マルチメディア通信と分散処理 88－11
（1998. 4. 24）

広域通信におけるTCPの高速化を目的とした輻輳回避手順の提案

三宅 優† 神崎 昭浩‡ 加藤 聰彦† 鈴木 健二†

KDD研究所†
電気通信大学‡

インターネットに使用される伝送路が高速化されるに従い、大容量データを高速で転送する必要があるマルチメディア・アプリケーションが今後も増えると予想される。インターネットで広く使用されているトランスポートプロトコルであるTCPは、確実なデータ転送を必要とするほとんどの通信アプリケーションで使用されているが、高速・広域ネットワークの通信でTCPを使用して通信を行うと、TCPの輻輳回避アルゴリズムが不適切であるために、帯域に見合った通信速度が得られないという問題が存在する。そこで筆者らは、ウィンドウスケールオプションと選択確認(SACK)オプションを有するTCPにおいて、輻輳回避アルゴリズムの改良により通信効率を向上させることを可能としたので、本稿では、そのアルゴリズムの詳細と、シミュレーションによる比較検討結果について述べる。

A Study of Congestion Control Algorithm for High Speed TCP Communication on Wide Area Network

Yutaka Miyake† Akihiro Kanzaki‡ Toshihiko Kato† Kenji Suzuki†

KDD R&D Laboratories†
The University of Electro-Communications‡

Multimedia applications will require high speed transfer of large size data along with increasing of transmission speed for internet link. Almost applications for communication use TCP as transport protocol for reliable data transfer within the internet widely. However, TCP cannot use the whole bandwidth on wide area and high speed network, because a congestion avoidance algorithm used for TCP is unsuitable for this kind of network. In this paper, we propose a new algorithm of the congestion avoidance for TCP with window scale and SACK (Selective Acknowledgment) option to accelerate communication speed, and evaluate this algorithm using simulation results.

1 はじめに

インターネットでは、確実なデータ転送を行うためのトランスポートプロトコルとして、TCP [1]が広く使用されている。TCPには、フロー制御の最大ウィンドウサイズが64Kバイトであり、また、紛失したセグメント以降のすべてのセグメントを再送するgo-back-Nに基づく誤り再回復手順を採用しているために、遠隔地との間に高速な伝送路を用意しても、帯域

に見合った通信速度が得られないという問題が存在した。これに対し、最大ウィンドウサイズを1Gバイトまで拡張可能なウィンドウスケールオプション [2] と、紛失したセグメントセグメントのみを再送する選択確認 (SACK: Selective Acknowledgment) オプション [3] が提案されている。

しかし、高速なネットワークを利用した遠隔地との通信では、これらのオプションを導入しただけでは、以下の理由により、高速なデータ通信が困難となる場合がある。フロー制御に使用するウィンドウサイズを拡大した状態で通信する場合は、連続して送信可能なセグメント数が増えるために、通信経路上のルータ等でバッファがオーバフローを起こして、セグメントを紛失する確率が高くなる。そこで、セグメントの紛失を抑制するために、往復遅延時間の変動を計測したり、ネットワーク上に存在するセグメント数を予測することによりネットワークの状態を推定し、セグメントの送出量を調整するアルゴリズムが提案されている [4–6]。これらのアルゴリズムを使用しても、他のトラフィックの影響等によりネットワークの状況が急激に変わり、セグメントの紛失が発生する確率は依然として高い。このようなネットワークの輻輳によりセグメントが紛失すると、輻輳制御機能が動作して送信セグメント数を一旦減少させ、その後に、確認応答セグメントの受信により徐々に送信セグメント数を増やしていく。しかし、現在の TCP で使用されている送信セグメントを増加させるアルゴリズムは、高速でも遅延の少ない LAN や、伝送速度が遅い回線を使用した遠隔地との通信に適するように設計されており、伝送遅延が大きい遠隔地との高速ネットワークを介した通信では、送信セグメント数が増えるのに時間を要してしまう。これにより、提案されているオプションを実装した TCP による通信でも、通信速度が低下してしまう。

本稿では、ウィンドウスケールオプションと選択確認オプションを実装した TCP において、伝送遅延が大きい遠隔地と高速ネットワークを介して通信した場合に、ネットワークの帯域を有効に使用するための輻輳回避アルゴリズムを提案している。本方式は、セグメントを紛失した時点を基準として、その時点での送信量に急激に近づけることにより、従来型の欠点を補っている。また、他の従来型の TCP によるコネクションや提案アルゴリズムを採用した TCP のコネクションが混在する場合でも、輻輳回避アルゴリズムの違いにより、特定のコネクションだけが著しく送信量を減らさないための対処も行っている。このアルゴリズムの有効性を確認するために、シミュレーションによる従来方式との比較を行なったので、その結果についても報告する。

