Vol.2009-IOT-7 No.7
2009/10/9

輻輳制御パラメータのリアルタイム推定の研究

茂木重憲[†1]　渡邊晶[†2]

TCP の輻輳制御動作の解析や再送原因の解明には，cwnd や ssthresh の値が必要である．しかしこれらの値は TCP の内部変数で，伝送されるパケットからは入手できない．そこで伝送されるパケットに対して TCP の輻輳制御アルゴリズムと同じ動作を行い，リアルタイムに値を推定する研究を行った．FreeBSD の TCP を対象として，tcpdump に推定機能を実装し，TCP 内部の値と推定値が一致することを確認した．

Real-time Estimation of TCP Congestion Control Parameters

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

To analyze the behavior of the TCP congestion contorol, the value of the congestion control parameters, such as cwnd and ssthresh are required. These values could not be obtained from packets because they are not stored in the TCP header but they are maintained in the operating system. We present this method to estimate the TCP congestion contorol parameters in real time. In the method, we apply the TCP congestion control algorithms implemented in FreeBSD 8.0 to each observed packet. We have modified tcpdump to support our estimation method. The result of simple comparision test between estimated values and the values inside the FreeBSD kernel shows our estimation method goes almost well.

†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
School of Information Science, Meisei University

1. はじめに

TCP は，輻輳制御パラメータである cwnd や ssthresh を用いて 1 度に送出できるデータ量や輻輳動作を制御している．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などを把握できる．しかし，cwnd と ssthresh の値はネットワーク上に送出されない為，輻輳制御動作の推定には cwnd や ssthresh を推定する必要がある．

図 1 は 2 台のコンピュータ間の通信をパケット情報のモニタソフトウェアである tcpdump によってモニタした結果の一部である．図 1 を使用して輻輳制御動作を解析する方法を示す．まず，MSS は 1448 バイト，これまでの計算から 1 番目のパケットの cwnd は 11584 バイト，ssthresh は 1073725440 バイトである．8 番目のパケットのシーケンス番号は，12609:14057 で，3 番目のパケットのシーケンス番号 22745:24193 と比べてみると，8 番目のパケットのシーケンス番号の方が小さく，これは再送パケットであることが分かる．次に，直前の ACK を見てみるとシーケンス番号が 12609 から始まるパケットを要求する ACK が 4 つ来ていることが分かる（Duplicate ACK）．また，7 番目のパケットと 8 番目のパケットのタイムスタンプを見てみると，差は 41 マイクロ秒であるので，タイムアウトによって再送されたパケットの可能性は低いと考えられる．従って，8 番目のパケットは Duplicate ACK による再送であると判断でき，ssthresh を cwnd とウインドウサイズの小さい方の半分の値に設定する．これは Fast Retransmition[1)] という輻輳制御アルゴリズムで，Reno で採用されている．1 番目のパケットの cwnd が 11584 バイトで，かつ，4 番目から 7 番目の ACK は再送を促す ACK なので，cwnd は 11584 バイトから増加していない．ウインドウサイズは 8326 バイトなので，ウインドウサイズの半分の値を ssthresh に設定する．また，cwnd，ssthresh は MSS の倍数にする．

$$ssthresh = 8326/2/1448$$
$$ssthresh = ssthresh * 1448$$

となり，ssthresh は 2896 バイトになる．cwnd は MSS に Duplicate ACK の数を掛けた値に ssthresh を足した値になるので，

