社団法人 電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE

Reno との親和性を考慮した TCP Vegas の改善とその応用

山口 一郎　　甲藤 二郎

早稲田大学大学院理工学研究科

〒169-8555 東京都新宿区大久保 3-4-1

E-mail:　{yamaguchi, katto}@katto.comm.waseda.ac.jp

あらまし

現在、TCP の標準として Reno が広く利用されているが、Reno よりも高いスループットが得られる方式として TCP Vegas が提案されている。しかし、Reno と混在したネットワーク環境においてはスループットが大幅に低下してしまうという問題が発生する。そこで本研究では、Reno とのスループット親和性を目的として Vegas の輻輳回避フェーズにおけるウィンドウサイズ更新アルゴリズムを改善し、Vegas アルゴリズムにおけるパラメータの自動化を行うことにより、不公平性を解消する。さらに、シミュレーションによって提案手法の有効性を定量的に明らかにし、提案手法の応用について示す。

キーワード　輻輳制御、TCP Reno、TCP Vegas、親和性、ラウンドトリップタイム

Performance Improvement of TCP Vegas with Reno Friendliness and Its Applications

Ichiro YAMAGUCHI　　and Jiro KATTO

Graduate School of Science and Engneering, Waseda University

3-4-1 Ohkubo, Shinjuku-ku, Tokyo, 169-8555 Japan

E-mail:　{yamaguchi,katto}@katto.comm.waseda.ac.jp

Abstract

TCP Reno is widely used in the current Internet as a de facto standard. On the other hand, TCP Vegas, which is able to achieve higher throughput than Reno, has been proposed. However, it has been also indicated that Vegas can't achieve comparative performance when Reno and Vegas co-exit. In this paper, we improve the window update algorithm of TCP Vegas at its congestion avoidance phase. In this method, Vegas's parameters are automatically updated and the throughput unfairness is expected to be solved, leading to Reno friendliness. Furthermore, we show effectiveness of the proposal by computer simulations.

Keyword　Congestion Control, TCP Reno, TCP Vegas, Reno Friendliness, RTT

1. はじめに

現在、インターネット上を流れるトラヒックの大部分は TCP(Transmission Control Protocol)であり、電子メール、WWW(World Wide Web)、ファイル転送など様々なアプリケーションで利用されている。最近では、ネットワークのブロードバンド化、無線インターネットなどインフラが急速に整備されてきているが、今後もトランスポート層のプロトコルとして TCP が利用されることが考えられる。

TCP の輻輳制御については、様々な研究が行われている。最初は、スロースタートアルゴリズム、輻輳回避アルゴリズム、Fast Retransmit アルゴリズムを採用した TCP Tahoe が提案されている。しかし、ロス検知するとウィンドウサイズを大きく下げてしまうため、スループットが低下してしまうという問題が生じる。そこで、Fast Recovery アルゴリズムを採用した TCP Reno が提案された。Fast Recovery アルゴリズムは、輻輳の度合いが小さい場合に転送速度を大きく下げないようにすることを目的としている。それに対して、1994 年、L.Brakmo らによって TCP Vegas[1]が提案されている。TCP Tahoe や TCP Reno と違い RTT を輻輳の指標としてウィンドウサイズの制御を行っている。また、[1]では TCP Reno よりも 37～71%改善されると言われており、今後、TCP Vegas が普及していくことが期待される。ところが、TCP Reno と TCP Vegas が混在した

環境において Vegas のスループットが低下し、本来の性能を示さないということが指摘されている[2,3]。そこで、両者の不公平性を改善した提案として、Reno と Vegas を動的に切り替える Vegas+[4]が提案されている。

一方、本研究では Reno と親和性を持ったスループットが得られるように Vegas のウィンドウサイズ更新アルゴリズムの改善を行う。本提案では、Vegas パラメータの自動化を行うことで、不公平性を解消する。提案手法の有効性は、ネットワークシミュレータ NS-2[7]を用いて示す。

以下、2 章では TCP の輻輳制御アルゴリズムについて説明し、3 章では、提案手法について説明し、4 章では、提案手法の有効性をシミュレーションによって示し、その応用についても検討する。最後に 5 章でまとめを行う。

2. TCP の輻輳制御アルゴリズム

TCP は輻輳ウィンドウ($cwnd$)を動的に変更し、転送レートを調節している。ここでは、TCP Tahoe/Reno および TCP Vegas の輻輳制御アルゴリズムについて説明する。

