいては，非現実的である。

3　シミュレーションによる検証

本節では，ネットワーク・シミュレータ OPNET Modeler［11］を用いて，現在最も普及している TCP Reno の特性を調べる。特に TCP の初期ウィンドウサイズ w_I，輻輳ウィンドウサイズと受信バッファーサイズと遅延の影響を検証する。

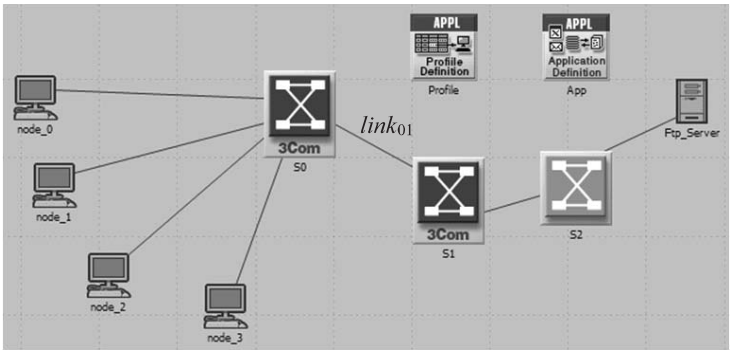

図 4 ：ネットワークトポロジー

ネットワークのトポロジーは図 4 に示す。s_0，s_1，s_2 は，1000BASE-T と 10GBASE-T のイーサネットインターフェースを持つスイッチである。s_0，s_1 間のリンク以外は全て 10Gbps のリンクで，長さは 50m とする（リンク上の遅延は無視できるほど微小である）。*Ftp_Server* は，FTP サーバである。*Node*_0 から *Node*_3 は通常の端末で，10Gbps のリンクによりスイッチ s_0 に接続している。

3．1　遅延の影響

シミュレーションのシナリオは，*Node*_0 から *Node*_3 において，*Ftp_Server* から 4.38GB のファイル（DVD メディア一枚のイメージファイルに相当する）をダウンロードすることを想定し，TCP のスループットとノード上のウィンドウサイズなどを調べる。$s_0 \leftrightarrow s_1$ のリンク（$link_{01}$）の遅延を d_{01} として，$link_{01}$ のスループットを図 5 （右）に示す。各ノードの TCP の詳細設定は図 5 （左）に示すが，主要なパラメータとしては受信バッファーサイズは 8,760 バイトで，スロースタート・イニシャル・カウントは 2 MSS で，ファーストリカバリーは Reno で，ファースト再送を有効に

2　単純化のため，ここではエンドツーエンド間に，TCP のコネクションが 1 本しかない場合に限定して考える。
3　遅延時間はネットワークの構成トポロジー，中間機器の処理にも依存するので，必ずしも地理的な距離と一致しない場合がある。たとえば広島市内のユーザ宅からブロードバンド接続を経由して，同じ市内にある本学のネットワークへアクセスする場合，往復遅延は 40 ミリ秒以上になることもある。

している。図から d_{01} はそれぞれ，0.1ms（図の *b*），1 ms（図の *a*）と 10ms（図の *c*）であるとき（各ノードと *Ftp_Server* 間の往復遅延は d_{01} の約 2 倍である），$link_{01}$ のスループットは，それぞれ，992Mbps，144Mbps，25Mbps であった。遅延時間が長くなると，回線の通信帯域を使い切れないことがわかった。

