広帯域無線アクセスむけデータ送信量自動調整方法の提案

五十嵐 健[1] 山崎 憲一[1]

Flight Size Auto Tuning for High Speed Wireless Access Networks

Ken Igarashi[2] Kenichi Yamazaki[2]

概要 インターネットの発展と共に, あらゆる場所からインターネット上の各種サービスを利用したいという要求が高まり, 様々な広帯域無線アクセス技術が開発されている. この様な無線ネットワークによって提供される広い帯域幅を効率的に利用するためには, 有線ネットワークむけに作られた TCP 輻輳制御の改良が必要となる. 本稿では, 広帯域無線ネットワークむけデータ送信量自動調整方法として Flight Size Auto Tuning (FS-AT) を提案する. FS-AT はネットワーク現在の帯域幅遅延積を推定し, 推定した帯域幅遅延積と帯域幅遅延積の変動に基づいてセグメントを送信することによって, 過度なデータ送信によるネットワークの輻輳を防止する. さらに, セグメントロスの原因を推定し推定結果に基づいて輻輳制御を行うか否かを判断する. 輻輳制御を行う場合にはネットワークの現在の帯域幅遅延積に基づいて再送閾値を設定することによってセグメントロス回復後の帯域幅を迅速に利用可能にする. 我々は FS-AT を Linux および NS-2 上に実装して効果の検証を行った.

1 はじめに

インターネットの発展と共に, インターネット上の様々なマルチメディアサービスをいつでも利用したいという要求が高まり, その実現にむけて Wide Area Network (WAN) 向けの様々な広帯域無線アクセス技術が開発されてきた. 3GPP や 3GPP2 においては High Speed Downlink Packet Access (HSDPA) [1] や 1xEvolution Data Only (1xEV-DO) [2] 等が標準化され, より広帯域を実現する無線アクセス技術として Long Term Evolution (LTE) [3] の標準化が進められている.

TCP を用いた通信において, この様な無線アクセス技術によって提供される広い帯域幅を使い切るためには, 輻輳ウィンドウをネットワークの帯域幅遅延積 (BDP: Bandwidth Delay Product) に合わせる必要がある. 輻輳ウィンドウの上限は, 受信側から送信側に送信される Acknowledgement (ACK) に含まれる広告ウィンドウによって決定されるため, Window Scale Option [4] を用いて, 送信側に大きな広告ウィンドウを通知する必要がある. 輻輳ウィンドウを BDP に合わせることによって帯域幅を有効利用できる一方, 無線環境の悪化などによってスループットが低下した場合, ネットワークへ過度にセグメントが送信される可能性を生じる.

図 1 に 3G ネットワークの一例を示す. ネットワークへ流入する TCP コネクションは透過プロキシである

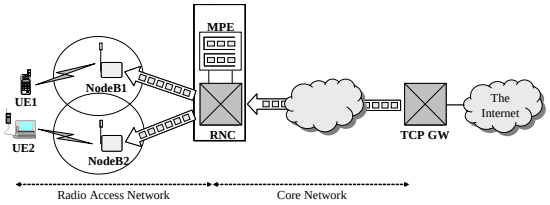

図 1: 3G ネットワーク例.

TCP Gateway (GW) によって, 無線アクセスに合わせた変換 [5] が行われる. コアネットワークに比べ, 無線アクセスの帯域幅は非常に小さいため, 3G ネットワークにおいては多くの場合 Radio Network Controller (RNC) がボトルネックとなる. そのため送信されたセグメントは, RNC が装備している Multimedia signal Processing Equipment (MPE) へ滞留する. Wireless WAN (WWAN) においては Hybrid Automatic Repeat reQuest (ARQ) [6] など User Equipment (UE) 毎に制御を必要とする機能があるため, セグメントは UE 毎に MPE に滞留する. MPE にはコアネットワークと無線アクセスの速度差を吸収するために, 比較的大きなバッファ量が用意されているため, バッファ量の不足によるセグメントロスが発生しにくい.

現在の TCP ではセグメントロスを検出して初めて輻輳制御が行われるため, セグメントロスが発生しない場合, MPE には (1) 式で示す様に, 広告ウィンドウ (snd_wnd) からスループット (th) × Round Trip Time (RTT (rtt)) を引いた分のセグメントが滞留する.

[1]株式会社 NTT ドコモ
[2]NTT DoCoMo, Inc.

$$q = snd\_wnd - th \times rtt \quad (1)$$

MPE にセグメントが滞留した場合, 次の様な問題が引き起こされる.

[**アプリケーションの応答性の劣化**]**:** FTP や HTTP 等の非リアルタイムアプリケーションによる通信を行っている最中に Telnet 等のリアルタイムアプリケーションを利用した場合, 滞留したセグメントによって RTT が増加するため, リアルタイムアプリケーションの応答性が劣化する.

[**アプリケーションの停止**]**:** 滞留したセグメントによって RTT が増加した際に無線環境の悪化等によってセグメントロスが発生した場合, ロストしたセグメントを回復するまでに長い時間が必要となる. その間は TCP の Head-of-Line Blocking によって, ビデオストリーム等においてはアプリケーションが停止した様に見える.

[**不要なセグメントの転送**]**:** 無線リンクのコネクションを切断せずにユーザがアプリケーションを途中で終了した場合, MPE に滞留しているセグメントは受信側に送信されてしまうため, これらの無駄なセグメントの送信によって, 無線アクセスの帯域が浪費されてしまう.

現在, セグメントロス以外の情報を用いて輻輳制御を行う様々な方法 [7] [10] [11] [13] が提案されている. しかし, 広帯域 WWAN のネットワーク特性は有線ネットワークとは大きく異なるため [12], これらの輻輳制御を広帯域 WWAN においてそのまま利用することは難しい. そこで, 本稿では, 広帯域 WWAN において, 送信するセグメント量を自動調整する Flight Size Auto Tuning (FS-AT) を提案する. FS-AT は送信側のみの制御で実現され, 受信側から返される ACK から現在の BDP を推定し, 推定した BDP 及び BDP の変動に基づいて過度なセグメントが送信されないように輻輳制御を行う. また, セグメントロス回復後の帯域幅を迅速に利用するため, セグメントロスの原因が無線環境の悪化もしくはネットワークの輻輳によるものなのかを判断して輻輳制御を行うか否かを決定し, ネットワークの輻輳によるものだと判断した場合には, セグメントロスの直前に推定された BDP に基づいてセグメントロス後の Slow Start Threshold (*snd_ssthresh*) を設定することによって, セグメントロス回復後の帯域幅を迅速に利用可能にする.