$$cwnd = 1448 * 4 + ssthresh$$

となり cwnd は 8688 バイトとなる．

このように tcpdump を用いて再送パケットやその原因を特定するには輻輳制御アルゴリズムに基づいて 1 パケット毎に 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．そこで，我々はモニタしたパ

```
1  17:04:32.674531 IP 160.194.128.8.40183 > 10.0.0.3.6666: . 19849:21297(1448) ack 1 win 8326
2  17:04:32.674542 IP 160.194.128.8.40183 > 10.0.0.3.6666: . 21297:22745(1448) ack 1 win 8326
3  17:04:32.674551 IP 160.194.128.8.40183 > 10.0.0.3.6666: . 22745:24193(1448) ack 1 win 8326
4  17:04:32.674556 IP 10.0.0.3.6666 > 160.194.128.8.40183: . ack 12609 win 8326
5  17:04:32.675465 IP 10.0.0.3.6666 > 160.194.128.8.40183: . ack 12609 win 8326
6  17:04:32.675473 IP 10.0.0.3.6666 > 160.194.128.8.40183: . ack 12609 win 8326
7  17:04:32.675598 IP 10.0.0.3.6666 > 160.194.128.8.40183: . ack 12609 win 8326
8  17:04:32.675639 IP 160.194.128.8.40183 > 10.0.0.3.6666: . 12609:14057(1448) ack 1 win 8326
```

図 1 再送パケットを含む tcpdump の出力の例
Fig. 1 Output of tcpdump that contain retransmited packets

ケットから輻輳制御パラメータを自動的に推定できれば有用であると考え，TCP 実装の輻輳制御動作に基づいてリアルタイムで輻輳制御動作を解析し，cwnd と sshtresh の値を推定するための手法を検討した．本稿では手法の詳細を示すとともに，tcpdump に cwnd と ssthresh のリアルタイム推定機能を組込み，簡易的な評価として推定値と TCP 実装の値を比較した結果を示す．

2. 輻輳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するが，我々は FreeBSD 8.0 に実装されている TCP のバージョンである NewReno（RFC 2582）[2]，SACK[3]，オプションとして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[4] を対象とした．FreeBSD 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．以下順に推定方法を示す．

2.1 推 定 方 法

再送されたパケットの検知

各パケット毎にパケットのシーケンス番号と今までモニタしたシーケンス番号の中で最も大きなシーケンス番号（以下，最大シーケンス番号）を比較し，パケットのシーケンス番号の方が大きければ最大シーケンス番号を上書きする．

```
max_seq: 現在までに送出されたパケットのシーケンス番号の最大値
seq: 現在のパケットのシーケンス番号

if(max_seq < seq)
        max_seq = seq;
```

再送パケットは送出したはずのパケットを再度送出するので，最大シーケンス番号よりも小さいシーケンス番号のパケットが送出される．従って，最大シーケンス番号よりも小さいシーケンス番号のパケットは再送されたパケットだということがわかる．

```
if(max_seq < seq)
        /* new packet */
else
        /* retransmit packet */
```

2.1.1 Duplicate ACK

Duplicate ACK をトリガとして再送するアルゴリズムは NewReno で実装された．基本的には Reno と同じであるが，New Reno の Fast Recovery は Duplicate ACK を 3 回以上受信した時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留める．FreeBSD 8.0 の TCP 実装が RFC 2582 を参照しているので，Duplicate ACK を 3 回以上受信した時，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+duplicate_ack_count*MSS に設定する．ただし，ssthresh は最低でも 2MSS である．

```
duplicate_ack_count: 現在の Duplicate ACK の数
ssthresh: 現在の ssthresh
cwnd: 現在の cwnd
win: window サイズ
min(value1,  value2): value1,  value2 を比べ小さい方を return する関数
MSS: Max Segment Size

if(duplicate_ack_count > 3){
        ssthresh = min(cwnd,  win) / 2;
        if(ssthresh < 2)
                ssthresh = 2;
        ssthresh = ssthresh * MSS;
        cwnd = ssthresh + duplicate_ack_count * MSS;
}
```

以降，最大シーケンス番号以上の ACK を受信した場合は cwnd を ssthresh に設定し，再送を終了する．それ以外は partial ACK として以下の処理を行う．cwnd の値を ocwnd という変数に代入する．cwnd の値が，PACK の値から una の値を引いた値より大きい場合，cwnd から PACK の値から una の値を引いた値を引く．それ以外の場合，cwnd は 0 となる．最後に，cwnd に 1MSS を足した値が現在の cwnd となる．

```
cwnd: 現在の cwnd.
PACK : partial ACK のトリガとなった ACK の要求するシーケンス番号.
una : 今までで送信が確認されているシーケンス番号の最大値.

if (cwnd > PACK - una)
        cwnd -= PACK - una;
else
        cwnd = 0;
cwnd += MSS;
```

partial ACK による再送が終了すると，ocwnd の値が PACK の値から最大シーケンス番号の値を引いた値より大きい場合，ocwnd の値から PACK の値を引き，una の値を足す．それ以外の場合は 0 になる．そして，ocwnd に 1MSS を足し，現在の cwnd に設定する．

```
if(ocwnd > PACK - una)
        ocwnd -= PACK - una;
else
        ocwnd = 0;
ocwnd += MSS;
cwnd = ocwnd;
```

