Title	Improvement of Reno's Performance using Conversion Technique into HighSpeed TCP
Author(s)	磯垣, 順
Citation	
Issue Date	2005-03
Type	Thesis or Dissertation
Text version	author
URL	http://hdl.handle.net/10119/1912
Rights	
Description	Supervisor:井口　寧, 情報科学研究科, 修士

修 士 論 文

Improvement of Reno's Performance using Conversion Technique into HighSpeed TCP

北陸先端科学技術大学院大学
情報科学研究科

磯垣 順

2005 年 3 月

修 士 論 文

Improvement of Reno's Performance using Conversion Technique into HighSpeed TCP

指導教官　井口 寧 助教授

審査委員主査　井口 寧 助教授
審査委員　松澤 照男 教授
審査委員　篠田 陽一 教授

北陸先端科学技術大学院大学
情報科学研究科

310007 磯垣 順

提出年月: 2005 年 2 月

概 要

現在インターネットで用いられている多くの処理系においては、TCP の輻輳制御方式は 4.3BSD(Reno)[Jac88] に基づいて行なわれている。近年普及している広帯域ネットワークにおいては、Reno はパケットが肯定された場合における輻輳ウィンドウの増加量が定数的であるため、ネットワークの帯域幅一杯の輻輳ウィンドウサイズに到達するには長い時間を要することが知られている。しかしながら、インターネットにおいてはパケットが無損失の環境など存在し得ないから、Reno の通信帯域幅は実質的に低くなってしまうことが知られている。

この問題に対処するため HighSpeed TCP[Flo03] と呼ばれる輻輳制御方式が提案されている。HighSpeed TCP では回線の帯域幅に対し比例的な輻輳ウィンドウの増加量となるよう、次の輻輳ウィンドウサイズの増加量を導出した輻輳制御方式であり、将来における広帯域高遅延の環境下においても十分なスループットが得ることができると期待されている。

しかしながら、実際のネットワークにおいて HighSpeed TCP を導入することには 2 つの問題があることが知られている。一つ目の問題は、HighSpeed TCP を Reno と併用した場合、Reno の実効帯域幅が大幅に低くなってしまうことである。本現象は、HighSpeed TCP のフローによってルータにおけるキューバッファが埋め尽されてしまい、結果として、Reno のスループットが大幅に減少してしまうことによって生ずる、不公平性の問題である。二つ目の問題は、一般に輻輳制御方式を変更するためには計算機のオペレーティングシステムを置き換える必要があることである。

本論文では、これらの Reno の問題を解決するために、Reno から HighSpeed TCP への変換を行うプロキシ機構を利用し以上の問題を解決する方法について試み、プロキシ機構の実装及び評価を行なった。

評価のために SuperSINET ナノテク VPN を用い、10 本の HighSpeed TCP のフローの影響下における Reno の性能と、HighSpeed TCP にフローを変換するプロキシを用いた場合のスループットの測定を netperf[Jon] を用いて行い、比較を行なった。

実験より、10 本の HighSpeed TCP による影響下で Reno を利用した場合は、Reno のスループットは 70.7Mbps であるが、プロキシ機構によって HighSpeed TCP に変換し中継を行なった場合では、93.0Mbps の速度を得ることができた。

本実験より、現在の計算機性能を考慮すると課題はのこるものの、プロキシによる TCP 中継は、HighSpeed TCP における公平性問題の解決に有効であることが分った。

本論文では以上の問題の解決に加え TCP 中継におけるレイテンシに着目し、明示的輻輳通知機構 (ECN) によって各種バッファに滞留するパケットの数を削減するよう、カーネル機構を拡張しユーザプロセスから輻輳経験返送 (ECE) を返す方法について試みた。結果として、ユーザプロセスプログラムの走行速度からこれによるコントロールは行うことができなかったのだが、パケットの到達間隔との同期する別の機構等の介入があれば可能

であると推測される。本論文では試みたユーザプロセスからの制御方法とその結果について報告する。

目 次

第 1 章　はじめに **1**

第 2 章　関連研究 **3**
2.1 広帯域・高遅延環境向けの輻輳制御方式 3
2.1.1 Reno の問題点 3
2.1.2 Reno と比較した場合の HighSpeed TCP の相異点 5
2.1.3 TCP Vegas 6
2.2 TCP の公平性 7
2.2.1 TCP の公平性 7
2.2.2 キューイングによる輻輳状態分離 8
2.2.3 まとめ 8
2.3 既存の性能改善のためのプロキシ機構 9
2.4 まとめ 9

第 3 章　提案する装置 **10**
3.1 装置の概要 10
3.2 本装置におけるフロー毎のパケットの流れ 10
3.3 実装方法と本実装における動作 10

第 4 章　提案装置の評価 **12**
4.1 実験方法 12
4.2 不公平性の実験 12
4.3 不公平性改善の実験 14
4.4 他にフローがない状態におけるスループットの実験 14
4.5 考察 15

第 5 章　改良の試み **16**
5.1 はじめに 16
5.2 TCP Splitting 方式における中継の概要 16
5.3 中継時のレイテンシ 18

5.3.1 各実装方式における通信回線特性が通信に与える影響を調べる実験の実験結果 . . . 18
5.3.2 バッファの制御を行わない中継機構の実験 . . . 19
5.4 ソケットバッファサイズが通信レイテンシに与える影響 . . . 19
5.5 明示的輻輳通知機構の拡張 . . . 22
5.6 明示的輻輳通知 API による通信レイテンシの改善アルゴリズムの提案 . . . 22
5.7 評価方法 . . . 25
5.7.1 評価項目 . . . 25
5.7.2 評価環境 . . . 25
5.8 計測結果 . . . 25
5.8.1 性能の低下の原因 . . . 26
5.8.2 RED ライクな ECN 制御方法による実験 . . . 26
5.8.3 問題点 . . . 33

第 6 章 まとめ 34
6.1 本研究のまとめ . . . 34
6.2 研究より発生した課題 . . . 34
6.2.1 実装方式に関する考察 . . . 34
6.2.2 最適なパラメタを求める方法 . . . 35
6.2.3 性能の評価方法等の改善 . . . 36

第 7 章 付録 37
7.1 実際のネットワークの挙動 . . . 37
7.1.1 本実験の目的 . . . 37
7.1.2 実験の方法 . . . 37
7.1.3 結果及び考察 . . . 39
7.2 Dummynet の挙動に関する調査 . . . 39
7.2.1 概要 . . . 39
7.2.2 結果 . . . 39
7.2.3 考察 . . . 43

第1章　はじめに

現在インターネットで用いられている多くの処理系においては、TCP の輻輳制御方式は 4.3BSD(Reno)[Jac88] に基いて行なわれている。近年普及している広帯域ネットワークにおいては、Reno はパケットが肯定された場合における輻輳ウィンドウの増加量が回線容量に対して定数的であるため、ネットワークの空帯域幅一杯の輻輳ウィンドウサイズに到達するためには長い時間を必要とすることが知られている。

例えば [Flo03] によれば、将来やってくるであろう 10Gbps の帯域幅を持つ往復遅延が 100ms 程度の環境において、Reno の理論性能を一杯に引出そうとするとき、パケットの破棄のレートは 5,000,000,000 パケットに一度でなくてはならない。これは時間に直すと 1 と 2/3 時間に一度輻輳のイベントが発生することになる。これをパケットのロスレートに直すと、2×10^{-10} 程度である。これを実現するためには、2×10^{-14} 程度の回線のビットエラー率を必要とするが、このような回線の特性を持つ回線は現在のネットワークにおいては、非現実的である。

実際の回線ではもっと短かい頻度でパケットロスが発生し、また上で述べたよりも多くのビットエラーによるパケット破棄が発生するから、広帯域のネットワークにおいて Reno を用いた場合、スループットはずっと低いものとなる。

この問題を解決するため現在 HighSpeed TCP[Flo03][dSaDA03] が提案されている。HighSpeed TCP は輻輳ウィンドウの増加量が定数的であった Reno の問題点を見直したもので、輻輳ウィンドウの増加量及び減少量をその時々のウィンドウサイズに基いて行う方式で、現在の輻輳ウィンドウサイズがある定数値よりも大きい場合にはより大きく輻輳ウィンドウを拡大する。これらの増加量及び縮小量はいくつかの定数的なパラメタを経験則で定め、そこから導出を行なったものである。HighSpeed TCP は将来来たるべき超広帯域環境下においても十分な性能を発揮することができると期待されている。

しかしながら、HighSpeed TCP を実際のネットワークに導入するにあたっては問題点があることが知られている。HighSpeed TCP と Reno のフローを同一回線上に流した場合、Reno のフローは送出を抑えようとするのにも関わらず、HighSpeed TCP のフローは送出量を増加させようとする。結果としてルータにおけるキューのバッファは HighSpeed TCP のフローによって埋めつくされてしまい、実効スループットに異差が生じてしまうことが知られている。

[Flo03] によれば、10Gbps, 100[ms] 往復時間, 1 パケット 1500 バイトの環境下で、HighSpeed TCP と標準的な TCP を用いた場合、1:22.1 程度のスループット差が出ることが知られている。本問題は Reno から HighSpeed TCP へ移行を行う際に重大な問題となり得

る。本論文では Reno と HighSpeed TCP を併用した場合の、Reno の性能低下の問題を「HighSpeed TCP による Reno の性能劣化問題」(Reno Performance Degradations Problem by HighSpeed TCP) として定義する。

本論文ではこの問題を解決するため、Reno の代理アクセス機構として TCP を中継するプロキシ機構を用いる方法について検討を行い、プロキシ機構の実装及び評価を行う。

第2章　関連研究

2.1　広帯域・高遅延環境向けの輻輳制御方式

広帯域高遅延環境における Reno の性能上の欠点を改良するための、他の輻輳制御方式に関する多くの研究がある。本論文で議論する HighSpeed TCP は Scalable TCP[Kel03] の結果を反映し考案された方法である。これらの輻輳制御方式ではルータのキューを常にパケットで満たすよう送出量を増す方式である。またもう一つの輻輳制御方式の分類として、Vegas[aSWOaLLP94], FAST がある。これらの輻輳制御方式ではルータのキューには常に最小のパケットだけがたまるように送出量のコントロールを行う。後者の方式では公平性に大きな問題があることが知られている。後者の方式を一般のネットワークで用いると、Vegas や FAST などはパケットを最小にしようと試みるのに対して、Reno などは常にパケットで満たそうとするので、Vegas や FAST のスループットが大幅に低下してしまうという問題があり、この輻輳制御方式の実施にあたってはこの点が障壁となっている。

2.1.1　Reno の問題点

TCP は信頼性のある通信の他、輻輳制御機能を提供する。TCP における輻輳制御はインターネットにおいて、エンドホスト間が最も適切なスループットでデータの転送が行なえるようにし、インターネットが輻輳崩壊を起すことを防ぐ働きを持つ。現在インターネットで用いられている標準の輻輳制御方式は 4.3BSD Reno 実装に基いた実装であり、本論文ではこれを単に Reno と呼ぶ。

ここで、Reno の説明に入る前に、まずルータにおけるパケット破棄と TCP のパケットに対するシーケンス番号、そして、TCP の肯定応答 (acknowledgement, 以下単に ACK と呼ぶ) について触れておく。

ルータはパケットに対するスイッチングシステムである。ルータは複数の通信ポートを持っており、届いた IP パケットの宛先を元に最適なポートに転送する働きを持つ。あるポートに届いた IP パケットは宛先を参考に次に送るべきポートが選択され、ポートのキューに格納される。

キューに格納されたパケットは順次転送される。もしこのキューに空きがなければ[1]、IP パケットは破棄されてしまう。この破棄されてしまう状態は、つまり回線の帯域幅いっぱ

[1] ここでは DT, Dropped Tail 方式のルータの動作を説明している。パケットの破棄のやり方としては他に RED[aVJ93] があり、キューイングの方式としては FQ[aSKaSS89], WFQ, CBQ などがある。

いに回線が活用されていることを示している。Reno における制御ではこのパケットの損失が輻輳の制御のために用いられている。

TCP のパケット 1 つにはパケットの連続性及び欠落性を通信先に知らせるためシーケンス番号が付与されている。受信側はあるシーケンス番号のパケットが届く度[2]にこのシーケンス番号に相当する肯定応答パケットを送信側に返す。受信側が受信するこのシーケンス番号が連続でなかった場合は、パケットが損失していることを意味する。もしシーケンス番号が不連続であるパケットを受け取った場合は、重複した肯定応答を送信することで、パケットの損失を送信側に知らせる。

Reno の輻輳制御は以下の 2 つのフェーズを持つ。

- スロースタートフェーズ
- 輻輳回避フェーズ

スロースタートフェーズは連続したデータの転送における、最初のフェーズである。スロースタートフェーズはなるべく早くネットワークの空帯域幅いっぱいにスループットを引き上げる目的で設けられている。スロースタートの手順を以下に示す。

1. 1 パケットをまず送る
2. 送信先からこのパケットに対する肯定応答が返ってくる
3. 2 パケットを送る
4. 送信先からこれら 2 つの肯定応答が順番に送られてくる。
5. 2 つの肯定応答を全て受信したら、4 パケット送る
6. 以下繰り返し
7. パケットの損失が発生するようになったら輻輳回避モードに移行する

以上のように、一度パケットを送信すると次は 2 倍パケットを送る。パケットの増え方は、スロースタートという名前に反して指数的な増加を示す。スロースタートという名前のゆえんは、パケットを送り終るまで次のパケットを送らないという意味で命名されている。

なおここで説明しているスロースタートは従来的なスロースタートの方法であり、限定的スロースタート [Flo02] とは異なっていることに注意されたい。

また、1 往復時間 (1 Round Trip Time, 1RTT) でいくつのパケットを送るのか、ということを、ウィンドウサイズと呼んでいる。これは、肯定されることなしに送ることができるパケットのサイズを示している。

パケットの損失が初まるようになったら、輻輳回避フェーズに入る。スロースタートにおける初めの損失でウィンドウサイズは 1/2 に縮小される。このフェーズでは 1 つの肯定応答を受信する度に現在のウィンドウサイズ分の 1 づつウィンドウサイズを拡大する。ルータによって破棄されたパケットは再転送される。以後パケットが破棄されるとウィンドウサイズは 1/2 に縮小される。

