卒業論文　　2005年度(平成17年度)

TranSwitch：ネットワークフロー毎における最適なTCPへの動的切替機構

指導教員

慶應義塾大学環境情報学部

徳田　英幸

村井　純

楠本　博之

中村　修

高汐　一紀

涌川　隆次

慶應義塾大学総合政策学部

山下 勝司

卒業論文要旨　　2005 年度 (平成 17 年度)

TranSwitch：ネットワークフロー毎における最適な TCP への動的切替機構

概要

近年、通信技術の発達により、通信速度、通信形態においてコンピュータネットワークが多様化している。このようなネットワークの多様化に対応する形で、広帯域高遅延ネットワークや無線ネットワークなどの特定のネットワークの特性を考慮して設計された新規 TCP (Transmission Control Protocol) がこれまでに数多く提案されている。これらの新規 TCP は想定されたネットワーク環境で汎用性を重視した TCP よりも高い性能を発揮する。しかし、現在一般に広く普及している OS (Operating System) は TCP を一つしか導入できないため、多様なネットワーク環境に対応できる汎用性の高い TCP を用いる必要がある。このため、特定のネットワークでの利用を想定した新規 TCP を現在の OS で利用することは難しいという問題がある。

本稿では、この問題を解決するための機構として TranSwitch を提案する。TranSwitch は OS へ複数の TCP の導入を可能にし、これら複数の TCP からアプリケーションが利用する TCP をエンドユーザがネットワークフロー毎に選択することを可能にする。複数の TCP を利用することにより、従来の汎用的な TCP による多様なネットワークへの汎用性の維持と新規 TCP による特定環境でのパフォーマンス向上の共存が可能となる。本論文ではこの TranSwitch の設計・実装について述べ、プロトタイプ実装の評価結果を用いて TranSwitch の有効性を示した。

慶應義塾大学　総合政策学部
山下 勝司

Abstract of Bachelor's Thesis
TranSwitch : an Architecture for Dynamically Switching to Suitable TCP for Each Network Flow

Abstract

The past few years have seen a rouse of improvement of network technology. This improvement of network technology makes diversification of computer network in terms of the transmission rate and the communication form. Speed of backbone network of ISP (Internet Service Provider) and broadband networks such as ADSL or FTTH are different from that of narrowband networks such as ISDN or dial-up. And, a wired communication exists together to wireless one in a network.

A lot of improvment TCP (transmission control protocol) have been proposed for correspondance to this diversification of the network, such as TCP for long fat pipe network and another for wireless network. These TCP achieve higher performance than general TCP in the assumed network environment. The TCP are, however, not deployed widely due to some reasons. One of the reasons is that for current OS's do not support multiple different TCP, it is necessary that the only TCP can correspond to various network environments. This makes it difficult for end users to use improved versions of TCP which do not have high generality.

We propose an architecture called TranSwitch to solve this problem. TranSwitch enables end users to introduce two or more TCP into an OS, and switch a TCP according to a network flow to another without modifying the application. In this paper, we present a design and implementation of TranSwitch and effectiveness of TranSwitch with results of evaluation.

Katsushi Yamashita

Faculty of Policy Management
Keio University

目 次

図 目 次

表 目 次

第1章　序論

本章では本研究の動機と目的および本論文の構成について述べる。

1.1 動機

近年、通信技術の発達により、通信速度、通信形態においてコンピュータネットワークが多様化している。インターネットサービスプロバイダが持つコアネットワークの通信速度は数 10Gbps と広帯域化し、一般用のインターネット接続においても、ADSL や FTTH といったブロードバンド接続の通信速度は数 Mbps から 100 Mbps と広帯域化している。同様に LAN (Local Area Network) においても 100 Mbps、1 Gbps と高い通信速度が実現されている。ブロードバンド接続の契約回線数は図 1.1 にあるようにここ数年で増加し続け、2004 年に 1951 万回線と広く普及しているが、その一方で、ブロードバンド接続が普及していない地域ではダイアルアップや ISDN 等の低速なインターネット接続が現在でも広く利用されている。図 1.1 は日本国内におけるブロードバンド、ISDN、ダイアルアップそれぞれを用いたインターネットへの接続の割合を示しており、2004 年度のダイアルアップと ISDN を用いた低速なインターネット接続は全体の約 38.49%を占め、現在でも広く利用されていることが分かる。また、携帯電話等のモバイル機器をモデムとして用いた低速なインターネット接続もブロードバンド接続と併用して利用されている。これらの低速なインターネット接続の通信速度は数 10 Kbps から数 100 Kbps であり、前述の高速化したネットワークと通信速度の面において大きな開きが生じている。通信形態においては、IEEE 802.3 [1] を用いた有線通信や IEEE 802.11 [2] を用いた無線通信等のネットワークが混在している。

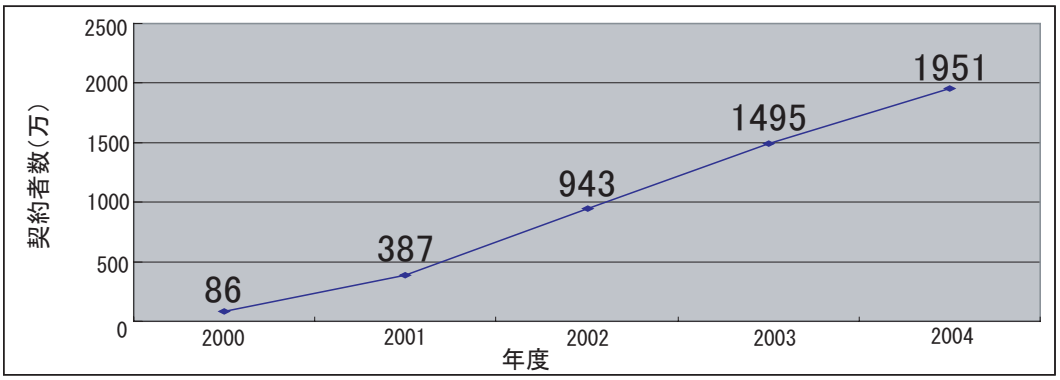


図 1.1: 日本国内におけるブロードバンド契約者数の推移
情報通信白書 平成 17 年度版 [3] より抜粋

このようなネットワークの多様化に対応する形で、それぞれのネットワークの特性を考慮した改良型の TCP (Transmission Control Protocol) [4] がこれまでに数多く提案されている。この改良型 TCP の代表的な例として、無線ネットワーク用の TCP、広帯域高遅延ネットワーク (LFN:Long Fat Network) 用の TCP 等がある。汎用的な TCP はどのような通信環境においても比較的高いパフォーマンスを発揮し、特定の通信環境でパフォーマンスが著しく低下することはない。一方、これらの改良型 TCP は、汎用的な TCP よりも、それぞれの想定された通信環境において、より高いパフォーマンスを発揮する。

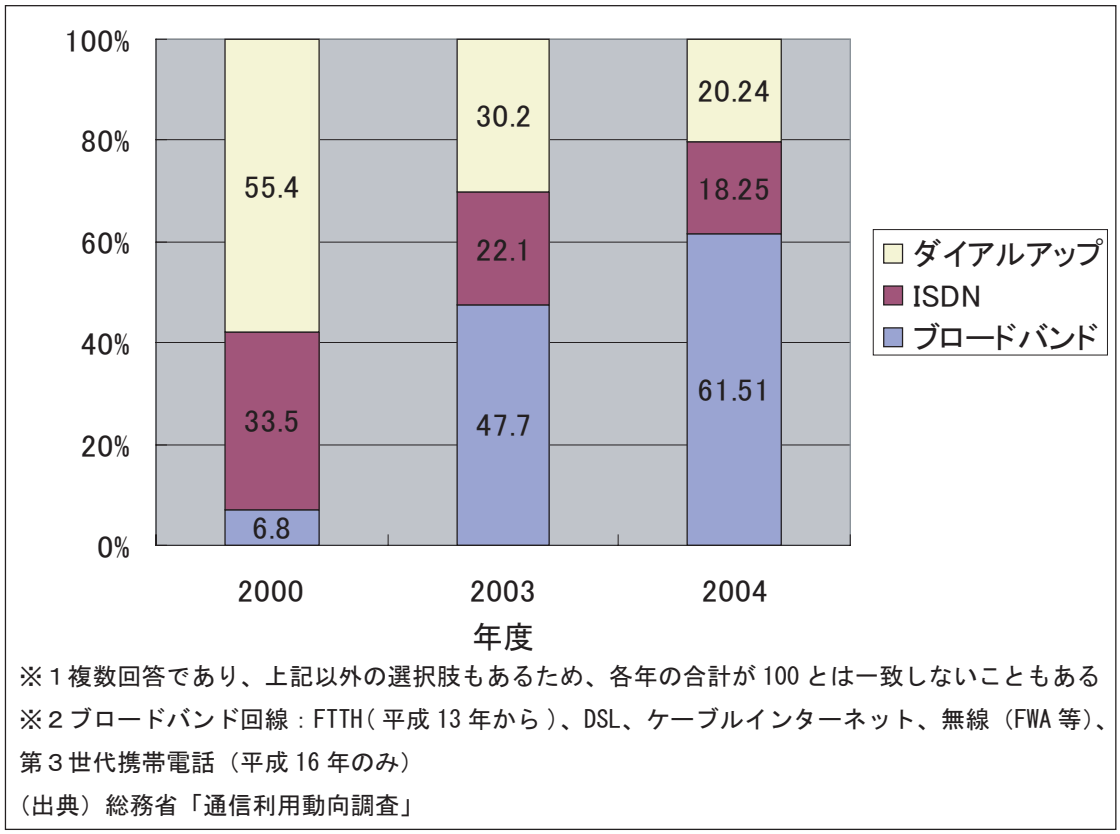


図 1.2: 日本国内におけるインターネット接続種類の内訳
情報通信白書 平成 17 年度版より抜粋

しかし、現在一般に広く普及している OS (Operating System) は TCP を一つしか導入できないため、多様なネットワーク環境に広く対応する必要性から、汎用的な TCP の導入が OS に要求される。このため、特定のネットワークでの利用を想定した改良型 TCP を現在の OS へ導入することは難しい。そのため、これらの改良型 TCP は特定環境において高いパフォーマンスを発揮することが証明されているものの、現在、ほとんどの場合において利用されていない状況にある。

本稿では、この問題を解決するための機構として TranSwitch を提案する。TranSwitch は OS へ複数の TCP の導入を可能にし、これら複数の TCP からアプリケーションが利用する TCP をエンドユーザがネットワークフロー毎に選択することを可能にする。複数の TCP を利用することにより、従来の汎用的な TCP による多様なネットワークへの汎用性の維持と、改良型 TCP による特定環境でのパフォーマンス向上の共存が可能となる。

1.2 本研究の目的

本研究の目的は通信環境にそれぞれ適した改良型 TCP を利用し、よりパフォーマンスの高い通信を可能にすることである。前述したように、従来の OS は一つの TCP しか導入できないため、多様な通信環境に広く対応する必要性から汎用的な TCP を利用しなければならず、特定の通信環境においてより高いパフォーマンスを発揮する改良型 TCP を利用することはできない。本機構は、複数の TCP を導入し、ネットワークフロー毎に適切な TCP の選択を可能にすることで、改良型 TCP を用いたより高いスループットでの通信を実現できる。

1.3 本論文の構成

本論文は全 7 章で構成され、次のように構成される。本章である第 1 章では本研究の動機および目的について述べた。2 章では本研究の背景としてトランスポート層プロトコルと TCP について説明し、現状の TCP が抱える問題点とそれらの解決を目的とした新規 TCP の研究の概要について述べる。3 章では 2 章で述べた新規 TCP が抱える問題点を指摘する。そしてその問題点の解決策を提案し、その提案に類似する関連研究について述べる。4 章では本研究が提案する機構である TranSwitch の設計について述べ、5 章で TranSwitch のプロトタイプ実装について述べる。6 章においてプロトタイプ実装した TranSwitch の評価を行い、オーバヘッドを計測する。7 章では本論文をまとめ、今後の課題について展望する。

第2章　背景

本章では本研究の背景を述べる。まず、本研究の研究対象であるトランスポート層プロトコルと TCP についてそれぞれの概要を述べる。次に、TCP が現状において抱える問題点を述べ、最後にこれまで行われてきた新規 TCP の研究の概要について述べる。

