平成 20 年度　修士論文

移動端末におけるリアルタイム系トランスポート層プロトコルの特性改善

早稲田大学理工学研究科　情報・ネットワーク専攻

3606U088-8

藤川　知樹

指導　甲藤二郎 教授

2008 年 2 月 7 日

指導教授印	受付印

第一章　序論

１．１　はじめに

近年，インターネットの高速化に伴い，テレビ会議やストリーミングといった，リアルタイム性を必要とするアプリケーションが急速に普及しつつある．これらのアプリケーションは主に，トランスポート層プロトコルとして UDP を用いている．しかし，UDP は単にデータを転送するためのプロトコルであり，ネットワークの輻輳に対する制御を行っていない. TCP と UDP との混在環境では, 帯域制御を行っている TCP のトラフィックが UDP によって阻害され，場合によっては輻輳崩壊を起こす恐れがある．そのため UDP は，その機能を別途実装する必要があり，VTP[8]，TFRC (TCP-Friendly Rate Control) [6]，DCCP (Datagram Congestion Control Protocol) [10]といった，トランスポート層通信プロトコルが設計されている．これらのプロトコルは，送信者の送信レートをネットワークの通信状態に合わせ，TCP の AIMD アルゴリズムや平衡状態のレート式を用いることで，TCP との公平性を実現するという特徴を持っている．

一方，携帯電話などの移動通信においても，FMC(Fixed Mobile Convergence)やモバイル WiMAX の普及に伴う通信速度の向上により，リアルタイム系サービスの急増が予想される． 移動通信では，利用者が移動するのに伴い，アクセスポイント(Access Point : AP)を切り替えるハンドオーバが必要となる．このハンドオーバを用い，次世代ネットワーク環境において，利用可能な通信帯域や往復遅延時間が大きく異なる異種無線ネットワークを統合することが考えられている．しかし,異種無線ネットワーク間ではその前後で環境が大きく異なることがあるため，特に TFRC では，移動後通信状況が悪化したり，既存のネットワークに悪影響を与えることがある．

また，移動端末に限らず，競合するフローと通信環境が異なる場合，スループットが不公平に分配されてしまうことがある．特に RTT が異なるとき，TCP Reno ではスループットが RTT に比に分配され，一定にならないことが分かっている[11]．AIMD アルゴリズムや平衡状態のレート式を用いる UDP 向けトランスポート層通信プロトコルにおいても，TCP Reno の AIMD アルゴリズムや平衡状態のレート式を用いているため，TCP Reno と同様の傾向が見られる．

本論文では，上記２つの問題についてそれぞれ提案を行い，その有効性を示す．異種間ネットワークにおける問題については，HON と新 AP 間のリンク遅延が大きい場合に着目し，HON が不必要なレート削減とタイムアウトの発生を抑制することで，HON による移動後のネットワークへの影響を減少させる手法を提案する．また，RTT の異なるフローにおける不公平性については，キューイング遅延を用い，公平性を実現する輻輳制御方式を提案する．

１．２　本論文の構成

第二章では，研究背景について述べる．
第三章では，一つ目の提案方式について述べる．
第四章では，二つ目の提案方式について述べる．
最後に，総括と今後の課題について述べる．

第二章　研究背景

本章では，TCP・UDP の輻輳制御と Mobile IP について説明する．

２．１ TCP

ここでは，インターネットにおける輻輳制御手法の例として TCP を用いた輻輳制御について述べる．現在提案されている UDP の輻輳制御手法は，TCP の制御方式を応用したものが多く，後に説明する TFRC や VTP もこのうちの一つである．

そこで TCP における輻輳制御方式として代表的な，TCP Reno，Scalable TCP，HighSpeed TCP，BI-TCP の４つを挙げ，以下で説明する．

２．１．１　TCP Reno

Reno[1] は ACK パケットの到着をもとにして，転送レートの調節を行う，ウィンドウフロー制御を行っている．Reno は，まず，送受信ホスト間で予め ACK の受信を待たずに連続して送信してよいパケットの最大数を設定する．この，パケットの最大数を輻輳ウィンドウサイズと呼ぶ．送信側ホストは，データパケットを送信済みであるが，まだ ACK パケットを受信していないパケットの数が輻輳ウィンドウサイズよりも小さい場合は，さらに続けてデータパケットを送信することができる．もし，ACK パケットを受信していないパケットの数が輻輳ウィンドウサイズと等しくなった場合は，パケットの送信を停止し，ACK パケットの到着を待つ．

Reno の送信側ホストは，ACK パケットを受信すると，輻輳ウィンドウサイズ cwnd を次式のように変更する．

$$cwnd \leftarrow \begin{cases} cwnd + 1 & if \quad cwnd < ssth \\ cwnd + \dfrac{1}{cwnd} & othrewise \end{cases}$$

ここで，ssth は，Reno が「スロースタートフェーズ」から「輻輳回避フェーズ」と呼ばれる動作モードに移行する時のしきい値である．つまり，Reno はスロースタートフェーズでは輻輳ウィンドウサイズを指数的に増加させ，輻輳回避フェーズでは輻輳ウィンドウサイズを線型増加させる．

次に，Reno がパケット棄却を検出した場合を説明する．まず，Reno は重複 ACK (Duplicate ACK) とタイムアウトといった 2 種類の方法で，ネットワーク中でのパケット棄却を検出する．Reno が重複 ACK によりパケット棄却を検出した場合(Fast Retransmission)，しきい値 ssth と輻輳ウィンドウサイズ cwnd を次式のように更新する．

ssth ← cwnd/2

cwnd ← ssth

このため，高速回復フェーズにおいて，輻輳ウィンドウサイズは指数的に増加することに

なる．ただし，再送したパケットに対応した ACK パケットを受信すると，高速回復フェーズから輻輳回避フェーズに移行する．同時に，輻輳ウィンドウサイズを高速回復フェーズ移行前の値に戻す．

Reno がタイムアウトによりパケット棄却を検出した場合，しきい値 ssth と輻輳ウィンドウサイズ cwnd を次式のように更新する．

ssth ← cwnd/2

cwnd ← 1

図２．２．１に Reno の輻輳ウィンドウサイズの動作を示す．

このような Reno のウィンドウフロー制御により，ネットワークに送出されるパケットの量を調節することが可能である．これにより，ネットワーク内部での輻輳を防ぐ，もしくは輻輳が発生した場合に，輻輳を早期に解消することが可能となる．

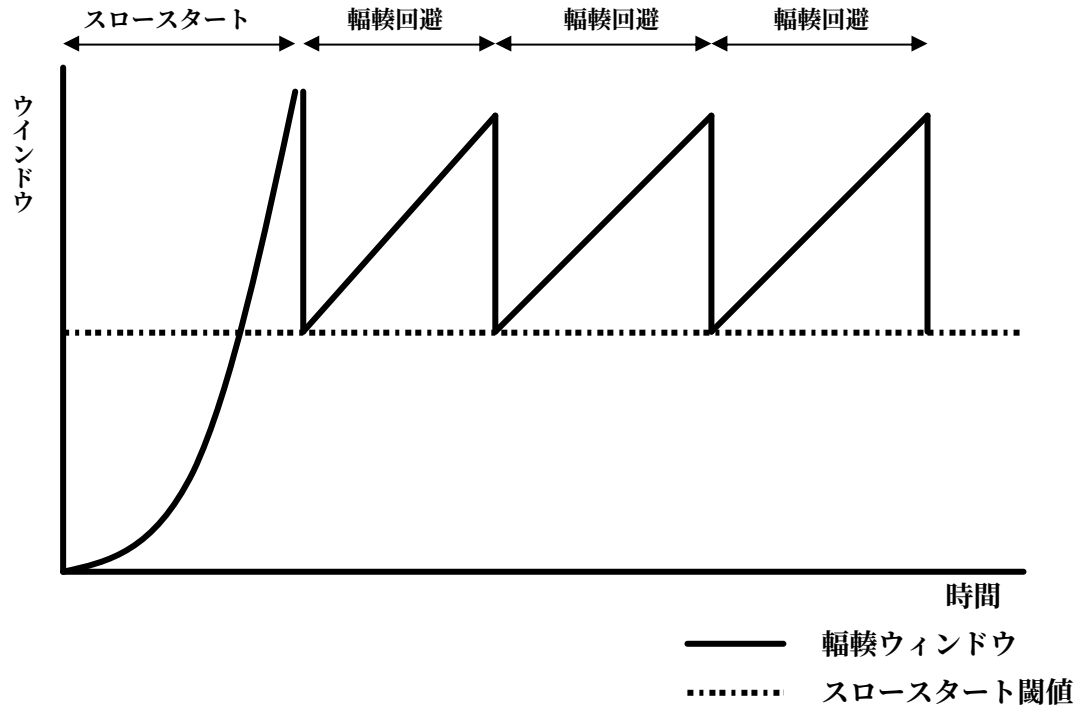

図２．２．１　TCP-Reno によるウインドウの変化

２．１．２　Scalable TCP

Scalable TCP[2]は Reno のアルゴリズムをより高速回線向けにしたものである．

以下で，Scalable TCP の輻輳回避フェーズの輻輳ウィンドウサイズの増減について説明する．

Reno の輻輳ウィンドウサイズの増減は

cwnd ← cwnd + 1/cwnd　　(通常の ACK 受信時)

cwnd ← cwnd×1/2　　(閾値以上の dup ACK 受信時)

となっている．Scalable TCP では，ACK が返ってくるごとに輻輳ウィンドウサイズをα足し，ロスが起こったときに，輻輳ウィンドウサイズのβ分減少させることによって，よりスケーラビリティの高い制御となる．

cwnd ← cwnd+α　　(通常の ACK 受信時)

cwnd ← cwnd－β×cwnd　　(閾値以上の dup ACK 受信時)

図２．２．２は，Scalable TCP のウィンドウの動作を表している．Reno より高速にウィンドウを増減しているため，高速回線においても高いスループットを維持することが可能となる．

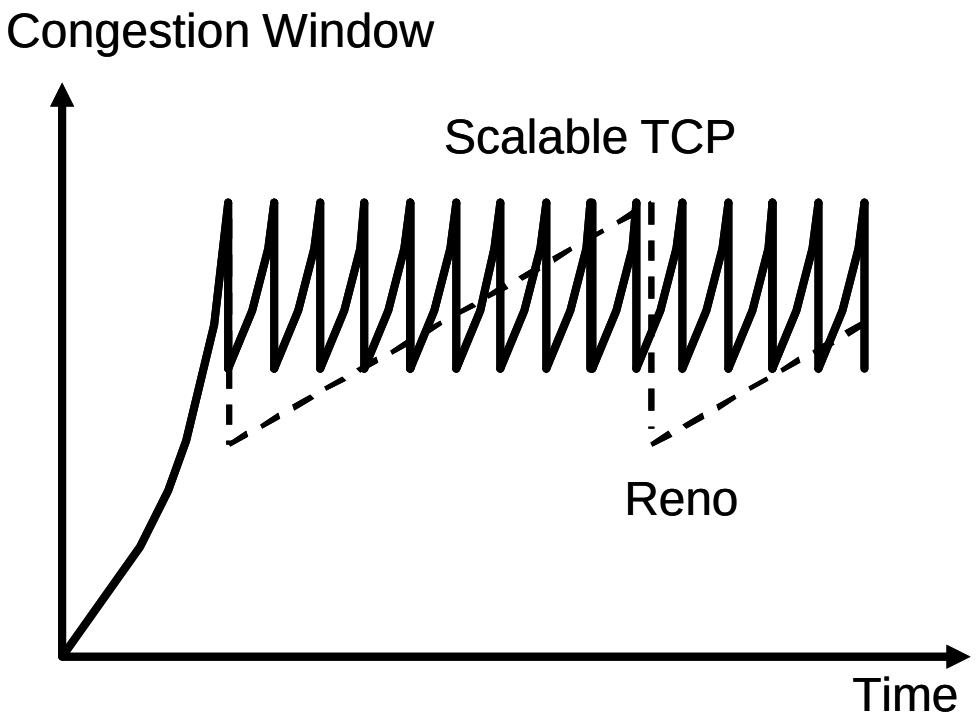

図２．２．２　Scalable TCP の輻輳ウィンドウサイズの振る舞い

２．１．３　HighSpeed TCP

HighSpeed TCP[3]は，TCP-Reno と同様の AIMD 型の輻輳制御方式を拡張し，輻輳ウィンドウサイズの増加幅，パケット廃棄時の減少幅といったパラメータを TCP-Reno よりも高速にすることにより，高速長距離ネットワークにおいても高いスループットを得ることが出来る方式である．

以下に，HighSpeed TCP の輻輳回避フェーズの輻輳ウィンドウサイズの増減について説明する．HighSpeed TCP は現在の輻輳ウィンドウサイズがW_{low}より小さい場合は，TCP-Reno と同じアルゴリズムに従って輻輳ウィンドウサイズを増減させる．一方，現在の輻輳ウィンドウサイズがW_{low}より大きい場合には，現在の輻輳ウィンドウサイズの大きさによって，増加幅，減少幅を調節する．以下に，HighSpeed TCP のパラメータの導出式を示す．ただし，

w は現在の輻輳ウィンドウサイズ，a(w)は 1RTT 毎の増加幅，b(w)は重複 ACK によってパケット廃棄を検出した際の輻輳ウィンドウサイズの減少幅を示す．

$$a(w) = \frac{2w^2 \times b(w) \times p(w)}{2 - b(w)}$$

$$b(w) = \frac{\log(w) - \log(W_{low})}{\log(W_{high}) - \log(W_{low})}(b_{high} - 0.5) + 0.5 \qquad (2.1.1)$$

$$p(w) = \exp[\frac{\log(w) - \log(W_{low})}{\log(W_{high}) - \log(W_{low})}\{\log(P_{high}) - \log(P_{low})\} + \log(P_{low})]$$

式（2.1.1）において，P_{low}は，$P_{low} = \frac{1.5}{W_{low}^2}$により定義され，TCP-Reno において平均輻輳ウィンドウサイズが W_{low} となる時のパケット廃棄率を示すものである．また，P_{high}，W_{high}，b_{high}，W_{low} は，HighSpeed TCP が持つパラメータであり，以下のような意味を持つ．HighSpeed TCP は，ネットワークにおけるパケット廃棄率が P_{high} の時に，平均ウインドウが W_{high} になるように，輻輳ウィンドウサイズの増加幅 a(w)，および減少幅 b(w)を決定する．また，現在の輻輳ウィンドウサイズが W_{high} 以上の時には，重複 ACK の受信によってパケット廃棄を検出した際の輻輳ウィンドウサイズを $(1-b_{high})W_{high}$ に減少する．以上より，HighSpeed TCP はTCP-Reno よりも高速ネットワークにおいても高いスループットを実現することが可能である．