[2]遅延 ACK を実装したホストではいくつかの ACK がまとめて送信側ホストに返される。

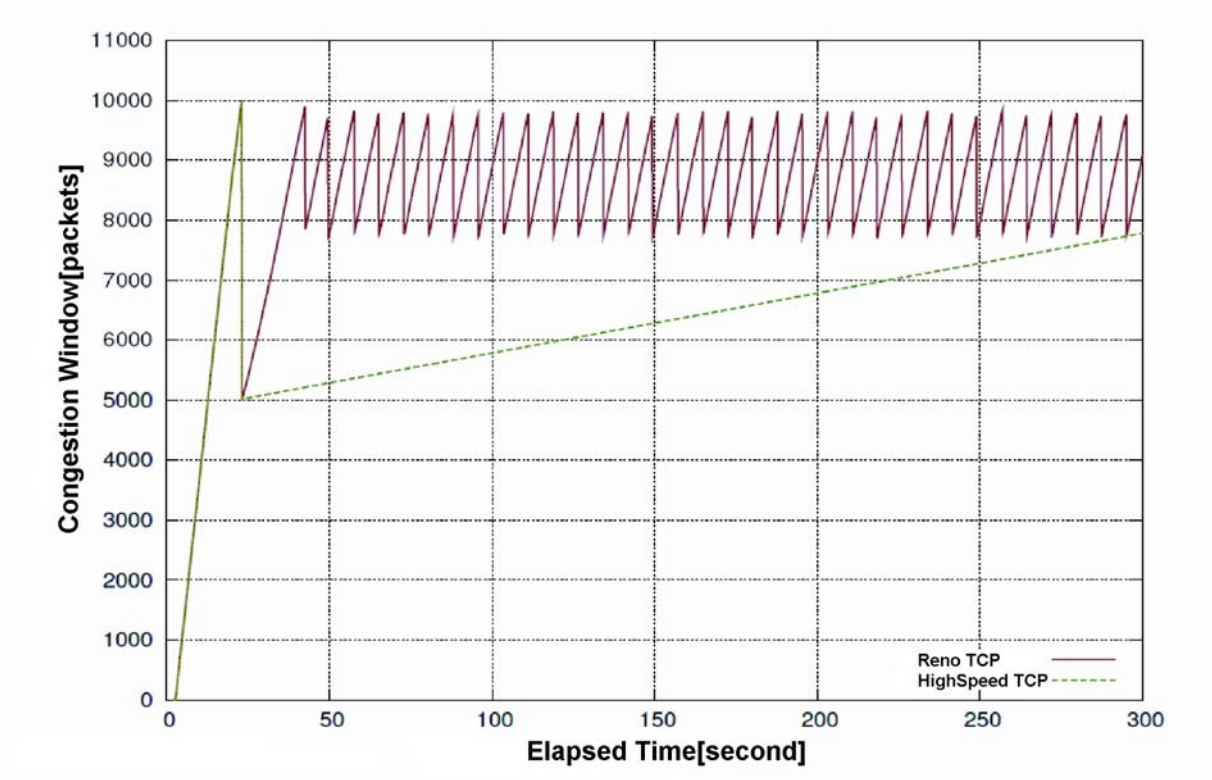

図 2.1: Reno 及び HighSpeed TCP の輻輳ウィンドウの変化

図 2.1 にあるネットワーク環境で Reno の挙動を調べるシミュレーションを行なった結果である。縦軸はウィンドウサイズを示している。このように、増減しながら、適切なスループットを保つようになっている。

Reno は回線の帯域幅によらず、輻輳ウィンドウの増加量は定数的である。従って回線の帯域幅が著しく大きい場合や、RTT の大きい環境下、またはその両方の環境下において、輻輳ウィンドウのサイズが回線の空帯域幅一杯になるまでに長い時間を要してしまうという問題がある。

2.1.2 Reno と比較した場合の HighSpeed TCP の相異点

Reno の輻輳回避フェーズにおける 1RTT 当りのウィンドウ増加量は、現在のウィンドウサイズを w とすると $1/w$ となり、またパケット損失時のウィンドウ減少量は $w/2$ となることを述べた。広帯域環境下においては、この増加量は小さすぎまた減少量が大きすぎるという問題が発生する。

[Flo03] によれば、1500 バイトのパケットサイズで 100ms の RTT 環境下において、10Gbps の恒常的なスループット得るためには、平均 83,333 セグメントの輻輳ウィンドウサイズを必要とする。この場合の輻輳発生のレートは、5,000,000,000 パケットに一度であるこ

とを意味する。これは、1 と 2/3 時間に相当する時間である。完全に回線を活用するために、平均のパケット破棄レートは 2×10^{-10} を必要とし、これを回線のビットエラーレートに直すと、2×10^{-14} となる。これは現在の回線の特性を考慮すると非現実的である。

以上のような問題を解決するため HighSpeed TCP[Flo03] は提案された。HighSpeed TCP では Reno のウィンドウ増減を以下のような数式に基いて行う。

$$
\begin{aligned}
w_{n+1} &:= w + \frac{2w^2 \cdot b(w) \cdot p(w)}{2 - b(w)} && (2.1)\\
w_{n+1} &:= w - \frac{\log(w) - \log(W_{low})}{\log(W_{high}) - \log(W_{low})} \\
&\quad \cdot (b_{high} - 0.5) - 0.5 && (2.2)\\
p(w) &= \exp(\frac{\log(w) - \log(W_{low})}{\log(W_{high}) - \log(W_{low})} \\
&\quad \cdot (\log(P_{high}) - \log(P_{low})) + \log(P_{low})) && (2.3)
\end{aligned}
$$

ここで、$W_{low} = 38$, $W_{high} = 83000$,$P_{high} = 10^{-7}$, $P_{low} = 15^{-3}$, $b_{high} = 0.1$ は RFC3649 で定義されている最適な値である。

以上のパラメタに基く、パケット損失率と送出量の関係は、図 2.2 のように表わすことができる。上式はこの図より導出したものであると考えられる。

直感的な意味としては、もし現在のウィンドウサイズが W_{low} よりも大きい場合は、より大きな平均パケット損失率を許容することで、より大きな輻輳ウィンドウサイズの増加を行うということである。図 2.1 に Reno 及び HighSpeed TCP の輻輳ウィンドウの遷移とその相違を示す。このグラフでは確認するのが困難であるが、HighSpeed TCP における輻輳ウィンドウの増加は対数的である。

2.1.3 TCP Vegas

Reno 及び HighSpeed TCP はパケットロスを利用し輻輳制御を行なっていた。TCP Vegas[aSWOaLLP94] では、本質的に異なったアプローチを取る。

パケットを送信するときにその時間を記録しておき、ACK パケットにより往復時間を測定する。ここの記録で得られた最小の RTT を $BaseRTT$ として定義する。

$Expected = \frac{WindowSize}{BaseRTT}$

また、Vegas は現在のスループットを $Actual$ として定義する。ここで 2 つの値 α と β を定義する。そして下記のような制御を行う方式である。

- $Diff = Expected - Actual$
- $\alpha < Diff < \beta$ ならば変化させない
- $Diff < \alpha$ ならば増す

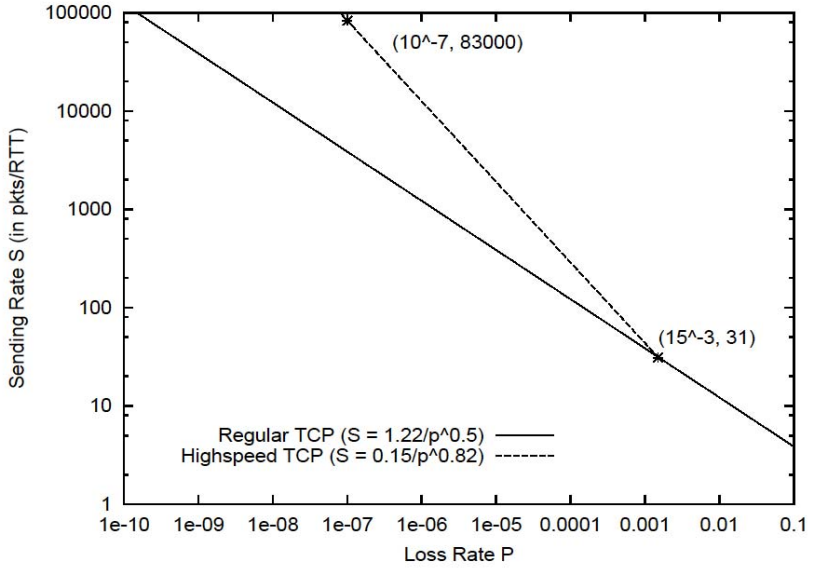

図 2.2: パケット損失率と送出量の関係

- $Diff > \beta$ ならば減らす

つまり、TCP Vegas はルータのキューにパケットが滞留することを、肯定応答の ACK の乱れによって感知することによって行う。

TCP Vegas はルータにおけるキュー滞在パケット数をアンダーフロー状態に保つためレイテンシの少ないという特徴を持つ。

2.2 TCP の公平性

2.2.1 TCP の公平性

TCP は同じ回線を使う競合するフローが存在する場合には、往復遅延時間やその輻輳制御方式、経路の相違による各経路のパケットロス率などの相違によって、スループットに差を生ずる。これを TCP の公平性 (fairness) と呼ぶ。

Reno 及び HighSpeed TCP を併用した場合に不公平性が発生することが知られている。[Flo03] によれば、回線帯域幅が 10Gbps、往復遅延速度が 100ms の環境下における相対的なフェアネスは HighSpeed TCP:Reno = 22:1 になることが知られている。

Reno または HighSpeed TCP と TCP Vegas を併用した場合には、TCP Vegas のスループットが大幅に低くなることが知られている。TCP Vegas はルータのキューに滞在するパケットの容量を常に小さく保とうとするのに対し Reno はルータのキューの空が無くなるまでパケットの送出量を増加し続けるからである。従って TCP Vegas は Reno と共存

することができず、TCP Vegas は実験用の回線でしか用いることができない。

輻輳制御方式の選択は大変困難である。主流派である Reno からの移行方法を考えることなしに、輻輳制御方式を単純に移行することはできない。また、もし実際のネットワークに導入した場合に比較的重い輻輳状態が発生する輻輳制御方式を選択した場合には、端末等のインタラクティヴなトラフィックに与える影響は大きく、回復不可能なネットワークの輻輳崩壊を招く可能性がある。

2.2.2 キューイングによる輻輳状態分離

TCP の公平性を保つためには、輻輳状態を分離するフェアキューイング [aSKaSS89] と呼ばれる方法を用いることで公平性を保つことができる。この方法はルータにおける各インタフェース間における輻輳状態を分離する場合に、そのインタフェース単位で用いられることがある。しかしながら、一般の回線において TCP のフローは何万ものフローが存在し得るから、そのコネクションの全てを識別しおのおののキューに挿入する処理をルータによって行うことは、現在のところ難しい。従ってルータにおける回線の分配は、パケットの損失や明示的輻輳通知 (ECN) によって自律的に行なわれている。

ルータにおいて各フローを識別し通信量を制御する方法は TCP/AQM と呼ばれている。理論的には優れた方式であるのにもかかわらず、今のところ普及していないのは、実際のルータにおいて AQM 制御を行うことは法外なコストが必要であるからである。

2.2.3 まとめ

本節では TCP の公平性について述べた。TCP の輻輳制御には考慮すべき様々な問題がある。新しい輻輳制御方式の導入にあたってはそれによりネットワークが崩壊することがないかをよく検討しなくてはならず、かつ高いスループットを得ることができなくてはならない。TCP Vegas といったルータのキュー容量をアンダーフロー状態に保つ輻輳制御方式では Reno との公平性に問題があり高いスループットを達成することができない。S. Floyd 女史の影響下であることと、併用時において問題の発生が最も少なく比較的移行しやすいことなどを考えると、今のところ最も有力なのが HighSpeed TCP である。

HighSpeed TCP 導入にあたって考慮しなければいけない問題の一つとして、公平性の問題がある。公平性の問題にはフロー毎に対するフェアキューイング等の手法を取ることも考えられるが、しかしながらそういったルータ今のところ実験レベルでしか存在しない。であるから、TCP は今迄通りルータの助けを借りることなしにエンドトゥエンドによって自律的な輻輳制御を行わなくてはならず、それによってインターネットの崩壊を防がなくてはいけないのが現状である。

しかしながら、広帯域時代に向けて輻輳制御方式移行は必須であり、輻輳制御方式を Reno のまま保ち続けることは困難である。従って本論文では、公平性の問題を解決するため、本論文では性能改善のためのプロキシ機構を用いる方法について検討を行う。

次の節では性能改善のためのプロキシ技術について述べる。

2.3 既存の性能改善のためのプロキシ機構

TCP の性能劣化はモバイルやサテライト環境などの特別な環境において、パケット損失が空帯域幅によらず発生する場合や、また衛星回線を経由するなど遅延が大きい場合において発生する。この問題を解決するため、TCP の性能劣化を代理アクセス機構によって防ぐことはかねてより行なわれてきた。これらの研究は性能改善のためのプロキシ [aMKaJBaGMaZS01](Performance Enhancing Proxies, PEP) としてまとめられている。

性能改善のための TCP プロキシ機構を大別すると 2 つに分けることができる。一つ目が TCP Spoofing と呼ばれる手法で、これは TCP の輻輳状態の感知が重複した肯定応答によって行なわれることを利用し、重複 ACK の通知を遅らせる方式である。二つ目は TCP Splitting または Indirect 方式と呼ばれる方式であり、尚早な ACK を送信することによってパケットの蓄積を行い、また再送信を行うものである。

[aMKaJBaGMaZS01] によれば、モバイル向けやサテライト向けの性能改善のための TCP プロキシ機構はこれまでにも多く研究されてきたが、これまでのところ広帯域・高遅延環境向けの性能拡張のための TCP プロキシ機構に関する十分ではない。従って本論文では、高帯域・高遅延環境向けの TCP Splitting 方式による、Reno から HighSpeed TCP にフローの変換を行う性能改善のためのプロキシ機構に関する研究を行う。

2.4 まとめ

以上の趨勢を考慮すると、広帯域・高遅延環境においては、HighSpeed TCP の導入が有力である。。しかし、HighSpeed TCP の利用に当っては不公平性の問題がある。であるから、この問題を解決するために HighSeeed TCP に変換を行う性能改善のためのプロキシの研究は重要であると考えられる。

第3章　提案する装置

3.1　装置の概要

本装置はGigabitEthernetのポートを2つ持っている計算機である。

本装置に搭載されているプログラムはGigabit Ethernetインタフェースからのパケットの処理, フローの識別, 早期肯定応答(premature ACK)の転送, パケットの蓄積, 異なる輻輳制御方式での再転送などの機能から成り立っている。

3.2　本装置におけるフロー毎のパケットの流れ

本論文で提案するPerformance Enhancing Proxiesにおけるフロー毎に見たパケットの流れを3.1に示す。

3.3　実装方法と本実装における動作

実装のため、バークレイソケットインタフェースを用い、パケット蓄積のためのバッファメモリーはユーザ空間に確保した。select()システムコールでブロッキングを行い、入出力があった場合には他方のソケットへの転送を行う。