2 広域・高速通信路における TCP 通信

2.1 TCP の輻輳制御機能

TCP では、セグメントが紛失した場合に、それを輻輳によるものと判断して送信側が自発的に送信するデータ量を制御する。送信側は、フロー制御で使用されるウィンドウとは別に、輻輳ウィンドウ (Congestion Window) と呼ばれる変数 (cwnd) を導入し、これらのウィンドウサイズの小さい方の値の範囲内でしか確認応答が届いていないセグメントを送信することができない。

現行の TCP では、以下のような手順で cwnd の値を更新している。(以下では、cwnd を最大セグメントサイズ単位で計算している。)

- コネクション確立直後は、cwnd を 1 に初期化し、受信側からの確認応答セグメントの受信毎に (1) 式により指数関数的に増加させる (Slow Start)。

$$\mathrm{cwnd} \leftarrow \mathrm{cwnd} + 1 \tag{1}$$

- 送信したセグメントに対する確認応答が到着しないためにタイムアウトが発生すると、その時点の cwnd の $\frac{1}{2}$ を変数 ssthresh に格納し、cwnd を 1 とする。その後、確認応答セグメントを受信する毎に、(1) 式により cwnd を増加する、すなわち Slow Start を行なう。
- Slow Start により増加している cwnd の値が ssthresh を越えると、確認応答セグメントを受信する毎に、(2) 式により cwnd を増加させる (Congestion Avoidance)。

$$\mathrm{cwnd} \leftarrow \mathrm{cwnd} + \frac{1}{\mathrm{cwnd}} \tag{2}$$

- 受信側から送られてくる確認応答セグメントに同じ確認応答番号が含まれているものを連続して 4 セグメント受信したときには、その確認応答番号を含むセグメントの次のセグメントが紛失したと判断して、その時点の cwnd の $\frac{1}{2}$ を新たな cwnd として、Congestion Avoidance の処理を行なう (Fast Retransmit & Fast Recovery)。

2.2 輻輳制御機能の問題点

伝送遅延が大きい遠隔地との高速ネットワークを介した通信で、ネットワークの帯域を有効に利用し

て通信を行なうには、大きなウィンドウサイズが必要となり、必然的に cwnd の値も大きくなる。cwnd が大きい状態でセグメント紛失が発生すると、輻輳を防ぐために cwnd を縮小してしまい、著しくスループットが低下する。そして、Congestion Avoidance により徐々に cwnd の値を増加させるが、ウィンドウスケールオプションによりウィンドウサイズが拡大されても従来と同じ方法であるため、もとの大きな値まで増加するのに時間がかかり、ネットワークの帯域を有効に利用できなくなってしまう。これにより、ウィンドウスケールオプションと SACK オプションを実装した TCP でも、ネットワークの伝送帯域に比べてスループットが低下してしまうのである。

3 輻輳回避アルゴリズムの提案

3.1 基本方針

選択確認オプションを実装した TCP では、Fast Retransmit とそれに続く選択再送により誤り回復が行われることが多く、タイムアウトによる再送の発生頻度は少ない。したがって、輻輳制御として Congestion Avoidance が使用される。そこで、ネットワークの伝送帯域の使用効率を高めるために、現在使用されている Congestion Avoidance に代わる新しい輻輳回避アルゴリズムを考案する。

前述のように、Congestion Avoidance の問題点は、伝送遅延が大きい通信において cwnd の増加速度が相対的に遅くなってしまうことである。帯域の使用効率を高めるには、cwnd を長時間、最適値と思われる値で維持すればよい。これを達成するためには、以下の方法が考えられる。

1. セグメント紛失の検出により cwnd が $\frac{1}{2}$ に縮小した (ここまでは、Congestion Avoidance と同様) 後に、最適値と考えられる値まで急速に拡大させ、その値を保持する。
2. 最適値を保持している場合には、必要以上に cwnd を縮小させない。

