情報処理学会論文誌 Vol.58 No.12 1993–2005 (Dec. 2017)

推薦論文

遅延時間制御手法の破棄制御によるTCP公平性の向上

花井 雅人[1,a)] 山口 実靖[1] 小林 亜樹[1]

受付日 2017年3月21日, 採録日 2017年9月5日

概要：計算機が得られるTCP通信性能はその計算機が搭載しているOSのTCP実装により異なり，使用するTCP輻輳制御アルゴリズム間で不公平が生じることがあるという課題がある．本稿では，主要なTCPアルゴリズムであるCompound TCPとCUBIC TCPと，今後普及が期待される待ち行列管理手法のCoDelに着目し，TCP間の性能公平性の改善手法について考察する．まず，両TCPがネットワークを共有していない環境における各TCPの性能を評価し，競合していない環境においては各TCPはネットワーク帯域を使いきることができTCPアルゴリズムによる性能差がないことを示す．次に，両TCPがネットワークを共有する環境において各TCPが得られる性能を評価し，両TCPで得られる性能が大きく異なり公平性が低いことを示す．そして，通信機器で帯域消費の大きい通信の推定を行い，その通信のパケットを優先的に破棄するようにCoDelを改変しTCP公平性を改善させる手法を提案する．最後に，提案手法の性能評価結果を示し，提案手法によりTCP公平性を大きく改善させることができることを示す．

キーワード：TCP公平性，Compound TCP，CUBIC TCP，CoDel

TCP Fairness Improvement by Modifying Packet Dropping of Controlled Delay

Masato Hanai[1,a)] Saneyasu Yamaguchi[1] Aki Kobayashi[1]

Received: March 21, 2017, Accepted: September 5, 2017

Abstract: Obtained TCP throughput depends on TCP congestion algorithms. In this paper, we focus on two popular TCP algorithms, CUBIC TCP and Compound TCP, and CoDel, which is one of promising queue controlling methods. We then discuss a method for improving performance fairness between TCP algorithms. First, we evaluate performances of both algorithms with and without sharing network, and demonstrate that their fairness is severe. Second, we propose a method for improving TCP fairness by modifying CoDel. The method monitors packets that pass through the network element and detects the most bandwidth-consuming communication. This method then preferentially drops packets of the most consuming communication in the CoDel process. Third, we evaluate our method and demonstrate that our method significantly improves TCP fairness.

Keywords: TCP fairness, Compound TCP, CUBIC TCP, CoDel

1. はじめに

従来のTCP輻輳制御アルゴリズムであるTCP Reno [1]では近年の高速な通信帯域を使いきることができず，この問題を解決するために多くの高速TCPアルゴリズムが提案されている [2], [3], [4], [5]．計算機が得られるTCP通信のスループットはTCPの実装に依存するため，これらの提案によりTCPアルゴリズム間の不公平性という新たな課題が生じている．

このTCPアルゴリズム間の性能公平性（TCP公平性）の改善手法として，RED [6] の適用やREDを改変して公平性を向上させるActive Packet Dropping [7] がある．し

[1] 工学院大学
Kogakuin University, Shinjuku, Tokyo 163–8677, Japan
[a)] cm16036@ns.kogakuin.ac.jp

本稿の内容は2016年7月のマルチメディア，分散，協調とモバイルDICOMO2016シンポジウムで報告され，インターネットと運用技術研究会主査より情報処理学会論文誌ジャーナルへの掲載が推薦された論文である．

かし，RED はパラメータチューニングが容易でないなどの指摘もあり [8]，必ずしも普及が進んでいない．また，RED の複雑さに着目しパラメータを減らし，運用を容易にした待ち行列管理手法として Nichols らが提案した CoDel [9] がある．今後は本手法が普及していくと期待でき，CoDel に基づく TCP 公平性向上に関する考察が重要であると考えられる．

本稿では主に，動的手法 [10] の通信管理の改善と複数台環境における評価を行い，より一般的環境における有効性を示す．我々はこれまで TCP 公平性改善手法として，通信帯域消費が最も大きいコネクションが通信機器にとって既知であるという前提に基づき CoDel を改変した静的手法 [11], [12], [13] と，既知であることを前提としない動的手法 [10], [14], [15] を提案し，簡易な試作実装を用いて小規模で初歩的な評価を行ってきた．本稿は TCP 不公平性が生じる理由の考察，提案手法における通信管理の改善，各 TCP を使用する計算機が複数台である環境における評価，破棄率と性能の関係の考察などの拡張を行い TCP 公平性改善手法について考察する．

本稿の構成は以下のとおりである．2 章で，著名な TCP アルゴリズム，本稿で着目する CoDel，既存の TCP 公平性改善に関する取り組みなどの関連する研究を紹介する．3 章で，著名な 2 つの TCP 実装を用いて TCP 公平性の評価を行い，その不公平性が生じる理由を述べる．4 章で，CoDel を改変し TCP 公平性を向上させる手法を提案する．5 章で，提案手法の性能評価を行い，提案手法により TCP 公平性が改善されることを示す．6 章で本稿をまとめる．

2. 関連研究

2.1 TCP 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 実装には輻輳制御機能が搭載されている．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 を用いる通信の性能はこのアルゴリズムに大きな影響を受ける．OS により様々な輻輳制御アルゴリズムが実装されており，それぞれの OS で輻輳制御の方式が異なる．

TCP 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の 3 つに分類できる．

ロスベース手法はパケットが損失したことを検出し，これに基づいて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サイズを増加させ，パケットロスが検出されたときには輻輳ウィンドウサイズを大幅に減少させる．従来の TCP である TCP Tahoe，TCP Reno [1] や，近年の Linux OS で使用されている BIC TCP [3]，CUBIC TCP [4] がロスベース手法である．

遅延ベース手法は，RTT の増減に基づいて輻輳ウィンドウを制御する手法である．ロスベース手法のように輻輳が発生してから速度を大幅に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況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を期待することができる．欠点としてロスベース手法と同一ネットワークにおいて同時に通信を行っ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 [16]．代表的な遅延ベース手法に TCP Vegas [2] がある．

ハイブリッド型手法はロスベースと遅延ベースを組み合わせた手法である．代表的なハイブリッド型手法の TCP に Windows 系 OS に搭載されている Compound TCP [5] がある．遅延ベース手法同様にハイブリッド型手法も，ロスベース手法と同時に通信を行ったときに得られる性能が低くなってしまうという問題が指摘されている [17], [18], [19], [20]．

本研究では，Linux OS の標準 TCP である CUBIC TCP と Windows 系 OS に搭載されている Compound TCP に焦点を当てて考察を行う．

2.2 CUBIC TCP

CUBIC TCP [4] は，BIC TCP [3] のスケーラビリティを維持しながら，既存の TCP アルゴリズムとの公平性である TCP 公平性や，RTT の異なる通信間での公平性である RTT 公平性，制御手法の複雑さを改善した高速 TCP アルゴリズムである．CUBIC TCP は Linux2.6.19 以降で標準の TCP アルゴリズムとして搭載されている．

CUBIC TCP では，BIC TCP のバイナリーサーチを用いて利用可能帯域を検索するアルゴリズムを式 (1)，(2) のような 3 次関数を用いた制御によって実現している．

$$cwnd = C(t-K)^3 + W_{max} \tag{1}$$

$$K = \sqrt[3]{\frac{W_{max}\beta}{C}} \tag{2}$$

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から経過した実時間，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．通常，C には 0.4 が，β には 0.2 が用いられている．

CUBIC TCP では，上記のようにパケットロス検出時からの経過時間を用いて輻輳ウィンドウの値を定めている．これは，RTT の影響を強く受ける ACK の受信を輻輳ウィンドウサイズの増加処理から排していることを意味し，これにより RTT 公平性が向上することが期待できる．加えて，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という問題もこれにより解決している．

また，TCP Reno を用いた場合に得られる輻輳ウィンドウサイズを式 (3) 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輻輳ウィンドウ