2.1. TCP Tahoe/Reno

TCP Reno/Tahoe は、ACK を受信する毎に以下のようにウィンドウサイズ($cwnd$)を変更していく。

$$cwnd = \begin{cases} [Slow\ Start] \\ cwnd + 1 \quad (cwnd < ssthresh) \\ \\ [Congetion\ Avoidance] \\ cwnd + 1/cwnd \quad (ssthresh < cwnd) \end{cases} \tag{1}$$

ただし、 $ssthresh$(*Slow Start Threshold*)は、スロースタートの動作から輻輳回避の動作に移行するための閾値である。コネクション開始時には、スロースタートでウィンドウサイズを指数関数的に増加させていき、輻輳ウィンドウ($cwnd$)が $ssthresh$ を越えると輻輳回避の動作で線形的にウィンドウサイズを増加させていく。このようにウィンドウサイズを更新している最中に送信側がセグメントのロスを検知すると、 $ssthresh$ を以下のように変更する。

$$ssthresh = \frac{cwnd}{2} \tag{2}$$

セグメントのロス検知法は、2 種類あって 1 つ目は、Duplicate ACK(重複 ACK)の受信によるロスの検知(Fast Retransmission)で、2 つ目は、タイムアウトによるロス検知である。

タイムアウトによるロス検知の場合は、Tahoe、Reno ともウィンドウサイズを 1 に戻して Slow Start を開始するが、Fast Retransmission によるロス検知の場合、両者で相違がある。Tahoe の場合は、タイムアウトの時と同様にウィンドウサイズを 1 に戻して Slow Start を開始するが、TCP Reno の場合は、ウィンドウサイズをロス検知直前の半分にする。その後、再送セグメントに対する ACK が受信されるまで、Duplicate ACK を受信している間はウィンドウサイズを増加させる。

2.2. TCP Vegas

Tahoe/Reno のウィンドウサイズ制御では、セグメントロスが発生するまでウィンドウサイズを増加させ、その後、ネットワークの輻輳状態を考慮せずにウィンドウサイズを指数的に下げるため、転送効率が悪くなる場合がある。

それに対して TCP Vegas は、RTT に基づいた輻輳制御を行っている。TCP Vegas では以下のようにウィンドウサイズを更新していく。

$$cwnd = \begin{cases} [Slow\ Start] \\ cwnd + 1/2 \\ \\ [Congestion\ Avoidance] \\ cwnd + 1 \quad (Diff < \alpha/baseRTT) \\ cwnd \quad (\alpha/baseRTT < Diff < \beta/baseRTT) \\ cwnd - 1 \quad (\beta/baseRTT < Diff) \end{cases} \tag{3}$$

ただし、 $Diff = cwnd/baseRTT - cwnd/RTT$ である。また、 $baseRTT$ は最小の RTT で、 $\alpha, \beta(\alpha < \beta)$ は定数である。

スロースタートでは Tahoe/Reno の場合の半分の割合でウィンドウサイズを増加させる。輻輳回避の動作になると、式(3)より、RTT が増加するとウィンドウサイズを減少させ、逆に RTT が減少するとウィンドウサイズを増加させる。理想的な状態になると、 $\alpha/baseRTT < Diff < \beta/baseRTT$ となり、ウィンドウサイズが安定する。これによって無駄なセグメントロスを防ぎ、かつ安定したスループットが得られる。

3. Reno との親和性を考慮した TCP Vegas の改善

3.1. 従来手法での問題点

Reno と Vegas が混在した環境において Vegas のスループットが低下することが指摘されているが、その原因について簡単に説明する。Reno では、式(1)より、セグメントのロスを検知するまで一方的にウィンドウサイズを増加させる。従って、ボトルネックとなるルータにおけるバッファをオーバーフローさせてしまう。従って、各コネクションの RTT が増大し、Vegas コネクションは、式(3)のウィンドウサイズを減少させる動作を行ってスループットが低下する。