本稿の構成は次の通りである. 2 章では関連研究を紹介する. 3 章では FS-AT の動作概要を説明し, 4 章において FS-AT の評価結果を紹介する. そして, 5 章においてまとめ及び今後の課題について述べる.

2 関連研究

セグメントロス以外の情報を用いて輻輳制御を行う方法としては, 推定した帯域幅を用いる TCP Paced-Start [7] や, RTT の変動を利用する TCP Vegas [10] や FAST TCP [11] が存在する. しかし, 広帯域 WWAN は次の様に, 有線ネットワークとは異なる特徴を持つため, これらの技術では適切な輻輳制御を行うことはできない.

[**セグメント間隔が保持されない**]**:** リンクレイヤにおいて, ロストしたセグメントの再送を行う ARQ が働いた場合, TCP では正確にセグメントの到着間隔を計ることができない.

[**帯域幅の変動**]**:** 無線品質に応じて変調方式を変えてセグメントを送信する Adaptive Modulation Coding (AMC) によって帯域幅が大きく変動する.

[**無線環境の悪化によるセグメントロス**]**:** 無線環境が悪化した場合, ネットワークが輻輳していないにも係わらず, セグメントロスが発生する.

2.1 帯域推定を利用した輻輳制御

TCP Paced-Start [7] は通信開始時に packet pair [8] を用いてボトルネックリンクの帯域幅を推定して, 推定に基づいて *snd_ssthresh* を設定する. ボトルネックリンクの帯域幅 (*bw*) は, 推定に用いたパケットのサイズ (*p_len*) と, 推定に用いた 2 つのパケットの到着間隔 (*gap*) を用いて (2) 式の様に推定される.

$$bw = \frac{p\_len}{gap} \quad (2)$$

ARQ によってリンクレイヤでセグメントが再送された場合, ARQ は TCP 同様 In-Sequence Delivery を実現するためにロストしたセグメントが回復されるまで, 以降のセグメントを TCP に送信しない. そのため, ロストしたセグメントが回復した場合 TCP へ連続してセグメントが到着し, *gap* が小さく計測されるため, *bw* が大きく推定され適切な輻輳制御が行われない [9].

2.2 RTT の変動を利用した輻輳制御

TCP Vegas では通信中の RTT の変動を利用して輻輳制御を行うため, セグメント間隔が保持されないという影響を小さくすることができる. TCP Vegas では次の様にして輻輳制御を行う.

- 輻輳ウィンドウ (*snd_cwnd*) と 通信中に計測された最小の RTT (*rtt_min*) から理想状態のスループット (*expected*) を計算する.

  $expected = \frac{snd\_cwnd}{rtt\_min}$

- 直近の 1RTT 間に計測された最小の RTT (*rtt*) と *snd_cwnd* から実際のスループット (*actual*) を計算する.

  $actual = \frac{snd\_cwnd}{rtt}$

- 計算された *expected* と *actual* の差と *rtt_min* からルータに滞留しているセグメント数 (*diff*) を計算する.

  $diff = (expected - actual) \times rtt\_min$

- TCP Vegas では計算された *diff* に従って, *n+1* 番目の輻輳ウィンドウサイズ (snd_cwnd_{n+1}) を (3) 式の様にして決定する.

$$snd\_cwnd_{n+1} = \begin{cases} snd\_cwnd_n + 1 & diff < \alpha \\ snd\_cwnd_n - 1 & diff > \beta \\ snd\_cwnd_n & otherwise \end{cases} \quad (3)$$

TCP Vegas では, ルータに滞留するセグメントを α〜β にするよう輻輳制御が行われる. しかし, 広帯域 WWAN においては帯域幅が大きく変動するため, 帯域幅が急激に増加した場合, ルータに滞留しているセグメントが少ないためスループットが低下する可能性がある.

HSDPA においては UE から通知される Channel Quality Indicator (CQI) に基づいて約 10.0ms 毎に変調方式を変えることができるため, 10.0ms 毎に帯域幅が大きく変動する可能性がある. 一方, TCP Vegas では, 1RTT 毎に計算される *diff* に基づいて輻輳制御が行われるため, この間に急激に帯域幅が増加した場合には, 増加した帯域幅はルータに滞留しているセグメントのみが利用する. そのため, セグメントの滞留量は帯域幅の変動を考慮して決定される必要があるが, TCP Vegas では α 及び β には定数が用いられているため, α 及び β が帯域変動に対して十分ではない場合には, 急激に増加した帯域幅を利用しできない. また, 輻輳ウィンドウが 1RTT 毎に 1 セグメントづつしか増減されないため, 帯域幅の変動への適応速度が遅い. さらに, セグメントロス発生時には TCP の輻輳制御によってネットワークに送信されているセグメント数[1] の半分が *snd_ssthresh* に設定される. 無線環境の悪化によるセグメントロスの場合, セグメントロス回復後に迅速にスループットを回復させる必要がある. そのためには BDP の 2 倍 のセグメントをネットワーク送信しておく必要がある. しかし, TCP Vegas における α や β が小さい場合には, BDP の 2 倍のセグメントをネットワークに送信することができないため, *snd_ssthresh* が小さく設定されてしまい, セグメントロス回復後の帯域幅を有効に活用することができない.

[1]送信済のセグメントであり, ACK もしくは Selective ACK (SACK) [14] によって受信側への到着が確認されていないセグメント.

2.3 Dynamic Right-Sizing

セグメントの滞留量を変化させる方式として Dynamic Right-Sizing (DRS) [13] が存在する. DRS はセグメントの滞留量を BDP に応じて決定する. DRS は当初, 受信側の受信バッファサイズを自動調整する目的で提案された技術であったが, 受信バッファサイズを調整することによって広告ウィンドウサイズが調整されるため, セグメントの送信量を調整する技術としても利用することができる. DRS の概要は次の通りである.

- 受信側において, BDP を推定するため, 通信中の最小の RTT (*rtt_min*) の間に, アプリケーションが TCP から受け取ったデータ量 (*rcv_data*) を計測する.
- 受信側では, 計測された最大の *rcv_data* に TCP/IP ヘッダ等を付加した合計バイト数の 2 倍に受信バッファを調整する.
- バッファサイズが調整された結果, 計測された最大の *rcv_data* の 2 倍が広告ウィンドウへ設定される.