サイズとして採用している．これにより，TCP 公平性の向上を目指している．

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

そして，ボルトネックを共有するフローが帯域を公平に分け合うまでの収束時間を短縮するためにパケットロスを検出したときの輻輳ウィンドウサイズが前回ロスを検出したときの値を下回っている場合は新たな W_{max} は次式のように設定される．

$$W_{max} = cwnd \cdot \frac{(2-\beta)}{2} \tag{4}$$

以上のような仕組みにより，CUBIC TCP は高いスケーラビリティ，RTT 公平性，TCP 公平性を目指している．

ただし，式 (1) のように RTT に依存せず確実に輻輳ウィンドウサイズを上昇させる本手法は，上昇速度が RTT の上昇にともない低下する手法よりも高い性能を得ることも予想される．また，本手法は輻輳ウィンドウの上昇に 3 次関数を用いており，1 次関数などの相対的に消極的な上昇を行う手法より多くの通信帯域を得てしまうことも予想される．よって，本手法が他の手法の通信帯域を圧迫し，結果として公平性が損なわれる可能性は考えられる．

2.3 Compound TCP

Compound TCP はロスベースの輻輳制御で動作するロスベースウィンドウと遅延ベースの輻輳制御で動作する遅延ベースウィンドウの両方を使ったハイブリッド型の TCP アルゴリズムである．

ロスベー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という 2 つのフェーズで構成されている．この 2 つのフェーズでウィンドウの増加量が異なり，それぞれ次式で与えられる．

$$cwnd(t+1) = \begin{cases} cwnd(t)+1 & (\text{スロースタートフェーズ}) \\ cwnd(t)+\dfrac{1}{swnd(t)} & (\text{輻輳回避フェーズ}) \end{cases} \tag{5}$$

ただし $swnd$ は現在のロスベースウィンドウサイズと遅延ベースウィンドウサイズの和であり，t は ACK を受信するたびに増加する値であり実時間に依存しない．スロースタートフェーズではロスベースウィンドウサイズを指数関数的に増加させ，輻輳回避フェーズではロスベースウィンドウサイズを線形的に増加させる．

パケットロスを検出した場合，ロスベースウィンドウサイズが大幅に減少する．この減少量はパケットロスの検出方法によって異なり，それぞれ次式で与えられる．

$$cwnd(t+1) = \begin{cases} \dfrac{cwnd(t)}{2} & (\text{重複 ACK による検出}) \\ 1 & (\text{タイムアウトによる検出}) \end{cases} \tag{6}$$

重複 ACK の受信によりパケットロスを検出した場合は，ネットワークにおいて軽度な輻輳が発生したと判断してロスベースウィンドウサイズを現在の値の半分の値まで減少させる．一方，タイムアウトによるパケットロスを検出した場合はネットワークにおいて重度な輻輳が発生したと判断してロスベースウィンドウサイズを 1 に減少させる．

また，遅延ベース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．遅延ベースウィンドウはスロースタートフェーズにおいては動作せず，輻輳回避フェーズにおいてのみ動作する．遅延ベースウィンドウサイズは，ネットワーク中に滞留しているパケット数 $Diff$ に基づいて次式で与えられる．

$$dwnd(t+1) = \begin{cases} dwnd(t) + (\alpha \cdot swnd(t)^k - 1) & (Diff < \gamma) \\ (dwnd(t) - \zeta \cdot Diff) & (Diff \geq \gamma) \end{cases} \tag{7}$$

$$\begin{aligned} Expected &= \frac{swnd(t)}{baseRTT} \\ Actual &= \frac{swnd(t)}{RTT} \\ Diff &= (Expected - Actual) \cdot baseRTT \end{aligned} \tag{8}$$

ここで，$baseRTT$ は実際に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．

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の帯域があると判断し遅延ベースウィンドウサイズを増加させる．推測値 $Diff$ が閾値 γ よりも大きい場合は，ネットワークに輻輳が発生していると判断し遅延ベースウィンドウサイズを減少させる．

Compound TCP では，ロスベースウィンドウおよび遅延ベースウィンドウに基づいて送出ウィンドウサイズを次式により決定する．

$$swnd(t+1) = cwnd(t+1) + dwnd(t+1) \tag{9}$$

公平性に関して，同文献 [5] において著者らは以下のように主張している．すなわち，ロスベースの手法群は非常に積極的でありこの積極性が劣悪な RTT 不公平性と TCP 不公平性の原因となっていること，遅延ベースの手法は優れた RTT 公平性を達成できるがロスベースの手法と競合した場合は高い性能を得られないことを主張している．そして，Compound TCP は標準のロスベース手法に，スケーラブルな輻輳ウィンドウの増加と遅延の増加に対して送信レートを大幅に減少できる遅延ベース手法を追加した手法であり，良い RTT 公平性と TCP 公平性を達成できると主張している．

ただし，本手法に関して以下のような不公平性の原因も考えられる．すなわち，本手法は，式 (5) のように単調に輻輳ウィンドウサイズを増加させ続ける側面と，式 (7) のように RTT の増加にともない輻輳ウィンドウサイズを低

下させる側面の 2 つの特徴を有している．後者の側面より，ロスベース手法と競合すると通信帯域をロスベース手法に明け渡してしまい，得られる性能が下がってしまうと予想することができる．また，前者の単調に輻輳ウィンドウサイズを増加させる側面に関しても，上昇の速度は 1 次関数であり，3 次関数などのより積極性の高い手法と競合した場合には得られる性能が相対的に低くなってしまう可能性も考えられる．

2.4 RED（Random Early Detection）

RED [6] は平均待ち行列長に応じた確率でパケットを破棄する手法である．TailDrop では待ち行列がバッファサイズに達するとすべてのコネクションのパケットが破棄される．一方 RED では平均待ち行列長が閾値 min_{th} に達するとパケットの破棄が開始され，平均待ち行列長が max_{th} に達すると破棄率が 1 となり到着したすべてのパケットが破棄される．RED では安定した輻輳制御が行うことができ，公平性の改善も実現できると期待されている．

2.5 Active Packet Dropping

文献 [7], [21] で，スイッチやルータなどのネットワーク機器におけるパケット破棄を制御することにより TCP 公平性を改善させる手法である Active Packet Dropping が提案されている．当該手法は RED を基にしており，帯域消費の大きいコネクションの RED におけるパケット破棄確率を増加させ，公平性の改善を図っている．

両文献では，静的手法と動的手法が提案されており，静的手法では帯域消費が大きいコネクションの情報を既知であるとの前提のもと，そのコネクションのパケット破棄確率を増加させる．動的手法では定期的にパケットを監視し，サンプリングされたパケットから帯域消費の大きいコネクションを推定し，そのコネクションのパケット破棄確率を増加させる．

両手法の実装，実機，実 TCP 実装を用いた性能評価がなされ，公平性の改善が確認されている．

2.6 Bufferbloat

近年のルータなどの通信機器には大容量のバッファが搭載されている．現状のインターネットのルータのバッファはつねにパケットで満たされており，各パケットは長い待ち行列を待つ必要があり，結果として現状のインターネット通信の通信遅延はつねに大きい状態にあるとの指摘 [8] があり，この問題は Bufferbloat と呼ばれている．Bufferbloat の解決策の 1 つとして，次節で述べる遅延時間制御手法 CoDel [9] がある．

2.7 CoDel（Controlling queue Delay）

CoDel [9] は Nichols らにより提案された待ち行列のスケジューリングアルゴリズムである．CoDel は，あるパケットが待ち行列に入ってから待ち行列を出るまでの時間が *target* 以上になると，パケットを破棄する．*target* はチューニングパラメータであり，初期値は 5 ms である．

RED と異なり CoDel はチューニングパラメータが少なく，パケット破棄するか否かは待ち行列の滞在時間のみで決定される．RED の普及が必ずしも進まない原因の 1 つとしてパラメータの多さと，それらの設定の難しさがあげられ，CoDel ではより簡単な設定で高い性能を実現でき，今後は普及が進んでいくと期待することができる．

RED と同様に，CoDel でスイッチやルータ上のパケット待ち行列を制御することにより TCP 公平性の改善が実現できると考えられ，公平性の改善などに関しては，今後は本手法を基にした考察を行うことが重要になっていくと考えられる．