2.1 トランスポート層プロトコルの概要

インターネットを初めとする現在のコンピュータネットワークは TCP/IP 通信方式の技術によって支えられている。ネットワークにおいて、OS やハードウェアが異なる不特定のホストと通信を行うためには、ホスト間の通信方式の互換性が必要である。この通信方式の互換性を定めたモデルが図 2.1 に示す OSI 参照モデル [5] であり、これをもとにして成された通信方式が TCP/IP 通信方式である。図 2.1 に示されてるように、TCP/IP 通信方式において第 4 層に位置するのがトランスポート層プロトコルである。

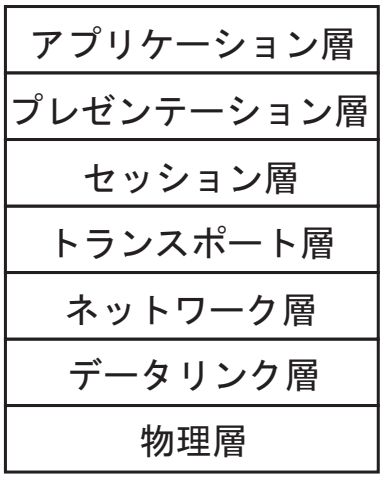


図 2.1: OSI 参照モデル

TCP/IP を用いたホスト間の通信は、パケットと呼ばれる単位に区切られたデータを複数の中間ノードが中継を繰り返すことにより実現する。この中間ノードへ流れるデータ量が中間ノードの処理可能なデータ量を上回った場合、また、回線の回線速度によって処理可能なデータ量を上回った場合、中間ノードは一定量のパケットを破棄する。この状態を輻輳と呼ぶ。

このため、輻輳が発生した場合、送信したパケットが受信側に届かないことが発生する。また、破棄されたパケットの送信は無駄にネットワーク資源を利用し、通信の効率が悪化する。この輻輳を防ぐためにデータ送信量を適切な値に調整し、輻輳の発生を最小限に止めることを輻輳制御 [6] と呼ぶ。また、輻輳によるパケットロスが発生した際、ロスしたパケットを送信ホストが再送し、信頼性を保することを通信の信頼性の保障と呼ぶ。

トランスポート層の下位に位置するネットワーク層は IP (Internet Protocol) [7] 等を用いて通信ホストを特定するが、通信の信頼性の保障や輻輳制御を行わない。このためネットワーク層のみでは、信頼性のある通信や輻輳の制御が不可能であるため、TCP/IP ではトランスポート層プロトコルが通信の信頼性、輻輳制御の有無の定義を行う。現在用いられている代表的なトランスポート層プロトコルとして TCP と UDP (User Datagram Protocol) [8] の二つがあり、TCP は通信を行う両エンドノード上で

動作し、通信の信頼性の保障及び輻輳制御を行う。一方、UDP は通信の信頼性を保障せず、フロー制御を行わない。アプリケーションはこれらのトランスポートプロトコルの特徴を考慮し、アプリケーションが行う通信の性質に適したトランスポート層プロトコルを利用する。

2.2 TCP の概要

TCP は UDP と異なり、通信の信頼性の保障を行い、輻輳制御を行う。TCP の機能のうち、この二つの機能を実現し、本研究に関連する二つの機構を TCP の概要として以下に述べる。

2.2.1 再送機構

2.1 章において前述したように、コンピュータネットワークではパケットロスが生じるため、通信の信頼性の保障が必要である。TCP は通信の信頼性の保障を実現するために再送機構を用いる。再送機構とは、送信ホストが送信したパケットを受信ホストが受信できなかった場合、再度、送信ホストが同じパケットを受信ホストへ送信する機構である。TCP では受信ホストがパケットを受信した際、送信ホストへ ACK と呼ばれる確認応答を送信する。これを図 2.2 に示す。図 2.2 は最初に送信ホストがセグメント 1 を受信ホストに送信し、受信ホストがセグメント 1 を受信した後、送信ホストにそのデータに対する ACK 2 を送信していることを表している。ACK の値は送信ホストが次に送信すべきセグメントの値となる。

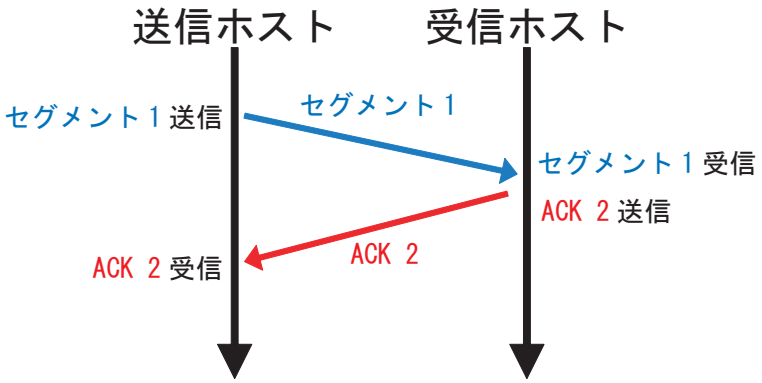


図 2.2: 受信ホストによる ACK の送信

送信ホストは ACK を受信することにより、送信したパケットが受信側に無事到達したことを検知する。送信ホストは ACK が一定時間内に受信できない場合、パケットが受信側に到達しなかったと判断して再送を行う。この一定時間を RTO (Retransmission Time Out) と呼び、またこのメカニズムを再送タイマと呼ぶ。RTO の値はパケットの送信と ACK の受信に必要な往復時間 (RTT:Round Trip Time) より大きな値であ

る必要があるため、通常、RTO の値は RTT を参考に設定される。RTO による再送を図 2.3 に示す。図 2.3 では、まず送信ホストがセグメント 1 を受信ホストに送信するが、途中経路にてセグメント 1 のパケットロスが発生する。送信ホストはセグメント 1 に対する ACK を待つが、ACK を受信できないため RTO により送信ホストはセグメント 1 を再送する。

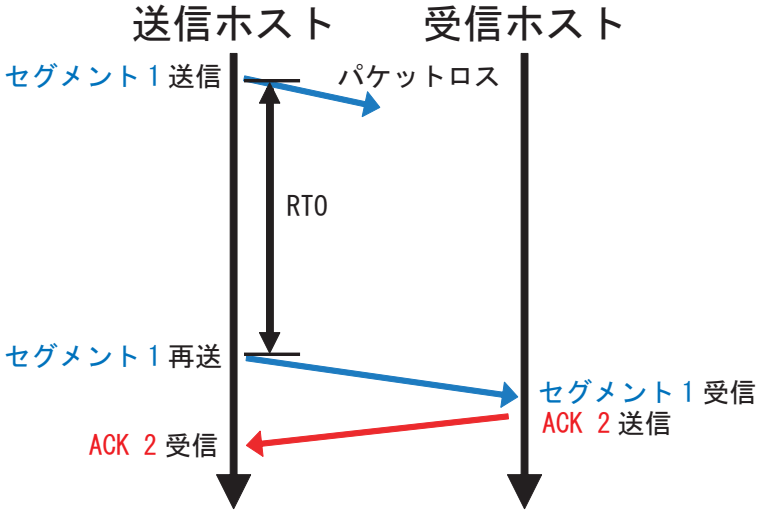


図 2.3: RTO による再送

また、受信側は本来と異なる順番でパケットを受け取った場合、最後に順番通りに受け取ったパケットに対する ACK を再度送信する。この ACK を重複 ACK と呼ぶ。同一の重複 ACK を 3 つ受信した場合においても、送信ホストはパケットロスを検知し再送を行う。重複 ACK によるパケットの再送を図 2.4 に示す。図 2.4 では、まず送信ホストが送信したセグメント 2 のパケットロスが途中経路にて発生する。その後、送信ホストはセグメント 3、4、5 を送信するが、受信ホストはセグメント 2 を受信していないため、セグメント 3、4、5 の受信の度に送信ホストへセグメント 2 の ACK を送信する。送信ホストはセグメント 2 の ACK を三度重複して受信することにより、セグメント 2 を再送する。重複 ACK により、TCP は再送タイマーによるパケットの再送より高速な再送が可能である。

2.2.2 輻輳制御機構

2.1 章において前述したように、輻輳はパケットロスを発生させ通信効率を下げるため、TCP は輻輳を最小限に抑制するため輻輳制御を行う。この輻輳制御により、送信ホストの TCP は一度に送信するデータの量を適切な値に抑制する。この送信ホストが一度に送信するデータのサイズのことをウインドウサイズと呼ぶ。送信ホストが 1 パケットを送信し、そのパケットに対する ACK の受信を待ち、ACK を受信後に次の 1 パケットを送信するといった 1 パケット毎の通信方法では、RTT 毎に 1 パケットしか送信されないためスループットが向上せず通信の効率が悪い。そのため TCP で

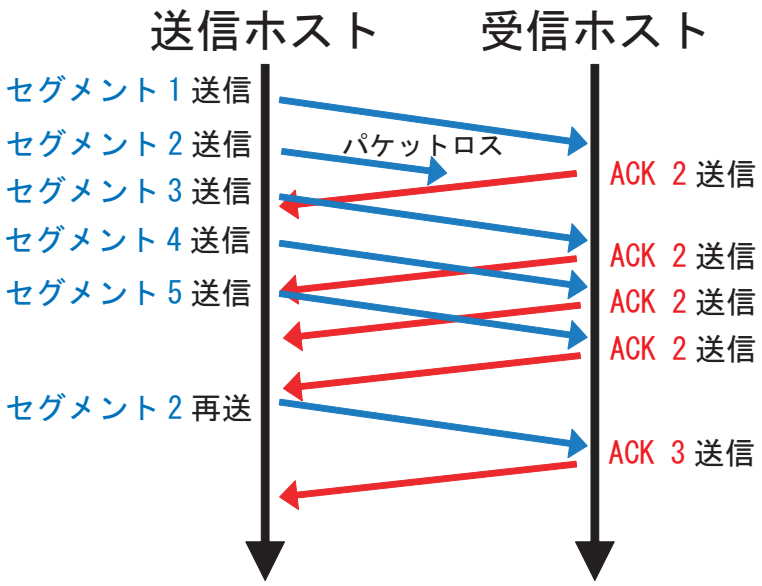


図 2.4: 重複 ACK による再送

は、送信ホストはウインドウサイズ分だけ ACK を待たずにパケットを送信することが可能とされている。これを図 2.5 に示す。図 2.5 では、ウインドウサイズが 4 セグメントの状態における送信ホストのデータの送信を表している。まず、送信ホストはセグメント 1、2、3、4 を ACK を待たずに送信し、セグメント 1 に対する ACK を受信した後、セグメント 5 を送信している。このように送信ホストはウインドウサイズ分だけ ACK を待たずに一度に送信できるため、ウインドウサイズが大きいほど通信のスループットは向上する。輻輳の発生を抑制しつつ、極力効率の良い通信を実現するために、TCP における輻輳制御はウインドウサイズを最適な値に設定することを目的としている。

通信ホスト間における TCP コネクションが確立後、ウインドウサイズは小さな値から始まり、除々に増加していく。これは、ネットワークの輻輳状況が常に変化するため、通信ホストが最適なウインドウサイズを通信開始時から検知することが困難であるためである。また、パケットロスが発生した場合、輻輳を抑えるためウインドウサイズは大きく減少する。このようなウインドウサイズの増減を AIMD (Additive Increase and Multiplicative Decrease) [9] と呼ぶ。以下にウインドウサイズの増減それぞれについて述べる。

- ウインドウサイズの増加
  ウインドウサイズの増加において、スロースタートフェイズと輻輳回避フェイズの二つのフェイズが存在する。ウインドウサイズがスロースタートの閾値を超えない状態をスロースタートフェイズと呼び、閾値を超えた状態を輻輳回避フェイズと呼ぶ。また、この閾値をスロースタート閾値と呼ぶ。それぞれのフェイズにおいて、ウインドウサイズは異なる幅で増加する。ウインドウサイズは ACK を受信する度に、スロースタートフェイズでは指数関数的に増加し、輻輳回避フェ