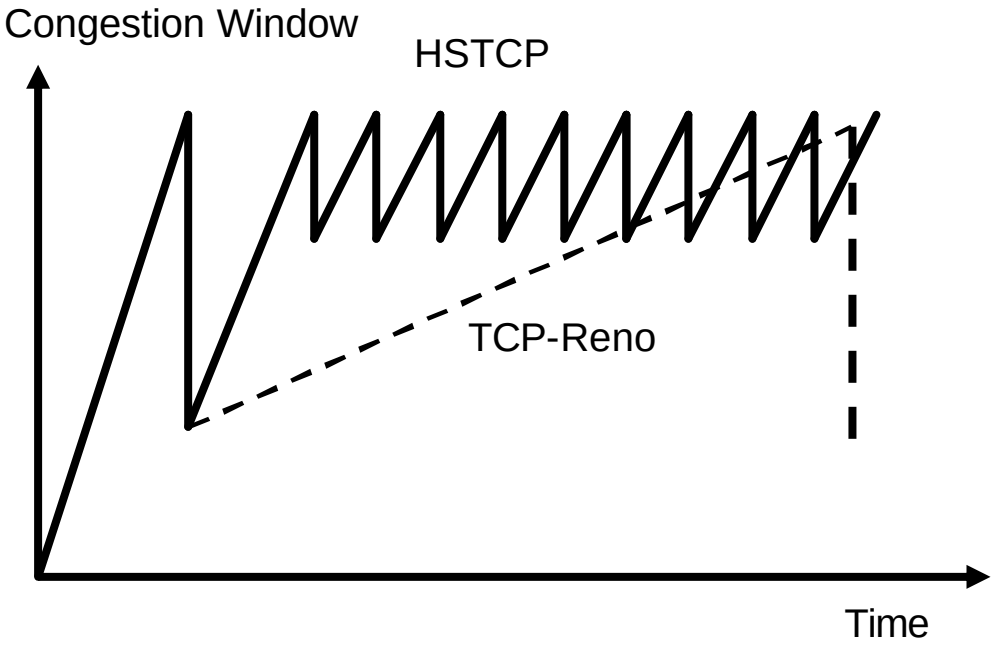

図２．２．３　HighSpeed TCP の輻輳ウィンドウサイズの振る舞い

２．１．４　BI-TCP

現在 Linux の default となっている方式である．BI-TCP[4](BIC)では，高速性だけでなく，RTT が異なるフロー間の公平性にも着目している．TCP-Reno，HSTCP では輻輳ウィンドウサイズの増加は線形的であるのに対し，BIC では２分探索的に輻輳ウィンドウサイズの増加幅を決定する．

BIC では，パケット廃棄発生時に，パケット廃棄発生直前の輻輳ウィンドウサイズを Wmax，そして減少させた直後の輻輳ウィンドウサイズを Wmin と設定する．そして，Wmin と Wmax の間で，２分探索的に輻輳ウィンドウサイズを増加させる．ただし，その増加幅には上限値が定められており，増加幅が上限値以下となるまでは線形的に輻輳ウィンドウサイズを増加させる（Additive Increase）．そして，輻輳ウィンドウサイズが Wmax に達した場合，スロースタートフェーズ同様，指数関数的に増加させる（Max. Probing）．

図２．１．４に BIC の輻輳ウィンドウサイズ変化を示す．このように，TCP-Reno，HSTCP とは違い，特徴的な振る舞いを行う．

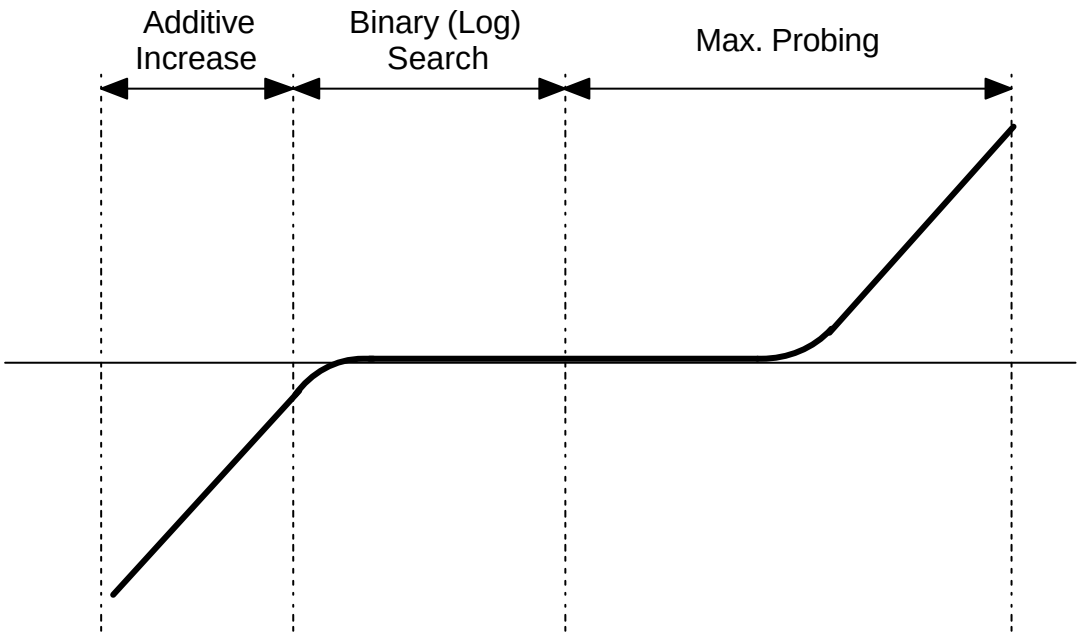

図２．１．４　BIC の輻輳ウィンドウサイズの振る舞い

２．１．５　TCP Libra

TCP Libra[5]は，従来の Additive Increase Multiple Decrease (AIMD)型輻輳制御方式を，異なる RTT のフローが混在する場合でも，公平な帯域分配が可能となるように設計された制御方式である．

TCP Libra の増加・減少は以下のように表せられる．

$$W_{n+1} \leftarrow W_n + \frac{1}{W_n}\frac{a_n T_n^2}{T_n + T_0} \qquad \text{(通常の ACK 受信時)}$$

$$W_{n+1} \leftarrow W_n - W_n \frac{T_1}{2(T_n + T_0)} \qquad \text{(閾値以上の dup ACK 受信時)}$$

Tn は RTT，T_0，T_1は RTT の初期値，α_nは表 3.1.1 に示すパラメータ，W はウィンドウサイズを表している．[5]では，$T_0 = T_1 = 1$，$k_1 = k_2 = 2$ と設定されている．

Penalty functions
$K_2\left(\frac{T_{r\max} - T_r(t)}{T_{r\max} - T_{r\min}}\right)$
$K_2\left(\frac{T_{r\max} - T_r(t)}{T_{r\max} - T_{r\min}}\left(\frac{2T_{r\max} - T_{r\min} - T_r(t)}{T_{r\max} T_{r\min}}\right)\right)$
$e^{-K_2\left(\frac{T_{r\max} - T_r(t)}{T_{r\max} - T_{r\min}}\right)}$

表２．１．１　αの値

一方，AIMD 型輻輳制御方式の平均スループットは，$\overline{RTT}$，$\bar{\rho}$ をそれぞれ平衡状態における RTT とロス率，α を増加係数，β を減少係数とすると

$$\overline{R} = \frac{1}{\overline{RTT}}\sqrt{\frac{a}{b}\frac{1-\overline{r}}{\overline{r}}} \qquad (2.1.2)$$

と表すことができる．この式に TCP Libra の増加係数・減少係数を代入すると，

$$\overline{R} = \frac{1}{\overline{RTT}}\sqrt{\frac{a}{b}\frac{1-\overline{r}}{\overline{r}}} = \sqrt{\frac{g}{RTT_1}\frac{1-\overline{r}}{\overline{r}}} \qquad (2.1.3)$$

となり，スループットが RTT に依存しない形となり，定常的に RTT fairness を実現できることになる．

２．２　UDP

リアルタイムストリーム通信を提供するアプリケーションでは，より単純な機能しかもたない UDP (User Datagram Protocol) を用いるのが主流となっている．しかし，UDP は帯域制御機能を持たないことから，現在広く用いられている TCP と共存すると，UDP による通信の方が帯域を占有してしまい，TCP の通信品質がひどく低下してしまうことが危惧されている．そのため TFRC や VTP のような輻輳制御が必要になる．

２．２．１ TFRC

TFRC[6](TCP Friendly Rate Control) は，TCP-Reno の動作から導かれたスループット方程式を基にした，ユニキャストの輻輳制御メカニズムで，UDP 同様，再送制御のないベストエフォート型通信を行う．(1) 有線ネットワーク環境において混在する TCP フローとの公平な帯域利用を実現することを目的としたメカニズムであり，TCP の輻輳制御と互換性をもつが，TCP ほどスループットが変動しないような動作を行うので，リアルタイム性が要求される音声・動画像などのマルチメディアの伝送に適したものである．
この TFRC は，TFRC レシーバから，フィードバック情報として，ロスイベント率や受信スループット等を TFRC センダに送信する．TFRC センダはスループット方程式を用いてネットワークの状況に応じて適応的にレートを変動させることで QoS の確保を実現する．

(1)センダ制御

<u>データパケット</u>

データパケットは以下の情報を保持する．

1. シーケンスナンバー
2. パケット送出時間を示すタイムスタンプ
3. センダの RTT 推定値

シーケンスナンバーはパケット送信毎に１インクリメントしていく．パケット送出時間を示すタイムスタンプはパケットがどのロスイベントに属するかを TFRC レシーバが判断するために用いられる．また，TFRC レシーバからエコーされて，TFRC センダが推定 RTT を用いるためにも用いられる．TFRC センダの RTT 推定値は TFRC レシーバはタイムスタンプとともに用い，複数のロスがどのロスイベントに属するかを判断する．

<u>推定レート算出</u>

TFRC センダは TCP-Reno の動作から導かれたスループット方程式を用い，伝送レートの上限を算出する．

$$X = \frac{s}{RTT\sqrt{\frac{2bp}{3}} + t_{RTO}\left(3\sqrt{\frac{3bp}{8}}p(1+32p^2)\right)} \qquad (2.2.1)$$

・X:伝送レート(bytes/second)
・s:TFRC のパケットサイズ(bytes)
・RTT:RTT(seconds)
・tRTO:TFRC の送信側ホストが計算する再転送タイムアウト時間(seconds)
・p:ロスパケット/転送パケット．つまりパケット廃棄イベント率
・b:パケット毎に返される ACK 数(TCP の肯定応答の単位)

R は Round Trip Time で，データパケットに含まれる TFRC センダでのパケット送信タイ

ムスタンプを TFRC レシーバがエコーすることにより測定することができる．また，p は TFRC レシーバが算出するロスイベント率であり，次の節で算出方法の説明を行う．tRTO は max(4R, 1) とするのがリーズナブルである．b は，TCP 肯定応答の単位であり，この値が 2 なら受信データパケット二つ毎に対し，レシーバが ACK を返すといった動作をする．これは TCP の delayed ACK の動作であるが，多くの TCP は受信データパケット一つ毎に対し ACK を返送するので，この値は 1 としている．
図 2.2.5 に，ロス率と RTT と伝送レートの関係を示す．

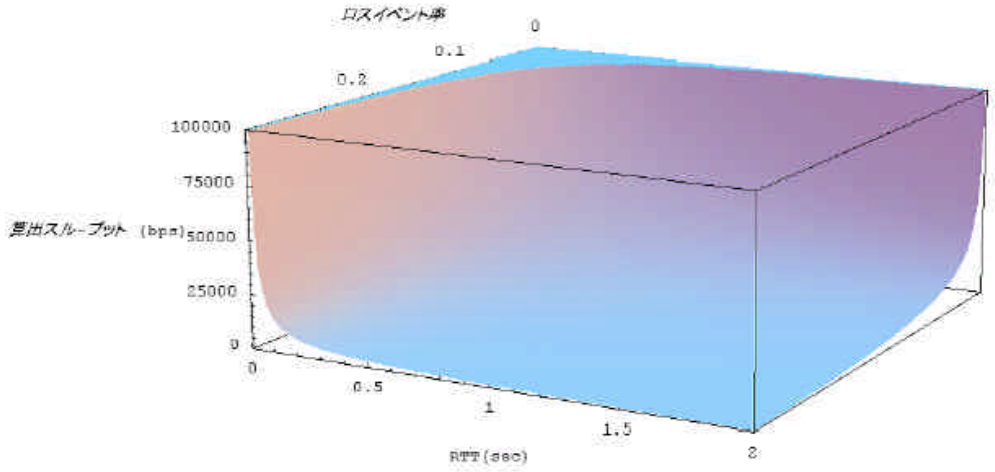

図２．２．５ ロス率と RTT と伝送レートの関係

<u>フィードバックパケット受信時</u>
フィードバックパケットを受信時の TFRC センダの動作を説明する．
1. Round Trip Time sample(Rsample) の測定をする

$$R_{sample} = (t_{now} - t_{recvdata}) - t_{delay}$$

まず，RTT サンプル値を算出するこの値はフィードバックパケットを受信した時のタイムスタンプと次の節で説明するフィードバックパケット内のタイムスタンプとの差分から TFRC レシーバでの処理遅延を差し引いた値である．
2. 推定 RTT(R) の更新

$$R = q \times R + (1 - q) \times R_{sample}$$

次に推定 RTT の更新を行う．この時，RTT の急激な変動を避ける為に，一定の値 q(初期値＝ 0.9) として，フィルターにかける．
3. タイムアウト時間の更新をする

$$t_{RTO} = 4 \times R$$

4. **伝送レートを更新する**

$If\,(p > 0)$

Calculate Xcalc using the TCP throughpu t equation

$$X = \max(\min(X_{calc}, 2 \times X_{recv}), s / t_{mbi})$$

Else

$If\,(t_{now} - t_{ld} \geq R)$

$$X = \max(\min(2 \times X, 2 \times X_{recv}), s / R)$$

$$t_{ld} = t_{now}$$

このように算出されたレートと，TFRC **レシーバからの実際のレートとを比較をし，伝送レートの更新を行う．フィードバック情報のロスイベント率** p **が** 0 **の場合，**TFRC **センダはスロースタートフェーズに入り，伝送レート更新を行う．この時の最低伝送レートは** s R **である．つまり，最低，**RTT **毎に一つのデータパケットを送信することを意味する．また，スロースタートフェーズではロスイベントが生じるまで，**RTT **毎に伝送レートを** 2 **倍ずつ増加させていく．**tld(=Time Last Doubled) **は最後に伝送レートが二倍になった時間である．図２．２．６に** RTT **と下限伝送レートとの関係を示す．最後に，**tmbi **は** 64seconds **で，フィードバックパケット間の最大バックオフ間隔を意味する．**

5. nofeedback timer **を** max(4R, 2sX) **にリセットする**

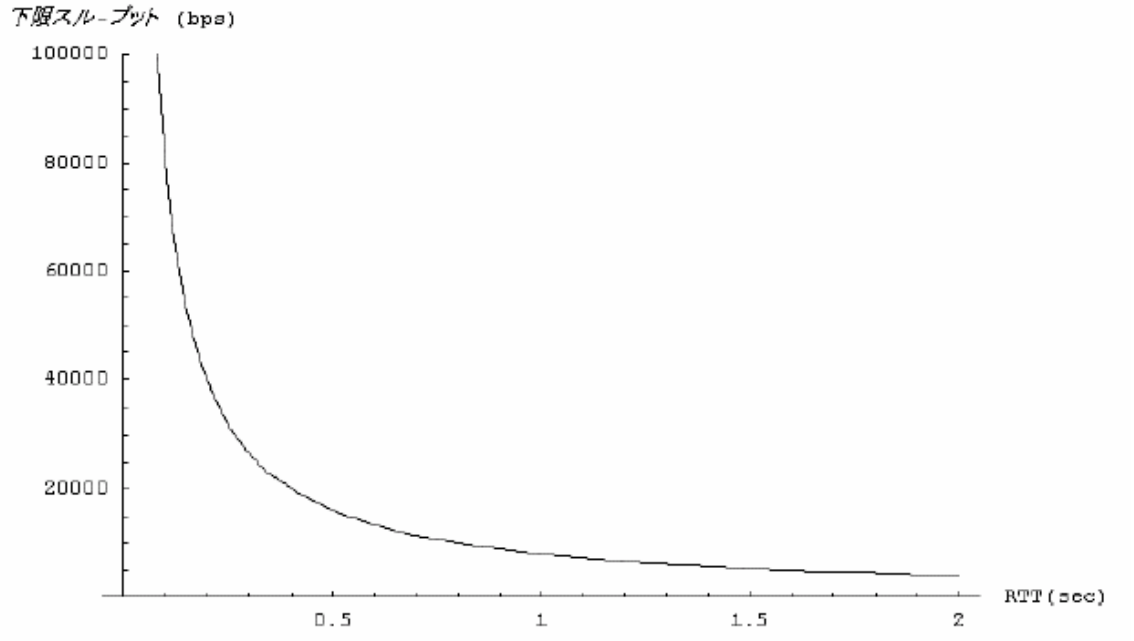

図２．２．６ RTT **と下限伝送レートの関係**

nofeedback timer が終了した場合の動作

ある時間の間(nofeedback timer), TFRC レシーバからのフィードバックパケットが TFRC センダに到達しなかった場合，TFRC センダは以下の動作を行う．