2.8 静的優先制御公平性改善手法および既存の動的手法

前節で述べたように，CoDel の適用により TCP 公平性の改善ができると期待できる．我々は文献 [12] において CoDel 適用環境と非適用環境における TCP 公平性の評価を行い，一部の環境で CoDel の適用により TCP 公平性の改善が可能であること，一部の環境では CoDel を適用しても公平性が低いことを示した．そしてネットワーク帯域を大きく消費する通信が通信機器にとって既知であることを前提とし，この前提のもとで TCP アルゴリズム間で CoDel の *target* 時間を変えて TCP 性能公平性を小規模ながら向上させる手法を提案した．また文献 [11], [13] で，同様の前提のもとで TCP アルゴリズム間でパケットの破棄率を変えて TCP 公平性の改善を行う手法を提案し，性能評価によりその有効性を示した．

しかし，一般にネットワーク帯域を大きく消費する通信は通信機器にとって既知ではなく，この前提なく適用可能な手法の考察が重要な課題となっている．我々は文献 [10] で，通信機器で通過するパケットの観察を行い最も通信帯域の消費の大きいコネクションを推定し，この推定に基づき破棄率を変えて TCP 公平性を向上させる手法を提案し基礎的な評価を行った．そして，文献 [14], [15] で，既存手法との比較，ネットワーク遅延時間と公平度の関係，TCP アルゴリズムごとの性能公平性などの詳細な評価を行い本手法の有効性を示した．また文献 [22] で，単一 TCP 環境における基礎性能評価を行い TCP 不公平性が TCP アルゴリズムに起因していることを示した．

しかし，本動的手法は各 TCP アルゴリズムを用いる計算機が 1 台のみである小規模な環境でのみ評価されており，各 TCP アルゴリズムを用いる計算機が複数台あるような異なる環境においても有効であるかは検証されていない．また，各計算機における TCP コネクション数が多い場合や少ない場合でも提案手法が有効であるかの検証もなされ

ていない．さらに，公平化の対象を MAC アドレスで管理する試作実装が用いられており，単一ホップ以外のネットワークへの適用などについての考慮がされていない．

3. CUBIC TCP と Compound TCP の TCP 公平性の評価

本章で，実機と実 TCP 実装を用いて TCP 公平性の評価を行う．

3.1 測定環境

図 1 のネットワークを構築し，iperf [23] を用いて CUBIC TCP と Compound TCP のコネクションが混在する環境における各 TCP のスループットを評価した．図内の CoDel 機は CoDel または TailDrop を実行，Delay 機は人工的に遅延をエミュレートする計算機である．人工遅延装置は Linux Netem を用いて構築した．ネットワーク機器はすべて 1 Gigabit Ethernet に対応している．Linux1，2，3 機，Windows1，2 機の仕様は**表 1** のとおり，CoDel 機，Delay 機の仕様は**表 2** のとおりである．

図内の Linux1，2 機，Windows1，2 機が送信機であり，Linux3 機が受信機である．本稿の以後の計測はすべて送信機から受信機にデータを送信することにより行っている．Linux1，2 機の TCP アルゴリズムは CUBIC TCP であり，Windows1，2 機の TCP アルゴリズムは Compound TCP である．各送信機では 10 本の TCP コネクションを確立している．複数の送信機から送信を行う場合は，通信群は CoDel 機から Linux3 機までのネットワーク機器を共有することになる．送信機，受信機のウィンドウサイズは 32 MB である．

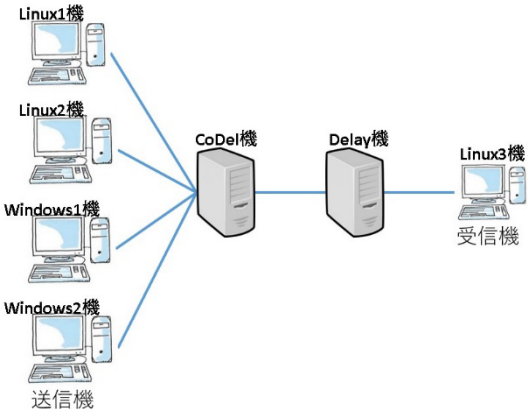

図 1 実験環境
Fig. 1 Experimental Network.

表 1 計算機仕様 1
Table 1 Specification of computers (1).

CPU	AMD Turion, 2.20 GHz
Memory	4 GB
OS	Linux1, 2 機: Linux 4.2.3 Linxu3 機: Linxu 3.3.4 Windows1,2 機: Windows7 Enterprise

表 2 計算機仕様 2
Table 2 Specification of computers (2).

CPU	Intel Celeron G540, 2.4 GHz
Memory	2 GB
OS	Linux 3.17.4

3.2 単独通信時性能

競合通信環境における性能を評価するに先立ち，単独通信環境における性能の評価を行う．図 1 の環境において，送信用計算機（Linux1，Windows1）の 1 台から受信機（Linux3）に iperf による TCP データ転送を行い，その性能を測定した．Delay 機において人工的に付加した遅延時間（以後「付加遅延」）は 1 ms から 64 ms に変動させた．

図 2 に評価結果を示す．図 2 の横軸は片道遅延，縦軸は 10 本のコネクションの平均のスループットである．図より，他の PC や他の TCP アルゴリズムの通信と競合しない環境であれば，各機の各 TCP 実装は単独で通信帯域をほぼ使いきれることが確認できる．

3.3 同時通信時性能（各 TCP 単一計算機）

続いて，複数の TCP アルゴリズムによる通信が競合して存在する環境における性能の評価を行う．図 1 の環境において，Linux1 と Windows1 を送信機とし，Linux3 を受信機とし，Linux1–Linux3 間と，Windows1–Linux3 間で並列に iperf のコネクションを確立し，CUBIC TCP コネクションと Compound TCP コネクションの通信速度を測定した．CoDel 機においては，TailDrop または CoDel を用いて待ち行列の管理を行った．

評価結果を**図 3** に示す．図の横軸の値は付加遅延，縦軸の値は各送信機の 10 コネクションの平均スループットであ

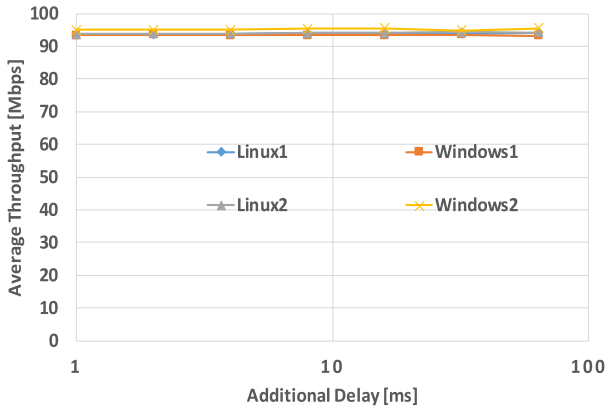

図 2 単独通信時の各 TCP のスループット
Fig. 2 Throughput of each TCP (non-competitive communication).

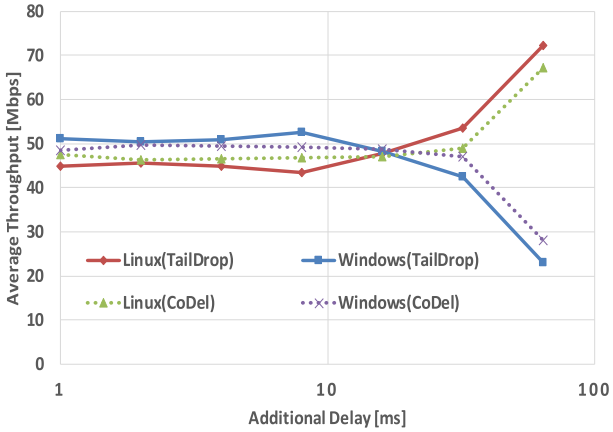

図 3 各 TCP のスループット（各 TCP 単一計算機）
Fig. 3 Throughput of each TCP (single host per TCP).