3.2. パラメータ α , β について

式(3)の $Diff$ は、コネクションがはられたパス上での最大転送レートと現在のレートとの差をあらわしている。よって、 $Diff \times baseRTT$ は、Vegas コネクションがネットワーク中に滞留させるセグメント数に等しくなる。従ってスループットを安定させることは、ネットワーク中に滞留しているセグメント数を安定させることと同等である。そのセグメント数を以下に示す。

$$\alpha < Diff \times baseRTT < \beta \tag{4}$$

Vegas では、ネットワーク中の滞留セグメント数を大きくしないように α , β を設定する。[1]では α =1, β =3 とされている。

よって、原理的にはα,βの値を大きくすれば、Renoと公平なスループットを得られるようになるが、そのチューニングは、動的に変化するネットワークの輻輳に対して、困難であると考えられる。

3.3. 提案方式

前節で述べたように、パラメータα,βの設定を適切に行えば、Renoと Vegasが混在した状況においてスループットの公平性が保たれる。

そこで本提案では、ネットワークの輻輳状態に応じてパラメータα,βを動的に変更する。Vegasコネクションのパス上に滞留している平均セグメント数 N を推測し、Vegas コネクションのネットワーク中に滞留するパケット数を N に近づける。それによって Renoと混在した状況において Vegas の著しいスループットの低下を解消し、Renoと同等のスループットを得られるようにする。 N は以下の式で表される。

$$N = cwnd - cwnd \cdot \frac{baseRTT}{aveRTT} \qquad (5)$$

$$aveRTT = (1 - w_{RTT}) \cdot aveRTT + w_{RTT} \cdot RTT$$

$aveRTT$ は平均 RTT であり、w_{RTT}は重み係数である。従って本提案では、輻輳回避で以下のようなウィンドウサイズの制御を行う。これは、RTT を平均 RTT に漸近させる制御に等価である。

$$cwnd = \begin{cases} cwnd + 1 & (Diff < N/baseRTT) \\ cwnd - 1 & (N/baseRTT < Diff) \end{cases} \qquad (6)$$

4. シミュレーション評価

NS-2[7](Network Simulator version2)を使って、従来方式と提案方式の性能評価を行う。

4.1. シミュレーション条件

図 1 のようなネットワークトポロジを用いる。左側の S1,S2 は送信側で D1,D2 は受信側である。また、中央の R1,R2 は、ルータである。バッファでのパケット廃棄制御は Drop-Tail とする。

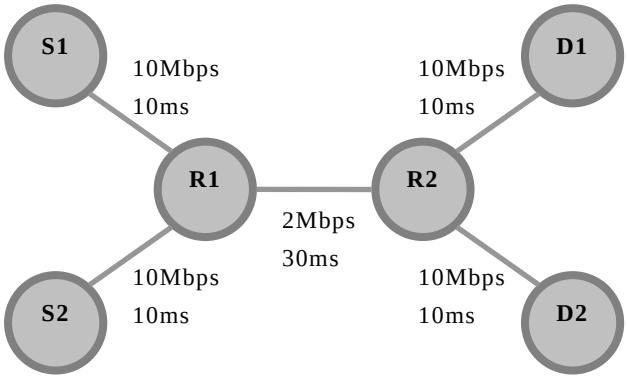

図 1.ネットワークトポロジ

4.2. TCP Reno との親和性

ここでは、Vegas および提案方式と Reno のコネクション開始時間を変えてシミュレーションを行った結果を示す。Vegas のパラメータ設定としては、α=1,β=3とした[1]。

4.2.1. 条件 1

図 2、図 3 は、最初に Vegas および提案方式のコネクションを開始してから、50 秒後に Reno のコネクションを開始したときのスループットである。

図 2 より、Reno のコネクションが開始されると、Vegasのスループットが急激に低下していることがわかる。Reno のコネクションが開始されると、ボトルネックルータのバッファをオーバーフローさせてしまうため、Vegas コネクションの RTT が増加していく。そうすると式(3)におけるウィンドウサイズを減少させ、スループットが低下してしまう。