1. 伝送の上限レートを半分に減少させる

$If\,(X_{calc} > 2 \times X_{recv})$

$$X_{recv} = \max\left(\frac{X_{recv}}{2}, \frac{s}{2 \times t_{mbi}}\right)$$

Else

$$X_{recv} = \frac{X_{calc}}{4}$$

この時，以前受け取った Xrecv が更新される．伝送レートは Xrecv の二倍が上限なので，Xrecv を更新することで，伝送レートを制限する．しかしながら，この時でも，スロースタートフェーズに移行することは可能である．

2. nofeedback timer を超過した時，TFRC センダが RTT sample を保持していなかったら，1 のステップは無視され，伝送レートは半分にされる

$$X = \max\left(\frac{X}{2}, \frac{s}{t_{mbi}}\right)$$

3. nofeedback timer が max(4R, 2s / X) にリセットされる

(2)レシーバ制御

TFRC レシーバは通常 RTT 毎に一つのフィードバックパケットを TFRC センダに送信する．しかしながら，ロスイベントが検知された場合，RTT の終了を待つことなしにフィードバックパケットは送信される．もし，一つのパケットを受信するのに RTT 以上の時間が経っている場合は，受信したパケット毎にフィードバックパケットを生成する．

また，もし TFRC センダが高いレートでパケットの伝送を行った場合，RTT に対して迅速に対応できるようにするために RTT に一つ以上のフィードバックパケットを送信するといった動作は有益である．それはまたフィードバックパケットのロスに対しても耐性がある．

また，パケットのシーケンスナンバーの順番に正しく受信されないパケットを Out-Of-Order(OOO) パケットと呼ぶのだが，この OOO パケットが受信された時は，後述するロスヒストリーからロスイベントを削除するといった動作を行う．

制御パケット

レシーバ側で生成される制御パケットは以下の情報を保持する．

1. 最後に受け取ったデータパケットのタイムスタンプ

2. 最後にデータパケットを受け取ってからフィードバックを返すまでの経過時間
3. レシーバ側で推定した伝送レート
4. レシーバ側で算出したロスイベント率
最後に受け取ったデータのタイムスタンプは TFRC センダが推定 RTT を算出する為に用いる．また，最後にデータパケットを受け取ってからフィードバックを返すまでの経過時間もオフセットとして TFRC センダが推定 RTT を算出する為に用いる．また，TFRC レシーバで推定した伝送レートは TFRC センダが伝送レートの上限を導く為に用いる．最後に，TFRC レシーバが算出したロス率は，TFRC センダがスループット方程式を用いて伝送レートを算出するために用いる．

データパケット受信時の TFRC レシーバの振る舞い
1. パケットヒストリーにそのパケットを加える
2. p の前の値を p_{prev} とし，p の新しい値を以降で説明する方式で算出する．
3. $p > p_{prev}$ の場合，feedback timer を終了させ，下記で説明する動作を行う
$p = p_{prev}$ の場合，何もしない
しかしながら，パケットが到着すると，パケットヒストリーに入り，結果，連続した二つのロス間隔が一つに合わさる場合がある．この場合，TFRC レシーバはすぐにフィードバックを送信する．

feedback timer が終了した場合の動作
TFRC レシーバの feedback timer が終了した時の動作は, 最後のフィードバックを送った後にパケットを受信したかどうかに依存する．
ここで，TFRC レシーバの受け取ったパケット中での最大のシーケンスナンバーを S_m とする．また，S_m に含まれている RTT 測定値を R_m とする．前のフィードバックパケットを送った後にデータパケットを受信していたなら以下の動作を行う．
1. 以降で説明する動作で平均ロスイベント率 p を算出する
2. 最後の R_m 秒内で受信したパケット数から X_{recv} を算出する
3. フィードバックパケットを送信
4. feedback timer が R_m 秒後に終了するように再起動する
一方，前のフィードバックパケットを送った後にデータパケットを受信していない場合，フィードバックを送信せず，feedback timer が R_m 秒後に終了するように再起動する．

実際のスループット X_{recv} の算出
レシーバ側においては実際の伝送レートを算出する．この算出の方法はセンダ側とは異なりスループット方程式を用いず，ある時間間隔 R_m でのパケット受信数から算出される．R_m は最後に受信したデータパケット(最大シーケンスナンバー) が保持している RTT で

ある．前のフィードバックパケットを送信した後のパケット受信数を n_{recv} とする．また，最後にフィードバックパケットを送信したタイムスタンプを $t_{lastrecv}$ とする．
TFRC レシーバは実際のスループット X_{recv} を以下のように算出する．

$If\,(t_{now} - t_{lastrecv} > R_m)$

$$X_{recv} = n_{recv} / (t_{now} - t_{lastrecv})$$

$Else$

$$X_{recv} = n_{recv} / R_m$$

<u>ロスイベント率 p 算出</u>
TFRC にとって，高精度かつ頑強なロスイベント率を算出することは非常に重要なことである．

<u>ロスヒストリーからロスイベントへの変換</u>
同じロスイベントの一部である連続したパケットロスに対して頑強である為に，ロスヒストリーパケットをロスイベントレコードに記録する．つまり，複数のロスパケットを一つのロスイベントとして扱うということである．
TFRC レシーバがロス，もしくは ECN でマーキングされたパケットを新しいロスイベントとして扱うべきか，既存のロスイベントとして扱うべきかを判断する為に，到着したパケットのシーケンスナンバーとタイムスタンプを比較する．

この時，TFRC レシーバは式 2.2.2 でロスパケット毎にその到着時間を推定する．

$$T_{loss} = T_{before} + ((T_{after} - T_{before}) \times (S_{loss} - S_{before}) / (S_{after} - S_{before})) \quad (2.2.2)$$

・S_{loss}：ロスパケットのシーケンスナンバー
・S_{before}：S_{loss} の前のシーケンスナンバーを持つ最後のパケットのシーケンスナンバー
・S_{after}：S_{loss} の後のシーケンスナンバーを持つ最初のパケットのシーケンスナンバー
・T_{before}：S_{before} の受信時間
・T_{after}：S_{after} の受信時間

<u>ロスイベント間隔</u>
ロス間隔 A がシーケンスナンバーS_A で開始し，次のロス間隔 B がシーケンスナンバーS_B で開始される場合，ロス間隔 A 内のパケット数は$(S_B - S_A)$ で与えられる．

<u>平均ロスイベント間隔</u>
ロスイベント率を算出するためには，まず平均ロス間隔を算出する必要がある．
また，最新の n 個のロス間隔に重み付けをして，ロスイベント率が滑らかな更新をするよ

うにする．

重み w_0 から $w_{(n-1)}$ の算出は以下のように行う．

$$If\ (i < \frac{n}{2})$$

$$w_i = 1$$

$$Else$$

$$w_i = 1 - \frac{i - \left(\frac{n}{2} - 1\right)}{\left(\frac{n}{2} + 1\right)}$$

n = 8 なら w_0 から $w_{(n-1)}$ は 1, 1, 1, 1, 0.8, 0.6, 0.4, 0.2 となる．本論文で用いた TFRC では n = 4 として設定している．

そして，平均ロスイベント間隔 I_{mean} の算出は以下のように行う．

$I_{tot0} = 0$

$I_{tot1} = 0$

$W_{tot} = 0$

for (i = 0 to n - 1)

$I_{tot0} = I_{tot0} + (I_i \times w_i)$

$W_{tot} = W_{tot} + w_i$

for (i = 1 to n)

$I_{tot1} = I_{tot1} + (I_i \times w_{(i-1)})$

$I_{tot} = \max(I_{tot0}, I_{tot1})$

$I_{mean} = I_{tot}$

W_{tot}

そして，ロスイベント率 p は以下のように算出される．

$$P = \frac{1}{I_{mean}}$$

２．２．２ LDA

有線ネットワークにおいて，パケットのロスはネットワークが輻輳状態にあることを意味する．これは，ネットワーク上のパケットの流量が多すぎて，ルーターにパケットが，キュー長の制限以上にたまってしまい，それによりパケットがドロップするからである．輻輳に対して，TFRC などの輻輳制御を行うプロトコルはパケット伝送レートを適応的に減少させることにより，ネットワーク上のパケットの流量を抑え，パケットのロスを緩和させるといった動作を行う．

しかしながら, 無線ネットワークにおいては, パケットロスは必ずしも輻輳を意味しない. 特にマルチホップ無線ネットワークにおいては，パケットのロスは輻輳だけでなく，ランダムなビットエラーロスや 802.11 のリンク層での再送制限超過によるロスやパケットを伝送させるルートが存在しないルートエラーロスなどによっても引き起こされる．つまり，無線ネットワークにおいてはパケットのロスは，必ずしもネットワークが輻輳状態にあることを意味しない.

これはつまり，パケットロスを検知したら，ネットワークが輻輳状態にあると判断し，パケットの伝送レートを減少させる本来の輻輳制御のメカニズムが輻輳以外が原因のパケットロスに対しても適用されてしまうことを意味する．この動作は，不必要なスループットの減少を導いてしまうという問題がある．こうした問題を解決するべく，無線ネットワークにおいて，パケットのロスを輻輳によるロスと無線特有の原因によるロスとを区別して，ネットワーク層のプロトコルは動作するべきだという提案がなされている．ロスを区別する手法を LDA(Loss　Dif ferentiation　Algorithm) という．[7]

もし，LDA によりパケットのロスが輻輳によるロスであると判断されたら，本来の輻輳制御を行い，パケットのロスが無線特有によるロスであると判断されたら，レートを下げないといった動作を行う．以下に主な LDA の手法の説明を行う.

Biaz 手法

[7]では，最後の 1 ホップのみがワイヤレスで，そのワイヤレスリンクがボトルネックである状況を想定して LDA の提案を行っている.

ワイヤレスリンクがボトルネックなので，基地局(BS) は基本的に一つ以上のパケットをバッファしている．よって，パケット間隔 T はパケットを伝送するのに必要な時間に相当する. 図 の(a) ではパケットのロスはなく, レシーバはパケットを T 毎に受信するが, (b) では, ランダムなロスにより, パケット 2 がロスしてしまっている. この場合, パケット 3 を受信するのはパケット 1 を受信してから 2T かかる．(c) ではパケット 2 が輻輳により有線ネットワークでロスしてしまっている．この場合，レシーバはパケット 3 を受信するのはパケット 1 を受信してから T 以上かかる.

If $(n + 1)T_{min} = T_i = (n + 2)T_{min}$

loss is wireless loss

Else

loss is congestion loss

以上より，biaz 手法では上式より，輻輳によるロスとワイヤレス特有のランダムロスとを区別する．Tmin は最小パケット間隔，n はロスパケットの数，Ti は実際のパケット時間間隔である.

Spike 手法

spike 手法は ROTT(Relative One-way Trip Time) を用いて，ロスの区別を行っている.

ROTT はパケットがセンダからレシーバまでに到着するのに要した時間(OTT) により測定される．この手法は，ネットワークが輻輳状態である場合，パケットの ROTT が急激な上昇(spike) をみせることから，名づけられた．ROTT は現在のコネクションが spike 状態かどうかを判断するのに用いられ，もしコネクションが spike 状態なら，その間に発生したパケットロスは輻輳によるロスで，そうでないならワイヤレス特有のランダムロスであると判定される．

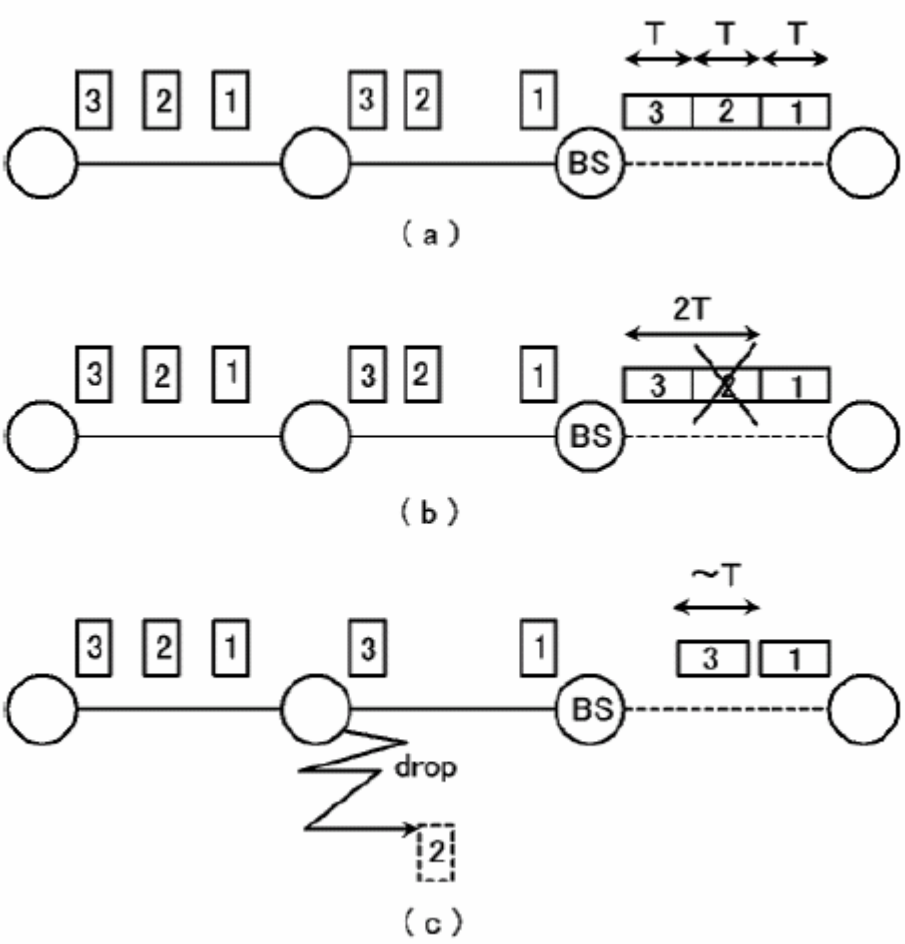

図２．２．７　パケット到着遅延

２．２．３ VTP

VTP[8](Video Transport Protocol) は，特に無線ネットーワークにおいて，高いスループットを保つように設計された，リアルタイム通信向けのエンド間でのレート制御方式である．その構成は，ロスの判別を行う LDA，推定レートを用いてレートの下限を保持する Achieved Rate(AR)という２つのコンポーネントから成り立っている．LDA はロスの状態を輻輳によるものか，そうでないかに分別する仕組みであり，特に Spike と呼ばれる RTT を閾値にする方式が用いられている．AR は，送信者がボトルネックに送出することのでき

たレートを表しており，受信者により計測される．
LDA により，ランダムロスに対してロバストであり，AR によって急激なレート変化を防ぐと同時に，帯域の有効活用を行うことが可能である．また，内部で仮想的に TCP の動作をエミュレートし，仮想 TCP のレート推定に応じたレート制御を行うことで，TCP との親和性を保っている．
VTP の基本的な戦略は図に示すように，輻輳によるロスが発生した際に，TCP がレートを急激に落とすのに対し，VTP は算出した推定レート（AR）までしかレートを落とさない．そして，TCP との公平性を保つために，仮想 TCP が VTP に追いつくまでは送信レートをＡＲのまま維持し，TCP が追いついたら VTP の送信レートを TCP と等しくなるように増加させる．
輻輳回避フェーズにおける仮想 TCP のレート推定には，ewnd(equivalent window)というウィンドウサイズを用いて算出を行う．まず，現在のレート R(i)と RTT(i)から

$$ewnd(i) = R(i) \times RTT(i) \tag{1.2.3}$$

を計算し，これを元に，ウィンドウの増加量を 1，RTT の増加量をΔ RTT(i)と仮定すると，レート R(i+1)は

$$R(i+1) = \frac{\left(R(i) \times RTT(i) + 1\right)}{\left(RTT(i) + \Delta RTT(i)\right)} \tag{2.2.4}$$

