情報処理学会論文誌 Vol. 52 No. 3 1308–1322 (Mar. 2011)

輻輳制御パラメータのリアルタイム推定

茂木重憲[†1,*1] 渡邊晶[†2]

TCP は輻輳制御アルゴリズムによって，輻輳制御パラメータである cwnd や ssthresh を設定し，1 度に送出できるデータ量や再送動作などを制御している．これらの値が分かれば，ネットワーク管理者が行う障害対応が容易になることなどが期待できるが，これらの値は TCP 実装内でのみ更新され，伝送路上に送出されない．そこで輻輳制御アルゴリズムに基づいて通信のやりとりを解析し，cwnd と ssthresh をリアルタイムに推定する方法を検討した．評価実験の結果，データパケットの損失がなければ，ほぼ正確な値を推定でき，ネットワーク管理者に有用な情報を提供できることを確認した．データパケットが損失する場合でも 80%以上の推定値が，設定した許容誤差である ±1 MSS 以内に収まっていることが確認できた．

Real-time Estimation of TCP Congestion Control Parameters

Shigenori Mogi[†1,*1] and Akira Watanabe[†2]

In this paper, we describe the method to estimate congestion control parameters, cwnd and ssthresh, in real-time from captured packets to help network administrators troubleshoot network problems. From the result of the evaluation study, if no data packets is lost, the most estimations matche values in the kernel. We think that our estimation method can offer the valuable information for network administrator. When data packets are lost, about 70% of the estimation of ssthresh match the value in the kernel, while the estimation of cwnd have an error from −1 MSS to +1 MSS.

†1 明星大学大学院情報学研究科情報学専攻
Department of Information Science, Graduate School of Information Science, Meisei University
†2 明星大学情報学部情報学科
Department of Information Science, Meisei University
*1 現在，株式会社 ACCESS
Presently with ACCESS

1. はじめに

TCP は輻輳制御アルゴリズムを用いて，なるべく輻輳を発生させないように通信量を制御している．輻輳制御アルゴリズムで通信を制御するために 1 度に送出できるデータ量を示す輻輳ウインドウ（cwnd）やその閾値（ssthresh）を設定する．TCP はコネクションを確立する際に cwnd を 1 パケットで送信できるデータ量の最大値を示す Maximum Segment Size（MSS，通常は 1,448 バイト）1 つ分に設定し，ssthresh を 65,535 バイト（ウインドウスケールオプションなしの場合）に設定する．cwnd <= ssthresh の場合，送出したパケットに対する ACK を受信すると，cwnd を 1 MSS 分増加させ，連続して送出できるパケットの量を 1 MSS 分増加させる．このような動作をスロースタート[1)] と呼ぶ．cwnd > ssthresh の場合，送出したパケットに対する ACK を受信した際に cwnd を 1 MSS 分増加させると，輻輳が発生する確率が高くなるとし，cwnd の増加量を 1 MSS 以下に抑える．このような動作を輻輳回避アルゴリズムと呼ぶ．

パケットが損失すると，パケットを送出してから ACK が返ってくるまでの時間である Round Trip Time（RTT）が，タイムアウトが発生したと判断する時間である Retransmission Time Out（RTO）を超えてしまい，タイムアウトが発生する．タイムアウトが発生した場合，1 度に送出できるデータ量を下げ，パケットの損失を回避する動作を行う．TCP は cwnd を 1 MSS，ssthresh をタイムアウト発生時の cwnd の半分または，ウインドウサイズの半分に制限してから，再度スロースタートを行って，1 度に送出できるデータ量を回復していく．

cwnd と ssthresh の値が分かれば，送出可能なデータ量や，再送が発生した原因が分かる．たとえば，送信すべきデータがあるにもかかわらず，パケットが連続して送出される量がきわめて少ないような障害に遭遇した場合，cwnd の値を見ることによって，タイムアウトまたは，Duplicate ACK による輻輳の発生が原因で cwnd の値が減少し，連続して送出できるパケット量が制限されていることが把握できるなど，ネットワークのトラブルシューティングが容易になる可能性がある．また，実トラフィックを観察する TCP の学習者にも有用であろう．しかし，cwnd と ssthresh の値は TCP 実装内で更新されるのみで，ネットワーク上に送出されないため，輻輳制御アルゴリズムの動作を推定するには TCP 実装の cwnd や ssthresh を推定する必要があるが，TCP に実装されている輻輳制御アルゴリズムに基づいて 1 パケットごとに cwnd や ssthresh を計算する必要があり，大量のパケットが送出されている通信を手作業で解析するには限界がある．

cwnd や ssthresh を観察するための方法として，NS-2[2] というシミュレータが存在する．NS-2 はトラフィックパターンをシミュレートしたり，任意のプロトコルを実装したりすることができる．cwnd や ssthresh を表示できるが，シミュレータであるため，実際の通信とは異なる．TCP の動作解析では，様々な研究[3)–6)] が行われているが，cwnd や ssthresh の値を推定してるものはない．Paxson は TCP 通信の挙動を観察し，どの TCP のバージョンを参照して TCP が実装されているかを検査するためのツールの研究[7)] の中で cwnd と ssthresh を推定しているが，保存された通信トレースからの推定しかできず，リアルタイム推定ができない．通信に問題が発生しているときに推定値が即座に提示されれば，現象の解析に利用する際に，障害を解決するまでの時間が短縮される可能性があるなど，ネットワークから受信したパケットに対して，リアルタイムに輻輳制御パラメータを自動的に推定できることは有用であると考え，TCP 実装の輻輳制御アルゴリズムに基づいて通信のやりとりを解析し，cwnd と sshtresh の値を推定するための手法を検討した．

我々はリアルタイムでの推定の前に tcpdump[8)] で保存したトレースのデータを読み込み，cwnd と ssthresh の値を推定するツールを作成し，コネクションのエンドポイントで保存したトレースデータに対しては推定が正しく行えることを報告した[9)]．また，リアルタイム推定のための基礎的検討と簡易的な評価結果を報告した[10)]．本稿では輻輳制御パラメータの推定手法の概要を述べ，リアルタイム推定における問題点とその解決方法を示す．さらに，リアルタイム推定機能を組み込んだ tcpdump を用いた評価実験結果を示す．

2. 輻輳制御パラメータの推定方法

輻輳制御アルゴリズムは様々なバージョンが存在し，アルゴリズムごとに cwnd と ssthresh の計算方法が異なる．本研究では FreeBSD 8.0 に実装されている NewReno アルゴリズム[11)]，オプションとして SACK オプション[12)]，タイムスタンプオプション，ウインドウスケールオプション，On Estimating End-to-End Network Path Properties[13)] を対象とした．FreeBSD のバージョン 8.0 に採用されている輻輳制御アルゴリズムで再送のトリガとなる原因はタイムアウトと Duplicate ACK である．以下順に検討した推定方法の概略を示す．

2.1 ACK パケットの処理

TCP は ACK パケットを受信すると TCP Header Prediction の処理を行う．TCP Header Prediction とは，データを一方的に送信している場合に受信することが多い ACK パケットの処理時間を減らす処理[14)] のことで，以下の 4 つの条件を満たした場合，そのセグメントは純粋な ACK であるとし，送出すべきデータがある場合には，輻輳制御パラメータを変更せずに TCP の送出関数を呼び，パケットを送出する．

(1)　セグメントがデータを含んでいない
(2)　受信 ACK 番号が，送信済だが未確認の最小シーケンス番号（以下 snd_una）よりも大きい．
(3)　受信 ACK 番号が，送信した最大のシーケンス番号（以下 snd_max）よりも小さい．
(4)　cwnd が現在の送信ウインドウよりも大きい．

本研究では実装と同様，上記 4 つの条件を満たした場合には cwnd を増加させない．上記 4 つの条件を満たさない場合には通常どおり cwnd を ACK の個数分増加させる．

2.2 Duplicate ACK

Duplicate ACK をトリガとして再送するアルゴリズムは Reno[1)] で実装され，Fast Retransmission として知られている．Fast Retransmission は Duplicate ACK を 3 回以上受信したときに対象のパケットを再送することで，不要な再送の発生を最小限にとどめる．本研究では FreeBSD 8.0 の TCP 実装同様，RFC2582[15)] を参照し，Duplicate ACK を 3 回以上受信したとき，ssthresh を cwnd とウィンドウサイズの小さい方の半分の値，cwnd を $ssthresh + duplicateack$ 数 $* MSS$ に設定する．ただし，ssthresh は最低でも 2 MSS である．以降，snd_max 以上の ACK を受信した場合は，cwnd を ssthresh に設定して再送を終了する．それ以外の ACK を受信した場合には，Partial ACK[11)] として次の処理を行う．cwnd の値を ocwnd という変数に代入する．cwnd の値が，Partial ACK の値から snd_una の値を引いた値より大きい場合，Partial ACK の値から snd_una の値を引いた値を cwnd から引く．それ以外の場合，cwnd は 0 となる．最後に，cwnd に 1 MSS を足した値が現在の cwnd となる．Partial ACK による再送が終了すると，ocwnd の値が Partial ACK の値から snd_max の値を引いた値より大きい場合，ocwnd の値から Partial ACK の値を引き，今まで ACK を受信した中で最大の ACK 番号の値を足す．それ以外の場合は 0 になる．そして，ocwnd に 1 MSS を足し，現在の cwnd に設定する．