通信インタフェースにパケットが届くと割込みが発生する。この割込みは遅延して行なわれる場合もある。割込みが発生すると、インタフェースからパケットを取り出し、またパケットの識別を行い、コネクション毎の受信側ソケットバッファに格納する。

プログラムはカーネルからタイムスライスを渡された場合には下記の動作を行う。

1. もし入力側ソケットにパケットがあれば、それをユーザ空間に確保したバッファに複写する。
2. もし出力側ソケットに余裕があれば、出力側ソケットにバッファを複写し、ユーザ空間にあるパケットを削除する。
3. もし入力側ソケットにも出力側ソケットにも余裕がなければブロッキングを続ける。

出力側ソケットに書き込まれたパケットは、カーネル内のスケジューラによってHigh-Speed TCPの方式による輻輳制御が行なわれ、インタフェースから送出される。

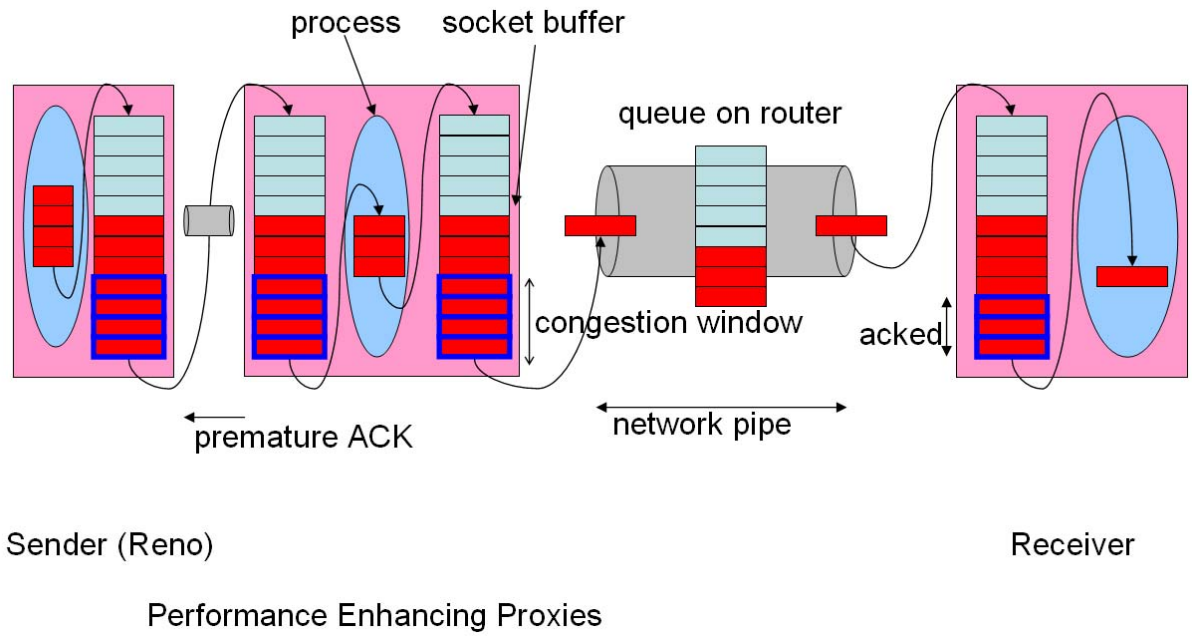

図 3.1: 本論文で提案する Performance Enhancing Proxies

第4章　提案装置の評価

4.1　実験方法

本実験で用いた実験環境を述べる。

実験のための、広帯域・高遅延環境として SuperSINET のナノテク VPN を用いた。ナノテク VPN は 1Gbps の専用線で東北大学と結ばれている。

LSW-GT-8C に接続された別の計算機から ping コマンドを利用し 1 秒の間隔で 10 回の往復遅延時間の測定を行なった。この結果では、最小往復遅延時間は 28.422[ms] であり、平均往復遅延時間は 28.508[ms] であり、最大往復遅延時間は 28.612[ms] であった。

本実験では 4 台の計算機を使用する。4 台の計算機及びそのオペレーティングシステムの構成と設定を表 4.1 に示す。図には Linux の sysctl の出力結果をそのまま掲載した。なぜならば、Linux においては各種パラメタに基いてウィンドウサイズが調整されてしまうため、これを切り捨ててしまうと実験の再現性に問題があると考えたからである。

計算機 HSTCP2 は HighSpeed TCP によって 10 本のフローを流すための計算機である。

計算機 HSTCP1 は提案の装置であり、Reno の通信の中継を行うための計算機である。

HSTCP1, HSTCP2 は本大学内に設置されている。

計算機 nano-web は東北大学内に設置された計算機で、HSTCP1, HSTCP2 からのデータの送信を受信する計算機である。

スループットの測定は netperf[Jon] を利用し行なった。netperf はネットワークを通じて TCP 及び UDP 利用におけるスループットとパケット損失率を調査するプログラムである。本測定では TCP を用いて 600 秒の通信を行い、スループットを算出した。

1 コネクション当りのプログラムプロセス中のバッファ容量は 16,777,216 バイトである。

4.2　不公平性の実験

1 つ目の実験は 10 本の HighSpeed TCP のフローが並行して流れる環境において Reno フロー 1 本及び HighSpeed TCP のフロー 1 本のスループットを比較し、不公平性を確認する実験である。(図 4.1)

1 つめの実験による Opteron(Reno) によるスループットは 70.7Mbps であった。これを HighSpeed TCP に変更すると 83.8Mbps のスループットが得られた。

HSTCP1, HSTCP2
SuperMicro SuperServer 5013G-i Linux-2.4.20+altAIMD0.3+web100-2.3.3+e1000-5.2.52 sack/dsack = off txqueuelen:1000 net.ipv4.tcp_mem = 8388608 8388608 8388608 net.ipv4.tcp_wmem = 8388608 8388608 8388608 net.core.wmem_default = 8388608 net.core.wmem_max = 8388608 HSTCP1 net.ipv4.tcp_rmem = 2097152 2097152 4194304 net.core.rmem_default = 524288 net.core.rmem_max = 4194304 HSTCP2 net.ipv4.tcp_rmem = 4096 87380 174760 net.core.rmem_default = 65535 net.core.rmem_max = 131071

Opteron
Gigabyte GA-7A8DW, PC3200(ECC Registerd), Opteron246 x 2 Linux 2.4.20+altAIMD0.3+web100-2.3.3 txqueuelen:1000 sack/dsack=off net.ipv4.tcp_wmem = 8388608 8388608 8388608 net.ipv4.tcp_mem = 8388608 8388608 8388608 net.core.wmem_default = 8388608 net.core.wmem_max = 8388608 net.ipv4.tcp_altAIMD = 0

nano-web
SuperMicro SuperServer 5013G-i Linux-2.4.22 sack/dsack = off net.ipv4.tcp_rmem = 8388608 8388608 8388608 net.ipv4.tcp_mem = 2097152 2097152 4194304 net.core.rmem_default = 8388608 net.core.rmem_max = 4194304

表 4.1: 実験に利用した計算機

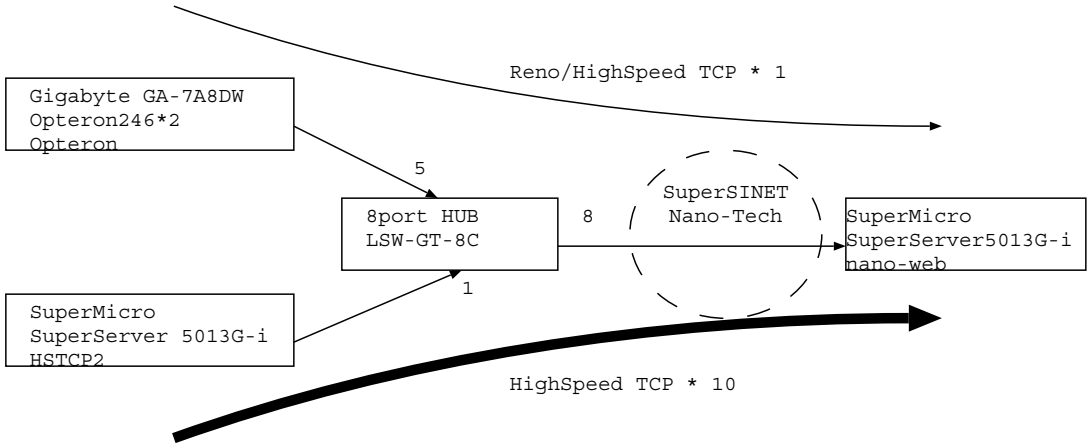

図 4.1: HighSpeed TCP 影響下の Reno のスループットを調べる実験

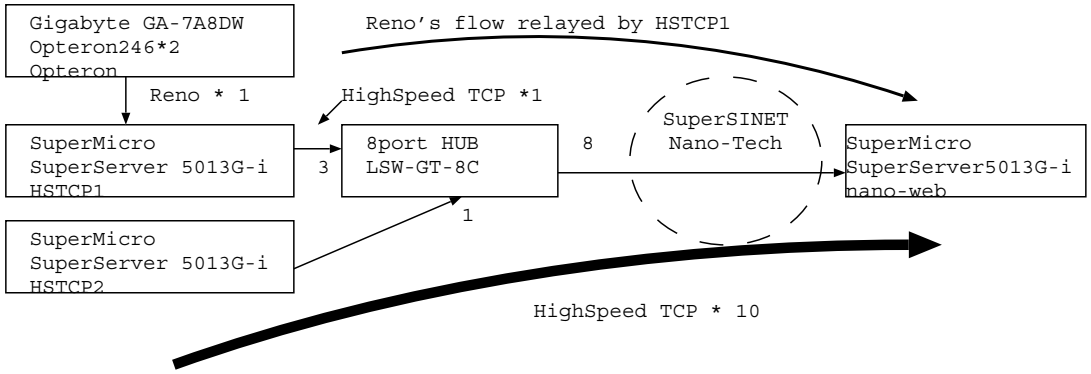

図 4.2: HighSpeed TCP 影響下における改善済 Reno フローのスループットを調べる実験

4.3 不公平性改善の実験

2 つ目の実験は、10 本の HighSpeed TCP のフローが並行して流れる環境において、提案装置を介した Reno フローのスループットを確認する実験である。(図 4.2)

2 つめの実験において、Opteron の Reno によるフローを HSTCP2 によって中継を行なったところ 93.0Mbps のスループットが得られた。

4.4 他にフローがない状態におけるスループットの実験

3 つ目の実験は他にフローのない状態で HSTCP1 を介した Reno の通信のスループットを計測することである。

他にフローがない状態では、67.36Mbps のスループットが得られた。

4.5 考察

本装置により不公平性が解決されているのが確認できた。しかし、本装置による改善の度合は予定していたより小さい。

原因としては、1つ目に変換のために用いた計算機の性能が十分でないためである。2つ目に、SuperSINET のナノテク VPN の遅延と帯域幅といった性能から判断すると、評価に用いた環境では不公平性が出にくいことが挙げられる。今回は 10 本の HighSpeed TCP のコネクションによって評価を行なったが、1 本の HighSpeed TCP 及び 1 本の Reno を併用した場合の Reno の帯域幅は 68.57Mbps であり、これを Reno から HighSpeed TCP に変更したところ、204.15Mbps の結果が得られた。したがって、もし本計算機が十分な性能を持っていたならば、本実験よりもうまく解決することができるはずである。

計算機の中では解決方法 2 回のメモリーコピーが発生しており、これが 1 度で済むように改変すれば性能が向上する可能性がある。

また、より高い性能を得るためには、計算機ベースでは困難であり、専用ハードウェアを用いなければならないと推測される。

また、他に通信の妨げとなる HighSpeed TCP がない状態においても 67.4Mbps しか出ないという問題が発生している。本現象についての調査はまだである。

第5章　改良の試み

5.1　はじめに

TCP とはデータ指向の通信であって、リアルタイム性を考慮して作成された通信プロトコルではない。であるから、レイテンシを議論することは徒労であるとの考え方が支配的である。

少なくとも多くの文献では、スループットを中心とし議論されている。しかし、今日 TCP はあらゆる通信に用いられ、本来使われるべきでない環境でも用いられている。例えば、並列分散計算システムやその他の可視化のためのシステムや、ストリーミング配信等である。また、TCP over TCP などの利用を考えた場合は、バースト転送時のレイテンシが大きければ通信のスループットは大幅に小さくなるという問題が発生する。

以上のことを考慮すると、レイテンシの削減は常に考慮すべき問題である。従って本章ではレイテンシの問題について検討を行なった。

またレイテンシの削減のため明示的輻輳通知をアプリケーションが行なえるようにし、スループットを抑制することでソケットバッファに滞留するパケット数をアンダーフロー状態に保つことができることを期待し実装を行なった。残念ながら今迄のところこれはうまく動作していないが、失敗の原因に関する考察を行い、結果として公表を行うものである。

5.2　TCP Splitting 方式における中継の概要

本節では 2 つのソケットを利用した中継システム (以降単に中継器と呼ぶ) の性能について議論する。

中継器をはさんだネットワークにおいて、輻輳制御がどのように行なわれるか考慮したい。図 3.1 は中継器をはさんだネットワークにおける概念図を表わしれている。左側が送信者、右側が受信者、中央が中継器である。中継器は 2 つのウィンドウ, それぞれ受信ウィンドウと輻輳ウィンドウを持つ。以後この図を使って解説する。

まず送信者が send() や write() といった API を使ってソケットバッファに対し書き込まれる。すると、システムが最適なタイミングでパケットの送信を行う。送出されたパケットは経路を通過し中継器の受信ウィンドウに蓄えられる。中継器のプロセスが select() のブロッキング状態からアクティブになると、受信ウィンドウに蓄えられたパケットがプロセス内のメモリー領域にコピーされ、次に、ソケットバッファに書き込みが行なわれる。