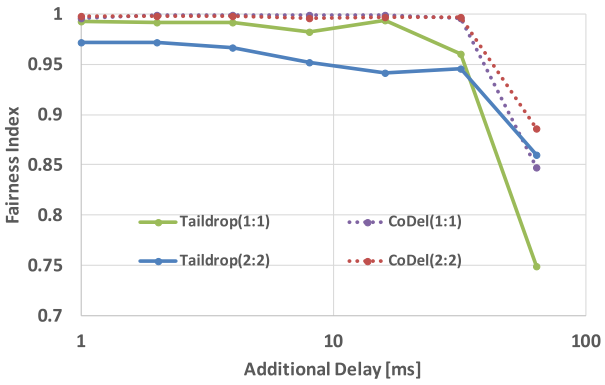

図 4 Fairness Index（TailDrop, CoDel）
Fig. 4 Fairness Index (TailDrop, CoDel).

る．実線が TailDrop，点線が CoDel 適用時の性能である．図より，付加遅延 32 ms 以下の低遅延環境での TCP 公平性は TailDrop 使用時も低くはなく，CoDel 適用時は一定程度保たれていることが分かる．一方，付加遅延 64 ms の高遅延環境では，TailDrop 適用時，CoDel 適用時ともに非常に TCP 公平性が低くなっていることが分かる．

本測定における両 TCP の性能の Fairness Index [24] を図 4 の “TailDrop (1:1)” と “CoDel (1:1)” に示す．Fairness Index は公平度を表す指標であり，次式で定義される

$$\frac{(\sum x_i)^2}{n\sum(x_i^2)} \tag{10}$$

ただし，x_i は各 TCP コネクションのスループット，n はコネクションの数である．0 から 1 の値をとり，1 に近いほど高い公平度を表している．Fairness Index の詳細は付録 A.1 に記す．同図からも，TailDrop においては付加遅延 32 ms 以下では公平度は低くはないが，64 ms において非常に公平度が低く（約 0.75）なることを確認できる．また，CoDel を用いることにより公平度を改善することができるが，64 ms においては CoDel を用いても公平度が低く（約 0.85）なっていることを確認できる．

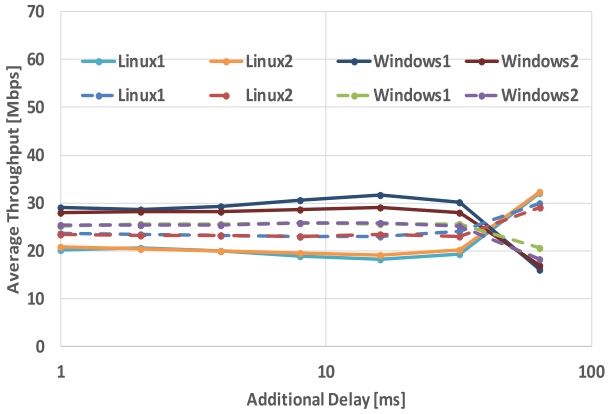

図 5 各 TCP のスループット（各 TCP 複数計算機）
Fig. 5 Throughput of each TCP (multiple hosts per TCP).

3.4 同時通信時性能（各 TCP 複数計算機）

図 5 に，各 TCP アルゴリズムを搭載した計算機が複数台（2 台）ある環境における性能評価結果を示す．縦軸および横軸は図 3 と同じである．実線が TailDrop，点線が CoDel のスループットである．図より，送信端末を複数台にした場合でも類似の結果が得られることが分かる．すなわち，付加遅延が 32 ms 以下の低遅延環境では，TailDrop 適用時は公平性がやや低く，CoDel 適用時はある程度保たれる．そして，付加遅延が 64 ms の高遅延環境では TailDrop，CoDel 適用環境ともに公平性が低くなっていることが分かる．

同様に，本節の測定における両 TCP の性能の Fairness Index を図 4 の “TailDrop (2:2)” と “CoDel (2:2)” に示す．図より，TailDrop と CoDel ともに付加遅延 64 ms において公平度が非常に低くなっていることが分かる．

3.5 考察

本節で，CUBIC TCP と Compound TCP における TCP 公平性が低くなった理由を考察する．

一般に，ロスベース手法と遅延ベース手法で性能の公平度が低くなることは主として通信機器の待ち行列長が大きくなり輻輳が近づいたときの両手法の振舞いの違いに起因しているといえる．遅延ベース手法は，待ち行列長が大きくなり RTT が増加していることを検出すると輻輳の低減を目指し輻輳ウィンドウサイズと自身の送出速度を低下させる．一方ロスベース手法は，RTT の増加や輻輳が近づいていることは考慮せず，輻輳に至りパケットロスを検出するまで単調に輻輳ウィンドウと自身の送出速度を上げ続ける．結果的に，遅延ベース手法が通信帯域を明け渡し，ロスベース手法が多くの帯域をとってしまうことになる．

本章の評価で低い TCP 公平性が生じた原因の 1 つ目として，これと類似のことが発生していることが考えられる．本章の評価で性能が低くなった Compound TCP は，式 (5) のロスベースウィンドウサイズおよび式 (7) の遅延

ベースウィンドウサイズの合計により輻輳ウィンドウサイズを決定している．ロスベースウィンドウサイズは，式 (5) のように輻輳によるパケットロスが発生するまで単調に増加し，遅延ベースウィンドウサイズはネットワークの遅延時間により増減する．一方 CUBIC TCP は，輻輳ウィンドウサイズを式 (1) の 3 次関数を用いてパケットロスが発生するまで単調に増加させていく．

よって，両 TCP がネットワーク機器を共有している状況で通信を行うと多くの場合は以下のような振舞いになると考えられる．Compound TCP や CUBIC TCP のウィンドウサイズが増加していき，両計算機の出力速度が時間とともに増加していくと，通信機器の待ち行列長が増加していく．通信機器の待ち行列が長くなっていくと TCP で観測される RTT が増加していくため，Compound TCP の遅延ベースウィンドウの値は減少していき，Compound TCP の出力速度は減少していくこととなる．一方で，CUBIC TCP は RTT が増加しても単調に輻輳ウィンドウの値を増加させていくこととなる．換言すると，ネットワークが混雑してくると，Compound TCP は混雑を緩和させるためにネットワークに入力する量を減少させるが，CUBIC TCP は Compound TCP から譲られた資源を使いつくそうとしてしまい，結果として得られるネットワーク帯域は CUBIC TCP の方が多くなってしまう．

2 つ目の理由として，輻輳ウィンドウサイズの成長速度が考えられる，Compound TCP にもロスベースウィンドウが含まれており，両 TCP とも出力速度を単調に増加させる側面を有している．しかし，その増加速度には大きな違いがある．Compound TCP の輻輳回避フェーズは 1 次関数で輻輳ウィンドウを増加させていくが，CUBIC TCP は 3 次関数で増加させていく．よって，両 TCP が並列に輻輳ウィンドウを増加させた場合，CUBIC TCP の方が短時間で大きな値に達し高い速度を得やすくなると考えられる．

最後に，スロースタート閾値の差の影響について考察する．パケットロス検出時には CUBIC TCP，Compound TCP ともに輻輳ウィンドウサイズを乗算を用いて減少させる．具体的には，パケットロス検出時の輻輳ウィンドウのサイズの定数倍をスロースタート閾値に定め，スロースタート閾値まではスロースタートフェーズによりきわめて短時間で（あるは高速再転送適用時にはゼロの時間で）到達する．標準的な設定の場合，CUBIC TCP のスロースタート閾値はパケットロス検出時の輻輳ウィンドウサイズの 0.8 倍であり（付録 A.2 参照），Compound TCP のスロースタート閾値は 0.5 倍となっている．よって，輻輳検出直後の輻輳ウィンドウサイズにおいても CUBIC TCP の方が大きくなっており，これもある程度の影響が与えていると予想できる．

以上のように，前章で紹介した両 TCP の振舞いが原因で性能不公平が生じると考えることができる．また，ウィンドウサイズの影響が大きい高遅延環境（64 ms）で特に不公平性が高くなっていることからも，この不公平性の原因が輻輳ウィンドウサイズによるものであることが分かる．