2.3 タイムアウト

送出セグメントに対する ACK の到着が RTO を超えた場合，タイムアウトとなり対象のパケットが再送される．RTO は送出セグメントに対する ACK の到着時間 RTT を基準に算出される．通信の中間地点で RTT を推定する場合には，文献 16) や文献 17) で述べられているように誤差が発生するため，RTO にも誤差が発生してしまう．本研究では，Duplicate ACK を 3 回以上受信していない場合にタイムアウトが発生したとする．しかし 3.2 節で後述するように，Duplicate ACK による再送と判定しても，実際にはタイムアウトによる再

送が発生している場合があるため，Duplicate ACK とタイムアウトによる再送の 2 つの推定値を算出，保持し，表示する段階で再送原因の誤判定を修正できるようにした．タイムアウト発生時の cwnd と ssthresh は，FreeBSD8.0 の実装に基づき以下のように推定を行う．

(1)　cwnd を 1 MSS，ssthresh を cwnd とウィンドウサイズの小さい方の半分に設定する．ただし ssthresh は最低でも 2 MSS である．

(2)　snd_max より大きな ACK を受信するまで，snd_max までのパケットの再送を続ける．

2.4　SACK

SACK が有効になっているかどうかは，コネクション確立時に SACK Permitted オプションが交換されているかどうかで判断できる．SACK ブロックが格納された ACK パケットを受信した場合，ACK フィールドの値は使用せず，SACK ブロックに格納されている値を使用して有効な ACK の数を数える．

2.5　On Estimating End-to-End Network Path Properties

On Estimating End-to-End Network Path Properties [13] は再送が 1 回で終了した場合，輻輳は軽微であると考え，直前の cwnd に戻す輻輳制御アルゴリズムである．FreeBSD8.0 では標準で有効となっている．検討した推定方法では，再送が発生した場合，cwnd を算出する前に直前の cwnd の値をバックアップしておき，次に送出されるパケットが通常のパケットであれば，cwnd をバックアップしておいた $cwnd + MSS * ACK$ 数に設定するよう実装した．

3.　リアルタイム解析の問題点と解決方法

Paxson の研究の中で，パケットをモニタする場所によって，検査機能が正しく動作しないという問題点が指摘されている．本研究の cwnd と ssthresh の推定においても同様の問題が発生する．本章では，この問題への対応方法と限界について述べたのち，リアルタイムでの推定方法を述べる．さらに，リアルタイム推定機能を tcpdump に追加する際の実装方法を述べる．

3.1　パケットをモニタする場所の問題

通信をキャプチャする場所の問題の例を**図 1** に示す．ホスト A とホスト B の通信をその間にある観測点で取得した場合，観測点とホスト B の間で損失したパケットは観測点のトレースには残っているがホスト B には届いていないため，観測点の通信トレースとホスト B で受信したパケットが一致しなくなってしまう．このような問題を回避するため，シーケンス番号の検査を行う．

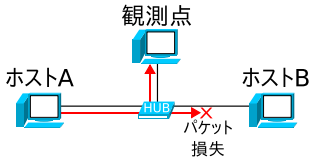

図 1　通信をモニタする場所の問題
Fig. 1　Problem of the location of monitoring packets.

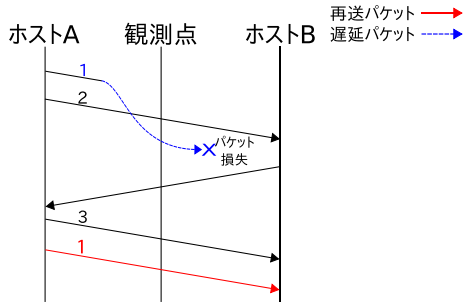

図 2　遅延パケットの問題
Fig. 2　Packet delay problem.

(1)　直前のパケットのシーケンス番号と同じシーケンス番号のパケットをキャプチャした場合に (2) の処理を行う．

(2)　遅延の問題もあるため，直前の同じシーケンス番号のパケットのタイムスタンプと現在のパケットのタイムスタンプを比較する．

- 最初にモニタしたパケットのタイムスタンプの方が古い場合
  直前のパケットは送信先には到着していないと分かるので，直前の推定値を削除し，現在のパケットで新たに推定した推定値に置き換える．
- 最初にモニタしたパケットのタイムスタンプの方が新しい場合
  現在のパケットは遅延パケットだと分かるので，推定には使用せず，無視する．

しかし，これは遅延パケットと再送パケットの間に新にパケットが送出されないことが条件となる．遅延パケットと再送パケットの間に新にパケットが送出されている場合の例を**図 2** に示す．ホスト A から送出されたパケット 1 が遅延し，観測点ではパケット 2 の後に到着する．このとき，観測点の受信した最大シーケンス番号はパケット 2 のシーケンス番号

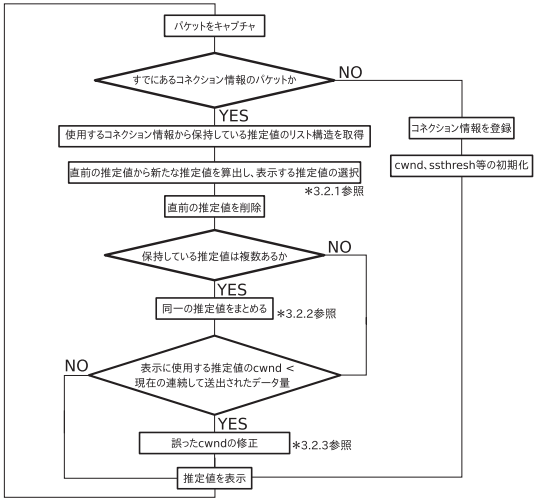

図 3　リアルタイム推定機能の処理の流れ
Fig. 3　Flow chart of real-time estimation.

なので，パケット 1 は再送されたパケットだと検知されてしまう．しかし，パケット 1 がホスト B と観測点間で損失すると B には到着していないので，パケット 3 を送信した後にホスト A でパケット 1 に対するタイムアウトが発生し，パケット 1 が再送される．このとき，すでに観測点ではパケット 1 が再送されたパケットだと誤検知している．過去のパケットの推定値をすべて保持しておけば，推定値の誤りを修正できるが，メモリ使用量が膨大になることや，計算時間が多くなり，リアルタイム推定でのレスポンスが悪くなるため，本研究ではこの問題への対応を行わないこととした．

3.2　リアルタイムでの推定手法

リアルタイム推定機能の処理の概略を**図 3** に示す．パケットをキャプチャすると，送信先 IP アドレス，ポート番号と送信元 IP アドレス，ポート番号の組合せを，現在保持しているコネクション情報から検索する．コネクション情報が存在しない場合，新しいコネクションとして登録する．コネクション情報が存在する場合には，保持している推定値から新たな推定値を算出する．このとき，Duplicate ACK が発生しているように見えても，実際はタイムアウトが発生している場合などがあり，以降の推定に誤差が生じてしまう．この問題に対応するため，再送発生時にタイムアウトと Duplicate ACK による再送の 2 つの推定値を計算し，リスト構造として保持する．したがって，再送発生時の誤判定によって表示する推定値が間違っていても，後に修正できるようになる．リスト構造の例を**図 4** に示す．図 4 (a)，(b) は上から下へ，点線の箇所でパケットを受信した際の推定値の保持の例を示している．矢印の元の部分は直前の推定値，矢印の先が新たに計算した現在の推定値を示している．図 4 (a) の場合は，2 番目のパケットが再送パケットなので，タイムアウトと Duplicate ACK による再送の 2 つの推定値を算出し，保持する推定値は 2 つとなる．3 番目のパケットは通常パケットで，保持する推定値の数は変わらない．図 4 (b) の場合は，2 番目のパケットまでは図 4 (a) と同じく 2 つの推定値を保持しており，3 番目のパケットで再送パケットを受信し，タイムアウトと Duplicate ACK による再送の 2 つの推定値を新たに算出し，最終的に保持する推定値は 4 つとなる．現在の推定値が算出された後に直前の値を削除する．図 3 の 2 番目の条件判定である，「保持している推定値は複数あるか」以下，および推定値の算出方法については，次項以降で詳細に説明する．

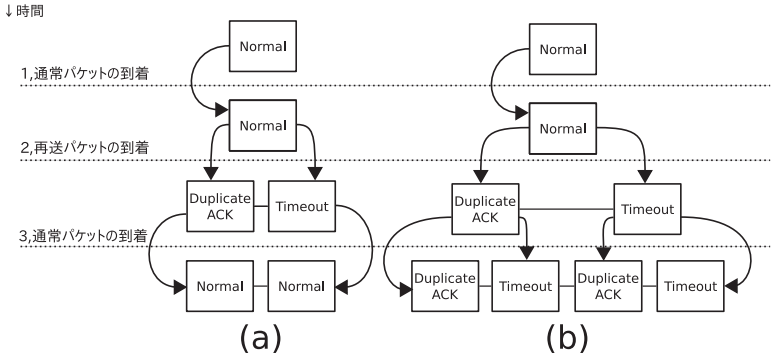

図 4　リスト構造の例
Fig. 4　Data structure for the real-time estimation.

