RL-006

パケット優先廃棄による TCP 公平性の改善
Fairness Improvement among Modern TCPs with Packet Dropping

秋山 友理愛† 神津 智樹† 山口 実靖†
Yuria Akiyama Tomoki Kozu Saneyasu Yamaguchi

1. はじめに

TCP は現在のインターネットにおいて標準的に用いられているトランスポート層プロトコルであり，その実装にはネットワークの輻輳制御機能が組み込まれている．従来の TCP 実装では輻輳制御アルゴリズムとして TCP-Reno が使用されてきた．しかし TCP-Reno では広帯域・高遅延なネットワーク環境においてネットワーク帯域を十分に使いきることができない問題が指摘されており[1][2][3]，BIC TCP[4]，CUBIC TCP[5]，Compound TCP[6]などの TCP-Reno に代わる新しい TCP(本稿ではこれらを“高速 TCP”と呼ぶ)が数多く提案されている．これらの高速 TCP の提案により，TCP アルゴリズム間の帯域公平性という新たな課題が生じ，これまでにシミュレーションに基づく公平性の研究が行われてきている[7][8]．しかし，実際の OS に搭載されている TCP 実装を用いての研究は十分にはされておらず，実 OS の実 TCP 実装を用いての評価や考察も行う必要があると考えられる．

本稿では，高速 TCP の中の代表的なロスベースの輻輳制御アルゴリズムである CUBIC TCP と，代表的なハイブリッド型輻輳制御アルゴリズムである Compound TCP を対象に，実 TCP 実装と実機ネットワークを用いて性能と公平性の評価を行い，公平性が低いことを示す．そして，公平性の向上手法を提案し，提案手法を実装したネットワーク装置と実 TCP 実装，実機ネットワークを用いて評価を行う．

2. 関連研究

2.1. 輻輳制御アルゴリズム

過剰なパケットを送出しネットワークの輻輳を招くことを避けるために，TCP 実装には輻輳制御アルゴリズムが搭載されている．TCP によるパケット送出量はこの輻輳制御アルゴリズムにより制限されており，TCP の性能はこのアルゴリズムに大きく依存している．

TCP の輻輳制御手法は主に，ロスベース手法，遅延ベース手法，両者を組み合わせたハイブリッド型の手法に分類することができる．

ロスベース手法はパケットロスの検出に基づき輻輳ウィンドウを制御する手法である．通常時は確認応答を受信するたびに輻輳ウィンドウを増加させ，パケットロス検出時に輻輳ウィンドウを大幅に減少させる．従来の TCP である TCP-Reno がロスベースの手法であり，代表的なロスベースの高速 TCP に BIC TCP[4]や，CUBIC-TCP[5]がある．

遅延ベースの輻輳制御アルゴリズムは，RTT の増減にあわせて輻輳ウィンドウを変化させる手法である．ロスベース手法の様に輻輳が発生してから速度を減少させるのではなく，RTT の増加からネットワークの混雑状態を推定し，輻輳が発生する前に速度を減少させるため安定した通信速度が期待できる．欠点としてロスベース手法とネットワークを共有したときに得られる性能が低くなってしまうということが指摘されている[9]．代表的な遅延ベースの手法に TCP-Vegas[10]がある．

ハイブリッド型手法は両者を組み合わせた手法であり，代表的なハイブリッド型手法の TCP に Compound-TCP(CTCP)[6]がある．

2.2. CUBIC TCP

CUBIC TCP は，BIC TCP のスケーラビリティを維持しながら，TCP Fairness(既存の TCP アルゴリズムとの公平性)，RTT Fairness(RTT の異なる通信間での公平性)，制御手法の複雑さを改善した高速 TCP である[5].