4. 提案手法

4.1 TCP 公平性向上手法

本章で，CoDel の改善により TCP 公平性を改善する手法を提案する．CoDel においては，パケットが通信機器の待ち行列に入ってから出るまでの時間（待ち時間）が指定値 *target* を超えるとパケットが破棄される．これに対して本稿では，通信帯域の消費が最も大きいと推定される通信のパケットが破棄対象となった場合は必ず破棄し，それ以外の通信のパケットが破棄対象となった場合は確率 p で破棄する手法を提案する．p はチューニングパラメータである．

通信帯域の消費が最も大きい通信の推定は次の方法で行う．提案手法を搭載した通信機器を通過するパケットのうち `stat_int` 個に 1 個のパケットの通信情報をログに記録する．そして，パケットを `hist_len` 個記録するごとにログの集計を行い，その時点でログ内に最も多く登場する通信を「通信帯域の消費が最も大きい通信」と推定する．ログ長は `hist_len` とする．通信情報としては，送信元 IP アドレスと送信先 IP アドレスを記録する．

4.2 実装

本節で，提案手法の我々の実装について述べる．

我々は Linux OS に搭載されている CoDel の実装を修正することにより提案手法を実装した．CoDel 実装内で通信情報 `hist_len` 個分の配列をログ用に確保し，同実装におけるパケット受信のたびに行われる処理に `stat_int` 回に 1 回対象パケットの通信情報をログに記録する機能を追加した．そして，パケットを `hist_len` 個記録するたびにログを解析し，ログ内における登場回数が最も大きい通信を「通信帯域の消費が最も大きい通信」とするようにした．

負荷の低減と実装の簡素化のために，確率 p での破棄は乱数により無記憶に行うのではなく，ループするカウンタを用いて $1/p$ 回に 1 回破棄を行うよう実装した．**図 6** に Linux における CoDel の実装および我々の修正例の概要を示す．斜体部が我々が追加した部分であり，非斜体部が元々の Linux の CoDel の実装に存在していた部分である．`codel_dequeue()` は CoDel において待ち行列からパケットを 1 個取り出す処理を実装した関数であり，`codel_should_drop()` は当該パケットを破棄するべきか否かを判断し，その真偽を返す関数である．図の非斜体部のように，`codel_dequeue()` で `codel_should_drop()` を呼び出し，取り出したパケットを破棄すべきであるか否かを調査し，破棄すべきとの判定が出た場合はそのパケットを

```
static bool codel_should_drop(...){
  :
  if( p_cnt % STAT_INT == 0 ){
    ログにパケット情報を記録
  }
  p_cnt ++;
  if( HIST_LEN <= ログ長 ){
    most_consuming = ログ内登場回数が最も大きい通信;
  }
  :
  /* 本パケットを破棄するか否かを返す */
}

static struct sk_buff *codel_dequeue(...){
  :
  drop = codel_should_drop(...);
  :
  if( drop ){
    :
    if( 本パケットのSRCとDSTのIPアドレス==
        most_consumingのSRCとDSTのIPアドレス ){
      パケットを破棄
    } else {
      if( d_cnt % DROP_INT == 0 ){
        /* DROP_INT に1回破棄 */
        パケットを破棄
      }
      d_cnt ++;
    }
    :
  }
  :
}
```

図 6 提案手法の実装概要
Fig. 6 Overview of the proposed method implementation.

破棄している．

この実装に対して，codel_should_drop() の斜体部のようにパケットをカウンタ（p_cnt）で数え，stat_int 個に 1 個のパケットをログに記録するようにした．また，記録が hist_len 個たまるたびにログ内で登場回数が最も大きい通信を特定し，それを変数 most_consuming に格納している．そして，codel_dequeue() の斜体部のように codel_should_drop() で破棄すべきであるとの判断が行われた場合は，そのパケットが most_consuming と同一の IP アドレスであるか否かにより処理を分岐させている．同一である場合は帯域消費が最も大きい通信と見なし積極的に（通常の CoDel と同様に必ず）破棄させ，同一でない場合は消極的に（カウンタ d_cnt で数え DROP_INT 回に 1 回）破棄させている．DROP_INT と破棄率 p は逆数の関係である．

ただし，codel_dequeue() 内に if(drop) および真の場合に破棄する処理は 2 カ所存在しており，我々の修正もその両方に対してほどこされている．

5. 性能評価

5.1 同時通信時性能（各 TCP 単一計算機）

提案手法の有効性を検証するために，前章で示した提案手法の実装を用い性能評価を行った．測定環境は 3 章と同一である．提案手法における破棄率 p は 1/32，stat_int と hist_len は 100 である．

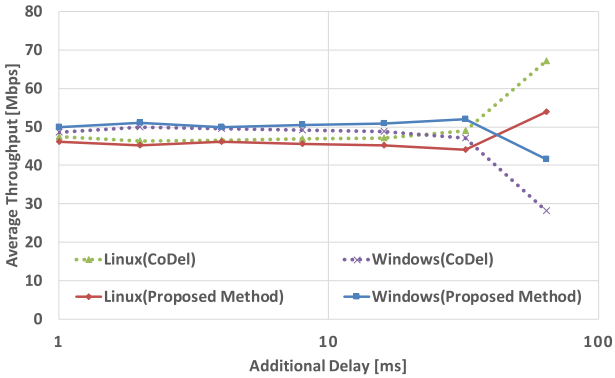

図 7 各 TCP のスループット（各 TCP 単一計算機，提案手法）
Fig. 7 Throughput of each TCP (single host per TCP, proposed method).

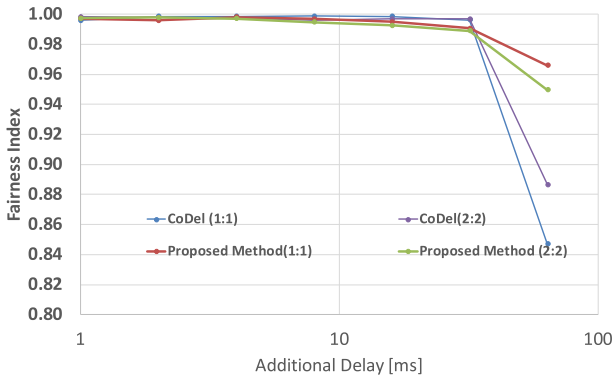

図 8 Fairness Index（CoDel，提案手法）
Fig. 8 Fairness Index (CoDel, proposed method).

各 TCP の送信機が 1 台の環境における提案手法適用時の付加遅延と各 TCP で得られた平均スループットの関係を**図 7** に示す．また，**図 8** に付加遅延と Fairness Index の関係を示す．図より，提案手法は CoDel と同等か CoDel よりきわめて高い公平度を実現していることが分かる．特に，TailDrop や CoDel で公平度が低かった付加遅延 64 ms で提案手法の適用により TCP 公平性が大きく向上していることが分かる．

5.2 同時通信時性能（各 TCP 複数計算機）

次に，各 TCP を実装した送信機が 2 台の環境における評価を行う．CUBIC TCP の送信機が 2 台あり，Compound TCP の送信機が 2 台あり，合計 4 台で 40 本の TCP コネクションが確立されること以外は測定環境は前節と同一である．

測定結果のスループットを**図 9** に，Fairness Index を図 8 に示す．両図より，物理計算機が 4 台の環境においても，提案手法は CoDel と同等か CoDel よりきわめて高い TCP 公平性を実現できていること分かる．

5.3 コネクション数と TCP 公平性

本節で，コネクション数と TCP 公平性の関係を評価す

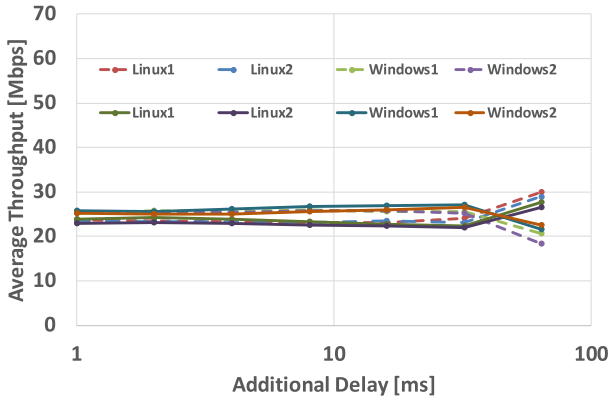

図 9 各 TCP のスループット（各 OS 複数計算機）
Fig. 9 Throughput of each TCP (multiple hosts per OS).