DRS は TCP Vegas 同様, 通信中の *rcv_data* に基づいてセグメントの送信量を決定するため, セグメント間隔が保持されないという影響を小さくすることができる. DRS では広告ウィンドウサイズが *rcv_data* に合わせて調整されるため, 送信側の輻輳ウィンドウに空きがある場合においても広告ウィンドウの制限によって過度なセグメントの送信を防止することができる. 広告ウィンドウサイズには *rcv_data* によって推定される BDP の 2 倍が設定されるため, BDP 分のセグメントがネットワークに滞留する. そのため, ネットワークの帯域幅が増加すると共にネットワークに滞留するセグメント数が増加し, 帯域幅の急激な増加に対応することが可能となる. しかし, DRS には次の様な問題がある.

第 1 の問題は, ネットワークに常に BDP 分のセグメントを滞留させてしまうことである. 広帯域 WWAN の様に帯域幅が急に変動したり, 無線環境の悪化によるセグメントロスを生じる場合には BDP 分のセグメントを滞留させることは有効であるが, 帯域幅の変動が小さく, 無線環境の悪化によるセグメントロスが起こらないネットワークにおいては, 過剰なセグメント送信となる.

第 2 の問題は, 受信側では RTT を正確に計測できないことである. 受信側では, ACK を返してから ACK + 広告ウィンドウサイズ分のセグメントを受信するまでの時間を 1RTT とする. しかし, データセグメントが Application

Limited [15] や輻輳ウィンドウの制限によって ACK の到着後すぐに送信されない場合, RTT が本来の値よりも長く計測され *rtt_min* が長くなる．その結果 *rcv_data* が本来の値よりも大きくなり, DRS の制御で意図した値よりも多くのデータが送信されてしまう．

第 3 の問題は, TCP の特別な輻輳制御が広告ウィンドウの制限によって働かなくなる事である．例えば, 3G の様な無線ネットワークにおいて無駄な再送を防ぐために用いられる F-RTO [16] では, 重複 ACK 受信時に輻輳ウィンドウサイズを 2 セグメント分増加させて, 新たな 2 つのセグメントを送信可能にする．しかし, DRS によって広告ウィンドウサイズが調整され, 新たなセグメントの送信が妨げられた場合 F-RTO が働かなくなる．

第 4 の問題は, 無線ネットワークの帯域幅が変動することによって *rcv_data* が大きく計測されることである．DRS では最大の *rcv_data* に基づいて広告ウィンドウサイズが決定されるため, 帯域幅の変動が大きな場合, 最大の帯域幅に基づいて計測された *rcv_data* によってセグメントの送信量が調整されてしまう．

3 Flight Size Auto Tuning

FS-AT は DRS 同様 BDP に基づいて送信セグメント量を調整する技術であるが, 常に BDP 分のセグメントを滞留させるのではなく, BDP の変動に合わせて滞留するセグメントを調整する．さらに, ACK を利用して送信側で制御を行うことよって, DRS で問題となった *rtt_min* の計測や F-RTO との連係の問題を解決する．FS-AT は次の様な特徴を持っている．

- ACK から 1RTT の間に受信側に到着したセグメント量を計算することによって, ネットワークの現在の BDP を推定し, 推定した現在の BDP と BDP の変動に合わせてセグメントの送信量を調整する．
- 帯域幅の変動に追従させるために, セグメントの送信量を積極的に増減させる．
- セグメントロス発生時に, セグメントロスの原因 (ネットワークの輻輳または無線環境の悪化) を推定して輻輳制御を行うか否かを決定する．
- セグメントロスを回復した後に, 帯域幅を迅速に利用できるように *snd_ssthresh* を決定する．
- F-RTO の動作を妨げないように制御を行う．

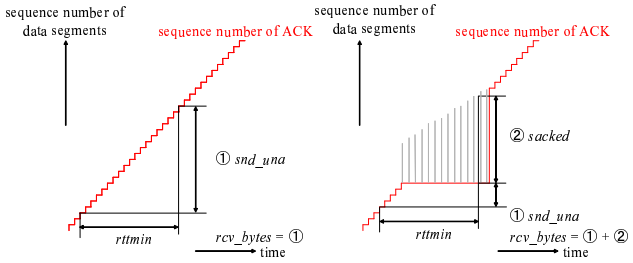

図 2: *rcv_bytes* の計測方法.

3.1 BDP の推定

セグメントの送信量を正確に制御するためには, 帯域幅の変動に合わせて変化する BDP を正確に推定する必要がある．現在の BDP を推定するために FS-AT では *rtt_min* 内に ACK によって受信側への到着が確認されたセグメント量 (*rcv_bytes*) を計測する．*rcv_bytes* は DRS における *rcv_data* とほぼ同様の値ではあるが, *rcv_data* は TCP とアプリケーションの間で計測される値であるため, アプリケーションの読み込み速度が影響するのに対して, *rcv_bytes* はネットワークと TCP の間で計測される値であるため BDP をより正確に計測することができる．

図 2 に *rcv_bytes* の計測方法を示す．通常は *rtt_min* の間に増加した最大の ACK のシーケンス番号 (*snd_una*) から計測される．一方, セグメントロスがあった場合には, BDP をより正確に計測するため *rtt_min* の間に SACK によって受信側への到着が確認されたセグメント数 (*sacked*) も考慮にいれて *rcv_bytes* を計測する[2].

3.2 セグメント送信量の調整

FS-AT では, 新たに *space* という新たな値を導入して輻輳制御を行う．これによって TCP の Congestion Avoidance における輻輳ウィンドウの増加率を変更する新たな TCP 輻輳制御 [17] [21] [18] と組み合わせても利用することが可能になる．FS-AT のセグメントの送信量 (*flight_size*) は, 輻輳ウィンドウ (*snd_cwnd*), 広告ウィンドウ (*snd_wnd*) そして *space* から (4) 式の様に決定される．

$$flight\_size = min(snd\_wnd, snd\_cwnd, space) \quad (4)$$

space の上限は *rcv_bytes* から計算される *target* によって決定される．図 3 を用いて *target* の更新方法について説明する．