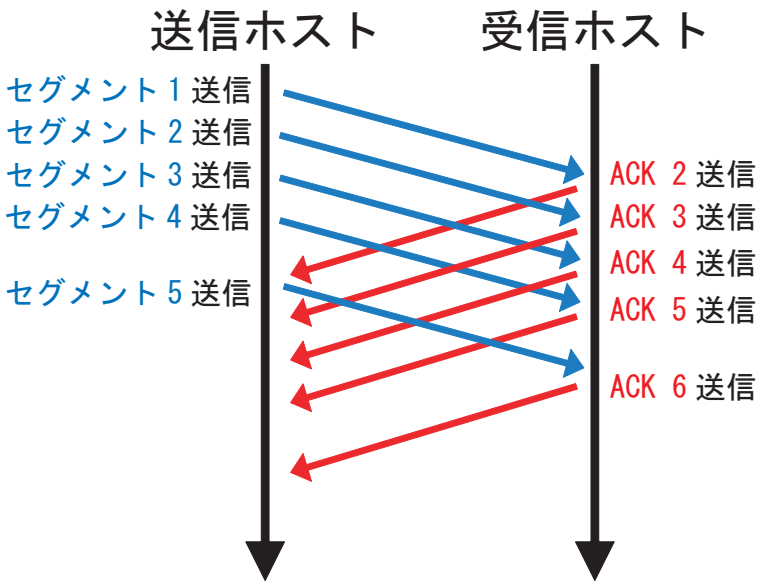


図 2.5: ウインドウサイズに基いた輻輳制御

イズでは線形的に増加する。基本仕様の TCP におけるウインドウサイズは RTT 毎にスロースタートフェイズでは 2 倍ずつ増加し、輻輳回避フェイズでは 1 セグメント分ずつ増加する。これを図 2.6 に示す。図 2.6 では、ウインドウサイズはスロースタートフェイズとして 1 セグメントから始まり、この場合スロースタート閾値が 8 セグメントとなっているため、8 セグメントまでウインドウサイズは RTT 毎に 2 倍ずつ増加する。Time 3 の時点でウインドウサイズがスロースタート閾値と同じ 8 セグメントとなり、輻輳回避フェイズに移行する。ここからウインドウサイズは RTT 毎に 1 セグメントずつ増加しているのがわかる。

- ウインドウサイズの減少
  TCP は 2.1 章にて前述した RTO と重複 ACK の受信という 2 通りの方法でパケットロスを検知する。パケットロスが検出された際、基本仕様の TCP ではそれぞれ以下の値にウインドウサイズは減少する。RTO による再送が発生した場合、ウインドウサイズは 1 セグメント分に減少する。一方、重複 ACK を受信した場合、ウインドウサイズは現在のウインドウサイズの半分に減少 [10] する。これを図 2.7 と図 2.8 に示す。図 2.7 では、Time 16 の時点で RTO によりウインドウサイズが 1 セグメントに減少している。図 2.8 では、Time 16 の時点で重複 ACK によりウインドウサイズが半分の 10 セグメントに減少している。RTO と重複 ACK を受信した場合とでウインドウサイズの減少幅が異なるのは、重複 ACK の受信の場合、順番は異なるが送信した他のパケットは受信されているため、RTO が発生した場合よりも輻輳の程度が軽いと考えられるためである。

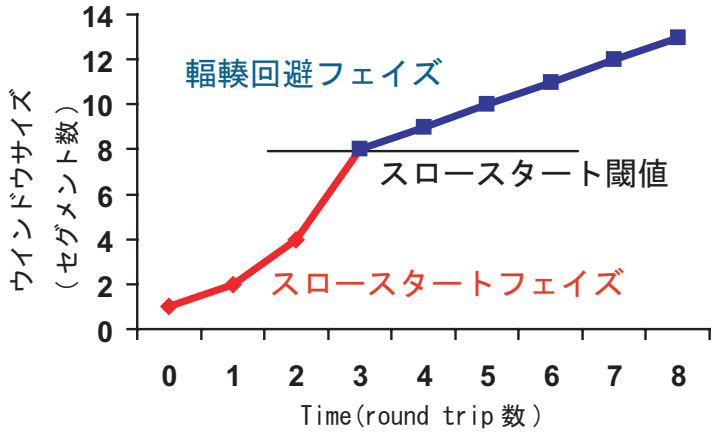


図 2.6: ウインドウサイズの増加

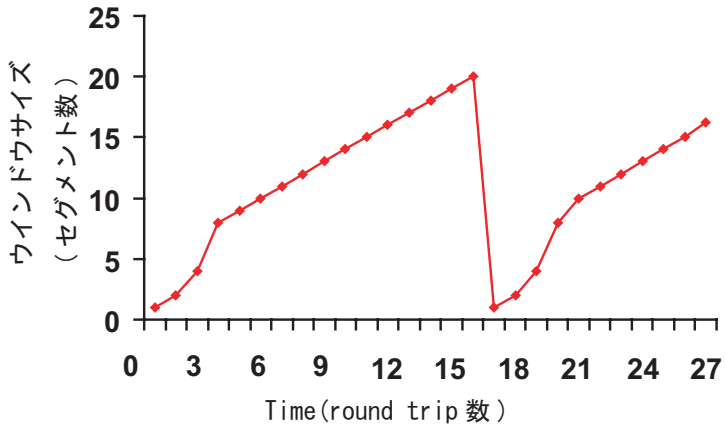


図 2.7: RTO によるウインドウサイズの減少

2.3 TCP の問題点

2.2 章にて前述した TCP の機構に関して、様々な問題点がこれまでに指摘されている。本節では、ネットワークの多様化に関わる TCP の問題点として、輻輳の指標に関する問題とウインドウサイズの増減に関する問題の二つを取り上げ、それぞれの概要を述べる。

2.3.1 輻輳の指標に関する問題

2.2.2 章で前述したように基本仕様の TCP はパケットロスの原因が輻輳のみと想定している。しかし、現在のコンピュータネットワークでは輻輳以外にもパケットロス

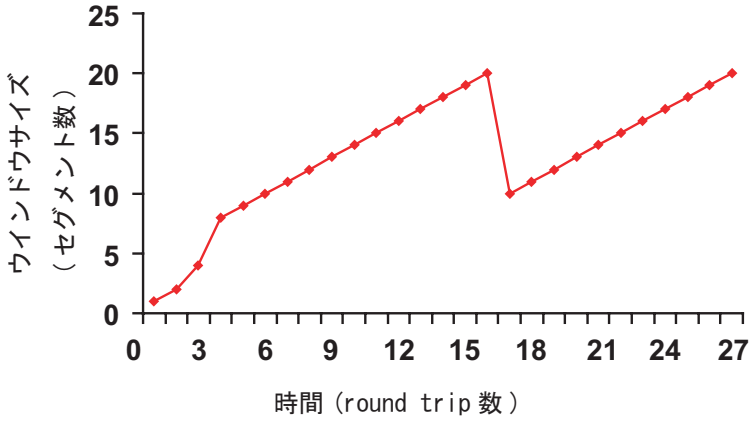


図 2.8: 重複 ACK の受信によるウインドウサイズの減少

を発生させる要因が存在する。その大きなものとして、無線通信の際のコリジョンと伝送品質劣化によるデータ誤りがある。

ホスト間の通信は信号の送受信によって成立するが、同一通信路で信号が同時に複数送信された場合、信号が衝突するため、信号は正常に受信されない。この信号の衝突をコリジョンと呼ぶ。このコリジョンを回避するため、全二重通信や CSMA/CD (Carrier Sense Multiple Access with Collision Detection) [11] などの通信方式が用いられている。しかし、無線ネットワークではコリジョンの発生をホストが検知できないため、現在の方式の無線通信ではコリジョンによるパケットロスを完全に回避することはできない。このため、昨今一般に広く普及している 802.11 を用いた無線通信では輻輳が発生していない状況下でも、無線通信のコリジョンによりパケットロスが発生する。

また、無線通信は有線通信と比較して伝送品質が大きく劣化するため、ビット誤りが頻繁に発生する。データリンク層におけるチェックサムによりビット誤りが検出され、ビット誤りを訂正できない場合、受信ホストはパケットを正常に受信できず、パケットロスが発生する。

輻輳以外のこれらの原因によるパケットロスの発生により、実際のネットワークにおける輻輳の程度とは関係なくウインドウサイズの減少が起こる。このように輻輳以外にもパケットロスの発生要因が存在する無線ネットワークのような通信環境において、輻輳のみをパケットロスの発生原因と見なす輻輳制御では正確に輻輳制御ができないという問題がある。

2.3.2 ウインドウサイズの増減に関する問題

従来の基本仕様の TCP において、ウインドウサイズは 2.2.2 章で前述したように増減する。しかし、昨今ではネットワークが広帯域化しており、これらの広帯域ネット

ワークにおいて、2.2.2 章で述べた増減の値は増加幅としては過小であり、減少幅としては過大であるという問題がある。また、RTT 毎にウインドウサイズは増加するため、高遅延なネットワークではウインドウサイズの増加タイミングが遅くなるため、スループットが高くなりにくい。2.2.2 章で述べた増減値を用いて MTU 1500byte、RTT 100ms の通信環境で 10Gbps の帯域を実現するためには、式 2.1 の帯域遅延積により、83333 セグメントの値のウインドウサイズが必要となる。

$$\text{スループット}\,(bit/s) = \text{ウインドウサイズ}\,(bit)/RTT(sec) \tag{2.1}$$

このウインドウサイズを実現するためには 50 億パケットに 1 回のパケットロスしか許容できない。これは現在のインターネットの通信では事実上不可能なパケットロスの割合である。ネットワークの通信速度はこれからも向上することが予想され、このような広帯域をアプリケーションが有効利用するには現在のウインドウサイズの増減値では難しいという問題がある。

2.4 新規 TCP の研究

2.3 章で示した問題点を解決するため、TCP を改良する研究がこれまでに多くなされてきた。新規 TCP の研究は、Vegas [12] などのネットワークの多様性に対する汎用性を持たせつつより高い性能を目指す研究と特定のネットワーク環境が持つ問題点を解決することで高い性能を得る研究の二つの方向性で行われている。本節ではこれらの新規 TCP の研究のうち、本研究とより強い関連性を持つ特定環境に特化した新規 TCP の代表的な研究として無線ネットワーク用 TCP と広帯域高遅延ネットワーク用 TCP を取り上げる。

2.4.1 無線ネットワーク用 TCP

2.2.2 章において前述したように、基本仕様の TCP はパケットロスを指標として用いて輻輳制御を行う。しかし 2.3.1 章において前述したように、パケットロスを指標とした輻輳制御では、無線環境におけるコリジョンや伝送品質の劣化によるパケットロスを輻輳として検知してしまう。そのため、基本仕様の TCP を用いた無線通信では、実際のネットワークにおける輻輳の程度とは関係なくウインドウサイズが減少する。これらの問題を解決する TCP として Westwood [13]、M-tcp [14]、Mobile-TCP [15] などの無線ネットワーク用 TCP が提案されている。

無線ネットワーク用 TCP の幾つかは輻輳の指標として RTT の利用を提案している。輻輳が発生した場合、パケットが中継ホストのキュー内に滞在する期間が長くなるため、輻輳が発生していない場合と比較して RTT が高くなる。そのため、RTT の変動を計測することで輻輳のより詳細な推測が可能である。このように RTT を輻輳の指標として用いることで、コリジョンやビット誤りによるパケットロスが発生した際のウインドウサイズの必要以上の低下を防止できる。

2.4.2 広帯域高遅延ネットワーク用 TCP

2.3.2 章において前述したように、広帯域ネットワークや高遅延ネットワークでは、基本仕様の TCP のウインドウサイズの増減値を用いて高いスループットを得ることは難しい。そのため、アプリケーションによる広帯域ネットワークの有効利用が難しいという問題がある。この問題を解決するため、これまで複数の広帯域ネットワーク用の TCP や高遅延ネットワーク用の TCP が提案されており、代表的な TCP として Highspeed TCP [16]、Scalable TCP [17]、FAST [18] などがある。これらの TCP は基本仕様の TCP とは異なる AIMD アルゴリズムを採用しており、広帯域高遅延用 TCP の AIMD アルゴリズムは基本仕様の TCP と比較し、ウインドウサイズの増加幅が大きく、パケットロスが発生した際のウインドウサイズの減少幅が小さくなっている。このような AIMD アルゴリズムを採用することでより高いスループットを実現している。

しかし、異なる AIMD アルゴリズムを利用する TCP が混在する場合、TCP の公平性が失われるという問題がある。ここでの TCP の公平性とは、N 本の TCP コネクションが同一のボトルネックリンクを使用する際、各コネクションがボトルネックリンクの帯域の 1/N を利用することを意味する。基本仕様の TCP と上記のような広帯域高遅延用 TCP が同一のボトルネックリンクを利用する場合、同一の帯域を利用できず、TCP の公平性が失われる。そのため、昨今では TCP の公平性を考慮した広帯域高遅延用 TCP の研究 [19] [20] [21] も行われている。