2.1.2 タイムアウト

タイムアウトは RTT が RTO を越えた場合，パケットがロスしたし，対象のパケットを再送する．tcp_input.c[5] の中の tcp_xmit_timer 関数と同様の処理を行い RTO を推定した．今回は各パケット毎に RTO を算出するようにした．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 1MSS，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 2MSS である．

(2) 最大シーケンス番号より大きな ACK が来るまで最大シーケンスまで再送を続ける．

```
if(max_seq > seq){
        cwnd = MSS;
        if(ssthresh < 2)
                ssthresh = 2;
        ssthresh = ssthresh * MSS;
}
```

2.1.3 SACK

SACK が用いられているかどうかは，TCP の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 は再送動作を行わないので，SACK ブロックに格納されているシーケンス番号で抜けているシーケンス番号のパケットが再送されているかだけを検査する．cwnd と ssthresh の推定値には影響を与えない．

2.1.4 On Estimating End-to-End Network Path Properties

On Estimating End-to-End Network Path Properties は再送が 1 回で終了した場合，直前の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．今回は再送が発生した場合，再送時の 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 を以下のように設定するよう実装した．

$cwnd =$ バックアップしておいた $cwnd + MSS *$ ACK 数

2.2 推定方法の検証

推定方法が正しいかを確認する為にリアルタイムでの推定の前に pcap 形式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[6] を作成し，推定値と TCP 実装の値の比較を行った．検証用ツールでの推定した cwnd と TCP 実装の cwnd との比較を**図 2** の A に，推定した ssthresh と TCP 実装の ssthresh との比較を図 2 の B に示す．図 2 を見るとほぼ推定値と TCP 実装の値が一致していると言える．誤差が発生した原因は，RTO を各パケット毎に計算することで誤差が発生し，タイムアウトと Duplicate ACK の判定を誤った為であり，タイムアウトと Duplicate ACK の判定を誤らなければ上記の推定方法で正確な推定値の算出が可能であることを確認した．

2.3 リアルタイムでの推定手法

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．また，リアルタイムで推定するため，例えば前のパケットに戻って計算をやり直すことは困難である．そこで，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，2 つの推定値を保持する．複数の推定値を保存する為，データは木構造化して保存するようにした．ただし，再送が発生しない限り推定誤りは起こらないので，再送が発生するまではパケットの情報を保存せず，次の輻輳パラメータの計算に必要な直前のパケットの情報のみを保持する．再送が初めて発生してからパケットの情報を累積して保持する． 再送が発生した場合の木構造の構造についての例を**図 3** に示す．図 3 の色付きの 3，4，9，10，11，12 は輻輳が発生し，パケットが再送されている部分である．しかし，複数の推定値の中でどの推定値が正しいのか分からない．また，再送が大量に発生した場合にメモリ不足になる可能性があるため，以下のような処理を行う．