CUBICTCP では，BIC TCP のバイナリサーチを用いて利用可能帯域を探索するアルゴリズムを，下記の式のような 3 次関数を用いた制御によって実現している(図 1 参照).

$$cwnd = C(t-K)^3 + W_{\max}$$

$$K = \sqrt[3]{\frac{W_{\max}\beta}{C}}$$

ここで，*cwnd* は輻輳ウィンドウサイズ，*t* はパケットロス検出時からの経過時間(実時間)，W_{max} はパケットロス検出時の輻輳ウィンドウサイズ，*C* は増加幅を決めるパラメータ，β はパケットロス検出時のウィンドウサイズ減少幅を表している．一般に *C* は 0.4，β には 0.2 が用いられる．

上記のように，ウィンドウサイズをパケットロス検出時からの経過時間により決めており，RTT の影響を排している．これにより RTT Fairness を向上するとともに，BIC TCP の低遅延環境で輻輳ウィンドウサイズを急速に成長させすぎる問題を解決している．また，TCP Reno を用いた場合に得られるウィンドウサイズを次式により計算し，現在のウィンドウサイズが計算値よりも小さい場合はその計算値をウィンドウサイズとして採用することとしている．

$$cwnd = W_{\max}(1-\beta) + 3\frac{\beta}{2-\beta}\frac{t}{RTT}$$

さらに，新規にネットワークに参加したフローにも公平に帯域を与えるために以下の Fast Convergence が採用されている．前回のパケットロス時のウィンドウサイズより今回のパケットロス時のウィンドウサイズが小さい場合は，W_{max} を次式により決定する．これにより既存のフローの W_{max} が減少し，新規に参加したフローがより多くの帯域を得られる仕組みとなっている．

$$W_{\max} = cwnd \times (2-\beta)/2$$

†工学院大学 工学研究科 電気・電子工学専攻
Electrical Engineering and Electronics, Kogakuin University Graduate School

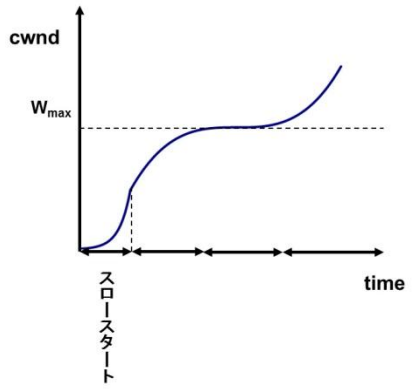

図 1　CUBIC TCP の輻輳ウィンドウサイズの推移

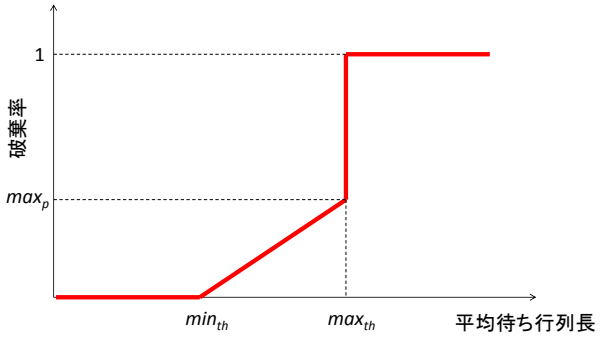

図 2　RED における平均待ち行列長とパケット破棄率の関係

2.3. Compound TCP

Compound TCP はロスベースの輻輳制御で動作するロスウィンドウと，遅延ベースの輻輳制御で動作する遅延ウィンドウの両方を用いて，ネットワークに送出するパケット数を調節する手法である．

ロスウィンドウは，スロースタートフェーズと輻輳回避フェーズの 2 つのフェーズから構成されている．それぞれのフェーズによってロスウィンドウの増加量が異なり，各フェーズでのウィンドウサイズは，次式で与えられる．これは TCP-Reno と同等である．

$$cwnd(t+1) \leftarrow \begin{cases} cwnd(t)+1 & (\text{スロースタートフェーズ時}) \\ cwnd(t)+\dfrac{1}{swnd(t)} & (\text{輻輳回避フェーズ時}) \end{cases}$$

ただし，*swnd* は現在のロスウィンドウと遅延ウィンドウの和であり，*t* は ACK を受信するごとに増加する値であり実時間とは比例しない．スロースタートフェーズでは，ロスウィンドウは指数関数的に増加し，輻輳回避フェーズでは線形的にロスウィンドウが増加することになる．