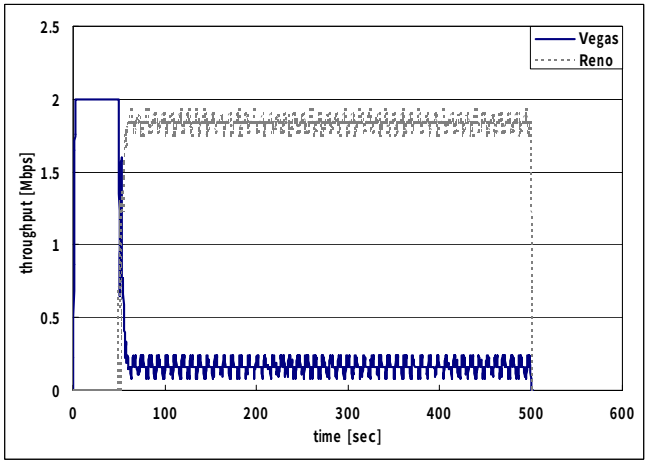

図 2. Vegas vs. Reno (条件 1)

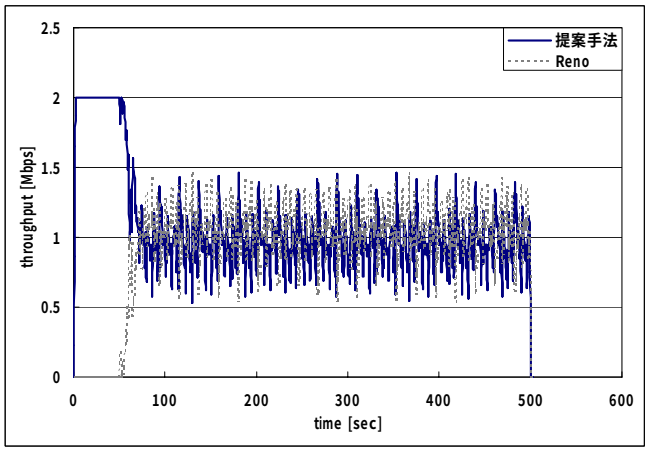

図 3. 提案方式 vs. Reno (条件 1)

それに対して図 3 の提案方式では、スループットが振動していて安定していないものの、平均としては親和性のあるスループットが得られている。Reno コネクションが開始されると $aveRTT$ が増加するため、N が増加する。すると式(6)よりウィンドウサイズが増加しやすくなるが、ある程度ウィンドウサイズを増加させると N によってその増加は抑えられる。そのため Reno と親和性のあるスループットが得られる。

4.2.2. 条件 2

図 4、図 5 には、最初に Reno コネクションを開始し、50 秒後に Vegas および提案方式のコネクションを開始したときのスループットの比較である。

図 4 では、Vegas のスループットは Reno のスループットよりも著しく低下している。ただし、この場合、Reno コネクション開始後に、TCP Vegas のコネクションが開始されるので、条件 1 の時よりも *baseRTT* が大きくなる。図 2 と図 4 を比較すると図 4 における Vegas のスループットが大きくなっているのが分かる。

それに対して、図 5 の提案方式では条件 1 の時と同様に振動の小さなスループットは得られていないが、Reno と親和性のある結果が得られている。

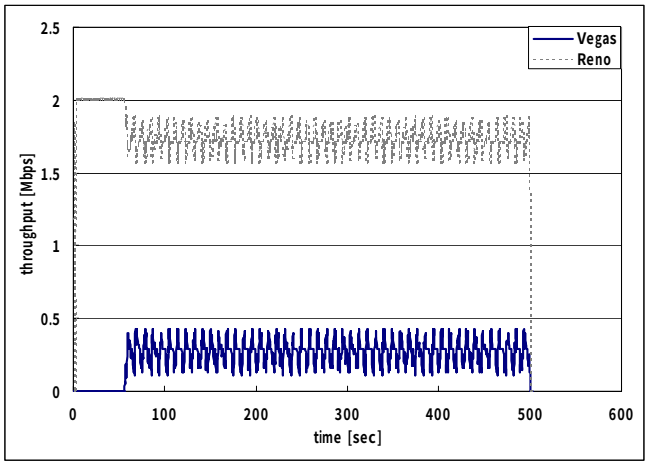

図 4. Vegas vs. Reno (条件 2)