として求める．また，レートに対して EWMA (Exponential Weighted Moving Average:指数加重移動平均)をかけることによって，より滑らかなレートの推移を行っている．

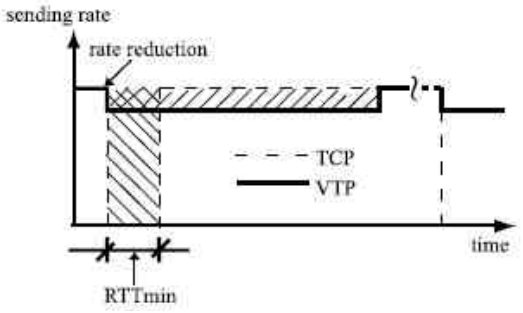

図２．２．８　VTP におけるレート制御[8]

２．３　Mobile IP

移動体通信においては，移動端末（MN）の位置にかかわらず，シームレスな通信サービスを提供できることが要求される．

IP パケットは基本的にはその宛先 IP アドレスに応じてルーティングされるが，各端末には通常それぞれが接続しているネットワーク（ホームネットワーク）に所属する IP アドレス（ホームアドレス：HoA）が割り当てられる．このため，MN が外出先で他のネットワーク（外部ネットワーク）に接続しても，そのままではホームネットワークでの IP アドレス

を使用しているので通信することができない．また，外部ネットワークへ移動する度に動的なアドレス割り当てを行う場合も，TCP や UDP といったインターネット上で用いられている 2 つのトランスポート層プロトコルは，エンドポイントを識別するのに IP アドレスを使用するため，通信中に IP アドレスを変更すると通信はそこで切れてしまう．

インターネット標準である Mobile IP[9]は，端末固有の IP アドレス，つまり HoA の変更を伴わずに，外出先で外部ネットワークに接続する場合でも，あたかもホームネットワークに接続しているかのように同一のネットワーク環境での通信を実現するプロトコルである．移動パケット網では，広範囲にわたって移動する端末が，その IP アドレスの変更を伴わずに，同一の IP アドレスで通信可能となるように Mobile IP を利用できる．

本論文の 1 つ目の提案では，異種ネットワーク間を移動するネットワーク構成を想定しており，そこで，AP を切り替える仕組みとして，以下で説明する方式が用いられている．

Mobile IP の動作概要

Mobile IP の動作概要図を図 2.3.1 に示す．

外部エージェント（FA）は定期的にエージェント広告を送信している．エージェント広告とは，自局の情報をサブネット内に広告するパケットである．MN が外部ネットワークに移動し，このエージェント広告を受け取ると，新しいサブネットに接続したものと判断して気付けアドレス(CoA)を獲得する．ここで，CoA とは，MN がホームネットワーク以外のサブネットで獲得するアドレスである．つまり，インターネット内での MN の位置識別子となり，MN の位置によって変化するアドレスである．MN は FA に対して，CoA と HoA を含んだ登録要求メッセージを送信する．登録要求メッセージを受信した FA は，その登録要求をホームエージェント（HA）に転送する．これによって，HA はつねに MN の CoA を保持することができるのである．

MN に対する通信は次のような手順で行われる（図 2.3.1）．

まず MN との通信相手端末 (CN)は，移動体の HoA を宛先とするパケットを送信する（図 2.3.1 内①）．MN 宛のパケットは，ルーティング方式（IP）に基づいて MN のホームネットワークまで送信される（同②）．MN がホームネットワークにいない場合は，HA は CoA を宛先とする IP ヘッダでパケットをカプセル化して送信する（トンネリングという）（同③）．なお，MN がホームネットワークに接続している場合は，従来と同様のルーティング方式で MN にパケットが送信される．このパケットを受け取った FA は，カプセル化した IP ヘッダを取り除き（デカプセル化という），MN の HoA を参照して MN にパケットを転送する（同④）．

このようにして，MN の移動を意識させない通信が可能となるのである．

なお，MN が送信するパケットは，一部のマルチキャストパケットとブロードキャストパケットを除いて宛先に直接ルーティングされる（同⑤）．

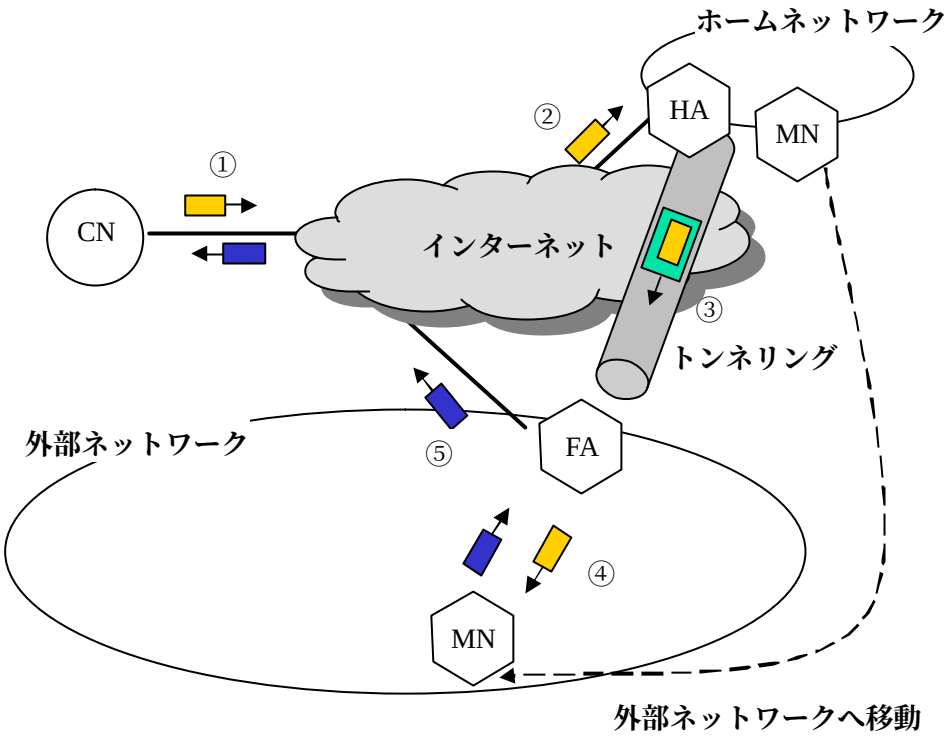

図２．３．１　Mobile IP 動作概要

第三章　異種間ネットワークにおける TFRC の特性改善

３．１　想定環境と問題点

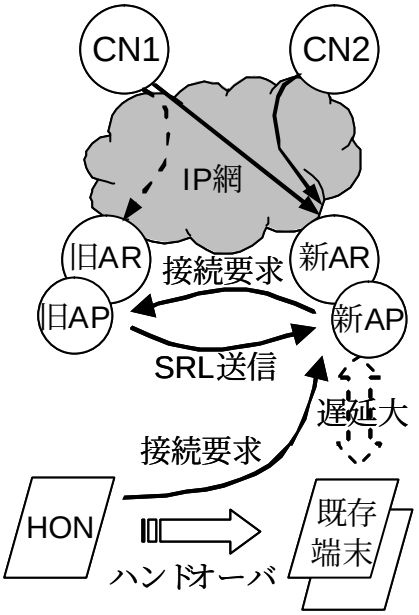

図３．１．１　異種ネットワーク間ハンドオーバ

本論文では図 3.1.1 に示すように，IEEE 802.16-2005 といった，広帯域で低遅延な AP から，携帯電話網といった，相対的に狭帯域で高遅延な AP へとハンドオーバするような環境を想定している．このような異種ネットワーク間におけるハンドオーバ前後では，TFRC のレシーバは，ハンドオーバ前の情報を元にパケット廃棄率を算出してしまうため，ハンドオーバ後のネットワークでは，その情報が正しくないことがある．特にハンドオーバ前の使用回線帯域が，ハンドオーバ後に比べて大きい場合，HON のハンドオーバ後のスループットが他のフローと不公平になる可能性がある．

図 3.1.2 は，MIPv4 を用い，ハンドオーバ前の往復伝播遅延，回線帯域を 30[ms], 10[Mbps], ハンドオーバ後の往復伝播遅延，回線帯域を 300[ms]，2[Mbps]，移動後の既存 TCP フロー数を 4 本とした場合の，HON のスループットを示している．通常の TFRC，及び後述する ACK 管理方式を用いた場合で，公平な帯域である 400[Kbps]を 200[Kbps]近く上回っており，その分既存 TCP フローのスループットを押し下げる傾向があることが分かる．

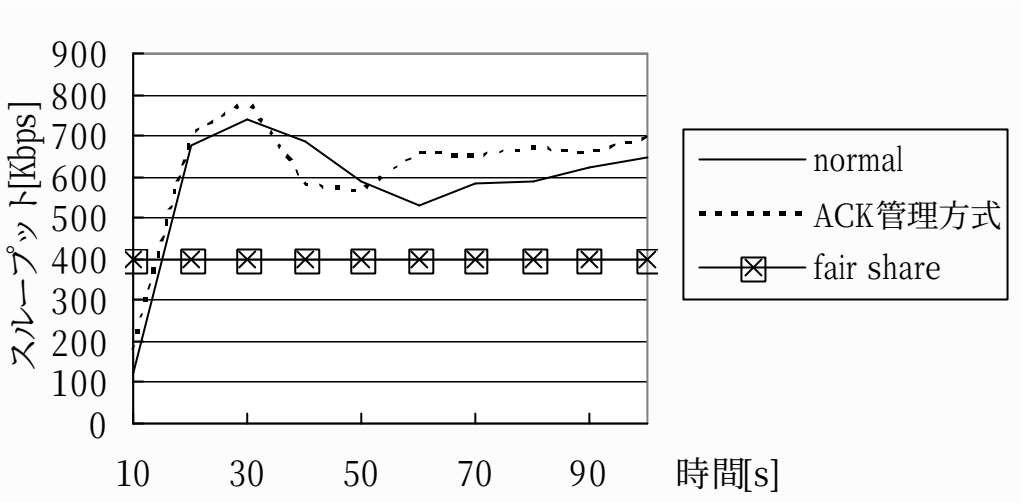

図 3.1.2 TFRC のハンドオーバ後のスループット

また，HON とハンドオーバ後の AP 間の遅延が大きい環境下では，ハンドオーバ手続き後，HON から送られるフィードバックパケットが CN に届くまでに時間がかかり，不要なレート削減やタイムアウトを引き起こす可能性がある．これはセンダの保持する RTT とタイムアウト時間が，ハンドオーバ後の HON とハンドオーバ後の AP 間の遅延より小さい場合に生じる．もしタイムアウトが発生すると，センダではレートを半分に設定するため，不要なレート削減が起こる．

そのため，(1)CN にフィードバックパケットが届くまでの遅延を減らすことで，不要なレート減少を抑え (2)パケット廃棄率の情報をハンドオーバ後の環境に対応させ，既存 TCP フローへの影響を軽減する，ことを可能にする手法を考える必要がある．

そこで，まず HON と新 AP 間のリンク遅延が大きい場合に，不必要なレート削減とタイムアウトの発生を抑制する ACK 管理手法を提案する．ただし，この方式だけでは移動先の既存フローを追い出す傾向があるため，さらにハンドオーバ後の環境に対応させる方法として，AP での ACK 管理を継続させる手法と，AP からの情報に基づき，パケット廃棄率を更新させる手法を提案し，以下で説明する．

３．２　提案手法

３．２．１　AP 介在型 ACK 管理方式

（1）ACK 管理の手順

HON は，ハンドオーバ後の新 AP に対して，MIP (Mobile IP) や DHCP を用いて移動登録を行う際に，送信者のアドレスとポート番号を通知する．次に，MIP や SIP モビリティを用いて HON の移動登録が完了すると，送信者から移動先への TFRC パケットの送信が開始される．これを新 AP がスヌープし，適切なフィードバック情報を生成し，HON の代わりに TFRC ACK を送信者に返送する．TFRC ACK を受信した送信者は，その情報に従っ

て迅速にレートの更新を行う．このように，AP が ACK 生成に関与することで，とりわけ HON と新 AP 間のリンク遅延が大きい場合に，不必要なレート削減とタイムアウトの発生を抑制することができる．

（2）パケットの廃棄手順

HON は，ハンドオーバ後の新 AP に対して，ネットワーク内の自身宛ての滞留パケット量を通知する．新 AP は，自身のパケットバッファ残量を元にパケット廃棄率を算出して，ハンドオーバ端末リスト(HandOver Node List : HONL)を更新する．新 AP 内に入ったパケットは HONL に従って通常のパケットと移動直後のパケットに分類され，宛先が HON の場合，受信パケットはパケットドロッパに送られ，設定されたパケット廃棄率に従って，ランダムに廃棄される．これによって移動直後，移動先ドメインにおける既存ノードへの影響を緩和することができる．

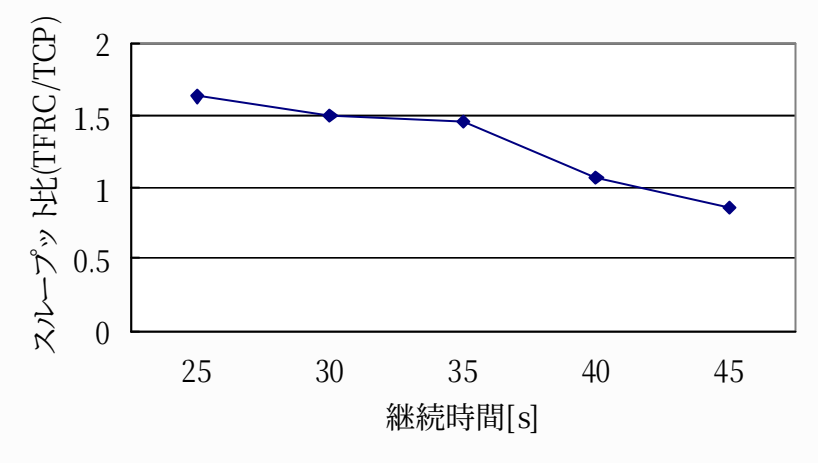

図 3.2.1 継続時間とスループット比

３．２．２　ACK 管理継続方式

3.1 によって，ハンドオーバ前後で HON と新 AP 間のリンク遅延が大きい場合にハンドオーバ後の HON が，その後正しいパケット廃棄率を元にした TFRC ACK を生成できるとは限らないため，ハンドオーバ先のネットワークに悪影響を与える可能性がある．

そこで，AP での ACK 管理をある時間継続し，また同時に，TFRC レシーバから送信される ACK を廃棄する．これはセンダにおいて，ACK 重複を避け，AP による適切な情報を持った ACK によって送信レートを計算させるためである．この動作を継続させることで，HON のパケット廃棄率の情報を徐々にハンドオーバ後の環境に対応させることができる．しかし，その動作を継続させる時間によっては，パケット廃棄率をハンドオーバ後の環境に十分適応できない場合がある．図 3.2.1 は，既存 TCP フローを４本とし，ハンドオーバ前の回線帯域を 10[Mbps]，ハンドオーバ後の回線帯域を 2[Mbps]とし，これらの動作を 25[s]から 45[s]継続させた場合の TFRC と TCP フローの平均スループット比を表している．

25[s]から 35[s]までは，スループットが安定せず，TCP を押し下げる傾向がある．つまり，レシーバ側のパケット廃棄率の情報が，ハンドオーバ後の環境にきちんと適応できていないことが分かる．しかし 40[s]から 45[s]では，既存フローのスループットにほとんど影響を与えることなく，ほぼ公平な帯域使用を実現している．そこで後述するシミュレーション評価では，ACK 管理の継続時間を 40[s]として実験を行った．

３．２．３　パケット廃棄率更新方式

3.2.2 では，AP において ACK 管理を一定時間継続し，レシーバのパケット廃棄率の情報をハンドオーバ後の環境に対応させる手法を提案した．しかし，図 3.1 より，短い継続時間では十分にハンドオーバ後の環境に対応させることができないことがわかる．また，レシーバのパケット廃棄率が落ち着くまで AP における ACK 管理を継続すると，ACK 生成による AP の負担を大きくしてしまう可能性がある．