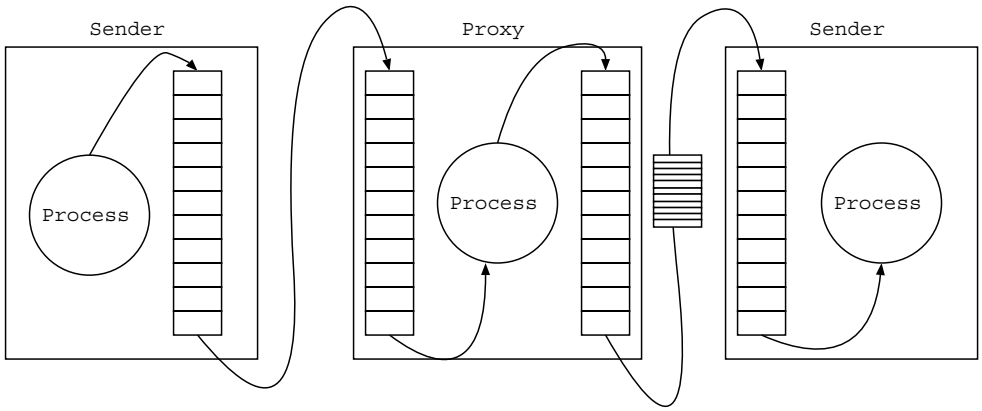

図 5.1: Performance Enhancing Proxies

そして中継器とエンドとの間の輻輳制御方式でパケットの送信が行なわれる。

もし送信者–中継器の間の帯域幅が中継器–受信者よりも大きかった場合はどうなるか? つまり、中継器から受信者の間にボトルネックが存在した場合には、中継器の受信者側ソケットバッファからなかなか送出されないので、中継器のプロセスはソケットに書き込むことができなくなり、ブロッキングされるか、書き込んだ場合でもエラーが返されるようになる。中継器のプロセスは書き込み可能になるまで待たなくてはならない。書きこみが可能になるまでに待っている間に中継器の送信者側ソケットバッファにパケットが溜っていく。ここの容量がなくなった時点で受信ウィンドウの空きサイズが 0 である通知が送信者になされ、送信者は送信を停止する。

本システム、つまり TCP Splitting を使った方式の中継器における問題点は次の通りである。パケットがさまざまな機構を通過するなかで各装置のバッファに蓄えられることになるのだが、そのバッファの総容量が大きくなりすぎ、レイテンシが低下してしまうという問題である。

5.3 中継時のレイテンシ

5.3.1 各実装方式における通信回線特性が通信に与える影響を調べる実験の実験結果

本実験の目的

本小節では、Reno 及び HighSpeed TCP のそれぞれの実装において、通信回線の特性がどう通信に影響を与えるかについて実験を行ったので、その結果について報告する。

本実験は Dummynet[Riz97] を使って、特定のネットワーク環境を作り出すことで、Reno 及び HighSpeed TCP 実装における TCP の通信とそれに関係する変数がどのように変化するのかについての検討を行うことである。また、Dummynet の性能限界がどの程度なのかを知るために行う。

実験方法

送信側ホストとして用いた計算機を下記に示す。

- Supermicro Superserver 5013G-I
- Linux 2.4.20 + AltAIMD+Web100 patch
- txqueuelen 100
- $net.core.wmem\_default = 2097152$
- SACK/DSACK off

模擬ネットワーク再現用のため用いた計算機を下記に示す。

- Supermicro Superserver 5013G-I
- FreeBSD 5.3 Release
- Dummynet(debug off), bridge, polling mode

受信側ホストとして

- Pentium IV, FSB800MHz, on-board gigabit ethernet
- txqueuelen 1000
- $net.core.rmem\_default = 2097152$
- Linux 2.6.5

スループットを 5.1 に示す。

片方向遅延	実装方式＼設定帯域幅	25Mbps	50Mbps	100Mbps	200Mbps	300Mbps
5ms	Reno	22.4	43.9	79.6	134.8	171.2
	HighSpeed TCP	22.3	43.2	67.6	112.5	163.1
10ms	Reno	22.1	40.0	62.8	100.6	109.5
	HighSpeed TCP	22.0	39.8	54.7	102.5	162.6
20ms	Reno	21.8	34.6	51.3	56.3	71.6
	HighSpeed TCP	20.4	33.3	44.7	70.0	89.6
40ms	Reno	19.8	25.6	31.7	38.4	38.8
	HighSpeed TCP	20.2	25.7	29.8	56.9	97.6

表 5.1: Reno 及び HighSpeed TCP におけるスループット特性

遅延＼帯域幅	25Mbps	50Mbps	100Mbps	200Mbps	300Mbps
5ms	22.5	43.0	74.3	118.8	139.3
10ms	22.4	41.0	57.3	112.8	117.9
20ms	20.5	30.7	48.0	70.5	114.5
40ms	16.0	25.9	35.6	46.1	47.7

表 5.2: 非管理バッファによる TCP リレー特性 (スループット [Mbps])

考察

Dummynet の実験に用いる計算機の性能として、Dummynet を有効にしない場合では 600Mbps 程度, Dummynet を有効にした場合では 400Mbps 程度であることを netperf によって確認した。

5.3.2 バッファの制御を行わない中継機構の実験

先程の実験のホストのもう一方のインタフェースに Sun Blade 1500 を接続し、TCP リレー機構を動作させた場合に得られるパフォーマンスを表 5.2 に示す。

この結果を、表 5.1 と比較すると、例えば、200Mbps, 40ms 程度の環境である場合に 120[%] 程度スループットが向上していることが判明した。

5.4 ソケットバッファサイズが通信レイテンシに与える影響

本小節では、ソケットバッファサイズがどのように通信に影響を与えるのかについての基本的な実験を行ったので、その結果を示す。

遅延＼帯域幅	25Mbps	50Mbps	100Mbps	200Mbps	300Mbps
5ms	2009	1113	612	372	313
10ms	2043	1147	827	396	399
20ms	2228	1482	1018	689	423
40ms	2851	1753	1352	1029	1002

表 5.3: 非管理バッファによる TCP リレー特性 (レイテンシ [ms])

項目名	送信側	受信側
計算機名	Fujitsu GP400S(Ultra5)	SuperMicro SuperServer 5013G-i
CPU	UltraSPARC 400MHz	PentiumIV 2.8GHz FSB800MHz
インタフェース	FastEthernet	GigabitEthernet
オペレーティングシステム	Linux 2.6.4	Linux2.4.20+altAIMD0.3+web100-2.3.3

表 5.4: ソケットバッファサイズが通信に与える実験に使った計算機

接続に用いた計算機を表 5.4 に示す。

本計算機同士は GigabitEthernet スイッチングハブ (BUFFALO LSW-GT-8C) を介して接続されている。

1GBytes のトラフィックを発生させそのスループットを計測する。また、1MBytes ごとにタイムスタンプを挿入しエコーバック速度の計測を行う。受信者側のソケットバッファサイズを、sysctl 値である “net.core.rmem.default” を変更することによって変化させ、これらがどう変化するかの測定を行う。

測定結果を表 5.5 及び図 5.2 に示す。

本実験より次のことが分った。ソケットバッファサイズを増加させることにより、レイテンシが増加する。また本実験結果より次のことが予想される。RTT が著しく小さい環境、例えば LAN 内における転送実験においては、ソケットバッファサイズを小さくしてもほとんどスループットに影響しないが高遅延の環境下においてはこれを小さくするとスループットが低下してしまうという問題が発生する。

この実験結果より、ソケットバッファサイズがレイテンシに悪影響を及ぼしているということがわかった。このレイテンシによる性能低下を回避するためには、適切なバッファサイズを選択する必要がある。しかし適切なバッファサイズは回線の帯域幅と負荷に依存するため、これを静的に行うことは困難である。

buff. size[bytes]	throughput[Mbps]	latency[ms]		
		avg.	max.	min.
2k	87.60	3.114	7.352	2.106
4k	87.63	3.137	3.222	0.566
8k	89.75	3.454	3.629	1.854
16k	89.72	3.905	4.076	1.982
32k	89.73	4.649	9.698	1.860
64k	89.73	6.119	6.229	1.853
128k	89.74	9.074	9.210	1.858
256k	89.74	14.97	15.17	1.988
512k	89.75	26.79	34.98	1.987
1024k	89.78	50.41	59.86	1.863
2048k	89.82	97.52	97.70	1.863

表 5.5: ソケットバッファサイズが通信に与える影響

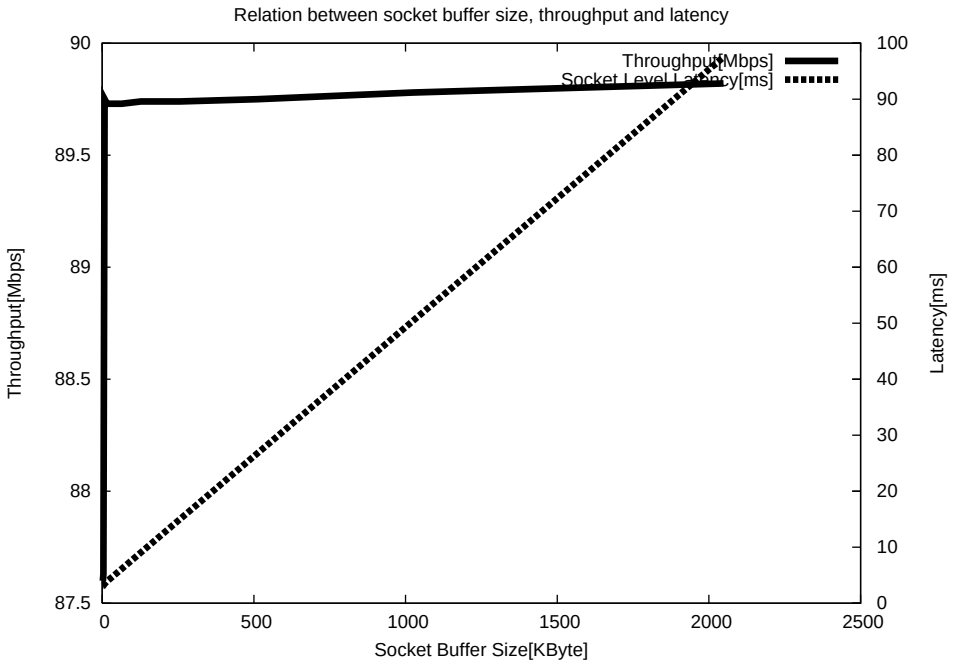

図 5.2: ソケットバッファサイズが通信に与える影響 (グラフ)

5.5 明示的輻輳通知機構の拡張

本研究では、ルータだけではなくホストも輻輳を通知することができるよう、カーネル API の拡張を行なった。

新たに実装したカーネル API は以下のものである。

ioctl(s, SIOECN, NULL);

ここで、s は現在の通信を行なっているソケットであり、SIOECN はシステムの持つ ioctls.h で定義されている定数である。なお NULL はヌル値を示している。

本カーネル API の実行を行うことで、ホストは経路の途中で輻輳の通知を受けとったのと同じように動作する。これを実行することで、送信側のパケットの送信を抑制することができるようになる。

実装は Linux-2.4.20 に altAIMD パッチのヴァージョン 0.3 及び web100 パッチの 2.3.3 を適用したものに対し行なった。機構が有効に働くことを実験によって示し、適切に輻輳を通知することによってレイテンシを削減することができることを示す。

本実験の目的は、前小節で挙げたカーネル API をどの程度の間隔で挿入すると最も効率良くレイテンシを削減することができるのかという問題について検討を行うことである。

本実験では、1024 バイトのパケットの読み出しを何回行うごとに前小節で述べたカーネル API を呼びだすと効率が最大になるのか実験を行う。最終的に 1GBytes のデータの送信を行なった場合のスループット及びレイテンシで評価を行う。

実験に用いた機器はソケットバッファサイズが通信に与える影響に関する実験で利用したものと同一のものである。

SuperMicro の `net.core.rmem_default` と `wmem_default` が 512k[bytes]、Ultra5 の `net.core.rmem_defaultnet.core.wmem_default` が 126976[bytes] である。

本実験の結果を表 5.6 及び図 5.3 に示す。

ECE 信号挿入の間隔を 128〜256 を send コール毎に一度実行することで、スループットを落すことなくレイテンシを削減できることが判明した。

本実験により、明示的な輻輳通知機構によってレイテンシが削減され、また適切な明示的輻輳通知を行うことによりスループットも減少しないことが判明した。

5.6 明示的輻輳通知 API による通信レイテンシの改善アルゴリズムの提案

これらの実験結果より、次節では明示的輻輳通知 API を用いた通信レイテンシの改善アルゴリズムの提案を行う。

パケットが滞留可能なパケット数を求めるための関数を以下のように定義する。

$f(TCPvars, Impl_{sender})$

ここで $TCPvars$ は刻々と変化する TCP コントロールブロックの変数であり、$Impl_{recv}$ は送信ホストの実装方式、例えば HighSpeed TCP などである。

count	throughput[Mbps]	latency[ms]		
		avg.	max.	min.
8	0.562	12.441	76.30	0.456
16	68.494	0.956	1.260	0.442
32	78.349	1.077	7.551	0.450
64	89.158	1.208	1.414	0.477
128	89.718	1.683	1.926	0.499
256	89.726	2.202	2.458	0.490
512	89.733	3.077	9.472	0.382
1024	89.740	4.789	4.998	0.436
2048	89.731	5.630	8.004	0.483
4096	88.627	4.888	5.838	0.495
8192	88.137	4.500	5.986	0.482
16k	87.661	4.330	5.859	0.496
32k	87.879	4.328	5.863	0.494
64k	87.668	4.207	5.869	0.477
128k	87.672	4.197	9.402	0.494
256k	87.617	4.178	8.685	0.495
512k	87.631	4.157	7.228	0.484
1024k	87.620	4.171	9.807	0.493

表 5.6: 明示的輻輳通知が通信に及ぼす影響

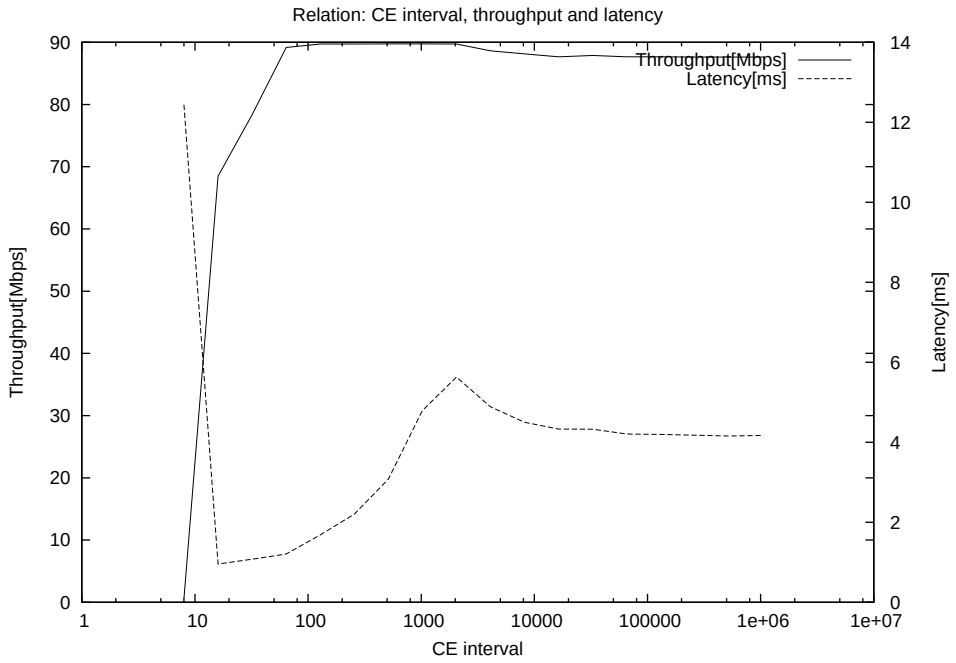

図 5.3: 明示的輻輳通知が通信に及ぼす影響

適切な送信バッファは輻輳ウィンドウの 2 倍から 3 倍程度と言われているが、これはシステム特性毎によって異なってくる。本論文では本関数のパラメタを既知であるものとして扱う。具体的には、ある特定のネットワーク特性においては、$TCPvars$ はいつも同じように振舞うものと仮定し、ある特定のネットワークではある特定のバッファ滞留許可容量を設定するものとする。