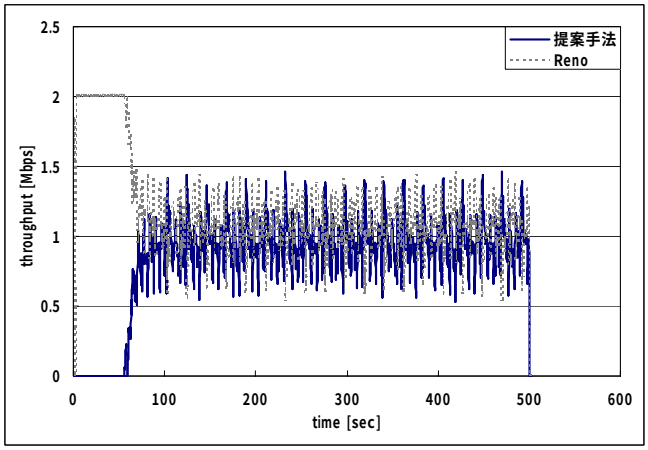

図 5. 提案方式 vs. Reno (条件 2)

4.2.3. 条件 3

図 6、図 7 は、最初に Vegas および提案方式のコネクションを開始し、100 秒後に Reno のコネクションを開始して、300 秒後に Reno のコネクションを停止したときのスループットである。

図 6 より、Reno コネクションが開始されると Vegas のスループットが急激に低下している。300 秒後に Reno のコネクションが停止すると、RTT が減少し、スループットがあがる。Vegas のスループットが急激に低下する原因は条件 1 の時と同じである。また、Vegas のコネクションのみの状態になると、コネクション開始時のスループットに回復する。これは、Reno のコネクションがなくなることで RTT が減少し、式(3)でウィンドウサイズを増加させる動作に入るためである。

それに対して、図 7 の提案方式では Reno のコネクションが開始されると、条件 1 の時のように N が増加することにより、ウィンドウサイズが増加し、親和性のあるスループットが得られる。提案方式のコネクションのみの状態になると Vegas の時と同様にスループットが回復する。

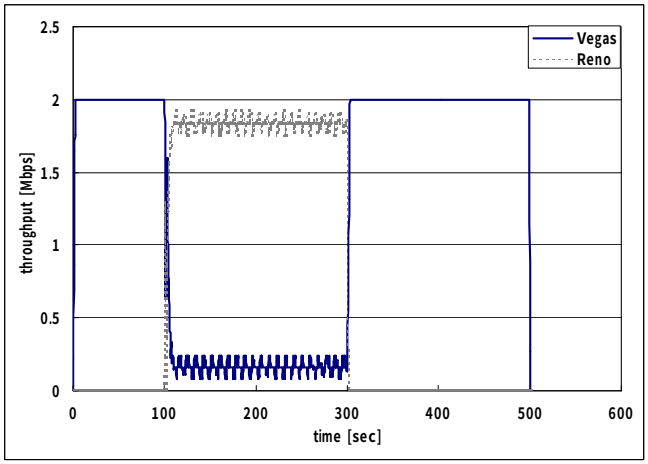

図 6. Vegas vs. Reno (条件 3)

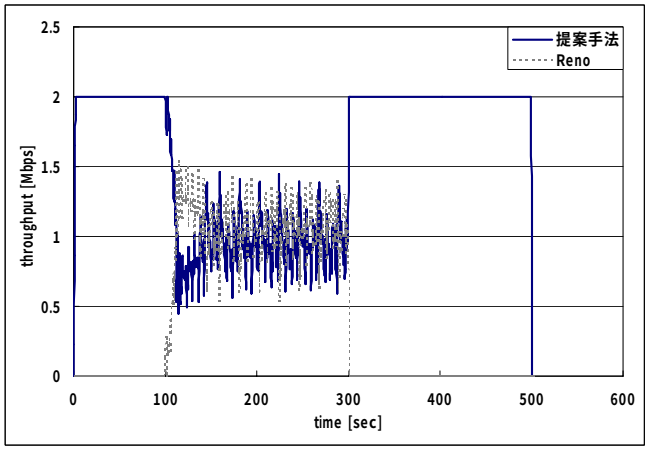

図 7. 提案方式 vs. Reno (条件 3)

4.3. 提案手法同士の親和性

次に、提案手法同士のコネクションが混在した場合、どのようになるか評価を行う。図 8、図 9 にスループットの結果を示す。

4.3.1. 条件 1

図 8 では、最初に 1 つ目のコネクションを開始して、50 秒後に 2 つ目のコネクションを開始している。

Vegas のコネクションが混在した環境においては、公平性の問題が[3]で指摘されているが、図 8 において両者のスループットを比較すると、200,270 秒付近で差が見られるものの、両者のスループットに親和性が認められる。