3.2.1　複数の推定値を保持している場合の推定値の算出と表示

算出した推定値が複数あった場合，表示に使用する推定値を 1 つ選ぶ必要がある．通常パケットの場合，直前に表示に使用した推定値から算出した推定値を表示に使用する．再送パケットの場合，基本的には Duplicate ACK を 3 つ以上受信していた場合には Duplicate ACK，それ以外はタイムアウトとして算出した推定値を表示する．より詳細な手順は以下のようになる．

if(直前のパケットが通常パケットだった場合)
　　/* 推定値の算出 */
　　タイムアウトによる再送*1 と Duplicate ACK による再送の推定値を算出.

　　/* 推定値の表示 */
　　if(Duplicate ACK を 3 個以上受信した場合)
　　　　Duplicate ACK による再送として算出した推定値を表示.
　　else
　　　　タイムアウトによる再送として算出した推定値を表示.
else/* 直前のパケットが再送パケットだった場合 */
　　/* 推定値の算出 */
　　if(直前のパケットがタイムアウトによる再送として算出された推定値だった場合)
　　　　前回のタイムアウトから続けて送出された再送*2 と Duplicate ACK による
　　　　再送の推定値を算出.
　　else
　　　　タイムアウトによる再送*1 と Duplicate ACK による
　　　　再送の推定値を算出する.
　　/* 推定値の表示 */
　　if(Duplicate ACK を 3 個以上受信した場合)
　　　　Duplicate ACK による再送として算出した推定値を表示.
　　else
　　　　タイムアウトによる再送として算出した推定値を表示.

*1 cwnd に 1MSS を設定する.
*2 現在の cwnd=直前の cwnd+ACK 数*MSS を設定する.

3.2.2　同一の推定値をまとめる機能

再送発生回数を N とした場合，最大で 2^N の推定値を保持することになるため，メモリ使用量が膨大になり，推定値の選択や算出に時間がかかってしまう．そこで，同じ推定値がリスト構造中に複数存在する場合には，それらを 1 つの推定値にまとめることでメモリの節約と推定値を計算する際の高速化を図った．現在の cwnd と ssthresh，現在のパケットの種類以外に，2.5 節で説明したようにバックアップした直前の値を現在の cwnd に設定する場合があるため，以下の値がすべて一致するパケットの推定値のみを 1 つにまとめる．

- 現在の推定した cwnd
- 現在の推定した ssthresh
- 現在のパケットの種類（通常パケットか再送パケット）
- 直前の推定した cwnd
- 直前の推定した ssthresh

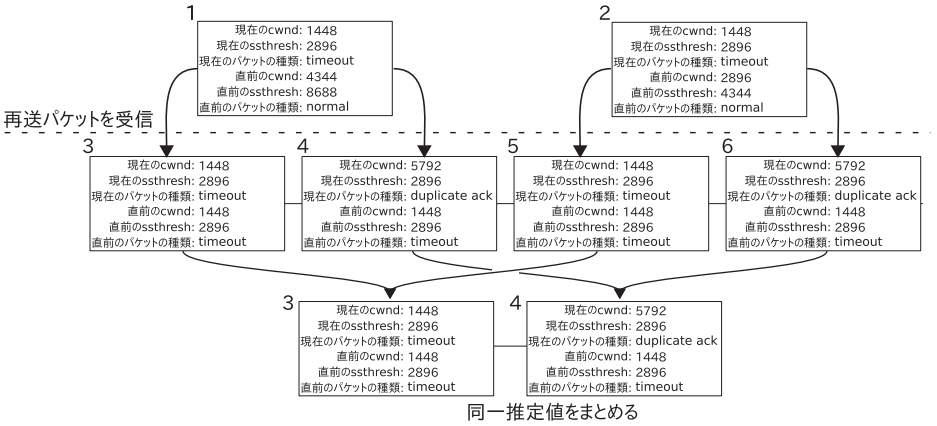

図 5　同一の推定値をまとめる
Fig. 5　An example of merging same estimations.

- 直前のパケットの種類（通常パケットか再送パケット）

図 5 に同一の推定値をまとめる例を示す．四角は上記 6 つの推定値の組を示している．再送パケットを受信し，タイムアウトと Duplicate ACK による再送の 2 つの推定値を推定値 1，2 から算出し，1 からタイムアウトとして算出した推定値を 3，Duplicate ACK として算出した推定値を 4，2 からタイムアウトとして算出した推定値を 5，Duplicate ACK として算出した推定値を 6 とする．このとき，算出した推定値 3 と 5，4 と 6 は上記の全項目が同一であるので，推定値をまとめる処理を行い，推定値 3 と 4 のみを残すこととなる．

3.2.3　誤った cwnd の修正

表示に使用する推定値の cwnd が，連続して送出されたパケットのデータ長の和より小さい場合，推定値が間違っている可能性が高い．たとえば，タイムアウトによる再送と判定したが，実際には Duplicate ACK による再送が発生していた場合，推定した cwnd は 1 MSS だが，1 MSS 以上のパケットが送出される可能性がある．この場合，連続して送出されたパケットのデータ長の和以上で，最も近いリスト中の推定値を表示に使用することで推定値を修正する．リストに保持したすべての推定値が，連続して送出されたパケットのデータ長の和よりも小さい場合は以下の手順をとる，まず，直前のリストを複製し，新たなリストを作成する．次にリストの各推定値を複製し，複製した推定値の cwnd に，連続して送出されたパケットのデータ長の和を設定し，現在のリストに追加する．複製した推定値の中で表示

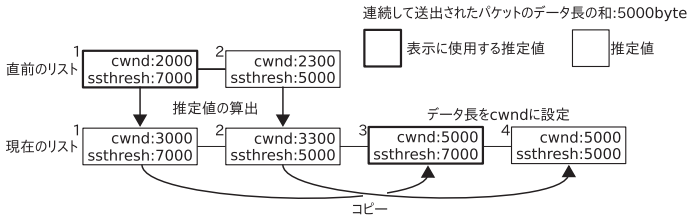

図 6　誤った推定値の修正
Fig. 6　Fix wrong values.

に使用する推定値は，直前に表示に使用していた推定値から算出された推定値を複製し，連続して送出されたパケットのデータ長の和を cwnd に設定した推定値である．間違っている値を削除しないのは，中間地点でのキャプチャでは，連続して送信されているように見えるパケットの間にあった ACK を見逃している可能性があるためである．ssthresh は連続したパケットの送出量では間違っているかどうかを判断できないため，複製元の ssthresh の値をそのまま使用する．**図 6** に誤った cwnd の修正の例を示す．図 6 の四角は推定値を示し，太枠は表示に使用する推定値を示す．連続して送出されたパケットのデータ長の和は 5,000 バイトであり，直前のリストの 1 番目の推定値は cwnd が 2,000，ssthresh が 7,000，2 番目の推定値は cwnd が 2,300，ssthresh が 5,000 とする．このとき，連続して送出されたパケットのデータ長の和より，表示に使用する推定値の cwnd もその他の推定値の cwnd も小さいため，推定値の cwnd が間違っているといえる．そこで直前のリストの 1 番目の推定値から算出した推定値は表示せず，以下の手順による cwnd の修正を行う．

(1)　直前のリストを現在のリストとして複製する．
(2)　現在のリストにある 1，2 番目の推定値を複製し，3，4 とする．
(3)　現在のリストの 3，4 番目の推定値の cwnd に連続して送出されたパケットのデータ長の和（5,000 バイト）を設定する．
(4)　直前に表示に使用していた 1 番目の推定値を複製し，連続して送出されたパケットのデータ長の和を設定した現在のリストの 3 番目の推定値を表示に使用する．