パケットロスを検出した場合にはロスウィンドウが減少するが，減少量はパケットのロスの検出方法によって異なる．減少後のロスウィンドウは次式で与えられる．

$$cwnd(t+1) \leftarrow \begin{cases} \dfrac{cwnd(t)}{2} & (\text{重複}ACK\text{による検出}) \\ 1 & (\text{タイムアウトによる検出}) \end{cases}$$

重複 ACK を受信した場合のパケットロスは，ネットワークに軽度の輻輳が発生したと判断してロスウィンドウを現在の値の半分に減少させる．一方，タイムアウトによるパケット棄却を検出した場合は，ネットワークに重度の輻輳が発生したと判断してロスウィンドウを 1 に減少させる．

もう一方のウィンドウである遅延ウィンドウもスロースタートフェーズと輻輳回避フェーズにより動作が異なる．遅延ウィンドウは，スロースタートフェーズ時においては動作せず，輻輳回避フェーズ時のみ動作する．

遅延ウィンドウの決定過程では，まず，以下の *Diff* を用いてネットワークの混雑状況を推定する．

$$Expected = \frac{swnd(t)}{baseRTT}$$

$$Actual = \frac{swnd(t)}{RTT}$$

$$Diff = (Expected - Actual) \cdot baseRTT$$

baseRTT は観測された往復遅延時間の最小値，*RTT* は現在の往復遅延時間である．

Expected は理想的状態(ネットワークが最も空いている状態)において得られる理想的な通信速度であり，*Actual* は現在の RTT にて実際に得られると予想される通信速度である．*Diff* はネットワーク内に滞留しているパケット数である．

遅延ウィンドウサイズは以下の式により決定される．

$$dwnd(t+1) = \begin{cases} dwnd(t) + (\alpha \cdot swnd(t)^k - 1) & (Diff < \gamma) \\ (dwnd(t) - \zeta \cdot Diff) & (Diff \geq \gamma) \end{cases}$$

推測値 *Diff* が閾値 γ よりも小さい場合は，ネットワークに未使用帯域があると判断し，遅延ウィンドウを増加させる．逆に，推測値 *Diff* よりも閾値 γ が大きい場合は，ネットワークに輻輳が発生していると判断して，遅延ウィンドウを減少させる．

CTCP の最終的な送出ウィンドウはロスウィンドウおよび遅延ウィンドウを用いて次式に従い決定される．

$$swnd(t+1) = cwnd(t+1) + dwnd(t+1)$$

2.4. RED(Random Early Detection)

RED[11]は平均待ち行列長に応じた確率でパケットの廃棄を行う方法である．Tail Drop では，待ち行列長がバッファサイズに達すると全てのコネクションのパケットが廃棄され，バースト的なパケット廃棄が行われるが，RED では安定した輻輳制御が行え，公平性の改善も実現されると期待される．

RED では，図 2 の様に平均待ち行列長によりパケット破棄率を決定し，その確率によりパケットが廃棄される．min_{th}, max_{th}, max_p は制御用パラメータである．平均待ち行列長は w_q を用いて以下に式により決定される．

$$\bar{q} = (1 - w_q)\bar{q} + W_q q$$

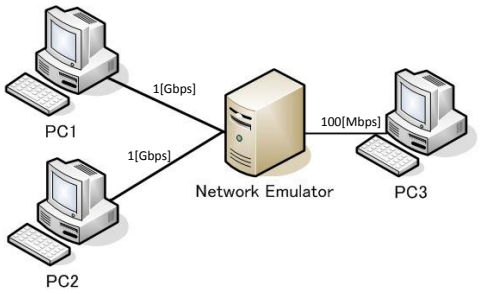

図 3　ネットワーク構成

表 1　計算機仕様

CPU	Intel Celeron G530, 2.4[GHz]
メモリ	2[GB]
OS	(PC1) Linux 2.6.35.6, (PC2, PC3) Windows 7 Enterprise, (Network Emulator) FreeBSD 8.0