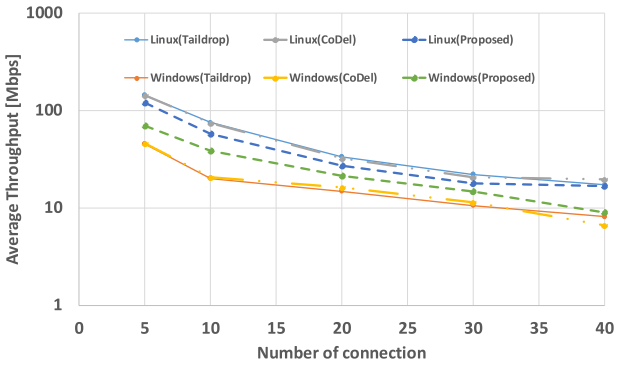

図 10 TCP コネクション数と各 TCP のスループット
Fig. 10 Number of TCP connections and throughput of each TCP.

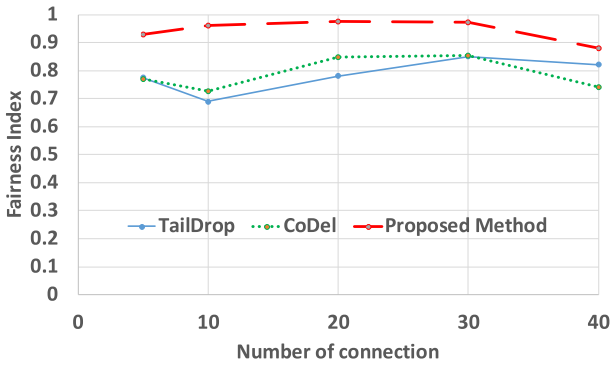

図 11 TCP コネクション数と Fairness Index
Fig. 11 Number of TCP connections and Fairness Index.

る．計算機 1 台あたりの TCP コネクション数を 5 から 40 に変えてコネクション数と公平度の関係を調査した．各 TCP 単一計算機環境，提案手法における破棄率 p は 1/32 とした．

評価結果を**図 10** と**図 11** に示す．図より，提案手法はすべてのコネクション数において TailDrop や CoDel よりも高い TCP 公平性を実現できており，コネクション数によらず本手法が有効であることが分かる．

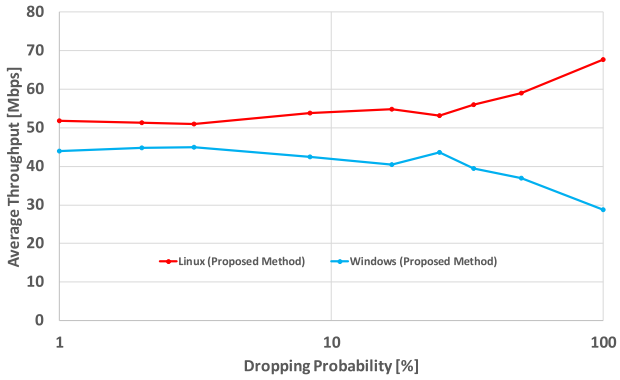

図 12 破棄率と各 TCP スループット（各 TCP 単一計算機）
Fig. 12 Dropping probability and throughput of each TCP (single host per TCP).

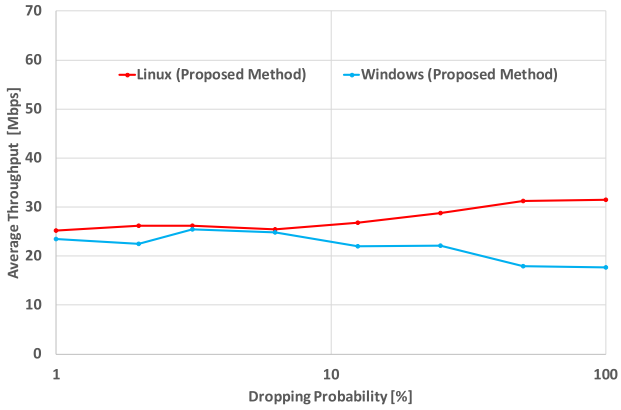

図 13 破棄率と各 TCP スループット（各 TCP 複数計算機）
Fig. 13 Dropping probability and throughput of each TCP (multiple hosts per TCP).

5.4 破棄率 p と TCP 公平性

本節で，破棄率 p と TCP 公平性の関係を示す．付加遅延を基礎評価で最も公平度が低かった 64 ms としてスループットと TCP 公平性を評価した．各 TCP 単一計算機および各 TCP 複数計算機における結果をそれぞれ**図 12**，**図 13** に示す．図内の $p = 100\%$ は CoDel と同一の動作である．両図より，p が小さい状態では提案手法の公平性は CoDel の公平性よりも優れていることが分かる．図 12 では $p = 50\%$ 以下において，図 13 では $p = 25\%$ 以下において提案手法が CoDel より優れる公平度となっている．

両測定における Fairness Index を**図 14** に示す．同図からも，p を小さい値に設定することで公平度が向上することが分かる．なお，図 14 より最高の公平度は数%から 25%程度で得られているが，p を小さな値に定めれば最高に近い公平度を得られることが分かり，本手法は正確に p をチューニングしなければ効果を得られない手法ではないことが確認できる．

5.5 提案手法のオーバヘッド

提案手法は CoDel の処理に新たな処理を追加している．

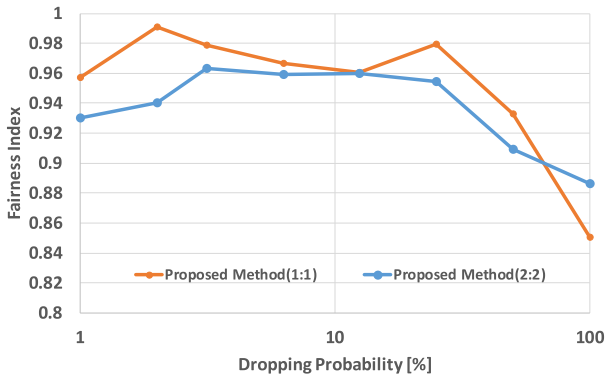

図 14 破棄率と Fairness Index
Fig. 14 Dropping probability and Fairness Index.

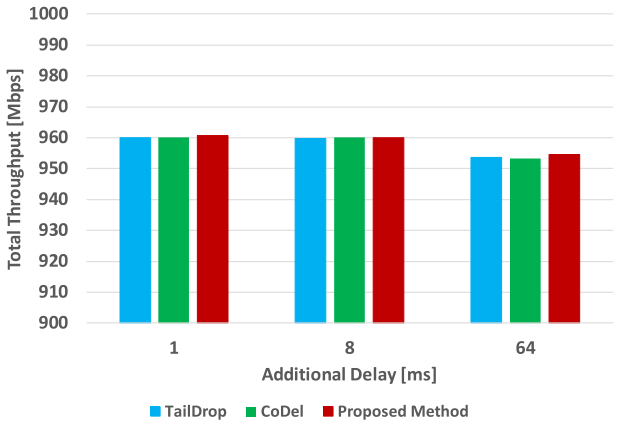

図 15 全通信合計スループット（各 TCP 単一計算機）
Fig. 15 Total throughput (single host per TCP).