3.3　tcpdump への推定機能の追加

検討した手法によって輻輳制御パラメータのリアルタイム推定を行うことができることを確認するために，tcpdump に検討した手法による輻輳制御パラメータの推定機能を実装した．tcpdump とは，ネットワークを流れるパケットを取り込み，1 パケットごとに即座にその内容を解析，表示できるプログラムである．解析内容をリアルタイムに確認できることから，ネットワークの各種設定の確認や，障害原因の解析，ネットワーク通信の学習など，様々な目的で利用されている．tcpdump の TCP パケットの情報を標準出力に出力する関数である tcp_print に輻輳制御パラメータを推定する機能を追加した．tcp_print は 1 パケットごとに tcpdump のメイン関数から呼び出される．実装した推定機能は SYN フラグがパケットにセットされていた場合，source の IP アドレスとポート番号，destination の IP アドレスとポート番号，MSS，ウインドウスケールをコネクション情報として設定し，この情報をもとにコネクション別に輻輳制御パラメータの推定を行う．パケットの情報はコネクション情報の中にホスト別のリスト構造として時間軸に沿って保持する．ただし再送が発生しない場合は，1 つの推定値しか保持しない．次に，推定値の算出，同一の推定値のまとめ，表示に使用する値の選択を行い，推定値を表示する．表示する推定した cwnd と ssthresh は元の tcpdump の表示を崩さないように，tcpdump が出力する情報の行末に `<cwnd_value ssthresh_value>`，タイムアウトによる再送のパケットの場合，`<ret cwnd_value ssthresh_value>` を，Duplicate ACK による再送の場合，`<dup cwnd_value ssthresh_value>` を追加表示する．**図 7** に推定機能を実装した tcpdump の表示結果を示す．

```
17:04:31.398906 IP 160.194.128.8.40183 > 10.0.0.3.6666: S 1283051085:1283051085 (0)  win 65535
17:04:31.399581 IP 10.0.0.3.6666 > 160.194.128.8.40183: S 1874944744:1874944744 (0)  ack 1283051086 win 65535
17:04:31.399605 IP 160.194.128.8.40183 > 10.0.0.3.6666: . ack 1 win 8326 <1448 1073725440>
17:04:31.406264 IP 160.194.128.8.40183 > 10.0.0.3.6666: P 1:1025 (1024)  ack 1 win 8326 <1448 1073725440>
17:04:31.506618 IP 10.0.0.3.6666 > 160.194.128.8.40183: . ack 1025 win 8326
17:04:31.506675 IP 160.194.128.8.40183 > 10.0.0.3.6666: . 1025:2473 (1448)  ack 1 win 8326 <2896 1073725440>
17:04:31.506685 IP 160.194.128.8.40183 > 10.0.0.3.6666: . 2473:3921 (1448)  ack 1 win 8326 <2896 1073725440>
17:04:31.507643 IP 10.0.0.3.6666 > 160.194.128.8.40183: . ack 3921 win 8145
17:04:31.507664 IP 160.194.128.8.40183 > 10.0.0.3.6666: . 3921:5369 (1448)  ack 1 win 8326 <4344 1073725440>
17:04:31.507673 IP 160.194.128.8.40183 > 10.0.0.3.6666: . 5369:6817 (1448)  ack 1 win 8326 <4344 1073725440>
17:04:31.507682 IP 160.194.128.8.40183 > 10.0.0.3.6666: . 6817:8265 (1448)  ack 1 win 8326 <4344 1073725440>
17:04:31.508500 IP 10.0.0.3.6666 > 160.194.128.8.40183: . ack 5369 win 8326
17:04:31.508560 IP 160.194.128.8.40183 > 10.0.0.3.6666: . 8265:9713 (1448)  ack 1 win 8326 <5792 1073725440>
17:04:31.508570 IP 160.194.128.8.40183 > 10.0.0.3.6666: . 9713:11161 (1448)  ack 1 win 8326 <5792 107372544>
17:04:31.608560 IP 10.0.0.3.6666 > 160.194.128.8.40183: . ack 6817 win 8326
17:04:31.608619 IP 160.194.128.8.40183 > 10.0.0.3.6666: . 11161:12609 (1448)  ack 1 win 8326 <7240 1073725440>
17:04:31.608629 IP 160.194.128.8.40183 > 10.0.0.3.6666: . 12609:14057 (1448)  ack 1 win 8326 <7240 1073725440>
17:04:31.609585 IP 10.0.0.3.6666 > 160.194.128.8.40183: . ack 6817 win 8326
17:04:31.980517 IP 160.194.128.8.40183 > 10.0.0.3.6666: . 6817:8265 (1448)  ack 1 win 8326 <ret 1448 2896>
17:04:31.981469 IP 10.0.0.3.6666 > 160.194.128.8.40183: . ack 8265 win 8145
```

図 7　輻輳制御パラメータの表示
Fig. 7　Display of congestion control parameters.

4. 実験と評価

すでに報告した簡易的な評価[10]では，コネクションのエンドポイントでパケットが送出された際に使用された TCP 実装の値とリアルタイムでの推定値がほぼ一致することを確認した．本稿ではコネクションのエンドポイントと通信の中間点で，データパケットの損失と ACK パケットの損失の輻輳制御パラメータの推定への影響を述べる．通信上の問題を詳細に分析するためには，観測点の違い，データパケットや ACK パケットの損失にかかわらず，パケットを送出する際に使用される TCP 実装内の値と推定した値が完全に一致することが必要である．しかし，3.1 節で述べた問題点に加え，コネクションのエンドホスト以外でパケットを観察する場合，パケットの損失によって，TCP 実装内で使用される値と推定した値が完全に一致することは期待できない．TCP は 1 MSS 単位でパケットを送出することが多いため，TCP 実装内で使用される値と推定した値の差が ±1 MSS 以下に収まれば，推定したパケットの送出量と実際のパケットの送出量の差を 1 パケット以内に抑えることができるため，ネットワークのトラブルシューティングに大きな影響を与えないと考えられる．そこで，本研究では推定した値の許容誤差を ±1 MSS 以下とすることにした．

さらに，tcpdump と輻輳制御パラメータの推定機能を実装した tcpdump でのリアルタイム性能の比較を行った結果を述べ，続いてリスト構造として複数の推定値を保持する場合のメモリ使用量の評価結果を述べる．

4.1 パケットの損失の影響の評価

4.1.1 評 価 方 法

図 8 に実験に用いた環境を示す．ルータ R1，R2 は PC に 2 枚のネットワークインタフェースカードを挿入して構成した．ホスト A，B，観測点用 PC，ルータ R1，R2 にはすべて FreeBSD 8.0 をインストールした．各端末は 100 Mbps のイーサネットで接続し，HUB には Shared HUB を用いた．

図 8 のホスト A からホスト B へ 1.5 GB のデータを転送し，ホスト A，ホスト B，観測点で輻輳制御パラメータの推定を行い，推定値と TCP 実装の値の一致率を評価した．ホスト B へ向かうパケットはデータパケット，ホスト A へ向かうパケットは ACK パケットとなる．ルータ R1 でホスト A 宛の ACK パケット，ルータ R2 でホスト B 宛のデータパケットに対して通信帯域を制限するためのツールである ipfw[18] を用いた通信帯域制御を行い，パケットを損失させた．今回の実験では FTP のスループットを基準としてパケットの損失率を決定した．損失なしの場合は約 8 MB/s の転送速度となったことから，スループットが

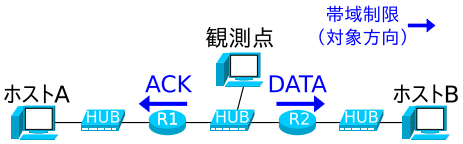

図 8　実験環境
Fig. 8　Study procedure.

1/100 の 80 KB/s になるように帯域制限を行った場合を損失率大とした．ルータ R1 とルータ R2 の ipfw に同じパラメータを設定し，データパケットと ACK パケットを約 10%ずつ損失させた場合にスループットが 80 KB/s になったことから損失率大は約 10%の損失とした．損失なしの場合のスループット 8 MB/s と 80 KB/s の間を 4 等分し，中は 2 MB/s，小は 4 MB/s とした．そのときの損失率は中が約 4%，小は損失率は約 1%となった．データパケットと ACK パケットに対する大，中，小，なしの損失率の組合せ全 16 通りの通信状態を作成した．実際に TCP 実装で使用されている輻輳制御パラメータの値と推定値を比較するために，データ部分を改竄しても影響がない FTP による通信を用い，FTP のデータ部分に TCP 実装の cwnd と ssthresh の値を挿入し，取得できるようにした．また，SACK オプション，タイムスタンプ，ウインドウスケールオプションが有効であり，MSS は 1,448 バイトである．

4.1.2 実 験 結 果

表 1 にデータパケットと ACK パケットの損失率の組合せについて，各ホストでキャプチャしたパケット数と損失したパケットの数が，キャプチャした全パケット数に占める割合を示す．図 8 で示した Shared HUB によってコリジョンが発生するため，表 1 のデータ損失なしと ACK 損失なしの組合せでも損失パケットが 2,544 個発生した．**表 2** にデータパケットと ACK パケットの損失率の組合せによる各ホストでの推定値が TCP 実装の値と一致するパケットの数が，キャプチャした全パケットに占める割合を示す．データ損失率なしと ACK の各損失率の組合せでは，どの場所で推定しても高い一致率となり，十分実用的であると考えられる．それ以外の損失率の組合せでは，cwnd の一致率は 50%以下で，10%以下の場合もある．ssthresh は 70%以上の一致率の組合せが多く，cwnd よりも一致率が高い．cwnd，ssthresh ともに，データ損失率が高いほど一致率が高く，ホスト A での推定よりも，ホスト B，観測点での一致率が低い．ホスト A で推定する場合，データパケットはすべてキャプチャできることから（データ損失なし），他の場所での推定に比べ一致率が上

表 1　キャプチャしたパケット数とドロップ率
Table 1　Received packets and rate of dropped packets.