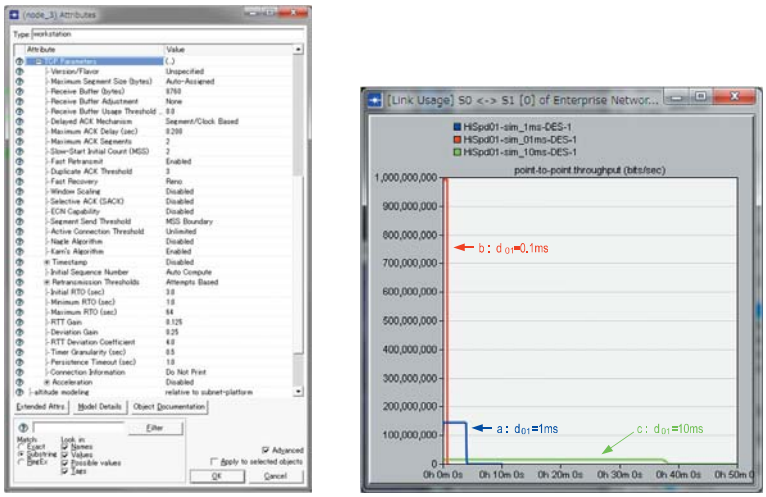

図 5 ：各ノードの TCP パラメータ（左）とリンク $link_{01}$ のスループット（右）

3．2　受信バッファーサイズの影響

図 6 に *Node_0* だけ，単一の TCP コネクション（FTP ダウンロード）の場合において，受信バッファーサイズ（R_{win}）によるスループットの変化を示す。TCP の設定は，$w_1 = 2$，*Fast Retransmit* ＝ *Enabled*，*Fast Recovery* ＝ *Reno*，*Window Scaling* ＝ *Disabled* とした。同図から $d_{01} = 1$ ms（エンドツーエンド間の *rtt* は約 2 ms である）の場合，受信ウィンドウサイズを TCP 標準ヘッダのウィンドウサイズの限界まで（65,535 バイト）利用しても，スループットは 263.65Mbps 程度である。また，R_{win} を 13,1070 バイト（65,535 バイト以上）に設定したとしても，TCP の *Window Scaling* オプション［6］が有効にならない限り，65,535 バイト以上の受信バッファーウサイズの利用ができないので，効果がないことがわかった。

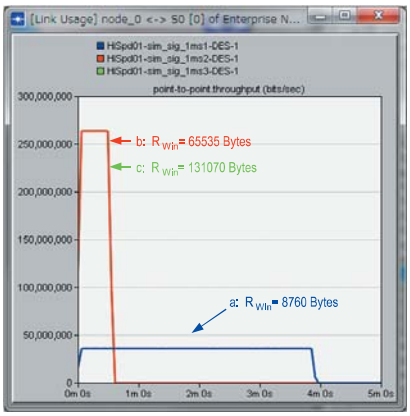

図 6 ：受信バッファーサイズによるスループットの変化（$d_{01} = 1\ ms$）

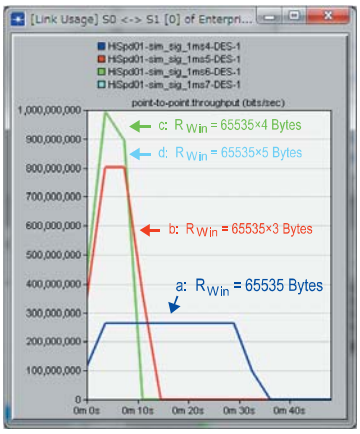

図 7 ：*Window Scaling* を有効にした場合のスループットの変化（d_{01} ＝ 1 *ms*）

続いて，*Window Scaling* を有効にした場合について検証する。*Window Scaling* は RFC 1323［6］で定義され，TCP 標準ヘッダの受信ウィンドウサイズ（16 ビット）を超えたウィンドウサイズを使用するときの TCP のオプションである。図 7 に *Window Scaling* を有効にして，ウィンドウサイズ R_{win} の変化によるスループットを示している。同図から，R_{win} が 65,535 バイトであるとき，TCP スループットは最大 263.65Mbps であった。一方 R_{win} を 65,535 × 3 バイト（図の *b*），65,535 × 4 バイト（図の *c*）に増加させると，スループットは，それぞれ最大 802.63Mbps（回線の帯域がまだ使いきれていない状態）と 992.20Mbps（回線の帯域上限まで使い切れている状態）であった。更に R_{win} ＝ 65,535 × 5 のとき（図の *d*），スループットは図の *c* と同じであった。TCP の輻輳ウィンドウサイズの変化については，図 8 （左）に示す。*Window Scaling* が有効になっている場合，輻輳が発生しない限り，輻輳制御ウィンドウが大きく上昇していることがわかる。同図から受信バッファーサイズの違いによって，輻輳ウィンドウサイズ増加の傾きが違うことも確認できた。