2.5. TCP 公平性向上の研究

TCP の公平性向上に関して以下の様な研究が行われている．

逸身らはルータ内のキュー長を観測し，キュー長が大きく変動した際にパケットを廃棄することで公平性を改善させる手法を提案している[7]．また，長谷川らは RED を用いる公平性改善の研究として，バックボーンルータにおける RED の動的閾値制御方式を提案している[8]．当該研究では，RED のパラメータを輻輳の状況に応じて動的に変化させている．

これらの研究はいずれもシミュレーションによるものである．よって，これらの研究に加え，実際の OS に搭載された実 TCP 実装を用いた検証も重要であると考えられる．

3. 実機と実 OS を用いた公平性評価

本章では，実ネットワーク上で異なる高速 TCP が混在する環境における各 TCP の通信性能と公平性についての評価を行う．

図 3 のネットワークを構築し，CTCP と CUBIC TCP が混在する環境における通信速度を netperf を用いて測定した．PC1，PC2，PC3，ネットワークエミュレータの仕様は表 1 の通りである．ネットワークエミュレータは人工的にネットワーク遅延時間やパケットロスを発生させる装置であり，FreeBSD Dummynet を用いて構築した．ネットワーク機器は全て 1Gigabit Ethernet 対応のものであり，ネットワークエミュレータと PC3 の間の通信速度はエミュレータにより 100Mbps に設定されている．

まず，非混在環境(単独通信環境)における性能について述べる．CUBIC TCP の単独通信の性能と，Compound TCP の単独通信の性能を図 4 に示す．Tail Drop は待ち行列バッファの空きがある間はパケットを破棄せず，空がない状態では全ての到着パケットを破棄する最も単純な手法

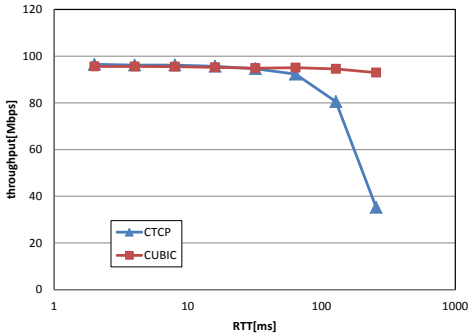

図 4　単独通信性能(Tail Drop)

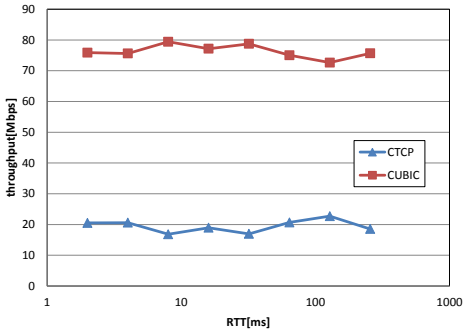

図 5　同時通信性能(Tail Drop)

である．横軸の値はネットワークエミュレータにより設定した往復遅延時間である．単独通信の実験では TCP コネクションは 1 本のみ確立され，CUBIC TCP 単独通信の例においては PC1 と PC3 の間に CUBIC TCP のコネクションが 1 本確立された．図より，RTT が 100ms 以上の例をのぞき，単独通信環境では CUBIC TCP の性能と，Compound TCP の性能はほぼ同等であることが分かる．

次に，同時通信時の性能について述べる．図 3 の PC1-PC3 の間と，PC2-PC3 の間に同時に netperf の接続を確立し，それぞれの通信速度を測定した．PC1 では CUBIC-TCP が，PC2 では CTCP が動作している．両コネクションの通信は同時に行われ，両者はネットワークエミュレータから PC3 までのネットワーク機器を共有している．実験結果を図 5 に示す．同様に横軸は往復遅延時間である．CUBIC-TCP と CTCP が同時に通信を行う環境においては，CUBIC-TCP の通信性能が CTCP の通信性能を大きく上回り，公平性が低いことが分かる．

4. 提案手法

2.5 節で述べたように，ネットワーク上のルータにおいて RED を用いることにより TCP アルゴリズム間の公平性を改善できると期待できる．本章にて，ネットワーク利用率の高いコネクションのパケットを優先的に破棄し，既存の RED より高い精度の公平性を目指す手法を 2 つ(静的優先破棄手法と動的優先破棄手法)提案する．