データパケットの損失率	ACK パケットの損失率	データパケット				ACK パケット				実際の損失率（%）	
		ホスト A	ホスト B	観測点	損失したパケット数	ホスト A	ホスト B	観測点	損失したパケット数	データパケットの損失率（%）	ACK パケットの損失率（%）
なし	なし	1,053,895	1,053,895	1,053,895	0	697,337	700,073	699,881	2,544	0.00	0.36
なし	小	1,053,763	1,053,763	1,053,763	0	688,460	700,145	699,984	11,524	0.00	1.67
なし	中	1,053,567	1,053,567	1,053,567	0	669,054	700,652	700,483	31,429	0.00	4.70
なし	大	1,053,367	1,053,367	1,053,367	0	629,502	700,821	700,637	71,135	0.00	11.30
大	なし	1,169,040	1,053,923	1,053,923	115,117	976,483	976,483	976,483	0	9.85	0.00
大	小	1,169,825	1,054,951	1,054,951	114,874	963,937	976,799	976,799	12,862	9.82	1.33
大	中	1,173,508	1,057,590	1,057,590	115,918	937,366	977,650	977,650	40,284	9.88	4.30
大	大	1,180,152	1,064,453	1,064,453	115,699	882,640	979,004	979,004	96,364	9.80	10.92
中	なし	1,098,980	1,053,707	1,053,707	45,273	882,435	882,435	882,435	0	4.12	0.00
中	小	1,100,129	1,054,036	1,054,036	46,093	872,934	884,551	884,551	11,617	4.19	1.33
中	中	1,100,162	1,054,525	1,054,525	45,637	845,972	882,943	882,943	36,971	4.15	4.37
中	大	1,102,550	1,056,789	1,056,789	45,761	793,968	880,605	880,605	86,637	4.15	10.91
小	なし	1,069,706	1,055,584	1,055,584	14,122	804,662	804,662	804,662	0	1.32	0.00
小	小	1,069,453	1,055,450	1,055,450	14,003	793,390	803,885	803,885	10,495	1.31	1.32
小	中	1,069,139	1,055,194	1,055,194	13,945	769,287	802,869	802,869	33,582	1.30	4.37
小	大	1,069,334	1,055,185	1,055,185	14,149	723,674	802,257	802,257	78,583	1.32	10.86

表 2　推定値と TCP 実装の値の一致率 1
Table 2　Accuracy of the estimation of TCP congestion parameters (No.1).

データパケットの損失率	ACK パケットの損失率	一致した割合-cwnd（%）			一致した割合-ssthresh（%）		
		ホスト A	ホスト B	観測点	ホスト A	ホスト B	観測点
なし	なし	99.95	99.94	99.94	100.00	100.00	100.00
なし	小	99.95	99.94	99.94	100.00	100.00	100.00
なし	中	99.96	99.94	99.94	100.00	100.00	100.00
なし	大	91.25	92.11	92.11	100.00	100.00	100.00
大	なし	46.57	26.62	29.94	80.74	85.69	82.02
大	小	46.59	26.19	29.62	79.33	84.83	81.42
大	中	45.70	25.27	28.44	76.40	83.45	79.39
大	大	44.24	24.08	27.19	70.16	80.65	75.61
中	なし	35.51	18.76	21.14	72.79	73.98	77.50
中	小	36.10	18.02	20.31	72.08	74.05	77.60
中	中	37.15	16.63	18.65	72.37	74.48	77.59
中	大	38.13	14.91	16.52	68.57	71.93	74.31
小	なし	11.82	10.08	11.17	64.12	62.83	70.05
小	小	12.11	9.80	10.74	65.40	64.40	70.68
小	中	12.55	8.47	9.08	65.67	63.49	69.97
小	大	16.14	6.87	7.02	50.43	61.11	64.59

昇する．損失率が高い方が一致率が高くなる理由は，損失率が高くなると輻輳が頻繁に発生し，cwnd が 1 MSS に設定され，ssthresh が 2 MSS に収束する確率が高くなるためである．一方損失率が低い場合には，輻輳が発生する確率が低く，再送原因を誤判定してしまうと，算出した複数の推定値からの表示用推定値の選択を誤ってしまい，さらに後続のパケットの推定値の表示でも，誤って表示した推定値から推定された値の表示を続けてしまうことで，いっそうの一致率の低下を招いてしまう．仮に再送原因を誤らず，複数の推定値の中から正しい値を選択できた場合，どの程度一致率が上昇するかを計算した結果を**表 3** に示す．データ損失なしの場合を除いて cwnd，ssthresh ともに数%から 20%程度，一致率が上昇している．**表 4** に ±1 MSS 以下の誤差を正しい推定値とした場合の一致率を示す．ほとんどの組合せで 80%以上の一致率となり，本研究の輻輳制御パラメータのリアルタイム推定手法による推定値が，設定した許容誤差である ±1 MSS 以内に収まっていることが確認できた．

4.2　輻輳制御パラメータの推定機能のリアルタイム性能についての評価

4.2.1　評 価 方 法

tcpdump は，ネットワークを流れるパケットをリアルタイムに解析し，結果を表示する機能に加え，ネットワークからキャプチャしたパケット情報をいったんファイルに記録して

表 3　推定値と TCP 実装の値の一致率 2
Table 3　Accuracy of the estimation of TCP congestion parameters (No.2).

データパケットの損失率	ACK パケットの損失率	一致した割合-cwnd（%）			一致した割合-ssthresh（%）		
		ホスト A	ホスト B	観測点	ホスト A	ホスト B	観測点
なし	なし	99.95	99.94	99.94	100.00	100.00	100.00
なし	小	99.95	99.94	99.94	100.00	100.00	100.00
なし	中	99.96	99.94	99.94	100.00	100.00	100.00
なし	大	91.25	92.11	92.11	100.00	100.00	100.00
大	なし	61.11	34.60	44.82	83.14	88.69	89.77
大	小	61.23	34.30	44.03	82.27	87.86	89.05
大	中	60.38	34.06	42.66	80.26	86.74	87.20
大	大	58.22	33.74	40.68	75.85	84.73	83.93
中	なし	59.07	24.99	31.95	79.55	81.95	87.19
中	小	59.94	23.96	30.55	79.57	80.44	86.86
中	中	60.07	22.40	28.10	79.63	79.05	85.16
中	大	60.61	20.85	25.31	77.57	75.49	80.67
小	なし	25.05	17.10	20.69	79.16	81.17	82.45
小	小	25.55	16.21	19.78	80.24	80.32	82.51
小	中	26.49	13.54	16.60	81.00	76.93	79.83
小	大	33.59	10.27	12.36	65.27	69.32	73.13

表 4　推定値と TCP 実装の値の一致率 3
Table 4　Accuracy of the estimation of TCP congestion parameters (No.3).

データパケットの損失率	ACK パケットの損失率	一致した割合-cwnd（%）±1 MSS			一致した割合-ssthresh（%）±1 MSS		
		ホスト A	ホスト B	観測点	ホスト A	ホスト B	観測点
なし	なし	99.95	99.95	99.95	100.00	100.00	100.00
なし	小	99.95	99.95	99.95	100.00	100.00	100.00
なし	中	99.96	99.94	99.95	100.00	100.00	100.00
なし	大	91.31	92.17	92.17	100.00	100.00	100.00
大	なし	93.01	91.90	94.09	85.87	92.10	92.27
大	小	92.15	91.27	93.42	85.23	91.54	91.78
大	中	89.90	90.20	91.80	83.35	90.76	90.33
大	大	85.01	87.48	88.36	79.42	89.47	87.69
中	なし	89.31	90.33	93.34	84.78	88.20	90.58
中	小	89.41	89.64	93.08	84.76	87.52	90.46
中	中	88.98	88.92	92.04	84.31	86.89	89.49
中	大	87.03	85.11	87.84	82.32	86.20	87.38
小	なし	84.28	88.32	87.82	82.81	86.30	86.38
小	小	85.35	87.94	87.43	83.88	86.08	86.28
小	中	85.90	84.09	84.61	84.48	83.67	84.38
小	大	80.50	69.61	71.41	76.56	81.18	80.80

表 5　コンピュータの仕様
Table 5　Specification for computer.