judge(1) では *rtt_min* 毎に (5)(6)(7) 式に従って *target* を更新する．

[2]SACK が利用できない場合には重複 ACK の数で代用する．

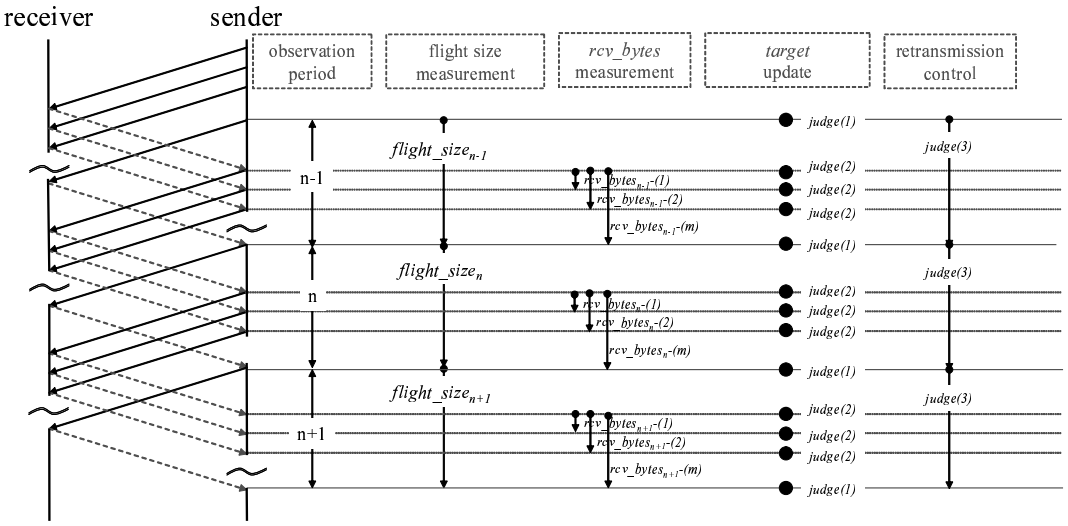

図 3: FS-AT におけるセグメント送信量の調整方法.

$$if\ (flight\_size_n \leq rcv\_bytes_n{-}m) \\ target_{n+1} = snd\_cwnd\_max \tag{5}$$

$$else\ if\ (target_{n+1} < twrcv\_bytes_n{-}m \\ \parallel target_{n+1} == snd\_cwnd\_max) \tag{6}$$

$$target_{n+1} = twrcv\_bytes_n{-}m$$

$$else \\ target_{n+1} = \frac{(twrcv\_bytes_n{-}m + target_n)}{2} \tag{7}$$

(5) 式を満たした場合, 送信したセグメントに対する ACK を *rtt_min* 内に全て受信できているため, ルータにセグメントが滞留していないと判断し, FS-AT による輻輳制御を行わない様 *target* を輻輳ウィンドウがとりうる最大の値 (*snd_cwnd_max*)[3] に設定する.

(6) 式を満たした場合, 送信したセグメントに対する ACK を *rtt_min* 内に全て受信できていないため, ルータにセグメントが滞留していると判断して, FS-AT による輻輳制御を開始する. 既に FS-AT による輻輳制御が行われている場合 (条件式の後半部分), BDP の増加に迅速に追従するため, 計測された *rcv_bytes* に (9) 式から決定される *twrcv_bytes* を設定する.

(7) 式を満たした場合, BDP が減少しているので, セグメントの送信量を減少させる必要がある. 但し, セグメントの送信量を急激に減少させた場合, スループットが低下する可能性があるため, セグメントの送信量を減少させる場合には, *rcv_bytes* の 2 倍と直近の *target* との平均の値を次の *target* に設定する.

judge(2) では, BDP の増加に迅速に追従するために *rtt_min* 期間内であっても (8) 式に従って *target* を更新する.

$$if\ (rcv\_bytes_n{-}(p) \times 2 > target_n) \\ target_n = rcv\_bytes_n{-}(p) \times 2\ (p{=}1, 2, ..., m) \tag{8}$$

twrcv_bytes は DRS と同様 *rcv_bytes* の 2 倍を上限値として, *rcv_bytes* の平均 (*s_rcv_bytes*) と平均偏差 (*dev_rcv_bytes*) を利用して (9) 式の様に決定する. これは TCP における Retransmission Time Out (RTO) タイマの計算方法 [22] と同様の考え方であり, これによって帯域変動の度合に応じて, セグメントの滞留量を調整することが可能となる.

$$dev\_rcv\_bytes = \\ \frac{3}{4} \times dev\_rcv\_bytes + \frac{1}{4} \times |s\_rcv\_bytes - rcv\_bytes| \\ s\_rcv\_bytes = \\ \frac{7}{8} \times s\_rcv\_bytes + \frac{1}{8} \times rcv\_bytes \\ twrcv\_bytes = \\ min(s\_rcv\_bytes + 4 \times dev\_rcv\_bytes, rcv\_bytes \times 2) \tag{9}$$

space は *target* が BDP に応じて急激に変化した場合においても, セグメントをバースト的に送信したり, 長い間セグメントの送信が止まるのを防ぐために段階的に増減する. *target* の増加に伴って, *space* を増加させる場合, 通常の ACK 受信時には RFC3465 [19] に従い, ACK によって受信が確認されたバイト数分 *space* を増加をさせる. ACK が重複 ACK であった場合には, SACK によって受信が確認されたバイト数分 *space* を増加させる. こうすることによって, *target* が急激に増加した場合においてもセグメントがバースト的に送信されることを防ぐことができる. 一方, *target* の減少に伴って *space* を減少させる場合, セグメントの送信が 1RTT の間止まらないようにするために rate-halving [20] と同様, 2 セグメント分の ACK を受信する毎に 1 セグメント分 *space* を減少させる.

[3] *snd_cwnd_max* の値は, 広告ウィンドウサイズ及び送信側の送信バッファサイズから決定される.

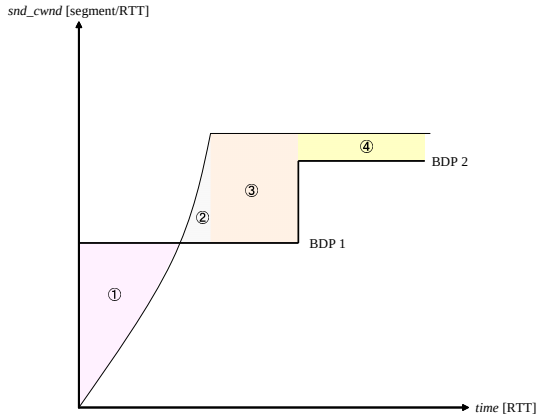

図 4: 輻輳ウィンドウがとりえる状態.