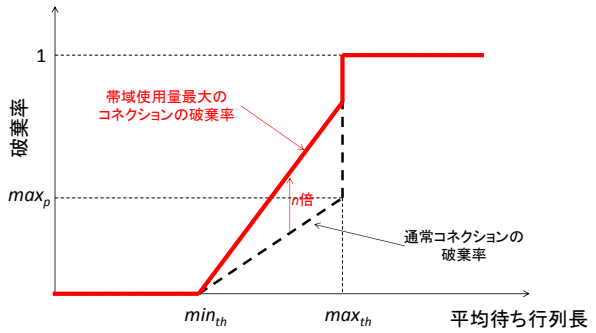

図 6　提案手法

4.1. 静的優先破棄

静的優先破棄手法では，通信帯域を最も多く使っている端末がルータにとって既知であるという前提のもと，図 6 の様にその端末によるコネクションの RED における パケットの破棄率を *n* 倍にする．

4.2. 動的優先破棄

動的優先破棄手法では，通信帯域を多く使っている端末はルータにとって既知でないという前提のもと，ルータが最も通信帯域を消費しているコネクションの推測を行い，そのコネクションのパケットの破棄率を *n* 倍にして優先的に破棄する．

最も通信帯域を消費しているコネクションの推測は，以下の様に行う．ルータに到着するパケットの数が最も多いコネクションを，最も通信帯域の消費が大きいコネクションと仮定する．ルータにて記録周期 *rec_int* 個のパケット毎に到着パケットの TCP コネクション情報を記録し，最新の *hist_len* 個のコネクション情報を履歴としてルータ内のメモリに保持する．そして，集計周期 *sta_int* 個のパケット毎に履歴中に最も多く登場するコネクションを抽出し，それを最も通信帯域を消費しているコネクションとする．

コネクション情報としては，送信元 IP アドレス，送信先 IP アドレス，送信元ポート番号，送信先ポート番号を保持する．

rec_int を 2 以上に指定することにより，特定のコネクションのパケットが短期間に連続して到着した場合などに，短期的な情報のみから通信帯域の消費が多いコネクションを決定することを避けることができると期待される．また，*rec_int* や *sta_int* が小さく *hist_len* が大きいほど，処理オーバーヘッドが増え，正確な推測が可能となると考えられる．

5. 評価

提案手法の有効性を検証するために，提案手法を実装し性能評価を行った．

5.1. 実験環境

3 章の実験同様に，図 3 の実験環境にて PC1-PC3 間と，PC2-PC3 間で netperf のコネクションを同時に確立し，通信速度を計測した．PC1-PC3 の通信は CUBIC TCP により行われ，PC2-PC3 間の通信は Compound TCP により行われた．パケット廃棄はネットワークエミュレータにて行われ，通常のパケット廃棄(Tail Drop)，RED，提案手法(静的優先廃棄)，提案手法(動的優先廃棄)により行った．全ての手法において，最大待ち行列長は 10,000 とした．

RED においては，min_{th}=1，max_{th}=9999，max_p=0.2，w_q=0.002 としてパケット破棄を行った．

提案手法(静的優先破棄)においては，*n*(パケット破棄率の倍率)を 10 とし，PC1-PC3 間に確立された netperf のコネクションを最も通信帯域の消費が大きいコネクションとして静的に登録した．また min_{th}=2500，max_{th}=9999，max_p=0.5，w_q=0.002 とした．

提案手法(動的優先破棄)においては，記録周期 *rec_int* を 10[パケット]，集計周期 *sta_int* を 100[パケット]とし，min_{th}=1，max_{th}=9999, max_p=0.2，w_q=0.002 とした．

5.2. 実験結果

同時通信環境における Tail Drop，RED，提案手法(静的優先破棄)，提案手法(動的優先廃棄)の CTCP と CUBIC TCP の通信速度を図 7, 図 8, 図 9 に示す．通常のパケット破棄(Tail Drop)における通信速度は前述の図 5 の通りである．また，全手法における CTCP と CUBIC TCP のスループットの比率を図 10 に，CTCP と CUBIC TCP のスループットの合計を図 11 に示す．図 10 の値は 1 に近いほど好ましい．図 11 の値は高いほど良く，最大で 100Mbps である．