そこで新たに，ハンドオーバ手続きの中で，HON のパケット廃棄率を AP からの情報によって更新させ，すばやく新しい環境に適応させるパケット廃棄率後進方式を提案する．これにより AP への負荷を軽くし，よりスムーズにハンドオーバ後の環境に対応させることができる．

ハンドオーバ後の環境のパケット廃棄率は，図 3.1 の新 AP がスヌープした情報に基づいて作成する．このパケット廃棄率の情報は，新 AP から MN へとハンドオーバ手続きをする際に上乗せすれば良く，その他移動通信プロトコルにも応用が可能であると考えられる．これを AP 介在型 ACK 管理方式と組み合わせることにより，不必要なレート削減とタイムアウトの発生を抑制し，ハンドオーバ後の環境に対応させることが可能になる．

図 3.2.2 は，MIPv4 を用いた場合の，ハンドオーバ手続きと ACK 管理方式とパケット廃棄率更新方式を組み合わせた場合の適用例を表している．パケット廃棄率更新方式は，新 AP から MN への手続きである，Registration Reply メッセージを拡張することで，また ACK 管理方式は，MN から新 AP への手続きである，Registration Request を拡張することで，適用可能である．

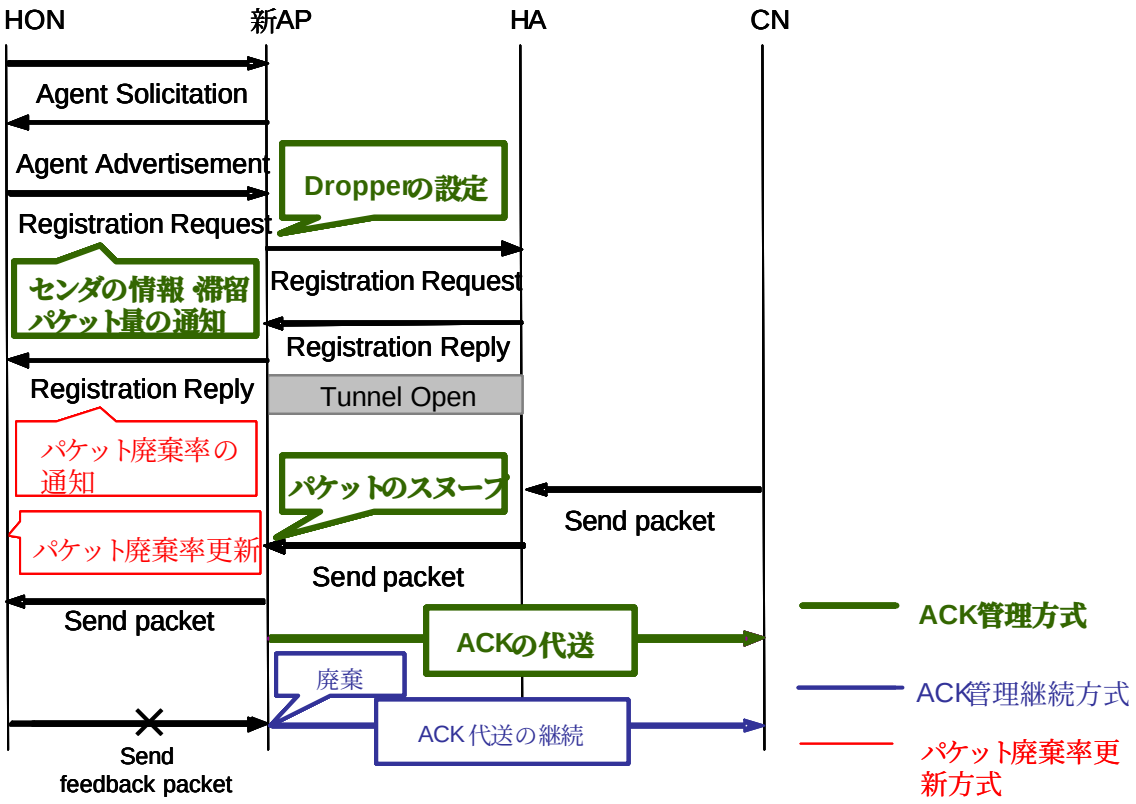

図 3.2.2 MIPv4 と提案手法の適用例

3．3 シミュレーション評価

本章では，通常の HON と，提案手法を加えた場合それぞれについての評価を行う．以下に示す結果は図 3.3.1 に示すネットワークモデルを用い，ns2[7]を用いて評価を行った．

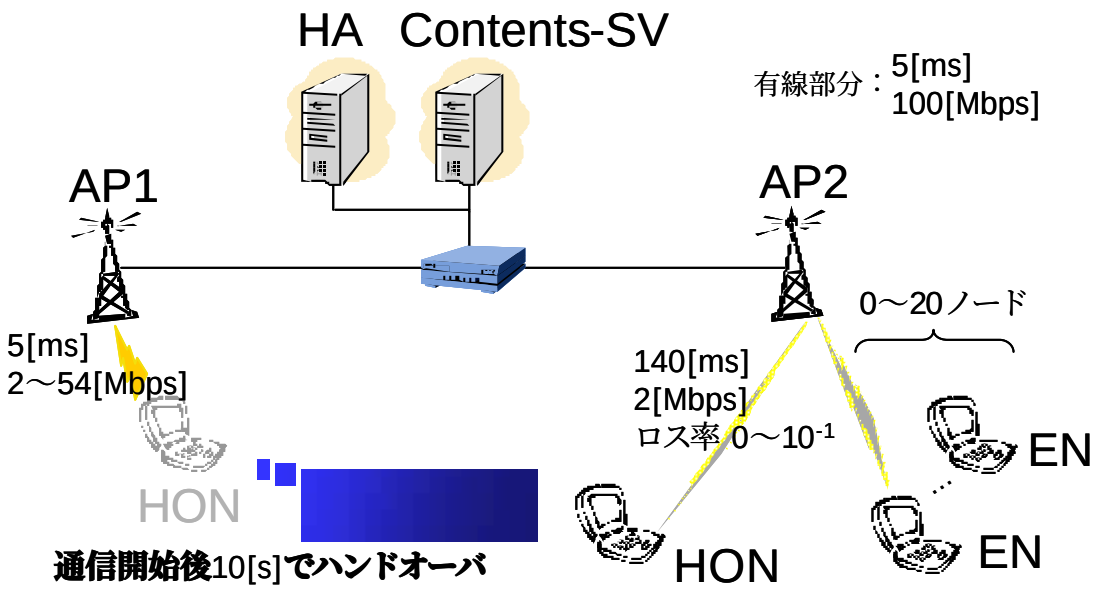

図 3.3.1 ネットワークモデル

３．３．１　既存フローが存在しないとき

図 3.3.1 に示すネットワークモデルで，ハンドオーバ後の AP である AP2 に競合フローがない場合の評価を行う．ネットワークパラメータは図 3.3.1 に示す通りで，ハンドオーバ前の往復伝播遅延時間は 30[ms]，ハンドオーバ後は 300[ms]となる．

(1)TFRC の場合

図 3.3.2 では，通常の TFRC と ACK 管理手法のスループットの比較を，図 3.3.3，3.3.4 では，１つの TFRC フローのみが存在する場合のスループットを示す．図 3.3.3 はハンドオーバ前の回線帯域を 2[Mbps], 11[Mbps], 54[Mbps]と変化させた場合のスループット評価，図 3.3.4 はハンドオーバ前の回線帯域を 10[Mbps]とし，ハンドオーバ後の回線のパケット廃棄率を 10^{-5} から 10^{-1} まで変化させた場合のスループット評価である．ただし，通常の TFRC，ACK 管理方式，パケット廃棄率更新方式については，ハンドオーバ後 5[s]～20[s]間の，ACK 管理継続方式については，継続終了後 35[s]～50[s]間のスループットを計測している．また，ハンドオーバ後の新 AP が保持する情報の初期値として，回線帯域を 2[Mbps]，RTT を 300[ms]として設定した．また図 3.3.4 では，理想的な状態として，ハンドオーバ後の環境で，ハンドオーバを行わなかった際の平衡状態のスループットを示した．

図 3.3.2 では，ACK 管理方式によってスループットの立ち上がりが改善されていることが確認できる．また，タイムアウト回数も同時に計測したところ，通常の TFRC が１０回なのに対し，ACK 管理方式では８回と減少していることを確認した．

図 3.3.3 では，全体的に，ハンドオーバ前の帯域が大きくなるほど，ハンドオーバ後のスループットが大きくなっている．これより，HON がハンドオーバ前のパケット廃棄率の情報に影響を受けていることが考えられる．また，パケット廃棄率更新方式では，どの回線帯域でも，従来より良好なスループットを得ることができている．ただ ACK 管理継続方式では，従来よりもスループットが小さくなってしまうことがある．

これは，AP が保持する情報の初期値によっては，パケットロスが起きる間隔を早め，ACK 管理の継続後のスループットを大きく低下させることがあるためである．また，レシーバ側の情報が上手く更新されないこともあった．これは，ロスのタイミングによってロス率の値が大きく上下するためである．

図 3.3.4 では，パケット廃棄率 10^{-3} 程度までは，提案によってスループットが改善されていることが分かる．しかし，パケット廃棄率が大きくなりすぎると，TFRC 自体が帯域を有効に使うことができないため，スループットが頭打ちになる．このため，タイミングによっ ては，提案手法のほうが，利用帯域が小さくなる場合がある．例えば，パケット廃棄率が 10^{-2} のとき，式(1)に基づいて計算すると，TFRC が使用できるスループットは 326[Kbps]であり，使用可能帯域である 2[Mbps]を大きく下回ることとなる．理想的な場合より他の方式のスループットが大きくなる場合があるが，これはハンドオーバ前の情報の影響を受けているためと考えられる．ACK 管理継続方式では，一様に高いスループットを

得ることができている．これは，一定時間 AP による，レシーバの情報に依存しない ACK によって送信レートが計算されるため，一時的にこのように優れた結果が得られたと考えられる．ただし，その後は TFRC が使用可能なスループットへと収束していく．

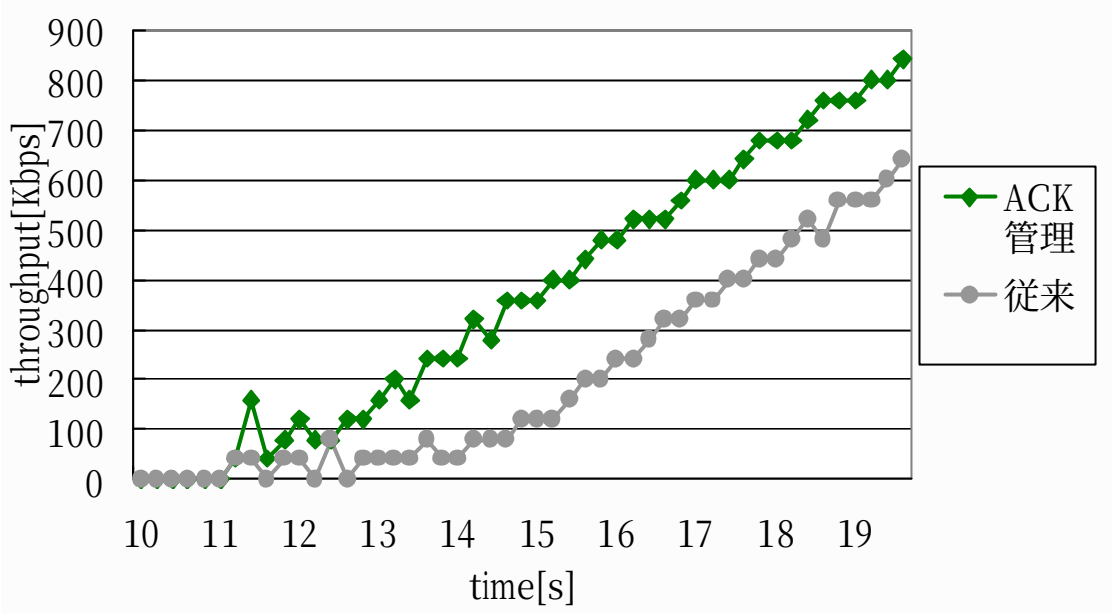

図 3.3.2　ACK 管理方式と従来手法の移動直後のスループット

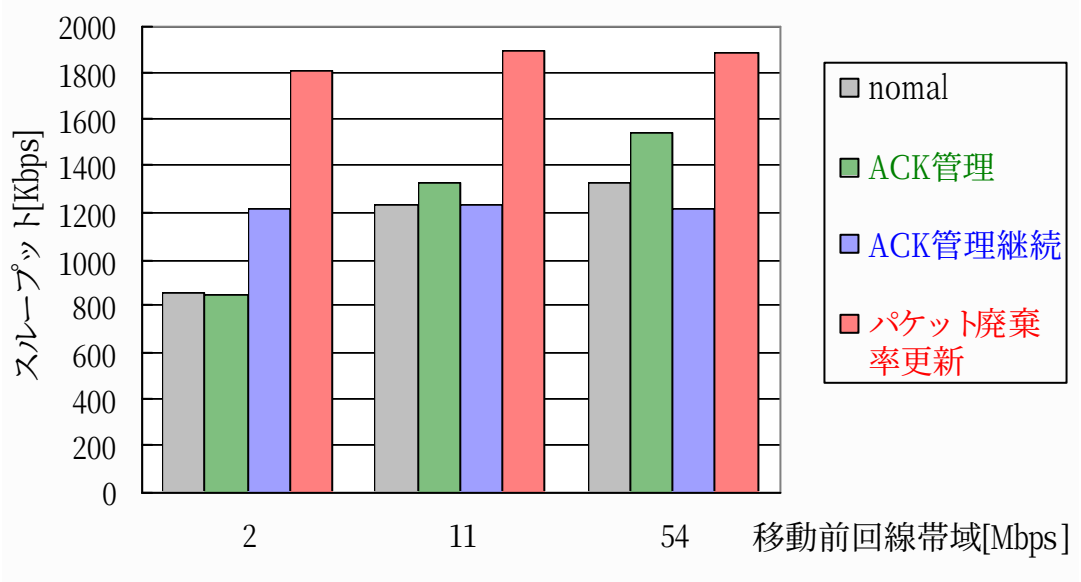

図 3.3.3　単フローの回線帯域とスループット(TFRC)

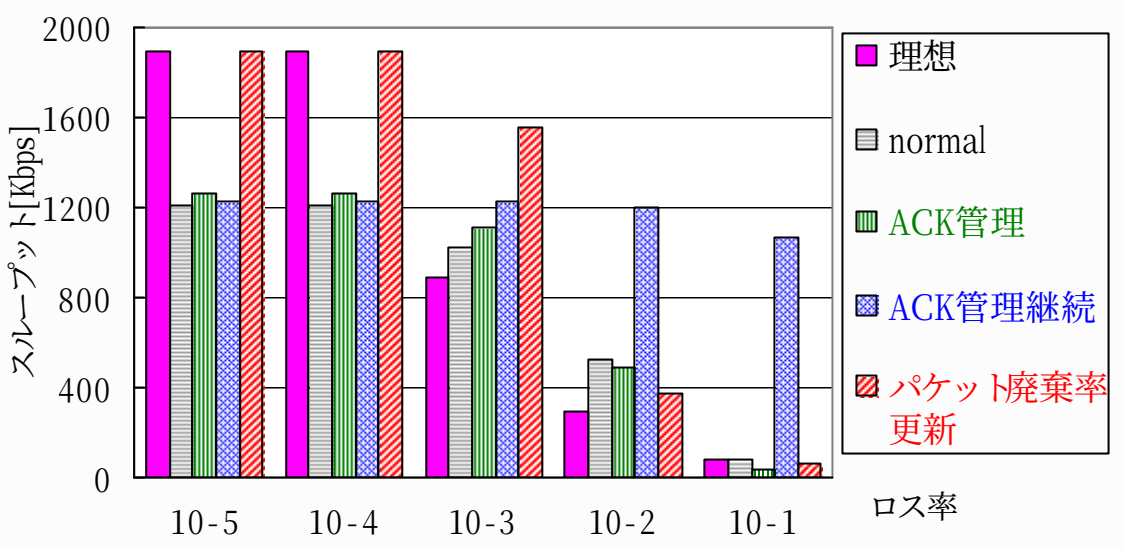

図 3.3.4　単フローのパケット廃棄率とスループット(TFRC)