3.3 セグメントロス発生時の輻輳制御

図 4 の ③ の様に, 送信済のセグメント量が BDP の 2 倍に調整されている場合, 無線環境の悪化によってセグメントロスが発生した場合, セグメントロス回復後にスループットを迅速に回復することができる. しかし, FS-AT では 送信済のセグメント量を BDP の 2 倍より小さくする様にセグメントの送信量を調整する. また, ①②④ の様に, セグメントの送信量が BDP の 2 倍になる前にセグメントロスが発生した場合や広告ウィンドウの制限によってセグメントの送信量を BDP の 2 倍にできない場合, セグメントロス回復後の帯域幅を有効に利用することができない.

FS-AT では, 広帯域ネットワーク向けの各種 TCP 輻輳制御 [21] [11] [17] の様に, Congestion Avoidance における輻輳ウィンドウサイズの増加速度を変更するのではなく, セグメントロスの発生原因を判別する共に, セグメントロス発生後の *snd_ssthresh* をセグメントロス発生直前の BPD に基づいて決定することにより, セグメントロス回復後の帯域幅を迅速に利用可能にする. セグメントロスがあった場合には, FS-AT は図 3 の *judge(3)* において (10)(11) 式に従って *snd_ssthresh* を決定する.

$$if\ (flight\_size_n \leq rcv\_bytes_n\text{–}(m))$$
$$snd\_ssthresh_{n+1} = max(snd\_cwnd_n, ssthresh_n) \tag{10}$$

$$if\ (flight\_size_n > rcv\_bytes_n\text{–}(m))$$
$$snd\_ssthresh_{n+1} = max(rcv\_bytes_n\text{–}(m), \frac{flight\_size_n}{2}) \tag{11}$$

セグメントロス発生時に (10) 式を満たしていた場合, 送信したセグメントに対する ACK を *rtt_min* 内に全て受信できているため, ネットワークの輻輳によるセグメントロスではなく, 無線環境の悪化によるセグメントロスと判定し, *snd_ssthresh* を減少しない (図 4 の ①). 一方, セグメントロス発生時に (11) 式を満たしていた場合, ネットワークの輻輳によるセグメントロスと判定して *snd_ssthresh* を減少させる. その際, ネットワークに送信されているセグメント数の半分以外に, セグメントロスの直前の *rcv_bytes* も考慮して *snd_ssthresh* を決定することによって, 図 4 の ② や ④ においても, セグメントロスが回復した後の帯域幅を迅速に利用可能にする.

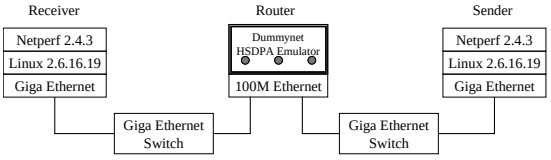

図 5: 有線及び無線ネットワーク.

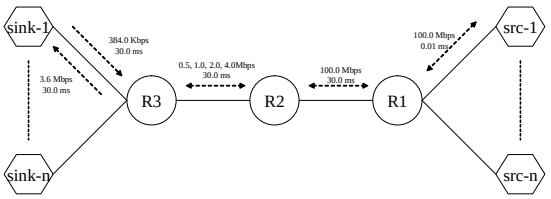

図 6: NS-2 Network.

4 性能評価

Linux 2.6.16.19 [23] 及び NS-2 based Linux TCP [24] 上に FS-AT を実装して評価を行った. Linux を用いた性能評価は, 図 5 に示すネットワークにおいて, 有線ネットワークの場合には Dummynet, 無線ネットワークの場合には HSDPA エミュレータをルータとして用いて行った. そして, ルータにおいて双方向に 90.0ms の遅延を加え RTT を 180.0ms に設定した. NS-2 を用いた評価では, 図 6 に示すネットワークを用いて評価を行った. 比較方式としては Linux 2.6.16.19 カーネル及び NS-2 based Linux TCP に実装されている Reno, DRS そして TCP Vegas[4] を用いた. 広告ウィンドウサイズは 96.0Kbytes とし, FS-AT の制御が最初のセグメントの送信を妨げないように *target* の最小値を 8.0Kbytes とした.

4.1 有線ネットワークにおける性能評価

本シミュレーションでは Dummynet によって帯域幅をそれぞれ 64.0Kbps, 384.0Kbps, 1.0Mbps, 2.0Mbps, 4.0Mbps に設定したネットワークに 10.0Mbytes のデータを送信し

[4] $\alpha = 1$, $\beta = 3$.

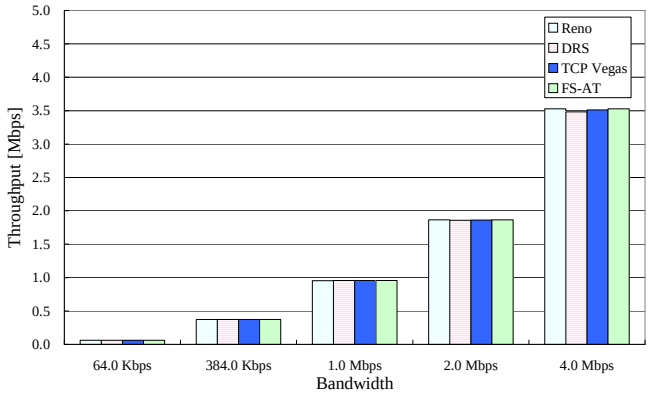

図 7: 有線ネットワークにおけるスループット.

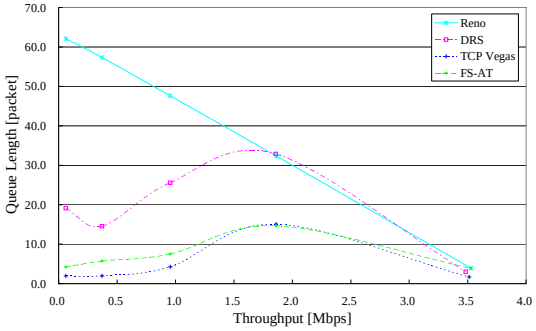

図 8: 有線ネットワークにおけるスループットとセグメントの滞留量.