結果より，RED により得られる公平性は，Tail Drop と比較しての改善は見られるものの，必ずしも高くないことが分かる．よって，TCP アルゴリズム間の帯域不公平の問題は，単純に RED を適用したのみでは解決されず，新しい手法の提案が必要であると考えられる．

両提案手法の公平性に着目すると，Tail Drop や RED の公平性より優れていることが分かり，提案手法が有効であることを確認できる．特に往復遅延時間が 32ms 以下の状況ではスループットの比率が 0.8 倍から 1.2 倍程度となっており，大幅な改善が実現されていることが分かる．

静的優先手法と動的優先手法の公平性を比較すると，静的優先手法は往復遅延時間に依らず安定してスループットの比率が 1 に近いが，動的優先手法は往復遅延時間の変化によりスループットの比率が変化し，往復遅延時間が大きいとスループットの比率が 0.6 程度まで悪化していることが分かる．これより，最大帯域消費コネクションの推定は遅延時間が大きい場合に正確さが低下していると考えられる．

次に，合計通信速度について考察する．往復遅延時間が小さい(10ms 未満)環境では，いずれの手法を用いてもほとんど合計通信速度の変化がなく，提案手法はスループットの低下なく公平性の向上を実現していると言うことができる．往復遅延時間が大きい(10ms 以上)環境において比較を行うと，提案手法(静的優先破棄)の合計性能は RED の合計性能よりも高く，帯域消費が大きいコネクションの推測を正確に行えれば提案手法は RED よりも高い公平性と高い性能の両方を実現できることが分かる．ただし，必要最低限のパケット廃棄しか行わない Tail Drop と積極的にパケット廃棄を行う RED や提案手法を比較すると，Tail Drop の方が合計性能が高く，提案手法にはさらなる改善が望まれると言える．

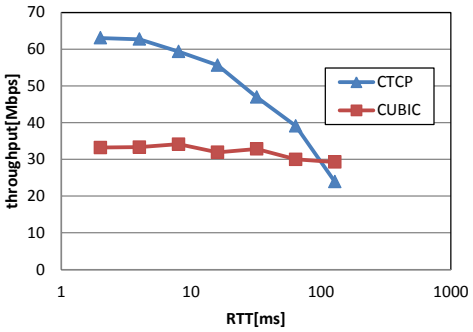

図 7　同時通信性能（RED）

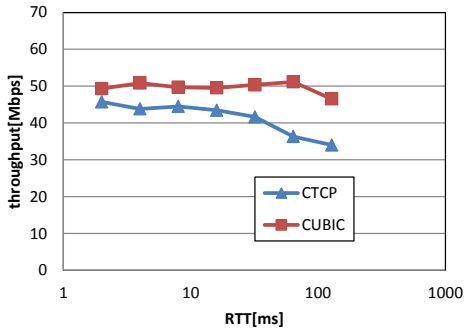

図 8　同時通信性能(静的優先破棄)

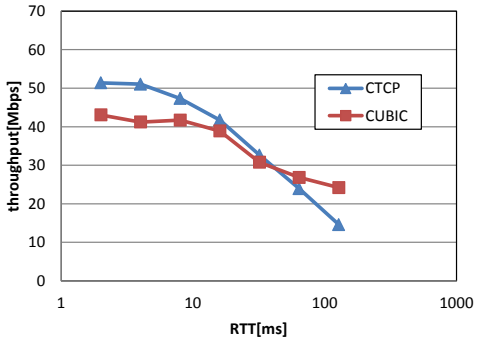

図 9　同時通信性能(動的優先破棄)

5.3. 複数コネクション環境における評価実験

PC1-PC2 間および PC1-PC3 間に確立する netperf のコネクションを 16 本ずつに増やし，同時通信時の性能評価実験を行った．5.1 節の実験同様に，図 3 の実験環境にて PC1-PC3 間と，PC2-PC3 間で netperf のコネクションを同時に 16 本ずつ確立し，通信速度を計測した．PC1-PC3 の通信は CUBIC TCP により行われ，PC2-PC3 間の通信は Compound TCP により行われた．5.1 節と同様に，パケット廃棄はネットワークエミュレータにて行った．全ての手法において，最大待ち行列長は 10,000 とした．