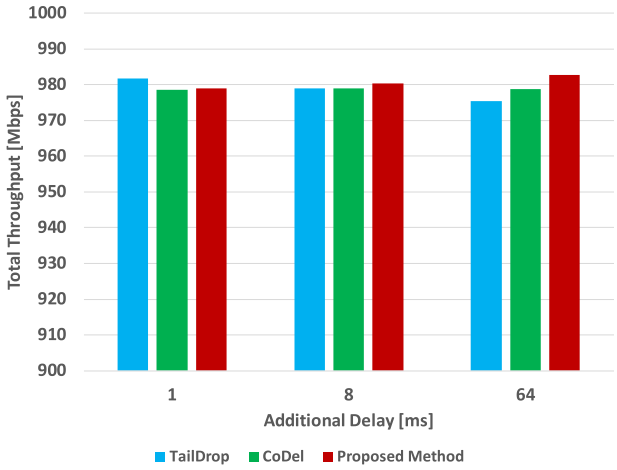

図 16 全通信合計スループット（各 TCP 複数計算機）
Fig. 16 Total throughput (multiple hosts per TCP).

よって，提案手法の処理オーバヘッドにより通信機器により得られる性能が低下する可能性が考えられる．本節で提案手法の処理オーバヘッドによる性能劣化について評価する．

図 15，**図 16**，**図 17** に付加遅延 1 ms，8 ms，64 ms のときの各 TCP の単一および複数計算機における TailDrop，CoDel，提案手法の全コネクションの合計性能を示す．全

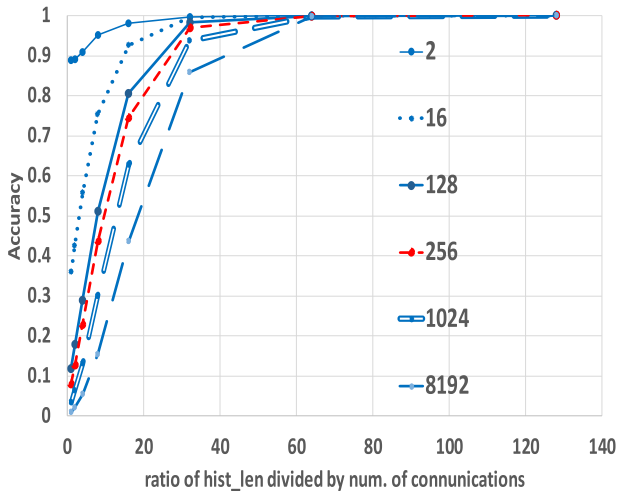

図 17 hist_len/通信数と正答率
Fig. 17 hist_len/num. of flows and accuracy.

コネクションの合計性能は，ネットワーク全体のスループットを示しているといえる．図 15 が各 TCP の送信機が 1 台，図 16 が 2 台の環境における合計性能である．これらの図より，ネットワーク全体の性能の劣化はほとんど存在せず，本計測の例において提案手法のオーバヘッドが非常に小さいものであったことが分かる．

6. 考察

本手法の適用対象に対する考察を述べる．本稿において少数の計算機が同時に通信する環境における性能評価を行い，その有効性を示した．よって，小規模な組織の対外接続ルータに適用することで，対象組織における公平性を改善できると考えられる．また，本手法をバックボーンルータに適用することによりさらなる広範囲の通信の公平性を改善できると期待できるが，そのためにはより多くの通信の観察のためのログ数などのパラメータの考察，多くのログの集計処理を行うための集計処理の低負荷化やルータ処理性能の向上などが必要になると考えられる．本稿では我々の実装により評価を行ったが，本手法が普及し低負荷高速な実装が行われることにより，本手法のさらなる性能向上や広範囲なルータへの適用が可能になると期待できる．

次に，TSO（TCP Segmentation Offload）などのオフロード機能を有するハードウェアの影響について考察する．TSO などによりパケット到着にバースト性が生じ，ログに格納される情報が偏る可能性も考えられる．本手法は `stat_int` パケットに 1 個の情報を記録しているため，本パラメータの調整によりこの影響を軽減できると考えられる．

最後に，提案手法のスケーラビリティとパラメータ設定について考察を行う．本手法は，最近の `hist_len*stat_int` 個のパケットを対象に公平性の改善を試みる．対象が多いほど長い時間に基づく公平性の確保が可能であり，短いほど直近の情報に基づく公平性の改善を行うことができる．

hist_len は記録の数を，stat_int は観察の密度を制御する．本手法は hist_len 個の履歴内における登場回数が最も大きい通信を「最も通信帯域の消費が大きい通信」と推定しているため，スケーラビリティを確保するためには，hist_len を行われている通信の数より十分に大きくする必要がある．

以下に，十分な hist_len の値に関する考察を行う．シミュレーションにより，通信の数と hist_len の数の比と，推定の精度の関係を調査した．シミュレーションでは，n 個の通信が存在し，そのうち 1 個の通信は他の通信の 2 倍のパケットを送出する環境を想定した．すなわち，1 個の通信は全体の $2/(n+1)$ の通信帯域を消費し，残りの $n-1$ 個の各通信は全体の $1/(n+1)$ の通信帯域を消費するとした．そして，この環境で各 hist_len の値にして推定がどの程度の確率で成功するかを調査した．簡略化のため，パケットの到着は無記憶の一様分布乱数に従うとし，1 つの通信のパケットは $2/(n+1)$ の確率で，それ以外の通信のパケットは $1/(n+1)$ の確率で到着することとした．そして，hist_len 個の履歴を調査し，最も登場回数の大きい通信を特定し，その通信が通信帯域が多い（他の通信の 2 倍の）通信と一致しているか否かを調査した．調査結果を図 17 に示す．各系列（各線）の名前は，通信数を表している．図より，我々が調査した範囲である通信数が 8,192 以下においては hist_len を通信数の 64 倍以上にすることにより非常に高い精度（99.7%以上）で推定を行えることが分かった．また，通信数が 256 以下であれば 32 倍でも 97%以上の精度を達成できていることが分かる．本手法は通信帯域の最も大きい通信を相対的に高い確率で選択できれば公平性を改善できると考えられ，精度が高いほどより良く改善を行える．図より，チューニングに関しておおむね以下のように予想することができる．通信の数が 8,192 以下程度であれば，hist_len は通信数の 16 倍程度でおおむね問題がなく，64 倍以上にすれば誤りをほぼ生じさせることなく実行することができる．通信数が 100 以下のような小規模な環境であれば 16 倍以上とすれば十分であると考えられる．

また，公平性を確保する単位を計算機（IP アドレス）でなくサブネット（上位ビットを共有する IP アドレス群）とすれば公平性の確保の単位の数を減少させることができる．さらに，階層化をして本手法を用いる手法も考えられ，上流の通信機器でサブネットごとの公平性の確保を行い，下流の通信機器で計算機ごとの公平性の確保を行うなどの手法も考えられる．

p は，5.4 節で示したように十分に小さな値（1/10 や 1/100）を指定すればよい．パケット損失が発生すると，損失を検出した通信が送出速度を大きく低下させ待ち行列長は減少する．よって，p を小さな値としパケット損失が発生する確率を低くすると，それだけ長い時間待ち行列長が増加し続ける．p が 1/10,000 の場合，「最も帯域消費が大きい通信」以外のパケットに関しては 10,000 パケットに 1 回のパケット損失が生じることとなる．仮に最も帯域消費が大きい通信におけるパケット破棄がまったく行われないとすると，10,000 パケット分程度のバッファが必要となる．1 パケットが 1,500 バイトである例ではこれは 15 MB 程度のバッファとなり，近年の通信機器であれば十分に確保できるサイズであると思われる．以上より，p は 1/10 から 1/100 程度であれば良い性能が得られることが 5.4 節の評価により確認されており，1/10,000 程度までは小さな値としてよいと期待できるといえる．

7. おわりに

本稿では，著名な TCP 輻輳制御アルゴリズムの CUBIC TCP と Compound TCP に着目し，性能評価により両 TCP 間の TCP 公平性が低いことを示した．そして，今後普及が期待される待ち行列管理手法 CoDel に着目し，これの改変による TCP 公平性の改善手法を提案した．提案手法は通信機器でパケットの情報を記録し，通信帯域の消費が最も大きい通信を推定し，その通信のパケットを優先的に破棄する．本手法を CoDel 上に実装し，提案手法適用時の TCP 公平性を評価した結果，これまでの待ち行列手法である TailDrop や本稿で着目した CoDel よりも高い TCP 公平性を実現できることが確認され，本手法が有効であることが示された．