実験に用いた一つ目の方法を次に示す。本方式はプロキシ内のソケットバッファに滞留するパケットの容量が滞留許可容量を越えた場合に、データを送信しているホストに対して ECE 信号を送信し、スループットを抑えることでソケットバッファに滞留するパケット数を削減する方式である。

```
until connection end
  if f() < (number of in-socket packets) then send ECE
end until
```

実験に用いた二つ目の方法を次に示す。本方式はバッファの残り容量に比例した確率でランダムに ECE を送る方式である。

```
until conneciton end
  if f()-red < (number of in-socket packets) then
    if m_prob(f-(number of in-socket packets)/red)==true then send
```

```
  ECE
end until
```

ソケットバッファに滞留するパケットを調節する方法として、相手に通知する受信ウィンドウのサイズを実際のサイズよりも小さくすることによってパケットの滞留するパケットの数を減らすという方法も考えられる。しかしながら、この方法では、もし長い間サイズ 0 の通知が行なわれてしまった場合はスロースタートフェーズに入ってしまい、急激なスループットの低下が発生すると考えられる。したがってより緩やかに調節を行うという目的のためには ECE による削減方法のほうが望ましいと考えられる。

5.7 評価方法

単なる中継と、本機構によって性能がどうかわってくるかによって評価を行う。Dummynet によって 300Mbps, 40ms の環境をエミュレートし 1024MBytes の連続したデータ転送を行うことによって特性の評価を行う。

5.7.1 評価項目

スループット及び平均遅延、及び中継器におけるパケット滞留容量によって評価を行う。

5.7.2 評価環境

評価に用いた環境を図 5.4 に示す。

- 送信側: SunBlade 1500
- 中継器: SuperMicro SuperServer 5013G-i, Linux-2.4.20 with AltAIMD patch, $wmem_default = 8192 \times 1024$[byte]
- ネットワークエミュレータ: SuperMicro SuperServer 5013G-i, FreeBSD 5.3
- 受信側: Pentium IV 搭載マシン, FSB800MHz

5.8 計測結果

結果を図 5.5 及び図 5.6 に示す。

計測の結果本機構を採用しない場合のスループットは 51.52Mbps であるのに対し本機構を採用した場合では 33.85Mbps 程度に低下した。またそれに伴いレイテンシも大幅に低下した。

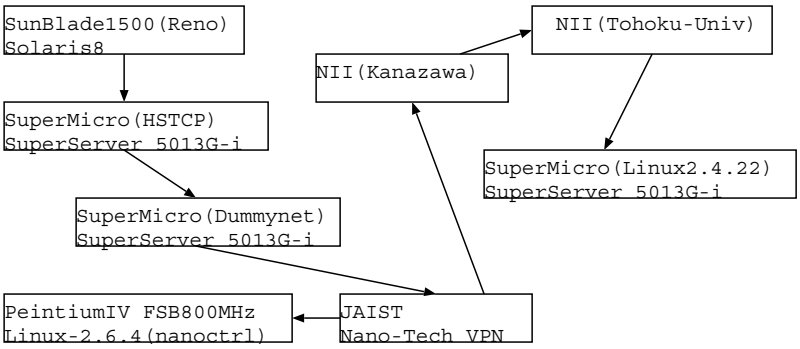

図 5.4: 実験用ネットワーク環境

5.8.1 性能の低下の原因

バッファに滞留する容量を制限したところスループットが低下した。図 5.6 より、バッファに滞留するパケットの数が 0 となる時間が発生し、その度に送出側がスロースタートモードに入ることでスループットが低下していることを確認した。

5.8.2 RED ライクな ECN 制御方法による実験

ECN を使い容量を越えた場合に ECE 信号を送る方法では急激に送出量が下ってしまい、スロースタートモードに入ってしまうことで性能が下ってしまうという問題があった。この問題を解決するため、あるしきい値を越えた場合はバッファの残り容量に応じて制限を加える方式を使って急激な送出量の降下を防ぐ実験を行う。このアイディアはルータのパケット破棄方式である RED–Random Early Detection から借りたものである。ルータにおける動作では破棄を行うが本制御においては破棄ではなく ECE 信号による送出量の抑制を行う。

300Mbit/s, 40ms の環境下において 512MBytes のデータ転送を行い、TCP スタックの挙動を確認する。先の実験より Dummynet のキューサイズは 48KBytes とした。

制御を行わない場合の滞留パケット容量の変化を図 5.7 に示す。変化を行なわない場合におけるスループットは 28.9Mbps, レイテンシは 2262.0[ms] である。また同じ条件で再度測定を行なったところ、30.64Mbps, レイテンシは 2120.5[ms] であった。

制御パラメタは多くのもので実験を行なったが、比較的うまくパラメタを選択することができていると思われるものを選んで図 5.9 から図 5.12 に掲載する。全体の特性については表 5.7 と表 5.8 に示す。

本実験の結果、急激な TCP 送出量の減少が抑えられスロースタートに入る回数は少なくなり、固定的な制限方法と比較するとスループットは向上した。例えば、バッファの容量を制限しない測定結果と、バッファの容量を 8192KB/4096KB の RED で制限を行なったものとの比較では、転送時間は 20 秒ほど長くなったがレイテンシは 2120.00[ms] から

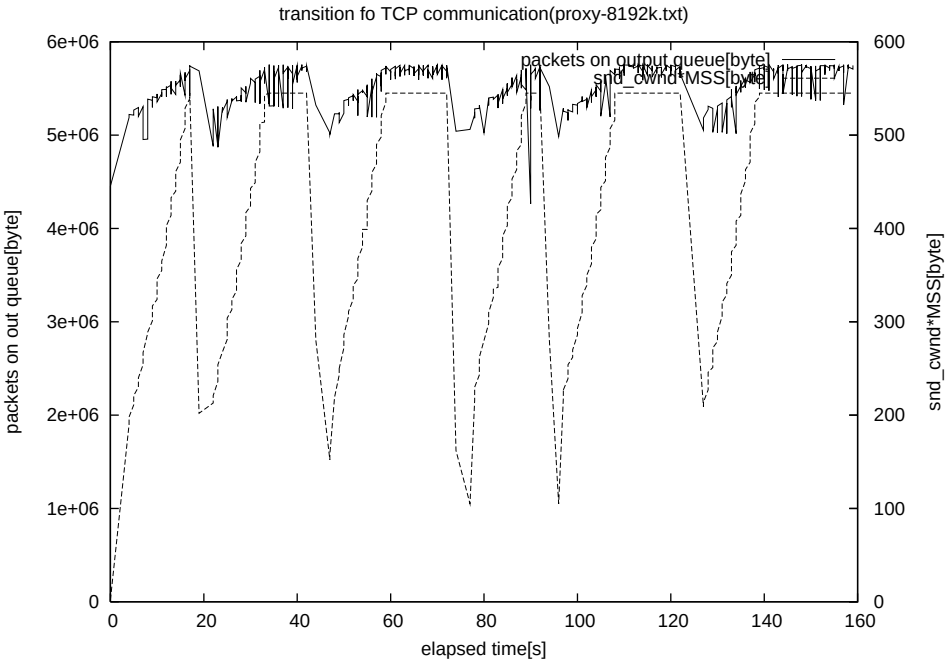

図 5.5: ECN による制御のない場合

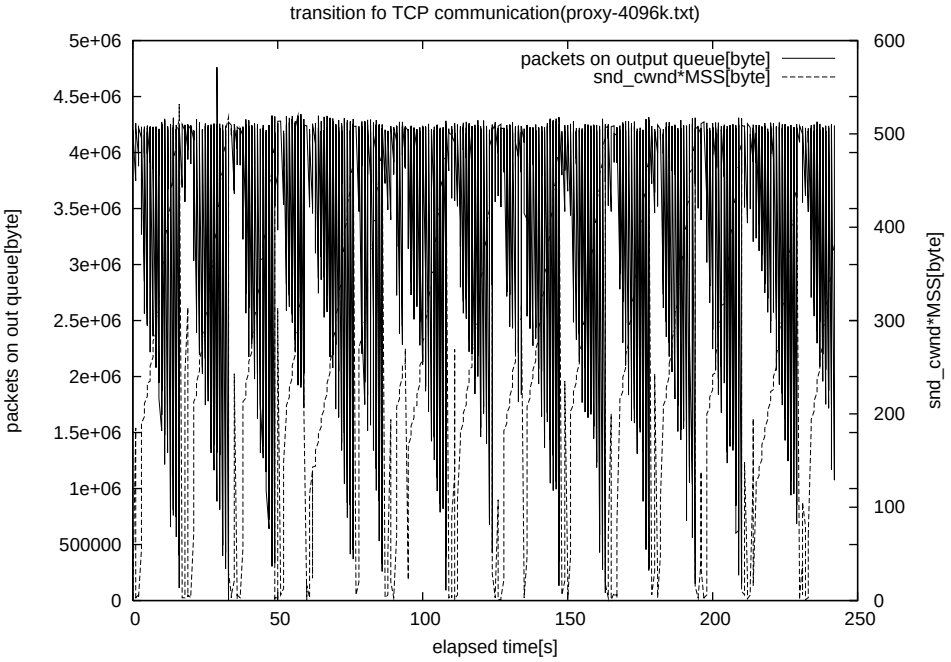

図 5.6: ECN によって 4096KByte に滞留パケットを制限した場合

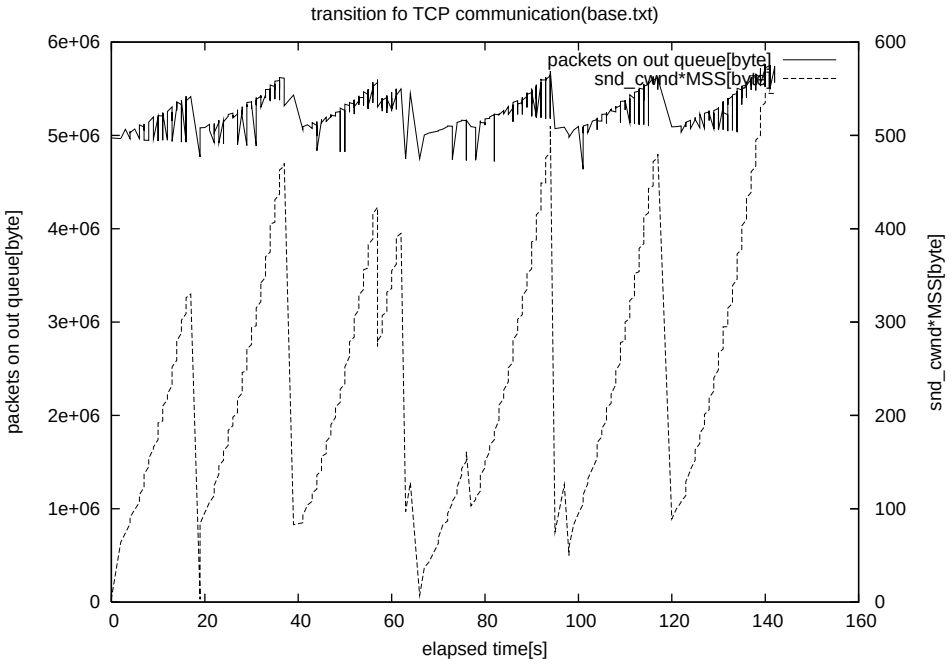

図 5.7: バッファの容量を制限しない場合

Absolute[KB] ╲ RED[KB]	64	128	256	512	1024
8192	30.7	26.4	22.3	33.7	39.7
7680	36.5	29.0	30.6	25.9	30.7
7168	26.7	24.5	29.5	30.3	30.7
6656	35.8	27.5	28.4	25.0	26.2
6144	27.0	26.5	27.3	30.9	24.2
5632	30.7	30.7	30.7	30.7	30.7

Absolute[KB] ╲ RED[KB]	2048	3072	4096	5120	6144
8192	34.5	25.9	23.2	20.1	16.1
7680					
7168		19.7	17.6		
6656					
6144					
5632					

表 5.7: RED 方式におけるスループット特性 [Mbps]

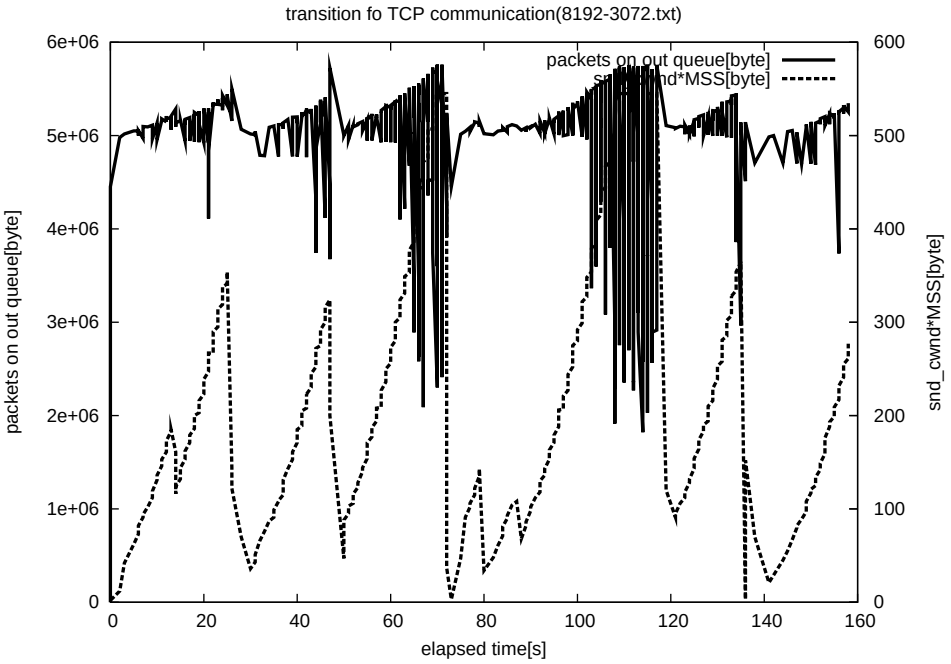

図 5.8: RED による制御 (8192KByte/3072KByte)