RED においては，min_{th}=1，max_{th}=9999，max_p=0.2，w_q=0.002 としてパケット破棄を行った．

提案手法(静的優先破棄)においては，n(パケット破棄率の倍率)を 2 とし，PC1 から PC3 に確立された netperf のコネクションを最も通信帯域の消費が大きいコネクション

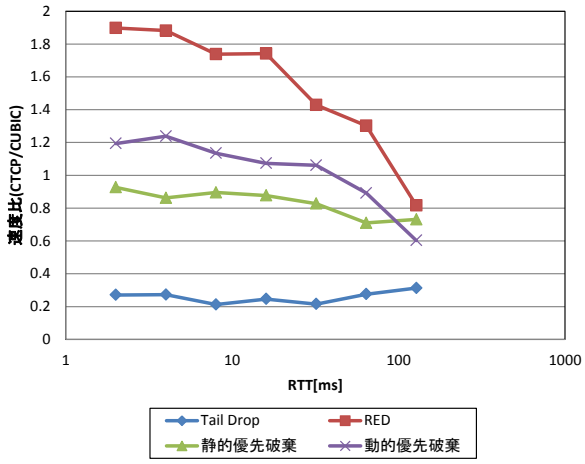

図 10　通信速度比

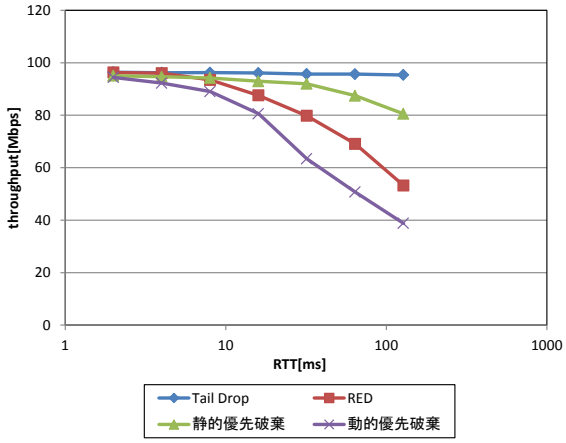

図 11　合計通信速度

として静的に登録した．また min_{th}=4000，max_{th}=9999，max_p=0.15，w_q=0.002 とした．

提案手法(動的優先破棄)においては，n(パケット破棄率の倍率)を 10，記録周期 rec_int を 10[パケット]，集計周期 sta_int を 100[パケット]とし，min_{th}=1，max_{th}=9999，max_p=0.2，w_q=0.002 とした．

複数コネクション環境での通常のパケット破棄(Tail Drop)，RED，提案手法(静的優先破棄)，提案手法(動的優先破棄)の同時通信性能を図 12，図 13 に示す．図 12 はそれぞれ 16 本ずつある CUBIC TCP と CTCP のコネクションの平均通信速度の比率であり，1 に近いほど好ましい．図 13 は全てのコネクションの通信速度の合計であり，最大で 100[Mbps]である．

図 12 より，複数コネクション環境においても両提案手法(静的優先破棄，動的優先破棄)を用いることにより Tail Drop，RED よりも公平性が向上することが確認できる．また，図 13 より，いずれの手法でも合計速度の変化はほとんど見られず，提案手法はスループットの低下なく公平性の向上を実現していると言うことができる．