て評価を行った. 図 7 に各方式のスループットを示す. 有線ネットワークを用いた場合 Reno, TCP Vegas, FS-AT のスループットに大きな違いは見られなかった. 一方 DRS は FS-AT が (5)(8) 式で行っている様なスループットを低下させないための制御を行わないため, ルータにセグメントが滞留する前からセグメントの送信量が調整されてしまい, 通信開始時のスループットが減少する. 今回は 10.0Mbytes のデータを転送しているため, 通信開始時のスループットの低下の影響は相対的に小さくなるが,Reno と比較した場合, 2.0Mbps の場合 0.3%, 4.0Mbps の場合 1.3% スループットが減少した.

図 8 に各方式のスループットに対するセグメントの滞留量示す. Reno においては帯域幅が最も小さい場合 (64.0Kbps) にセグメントの滞留量が最大になる. Reno においては (1) 式に示す様に広告ウィンドウサイズが一定の場合, 帯域幅の増加と共にセグメントの滞留量は減少する.

DRS のセグメントの滞留量は帯域幅に応じて決定されるため, 帯域幅が増加すると共に, セグメントの滞留量も増加する. DRS の制御によってルータに滞留するセグメント量は (12) 式で示すことができる.

$$q = min(th \times rtt, snd\_wnd - th \times rtt) \qquad (12)$$

表 1: 有線ネットワークにおける各方式のセグメント滞留量の最大値.

Bandwidth (BDP [packet])	64.0Kbps (1.0)	384.0Kbps (5.8)	1.0Mbps (15.0)	2.0Mbps (30.0)	4.0Mbps (60.0)
Reno	65.0	61.0	51.0	36.0	28.0
DRS	21.0	17.0	27.0	37.0	23.0
TCP Vegas	15.0	27.0	49.0	36.0	28.0
FS-AT	6.0	13.0	37.0	36.0	28.0

DRS によってセグメントの滞留量が削減できる領域は (13) 式から求めることが可能であり, 広告ウィンドウサイズが 96.0Kbytes の場合 1.4Mbps となり, これ以下の帯域幅では Reno と比較した場合のセグメントの滞留量の削減率は 46.4%〜74.6% となる. なお, 広告ウィンドウサイズの下限値が 32.0Kbytes に設定されているため, 帯域幅が 64.0Kbps の場合のセグメントの滞留量が 384.0Kbps の場合のセグメントの滞留量に比べて大きくなる.

$$th \times 2 \times rtt < snd\_wnd \times 8 - th \times rtt \qquad (13)$$

FS-AT のセグメントの滞留量は帯域幅の変動に応じて決定されるため, 有線ネットワークの様に帯域幅の変動が小さい場合には DRS よりも少なくなり, Reno と比較した場合のセグメントの滞留量の削減率は 55.3%〜78.0% となる.

TCP Vegas は Reno と比較した場合 91.1%〜96.8% セグメントの滞留量を削減できる. しかし, セグメントの滞留量が理論値[5] よりも上回る場合がある. これには FS-AT 及び TCP Vegas がスループットを低下させないために, ネットワークの輻輳状態を検出するまでは制御を行わないことが関係する.

表 1 に各方式のセグメント滞留量の最大値を示す. DRS は通信開始時からセグメントの送信量を調整するため, スループットが低下する半面, セグメント滞留量の最大値は最も小さくなる[6]. 一方, TCP Vegas はスループットを高めるために一つの輻輳ウィンドウ内で送信された複数のセグメントにおける最小の RTT に基づいて *diff* の計算を行う. そのため, 輻輳ウィンドウサイズが BDP を越えた RTT [7] の次の RTT で初めてネットワークの輻輳状態が検出されるので, Slow Start 中は, 最短の場合, 輻輳ウィンドウサイズが BDP の 4 倍, 最長の場合, BDP の 8 倍になって初めて TCP Vegas の制御が行われる. 輻輳ウィンドウサイズから BDP を引いた分のセグメントがルータ

[5] TCP Vegas の場合 α〜β.
[6] セグメント滞留量の最大値とセグメントの滞留量が解離しているのは TCP によるバースト転送の影響によるものである.
[7] このとき輻輳ウィンドウサイズは BDP の 2 倍.

Name	Scenario
St	Standstill
PA3	Pedestrian (3km/h)
VA30	Vehicle (30km/h)

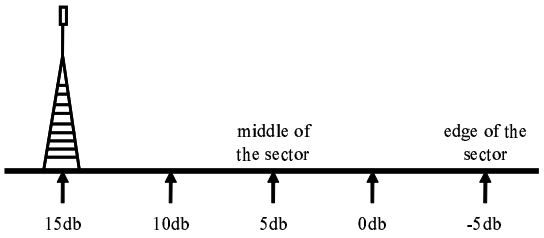

図 9: HSDPA エミュレータによって模擬される無線環境.

に滞留するため, 広告ウィンドウの制限を受けない帯域幅 (1.4Mbps) 以下の場合には TCP Vegas のセグメント滞留量の最大値は BDP の 3 倍以上となる. TCP Vegas の制御においては (3) 式に基づいて 1RTT につき輻輳ウィンドウが 1 セグメントづつしか増減しないため, BDP が大きなネットワークにおいてはキュー長が β 以下になるまでに長い時間を必要とする. その結果, 帯域幅 1.0 Mbps の場合のセグメントの滞留量は理想値である 3.0 パケットよりも 1.3 パケット, 帯域幅 2.0Mbps の場合は 12.1 パケット増加する.

FS-AT は *rcv_bytes* を利用して (5) 式に基づいてネットワークの輻輳状態を判定しているため, 最短及び最長の場合共に TCP Vegas と比べて 1RTT 早く制御を開始することができる. そのため, Slow Start 中は, 最短の場合, 輻輳ウィンドウサイズが BDP の 2 倍, 最長の場合, BDP の 4 倍になって初めて FS-AT の制御が行われる. そのため FS-AT のセグメント滞留量の最大値は BDP の 3 倍以下に保つことができる.

4.2 無線ネットワークにおける性能評価

本シミュレーションでは, HSDPA エミュレータにおいて図 9 に示す 3 つ移動モデル; Standstill(ST), 3km/h で移動する Pedestrian (PA3) そして 30km/h で移動する Vehicle (VA30) について, それぞれ 5 種類のジオメトリ; 15db, 10db, 5db, 0db, -5db における CQI とエラーパターンを利用して HSDPA ネットワークを模擬した. 図 10 に Reno を評価した際に 10 セグメント毎に計算したスループットの最大値及び最小値 (Max, Min), 平均及び標本分散 (Average, Deviation) を示す. HSDPA エミュレータによって達成される最大のスループットは 3.6Mbps であり, 10.0Mbytes のデータを 10 回送信して評価を行った.