(2) TFRC + LDA の場合

LDA を組み合わせた場合，ロスが無線によるエラーかどうかを判別し，輻輳によるロスのみでパケット廃棄率を計算することができるため，ワイヤレスエラーに強い制御を行うことができる．以下(1)の図 3.3.3，3.3.4 と同様の実験を TFRC+LDA の場合について行った結果を図 3.3.5，3.3.6 に示す．ただし，LDA が RTT を用いたロスの判別をしているため，ハンドオーバ後 RTT の更新が必要であり，更新した場合の結果を図中に refresh として表した．また図 3.3.6 では，理想的な状態として，ハンドオーバ後の環境で，ハンドオーバを行わなかった際の平衡状態のスループットを示した．

図 3.3.5 では，(1)とほぼ同様の結果が得られた．ただし，ACK 管理継続方式は更新が不安定な場合があった．これは，ロスのタイミングによってロス率の値が大きく上下するためである．

図 3.3.6 では，通常の場合以外で，ロス率の大きさに関わらず帯域を使用することができている．これは，RTT を更新することによって，LDA により適切なロスの判別ができているためであると考えられる．また，ACK 管理継続方式とパケット廃棄率更新方式では，他の方式に比べ若干高いスループットを出すことができている．これは TFRC の場合と同じである．

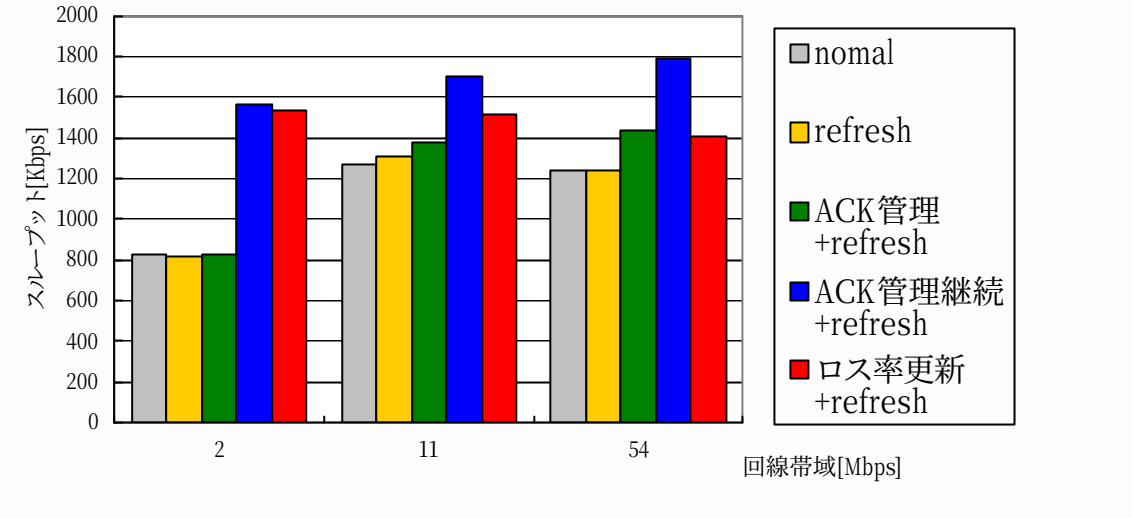

図 3.3.5　単フローの回線帯域とスループット(TFRC+LDA)

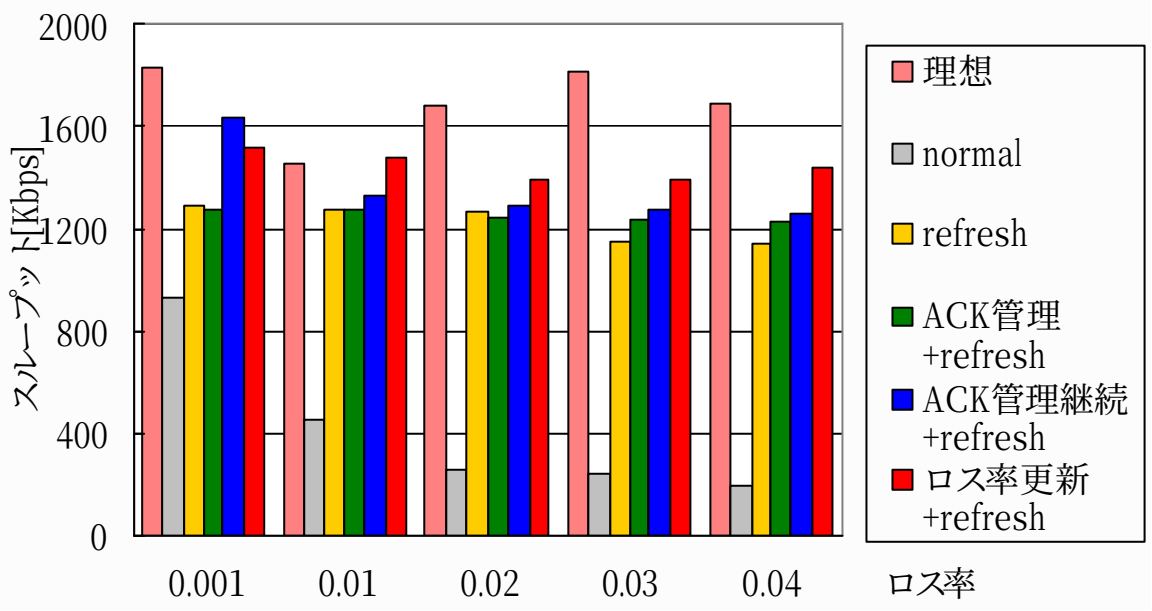

図 3.3.6　単フローのパケット廃棄率とスループット(TFRC+LDA)

３．３．２　既存フローが存在するとき

(1) TFRC の場合

図 3.3.1 のネットワークモデルで，ハンドオーバ後の AP である AP2 に競合 TCP フローを付加した場合の評価を行う．図 3.3.8 はハンドオーバ前の回線帯域を 2[Mbps]，11[Mbps],

54[Mbps]と変化させた場合，図 3.3.9 はハンドオーバ前の回線帯域を 10[Mbps]とし，ハンドオーバ後の回線のパケット廃棄率を 10-5 から 10-1 まで変化させた場合のスループット評価結果を示している．図 3.3.8，図 3.3.9 ともに，競合 TCP フロー数は 4 とする．また図 3.3.9 では，理想的な状態として，ハンドオーバ後の環境で，ハンドオーバを行わなかった際の平衡状態のスループットを示した．

図 3.3.7 では，図 3.3.2 と同様に，ACK 管理方式によってスループットの立ち上がりが改善されていることが確認できる．また，タイムアウト回数も同時に計測したところ，通常の TFRC が１０回なのに対し，ACK 管理方式では８回と減少していることを確認した．

図 3.3.8 では，通常の TFRC，ACK 管理方式の場合，ハンドオーバ前の回線帯域が大きいほど，ハンドオーバ後に影響を残し，TCP のスループットを押し下げていることが分かる．一方 ACK 管理継続方式，パケット廃棄率更新方式では，ハンドオーバ前の回線帯域に関わらず，ほぼ公平な帯域使用を実現していることがわかる．

図 3.3.9 では，図 3.3.4 とほぼ同様にパケット廃棄率が大きくなるにつれて，使用帯域が小さくなっている．ACK 管理継続方式についても，AP が保持する回線情報が，既存フローから得られるものに基づいているため，既存フローと同様に，パケット廃棄率に反比例して使用帯域が小さくなっている．また，その中でも ACK 管理継続方式，パケット廃棄率更新方式については，パケット廃棄率に関わらず，既存 TCP フローのスループットを押し下げる傾向があることがわかる．一方，ACK 管理継続方式，パケット廃棄率更新方式では，ほぼ公平な帯域使用ができている．

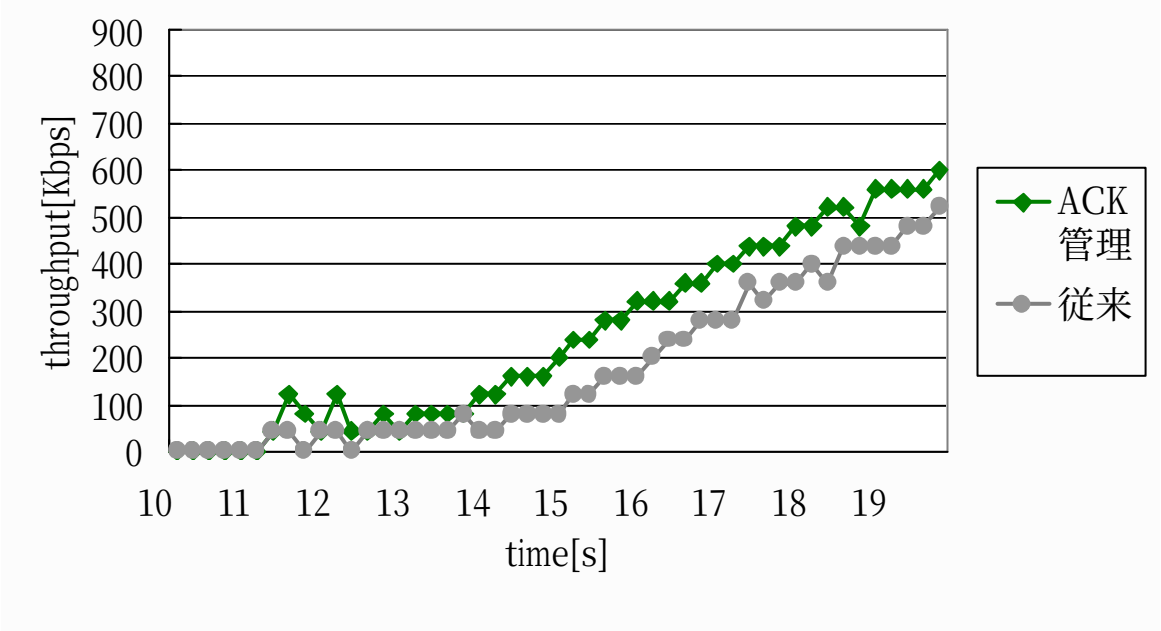

図 3.3.7　既存フロー存在時の ACK 管理方式と従来手法の移動直後のスループット

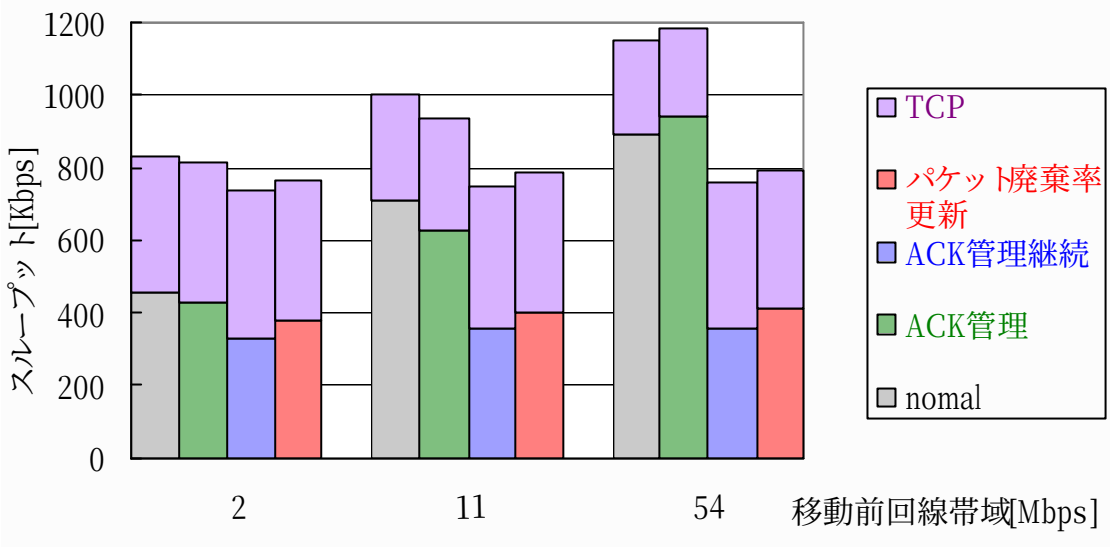

図 3.3.8　既存フロー存在時の回線帯域とスループット(TFRC)

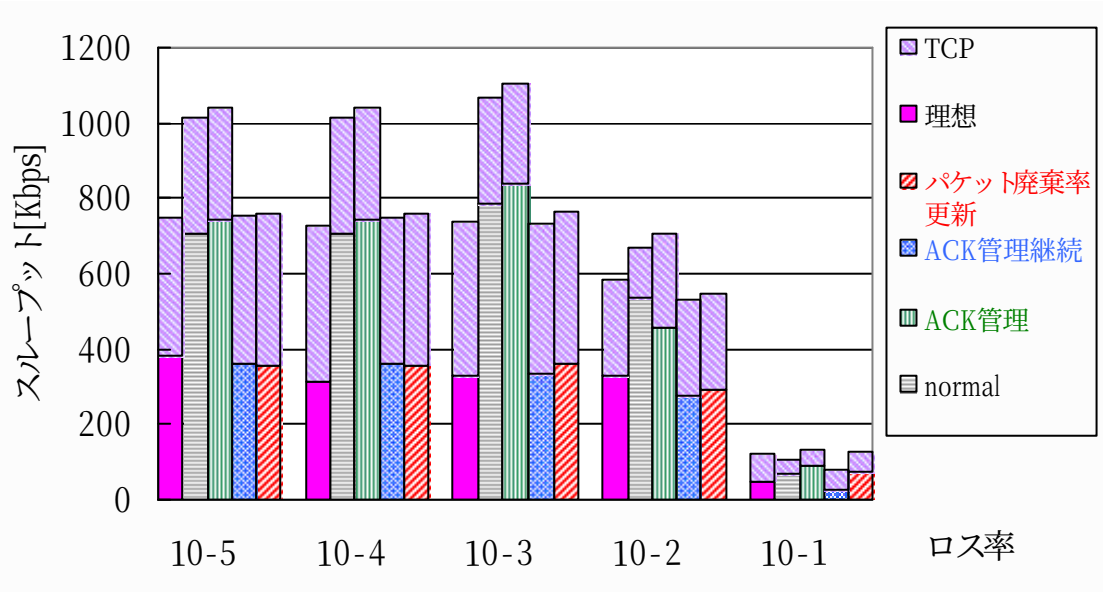

図 3.3.9　既存フロー存在時のパケット廃棄率とスループット(TFRC)

(2) TFRC+LDA の場合

以下(1)の図 3.3.8，図 3.3.9 と同様の実験を TFRC+LDA の場合について行った結果を図 3.3.11，3.3.12 として示す．ただし，LDA が RTT を用いたロスの判別をしているため，ハンドオーバ後 RTT の更新が必要であり，更新した場合の結果を図中に refresh として表した．また図 3.3.12 では，理想的な状態として，ハンドオーバ後の環境で，ハンドオーバを行わなかった際の平衡状態のスループットを示した．

図 3.3.11 では，TFRC の場合の図 3.3.8 とほぼ同様の傾向が見られた．
また図 3.3.12 では，ACK 管理方式と refresh 方式が高いスループットを得ている．これは，RTT の更新により LDA が適切なロスの判断を行っているためであると考えられる．しかし，ACK 管理継続方式とパケット廃棄率更新方式では，TCP と同程度か，それより若干高い程度のスループットしか出ていない．これは，AP でスヌープしている情報が，既存フローのスループットや RTT であり，それに基づいた情報を与えられたため，TCP と同じくらいの性能になっていると考えられる．その後通信を続けると，両方式とも ACK 管理方式，refresh 方式と同程度の性能に収束した．