輻輳回避アルゴリズムは、高いスループットを得られると同時に、既存の TCP も含めて複数のコネクションが通信経路上に存在した場合に、他のコネクションのスループットをむやみに低下させ過ぎないことも必要である。既存の TCP では、セグメント紛失の検出を輻輳と判断し、cwnd を $\frac{1}{2}$ にして送信データ量を減らす。このようなコネクションが存在するときに後者の方法で通信を行うと、cwnd を縮小させないため、既存の TCP が cwnd を縮小させて空けた帯域を即座に確保してしまう。これが繰り返して発生するため、既存の TCP は徐々に cwnd を縮小させて通信速度が低下するのに対し、cwnd を縮小させない TCP は一方的に通信速度が向上する。このようなコネクション間の不公平を生じさせるアルゴリズムは、現在のインターネット環境に導入すべきでないと考えられる。

そこで、本稿では、前者の方法を輻輳回避のアルゴリズムのベースとして採用した。すなわち、セグメントの紛失を検出すると cwnd を $\frac{1}{2}$ に縮小し、その後、cwnd を最適値と考えられる値まで急速に拡大させている。cwnd の最適値としては、ネットワークの状況が時々刻々と変化すると考えられるため、前回セグメントが紛失した時点での cwnd の値を使用する。

3.2 輻輳回避アルゴリズム

本稿で提案する輻輳回避アルゴリズムも、cwnd を操作することにより、輻輳時のデータ送信量を調整する。本アルゴリズムは、以下の事柄を考慮して値を操作している。

- **セグメント紛失直後の cwnd の拡大は、緩やかにする。**
  cwnd を大きく変化させた直後はネットワークの状況が不明であるため、急な変化は危険である。
- **前回セグメントが紛失した時点の cwnd の値付近では、拡大を緩やかにする。**
  最適値としている前回セグメントが紛失したときの cwnd の値付近では、cwnd の拡大を緩やかにする。そして、緩やかな拡大においてセグメントの紛失が無い場合には、さらに拡大する。
- **急すぎる拡大は行わない。**
  拡大が急すぎると、セグメントの紛失と再送が頻発して、逆にスループットが低下する可能性がある。

これらを勘案し、筆者らが提案する輻輳回避アルゴリズムは、Congestion Avoidance で使用されている (2) 式の代りに、以下の式を使用することとした。

$$\mathrm{cwnd} \leftarrow \mathrm{cwnd} + \underbrace{\frac{\left|\mathrm{cwnd} - \frac{\mathrm{c_peak}}{2}\right|}{\mathrm{cwnd}}}_{\mathrm{A}} \times \underbrace{\frac{|\mathrm{c_peak} - \mathrm{cwnd}|}{\mathrm{cwnd}}}_{\mathrm{B}} \times \underbrace{\frac{\frac{\mathrm{c_peak}}{2}}{\mathrm{cwnd}}}_{\mathrm{C}} + \underbrace{\frac{1}{\mathrm{cwnd}}}_{\mathrm{D}} \qquad (3)$$

この式において新たに導入された変数 c_peak には、セグメントの紛失を検出した時点での cwnd が保存されている。

(3) 式は、(2) 式に対して A 項、B 項、C 項の積が追加されている。これらの各項および D 項は、以下のような意味を持っている。

A 項: セグメント紛失を検出して Fast Recovery の手順により cwnd の値を半分に縮小した直後の急激な拡大を防ぐ。

B 項: cwnd が拡大する過程で c_peak 付近に近づいたときに、cwnd の値の拡大を緩和する。

C 項: cwnd の拡大率を調整するための係数。cwnd が大きくなるにしたがってセグメントが紛失する可能性が高くなると考えられるため、それに合わせて拡大率を下げる。

D 項: 既存の Congestion Avoidance にも存在するが、ここでは、再送開始時および c_peak の値に等しくなった時点で、cwnd の増加量が 0 になることを防いでいる。

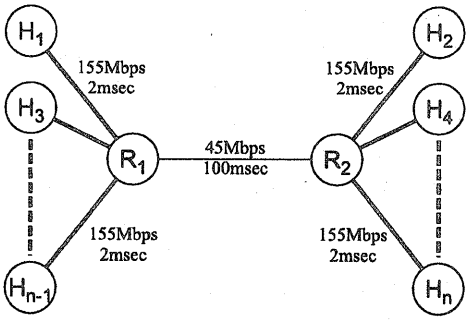
4 性能評価

本稿で提案しているアルゴリズムを評価するために、ネットワークシミュレータである ns (version 2) [7] を用いた性能評価を行なった。ns は、カルフォルニア大バークレイ校およびローレンスバークレー国立研究所により開発され、無線ネットワーク上での TCP の性能評価 [8]、SACK オプションを実装した TCP の性能評価 [9]、TCP の改良に関する検討 [5]、等、TCP/IP に関するプロトコル研究に幅広く利用されている。