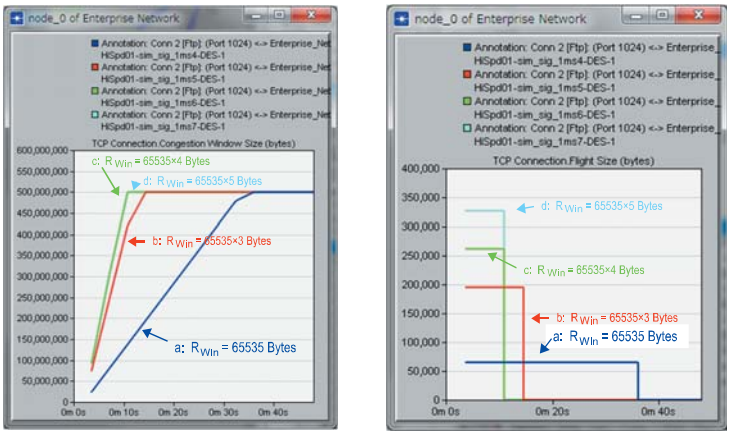

図 8 ：輻輳ウィンドウとフライト・ウィンドウ・サイズの変化（d_{01} ＝ 1 *ms*）

同じ条件の下，TCP コネクションのフライト・サイズの変化を図 8 （右）に示す。フライト・

サイズとはTCPの送信側から送出したデータのうち，受信側から応答確認を受けていないデータ量のことである。図8の輻輳ウィンドウサイズとフライトサイズから，輻輳ウィンドウサイズがパケットロスが発生していない限り，増えていく一方であるが，フライト・サイズはちょうど受信バッファーサイズになっていることがわかる。つまりネットワークの輻輳制御に輻輳ウィンドウが用いられることに対して，エンドツーエンド間のフロー制御には受信バッファーサイズが用いられる。スループットは輻輳ウィンドウサイズと受信バッファーサイズの小さい方に制限されていることも確認できた（式(2)）。

4 高速通信における関連技術

現在，最も普及しているTCP Renoは，パケットロスをベースにした輻輳制御と受信バッファーサイズをベースにしたフロー制御でうまく機能しているといえようが，長距離・高速ネットワーク環境においては，TCPの実効スループットが回線の帯域に見合わない場合がある。本節では，この問題を解決するための関連研究について述べる。

4．1 TCPの拡張

TCPのオプションを利用して，パフォーマンスを改善するための提案がいくつかあるが，RFC 1323［6］は広く実装されるようになった。このRFCでは，(1)受信ウィンドウサイズを拡張して，最大1 GiB（2^{30}バイト）まで，利用可能となった。(2)タイムスタンプを利用してパケットの再送を早めに行うことが可能となった。　選択的応答確認（SACK）を利用する［6, 9］。SACKは，TCPの誤りやロスが発生した時点からのすてべの再送（いわゆるGo-Back-N方式）に対して，部分的に誤りやロスのパケットだけを再送する方式である。ただし，SACKは受信側での順序制御が必要でその処理を行うためのリソースが求められる。

その他，TCPのスロースタートのイニシャルウィンドウw_Iを大きくすることでTCPの性能改善を図る提案もある［3, 4, 16］が，大きな初期ウィンドウは場合によっては輻輳を引き起こすことがあるから，提案のRFC 2414は実験的カテゴリに分類されている。更にTCP Renoの輻輳回避フェーズのウィンドウサイズの増加幅を制御する“HighSpeed TCP”［5］（RFC 3649, Category : Experimental），“Scalable TCP”［10］も提案されているが，これらの方式はスループットを重視する一方，ネットワークの状況によってはTCP Renoよりも輻輳を引き起こす恐れがあるから，更なる評価が必要であろう。