第3章　問題意識

本章では、本研究の問題意識に関して述べる。まず、現状における新規 TCP の普及における問題点について述べる。次にその問題点に対して本論文が提案する解決策を述べ、最後に本研究の関連研究を述べる。

3.1 新規 TCP の普及における問題点

2.4 章で前述したように、無線ネットワークや広帯域高遅延ネットワーク等の特定環境において汎用的な TCP より高いパフォーマンスを発揮する新規 TCP がこれまで多く提案されている。しかし、現在これらの新規 TCP は一般に広く普及していない。その原因の一つが現状一般に広く普及する OS の TCP の利用形態にある。OS はトランスポート層プロトコルとして、TCP、UDP、SCTP [22] といった複数のプロトコルを所持できるが、TCP としては一つしか所持できない。一つの TCP で多様なネットワーク環境に対応する必要性から、汎用性を重視した TCP を OS は利用する必要がある。

LAN や専用回線を用いた通信のようにネットワークの帯域や遅延が特定できる場合や無線通信のみを利用するといった通信形態が特定できる場合、そのネットワーク環境に特化した TCP の利用が可能である。しかし 1.1 章で前述したように、インターネットを用いた通信の場合、通信相手が多様であり、ネットワークの輻輳状況も常に変化するため、遅延、帯域が多様化する。また、ノートパソコンの利用などにより有線通信と無線通信が併用されることも多い。このように利用するネットワーク環境が特定できない場合、多様なネットワーク環境への柔軟な対応が TCP には求められる。そのため、OS は汎用性を重視した TCP を利用する必要があり、特定環境に特化した TCP の利用は難しい。

3.2 解決策の提案

前節で述べた問題点を解決するため、本研究では TranSwitch を提案する。TranSwitch は OS に複数の TCP を導入し、これら複数の TCP からアプリケーションが利用する TCP をネットワークフロー毎に選択可能にする。OS へ複数の TCP を導入し、ネットワークフロー毎に異なる TCP を利用することで、ユーザは汎用性を重視した TCP のみを利用する必要がなくなり、それぞれのネットワークフローの特性に適した TCP の利用が可能となる。

3.3 関連研究

本節では本機構の関連研究として WAD [23]、X-kernel [24]、AS-TCP [25] を取り上げ、その概要を述べる。

3.3.1 WAD

ソケット通信ではソケットが持つバッファサイズを超える容量の送受信はできない。そのため高いスループットで通信を行う場合、ソケットの持つバッファサイズがボトルネックとなり、ウインドウサイズが増加しなくなる問題がある。この問題はソケット

バッファサイズをより大きな値に設定することにより解決できる。WAD (Work Around Daemon) はネットワークフロー毎にソケットバッファサイズ等のチューニングが可能な機構である。WAD を用いることにより、高いスループットで通信を行う必要のあるネットワークフローのソケットバッファサイズを増加させ、バッファサイズがボトルネックとなりウインドウサイズが向上しない問題を改善できる。図 3.1 に WAD の設定ファイルの例を示す。この設定では通信先の IP が 10.5.128.74 とする全てのソケットの送信バッファサイズを 3751239Byte に、受信バッファのサイズを 6571899 Byte に設定している。

```
[bob]
src_addr: 0.0.0.0
src_port: 0
dst_addr: 10.5.128.74
dst_port: 0
mode: 1
sndbuf: 3751239
rcvbuf: 6571899
wadai: 6
wadmd: .3
maxssth: 100
divide: 1
reorder: 9
delack: 0
```

図 3.1: WAD 定義ファイル

このように WAD はネットワークフロー毎のソケットバッファ等のチューニングが可能であるが、TranSwitch のような TCP の切替が不可能なため、2.3 章で前述した TCP が抱える問題点を解決できない。

また、WAD はネットワークフローとチューニングを関連付けた定義情報をユーザランドで動作するデーモンで管理し、TCP のフローが生成される毎にこの定義情報を参照する。そのためネットワークフローが生成される際、一度カーネルレイヤからユーザレイヤに処理を移し、定義情報を検索する必要があり、この処理がオーバヘッドとなる。

3.3.2 X-Kernel

X-Kernel は各プロトコルをモジュールとして管理し、これらを組み合わせることでプロトコルスタックが再編可能なマイクロカーネルである。しかし、X-Kernel はマイクロカーネルとして動作し、プロトコルをユーザランドで管理するため、現在一般に普及している OS とネットワークのアーキテクチャが大きく異なる。一方で TranSwitch は現在普及している OS の TCP/IP 実装において最小限の変更で TCP の切替を可能にする。このため TranSwitch 導入後も従来の OS のネットワークアーキテクチャに依存して動作する他の機構が利用可能であり、この点において X-Kernel より優れている。

3.3.3 AS-TCP

AS-TCP (Application Specific TCP) は TranSwitch と同様に OS に複数の TCP を導入し、アプリケーションが利用する TCP をエンドユーザが選択可能な機構である。AS-TCP における TCP の選択はアプリケーション毎に行い、TranSwitch のネットワークフロー毎の TCP の選択と異なる。同一のアプリケーションによる通信であっても通信ホストが異なる場合、通信経路が異なるため、輻輳、遅延、ボトルネックリンクといったネットワークの特性が異なる。そのためアプリケーションと TCP の関連付けではアプリケーションの通信の特性を考慮した TCP の選択は可能であるが、それぞれのネットワークフローの特性を考慮した TCP の選択は行えないという問題点がある。TranSwitch はネットワークフロー毎の TCP の選択が可能なため、アプリケーションの通信の特性だけでなく、ネットワークフローの特性を考慮した TCP の選択が可能という点において AS-TCP より優れている。

第4章　設計

本章では TranSwitch の設計について述べる。最初に TranSwitch の概要を述べ、次に TranSwitch のモジュール構成と各モジュールを説明する。

4.1 概要

本論文では TranSwitch を提案する。TranSwitch は OS に複数の TCP を導入し、これら複数の TCP からアプリケーションが利用する TCP をネットワークフロー毎に選択可能にする。また、エンドユーザによるユーザレイヤからの TCP の選択により、TranSwitch は通信環境の多様性や変化に柔軟かつ即時的な対応が可能である。

4.1.1 想定シナリオ

TranSwitch を用いて OS に複数の TCP を導入することで、汎用性を重視した TCP のみの利用が必要でなくなり、それぞれのネットワークフローの特性に適した TCP の利用が可能となる。これを図 4.1 に示す。

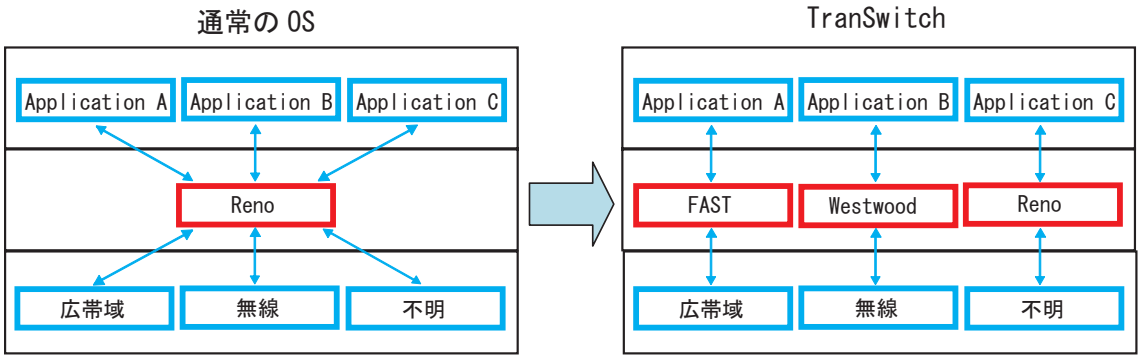


図 4.1: ネットワークフロー毎の異なる TCP の利用

図 4.1 は左に通常の OS を用いた場合、右に TranSwitch を用いた場合のアプリケーションからの TCP の利用を表している。この例では通常の OS は TCP を一つしか所持できないため、汎用性の高い TCP である Reno を TCP を利用する全てのアプリケーションで利用している。一方、右の TranSwitch を用いた場合では、OS は複数の TCP を所持できるため、Reno だけでなく広帯域ネットワークに適した TCP である FAST と無線通信に適した TCP である Westwood を所持している。アプリケーション A の通信は通信ホストとの広帯域な通信が可能であるため、広帯域な通信に適した TCP である FAST を利用し、アプリケーション B の通信は無線を用いた通信を行うため、無線通信に適した TCP である Westwood を利用し、アプリケーション C の通信はネットワークの特性が不明なため、従来と同様に汎用性の高い Reno を利用している。このように TranSwitch を用いてより適した TCP を各ネットワークフローで利用することにより、すべてのネットワークフローで汎用性を重視した TCP を利用する場合と比較してより効率の良い通信が可能である。

4.1.2 機能要件

TranSwitch は OS に複数の TCP を導入可能にし、エンドユーザはこれらの TCP のネットワークフロー毎の利用が可能となる。また、この選択はユーザレイヤからエンドユーザが設定可能である。よって、TranSwitch は以下の機能を持つ。

- 複数 TCP 混在機能
  TranSwitch は複数の TCP を導入可能に OS を拡張する。そのため、一つしか TCP を所持できない従来の OS におけるトランスポート層の設計を、複数の TCP を所持可能に変更する必要がある。

- TCP 追加および削除機能
  TranSwitch を用いて複数の TCP を利用するためにはユーザが TCP を OS に追加する機能が必要である。また、利用しない TCP を OS から削除する機能も必要である。追加される TCP は TranSwitch 上での利用を可能にするため、TranSwitch の定める実装形式で実装される必要がある。

- 定義情報設定ユーザインタフェース
  通信における遅延、利用可能な帯域は通信ホストや利用するネットワークによって異なりかつ動的に変化する。また通信形態も多様であり、TCP の通信の性質はアプリケーションによっても多様である。そのため、より適した TCP を選択するためには即時的かつ柔軟に TCP を選択できる必要がある。よって、各ネットワークフローが利用する TCP はエンドユーザによる選択が望ましい。これを実現するため、ネットワークフローと TCP の選択を関連付けた定義情報をユーザレイヤから設定するユーザインタフェースが必要である。

- TCP 切替機能
  TranSwitch は OS が所持する複数の TCP からネットワークフロー毎に異なる TCP が利用可能である。そのため、TCP を用いた通信が行われる際にネットワークフローの確立を検知し、ネットワークフローと TCP を予め関連付けた定義情報を参照し、それに沿って利用する TCP を切替える機能が必要である。

4.2 モジュール構成

本節では TranSwitch を構成するモジュールの詳細について述べる。TranSwitch のモジュール構成を図 4.2 に示す。

TCP を用いたデータの送信、受信があった場合、フロー生成検知モジュールが TCP を用いたネットワークフローの生成を検知する。ネットワークフローの生成が検知されると、TCP 切替モジュールがネットワークフローと TCP の関連付けを定義した定義情報を参照する。検知したネットワークフローが定義情報の定義に一致する場合、TCP 切替モジュールはそのネットワークフローの TCP を通常の TCP から定義情報

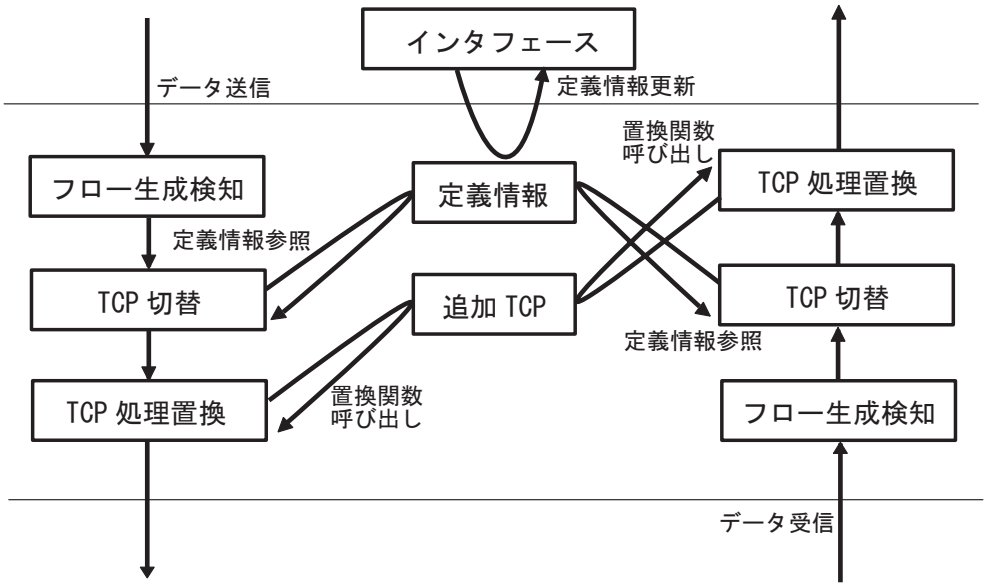


図 4.2: TranSwitch モジュール構成

で定義された追加 TCP へと切替える。その後、TCP の受信処理、送信処理が行われる際、切替先の TCP として定義された追加 TCP に処理が適時置換される。定義情報はエンドユーザがアプリケーションとして実装されたインターフェイスを用いて更新する。