今後は，CoDel のパラメータ *target* を用いた公平性制御についての考察，公衆ネットワークにおける評価，より大規模な環境での評価，手法の低負荷化やさらなる性能向上に関する考察を行っていく予定である．

謝辞 本研究は，JST，CREST の支援を受けたものである．本研究は JSPS 科研費 25280022，26730040，15H02696 の助成を受けたものである．

参考文献

[1] Stevens, W.R.: TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms, IETF RFC 2001 (1997).
[2] Brakmo, L.S. and Peterson, L.L.: 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 (1995).
[3] Xu, L., Harfoush, K. and Rhee, I.: Binary Increase Congestion Control for Fast Long-Distance Networks, *Proc. IEEE Info COM 2004* (Mar. 2004).
[4] Rhee, I. and Xu, L.: CUBIC: A New TCP-Friendly High-Speed TCP Variant, *Proc. Workshop on Protocols for Fast Long Distance Networks* (2005).
[5] Tan, K., Song, J., Zhang, Q. and Sridharan, M.: A Compound TCP Approach for High-speed and Long Distance Networks, *Proc. IEEE Info COM 2005* (July 2005).
[6] Floyd, S. and Jacobson, V.: Random early detection gateways for congestion avoidance, *IEEE/ACM Trans.*

Networking, Vol.1, pp.397–413 (Aug. 1993).
[7] Akiyama, Y., Kozu, T. and Yamaguchi, S.: Active packet dropping for improving performance fairness among modern TCPs, *Proc. Asia-Pacific Network Operations and Management Symposium* (*APNOMS*) (2014).
[8] Gettys, J. and Nichols, K.: Bufferbloat: Dark Buffers in the Internet, *Queue*, Vol.9, No.11, p.40, DOI: http://dx.doi.org/10.1145/2063166.2071893 (2011).
[9] Nichols, K. and Jacobson, V.: Controlling queue delay, *Comm. ACM*, Vol.55, No.7, pp.42–50, DOI: http://doi.acm.org/10.1145/2209249.2209264 (2012).
[10] 花井雅人，山口実靖，小林亜樹：遅延時間制御手法の動的調整による TCP 公平性の向上，マルチメディア，分散，協調とモバイル（DICOMO 2016）(June 2016).
[11] 花井雅人，山口実靖：実機実装を用いた遅延時間制御手法の改良による TCP 公平性の向上，電子情報通信学会ネットワークシステム研究会（NS），信学技報，Vol.115, No.483, NS2015-194, pp.151–156 (2016).
[12] 花井雅人, 山口実靖：遅延時間制御手法の改良による TCP 公平性の向上，電子情報通信学会総合大会 (2016).
[13] Hanai, M., Yamaguchi, S. and Kobayashi, A.: Improving TCP Fairness Between Modern TCP Algorithms Based on Controlling Delay, *2016 IEEE 17th International Conference on High Performance Switching and Routing* (2016).
[14] 花井雅人，山口実靖，小林亜樹：遅延時間制御の改変による TCP 公平性改善手法における破棄対象の動的制御，電子情報通信学会ネットワークシステム研究会（NS），信学技報，Vol.116, No.111, NS2016-47, pp.107–112 (2016).
[15] Hanai, M., Yamaguchi, S. and Kobayashi, A.: Modified Controlling Queue Delay for TCP Fairness Improvement, *The 18th Asia-Pacific Network Operations and Management Symposium* (*APNOMS 2016*) (Oct. 2016).
[16] Mo, J., La, R.J., Anantharam, V. and Walrand, J.: Analysis and comparison of TCP Reno and Vegas, *Proc. INFOCOM '99, 18th Annual Joint Conference of the IEEE Computer and Communications Societies*, Vol.3, pp.1556–1563, IEEE, DOI: 10.1109/INFCOM.1999.752178 (1999).
[17] 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).
[18] Oura, R. and Yamaguchi, S.: Fairness Comparisons Among Modern TCP Implementations, *The 6th International Workshop Telecommunication Networking, Applications and Systems* (*TeNAS 2012*) (2012).
[19] 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 (2011).
[20] Oura, R. and Yamaguchi, S.: Fairness Analysis among Modern TCP Congestion Avoidance Algorithms Using Actual TCP Implementation and Actual Network Equipments, *2011 2nd International Conference on Networking and Computing*, pp.297–299, DOI: 10.1109/ICNC.2011.56 (2011).
[21] 秋山友理愛，大浦　亮，神津智樹，山口実靖：実機と実 TCP 実装を用いた TCP 公平性の評価，電子情報通信学会信学技報，NS2012-149, pp.49–54 (2012).
[22] 花井雅人，山口実靖，小林亜樹：遅延時間制御手法の改変による TCP 公平性改善手法における破棄率の動的制御，FIT2016 第 15 回情報科学技術フォーラム (2016).
[23] iperf homepage, available from ⟨https://iperf.fr/⟩.
[24] Jain, R., Chiu, D. and Hawe, W.: A Quantitative Measure of Fairness and Discrimination for Resource Allocation in Shared Computer System, Technical report, Digital Equipment Corporation (1984).

付　　録

A.1 Fairness Index

Fairness Index [24] は Jain らによって提案された公平性を表す指標であり，式 (10) により定義される．

x_i はそれぞれが得た性能であり，本稿の例においては各 TCP コネクションが得たスループットを表す．n は公平度の集計に含まれる要素の数である．本稿の例においては TCP コネクションの数となり，Linux 機および Windows 機が各 2 台ずつあり各機で 10 本のコネクションが確立されている例では $n = 40$ となる．

Fairness Index は 0 より大きく 1 以下の値をとり，1 に近いほど公平となる．$n = 2$, $x_0 = 2$, $x_1 = 1$ の例で Fairness Index は 0.9 となる，

A.2 CUBIC TCP のスロースタート閾値

標準設定の $\beta = 0.2$ においてスロースタート閾値が輻輳検出時の輻輳ウィンドウサイズ W_{max} の 0.8 倍となることを述べる．2.2 節の式 (1) より，輻輳検出時の $t = 0$ における輻輳ウィンドウサイズは $-C \cdot K^3 + W_{max}$ であることが分かる．式 (2) より $K = \sqrt[3]{\beta \cdot W_{max}/C}$ であるため，$t = 0$ における輻輳ウィンドウサイズは $-C \cdot \beta \cdot W_{max}/C + W_{max} = (1 - \beta)W_{max}$ となる．よって，$\beta = 0.2$ においては輻輳検出直後の輻輳ウィンドウサイズは輻輳検出時の輻輳ウィンドウサイズ W_{max} の 0.8 倍となる．

推薦文

DICOMO2016 において著者らは複数の TCP 輻輳制御アルゴリズム間での公平性を効果的に改善する方法を提案し，特に高い評価を得た．TCP 公平性は非常に重要なテーマであり，今後の進展が期待できるため，本稿を推薦する．

（インターネットと運用技術研究会主査　山井成良）

花井 雅人　（学生会員）

2016 年工学院大学工学部情報通信工学科卒業．同年より同大学大学院工学研究科電気・電子工学専攻に所属．TCP 通信の性能，TCP 公平性，遅延制御手法に関する研究に従事．

山口 実靖 （正会員）

2002 年東京大学大学院工学系研究科電子情報工学専攻博士課程修了．博士（工学）．同年より東京大学生産技術研究所学術研究支援員，産学官連携研究員，日本学術振興会特別研究員．2006 年工学院大学工学部講師．2007 年同大学同学部准教授．通信プロトコル，オペレーティングシステム，I/O 高速化の研究に従事．電子情報通信学会，日本データベース学会各会員．

小林 亜樹 （正会員）

工学院大学情報学部情報通信工学科准教授．2000 年東京工業大学大学院博士後期課程修了．博士（工学）．主に，情報推薦，ネットワーク情報検索，画像認識，インタラクティブシステムの研究に従事．電子情報通信学会，日本データベース学会等各会員．