4．2 遅延ベースの方式

輻輳が発生してから（パケットロスベース）ウィンドウサイズを調整するTCP Renoに対して，パケットの往復遅延（*RTT*）を監視し，その変化に応じてネットワークの輻輳具合を予測して，輻輳ウィンドウサイズを調整する方式もある［7, 15］（いわゆる，遅延ベースの方式である）。遅延ベースの方式ではTCP Vegas［7, 8］がよく知られている。TCP Vegasの輻輳ウィンドウ調整アルゴリズムは以下のアルゴリズムになる。

$$w_c(t') = \begin{cases} w_c(t) + 1, & if \quad Diff \times baseRTT \leq \alpha\,, \\ w_c(t), & if \quad \alpha < Diff \times baseRTT \leq \beta\,, \\ w_c(t) - 1, & if \quad Diff \times baseRTT > \beta\,, \end{cases} \tag{4}$$

但し，α，β は，定数であり，*baseRTT* は今まで測定された最小の *RTT* である。*Diff* は往復遅延の変化である。また，TCP Vegas を改良した FAST TCP［12］も提案されている。

4．3　TCP 以外の方式

TCP では，フロー制御，輻輳制御および誤り制御は全てエンド端末で行うことで，ネットワーク中間ノード上の負荷を軽減しているが，通信のエンドはネットワークの状態を正確に把握できない。それに対して，ネットワーク上のルータに拡張機能を組み込み，ネットワークの状態を反映した通信を行う方式が提案されている（XCP［17］や ECN［18］）。XCP は TCP と同じ位置付けでトランスポート層のプロトコルとして提案されているが，ECN は TCP のヘッダを拡張して利用することで TCP との互換性を保つように設計されている。ネットワーク上のルータに ECN を実装していなければ，ECN のための拡張部が無視され，従来の TCP で動作すればよいという原理である。これらの方式は，ネットワーク上のルータがこの機能をサポートすれば広帯域のネットワーク環境でも，コネクション間の公正性を保ちながら，高いスループットが得られるが，現在普及している TCP Reno からの移行は時間がかかるだろう。

5　TCP の現実

前述のように，広帯域ネットワークに対応するため様々な方式が提案されているが，ネットワーク機器への実装は容易ではない。その理由は経路制御やパケット転送のほかに，ネットワークトラフィックの状況を把握しながら，各コネクションに対応することは，ネットワーク機器に大きな負荷を与えるからである。本節では Jumbo フレームへの拡張やクライアント OS での実装について考察してみる。

5．1　Jumbo フレーム

TCP/IP ネットワークにおいて，プロトコルのオーバーヘッド（パケットの配送に必要な情報部分）があるため，データ転送のスループットは回線の速度よりも低下する。例えば，イーサネット LAN ベースの TCP/IP ネットワークでは，MTU（Maximum Transfer Unit）が 1,500 バイトであるが，それぞれ 20 バイトの TCP と IP のヘッダを除くと，データサイズ（MSS : Maximum Segment Size）は 1,460 バイトとなる。さらにイーサネットのフレームヘッダとプリアンブルなど（あわせて 26 バイト）を考慮すると，データ伝送のスループットは約 95.6％である。

近年ネットワーク機器の性能および回線の品質が向上し，ギガビットの LAN では 1,500 バイトの MTU では小さすぎて通信の効率が悪いと指摘されている。そこで，MTU を大きくして（いわゆる Jumbo フレーム），データ部分の効率を高める傾向がある。最近のネットワーク機器やパソコンの LAN アダプターでも Jumbo フレームをサポートしている製品が多く手軽に利用可能となってきた。Jumbo フレームのサイズは機器によってまちまちであるが，概ね 8,000 ～ 15,000 バイト程度である。Jumbo フレームの効果につては，MTU を 9,000 バイトとした場合，データ転送の効率は理論的に 99.3％まであがる。3.7％しか向上していないように見えるが，TCP のスロースタートとウィンドウ制御のメカニズムにおいて，Jumbo フレームを 9,000 バイトとして，LAN 上のサーバからファイルダウンロードを行った際の実測値で，TCP の平均スループットは約 30％向上した結果となった。