フロー生成検知モジュール

フロー生成検知モジュールは TCP を利用する全てのネットワークフローの生成を検知する。ネットワークフローの検知後、検知したネットワークフローの情報を TCP 切替モジュールへ渡す。これにより、TCP を利用する全てのネットワークフローを定義情報から検索可能になる。

TCP 切替モジュール

TCP 切替モジュールは生成されたネットワークフローの情報をフロー生成検知モジュールから受け取り、この情報と一致する定義を定義情報モジュールを参照し、検索する。一致する定義がある場合、TCP 切替モジュールはネットワークフローが利用する TCP を通常の TCP から定義情報で定義された TCP に切替える。この切替えの情報はネットワークフローに TCP を関連付けることで表現し、TCP の処理が行われる際、TCP 処理置換モジュールによって参照される。これにより、ネットワークフロー毎に異なる TCP を関連付けることが可能となる。

TCP 処理置換モジュール

OS における TCP の実装は多数の関数によって成り立っている。従来の新規 TCP は改良部分のみを差分として実装し、それをカーネルパッチで配布するという形をとっている。よって、TranSwitch 上で動作するそれぞれの新規 TCP は TCP の処理すべてを実装する必要はなく、差分のみを実装すれば良い。そのため、TranSwitch による新規 TCP への切替えは基本仕様の TCP と新規 TCP の関数単位の置換という形を取る。TCP 処理置換モジュールは TCP 切替モジュールによって定義されたネットワークフローと TCP の関連付けに従って、この関数単位の置換を行う。基本仕様の TCP の主要な関数に処理が遷移する度に、TCP 処理置換モジュールはネットワークフローに関連付けられた TCP を参照し、対応する新規 TCP の関数に処理を移す。これにより、異なる TCP の同時利用が可能となる。

定義情報モジュール

定義情報モジュールはネットワークフローと TCP の関連付けを保持する。定義情報はユーザインタフェースから更新、参照され、また TCP 切替モジュールが定義情報を検索する際に参照される。

ユーザインタフェース

ユーザインタフェースはユーザランドにおけるアプリケーションとして動作し、エンドユーザによる定義情報モジュールへの参照、更新を可能にする。ユーザインタフェースを用いることで、エンドユーザは定義情報の作成と現在定義されている定義情報の参照と削除が可能となる。

第5章　実装

本章では TranSwitch のプロトタイプ実装について述べる。最初に今回のプロトタイプ実装における実装環境について述べ、その後 TranSwitch の各機能の実装について述べる。

5.1 実装環境

TranSwitch の実装環境を表 5.1 に示す。

表 5.1: 実装環境

項目	説明
OS	Fedora Core 1
カーネル	Linux 2.4.32
実装言語	C
コンパイラ	GCC 3.2.3

5.2 TranSwitch の起動および終了

TranSwitch は利用の選択が可能である。TranSwitch はトランスポート層の拡張をおこなうためカーネルレイヤで実装される。そのため、TranSwitch の起動は sysctl [26] を用いた OS のパラメータ設定という形で行う。これを図 5.1 に示す。図 5.1 の例 1 では

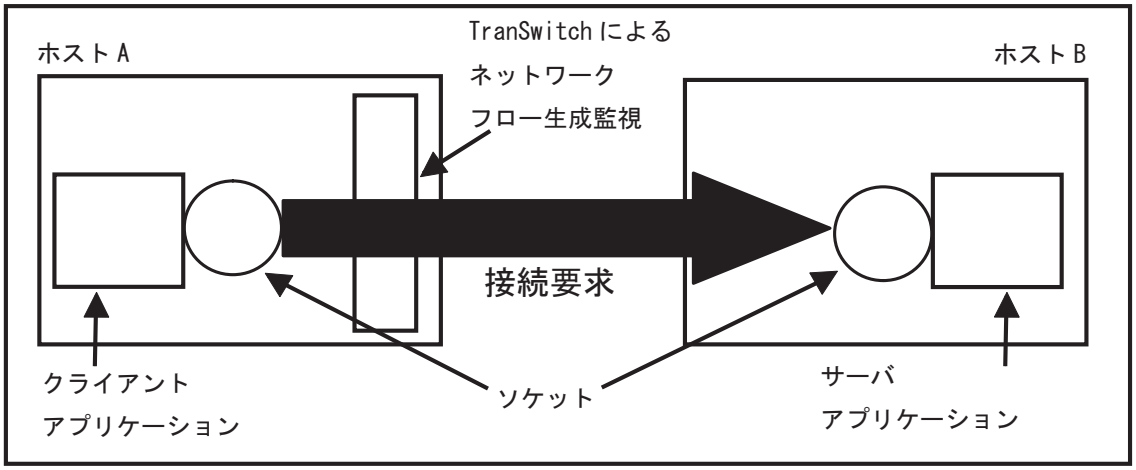
```
例 1.
% sysctl net.core.transwitch=1

例 2.
% sysctl net.core.transwitch=0
```

図 5.1: sysctl を用いた TranSwitch の起動と停止

TranSwitch を起動させるために sysctl の該当するパラメータである net.core.transwitch の値を 1 に設定している。また、図 5.1 の例 2 では TranSwitch を停止させるために net.core.transwitch の値を 0 に設定している。

TranSwitch の起動により、ネットワークフロー生成の監視が開始される。これを図 5.2 に示す。図 5.2 の二つの例はホスト A のクライアントアプリケーションによるホスト B のサーバアプリケーションへのソケットを用いた接続要求を表している。例 1 ではホスト A で TranSwitch が起動しており、ホスト A のクライアントアプリケーションがホスト B のサーバアプリケーションに対して接続要求を行う際、TranSwitch によってネットワークフローの生成が検知されている。例 2 ではホスト B 上で TranSwitch が起動しており、ホスト B のサーバアプリケーションに対してホスト A のクライアントアプリケーションが接続要求を行った際、同様に TranSwitch によってネットワー

例 1.



例 2.

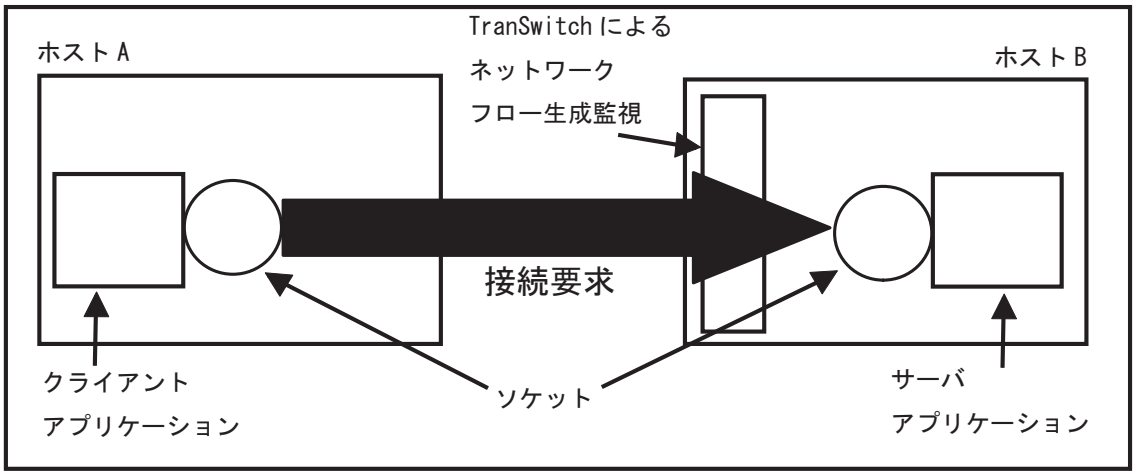


図 5.2: ネットワークフロー生成の検知

クフローの生成が検知されている。TranSwitch はこのようにして TCP を用いるネットワークフローの生成を検知した後、5.6 章で述べる TCP の切替処理に移る。

5.3 定義情報の入力

TCP を用いるネットワークフローの生成を検知した場合、TranSwitch は定義情報を参照する。定義情報はネットワークフローを表す送受信ホストの IP アドレス、ポート番号と利用する TCP の名前によって構成される。即時的かつ柔軟な TCP の選択を可能にするため、TranSwitch ではユーザランドからエンドユーザがコマンドを用いた定義情報の追加、削除が可能である。これを図 5.3 に示す。図 5.3 は TranSwitch の定義情報を追加するコマンドである trswadd コマンドを用いた 2 つの例と既に追加してある定義情報を削除するコマンドである trswdel コマンドを用いた例を表している。

例 1 は受信ホストの IP アドレスが 192.168.1.3、受信ホストのポート番号が 100 から 200、送信ホストの IP アドレスが 192.168.1.5、送信ホストのポート番号が 20 である全てのネットワークフローの TCP として FAST を利用し、この定義の ID を 10 とする設定の例である。例 2 は送信ホストの IP アドレスを 127.0.0.1 とする全てのネットワークフローの TCP として Westwood を利用するという設定の例である。例 2 の場合、送信ホストの IP アドレス、送信ホストおよび受信ホストのポート番号、定義情報の ID が入力されていない。ネットワークフローを示す値で特に定義されていない項目は全ての値に対応することを意味する。また、定義情報の ID は入力されていない場合、既存の定義情報に利用されてない値の最小値が自動で割り当てられる。そして例 3 では、事前に追加されている定義を trswdel コマンドで ID を指定して削除している。最後の例 4 では、trswlist コマンドを用いてこれまで定義された定義情報のリストを一覧表示している。これらの定義情報は 5.6 章で解説する TCP 切替時に参照さ

```
例1.
% trswadd -i 10 -si 192.168.1.5 -di 192.168.1.3 -sp 20 -dp 100-200 fast

例2.
% trswadd -si 127.0.0.1 westwood

例3.
% trswdel 10

例4.
% trswlist
id      src IP        dst IP     src port     dst port       TCP
1       127.0.0.1     0.0.0.0       0            0         westwood
```

図 5.3: 定義情報の入力例

れる。参照時のオーバヘッドを最小限に抑えるため、この定義情報はカーネルレイヤに保存される。

5.4 TCP の追加および削除

複数の TCP を OS が所持し、これらの TCP からネットワークフロー毎に適した TCP を選択可能にするには、OS が初期状態で持つ TCP に加えて新たな TCP を追加する機能が必要である。また一度追加した TCP を OS から削除する機能も重要である。この機能を実現するため、TranSwitch では各 TCP をカーネルモジュールとして実装し、そのカーネルモジュールをカーネルへ組み込むことでその TCP を OS に追加し、同様にカーネルモジュールを取り除くことでその TCP を OS から削除する形式を用いている。従来の新規 TCP の配布形態はカーネルパッチを用いてカーネルの

ソースを上書きする方法が一般的であった。しかしカーネルモジュールを利用することで、従来のカーネルパッチの利用と比較し、カーネルコンパイルが不要なため即時的な TCP の追加、削除が可能であり、図 5.4 に示すような簡易なコマンドの利用により、エンドユーザによる容易な TCP の追加および削除が可能である。図 5.4 では、ま

```
例 1.
% insmod TRSW_TestTCP.mod

例 2.
% lsmod | grep TRSW_
TS_TestTCP        5240        0

例 3.
% rmmod TRSW_TestTCP.mod
```