評価は、図 1 のように、ルータ間に伝送遅延の大きなボトルネックリンクが存在するネットワーク構成を使用し、ホスト間で 30 秒間のデータ転送をしたときの通信スループットを測定した。このシミュレーションでは、ヘッダを含めたセグメントサイズは 1500 バイト、ルータのバッファは 50 セグメント (73.2K バイト) とした。

4.1 1 コネクション

ホスト H1 とホスト H2 間で、1 本のコネクションのみが存在する場合について、スループットを測定した。測定した TCP は、SACK オプションなし (TCP)、SACK オプションあり (SACK-TCP)、提案型 TCP の 3 種類である。結果を表 1 に示す。表において、maxcwnd は、ウィンドウサイズ (セグメント単位) を示している。また、図 2 は、maxcwnd が 512 セグメントの時の、各 TCP の cwnd の時間的変動を示している。



H_n: Host　　Segment Size = 1500 byte
R1, R2: Router　　Queue Size = 50 × 1500 byte

図 1: シミュレーションに使用したネットワーク構成

表 1: 1 コネクション時のスループット

maxcwnd	Throughput (Mbps) 128	256	512	1024	2048
TCP	6.55	4.02	2.78	2.78	2.78
SACK-TCP	6.55	3.10	8.70	8.70	8.70
提案型 TCP	6.55	7.96	18.71	12.01	20.14

通信開始時は、Slow Start により急激に cwnd の値が増加しているが、これによりルータのバッファがオーバーフローし、パケットの紛失が連続して発生している。その結果、TCP では、タイムアウト発生後に Slow Start が行なわれ、SACK-TCP では、Slow Start は回避しているが、いずれの場合も Congestion Avoidance による cwnd の増加が線形で緩やかなものであるため、適切なウィンドウサイズで通信が行なえず、スループットが低下する原因となっている。これに対し、提案型 TCP では、再送終了後に cwnd を急速に拡大しているため、帯域を効率よく使用することができ、スループットが向上した。

4.2 複数コネクション

ボトルネックとなるリンクに、同時に複数のコネクションが存在して、データの伝送が行なわれたときのスループットを測定した。表 2 は、ボトルネックリンクの両側に位置する 4 対のホストが、同じ種類のプロトコルにより同時にコネクションを確立してデータを転送を行なったときの各コネクションのスループットと、全コネクション合計のスループットを示している。測定においては、各コネクションの通信開始時間が同じにならないように、1 秒づつずらしてある。図 3 は、提案型 TCP におけるウィンドウサイズが 512 セグメントの時の各コネクション毎の cwnd の時間的変動である。

TCP では、タイムアウトによる再送が発生して Slow Start が起動するために、スループットが低下

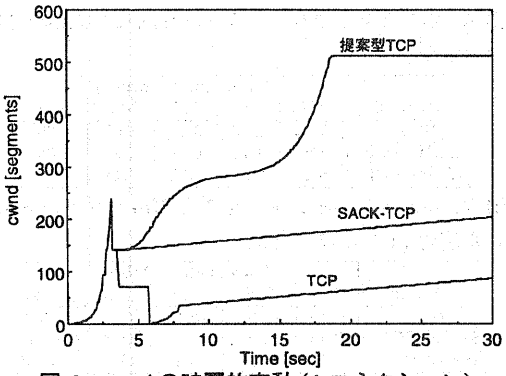


図 2: cwnd の時間的変動 (1 コネクション)

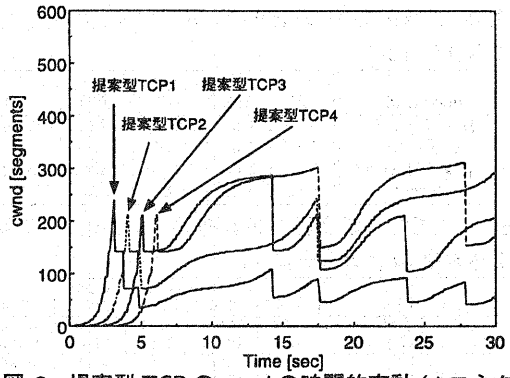


図 3: 提案型 TCP の cwnd の時間的変動 (4 コネクション)