Jumbo フレームの効果は十分期待できる一方，その効果を発揮するためにはエンド通信間の全ての機器が Jumbo フレームをサポートする必要がある。さもなければ，パケット分割が大量に発生し，逆効果につながる。また，Jumbo フレーム未対応の機器で，Jumbo フレームを取りこぼしたりすると，再送信が大量に発生し，効率が極端に悪くなってしまうこともある。更にネットワークの構築・運用の立場からみると，Jumbo フレームはネットワークのトラフィックに大きな揺らぎをもたらし，瞬発的なトラフィックが大きくなる。ネットワーク帯域をより高く求められる。しかし，逆に言えばネットワークリソースが潤沢であれば，Jumbo フレームを適切に運用することで，ネットワークサービス（例えば，NAS などのファイルサービス）がより快適なものになる。

5．2 クライアント OS への実装状況

新しい方式の実装は，既存の方式やその他の方式との互換性，異なる方式が混在した時に，片方の方式に悪影響を与えるような相性の悪いケースがあるといったことを配慮しなくてはならない。実装状況はクライアント OS によって対応が違うが，既に Standards Track として RFC で承認された方式については，各クライアント OS に徐々に取り組まれていくが，オープンソースの Linux 系 OS への取り込みが積極的だといえよう（提案方式は，殆ど Linux kernel 上で実装と評価を行っているため）。例えば TCP Vegas，ECN，SACK など標準化された RFC は Linux Kernel に取り組まれている。また，HighSpeed TCP や Scalable TCP など，Linux kernel への実装も公開されている［19］［20］。

Windows 系の OS については，圧倒的なシェアを占めている[4]こともあって，Windows 系 OS の各方式への対応状況は大きな影響を与える。Windows Vista 以降（Windows Server 2008，Windows 7 を含む）は，Compound TCP（CTCP）と呼ばれる高速 TCP を実装している。CTCP はパケットロスベースの方式（TCP Reno，HighSpeed TCP）とパケット遅延ベース（TCP Vegas）の方式を組み合わせて設計された方式である。その考え方は，TCP の効率性に加え，他の TCP コネクションとの相性を配慮した通信の実現を目標としている。通信パスの帯域に余裕があれば，素早くウィンドウサイズを増やし，混雑してきたら緩やかに減少させ，輻輳が発生したら，TCP Reno のようにウィンドウサイズを減少させることで実現してる。

Windows Server 2003 または Windows XP 64 ビットの OS の場合には，ホットフィックス［21］を適用してレジストリキーを修正すれば，CTCP の利用が可能であるが，既に発売終了の Windows XP 32 ビット版への CTCP 対応はないようである（ソフトウェア特許の関係で Linux Kernel への実装は困難のようである）。従って，Windows XP 32 ビット版においては，CTCP の利用ができないが，TCP のパラメータにかかわるレジストリキー（Tcp1323Opts，TcpWindowSize および SackOpts など）［22］を修正することでパフォーマンスの改善が期待できる。

6 まとめ

本稿では，広帯域ネットワーク環境を取り巻く TCP の輻輳制御と効率性について考察を行っ

[4] 現時点 Windows 系 OS のシェアは約 91.12％である。その内訳は Windows XP は 59.07％で，Windows 7 は 18.24％で，Windows Vista は 12.88％である。

た。最も普及している TCP Reno についてはネットワークシミュレータを利用して TCP のスループットとエンド間の往復遅延，受信バッファーサイズの関係を検証した。長距離・広帯域の環境で受信ウィンドウサイズの小さいクライアントにおいて，受信ウィンドウサイズが TCP スループットの足かせになりかねないことがわかった。更に新しい技術の現状とクライアント OS への実装状況について考察した。