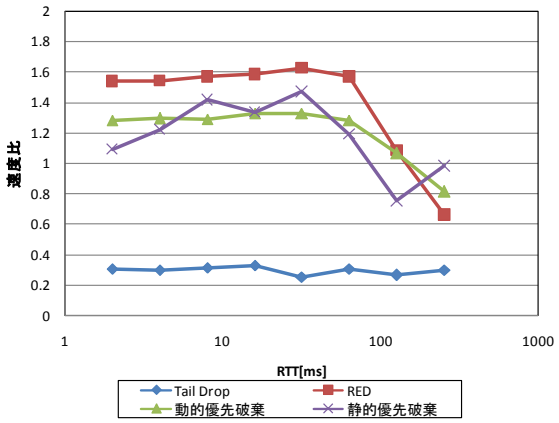

図 12　複数コネクション環境での通信速度比

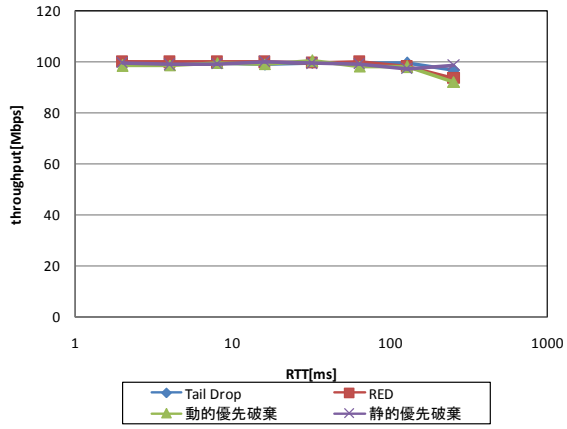

図 13　複数コネクション環境での合計通信速度

6. おわりに

本稿では，高速 TCP アルゴリズム間の帯域公平性に着目し，実 OS に搭載されている実 TCP 実装と実ネットワーク機器を用いて帯域公平性の評価を行った．そして，帯域公平性の改善手法を提案し，それを実機上に実装し，実 TCP 実装を用いた評価実験により有効性を示した．

今後は，実ネットワークを用いての評価，多数コネクション環境(100 以上)での評価，CUBIC TCP や Compound TCP 以外の TCP 輻輳制御アルゴリズムを用いての評価，様々な通信アプリケーションを用いての評価，履歴の記録周期や履歴サイズなどに関する考察を行っていく予定である．

謝辞

本研究は JSPS 科研費，24300034，25280022 の助成を受けたものである．

参考文献

[1] D.Katabi, M.Handley, and C.Rohrs, “Congestion control for high bandwidth-delay product networks,” in Proceedings of ACM SIG-COMM 2002,Aug.2002.

[2] 大浦亮, 山口実靖, "実機を用いた高速 TCP の公平性の評価", FIT2011 第 10 回情報科学技術フォーラム, RL-003, Sep. 2011

[3] Ryo Oura, Saneyasu Yamaguchi, “Fairness Comparisons Among Modern TCP Implementations,”The 6th International Workshopon Telecommunication Networking, Applications and Systems (TeNAS 2012), Mar. 2012.

[4] L. Xu, K. Harfoush and I. Rhee, “Binary Increase Congestion Control for Fast Long-Distance Networks,” Proc. IEEE Info COM 2004, March 2004

[5] Injong Rhee and LisongXu “CUBIC: A New TCP-Friendly High-Speed TCP Variant,” Proc. Workshop on Protocols for Fast Long Distance Networks, 2005, 2005.

[6] Kun Tan, Jingmin Song, Qian Zhang, and MurariSridharan,” A Compound TCP Approach for High-speed and Long Distance Networks” Proc.IEEE Info COM 2005,July 2005.

[7] 逸身勇人，山本幹，“CUBIC と Compound TCP 間の公平性改善手法の提案，”電子情報通信学会信学技報 vol. 110, no. 372, NS2010-160，pp. 103-108

[8] 長谷川剛，板谷夏樹，村田正幸，“バックボーンルータにおける RED の動的閾値制御方式，”電子情報通信学会信学技報　NS2001-11

[9] Jeonghoon Mo, Richard J. La, VenkatAnantharam, and Jean Walrand, “Analysis and comparison of TCP reno and vegas”, in Proceedings of IEEE INFOCOM’99, March 1999.

[10] 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.

[11] S. Floyd and V. Jacobson, "Random early detection gateways for congestion avoidance," IEEE/ACM Transactions on Networking, vol. 1, pp. 397.413, Aug. 1993.