4.3.2. 条件 2

図 9 では、最初に 1 つ目のコネクションを開始して、

100 秒後に 2 つ目のコネクションを開始し､300 秒後に 2 つ目のコネクションを停止させている。
条件 1 のときと同様に 2 つのコネクションが競合しているとき、親和性が認められる。また、2 つ目のコネクションを停止させた後、1 つ目のコネクションは開始時のスループットに回復している。

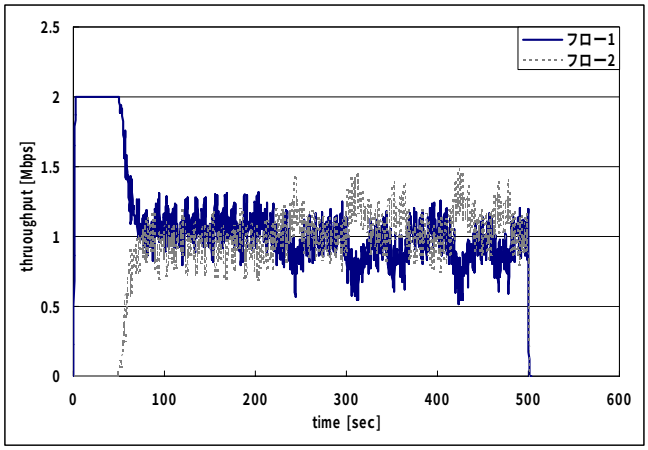

図 8. 提案方式 vs. 提案方式 (条件 1)

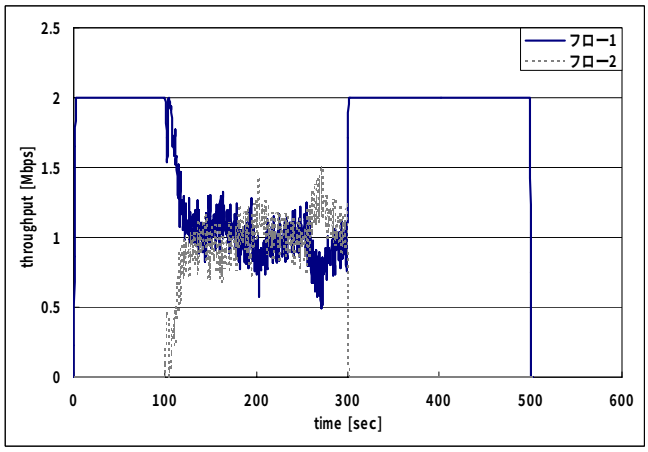

図 9. 提案方式 vs. 提案方式 (条件 2)

5. 提案手法の応用について

ここでは、提案方式の応用について述べる。1 つ目は、リアルタイムアプリケーションに対する応用で、2 つ目は、無線環境への適応性である。

5.1. リアルタイムアプリケーションへの対応

リアルタイムアプリケーションは、一般的にトランスポート層のプロトコルとして UDP を用いている。しかし、UDP にはレート制御機能はなく、TCP の通信を妨げることが指摘されている。そこで、TCP との親和性を考慮した TCP Friendly Rate Control[5]が提案されている。

TCP Vegas は、高いスループットが得られることと同時に、安定した RTT で通信が行える輻輳制御方式である。従って現在広く使われている Reno と公平性のあるスループットさえ得られれば、RTT の変動が小さい、すなわちジッタが小さく抑えられることなり、プレイアウト遅延 (ジッタ吸収時間) が小さく抑えられる。よって、Vegas 的なレート制御機能は、リアルタイムアプリケーションに向いていると考えられる。

以下に、提案方式と Reno(4.2.1 での条件)、提案方式同士(4.3.1 での条件)のフローが競合しているときの提案方式の RTT 変動の様子を示す。

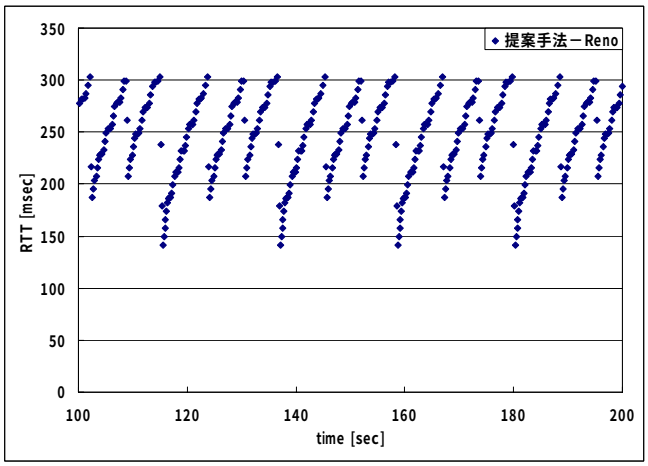

図 10. RTT 変動の様子 (提案方式 vs. Reno)