		15db	10db	5db	0db	-5db
VA30	Max (Min) [Kbps]	2326.3 (152.1)	2051.2 (151.0)	2055.6 (145.4)	1338.1 (142.4)	2107.3 (115.2)
	Average (Deviation) [Kbps]	1585.0 (197.1)	1474.9 (167.4)	1251.8 (150.0)	849.1 (110.5)	456.1 (814.8)
PA3	Max (Min) [Kbps]	3354.2 (161.3)	3334.5 (154.4)	3332.2 (151.4)	3282.1 (129.2)	1731.2 (106.4)
	Average (Deviation) [Kbps]	25840.0 (680.2)	2371.1 (680.2)	2026.3 (728.7)	1230.0 (571.8)	519.6 (274.1)
ST	Max (Min) [Kbps]	3346.6 (152.4)	3349.9 (159.9)	2781.1 (148.0)	1383.1 (143.1)	586.4 (134.8)
	Average (Deviation) [Kbps]	2921.1 (676.9)	2888.5 (687.5)	2252.3 (344.8)	1271.3 (87.4)	522.8 (29.5)

図 10: 各シナリオの特性.

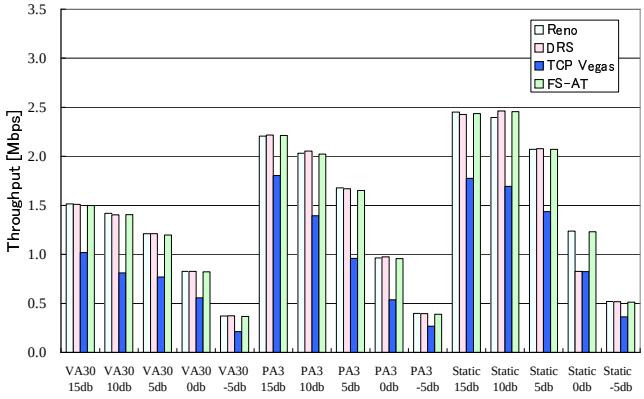

図 11: 無線ネットワークにおけるスループット.

図 11 に各方式のスループットを示す. FS-AT や DRS は, 現在の BDP や BDP の変動に合わせてセグメントの送信量を調整するため, 帯域幅の変動が大きなネットワークにおいてもスループットが低下しない. 一方 TCP Vegas は, セグメントの滞留量を α〜β に調整しようとするため, α や β が小さい場合には, 帯域幅が急激に増加した場合にスループットが低下する.

図 12 に各方式のスループットに対するセグメントの滞留量を示す. 有線ネットワークの場合と同様 Reno のセグメント滞留量が最も大きくなる. FS-AT では, 広告ウィンドウの制限を受けない帯域幅 (1.4Mbps) 以下の場合においては Reno と比較した場合, 常にセグメントの滞留量を削減することができる. DRS の場合は, *rcv_data* の最大値に基づいてセグメントの送信量を調整するため, PA3 や VA30 といった平均スループットと最大スループットの差が大きなシナリオの場合にセグメントの滞留量が増加し, Reno に対するセグメントの滞留量の削減効果が減少する. TCP Vegas の場合は, スループットが減少するため, ほとんどの場合においてセグメントの滞留量を β 以下にすることができる.

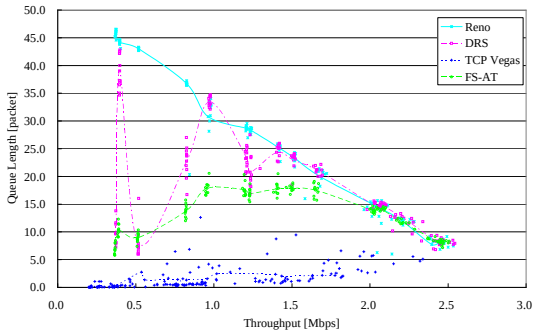

図 12: 無線ネットワークにおけるスループットとセグメントの滞留量.

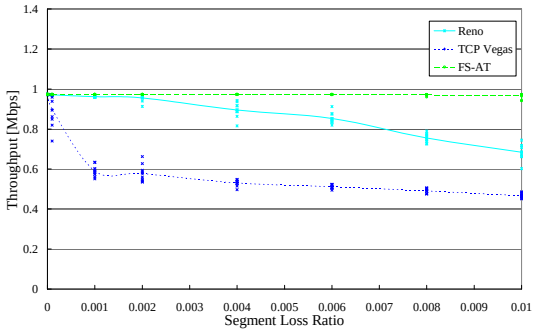

図 13: セグメントロス率とループット.

4.3 セグメントロス時の性能評価

本シミュレーションでは, NS-2 ネットワークにおいて Router2 と Router3 の間のネットワークの帯域幅を 1.0Mbps に設定しセグメントロス率を 0.0%〜1.0% まで変化させ, 10.0Mbytes のデータを 10 回送信して評価を行った.

図 13 に各方式のスループットを示す. Reno のスループットはセグメントロス率の上昇と共に減少する. これは, パケットロス率が上昇すると共に, 図 4 における①② の様にセグメントの送信量が BDP の 2 倍になる前にセグメントロスが発生するためである. TCP Vegas はセグメントの滞留量を β 以下に調整しようとするため, セグメントロス率が低い場合においても大きくスループットが減少する. FS-AT では, 3.3 節で説明した輻輳制御をおこなっているため, セグメントロス率が増加した場合においてもスループットをほぼ一定に保つことができる.

4.4 Reno との公平性の検証

図 1 に示した 3G ネットワークの場合, GW によって全ての TCP コネクションに FS-AT を利用することができる. また, ボトルネックノードとなる RNC では UE 毎にバッファが用意されているため Reno との公平性の問題は生じない. しかし FS-AT をインターネットへ普及させるためには Reno との公平性の検証が重要となる. 本シミュレーションでは, NS-2 ネットワークにおいて Router2 と Router3 の間のネットワークの帯域幅を 1.0Mbps としてバッファサイズを変化させ 10.0Mbytes のデータを 10 回送信して評価を行った. 送信ノードには Reno と FS-AT について同数を用意し, 各ノードの通信を 0〜5 秒の間にランダムで開始した. 図 14 にそれぞれの場合の結果を示す.