TCP の効率性と輻輳制御において，様々な新技術が提案されている中，既存方式との互換性，そして TCP コネクション間の公平性などまだ問題が多いが，本研究は今後ネットワークの運用・管理や TCP のパフォーマンスチューニングに寄与するものと考えられる。今後もネットワーク機器メーカーやクライアント OS の対応に注視して取り組んでいくべきであろう。

謝　辞

本研究の一部は，平成 22 年度科学研究補助金（基盤研究（C）：22500066）の研究助成を受けて行なったものである。

参考文献

［1］V. Jacobson, “Congestion Avoidance and Control,” SIGCOMM ’88, Aug. 1988.
［2］W.Stevens, “TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms,” RFC 2001, Jan. 1997.
［3］M. Allman and S. Floyd, “Increasing TCP’s Initial Window,” RFC 2414, Sept. 1998.
［4］M. Allman, V. Paxson and W. Stevens, “TCP Congestion Control,” RFC 2581, April 1999.
［5］S. Floyd, “HighSpeed TCP for Large Congestion Windows,” RFC 3649, Dec. 2003.
［6］V. Jacobson, R. Braden and D. Borman: “TCP Extensions for High Performance,” RFC 1323, May 1992.
［7］L. Brakmo, S. O’Malley, and L. Peterson: “TCP Vegas: New techniques for congestion detection and avoidance,” In Proceedings of the SIGCOMM ’94 Symposium, pp.24-35, Aug. 1994.
［8］L. Brakmo and L. Peterson: “TCP Vegas: End to End Congestion Avoidance on a Global Internet,” IEEE Journal on Selected Areas in Communications, Vol.13, No.8 pp.1465-1480, Oct. 1995.
［9］M. Mathis, J. Mahdavi and S. Floyd: “TCP Selective Acknowledgment Options,” RFC 2018, Oct. 1996.
［10］T. Kelly: “Scalable TCP: Improving Performance in Highspeed Wide Area Networks,” Computer Communication Review 32　, April 2003.
［11］OPNET Technologies, Modeler: http://www.opnet.com/solutions/network rd/modeler.html
［12］C. Jin, D. Wei, D. H. Choe : “FAST TCP: From Theory to Experiments,” IEEE Network, Vol.19, Issue:1 Feb. 2005.
［13］S. Floyd, T. Henderson, and A. Gurtov. The NewReno Modification to TCP’s Fast Recovery Algorithm. RFC 3782, April 2004.
［14］N. Parvez, A. Mahanti, and C. Williamson, “TCP NewReno: Slow-but-Steady or Impatient?” Proceedings of the 2006 IEEE International Conference on Communications (ICC), Istanbul,

Turkey, June 2006.
［15］竹田稔，竹中豊文 : “広帯域に対応した RTT ベース TCP 輻輳制御方式, ” 信学技報，IEICE Techincal Report NS2005-158, Mar. 2006.
［16］T. Kitagawa and C.-X. Chen, “Initial Window Size of TCP in Broadband Network Environment”, Proc. of the IEICE ’06 General Conference,B-16-16, p.625, Mar. 2007.
［17］S. Low, L. Andreq, and B. Wydrowski: “Understanding XCP: Equilibrium and Fairness,” Proc. IEEE INFOCOM ’05, Vol.2, pp.1025-1036, Mar. 2005.
［18］K. Ramakrishnan, S. Floyd, and D. Rosen, “The Addition of Explicit Congestion Notification (ECN) to IP,” RFC 3168, Sept. 2001.
［19］“HighSpeed TCP Implementation,” http://www.icir.org/.oyd/hstcp.html
［20］“A Scalable TCP Implementation for Linux,” http://www.deneholme.net/tom/scalable/
［21］Microsoft 複合 TCP，http://support.microsoft.com/kb/949316
［22］http://support.microsoft.com/kb/418053/ja