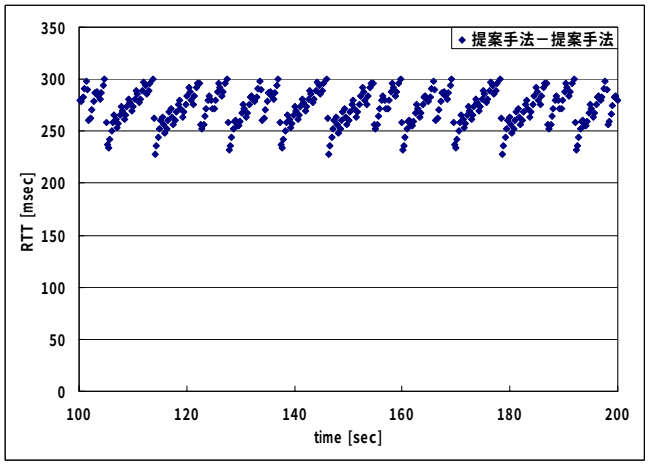

図 11. RTT 変動の様子 (提案方式 vs. 提案方式)

図 11 では、150[msec]以上の RTT 変動が見受けられるが、図 12 では、数十[msec]程度の変動に抑えられている。提案方式と Rneo が競合した場合、Reno はセグメントのロスを検知するまでウィンドウサイズを増加させ、ロスを検知するとウィンドウサイズを半分にする。このため提案方式の RTT が、提案方式どうしのフローが競合した場合に比べて変動が大きくなる。しかし、提案方式の TCP が主流になれば、リアルタイムアプリケーションで提案方式のレート制御機能を利用することで、既存の問題点は解消されるのではないかと考えられる。

5.2. 無線環境への適応

既存の TCP は、ネットワークの輻輳によるセグメントロスか、無線環境におけるビット誤りによるセグメントロスを判断することができない。従って、無線環境で TCP Reno をそのまま用いると、無線リンクにおけるビット誤りによるセグメントロスを輻輳と判断してしまい、ウィンドウサイズを半分にしてしまう。そ

れに対して Vegas は式(3)のようにウィンドウサイズを制御しているため、Reno のように急激にウィンドウサイズを減少させることはない。従って、Vegas と Reno の公平なスループットを実現できれば、無線環境への適応性は Reno よりも高いと考えられる。そこで、提案方式を無線環境で利用した場合の評価を行う。

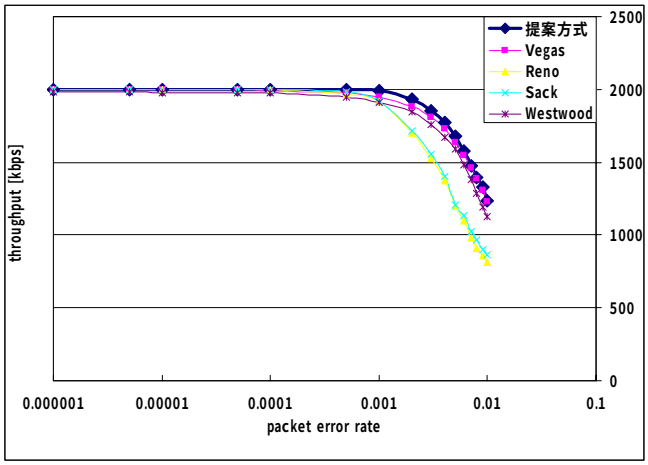

図 12.PER とスループットの関係 (輻輳なし)