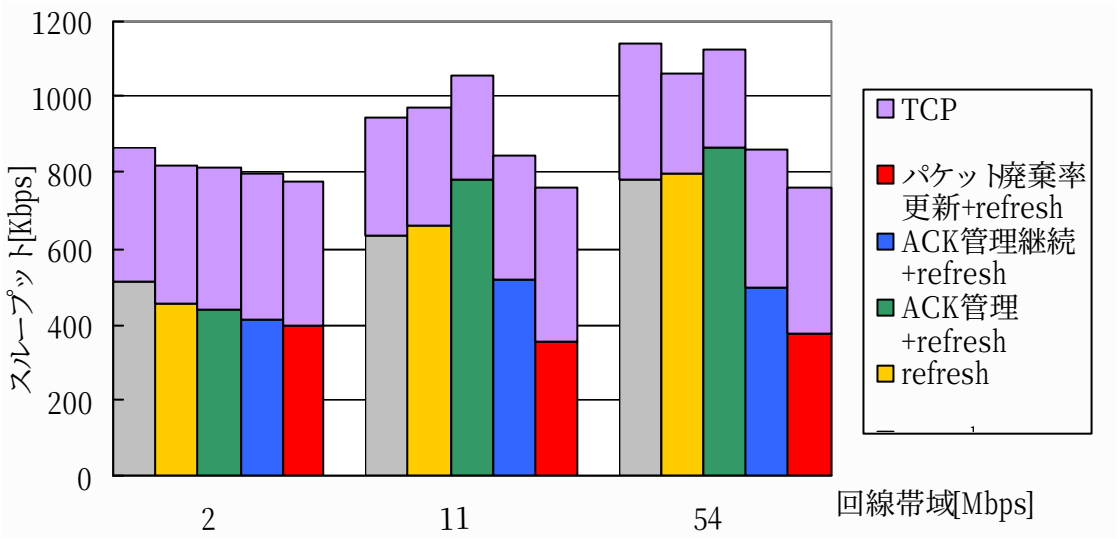

図 3.3.11　既存フロー存在時の回線帯域とスループット(TFRC+LDA)

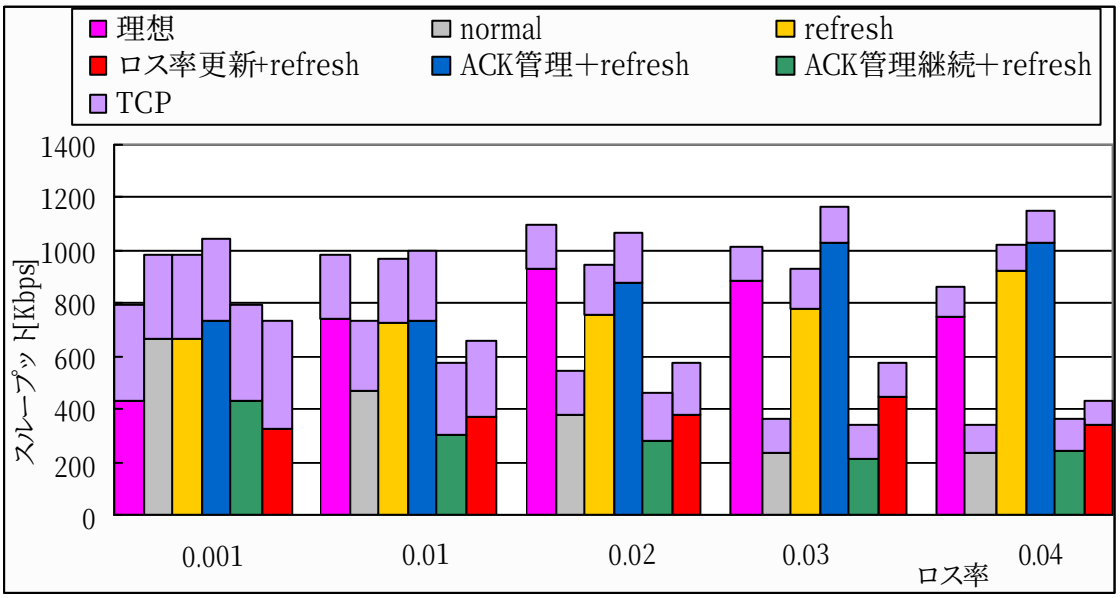

図 3.3.12　既存フロー存在時のパケット廃棄率とスループット(TFRC+LDA)

３．４　まとめと今後の課題

本論文では，AP の ACK 管理手法を元に，TFRC レシーバの保持する情報を，適切にハンドオーバ後の環境に対応させる手法を提案した．１つは AP での ACK 管理を継続し，もう１つはレシーバ側のパケット廃棄率を AP の情報を元に更新させるものである．本提案では，AP と HON，TFRC の拡張が必要となるが，送信端末，アプリケーションへの変更は不要である．それぞれについてシミュレーションを行い，パケット廃棄率，ハンドオーバ前の使用可能な回線帯域，既存 TCP フロー数に依らず，その有効性を示した．ただし，AP での ACK 管理を継続する方式では，既存フローが存在しない場合，適切に対応できない可能性があることが分かった．また更新のタイミングによってはレシーバの情報が上手く更新されないことがあり，適切なタイミングを計ることが課題である．

第四章　RTT 公平性を実現する UDP 輻輳制御方式の提案

４．１　従来の輻輳制御方式の問題点

従来の AIMD(Additive Increase Multiple Decrease)型の輻輳制御方式である Reno や BI TCP, HighSpeed TCP, Scalable TCP は, 輻輳ウィンドウの増加周期が RTT に比例する．つまり，RTT の長いコネクションほどウィンドウの増加速度が遅くなり，輻輳ウィンドウの増加速度は RTT に反比例する傾向がある．[4]ではそれぞれの輻輳制御方式を解析し，RTT の異なるフロー同士のスループットの比が

$$\frac{w_1}{w_2} \propto \left(\frac{RTT_2}{RTT_1} \right)^{\frac{d}{1-d}} \qquad (4.1.1)$$

に収束するとしている．ここで, d の値はそれぞれ Reno と BI TCP が 0.5, HighSpeed TCP が 0.82，Scalable TCP が 1 である．式(2)より，相対的に輻輳ウィンドウ増加速度の遅いコネクションはネットワークの状態を速やかに反映したレート調整を行うことが出来ず，輻輳ウィンドウ増加速度の速いコネクションに利用可能な帯域を多く奪われてしまうことになる．結果として，複数コネクション内に RTT の異なるフローが存在する場合，帯域が不公平に割り当てられるという問題が生じる．

この傾向は，Reno の AIMD アルゴリズムや平衡状態のレート式を用いた RAP，TEAR, VTP，TFRC などにも，同様に見られることが予想される．

表 4.1 は，TFRC，VTP を用い，ボトルネック回線の帯域を 11[Mbps]，フロー１の RTT を 10[ms]，フロー２の RTT をフロー１の１～６倍に変化させた場合のスループット比(フロー２/フロー１)を表している．

本論文では，Reno の動作をエミュレートする VTP に着目し，RTT の異なるフロー同士でも，公平な帯域分配を実現するように改善を行う．

RTT 比	1	3	6
Reno	0.999912	0.097165	0.103266
TFRC	1.0169	0.543656	0.276768
VTP	0.835414	0.086469	0.053998

表４．１　RTT を変化させた場合のスループット比

４．２　提案手法

VTP は TCP の動作をエミュレートし, そのレート推定に応じたレート制御を行っているので，理想的には平衡状態で Reno のスループットと一致する．Reno の平衡状態における送信レート$\bar{R}$は，式 2.1.2 に収束することが予想される．

TCP Libra では，この送信レートが RTT に依存しないように,

$$a = g\frac{RTT^2}{RTT + RTT_0} \approx gRTT^2$$

$$b = \frac{RTT_1}{2(RTT + RTT_0)} \approx \frac{1}{2}$$

と設定されている．これを式 2.1.2 に代入すると，式 2.1.3 のようになり，RTT に非依存となっていることが分かる．

この設定を Reno の動作をエミュレートしている VTP に応用することで，VTP においても RTT 公平性が実現できることが期待される．VTP では，Reno の増加係数にあたるウィンドウの増加量が，1 となるようにレートの計算が行われている．これを RTT^2 に比例するパラメータとして式 2.2.4 を置き換えてやると，

$$R(i+1) = \frac{\left(R(i) \times RTT(i) + kRTT(i)^2\right)}{\left(RTT(i) + \Delta RTT(i)\right)} = \frac{\left(R(i) + kRTT(i)\right)}{\left(1 + \Delta RTT(i) / RTT(i)\right)} \qquad (4.1.2)$$

となり，Libra のように，平衡状態でのスループットが RTT に依存しないレート制御となる．直感的にも，ΔRTT(i)=0 だとすると，式 4.1.2 のレート増加量は kRTT(i)となり，1[s]間に 1/RTT 回レートを増加させることから，その平均増加量は

$$kRTT(i) \times 1/RTT(i) = k \qquad (4.1.3)$$

となり，1[s]辺りに増加させるレート量が RTT に依存しないことが分かる．

４．３　シミュレーション結果

図 4.3.1 に示すようなトポロジで実験を行った．S1～S4 はセンダ，R1～R4 はレシーバを表し，センダは同番号のレシーバと通信を行うものとする．また，ボトルネックのバッファは帯域遅延積分あるとする．今回は実験的に，$k=(0.02)^2$ と設定した．

４．３．１　ロス率を変化させたとき

図 4.3.1 のうち，S1～R1 のみで通信を行い，D1=10[ms]とした場合の結果を図 4.3.2 に示す．ただしボトルネックリンクでは，パケットロス率を 0～0.05 まで変化させている．

Reno，TFRC ではロス率が大きくなるにつれスループットが低下してしまっている．つまり Reno や TFRC では，ロス率が高くなると帯域を有効に活用できなくなる傾向があることが分かる．それに対し VTP と Proposal ではロス率が大きくなっても，安定して高いスループットが維持できていることが分かる．つまり，VTP と Proposal では，ロス率に関わらず，帯域を有効に活用できることが分かる．

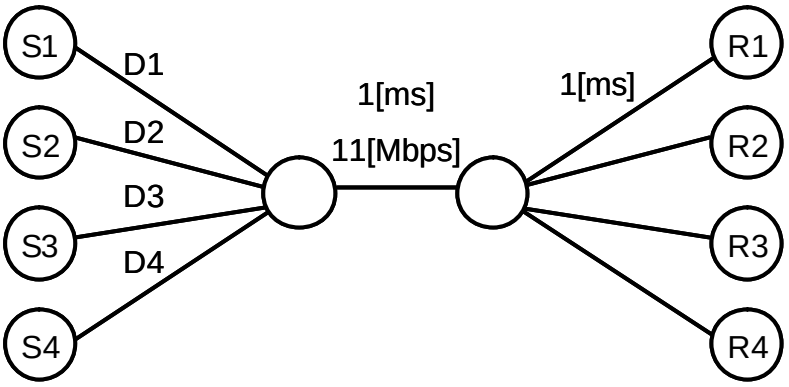

図４．３．１　実験トポロジ

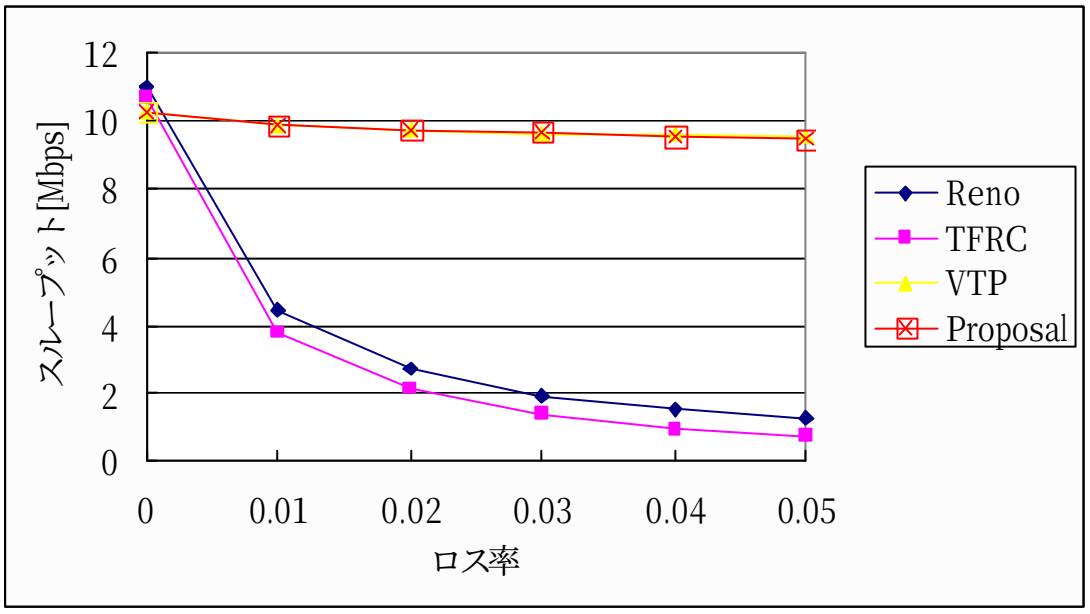

図４．３．２　ロス率を変えた場合のスループット

４．３．２　RTT の異なるフローが混在するとき

リンク遅延 D1～D4 はそれぞれ表４．３に示すような値に設定した．

	D1	D2	D3	D4
Delay [ms]	3	8	28	38

表４．３　遅延の設定値

図 4.3.3～4.3.6 にそれぞれの制御方式でのスループット遷移を示す．図 4.3.3，4.3.4，4.3.5 より，Reno，TFRC，VTP が RTT の小さなフローに他のフローがスループットを奪われているのが分かる．これは Reno と VTP の場合，式 2.1.2 から，TFRC の場合，式 2.2.1 から平衡状態の送信レートが RTT に反比例する傾向が現れているためと考えられる.一方，図 4.3.6 より，Proposal ではどの RTT のフローでもほぼ公平に帯域が分配できている．提案

によってスループットが RTT に依存しないことが分かる.
この結果を数値的に分析するため，それぞれの制御方式で得られたスループットから，Fairness Index を計算し，表 4.3.2 に示した．これは，

$$F_i = \frac{\left(\sum_{i=1}^{N} x_i\right)^2}{N \times \sum_{i=1}^{N} x_i^{\,2}} \qquad (4.3.1)$$

で与えられる, 0 < Fi < 1 となる値であり, 1 に近づくほど公平性が高いことを示す. 表 4.3.2 より，提案では他の制御方式に比べ最も 1 に近い値をとっており，高い RTT 公平性があることが分かる.