項目	内容
CPU	AMD Athlon 64 X2 Dual Core Processor 6000+
Memory	2 GB
HardDisk	80 GB 7200 rpm (SATA2)

おき，後で読み込んで解析して結果を表示する機能も有している．ファイルから読み込む場合，ファイルに格納された最後のパケットまで，パケットのキャプチャ時間に関係なく連続して解析と結果の表示を行うことから，この際の 1 パケットあたりの処理時間を求めることで，tcpdump のリアルタイム性能を求めることができる．そこで，オリジナルの tcpdump と，リアルタイム推定手法を組み込んだ tcpdump に対して 1 パケットあたりの処理時間を求め，組み込んだリアルタイム推定方法が，tcpdump のリアルタイム性能にどの程度の影響を与えているかを評価した．評価に使用したパケットは，4.1 節で使用した FTP 通信で，図 1 の観測点でキャプチャしたものである．1 パケットあたりの処理時間を計測するために使用したコンピュータの仕様を**表 5** に示す．

4.2.2　評 価 結 果

表 6 に tcpdump と輻輳制御パラメータの推定機能を実装した tcpdump の，読み込み開始から最後のパケットが表示されるまでの処理時間と，1 パケットあたりの処理時間，輻輳制御パラメータの推定機能を実装したことによる処理時間の増加率を示す．結果から，データパケットと ACK パケットの損失率の組合せにほとんど影響を受けず，平均して約 24%，輻輳制御パラメータの推定機能を実装した tcpdump の処理が遅いことが分かった．しかし，1 パケットあたりの処理時間の最も遅いものでも 75.9 マイクロ秒であり，たとえば，ネットワーク管理者が，実際に通信を観測する場面において，ほとんど影響はないと考えられる．このことから，本研究の推定手法は，tcpdump のリアルタイム性能に影響を与えていないことが分かった．また，実験の結果から，使用したコンピュータにおいて，輻輳制御パラメータの推定機能を実装した tcpdump は，1 秒間に約 13,000 パケット以上の処理を行えると考えられる．

4.3　メモリ使用量の評価

4.3.1　評 価 方 法

ファイル転送を行うためのプロトコルである FTP による通信に対して，輻輳制御パラメータの推定機能を実装した tcpdump でリアルタイム推定を行う．キャプチャした全パケッ

表 6　tcpdump と輻輳制御パラメータの推定機能を実装した tcpdump の 1 パケットの処理にかかる時間の比較
Table 6　Processing speed per one packet for original tcpdump and tcpdump with real-time estimation of TCP congestion control parameters.

データパケットの損失率	ACK パケットの損失率	観測点でのパケット数	tcpdump		輻輳制御パラメータの推定機能を実装した tcpdump		
			表示完了までの処理時間（秒）	1 パケットあたりの処理時間（マイクロ秒）	表示完了までの処理時間（秒）	1 パケットあたりの処理時間（マイクロ秒）	1 パケットあたりの処理時間の増加割合（%）
なし	なし	1,753,776	50.7	28.9	64.7	36.9	27.8
なし	小	1,753,747	50.6	28.8	65.1	37.1	28.8
なし	中	1,754,050	51.5	29.4	64.8	37.0	25.9
なし	大	1,754,004	51.3	29.2	64.5	36.8	25.7
大	なし	1,053,923	65.2	61.8	80.0	75.9	22.7
大	小	1,054,951	65.0	61.6	80.0	75.8	23.0
大	中	1,057,590	65.3	61.8	80.3	75.9	22.8
大	大	1,064,453	65.6	61.6	80.3	75.4	22.4
中	なし	1,936,142	59.6	30.8	73.3	37.9	22.9
中	小	1,938,587	59.5	30.7	73.5	37.9	23.5
中	中	1,937,468	59.6	30.8	73.6	38.0	23.5
中	大	1,937,394	59.6	30.7	73.4	37.9	23.2
小	なし	1,860,246	54.9	29.5	68.4	36.8	24.6
小	小	1,859,335	55.2	29.7	68.5	36.8	24.0
小	小	1,858,063	55.1	29.7	68.6	36.9	24.5
小	大	1,857,442	55.7	30.0	69.2	37.2	24.2

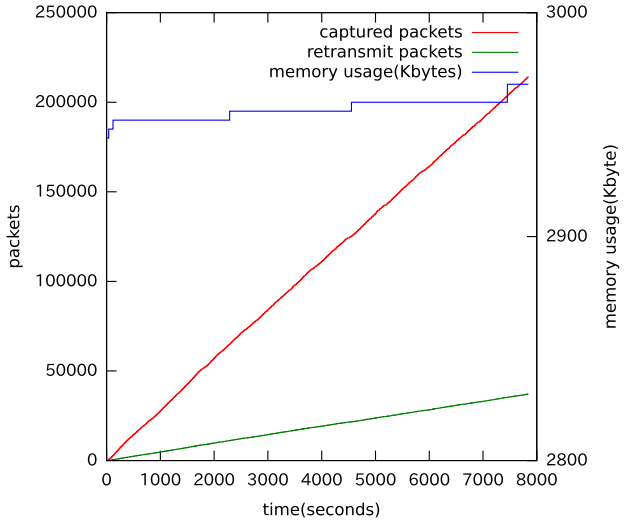

図 9　メモリ使用量
Fig. 9　Memory usage.

ト数と再送パケットの数が輻輳制御パラメータの推定機能を実装した tcpdump のメモリ使用量に与える影響を評価する．

4.3.2　実験結果

キャプチャした全パケット数，再送パケット数，輻輳制御パラメータの推定機能を実装した tcpdump のメモリ使用量をグラフにしたものを**図 9** に示す．横軸は tcpdump を起動してからの経過時間（秒），左の縦軸はパケット数，右の縦軸はメモリ使用量（バイト）である．赤線がキャプチャした全パケット数，緑線が再送パケット数，青線が輻輳制御パラメータの推定機能を実装した tcpdump のメモリ使用量である．図 9 から，再送パケットの数が増加しても，メモリ使用量は常時約 3 M バイト程度であり，計算に必要な直前の推定値のみを保持するようにしたことと，同一の推定値をまとめる機能によって，使用メモリ量を低く抑えることができていることを確認できた．

5.　考　　察

表 2 および表 3 から，±1 MSS の誤差を正しい推定値としなかった場合，特に cwnd の一致率が低いことが分かる．本章では，この原因について検討した結果を述べる．

表 7　推定した cwnd の誤差の平均値
Table 7　Average error in the estimated cwnds.

データパケットの損失率	ACK パケットの損失率	全体			マイナス						プラス					
		cwnd の誤差の平均（byte）			パケット数			誤差の平均（byte）			パケット数			誤差の平均（byte）		
		ホスト A	ホスト B	観測点	ホスト A	ホスト B	観測点	ホスト A	ホスト B	観測点	ホスト A	ホスト B	観測点	ホスト A	ホスト B	観測点
なし	なし	−8.62	−0.38	−1.15	504	358	404	−17,947.73	−13,865.21	−13,444.18	0	250	199	0	18,261.82	21,246.59
なし	小	−9.13	6.01	2.18	532	263	367	−18,002.02	−9,139.47	−9,362.68	1	350	229	1,448.00	24,881.12	24,976.94
なし	中	−3.11	15.47	12.78	305	78	82	−12,471.79	−7,852.62	−8,193.56	131	528	521	4,167.15	31,891.68	27,008.98
なし	大	−1.52	−9.41	−9.46	91,388	82,091	82,091	−11,726.83	−11,738.82	−11,743.08	393	664	663	9,270.15	30,866.35	31,098.55
大	なし	540.80	758.65	486.48	151,693	75,739	83,075	−1,290.73	−1,379.21	−1,368.52	470,339	694,466	732,565	1,994.83	1,346.68	1,638.15
大	小	603.83	799.17	529.10	151,433	79,161	84,508	−1,300.08	−1,402.05	−1,403.29	470,875	696,425	735,535	2,165.48	1,424.46	1,715.84
大	中	748.95	824.62	630.18	146,182	86,699	84,807	−1,360.69	−1,508.23	−1,532.82	488,581	700,639	751,659	2,521.90	1,582.57	1,922.71
大	大	1,035.07	853.61	787.98	142,404	110,010	97,360	−1,506.11	−1,673.74	−1,707.50	513,559	695,551	759,117	3,260.19	1,969.93	2,367.5
中	なし	729.04	1,147.86	750.75	383,844	167,883	162,781	−1,066.45	−972.19	−934.34	322,358	685,191	700,826	2,673.70	1,453.43	1,722.30
中	小	742.06	1,299.17	807.37	377,344	152,913	143,979	−1,053.29	−1,005.02	−987.26	323,171	708,191	729,732	2,710.10	1,464.61	1,721.92
中	中	868.28	1,429.12	1,008.20	342,403	131,272	118,083	−1,060.46	−1,053.82	−1,059.37	346,876	745,039	774,009	2,976.75	1,530.01	1,817.97
中	大	1,237.38	1,612.05	1,405.79	293,041	108,042	90,532	−1,052.43	−1,138.25	−1,194.48	387,040	788,495	827,178	3,891.69	1,815.47	2,079.29
小	なし	1,797.32	1,208.76	1,300.98	204,157	278,580	288,931	−953.85	−1,079.80	−1,043.43	735,513	667,083	657,692	3,274.75	1,990.53	1,992.74
小	小	1,669.56	1,342.06	1,500.23	209,981	238,242	241,494	−928.98	−1,114.21	−1,101.00	726,556	710,234	709,647	3,087.42	1,953.32	2,108.54
小	中	1,692.60	1,877.24	2,005.23	205,200	177,283	168,423	−917.04	−1,291.98	−1,296.36	726,159	784,840	799,866	3,174.33	2,022.47	2,177.87
小	大	1,260.61	2,500.15	2,907.32	425,065	118,332	103,367	−1,211.98	−1,496.53	−1,580.62	468,417	860,813	887,295	4,041.03	2,145.75	2,483.34