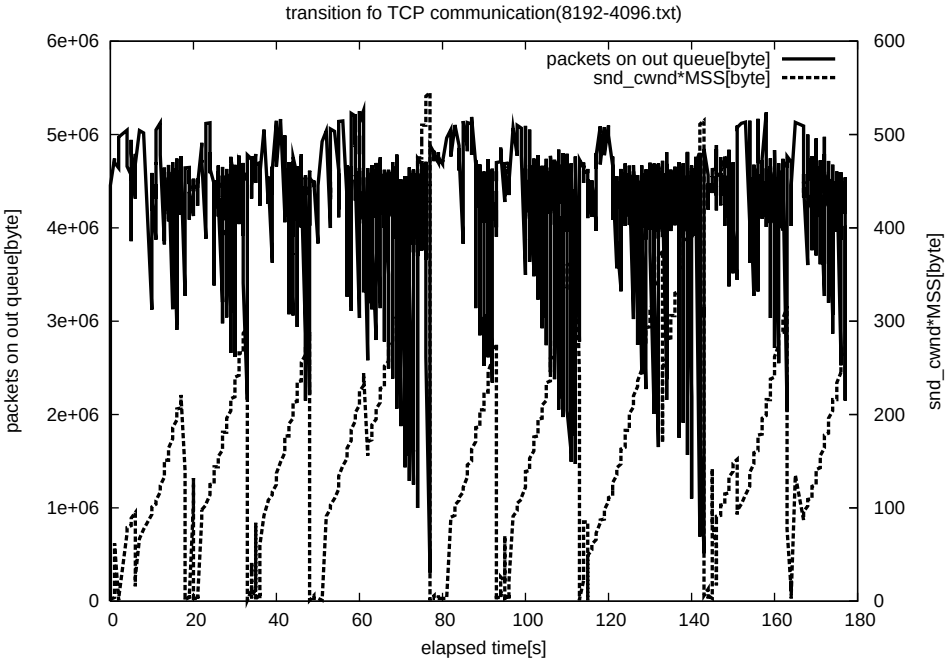

図 5.9: RED による制御 (8192KByte/4096KByte)

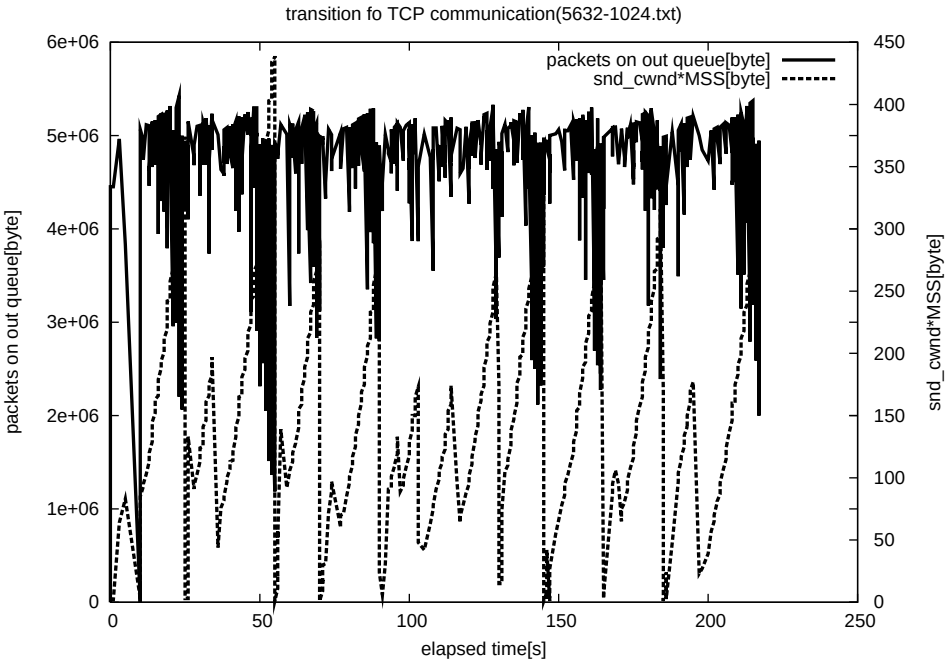

図 5.10: RED による制御 (5632Byte/1024KByte)

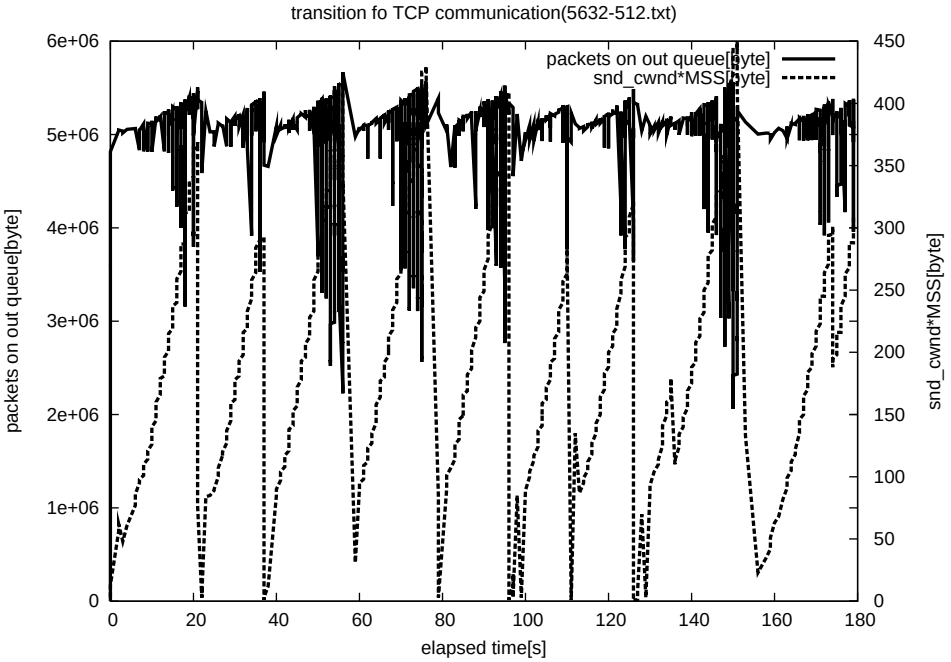

図 5.11: RED による制御 (5632KByte/512KByte)

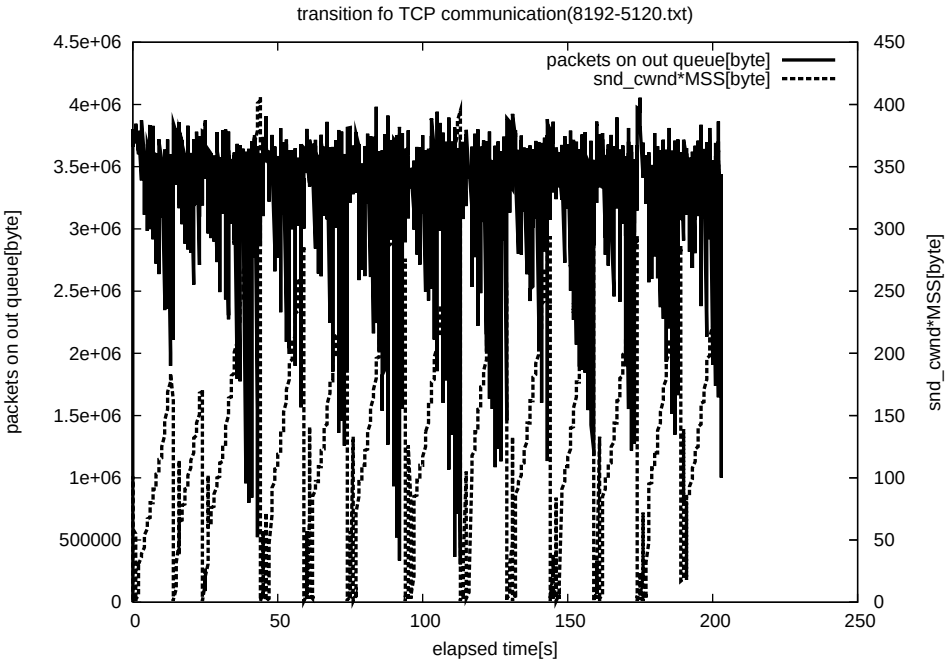

図 5.12: RED による制御 (8192KByte/5120KByte)

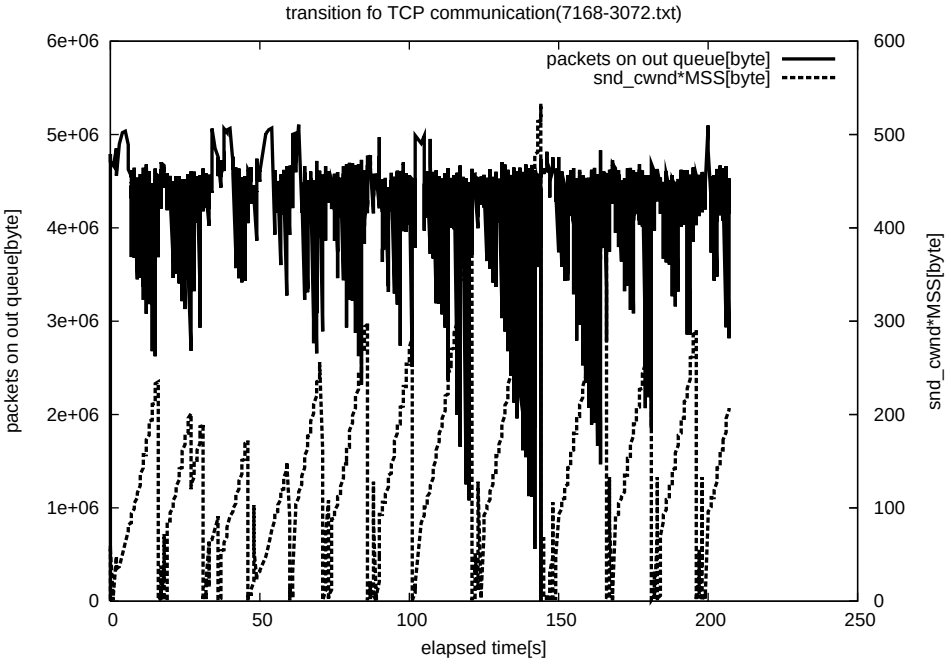

図 5.13: RED による制御 (7168KByte/3072KByte)

Absolute[KB] ╲ RED[KB]	64	128	256	512	1024
8192	2071.8	2426.0	2851.4	1961.8	1660.1
7680	1758.5	2210.9	2117.2	2479.5	1714.9
7168	2384.7	2104.8	1993.2	2293.6	2178.0
6656	1607.2	2104.8	1993.2	2293.6	2178.0
6144	2088.8	2167.1	2098.5	1814.3	2535.4
5632	2481.8	2076.8	2335.7	2800.0	3025.5

Absolute[KB] ╲ RED[KB]	2048	3072	4096	5120	6144
8192	1898.0	2684.6	1981.4	1680.9	2389.2
7680					
7168		2458.0	1888.9		
6656					
6144					
5632					

表 5.8: RED 方式におけるレイテンシ特性 [ms]

1981.35[ms] に減少し、139[ms] の性能改善がみられた。

しかしながら、グラフを見ると制御なしの場合と比較すると、やはり最大輻輳ウィンドウサイズが小さいままであることから、輻輳ウィンドウのサイズが縮小されるごく直前ではバッファのパケット滞留容量が 0 となりそのままスロースタートに戻っていると予想される。

現在の実装では select システムコールによって入出力の準備ができた場合にアクションを発生させるようになっており、また select によってブロックされていない場合には単純にループしているだけとなっている。正しくは細粒度単位で OS を分析してみないことには分らないが、ECE 信号による送信側制御を行なった場合に select の呼び出し回数が増加し大きく削減しすぎてしまうという問題が発生している可能性がある。

また、プロセスは送信側のウィンドウサイズがどうであれ滞留するパケットの数が一定以上だと確率的にせよそうでないにせよ、ECE 信号を送ってしまう。従って、もし送信側のホストの TCP 変数が輻輳ウィンドウの輻輳直後であっても ECE を受け取ってしまうことになる。したがって、この場合では半分になった輻輳ウィンドウがさらに半分になることになる。この輻輳ウィンドウサイズが回復する速度は LAN 内では十分に早いが、逆に円滑な制御を妨げているとも考えられる。

もうひとつの可能性として、今回乱数については GNU glibc における乱数発生実装を利用した。しかしこの乱数の性質については現在のところ調査が終っておらず、これが原因となっている可能性がある。

5.8.3 問題点

実は RED-like の本実装には誤りがある。時間あたりの入出力のデータバイト数が増加すると実質的に多くの ECE 信号が発っせられてしまい、うまくコントロールできていないことがわかった。

この問題の解決のため、インターバルタイマを用いて再実装を行なったが、カーネルのリゾリューションを HZ=100 から HZ=1000 としてもなお、Linux の実装では ECE 信号によるコントロールは難しいことが判明した。

具体的には 100[ms] 程度の間隔でまとまって ECE が実行されてしまう問題があるということである。

カーネルのリゾリューションをこれ以上精細にすることは、Linux の性能の限界を越えてしまうため、インターバルタイマを利用した方法を利用した場合、現在の計算機の性能では目的の機能が達成できないことが判明した。

アプリケーションのプロセスにとってパケットという概念は存在せず、アプリケーションにとってデータはストリームでしかない。従って、ECE による削減方法を実現するためには、(カーネルレベルで実現することができるかは不明だが)、1 つパケットの到着と同期して働く別の機構が必要であると考えられる。

第6章　まとめ

6.1　本研究のまとめ

本論文では広帯域・高遅延ネットワーク環境下における Reno を用いた場合における性能劣化の問題及び HighSpeed TCP を導入した場合における不公平性の問題を取り挙げ、その解決方法として性能改善のためのプロキシ機構を用いて、不公平性を改善する方法を示した。

評価のため SuperSINET ナノテク VPN を用いた。本環境において、netperf を用いて、10 本の HighSpeed TCP フロー環境下における Reno の性能測定を行ない、70.7Mbps のスループットを計測した。次の本提案装置を挿入することでこのスループットが 93.0Mbps に向上しスループットが向上していることが確認できた。

また、レイテンシの発生原因について考察し、レイテンシがソケットバッファのサイズに左右されることを明らかにした。明示的な輻輳通知機構 (ECN) をアプリケーションが利用可能となるようカーネルに修正を加え、もしソケットバッファに滞留するパケットの数が多すぎる場合には ECE 信号を送ることでソケットバッファに滞留するパケット数を削減できるよう試みた。残念ながら、今までのところ、この試みは失敗に終っている。原因としては、