	Reno	TFRC	VTP	Proposal
Fairness Index	0.491138	0.815129	0.472159	0.951139

表４．３．２　Fairness Index

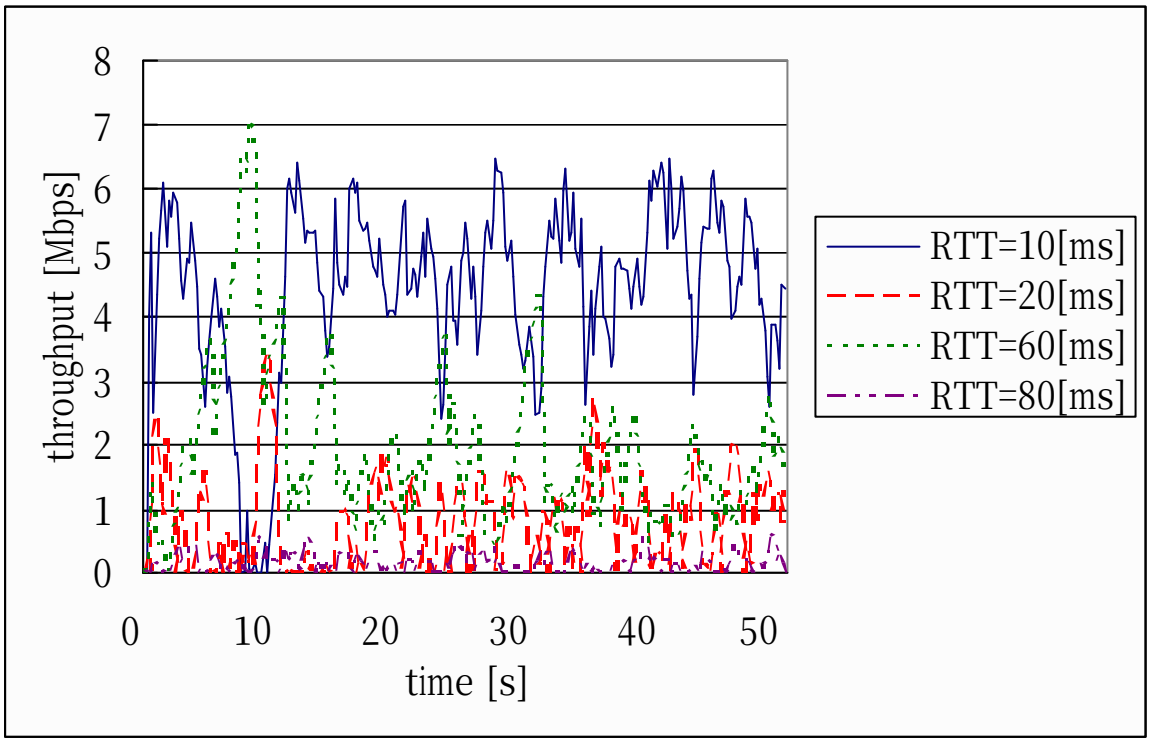

図４．３．３　Reno のスループット遷移

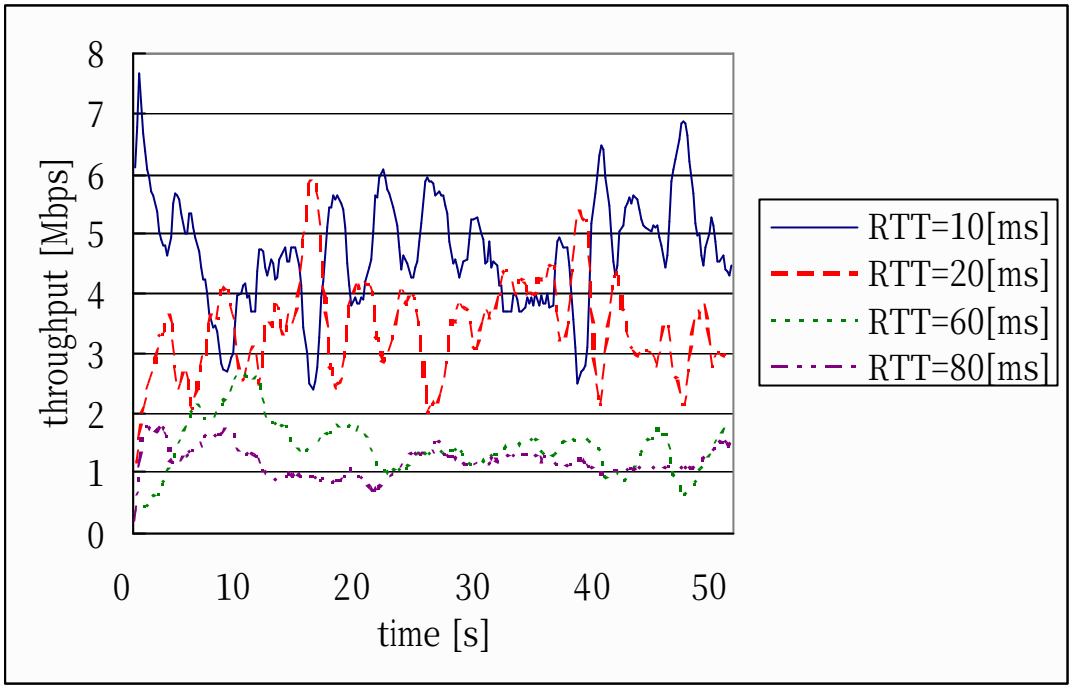

図４．３．４　TFRC のスループット遷移

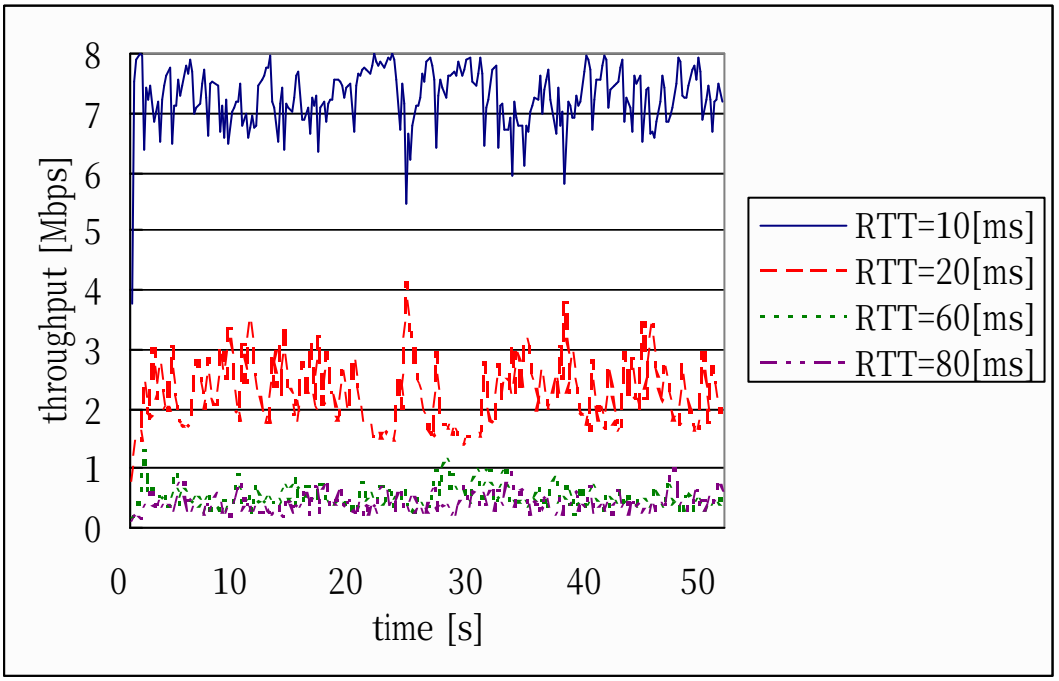

図４．３．５　VTP のスループット遷移

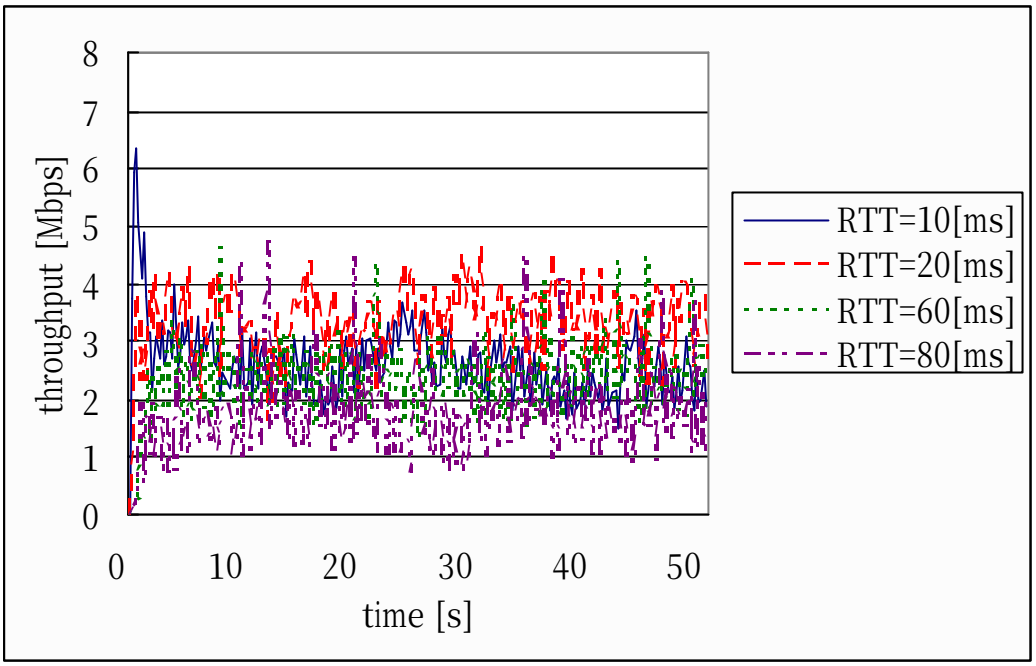

図４．３．６　Proposal のスループット遷移

４．３．３　TCP と競合させたとき

図 4.3.7 **は** flow1，flow2 **を** Proposal，flow3，flow4 **を** Reno **とした場合のそれぞれの平均スループットを表している．**RTT **はそれぞれのフローで同じとし，**delay **は** 3，8，13，18，23，28[ms]**と変化させている．**

提案では，増加量を１とする代わりに，$kRTT^2$ **としている．**RTT **の同じ** Reno **のフローと競合した場合，**k **の設定によっては，その増加量に差が出るため，**Reno **のフローに影響を及ぼす可能性がある．今回の場合，**RTT **が**$\sqrt{k}$ **の設定値と同じ** 20[ms]**であれば** Reno **と高い親和性があるが，**RTT **が**$\sqrt{k}$ **より大きいと増加幅**>1 **となり，**Reno **のスループットを押し下げる傾向がある．図** 4.3.7 **では** RTT **が** 30～60[ms]**の区間がそれにあたる．同様に** RTT **が**$\sqrt{k}$ **より小さいと増加幅**<1 **となり，**Reno **に押し下げられる傾向がある．図** 4.3.7 **では** RTT **が** 10[ms]**の場合がそれにあたる．**

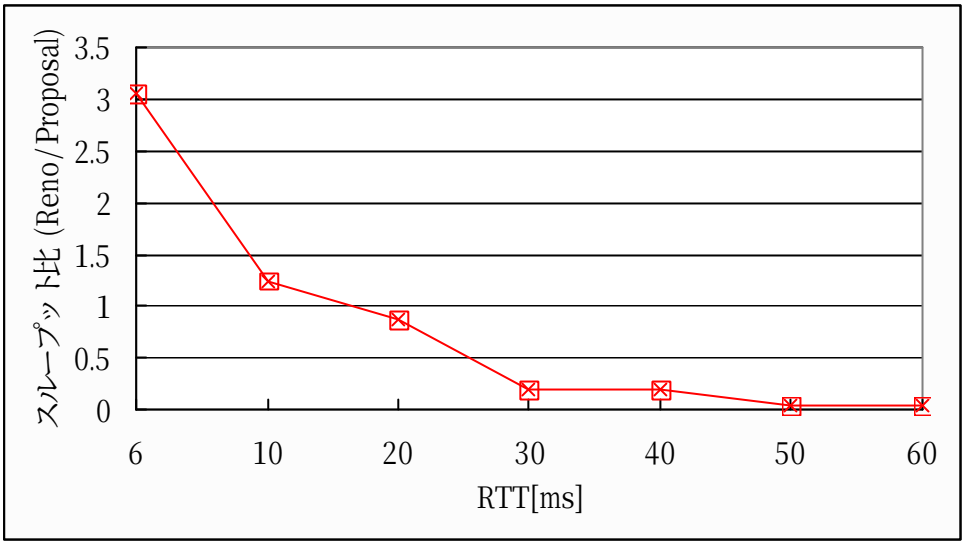

図４．３．７　Reno vs Proposal

４．４　まとめと今後の課題

VTP を元に, 平衡状態の送信レートが RTT に依存しないように, その増加量を適応させた. シミュレーションを行い，VTP の機能として，ロス率が高い場合でも効率的に帯域を使用することができ，かつ RTT の異なるフローが混在するような場合でも公平に帯域を分配することができることを確認した.

ただし，増加量を決めるパラメータ k と，競合する Reno との RTT が一致しない場合，どちらかのスループットが押し下げられる傾向があるため，適切なパラメータ設定の検討が必要になる.

第五章　まとめ

５．１　総括

本論文では，２つの提案についてそれぞれ検証を行った．
１つ目の提案では，異種間ネットワークにおいて TFRC を使用し，AP で ACK の代返を行って到着遅延を減少させる方式を提案した．また ACK の代返の継続や，AP からの情報を元にレシーバの情報を更新させることで，ハンドオーバ後の環境に適応する方式を提案した．またそれを元に，ハンドオーバ前後の帯域やロス率を変化させたシミュレーションを行い，その有効性を確認した．
２つ目の提案では, VTP を使用し, RTT の異なるフロー同士が公平になる方式を提案した．またそれを元に，ロス率や RTT を変化させたシミュレーションを行い，その有効性を確認した．

５．２　今後の課題

1 つ目の提案では，ACK 管理継続方式の更新のタイミングについて，検証が必要である．また本論文では，ハンドオーバ後レシーバの情報を更新する際，AP の協調動作が必要になる．これをレシーバ側のみで，L2 通知などを元にロス率等の情報の重み付けを変え，すばやく移動後の環境に対応させることも考えられる．また TFRC 以外の制御方式での動作規定も課題となる．
２つ目の提案では，増加量を決めるパラメータ k と，競合する Reno との RTT が一致しない場合，どちらかのスループットが押し下げられる傾向があるため，適切なパラメータ設定の検討が必要になる．また，スケーラビリティが要求される場合，増加量を帯域に合わせ，適切に変化させることも必要になる．

参考文献

[1]W. Richard Stevens: "TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms," IETF RFC 2581, 1999.

[2]Tom Kelly, "Scalable TCP: Improving Performance in Highspeed Wide Area Networks. "Computer Communication Review 32(2), April 2003

[3]S.Floyd: “Highspeed TCP for Large Congestion Window”, IETF RFC3649, 2003.

[4]L. Xu, K. Harfoush and I. Rhee: “Binary Increase Congestion Control for Fast, Long Distance Networks”, in Proc. of INFOCOM 2004.

[5]G. Marfia, Ed.,”TCP-Libra: Exploring RTT Fairness for TCP”, UCLA Computer Science Department Technical Report # UCLA-CSD TR-050037

[6]Handley, M., Floyd, S., Pahdye, J., and Widmer, J. :“TCP Friendly Rate Control (TFRC): Protocol Specification” RFC 3448, Proposed Standard, January 2003

[7]Song Cen et al, ”End-to-End Differentiation of Congestion and Wireless Losses”, IEEE/ ACM Transactions on Networking, 11(5):703-717.

[8]Guang Yang et al, " Smooth and efficient real-time video transport in the presence of wireless errors ",ACM Transactions on Multimedia Computing,Communications and Applications, vol.2, issue 2,May 2006.

[9]C. Perkins, Ed., “IP Mobility Support,” Internet RFC2002, Oct. 1996.

[10]E. Kohler, M. Handley and S. Floyd: “Designing DCCP:Congestion Control Without Reliability”, Submitted to ICNP (2003).

[11]J. Padhye, V. Firoiu, D. F. Towsley, and J. F. Kurose, “Modeling TCP Reno Performance: A Simple Model and Its Empirical Validation,” IEEE/ACM Trans. networking, vol.8, no.2, pp.133-145, April 2000.

謝辞

本研究を行うにあたり，丁寧なご指導と適切なご助言をして下さった甲藤二郎教授に心より感謝致します.
また, 様々な助言やご指導をして下さった KDDI 研究所の泉川さんには大変お世話になり，感謝申し上げます.
最後に，３年間研究生活をともにしてきました修士 2 年の皆様をはじめ，円滑な研究を支えてくださいました甲藤研究室の皆様，家族に感謝致します.
ありがとうございました.

平成 20 年 2 月 7 日

藤川　知樹