表 2: 4 コネクション時のスループット

maxcwnd	Throughput (Mbps) 128	256	512	1024
TCP1	6.51	4.00	2.77	2.77
TCP2	6.21	3.75	2.59	2.59
TCP3	6.00	3.54	2.44	2.44
TCP4	5.77	3.34	2.29	2.29
TCP Total	24.49	14.63	10.09	10.09
SACK-TCP1	6.51	4.71	3.45	3.45
SACK-TCP2	6.21	4.50	4.94	4.94
SACK-TCP3	6.00	4.30	7.87	7.87
SACK-TCP4	5.77	4.09	7.54	7.54
SACK-TCP Total	24.49	17.60	23.80	23.80
提案型 TCP1	6.51	4.37	3.51	3.51
提案型 TCP2	6.21	5.74	8.47	8.47
提案型 TCP3	6.00	6.86	8.38	8.38
提案型 TCP4	5.77	5.98	10.49	10.49
提案型 TCP Total	24.49	22.95	30.85	30.85

してしまう。SACK-TCP と提案型 TCP のスループットの差は、cwnd の拡大速度の差によるものである。なお、SACK-TCP と提案型 TCP において、maxcwnd が 512 セグメント以上の場合に各コネクション間のスループットに差が生じたのは、通信開始時のセグメント紛失のタイミングによるものである。

表 3 は、ボトルネックリンクの両側に位置する 4 対のホストが、異なる種類のプロトコルが混在する状態で同時にコネクションを確立してデータを転送したときの各コネクションのスループットを示している。表 2 の場合と同じく、通信開示時間が 1 秒づつずれており、表 3 の最初の計測では、TCP1、SACK-TCP1、SACK-TCP2、TCP2 の順に通信が開始されている。

提案型 TCP は他の種類の TCP が混在する場合においても高い通信スループットを達成するとともに、表 2 の計測結果と比べて、他の種類の TCP に対しても著しく不利になるような影響を与えないことが確認された。

表 3: 4 コネクション時のスループット

maxcwnd	Throughput (Mbps) 128	256	512	1024
TCP1	6.51	3.99	2.76	2.76
SACK-TCP1	6.21	4.50	8.22	8.22
SACK-TCP2	6.00	4.30	7.88	7.88
TCP2	5.77	3.34	2.29	2.29
TCP1	6.51	3.96	1.78	2.09
提案型 TCP1	6.21	9.01	12.57	12.60
提案型 TCP2	6.00	8.55	12.43	11.29
TCP2	5.77	3.34	2.24	2.26
SACK-TCP1	6.51	4.71	2.67	2.67
提案型 TCP1	6.21	9.01	9.01	9.01
提案型 TCP2	6.00	8.54	12.28	12.28
SACK-TCP2	5.77	4.09	5.63	5.63

5 考察

5.1 cwnd の拡大率の調整について

C 項は、ABD 項の拡大速度を調整する役割を受け持っている。そこで、C 項を変更することにより cwnd の拡大速度を変化させたときの、1 コネクション時の TCP によるスループットを計測した。その結果を表 4 に示す。C 項を除去した場合には、スループットが僅かに高くなるが、ほぼ同程度の速度である。拡大速度を緩めるために、C 項を (c_peak/4)/cwnd とした場合は、25%程度のスループット低下が見られた。1/cwnd とした場合には、SACK-TCP とほぼ同様の結果となり、C 項の拡大速度を遅くするとスループットが低下することが確認された。

表 5 は、C 項を除去した提案型 TCP が、他の種類の TCP と同時にデータ転送をしたときのスループットを示している。この場合、SACK オプションがない TCP と同時に通信を行なうと、ウィンドウサイズが 512 セグメント以上において、TCP のスループットが極端に低下する。つまり、C 項の導入により、他の TCP トラフィックに与える影響が軽減されている。

これらにより、C 項の式が適切なものであること

表 4: C 項を変更したときのスループットの変化 (1 コネクション)

maxcwnd	Throughput (Mbps) 128	256	512	1024	2048
提案型 TCP	6.55	7.96	18.71	12.01	20.14
C 項なし	6.55	9.39	7.76	15.31	20.34
(c_peak/4)/cwnd	6.55	6.03	14.82	15.02	15.02
1/cwnd	6.55	3.20	8.81	8.81	8.81

表 5: C 項なし提案型 TCP が他のコネクションに与える影響 (2 コネクション)

maxcwnd	Throughput (Mbps) 128	256	512	1024	2048
C 項なし提案型 TCP	6.52	9.36	8.39	13.85	13.85
C 項なし提案型 TCP	6.11	10.79	7.12	12.26	12.26
TCP	6.52	4.00	1.11	0.88	0.88
C 項なし提案型 TCP	6.11	10.79	6.17	16.32	16.32
SACK-TCP	6.52	3.07	8.01	7.41	7.41
C 項なし提案型 TCP	6.11	10.79	6.34	16.24	16.24