(1) 確実に間違っている推定値を削除する．

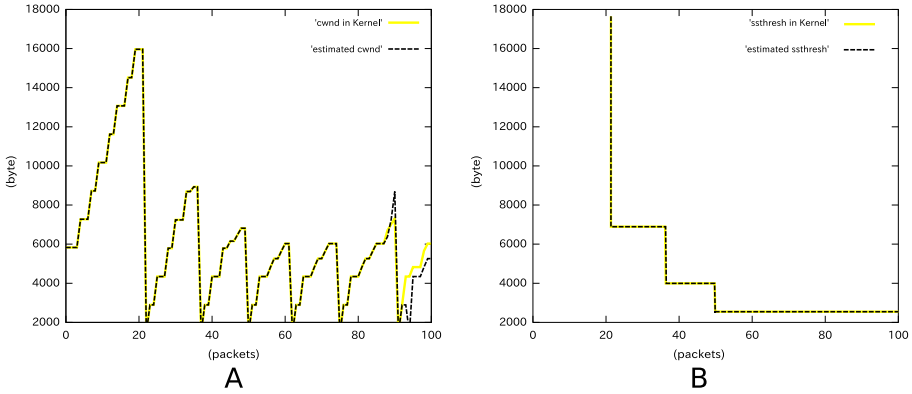

図 2 検証用ツールでの推定値と TCP 実装の値
Fig. 2 Congestion parameters estimated by the prototype tool and congestion parameters inside the FreeBSD kernel

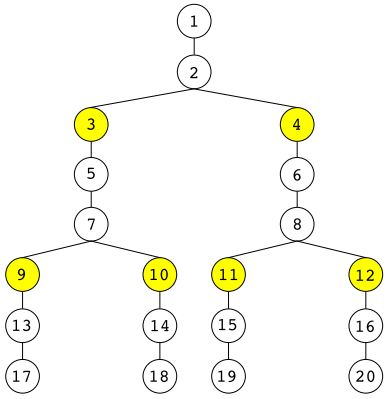

図 3 複数の値を保持する為の木構造の例
Fig. 3 Tree structure to store estimated values

各パケット毎に連続して送出されたデータ量とその時点での推定したそれぞれの cwnd を比較し，連続して送出されたデータ量よりその時点での cwnd が小さい場合，推定した cwnd は間違っているとして削除する．

(2) 間違っている推定値を削除しても複数の推定値が残っている場合．

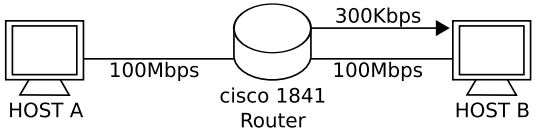

図 5 実験方法
Fig. 5 Study Procedure

現在の連続送出されたデータ量に最も近い cwnd の値を持つ推定値を表示に使用する．しかし，この場合でも cwnd の値がまったく同じ推定値が複数存在する可能性がある．この場合は最も古い枝の推定値を表示に使用する．

2.4 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 に実装した推定機能は次のように動作する．SYN を検知すると，コネクションの情報として 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， wscale を設定する．この時，source の IP アドレスとポート番号には SYN パケットの source，destination の IP アドレスとポート番号には SYN パケットの destination の IP アドレスとポート番号を設定する．パケットの情報はコネクション情報の中にホスト別の木構造リストに時間軸に沿って保存する．

cwnd と ssthresh の推定値は元の 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:cwnd_value ssthresh:ssthresh_value>`，タイムアウトによる再送のパケットの場合，`<ret cwnd:cwnd_value ssthresh:ssthresh_value>` を，Duplicate ACK による再送の場合，`<dup cwnd:cwnd_value ssthresh:ssthresh_value>` を行末に追加する．**図 4** に実際の輻輳パラメータの表示結果を示す．