図 5.4: カーネルモジュールを用いた TCP の追加と削除

ず例 1 でカーネルモジュールを組み込むコマンドである insmod を用いて、Test TCP をカーネルモジュールとして実装した TRSW_TestTCP を OS に追加している。これにより OS に Test TCP が追加される。カーネルモジュールを用いるため、例 2 で示すように OS に組み込まれているカーネルモジュールを一覧できる lsmod の利用により、現在 OS に導入された TCP を閲覧可能である。また例 3 に示すように、カーネルモジュールを OS から削除するコマンドである rmmod を用いて、Test TCP を OS から削除している。このように追加する TCP を一つのカーネルモジュールとして利用することで、容易かつ即時的に TCP の追加と削除が可能である。

このようなカーネルモジュール化した TCP を TranSwitch 上で利用可能にするため、次節で述べる TranSwitch が定める TCP の実装形式に準拠する必要がある。

5.5 TranSwitch が定める TCP の実装形式

前節で述べたように TranSwitch 上で動作する TCP はカーネルモジュールとして実装される。カーネルモジュールは TranSwitch が TCP として利用するために TranSwitch が定める TCP の実装形式に準拠する必要がある。TranSwitch が定める TCP の実装形式は以下の通りである。

- TCP 名の表記
  TranSwitch はエンドユーザが定義した定義情報に従って TCP を切替える。この際、TCP を特定するために一意な TCP の識別子が必要となる。そのため TCP

```
static struct trsw_regist test_tcp = {
        .name ="test_tcp"
};
```

図 5.5: TranSwitch への TCP 名の登録

はカーネルモジュール内でこれを明記する必要がある。これを図 5.5 に示す。図 5.5 では trsw_regist 構造体のメンバ変数 name に test_tcp を代入している。これによりこの TCP の名前が test_tcp であると識別され、TranSwitch が管理する TCP のリストに test_tcp として登録される。

- 置換関数の実装
  OS における TCP の実装は多数の関数によって成り立っている。従来の新規 TCP は部分的に実装を置換し、それをカーネルパッチで配布するという形をとってきた。よって、TranSwitch 上で動作するそれぞれの新規 TCP は TCP の処理すべてを実装する必要はなく、差分のみを実装すれば良い。そのため、TranSwitch が定める新規 TCP の実装方法は関数単位の置換という形を取る。これを図 5.6 に示す。図 5.6 は TCP の処理の一部として tcp_transmit_skb 関

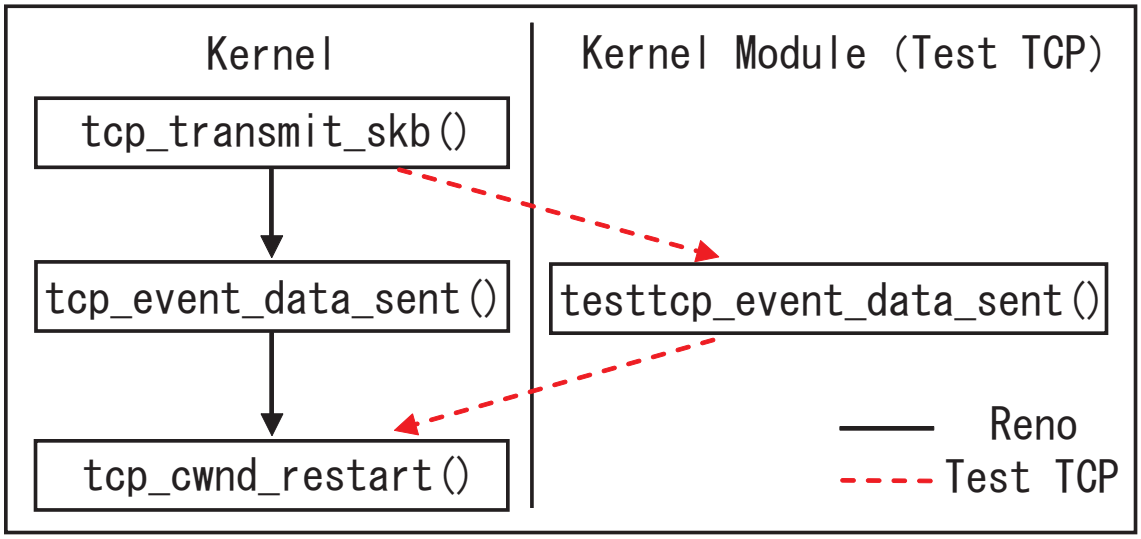


図 5.6: 置換関数を用いた TCP 処理

数、tcp_event_data_sent 関数、tcp_cwnd_restart 関数の順の関数呼び出しを抜粋している。中央の線から左側半分がカーネルで実装される OS が本来持っている TCP の実装を表現し、右側半分は TranSwitch を用いて OS に追加された Test TCP の実装を表現している。Test TCP の実装は tcp_event_data_sent 関数を testtcp_event_data_sent 関数に拡張している。OS が従来持っている TCP である

Reno は tcp_transmit_skb 関数、tcp_event_data_sent 関数、tcp_cwnd_restart 関数の順で関数呼び出しを行っている。それに対して、Test TCP は tcp_transmit_skb 関数から tcp_event_data_sent 関数ではなく、Test TCP のカーネルモジュールに実装されている独自の testtcp_event_data_sent 関数を呼んでいることがわかる。そして testtcp_event_data_sent 関数の処理後、Reno と同様に tcp_cwnd_restart 関数を呼び出している。このような差分のある関数のみを置換して処理することで、最小限の実装で TCP の追加が可能である。

- 置換対応の表記
  前項で述べた置換関数を適切に呼び出すために、OS が持つ従来の TCP の関数と置換関数の対応を表現する必要がある。この例を図 5.7 に示す。この例では

```
static struct trsw_regist test_tcp = {
      .tcp_event_data_sent = testtcp_event_data_sent,
      .name ="test_tcp"
};
```

図 5.7: カーネルモジュールにおける置換対応の表記

tcp_event_data_sent 関数と testtcp_event_data_sent 関数が対応していることがわかる。カーネルモジュール内で置換関数として表記されていない関数は置換が行われず、図 5.7 にあった tcp_transmit_skb 関数や tcp_cwnd_restart 関数のように OS が従来 TCP の処理に利用する関数を呼び出す。このようにカーネルモジュールによって置換可能な関数は、カーネル内の TCP の実装で用いられる全ての関数ではなく、TranSwitch で置換可能と定義される関数に限られる。

これは呼び出し側の関数を置換することで、その関数によって呼び出される関数を結果に置換が可能であり、また TCP の処理を担う関数が TCP のアルゴリズムを実装した関数や、実装の便宜上関数として実装された関数などがあり、関数によって TCP の改良における重要度が異なるためである。TranSwitch による TCP の切替は 5.6 章で後述するように、置換可能な関数呼び出し時にソケットに登録されている TCP を参照することで実現するため、置換可能な関数が多いほど切替えのオーバヘッドが高くなる。よって、TranSwitch の設計では置換可能な関数は一部の関数に限られる。TranSwitch のプロトタイプ実装において置換可能な関数として用意した全ての関数を図 5.8 に示す。

- TranSwitch への登録関数および削除関数の実装
  カーネルモジュールは insmod を用いてカーネルに組み込まれる際、また、rmmod によってカーネルモジュールが OS から削除される際、カーネルモジュールの初期化処理、終了処理としてカーネルモジュール内で定義する関数をそれぞれ呼び出すことができる。このカーネルモジュールの初期化処理と終了処理において、

TranSwitch が TCP を利用可能にするために TranSwitch への登録と削除を行う。これを図 5.9 に示す。図 5.9 の例 1 はカーネルモジュールの初期化関数である init_module 関数で TranSwitch の trsw_regist 関数を呼び出している。trsw_regist 関数に図 5.5 で示した trsw_regist 構造体のアドレスを渡すことで TranSwitch へ test TCP を登録が可能である。また、例 2 でカーネルモジュールの終了関数である cleanup_module 関数で TranSwitch の trsw_unregist 関数を呼び出している。trsw_unregist 関数に trsw_regist 構造体のアドレスを渡すことで TranSwitch から test TCP の削除が可能である。

```
■tcp_output.c
tcp_cwnd_restart
tcp_select_window
tcp_send_skb
tcp_fragment
tcp_simple_retransmit
tcp_retransmit_skb
tcp_connect
tcp_send_delayed_ack
tcp_send_ack

■tcp_output.c
tcp_rtt_estimator
tcp_grow_window
tcp_set_rto
tcp_init_cwnd
tcp_time_to_recover
tcp_moderate_cwnd
tcp_cwnd_down
tcp_fastretrans_alert
tcp_ack_update_rtt
tcp_ack_update_window
tcp_cwnd_application_limited
tcp_parse_options
tcp_disordered_ack
tcp_fin

■tcp_ipv4.c
do_pmtu_discovery
tcp_v4_err
```

図 5.8: TranSwitch における置換可能関数一覧

```
例1.
int init_module(void)
{
        trsw_regist(&test_tcp);
        return 0;
}

例2.
void cleanup_module(void)
{
        trsw_unregist(&test_tcp);
}
```

図 5.9: TranSwitch への TCP の追加と削除

5.6 TCP 切替処理

TCP を用いるネットワークフローが生成された場合、TranSwitch はこれを検知し、定義情報を参照する。生成を検知したネットワークフローと合致するネットワークフローが定義情報にある場合、選択された TCP と生成されたネットワークフローを関連付けるため、そのネットワークフローのソケットに TCP を登録する。これを図 5.10 に示す。図 5.10 ではソケットそれぞれに TCP が関連付けられていることを示している。ソケット A には Reno が、ソケット B には Test TCP がそれぞれ関連付けられている。tcp_transmit_skb 関数から tcp_event_data_sent 関数を呼び出す際、ソケットが持つ TCP への参照に従って次の関数を呼び出している。ソケット A は Reno への参照をもつため、Reno を実装しているカーネルの tcp_event_data_sent 関数を呼び出す。一方、ソケット B は TCP への参照として Test TCP への参照を持つため、カーネルモジュール内にある testtcp_event_data_sent 関数を呼び出す。5.5 章で前述したように、このようにソケットが持つ TCP への参照は TranSwitch が置換可能と定義する全ての関数で行われる。TranSwitch はソケットが持つ TCP への参照を利用することで、カーネルモジュール内で実装された新規 TCP の関数群へ適切な処理の切替えが可能である。

TCP の切替における一連の処理を図 5.11 に示す。まず、TranSwitch は TCP を用いるネットワークフローの生成を検知し、定義情報を参照する。生成を検知したネットワークフローと合致するネットワークフローが定義情報にある場合、その定義で関連付けられている TCP をそのネットワークフローの TCP としてソケットに関連付ける。そして 5.5 章で前述したように TCP の処理中で置換関数として登録された関数の処理を行う際は、ソケットに登録された TCP を実装したカーネルモジュール内の関数に処理を移す。以上の一連の処理により、TranSwitch は TCP の切替を実現する。

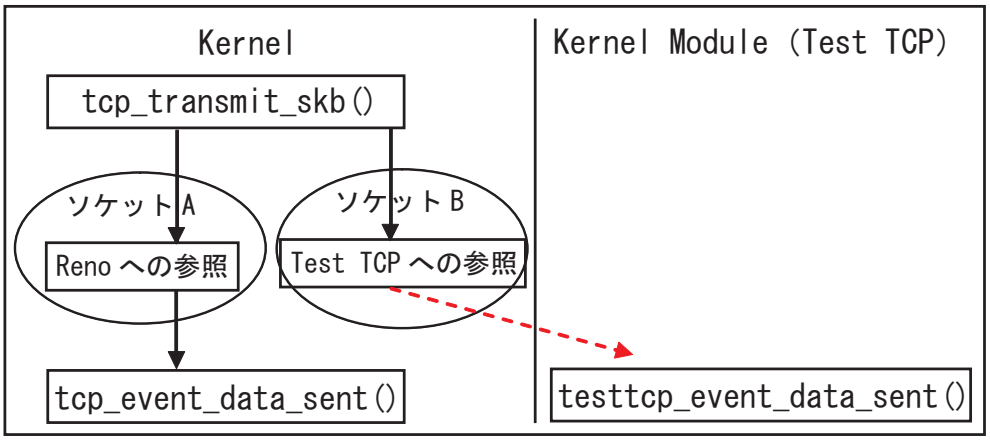


図 5.10: ソケットと TCP の関連付け

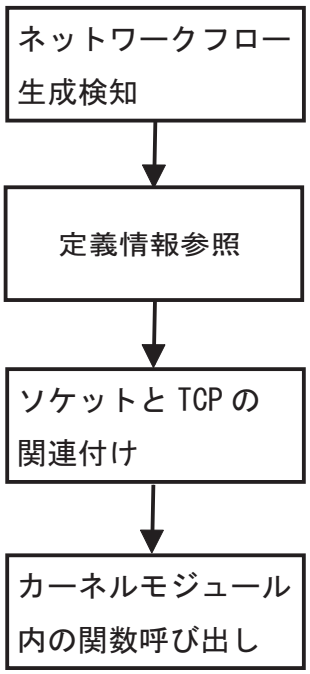


図 5.11: TranSwitch による TCP 切替までの動作

第6章　評価

本章では TranSwitch のプロトタイプ実装の評価について述べる。まず最初に TranSwitch の定性的評価について述べ、次に TranSwitch の定量的評価について述べる。