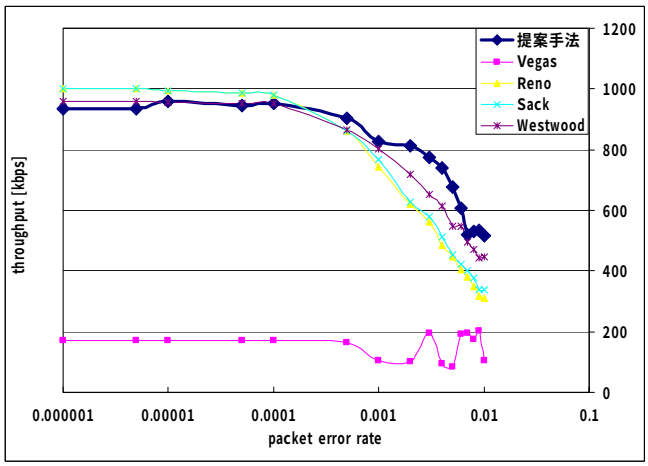

図 13.PER とスループットの関係 (輻輳あり)

図 1 のネットワークにおいて、ルータと受信者との間を無線リンクとしてシミュレーションを行った。輻輳がある場合と、ない場合における結果を以下に示す。

TCP のバージョンとしては、提案方式, Vegas, Reno, Sack, Westwood を用いた。

図 12、図 13 を見ると、輻輳がない場合もある場合も提案手法の結果がよいことが分かる。また、パケットエラー率が 10^{-3} を越えると、無線リンクに起因するスループットの低下が顕著になっている。

輻輳がない場合 (図 12)、提案方式と Vegas は、ほぼ同じスループットが得られている。その次に高いスループットが得られているのは Westwood で、これは無線環境に適した TCP と言われている。この Westwood では、セグメントがロスしたときに返される重複 ACK に対して、重複 ACK の到着間隔を利用して利用可能帯域を見積り、それにあわせたウィンドウサイズの変更を行っている(Faster Recovery)[6]。よって、急激なウィンドウサイズの削減は行なわれず、Reno や Sack よりも高いスループットが得られていると考えられる。Sack は、同一ウィンドウ内で複数のセグメントロスが同時に起こった場合に有効で、エラー率が高い場合において Reno よりも高いスループットが得られている。

輻輳がある場合は (図 13)、輻輳を起こさせるフローとして Reno コネクションを用いた。タイムアウトの影響もあり、輻輳がない場合に比べてスループットは安定していないが、この場合も提案方式のスループットが一番高い。Vegas は、Reno との公平性が取れないため著しくスループットが低下している。Westwood では、Reno(輻輳用のフロー)が、ボトルネックリンクに負荷をかけているため、重複 ACK で見積もる利用可能帯域が、輻輳がない場合に比べて小さくなる。Sack と Reno では、輻輳がない時と同様に、急激なウィンドウサイズの削減がスループットの低下を招いている。

6. まとめ

Reno との親和性を目的して Vegas のウィンドウサイズ制御方式の改善を行った。シミュレーションによって定量的に評価を行い、提案方式の有効性を示した。また、提案方式の応用についても検討を行った。

参考文献

[1] L. S. Brakmo, S. W.O'Mally, and L. L. Peterson, "TCP Vegas: New techniques for congestion detection and avoidance," Proc. ACM SIGCOMM'94, pp. 24-35, October 1994.

[2] J. Mo, R. J. La, V. Anantharam and J. Walrand, "Analysis and comparison of TCP Reno and Vegas," Proc IEEE INFOCOM'99, March 1999.

[3] 倉田謙二,長谷川剛,村田正幸, " TCP Reno と TCP Vegas の混在環境における公平性の評価, " 電子情報通信学会技術報告(SSE99-98),pp.67-72,November 1999

[4] 長谷川剛,倉田謙二,村田正幸, " バージョン間の公平性を考慮した TCP Vegas の改善方式, "電子情報通信学会技術報告(SSE2000-31),pp.1-6 May 2000.

[5] S. Floyd, M. Handley, J. Padhye and J.Widmer, "Equation-Based Congestion Control for Unicast Applications," Proc ACM SIGCOMM2000

[6] C. Casetti, M. Gerla, S. Mascolo, M. Y. Sanadidi and R.Wang, "TCP Westwood: Bandwidth Estimation for Enhanced Transport over Wireless Links," Proc ACM Mobicom 2001, pp.287-297, Rome, Italy, July, 16-21 2001

[7] "Network Simulator-ns(version2)," http://www.isi.edu/nsnam/ns, Ucb/LBNL/VINT.