1. アプリケーションにとってデータの受信はパケットの受信間隔と等しくなく、ただのストリームにすぎないこと。
2. インターバルタイマを使った試みでは、Linux におけるタイマ精度がパケットの到着する間隔と比較して十分に高くないこと。

が考えられる。

6.2　研究より発生した課題

6.2.1　実装方式に関する考察

今回 OS として Linux を用いソケットを利用し実装を行なった。しかしながら、最大の帯域幅はおよそ 90Mbps 程度であるという結果が得られた。原因がバス幅に由来するのか、そうでないのかは不明だが、現実的な問題として 90Mbps では不十分であると考えられる。

本実装では、アプリケーションプロセス空間へのデータのコピー及びアプリケーションプロセス空間からのデータのコピーの 2 回の余分なコピーが発生しており、これをゼロコピーに置き換えるなどのことにより、スループットの向上を図れる可能性がある。

また、今回はソフトウェアによる実装を用いたが、ハードウェアによる実装も考えられる。しかしながら、一般に TCP の再実装を行うことは高いコストを要するため、今回は採用を見送った。しかし、ハードウェアならば、一般的には各通信に対し完全に並列に動作させることが可能であろう、つまりソフトウェアで必要であった超高速な文脈切り替えを必要としないので、今回達成できなかった明示的輻輳通知による制御方法を成功させることができた可能性がある。

また、ソフトウェアとハードウェアによる複合的な構成も考えられる。例えば高度な並列性を要するフロー毎の制御においては、ハードウェアの制御機構を利用し、そうでない部分にはソフトウェアを利用するといった具合である。例えば、TCP のフロー毎のキューイングを専用の FPGA ボードによって行なって帯域幅を精細にコントロールするようなシステムの構成を検討する。現在の Linux ないしはフリーの BSD システムにおいては、インタフェースカードから入力されたパケットは直ちに主記憶装置に取り込まれ処理がなされる設計となっている。しかしながら、これでは必ず主記憶を経由してしまうため、これから来たるべき広帯域環境にとって、性能が低下してしまう原因となってしまうことは明らかである。これを防ぐためには、インタフェースのドライバを改変し、キューイング用の FPGA ボードに直接転送するといったことによって可能である。しかし、これでは柔軟性を損なってしまう。そこで、デバイスからデバイスへの直接データ転送を、任意のデバイス間同士で行うことができる、こういった指示を抽象化して記述するシステムなどがあれば、メモリーボトルネックが発生せず、高速なシステムが実現できると考えられる。

6.2.2 最適なパラメタを求める方法

今回最適なソケットバッファに滞留するパケットの数は、別の方法で求められているという条件の元に明示的輻輳通知による制御方法を用いて制御を行なった。しかし、実際にうまく動作させるためには、最適値を何らかの方法で求めねばならず、それに対する追及が現状では不十分である。

最適なソケットバッファのサイズとは、おそらく、システムによって異なるであろうし、また回線のパケット損失率・遅延等も影響するであろう。また、工学的応用の見知から察っするに、これらのパラメタから最適なバッファサイズを求めることができることが大変望ましい。

上で述べたことを実現するためには、理論モデルまたはシミュレーションによって求めることができなくてはならないし、また、実際のネットワークの観測に基いて行うことができなくてはならない。しかしながら、そのための方法の追及が十分でない。

6.2.3 性能の評価方法等の改善

今回性能評価を行なった。しかしながら、システムの動作に当っては相互に影響を与える多くのパラメタが存在する。例えば、ある Linux におけるシステムのパラメタ値である sysctl の出力行は総数にして 293 行ある。また、影響を与えるのは動的に可変可能なパラメタ値ばかりではなく、例えばカーネル内に静的に埋め込まれたパラメタや、コンパイル時に失なわれてしまった情報など、カーネルのヴァージョン、コンパイラによる出力コードの相違、その時々のネットワークの状態など多くの要素が関係する。

一般的には、多くの実験ではこのうちの 1 条件を選択し変化させることでどう変化するかを見ていくことになるのであろう。しかしながら、こうした方法における実験効率は著しく悪い。

実験の効率化のためには、実験の各種パラメタを自動的に設定取得するためのシステム及び実験を効率良く記述できるシステムの開発が重要であると考えられる。実験のシステマティックな記録方法として、実験をリポジトリツリーとして記録したり、または多次元データとして記録するなどの工夫が考えられる。

第7章　付録

7.1　実際のネットワークの挙動

7.1.1　本実験の目的

本実験はSuperSINET-ナノテクVPN上でTCPを利用した場合における性能を確認するために行う実験である。

7.1.2　実験の方法

本実験は当大学から東北大学を結ぶ専用線上でTCPによって連続した通信を行なった場合に送信者の輻輳ウィンドウがどのように変化するかを観測したものである。図5.4からDummynet用の計算機を取り除いて直結し、マシンHSTCPからデータの送信を東北大学にあるマシンに向けて行なった。

本大学から東北大学に置かれている計算機に対し連続したデータ転送を行なう。1024バイト単位のデータを連続してソケットバッファに5120MByte書き込んだ場合におけるTCPの輻輳ウィンドウサイズの変換を記録する。また1MByte毎にgettimeofday()を使って取得した時間を埋め込み、エコーバック時間の計測を行なう。

本実験用いた環境を次に示す。

- 送信側: Linux 2.4.20 with AltAIMD-0.3 and Web100-2.3.3
- 受信側: Linux 2.4.22
- 送信側・受信側: Pentium IV 2.8GHz, FSB800MHz
- 送信側ソケットバッファサイズ: 2MByte, 受信側ソケットバッファ: 8MByte
- 送信側txqueuelen: 1000, 受信側txqueuelen: 1000
- ECN及びSACK/DSACK: 無効

図7.1と図7.2に計測の結果を示す。

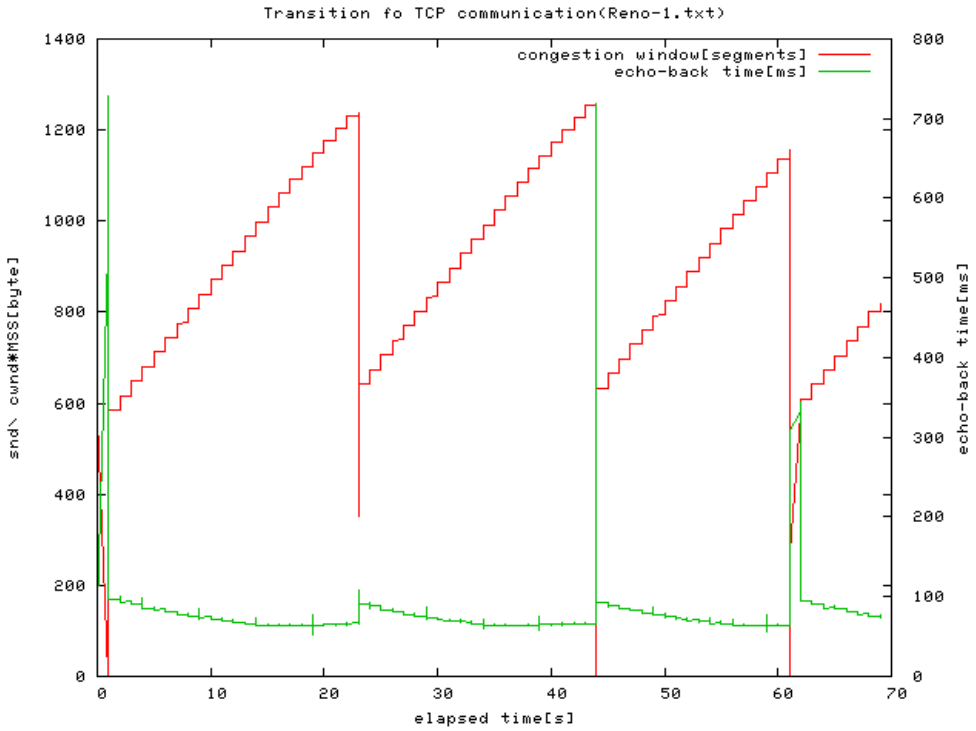

図 7.1: Reno における輻輳ウィンドウと遅延

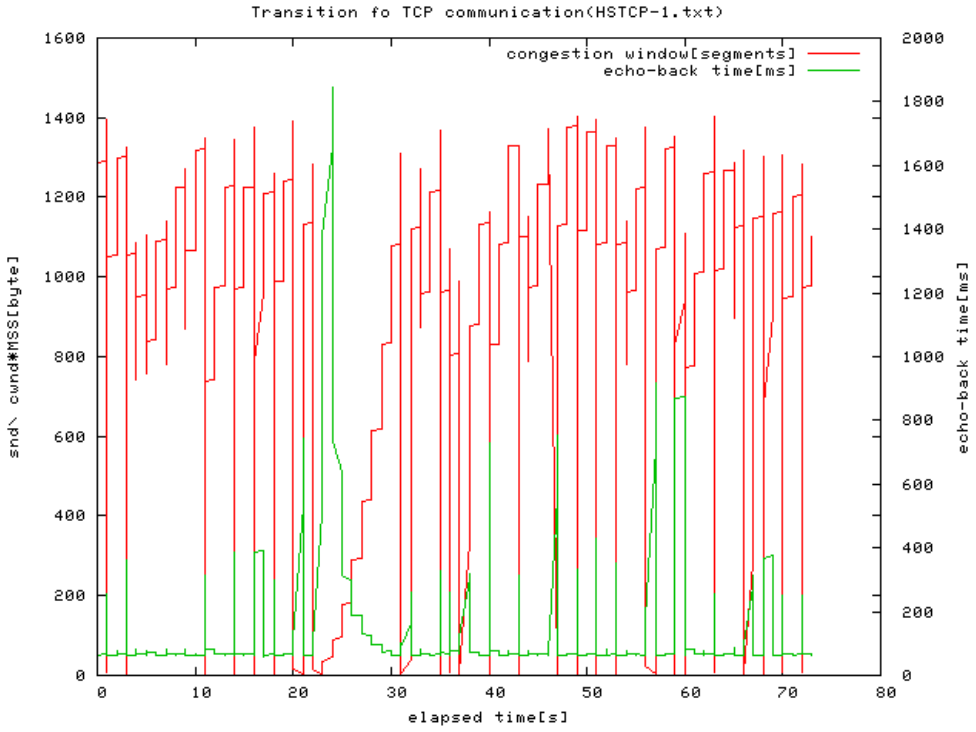

図 7.2: HighSpeed TCP における輻輳ウィンドウと遅延

	Round1		Round2		Round3		Average	
	Thr.	Avg.	Thr.	Avg.	Thr.	Avg.	Thr.	Avg.
Reno	593.6	74.5	561.1	77.3	487.6	86.1	547.4	79.3
HighSpeed TCP	561.1	80.2	568.9	80.1	493.5	89.8	541.2	83.4

表 7.1: スループット [Mbps] とレイテンシ [ms]

7.1.3 結果及び考察

このように、実際のネットワークでは帯域幅が一杯になった場合には必ずウィンドウサイズが縮小されることがわかった。また、本環境下では Reno と HighSpeed TCP のスループットはほとんど変らないものの、HighSpeed TCP では突如としてレイテンシが大きくなる問題点があることも判明した。広帯域・高遅延環境下では望ましいと一般的に言われている HighSpeed TCP よりも、SuperSINET-ナノテク VPN 上では HighSpeed TCP より Reno のほうが望ましい特性が得られることも判明した。

HighSpeed TCP が Reno よりもネットワークに与える負荷は大きいということは従来より知られていたが、本結果は HighSpeed TCP の優位性を反証を補強するものである。

7.2 Dummynet の挙動に関する調査

7.2.1 概要

調査を行う段階で Dummynet[Riz97] を用いた。しかしながら、送信側ホストの輻輳ウィンドウ変数の挙動を観察していると、実際のネットワークにおける挙動と一致しない点があるので、実験を行い確認を行なった。

いくつかの実験結果から、Dummynet のキューサイズのパラメタの値によりこの問題が発生することが判明した。

7.2.2 結果

実験の結果を図 7.3 から図 7.9 に示す。Dummynet のキューサイズにおいて 16KBytes〜32KBytes 程度までは輻輳ウィンドウは剣山のように変化するのに対し、64KBytes〜128KBytes の間では凸面状になっていき、256KBytes 以上では輻輳制御機能がうまく働かなくなることが判明した。

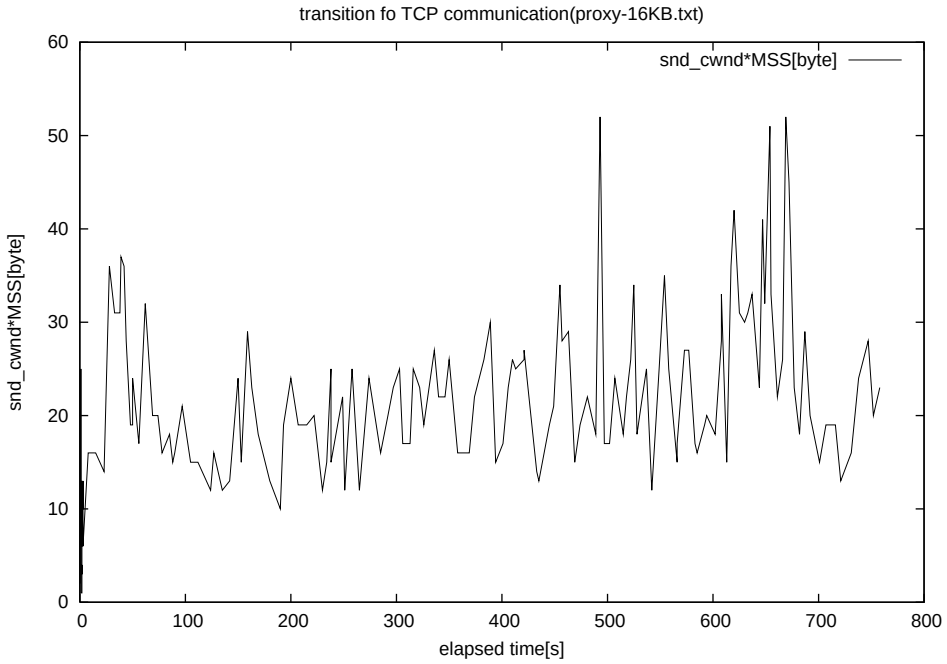

図 7.3: Dummynet におけるキューサイズと輻輳制御 (16KByte)