6.1 定性的評価

本節では TranSwitch の定性的評価として関連研究である WAD、X-Kernel との比較を以下の項目で行った。

- OS のアーキテクチャ変更
  各機構の導入によって生じる OS のアーキテクチャの変更の程度を評価する。
- TCP の変更可能範囲
  各機構によって可能となる TCP の機能の変更範囲を評価する。

表 6.1: 関連研究との比較評価

	TranSwitch	WAD	X-kernel	AS-TCP
OS のアーキテクチャ変更	○	○	×	○
TCP の変更範囲	○	×	○	○

比較評価した結果を表6.1 に示す。OS のアーキテクチャ変更に関してはX-kernel がマイクロカーネルアーキテクチャを導入することでネットワークのアーキテクチャを大きく変更してしまうのに対して、TranSwitch、WAD、AS-TCP はネットワークのアーキテクチャの変更を必要最小限に止め、OS のネットワークアーキテクチャの大枠を変更しない。TCP の変更範囲において、WAD はソケットバッファ等の一部しか変更できないのに対し、TranSwitch、X-Kernel、AS-TCP がほぼすべてにおいて変更できる。TranSwitch と AS-TCP は OS のアーキテクチャ変更、TCP の変更可能範囲の 2 点において同様の評価であるが、3.3.3 章で前述したように AS-TCP がアプリケーション毎に TCP を切替えるのに対し、TranSwitch はネットワークフロー毎に TCP を切替可能である。そのため、AS-TCP における TCP の選択がアプリケーションの通信の特性しか考慮できないのに対し、TranSwitch はアプリケーションの通信の特性に加え、ネットワークフローの特性を考慮した TCP の選択が可能という点において TranSwitch の方が優れている。

6.2 定量的評価

TranSwitch のプロトタイプ実装の定量的評価として、TranSwitch の利用によるオーバヘッドの計測を行った。本節では評価に先立って行った予備実験に関して最初に説明し、次に本評価の評価環境について述べ、最後に TranSwitch による切替のオーバヘッドをスループットで示す。

6.2.1 予備実験

プロトタイプ実装の評価を行うにあたり、まず最初に予備実験を行った。今回の評価では TranSwitch を利用した場合のオーバヘッドをスループットで計測する。しかし、評価に用いる評価機器が TCP に対するボトルネックとなった場合、TranSwitch によるオーバヘッドがそのボトルネックに吸収されるため、正確なオーバヘッドの計測ができない。評価端末の CPU、メモリの速度やバスの幅等のハードウェア的制限によって、TCP のスループットが仮に 100 Mbps に制限されてしまう場合、その TCP を 200 Mbps のスループットで動作可能にした実装と 150 Mbps のスループットでしか動作しない実装の両方が 100 Mbps のスループットで動作する。そのため、それぞれの TCP の実装をスループットで正確に評価できない。このような TCP の処理以外におけるボトルネックの発生を防ぐため、予備実験として本評価で利用する評価端末のボトルネックを調査した。

表 6.2: 評価端末

項目	説明
PC	IBM Thinkpad t42p
OS	Fedora Core 1
カーネル	Linux 2.4.32
CPU	Intel Pentium M 2GHz
NIC	1 Gigabit Ethernet
RAM	PC2700 2GByte

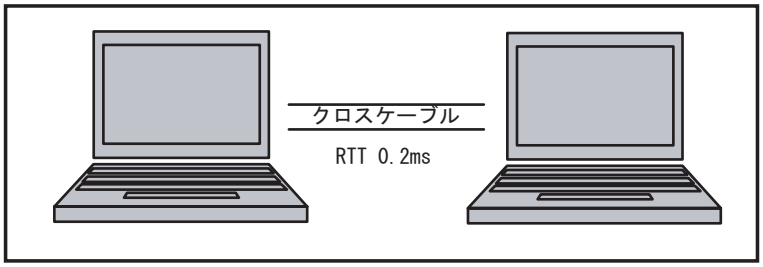


図 6.1: 予備実験ネットワーク構成

表 6.2 に示す評価端末 2 台を図 6.1 に示すように、評価端末 2 台をクロスケーブルで直接接続し、UDP、TCP のスループットを iperf 2.0.2 [27] を用いてそれぞれ 30 回計測した。なお、この環境における RTT は 0.2 ms である。その結果を表 6.3 に示す。表 6.3 で示すように、輻輳制御を行わない UDP の平均スループットが 1 Gigabit Ethernet の NIC を利用しているにも関わらず、664.6 Mbps と 1 Gbps に届かない値になっている。また、TCP のスループットも UDP と同様に 1 Gbps に届かず、ほぼ UDP と同じ値になっている。これらのことから、この予備実験では評価端末のハード

ウェア性能が UDP、TCP のボトルネックになっていると考えられる。よってこの評価機器を用いた TranSwitch の評価では、TranSwitch の利用によるオーバヘッドが評価端末のハードウェア性能によるボトルネックに吸収され、正確な評価が不可能である。これを解決するため、本評価では時節で述べる評価環境を用いて TranSwitch の評価を行う。

表 6.3: 実験環境における UDP、TCP スループット計測

プロトコル	平均スループット	標準偏差
UDP	664.6	23.8
TCP	647.1	6.57

6.2.2 評価環境

評価機器

5.2.1 章の予備実験で用いた評価機器を同様に用いる。

ネットワーク

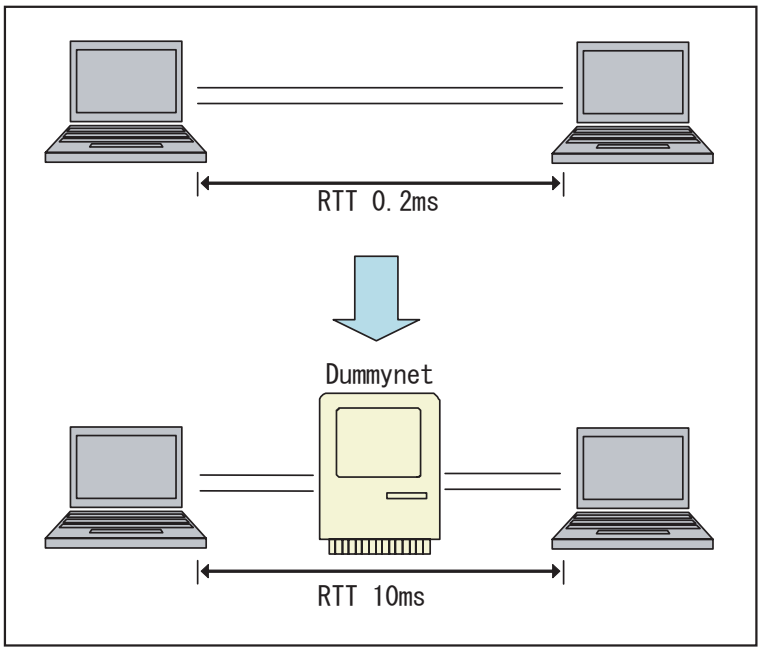


図 6.2: Dummynet を用いた遅延のコントロール

前節で述べたように、図 6.1 で示すネットワークでは本評価で用いる評価端末のハードウェア性能が TCP のスループットに対するボトルネックとなり、TranSwitch のオーバヘッドによる TCP のスループットの違いを吸収してしまう。これを防ぐため、本評価では図 6.2 に示すようにネットワークエミュレータとして Dummynet [28] を導入し、RTT の値を高めることで TCP のスループットをハードウェア性能によるボトルネックの値よりも低い値に抑える。これにより、TranSwitch のオーバヘッドによるスループットの低下を正確に把握できる。本評価では RTT を 10 ms に設定して評価を行う。図 6.2 は 2 台の評価端末の間に Dummynet を導入し、RTT を導入以前の 0.2 ms から 10 ms に増加させたことを表している。次項で本評価における Dummynet について説明する。

Dummynet

本評価で用いる Dummynet は IP 単位でネットワークのパケットロス率、ボトルネック、遅延を自由に設定可能な FreeBSD 上で利用できるネットワークエミュレータである。図 6.3 に示すように Dummynet を用いることで、論理的にインターネットと同じネットワーク環境を構築できる。

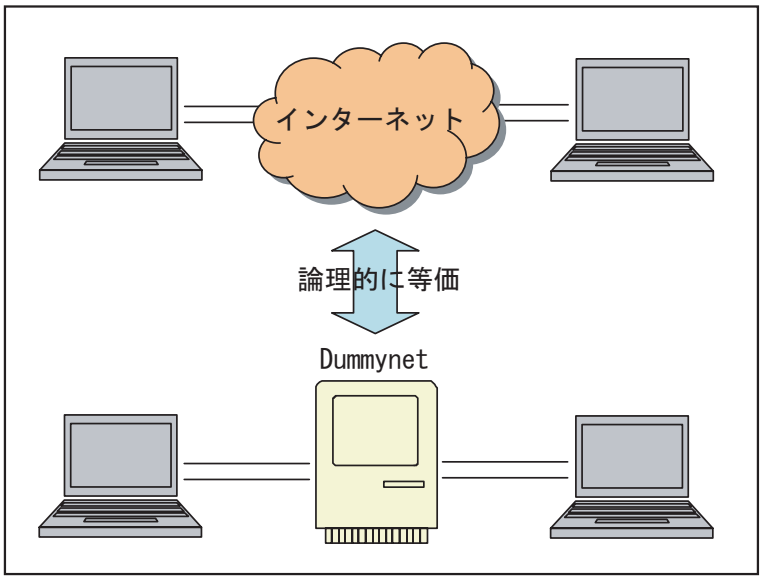


図 6.3: Dummynet によるネットワークエミュレート

本評価では Dummynet による遅延制御の精度を上げるため、カーネルの割り込みの精度を従来の 10 ms から 0.5 ms に上げている。これにより RTT は表 5.4 に示すように、設定した RTT である 10 ms に対してより収束しており、RTT の変動によるスループットの変動を最小限に抑えている。表 6.4 は割り込み精度が 5 ms と 0.5 ms それぞれの場合における RTT の平均値、最大値、最小値、標準偏差を表している。RTT は Ping を用いてそれぞれ 30 回計測し、Dummynet によって 10 ms に設定されてい

る。割り込み精度が 0.5 ms の場合は RTT の平均値が 10.18、標準偏差が 0.27 であるのに対し、割り込み精度が 5 ms の場合は RTT の平均値が 7.94、標準偏差が 1.46 と RTT の精度が大きく異なることがわかる。本評価ではより正確な TCP のスループットを計測するため、カーネルの割り込み精度を 0.5 ms に設定する。

表 6.4: Dummynet における RTT の精度 (RTT=10ms)

割り込み精度	最大値	最小値	平均値	標準偏差
5ms	10.2	5.53	7.94	1.46
0.5ms	11	9.87	10.18	0.27

6.2.3 スループット計測

TranSwitch はネットワークフロー毎に利用する TCP の選択が可能である。この機能は 4.2.4 章で前述したように、TranSwitch が関数を置換することで実現される。この際、被置換関数から置換関数へ処理を切替えることでオーバヘッドが生じる。本評価では TranSwitch のオーバーヘッドとして、この関数の置換によって生じるスループットへの影響を前節で述べたネットワーク環境において評価した。TranSwitch のオーバヘッド計測の際、TCP のアルゴリズムの違いによるスループットの差が生じることを防ぐため、本評価の評価端末が当初からカーネル内で実装している TCP である Reno を TranSwitch が切替える TCP として利用した。よって、TranSwitch による TCP の切替えはカーネル内で実装された Reno から TranSwitch における実装形式の Reno へ行われる。これを図 6.4 に示す。図 6.4 では実線がカーネル内で実装された Reno の処理を示し、点線がカーネルモジュールとして実装され TranSwitch によって利用される Reno の処理を示す。本評価では 4.12 で示した全ての置換関数をカーネルの実装と全く同じ実装でカーネルモジュール内で実装した。よって、被置換関数と置換関数は関数内の実装がまったく同じである。このときのスループットを計測することで、TranSwitch による被置換関数から置換関数への切替処理のオーバヘッドを計測可能である。