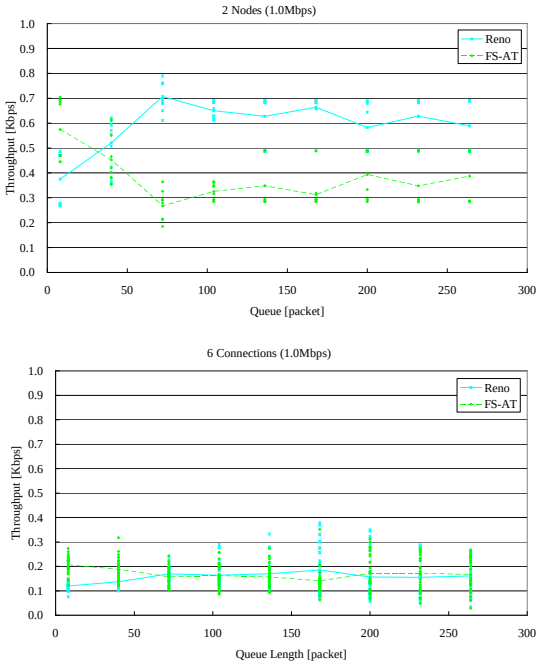

図 14: Reno と FS-AT のスループット.

FS-AT は 3.3 節で説明した輻輳制御によって, バッファ量が十分ではなくセグメントロスが発生する場合 Reno に比べてスループットが高くなる. 十分なバッファ量が用意されている場合 Reno によってセグメントの滞留量が増加してしまうため FS-AT のスループットが低下する. 一方で, ノード数が増加した場合, セグメントロスが発生する領域が増加するため FS-AT のスループットが Reno よりも高くなる領域が増加する.

5 まとめ

広帯域無線ネットワークにおいて TCP を用いた通信を行う場合, 大きな BDP を満たすためにはセグメントの送信量の上限を決定する広告ウィンドウの拡大が必要になる. しかし, 無線環境の悪化などによってスループットが低下した場合, 広告ウィンドウの拡大によって過度なセグメントがネットワークに送信されルータにセグメントが滞留し, アプリケーションの応答性の劣化, アプリケーションの停止, 不要なセグメントの転送などの問題を引き起こす. 本

稿では FS-AT を提案し, 現在のネットワークの帯域幅遅延積に応じてセグメントの送信量を自動調整する輻輳制御方法を提案した. さらにセグメントロスが発生した場合, セグメントロスの原因やセグメントロス直前の帯域幅遅延積に基づいて再送閾値を設定することによってセグメントロス発生後の帯域幅を迅速に利用可能にする. シミュレーションにより評価を行った結果, 有線ネットワーク及び帯域幅の変動が大きい無線ネットワーク双方の場合において, FS-AT はスループットを損なうこと無く, ルータに滞留するセグメント量を削減できることが示された. また, セグメントロスが発生した場合においても, FS-AT は Reno と比較した場合, スループットの低下を防ぐことができることが示された. 今後はインターネットに普及させるために, 公平性の検証をさらに進めていくつもりである.

参考文献

[1] High Speed Downlink Packet Access(HSDPA); Overall description; Stage 2, 3GPP TS 25.308, V.5.2.0, March 2002.

[2] cdma2000 High Rate Packet Data Air Interface Specification, 3GPP2 C.S0024-B, V.1.0, May 2006.

[3] LTE Physical Layer - General Description (Release 8), 3GPP TS 36.201, V1.1.0, May 2007.

[4] V. Jacobson, R. Braden and D. Borman, TCP Extensions for High Performance, RFC 1323, May 1992.

[5] H. Inamura, G. Montenegro, R. Ludwig, A. Gurtov and F. Khafizov, TCP over Second (2.5G) and Third (3G) Generation Wireless Networks, RFC 3481, February 2003.

[6] D. Chase, : Code combining - a maximum - likelihood decoding approach for combining an arbitrary number of noisy packets, IEEE Trans. Communication., Vol. 33, May 1985.

[7] N. Hu, P. Steenkiste, Improving TCP startup performance using active measurements: algorithm and evaluation, Proc. ICNP'03 , November 2003.

[8] S. Keshav, Packet pair flow control, IEEE/ACM Transactions on Networking, February 1995.

[9] K. Saito, K. Igarashi, T. Ihara and A. Miura, Data Link Layer Aware Congestion Control for wireless networks, Trans. IEICE Part.B, Vol.90-B, No.4, April 2007.

[10] L. Brakmo, S. O'Malley and L. Peterson, TCP Vegas: New techniques for congestion detection and avoidance, Proc. SIGCOMM '94, August 1994.

[11] C. Jin, D. Wei and S. Low, FAST TCP: Motivation, Architecture, Algorithms, Performance, Proc IEEE Infocom'04, March 2004.

[12] M. Chan, and R. Ramjee, TCP/IP Performance over 3G Wireless Links with Rate and Delay Variation, ACM MOBICOM'02, September 2002.

[13] M. Fisk and W. Feng, Dynamic Right-Sizing in TCP, 2nd Annual Los Alamos Computer Science Institute Symposium (LACSI 2001), October 2001.

[14] M. Mathis, S. Floyd and A. Romanow, TCP Selective Acknowledgment Options, RFC 2018, October 1996.

[15] M. Handley, J. Padhye and S. Floyd, TCP Congestion Window Validation, RFC 2861, June 2000.

[16] P. Sarolahti and M. Kojo, Forward RTO-Recovery (F-RTO): An Algorithm for Detecting Spurious Retransmission Timeouts with TCP and the Stream Control Transmission Protocol (SCTP), RFC 4138, August 2005.

[17] L. Xu, K. Harfoush and I. Rhee, Binary Increase Congestion Control (BIC) for Fast Long-Distance Networks, Proc. IEEE Infocom'04, March 2004.

[18] T. Kelly, Scalable TCP: improving performance in highspeed wide area networks, SIGCOMM Computer Communication Review, Vol. 33, No. 2, April 2003.

[19] M. Allman, TCP Congestion Control with Appropriate Byte Counting (ABC), RFC 3465, February 2003.

[20] J. Hoe, Startup Dynamics of TCP 's Congestion Control and Avoidance Schemes. Master's Thesis, 1995.

[21] S. Floyd, HighSpeed TCP for Large Congestion Windows, RFC 3649, December 2003.

[22] V. Jacobson, Congestion Avoidance and Control, SIGCOMM '88, August 1988.

[23] http://www.kernel.org/.

[24] A Linux TCP implementation for NS2, http://www.cs.caltech.edu/~weixl/technical/ns2linux.