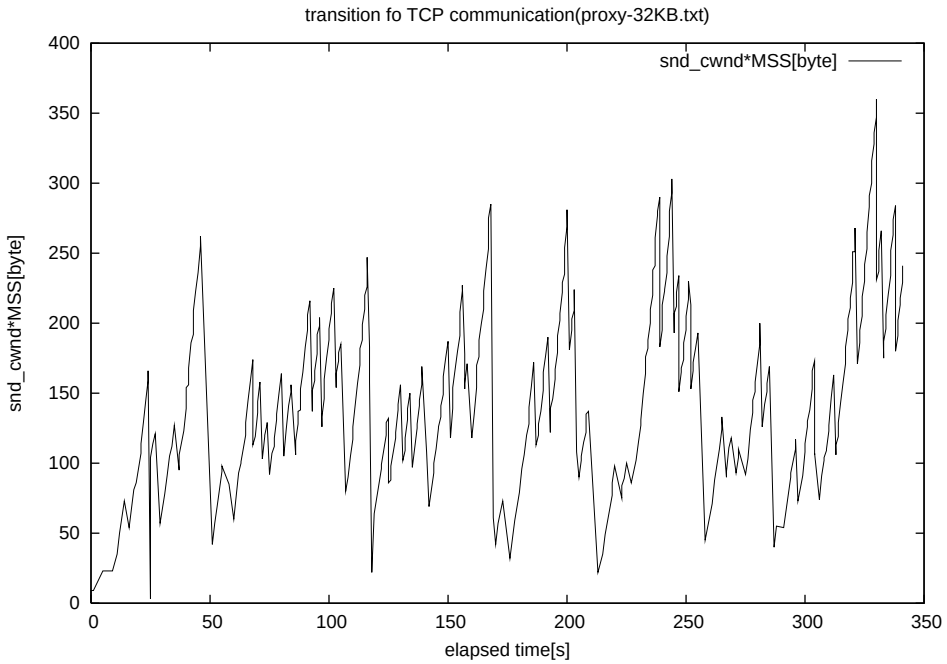

図 7.4: Dummynet におけるキューサイズと輻輳制御 (32KByte)

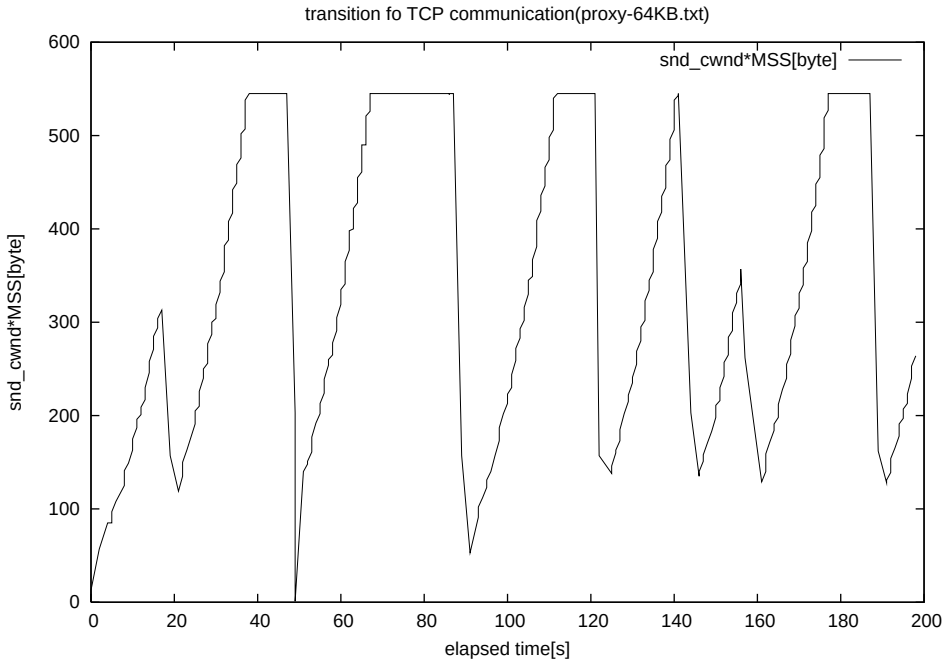

図 7.5: Dummynet におけるキューサイズと輻輳制御 (64KByte)

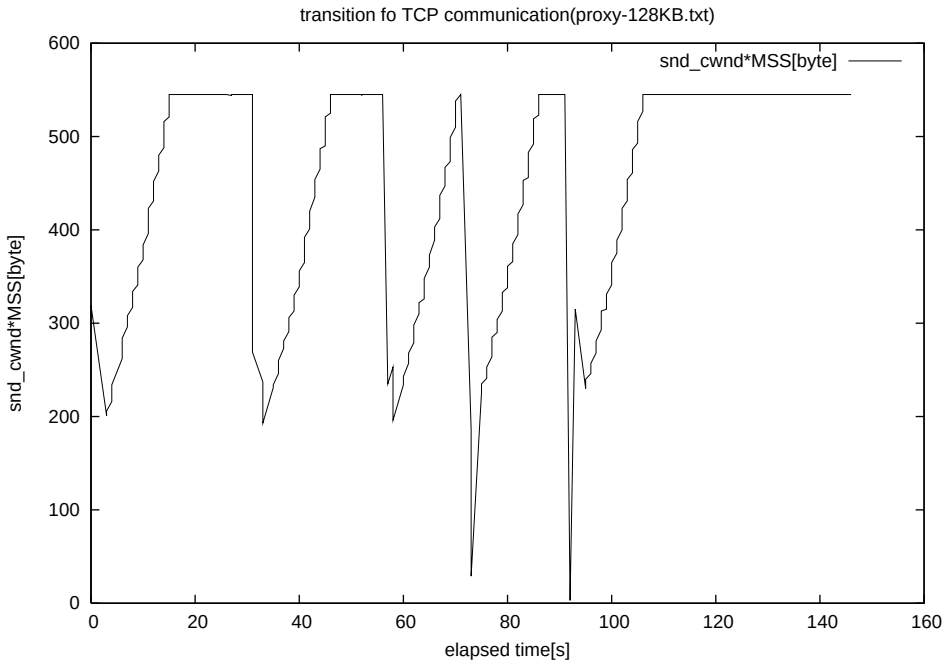

図 7.6: Dummynet におけるキューサイズと輻輳制御 (128KByte)

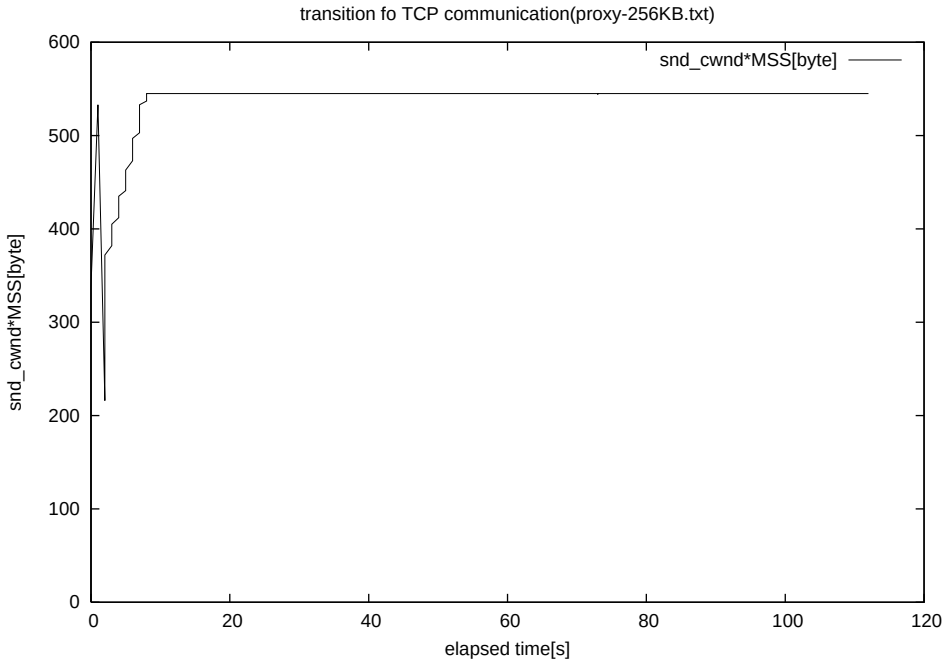

図 7.7: Dummynet におけるキューサイズと輻輳制御 (256KByte)

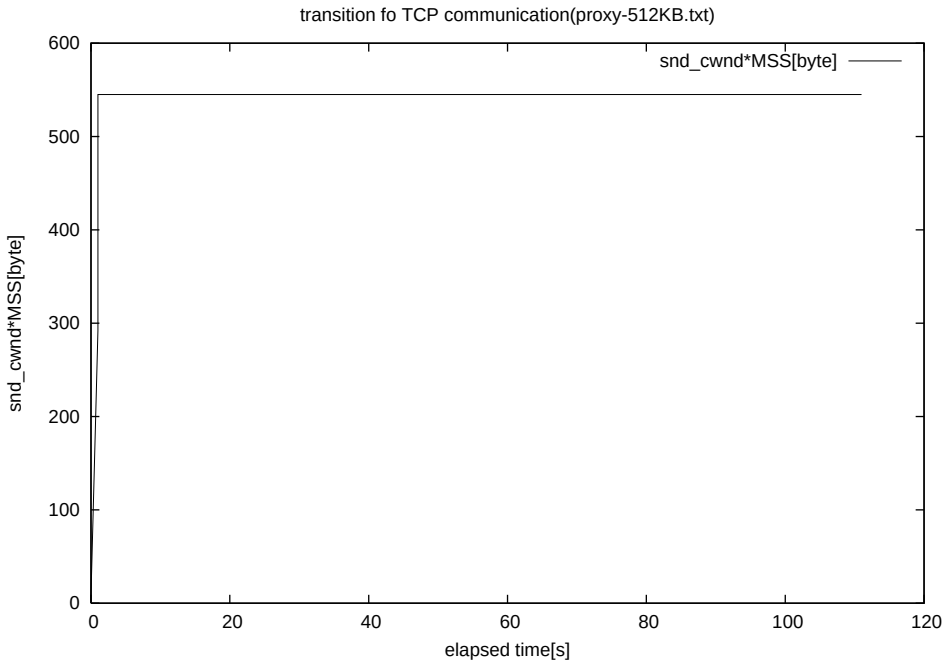

図 7.8: Dummynet におけるキューサイズと輻輳制御 (512KByte)

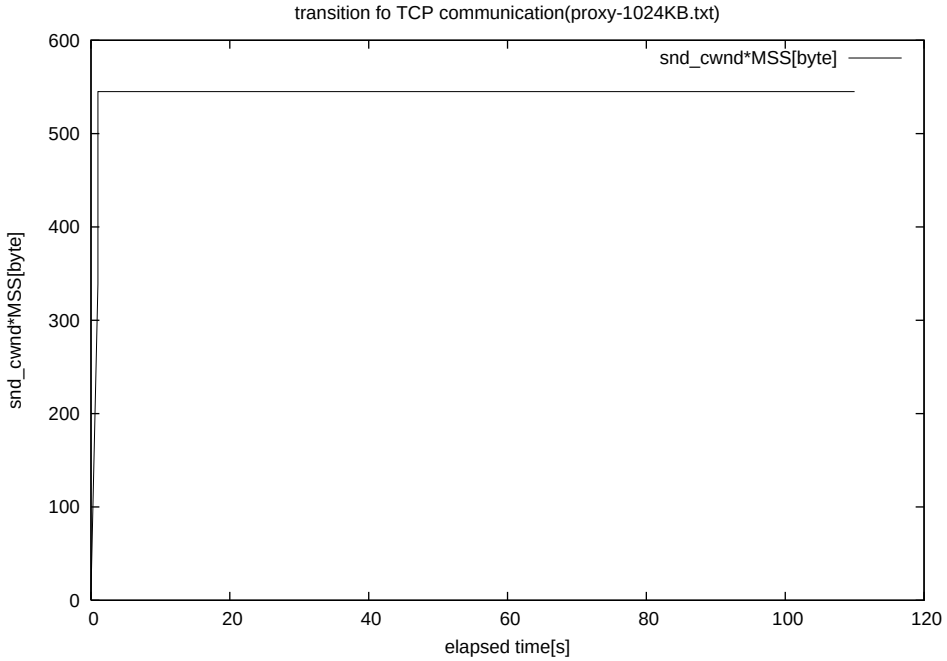

図 7.9: Dummynet におけるキューサイズと輻輳制御 (1024KByte)

7.2.3 考察

原因としては IEEE802.3x によるフロー制御が考えられる。IEEE802.3x によるフロー制御によってデータの通信の輻輳制御が行なわれている場合には、それによってパケットのロスが発生しないまま現在のウィンドウサイズが維持されていると考えられる。

参考文献

[aMHaJPaJW00] Sally Floyd and Mark Handley and Jitendra Padhye and Jorg Widmer. Equation-based congestion control for unicast applications. In *SIGCOMM 2000*, pages 43–56, Stockholm, Sweden, August 2000.

[aMKaJBaGMaZS01] J. Border and M. Kojo and J. Briner and G. Montenegro and Z. Shelby. Performance enhancing proxies intended to mitigate link-related degradations, June 2001.

[aSKaSS89] A. Demers and S. Keshav and S. Shenker. Analysis and simulation of a fair queueing algorithm. *SIGCOMM Comput. Commun. Rev.*, 19(4):1–12, 1989.

[aSSaRKaSSaCP02] Aditya Akella and Srinivasan Seshan and Richard Karp and Scott Shenker and Christos Papadimitriou. Selfish behavior and stability of the internet:: a game-theoretic analysis of tcp. *SIGCOMM Comput. Commun. Rev.*, 32(4):117–130, 2002.

[aSWOaLLP94] Lawrence S. Brakmo and Sean W. O'Malley and Larry L. Peterson. TCP vegas: New techniques for congestion detection and avoidance. In *SIGCOMM*, pages 24–35, 1994.

[aVJ93] Sally Floyd and Van Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking*, 1(4):397–413, 1993.

[dSaDA03] Evandro de Souza and Deb Agarwal. A highspeed tcp study: Characteristics and deployment issues, 2003.

[Flo02] S. Floyd. Limited slow-start for tcp with large congestion windows, June 2002.

[Flo03] S. Floyd. HighSpeed TCP for large congestion windows, February 2003.

[Jac88] Van Jacobson. Congestion avoidance and control. In *ACM SIGCOMM '88*, pages 314–329, Stanford, CA, August 1988.

[Jon] Rick Jones. Welcome to the public netperf homepage. http://www.netperf.org/netperf/NetperfPage.html.

[Kel03] T. Kelly. Scalable tcp: Improving performance in highspeed wide area networks, 2003.

[Riz97] Luigi Rizzo. Dummynet: a simple approach to the evaluation of network protocols. *ACM Computer Communication Review*, 27(1):31–41, 1997.