が確認された。

5.2 ネットワーク条件が異なる状況下での通信性能

4 における性能評価で使用したネットワーク構成とは異なる構成でシミュレーションを行ない、通信性能の差を調べた。

まず、ボトルネックリンクの帯域を 10Mbps に変更した場合の性能評価を行なった。その結果を、表 6 に示す。提案型 TCP は、他の TCP と比べて、常に高いスループットを達成しており、ボトルネックリンクの帯域に依存することなく、高い通信性能が得られることが確認された。

次に、ルータのバッファサイズを 50 セグメントから 200 セグメントに変更した場合の通信性能を、表 7 に示す。この場合も、最大バッファサイズが 1024 セグメントのときに、スループットが若干低下するものの、提案型 TCP は全体的に高い通信スループットを達成している。

上記のように、提案型 TCP は、ネットワークの構成に依存することなく、従来の TCP に比べてほとんどの場合において高い通信性能を発揮する。

6 おわりに

本稿は、伝送遅延の大きな 2 地点間を高速な伝送路を使用して TCP により通信を行なう場合において、既存の TCP における輻輳制御に関する問題点を明らかにすると共に、ウィンドウスケールオプションと SACK オプションを持つ TCP において、この問題を解決する輻輳制御のためのアルゴリズムを提案して、シミュレーションによりその有効性を示した。

提案したアルゴリズムは、セグメント紛失により輻輳ウィンドウサイズを減少させた後、そのウィンドウサイズを最適と判断した値まで急速に復帰させることにより、ネットワークの帯域を有効に利用した通信を実現する。また、このアルゴリズムを実装した TCP の通信が、他の TCP の通信に著しい影響を与えないことも考慮されている。シミュレーションの結果、提案したアルゴリズムが有効に動作することを確認するとともに、他の TCP のトラフィックとの共存が可能であることも明確となった。TCP に提案したアルゴリズムを実装することにより、高速・広域ネットワーク上においても、大容量データ転送が高速化されると考えられる。

表 6: ボトルネックリンクを 10Mbps にした場合 (1 コネクション)

maxcwnd	Throughput (Mbps) 128	256	512	1024	2048
TCP	6.45	3.90	2.69	2.69	2.69
SACK-TCP	6.45	4.71	5.09	5.09	5.09
提案型 TCP	6.52	5.93	7.11	7.11	7.11

表 7: ルータのバッファサイズを 200 セグメントにした場合 (1 コネクション)

maxcwnd	Throughput (Mbps) 128	256	512	1024	2048
TCP	6.55	12.91	25.42	18.02	18.02
SACK-TCP	6.55	12.91	25.42	14.48	26.47
提案型 TCP	6.52	12.91	25.42	15.31	33.33

参考文献

[1] J. Postel, "Transmission Control Protocol – Protocol Specification," RFC793, Sept. 1981.

[2] J. Jacobson, R. Braden, and D. Borman, "TCP Extensions for High Performance," RFC1323, May 1992.

[3] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow, "TCP Selective Acknowledgment Options," RFC2018, Oct. 1996.

[4] L. Brakmo, and L. Peterson, "TCP Vegas: End to End Congestion Avoidance on a Global Internet," IEEE Journal on Selected Areas in Communications, vol.13, No.8, Oct. 1995.

[5] M. Mathis, and J. Mahdavi, "Forward Acknowledgment: Refining TCP Congestion Control," ACM SIGCOMM, vol.26, No.4, pp.281–291, Oct. 1996.

[6] 釘本健司、天海良治、"帯域見積りに基づく輻輳回避アルゴリズム、" 情報処理学会論文誌、vol.39、No.2、pp.228–238、Feb. 1998.

[7] K. Fall, and K. Varadhan, "*ns* Notes and Documentation," http://www.isi.edu/~kannan/ns-Doc.ps.gz, Nov. 1997.

[8] B. Bakshi, P. Krishna, D. Pradhan, and N. Vaidya, "Performance of TCP over Wireless Networks," 17th Intl. Conf. on Distributed Computing Systems (ICDCS), May 1997.

[9] K. Fall, and S. Floyd, "Simulation-based Comparisons of Tahoe, Reno, and SACK TCP," Computer Communications Review, Vol.26, No.3, pp.5–21, July 1996.