3. 実験と評価

輻輳制御パラメータの推定値と TCP 実装の値の比較を行い，推定が正しく行われているかを評価した． 実験環境を**図 5** に示す．実験には FreeBSD 端末 2 台を用いる．TCP 実装の cwnd と ssthresh を取得する為，通信にはデータ部分を改竄しても影響がない FTP を用い，TCP のデータ部分に cwnd と ssthresh を挿入し，TCP 実装の cwnd と ssthresh を取得できるようにした．解析対象の TCP は SACK を用い， タイムスタンプ， ウインドウスケールオプションを有効にした FTP 通信を使用する．A-B 間に設置したシスコ社製ルータ

```
17:04:31.398906 IP 160.194.128.8.40183 > 10.0.0.3.6666: S 1283051085:1283051085(0) win 65535 <mss 8960,nop,wscale 3,sackOK,timestamp 173933 0>
17:04:31.399581 IP 10.0.0.3.6666 > 160.194.128.8.40183: S 1874944744:1874944744(0) ack 1283051086 win 65535 <mss 1460,nop,wscale 3,sackOK,timestamp 3898129189 173933>
17:04:31.399605 IP 160.194.128.8.40183 > 10.0.0.3.6666: . ack 1 win 8326 <nop,nop,timestamp 173934 3898129189> <cwnd:1448 ssthresh:1073725440
17:04:31.406264 IP 160.194.128.8.40183 > 10.0.0.3.6666: P 1:1025(1024) ack 1 win 8326 <nop,nop,timestamp 173940 3898129189> <cwnd:1448 ssthresh:1073725440>
17:04:31.506618 IP 10.0.0.3.6666 > 160.194.128.8.40183: . ack 1025 win 8326 <nop,nop,timestamp 3898129296 173940>
17:04:31.506675 IP 160.194.128.8.40183 > 10.0.0.3.6666: . 1025:2473(1448) ack 1 win 8326 <nop,nop,timestamp 174041 3898129296> <cwnd:2896 ssthresh:1073725440>
17:04:31.506685 IP 160.194.128.8.40183 > 10.0.0.3.6666: . 2473:3921(1448) ack 1 win 8326 <nop,nop,timestamp 174041 3898129296> <cwnd:2896 ssthresh:1073725440>
17:04:31.507643 IP 10.0.0.3.6666 > 160.194.128.8.40183: . ack 3921 win 8145 <nop,nop,timestamp 3898129297 174041>
17:04:31.507664 IP 160.194.128.8.40183 > 10.0.0.3.6666: . 3921:5369(1448) ack 1 win 8326 <nop,nop,timestamp 174042 3898129297> <cwnd:4344 ssthresh:1073725440>
17:04:31.507673 IP 160.194.128.8.40183 > 10.0.0.3.6666: . 5369:6817(1448) ack 1 win 8326 <nop,nop,timestamp 174042 3898129297> <cwnd:4344 ssthresh:1073725440>
17:04:31.507682 IP 160.194.128.8.40183 > 10.0.0.3.6666: . 6817:8265(1448) ack 1 win 8326 <nop,nop,timestamp 174042 3898129297> <cwnd:4344 ssthresh:1073725440>
17:04:31.508500 IP 10.0.0.3.6666 > 160.194.128.8.40183: . ack 5369 win 8326 <nop,nop,timestamp 3898129298 174042>
17:04:31.508560 IP 160.194.128.8.40183 > 10.0.0.3.6666: . 8265:9713(1448) ack 1 win 8326 <nop,nop,timestamp 174043 3898129298> <cwnd:5792 ssthresh:1073725440>
17:04:31.508570 IP 160.194.128.8.40183 > 10.0.0.3.6666: . 9713:11161(1448) ack 1 win 8326 <nop,nop,timestamp 174043 3898129298> <cwnd:5792 ssthresh:107372544>
17:04:31.608560 IP 10.0.0.3.6666 > 160.194.128.8.40183: . ack 6817 win 8326 <nop,nop,timestamp 3898129398 174042>
17:04:31.608619 IP 160.194.128.8.40183 > 10.0.0.3.6666: . 11161:12609(1448) ack 1 win 8326 <nop,nop,timestamp 174143 3898129398> <cwnd:7240 ssthresh:1073725440>
17:04:31.608629 IP 160.194.128.8.40183 > 10.0.0.3.6666: . 12609:14057(1448) ack 1 win 8326 <nop,nop,timestamp 174143 3898129398> <cwnd:7240 ssthresh:1073725440>
17:04:31.609585 IP 10.0.0.3.6666 > 160.194.128.8.40183: . ack 6817 win 8326 <nop,nop,timestamp 3898129399 174042,nop,nop,sack 1 11161:12609>
17:04:31.980517 IP 160.194.128.8.40183 > 10.0.0.3.6666: . 6817:8265(1448) ack 1 win 8326 <nop,nop,timestamp 174515 3898129399> <ret cwnd:1448 ssthresh:2896>
17:04:31.981469 IP 10.0.0.3.6666 > 160.194.128.8.40183: . ack 8265 win 8145 <nop,nop,timestamp 3898129771 174515,nop,nop,sack 1 11161:12609>
```

図 4 輻輳パラメータの表示
Fig. 4 Display of congestion parameters

1841 の機能を使って，ルータから B への帯域を 300Kbps に制限する．A から B へデータを送信し，A で推定機能を追加した tcpdump を用いて推定を行った．