二つの環境それぞれにおける TCP のスループットを iperf 2.0.2 を用いて 30 回ずつ計測し、この評価結果を表 6.5 に示す。

表 6.5: TranSwitch オーバヘッド計測

Reno 利用形態	平均スループット	標準偏差
カーネル実装	39.96Mbps	0.08
TranSwitch による LKM への切替	39.01Mbps	0.07

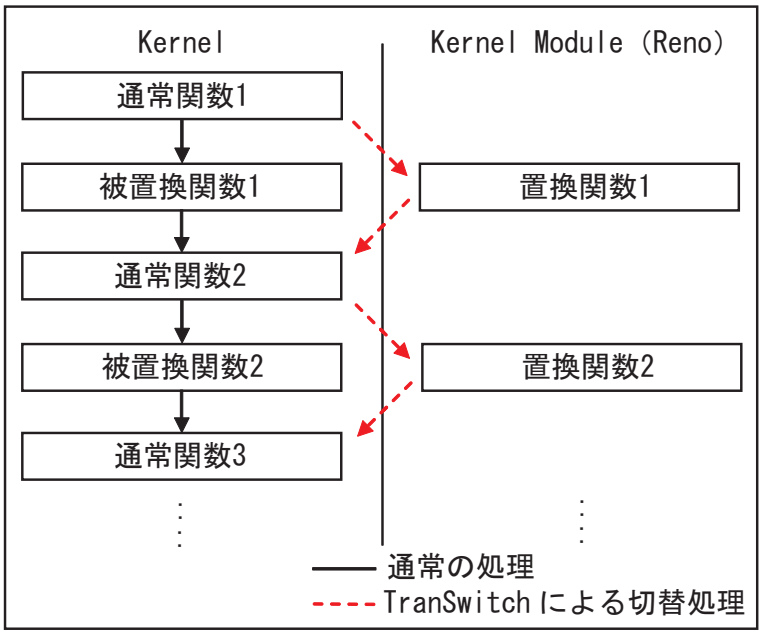


図 6.4: TranSwitch による Reno

カーネル内で実装された Reno の平均スループットが 39.96 Mbps、標準偏差が 0.08 であり、カーネルモジュールとして実装され、TranSwitch によって呼び出される Reno の平均スループットが 39.01 Mbps、標準偏差が 0.07 である。平均スループットの値の差は 0.85 Mbps であり、TranSwitch の切替によりスループットは 2.1 % 減少している。このように TranSwitch の利用によりオーバヘッドが発生し、スループットが 2.1 %減少するが、新規 TCP の多くが今回計測したオーバヘッド以上の改善を見せる。本評価により、TranSwitch の TCP 切替によってオーバヘッドは生じるものの、TCP の切替えによるスループットの向上において十分に有効であると示すことができた。

第7章　結論

本章では結論として、本論文のまとめと今後の課題について述べる。

7.1 まとめ

本論文ではネットワークフロー毎により最適な TCP を利用可能とする機構として TranSwitch を提案した。昨今のネットワーク環境は通信速度や通信形態の面において多様化しており、これを背景として、広帯域ネットワーク用 TCP や無線ネットワーク用 TCP などの特定のネットワーク環境に最適化した TCP がこれまで多く提案されている。しかし、従来の OS は TCP を一つしか所持できず、また利用するネットワーク環境が特定できないことが多いため、汎用性の高い TCP を利用する必要があり、これら特定環境での利用を想定した TCP を従来の OS で利用するのは難しい。この問題点の解決策として、複数の TCP を所持可能に OS を拡張し、ネットワークフロー毎に利用する TCP をエンドユーザが選択可能にする機構として TranSwitch を提案した。本論文ではこの TranSwitch の設計・実装について述べ、プロトタイプ実装の評価結果を用いて TranSwitch の有効性を示した。

7.2 今後の課題

今後の課題として以下の項目を挙げる。

- 通信途中における TCP の動的切替
  現在の TranSwitch の設計ではネットワークフローが生成される際、エンドユーザによって予め定義されている TCP がネットワークフローに関連付けられる。ネットワークフローの生成以降はネットワークフローと TCP の関連付けは変更されない。これはネットワークフローが TCP を用いて通信を行っている最中に関連付けを変更した場合、TCP の処理における一貫性が失わるためである。

  しかし、輻輳、遅延、通信形態といったネットワーク環境やアプリケーションの特性等の TCP の選択で考慮される全ての要因を通信が開始される前の段階で把握できるとは限らず、通信が開始された後により適した TCP の存在に気づく場合がある。また、インターネットのような公衆回線の輻輳や遅延は通信の最中で変化するため、当初適切として選択した TCP が通信の最中で不適切な TCP となる可能性がある。これらの問題は通信の最中にネットワークフローが利用する TCP を変更することで解決できる。このような通信途中における TCP の動的切替を実現することで TranSwitch の有効性を向上できる。

- 定義情報の定義における利便性向上
  TranSwitch の実装ではネットワークフローと TCP の関連付けをネットワークフローを表現する送受信ホストの IP アドレス、ポート番号と TCP 識別子で行っている。しかし、これ以外の定義方法がより豊富であるほうが定義を行いやすい。例えば、無線ネットワーク用 TCP を無線通信を行うすべてのネットワークフローで利用する場合、IP アドレスやポート番号ではなく無線 NIC のネットワークインタフェース名と TCP 識別子を用いてネットワークフローと TCP の

関連付けを行う方が容易である。このようにより多くの定義方法を用いた定義情報の定義を可能にすることで、容易な TranSwitch の利用が可能となる。

- TCP 選択補助機構
  TranSwitch では TCP とネットワークフローの関連付けの定義をエンドユーザが行う。これは TCP を選択する上で要因となる輻輳、遅延、通信形態といったネットワーク環境やアプリケーションの特性等の情報を最も詳細に把握可能であるのがエンドユーザだからである。しかし、エンドユーザはネットワークに関する知識が豊富であるとは限らず、適切でない TCP の選択を行う可能性がある。このように不適切な TCP によって得られるスループットは TranSwitch を利用せずに汎用性を重視した TCP を利用した場合に得られるスループットに比べ低くなるという問題点がある。この問題は TranSwitch がエンドユーザに代わって TCP の選択を行うなど、TranSwitch がユーザの TCP の選択を補助する機能を持つことで解決できる。このような TCP 選択補助機構により TranSwitch の有効性を向上できる。

謝辞

本研究を進めるにあたり、御指導を頂きました、慶應義塾大学環境情報学部教授徳田英幸博士に深く感謝致します。また、貴重な御助言を頂きました間 博人氏、斉藤 匡人氏、高橋 ひとみ氏をはじめとする慶應義塾大学徳田研究室 ECN 研究グループの諸氏、および徳田・村井・楠本・中村・高汐・湧川研究室の方々には、研究会の活動を通じて多くの御意見、助言を頂きましたこと拝謝します。そして研究の日々を共に過ごした青柳 禎矩氏、金田 裕剛氏、その他多くの友人に感謝します。最後に、遠く陰ながら私を支えてくれた家族に深い感謝と敬愛の意を表し、謝辞と致します。

2005 年 12 月 28 日
山下 勝司

参照論文

- 山下 勝司, 高橋 ひとみ, 斉藤　匡人, 徳田　英幸.
  “TranSwitch:トランスポート層におけるプロトコル切替機構” 日本ソフトウェア科学会 第 4 回 SPA サマーワークショップ,
  Aug. 2005.

- 山下 勝司, 高橋 ひとみ, 斉藤　匡人, 徳田　英幸.
  “TranSwitch:ネットワークフロー毎における最適な TCP への動的切替機構” 情報処理学会 第 10 回ユビキタスコンピューティング研究会,
  Feb. 2006.

参考文献

[1] IEEE Standards for Local Area Networks: *Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications*, 1985.

[2] IEEE 802.11 Standard (IEEE Computer Society LAN WAN Standards Committee). *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications,* 1999.

[3] 総務省編 情報通信白書 (平成 17 年度版). http://www.johotsusintokei.soumu.go.jp/whitepaper/ja/h17/.

[4] Jon Postel. Transmission Control Protocol, RFC 793, August 1981.

[5] J. D. Day and H. Zimmermann. The OSI Reference Model. In *Proceedings of the IEEE*, volume 71, pages 1334-1340. IEEE, Dec. 1983.

[6] Jacobson, V. Congestion Control Avoidance and Control. *Computer Communication Review*, vol. 18, no. 4, pp. 314-329, August 1988.

[7] Jon Postel. Internet Protocol. RFC 791, September 1981.

[8] Jon Postel. User Datagram Protocol. RFC 768, August 1980.

[9] Dah-Ming Chiu and Raj Jain. Analysis of the increase and decrease algorithms for congestion avoidance in computer networks. *Computer Networks and ISDN Systems,* 17, June 1989.

[10] Jacobson, V. Modified TCP Congestion Avoidance Algorithm. end2end-interest mailing list, April 1990.

[11] R. M. Metcalfe and D. R. Boggs. Ethernet: Distributed Packet Switching for Local Computer Networks. *Communications of the ACM* , vol. 19, no. 7, July 1976.

[12] L. S. Brakmo, S. W. O 'Malley, and L. L. Peterson. TCP Vegas: New Techniques for Congestion Avoidance. *Proceedings of ACM SIGCOMM' 94,* pp. 24-35, London, U. K., October 1994.

[13] S. Morris and C. Casetti. TCP Westwood: Bandwidth Estimation for Enhanced Transport over Wireless Links. In *MobiCom 2001: The Seventh Annual International Conference on Mobile Computing and Networking,* Rome, Italy, July 2001.

[14] K. Brown and S. Singh, M-TCP: TCP for mobile cellular networks. ACM Comp. Comm. Review, vol. 27, no. 5, pp. 19?43, 1997.

[15] Zygmunt J. Haas. Mobile-TCP: An Asymmetric Transport Protocol Design for Mobile Systems. 3rd InternationalWorkshop on Mobile Multimedia Communications, September 1995.

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

[17] T. Kelly, Scalable tcp: Improving performance in highspeed wide area networks. In *Proceedings of the First International Workshop on Protocols for Fast Long-Distance Networks (PFLDnet)*, Feb. 2003.

[18] C. Jin, D. X. Wei, and S. H. Low. FAST TCP: motivation, architecture, algorithms, performance. *IEEE Infocom*, March 2004

[19] L. Xu, K. Harfoush, and I. Rhee. Binary Increase Congestion Control for Fast, Long Distance Networks. *IEEE INFOCOM*, March 2004.

[20] Injong Rhee, and Lisong Xu. CUBIC: A New TCP-Friendly High-Speed TCP Variant. *Third International Workshop on Protocols for Fast Long-Distance Networks*, Feb 3,4 2005

[21] T. Hatano, M. Fukuhara, H. Shigeno, and K. Okada. TCP-friendly SQRT TCP for High Speed Networks. In *Proceedings of APSITT 2003*, pp455-460, Nov 2003.

[22] R. Stewart, Q. Xie, et al. Stream Control Transmission Protocol. RFC 2960, October 2000.

[23] T. Dunigan, M. Mathis, and B. Tierney, A TCP tuning daemo. In *Proceedings of SuperComputing: High-Performance Networking and Computing*, Nov. 2002.

[24] N. C. Hutchinson and L. L. Peterson. The x-kernel: An architecture for implementing network protocols. *IEEE Transactions on Software Engineering*, 17(1):64-76, Jan. 1991.

[25] 小野 祐介, 斉藤 俊介, 田中 康之, 寺岡 文男. 通信の特性に応じた TCP アルゴリズム切り替え機構の設計と実装. マルチメディア，分散，協調とモバイルシンポジウム (DICOMO2005), Jul. 2005.

[26] Salvatore DeSimone and Christine Lombardi. Sysctl: A Distributed System Control Package. In *Proceedings of the 7th USENIX Systems Administration (LISA VII) Conference.*

[27] Iperf.
http://dast.nlanr.net/Projects/Iperf/

[28] Dummynet.
http://info.iet.unipi.it/ luigi/ip_dummynet/