表 7 に cwnd の推定値と TCP 実装の値の誤差の平均値を示す．表 7 の全体の項目には推定値から TCP 実装の値を引いた誤差の平均値，マイナスの項目には各パケットごとに cwnd の誤差を算出した場合にマイナスの誤差になるパケットの数とマイナスの誤差の平均値，プラスの項目には各パケットごとに cwnd の誤差を算出した場合にプラスの誤差になるパケットの数とプラスの誤差の平均値を示している．全体の項目の平均値を見ると，ほとんどの組合せでプラスの誤差になり，誤差は +1 MSS 程度となっている．また，マイナスの誤差のあるパケットはプラスの誤差のパケット数に比べ少なく，誤差の平均値も −1 MSS 程度であることと，プラスの誤差のあるパケットの数が多く，誤差の平均値も +1 MSS から +2 MSS と大きいことが分かる．

マイナスの誤差の原因は，ACK パケットをキャプチャできなかった場合，cwnd がキャプチャし損ねた ACK パケット分だけ増加しないためと，再送原因の誤判定によって，実際は Duplicate ACK による再送だが，タイムアウトと判定してしまった場合が考えられる．Duplicate ACK では cwnd に $ssthresh + DuplicateACK$ 数 $* MSS$ を設定するが，Duplicate ACK は同じシーケンス番号を要求する ACK を 3 つ以上受信している場合に再送が発生するため，cwnd は最小でも 5 MSS（Duplicate ACK の数の最小値が 3，ssthresh の最小値が 2 MSS であるため）となる．一方，タイムアウトでは cwnd に 1 MSS を設定する．この結果，推定値の方が小さい値になってしまうため，マイナスの誤差が発生してしまう．

プラスの誤差の原因は，(1) エンドホスト間と中間地点でパケットの到着順が異なってしまう場合があること，(2) ネットワークインタフェースからの古い cwnd を持った送信待ちパケットが送出される場合があることが分かった．以下にこれら 2 つの原因について詳しく述べる．

(1)　パケットの到着順がエンドホスト間と中間地点で異なってしまう場合の例

パケットの到着順がエンドホスト間と中間地点で異なってしまう場合の例を**図 10** に示す．丸で囲まれた数字はパケットの送出時間の順番である．**表 8** に図 10 の通信に対して cwnd の値を推定した場合の例を示す．MSS は 1,000 バイト，この時点での cwnd は 4,000 バイトとする．まず，ホスト B からパケット 1 が送出されるので，各ホストでは cwnd が 5,000 バイトに増加したと推定できる．次にホスト B からパケット 2 が送出される．このパケット 2 がホスト A へ届く前にパケット 3 が送出されるので，ホスト A から cwnd が 5,000 バイトでパケット 3 が送出される．しかし，中間地点とホスト B では，パケット 2 がパケット 3 より早くキャプチャされてしまう．したがって，中間地点とホ

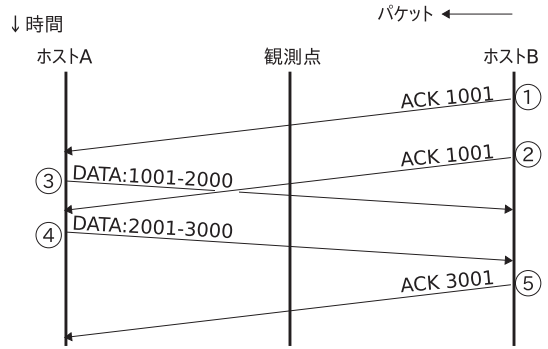

図 10　パケットの到着順がエンドホスト間と中間地点で異なってしまう場合の例
Fig. 10　Problem of the sequence of packets at the connection end point and at the halfway of connection.

表 8　パケットの到着順がエンドホスト間と中間地点で異なってしまう場合の例の cwnd
Table 8　cwnd in case of Fig. 9.

ホスト A		観測点		ホスト B	
到着順	cwnd	到着順	cwnd	到着順	cwnd
1	5,000	1	5,000	1	5,000
3	5,000	2	6,000	2	6,000
2	6,000	3	6,000	3	6,000
4	6,000	4	6,000	4	6,000
5	7,000	5	7,000	5	7,000

スト B ではまずパケット 2 を受信し，cwnd が 6,000 バイトに増加した後にパケット 3 が到着するため，パケット 3 は cwnd が 6,000 バイトで送出されたパケットと推定してしまう．この結果，+1 MSS の誤差が発生してしまう．このような現象は伝送路上で起こるだけでなく，OS のパケット処理順を原因とするものもある．**図 11** はホスト A の伝送路から受信したパケットと OS で処理を行ったパケットの順番を表した図である．まず，ACK1001 が伝送路から届き，OS で処理される．tcp_input 関数が呼び出され，そこからパケット送出関数が呼び出され，データ 1001 が送信要求される．このとき，キャプチャ処理も行われる．しかし，この一連の処理中に伝送路から届いた ACK2001 は待ちになり，データ 1001 の送出が完了した後に OS で処理される．結果として，伝送路でキャプチャしたパケットの順番，すなわち観測点，ホスト B でキャプチャした順番は ACK1001，

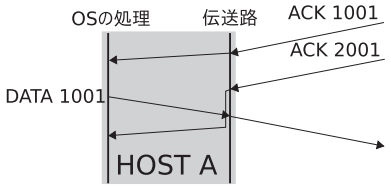

図 11　伝送路に送出されたパケットと OS で処理を行ったパケットの順番が異なる場合の例
Fig. 11　Problem of the sequence of packet processed by OS and monitored from the transmission channel.

ACK2001，データ 1001 となるが，OS で処理を行ったパケットの順番，すなわちホスト A でキャプチャした順番は ACK1001，データ 1001，ACK2001 となる．したがって，パケットの到着順がエンドホスト間と中間地点で異なってしまう場合と同様，プラスの誤差が発生してしまう．

(2)　ネットワークインタフェースからの送信待ちパケット

TCP は ACK を受信した時点で cwnd を増加させ，パケットの送信処理を行うが，送出されたパケットがネットワークインタフェースで送信待ちになることがある．後にこの送信待ちパケットはネットワークインタフェースから送信されるが，観測点と反対側のエンドホストでは 1 つ前の cwnd で送信されたパケットとして受信される．この現象が観測点で発生した例を**表 9** に示す．表中の cwnd と ssthresh はパケットが送信されたときのホスト A の TCP 実装で使用された cwnd と ssthresh である．13 番目から 17 番目までのパケットが連続して送信されているが，このときの cwnd は 19,356 バイトで，5 パケットの送信後でも 19,356 − 6,704 = 12,652 バイトの余裕がある．次に 18 番目のパケットで ACK を受信し，通常であれば cwnd が増加する．しかし，19 番目のパケットから 26 番目のパケットは cwnd が 19,356 バイトのままで送信され，27，28 番目のパケットは 19,464 バイトで送信されている．これは cwnd の残り 12,652 バイトのパケットが，18 番目の ACK を受信する前にネットワークインタフェースに届いて送信待ちになり，18 番目の ACK を受信した後にネットワークインタフェースから送信されたと考えられる．19 番目から 26 番目のパケットのデータ長の和は 11,584 バイトとなり，13 番目から 17 番目までのデータ長の和 6,704 バイトを足してみると 18,288 バイトとなる．cwnd が 19,356 バイトで送出されたパケットのデータ長の和に 1 MSS を足すと 19,736 となり，cwnd の 19,356 バイトを超えてしまう．TCP は 1 MSS 区切りでパケットを送出するため，cwnd が 19,356 バイトで送出できるパケットは 13 番目のパケットから 26 番目のパケットまで

表 9　ネットワークインタフェースで送信待ちだったパケットが送出される例
Table 9　An example of packets defered at NIC.