推定した cwnd と TCP 実装の cwnd の比較を**図 6** に，推定した ssthresh と TCP 実装の ssthresh の比較を**図 7** に示す．図 6，図 7 を見るとほぼ推定値と TCP 実装の値が一致しているのがわかる．一致していない箇所は，複数の推定値を保持している場合に表示に使用する値の選択が正しく行われていないことによるものである．

4. まとめと今後の課題

今回は輻輳制御パラメータの推定機能を tcpdump に実装し，リアルタイム推定でも，推定値と TCP 実装の値がほぼ一致することを確認した．今後の課題として，その他オプションへの対応や実際の通信での実験，複数の推定値を保持した場合に使用されるメモリ量の把握，複数の推定値を保持している場合にどの推定値を表示に使用するかを選択する方法の改善などが今後の課題となる．

参 考 文 献

1) Jacobson, V.: Congestion Avoidance and Control, *Computer Communication Review*, Vol.18, No.4, pp.314–329 (1988).
2) Floyd, S. and Henderson, T.: The NewReno Modification to TCP's Fast Recovery Algorithm, RFC 2582 (Experimental) (1999).
3) Floyd, S., Mahdavi, J., Mathis, M. and Podolsky, M.: An Extension to the Selective Acknowledgement (SACK) Option for TCP, RFC 2883 (Proposed Standard) (2000).
4) M.Allman and V.Paxson: On Estimating End-to-End Network Path Properties, *ACM SIGCOMM* (2001).
5) FreeBSD Foundation: *tcp_input.c*, v 1.397 edition (2009). 1/15 06:44:22.
6) 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).

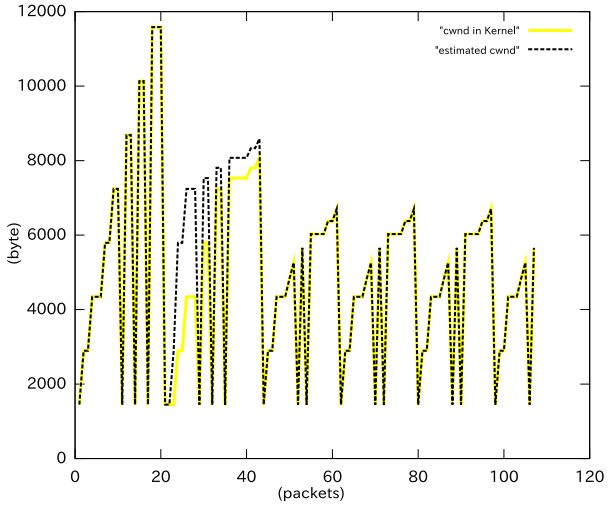

図 6 cwnd の推定値と TCP 実装の cwnd
Fig. 6 Estimated cwnd and cwnd inside the FreeBSD kernel

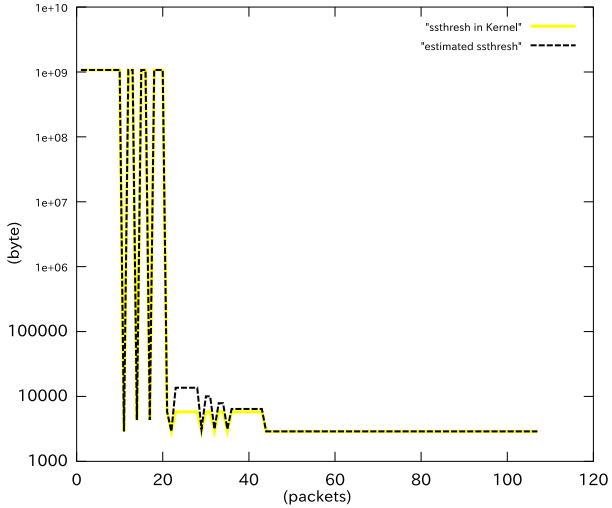

図 7 ssthresh の推定値と TCP 実装の ssthresh
Fig. 7 Estimated ssthresh and ssthresh inside the FreeBSD kernel