No	capturetime	src	dst	ack	seq begin	datalen	cwnd	ssthresh
1	18:11:03.094607	A	B	1	80,377	1,448	18,357	4,344
2	18:11:03.094611	B	A	67,345				
3	18:11:03.094949	A	B	1	81,825	1,448	18,471	4,344
4	18:11:03.094957	A	B	1	83,273	1,448	18,471	4,344
5	18:11:03.094960	B	A	68,793				
6	18:11:03.095301	B	A	71,689				
7	18:11:03.095629	B	A	73,137				
8	18:11:03.095958	B	A	76,033				
9	18:11:03.096290	B	A	77,481				
10	18:11:03.096622	B	A	80,377				
11	18:11:03.096950	B	A	81,825				
12	18:11:03.097286	B	A	84,721				
13	18:11:03.097623	A	B	1	84,721	912	19,356	4,344
14	18:11:03.119580	A	B	1	85,633	1,448	19,356	4,344
15	18:11:03.119589	A	B	1	87,081	1,448	19,356	4,344
16	18:11:03.119596	A	B	1	88,529	1,448	19,356	4,344
17	18:11:03.119602	A	B	1	89,977	1,448	19,356	4,344
18	18:11:03.120352	B	A	87,081				
19	18:11:03.120450	A	B	1	91,425	1,448	19,356	4,344
20	18:11:03.120461	A	B	1	92,873	1,448	19,356	4,344
21	18:11:03.120473	A	B	1	94,321	1,448	19,356	4,344
22	18:11:03.120485	A	B	1	95,769	1,448	19,356	4,344
23	18:11:03.120494	A	B	1	97,217	1,448	19,356	4,344
24	18:11:03.120505	A	B	1	98,665	1,448	19,356	4,344
25	18:11:03.120514	A	B	1	100,113	1,448	19,356	4,344
26	18:11:03.120523	A	B	1	101,561	1,448	19,356	4,344
27	18:11:03.120946	A	B	1	103,009	1,448	19,464	4,344
28	18:11:03.120959	A	B	1	104,457	1,448	19,464	4,344
29	18:11:03.120963	B	A	88,529				
30	18:11:03.121333	A	B	1	105,905	1,448	19,571	4,344

となる．cwnd 分のパケットを送出し終えた後は，18 番目の ACK を受信した時点で TCP 実装の cwnd は 19,464 バイトに設定されているので，27，28 番目のパケットの cwnd は 19,464 バイトで送信されている．このような状況が発生しても，外部からは判別できない．検討した推定方法では ACK 受信直後に cwnd を増加させるため，推定値が TCP 実装の値よりも大きいものになってしまう．

実際の推定誤差の原因は，以上のような現象が組み合わさって発生しているため，個別の現象がどの程度発生するかを調べることは難しい．特に，今回の実験の観測点のように，通信の中間点で推定を行う際は，発生している推定誤差を招く状況を正確に判断できない．

6. ま と め

本研究では，TCP の輻輳制御アルゴリズムに基づいて通信のやりとりを解析し，cwnd と sshtresh の値をリアルタイムに推定するための手法を検討した．評価実験の結果，データパケットの損失がほとんどなければ，ACK パケットの損失数にかかわらず，正しい値を推定でき，ネットワーク管理者や TCP の学習者にとって十分実用的であることが分かった．また ±1 MSS の誤差を許容すると，データパケットおよび，ACK パケットの損失率の組合せのほとんどの場合で，80%以上正しい値を推定できた．±1 MSS の誤差を許容しない場合，cwnd の推定値の一致率が 7〜30%と低くなる場合があった．これはデータの送信側ホストと，コネクションの中間点および受信側ホストで，パケットの到着順が異なることや，送信側ホストのネットワークインタフェースに，古い cwnd の値を持つパケットが送信待ちとなっていること，再送原因の誤判定など，いくつかの現象が組み合わさって発生することが分かった．これらのうち，パケットの到着順と，再送原因の誤判定については，今回の実験環境に依存する可能性がある．実際の通信では，パケットはいくつものルータを経由するため，パケットの到着順が大きくずれてしまう可能性があることや，RTT の揺らぎが大きくなることによる再送原因の誤判定が増加する．パケットの到着順については，到着順がずれたパケットについては推定値と TCP 実装の値が一致しないが，cwnd の増加量を決める ACK パケットの数が変化しなければ，最終的には推定値と TCP 実装の値は一致することになり，問題とはならないであろう．一方再送原因を誤ると，間違った値で推定を続けてしまい，推定値と TCP 実装の値が一致しないパケットが他数発生してしまう可能性がある．これに関しては，今後，実際の通信で検証する必要性があると考えている．

輻輳制御パラメータの推定機能のリアルタイム性について評価を行った結果，オリジナルの tcpdump の 1 パケットあたりの処理時間に比べ，推定機能を組み込んだ tcpdump は，約 24%処理時間が増加することが分かったが，1 パケットあたりの処理時間は最も遅い場合でも約 76 マイクロ秒であり，実際の通信の観測において，リアルタイム性にほとんど問題のないことが分かった．

メモリ使用量の評価を行った結果，ほとんどメモリ使用量は常時 3 M バイト程度であり，推定値を計算に必要な直前のパケットのみ保持するようにしたことと，同一の推定値をまとめる機能により，メモリ使用量を低く抑えることに成功した．

今後の課題としては，すでに述べたように，実通信環境でのテストを行うことに加え，再送原因の誤判定をなくすための方法を検討することなどがあげられる．すでに述べたようにRTT，RTOを推定し，タイムアウトの判定を行えれば改善できるが，中間地点で正確なRTTを計測できないという問題がある．この問題を解決できたとしても，RTTの計測には前のタイムスタンプの値などを保持しなければならないため，メモリ使用量が膨大になる可能性がある．また，今回対象としたFreeBSDで採用されているNewReno以外の輻輳制御アルゴリズムを採用するOSもある．たとえばLinuxではBIC[19]やCUBIC[20]といったアルゴリズムを用いることができる．そこで，他の輻輳制御アルゴリズムにも対応した推定方式を追加し，評価することもあげられる．

参 考 文 献

1) Jacobson, V.: Congestion avoidance and control, *SIGCOMM*, pp.314–329 (1988).
2) Steven McCanne, S.F.: The Network Simulator-ns-2. http://www.isi.edu/nsnam/ns/
3) 丸山純一，長谷川剛，村田正幸：エッジルータにおける改造TCPの検出・制御手法の提案（トラヒック解析・制御（1）），電子情報通信学会技術研究報告，IN，情報ネットワーク，Vol.106, No.578, pp.231–236 (2007-03-01).
4) 比嘉玲華，松原幸助，岡廻隆生，山口実靖，小口正人：輻輳ウィンドウ及びパケット解析を用いたiSCSI遠隔ストレージアクセスの評価（遠隔データアクセス，ネットワーク技術及び一般），電子情報通信学会技術研究報告，CPSY，コンピュータシステム，Vol.108, No.361, pp.37–42 (2008-12-11).
5) 大岸智彦，井戸上彰，加藤聰彦，鈴木健二：プロトコルエミュレーション機能を用いたTCP/IPトラヒックの動作解析，全国大会講演論文集，Vol.55, No.3, pp.798–799 (1997-09-24).
6) 阿野茂浩，長谷川亨，加藤聰彦：ATM網におけるAvailable Bit Rate（ABR）上のTCPの動作解析，情報処理学会研究報告，マルチメディア通信と分散処理研究会報告，Vol.96, No.20, pp.55–60 (1996-02-29).
7) Paxson, V.: Automated Packet Trace Analysis of TCP Implementations, *SIGCOMM*, pp.167–179 (1997).
8) tcpdump org: tcpdump. http://www.tcpdump.org/
9) 茂木重憲，渡邊　晶：輻輳制御パラメータの自動解析を行うTCP通信解析ツールの研究，情報処理学会第71回全国大会 (2009).
10) 茂木重憲，渡邊　晶：輻輳制御パラメータのリアルタイム推定の研究，IPSJ SIG-IOT第7回研究会 (2009).
11) Floyd, S.: The NewReno Modification to TCP's Fast Recovery Algorithm, RFC 3782 (2004).
12) Floyd, S.: An Extension to the Selective Acknowledgement (SACK) Option for TCP, RFC 2883 (2000).
13) Allman, M. and Paxson, V.: On Estimating End-to-End Network Path Properties, *SIGCOMM*, pp.263–274 (1999).
14) Jacobson, V.: 4BSD header prediction, *ACM Computer Communication Review*, Vol.20, No.2, pp.13–15 (1990).
15) Floyd, S. and Henderson, T.: The NewReno Modification to TCP's Fast Recovery Algorithm, RFC 2582 (1999).
16) Jaiswal, S.: Inferring TCP Connection Characteristics Through Passive Measurements, *Proc. IEEE INFOCOM'04* (2004).
17) 大坐畠智，鈴木秀章，萩原洋一，寺田松昭，川島幸之助：TCP通信に対するパッシブ測定によるRTT推定法の一考察（インターネットの新しいサービスとその基盤技術及び一般），電子情報通信学会技術研究報告，IN，情報ネットワーク，Vol.104, No.659, pp.1–5 (20050211).
18) FreeBSD Foundation: ipfw. http://www.freebsd.org/doc/en/books/handbook/firewalls-ipfw.html
19) Xu, L., Harfoush, K. and Rhee, I.: Binary increase congestion control (BiC) for fast long-distance networks, *Proc. IEEE INFOCOM 2004, NSDI'06: 3rd Symposium on Networked Systems Design & Implementation*, USENIX Association (2004).
20) Ha, S., Rhee, I. and Xu, L.: CUBIC: A new TCP-friendly high-speed TCP variant, *SIGOPS Oper. Syst. Rev.*, Vol.42, No.5, pp.64–74 (2008).

(平成22年 3 月20日受付)
(平成22年12月 1 日採録)

茂木　重憲（正会員）

平成22年明星大学大学院情報学研究科卒業．同年株式会社ACCESS入社．

渡邊　　晶（正会員）

昭和 62 年同志社大学工学部電気工学科卒業．同年日立電線（株）入社．平成 8 年明星大学情報学部助手，平成 14 年より同大学専任講師，現在に至る．ネットワーク管理，コンピュータを利用した教育システムの研究に従事．教育システム情報学会会員．