コネクションごとのTCP最適化機構の設計と実装

田原 裕市郎[*] , 小川 晃通[*] , 杉浦 一徳[**] , 中村 修[***]
慶應義塾大学 政策・メディア研究科[*] , 通信総合研究所[**] , 慶應義塾大学 環境情報学部[***]

インターネットの基盤技術であるTCPは転送効率を高めるための研究がなされている．そのような研究の多くは汎用性が高いものではなく, 特定の環境に適合した拡張実装である. そのため, 標準的なTCPとして採用されることがない. 本研究では, コネクションごとにTCPの挙動を変化させ, 様々なネットワーク環境に適合するTCPの利用方法を提案する. 今回, ソケットを生成する際に, 与えられた条件により動作の切り替わるTCPの拡張機構の設計と実装を行った. この機構により, 柔軟性の高いTCP最適化機構の提供を目指す.

キーワード ： TCP，転送効率，環境適応

The Design and Implementation of Optimezed TCP Mechanism per Connections

Yuichiro Tahara[*] , Akimichi Ogawa[*] , Kazunori Sugiura[**] , Osamu Nakamura[***]
Graduate School of Media and Governance, Keio University[*] , Communication Research Laboratory[**] , Faculty of Environmental Information, Keio University[***]

TCP is the basic technology of the Internet. Many researchers have improved its mechanism to enhance the transference efficiency. Most of the researches are the extended reforms of TCP for a limited environment. Therefor, their TCP lost its generality, and a chance being the standard. The aim of this paper is a suggetion of efficient TCP in any network environment by switching its behavior per the connections.

Key words ： TCP，efficient transporting，adaptable mechanism

1 はじめに

現在, インターネットに接続されているホストコンピュータ, またはそれに類する情報端末の種類は多岐に渡っている. 利用可能なネットワーク帯域に限ってみても, 広く一般的に利用されている携帯電話を利用した10Kbps程度のものからイーサネットを利用した100Mbpsまで存在し, 最

近実用化が進んでいる WDM 伝送装置を利用した場合は 10Gbps という巨大な通信帯域を提供する. この通信環境の格差は開く一方である.

インターネットは, 通信に伴うハードウェアの特性を柔軟に吸収することによって広く普及することができたネットワークシステムである. しかし, 今日のこれほどの通信環境の格差は当初の想定を遥かに超えている. そのような格差があるにも関わらず単一の仕組みを適用しようとしているため通信効率の理論値と実測値が大きく開いてしまうという問題が発生している.

そのため近年では, 特に通信効率が悪化する場合に対応するため, インターネットの通信方式である TCP に改良を施す研究が数多くなされている. しかし, それらの研究成果は優れた結果が得られても, あまり普及していない. その原因として, 既存の TCP が高い汎用性を持つのに対して, 特定の環境下での効率化を前提になされた TCP 改良では別の環境下で性能がでないという問題を抱えているためである.

そこで本研究では, 通信環境に応じて TCP の挙動を変更させることで, 既存の研究成果を活用しつつも汎用性を維持し効率的に転送を行う機構を提案し, その有効性を示す.

2 環境設定

本稿では, 特に次に述べるような環境下における通信効率化を目指す.

イントラネットのように, 同一の組織や建物内部における通信でも,TCP/IP を用いて通信を行うケースは多い. これはユーザから見て, インターネットと内部への通信それぞれに対して同一のアプリケーションを使用できる利便性があるためである.

しかし, インターネットを利用した通信では, 輻輳などによるパケット損失や, 十分な通信帯域が確保できないなどの問題があるため, エンドノードの性能限界まで転送効率を高めることができない場合が多い.TCP は, そのような場合に対応するためフロー制御機構や輻輳制御機構を備えている.

そのような TCP の機構は, インターネットを介する通信では効率的に作用するが, 通信効率の面から述べるとイントラネットなどの内部間通信では, その限りではない. 同一ネットワークセグメントにおける通信ではルータによるパケット損失の可能性は無い. また, 同一組織ならば, 通信インフラの問題は高性能な機器を導入などすることで改善が容易である. しかしながら, 通信インフラとエンドノードの能力に余力があっても, 通信性能が TCP によって抑えられていることは多い.

そのため, 図 1 のように通信先によって挙動を変化させる TCP が存在すれば内部間での通信効率を高めることができる. 図 1 では,TCP A が通常の TCP による通信を,TCP B が挙動を変えた TCP による通信を表す. 本実装は, 外部との通信では通常の TCP 機構を, 内部との通信では高速に通信を行う TCP 機構を動的に切り替えながら利用できる機構を実現する.

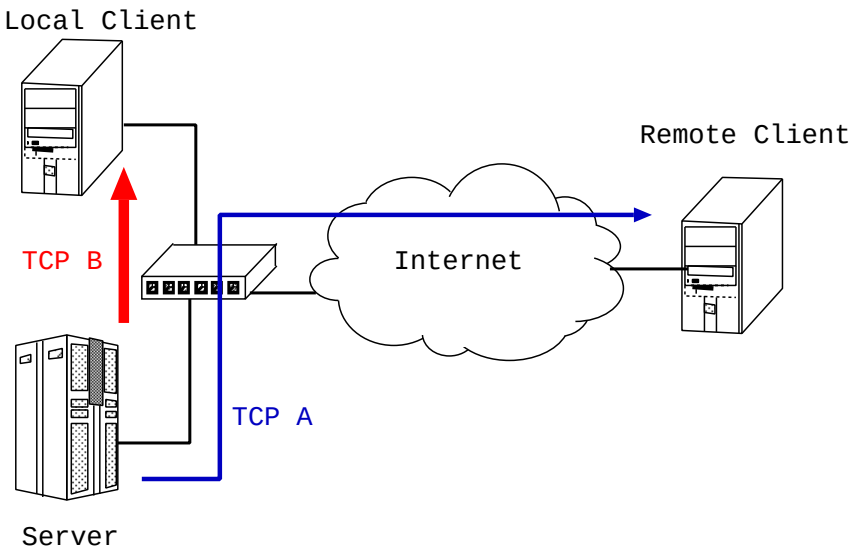

Fig. 1.　本研究のモデル環境.

3　設計

3.1　方針

TCP 通信を高速に行う手段として次のようなものが考えられる.

1. コネクションの確立処理を省略する.
2. 初期輻輳ウィンドウサイズを大きくする.
3. 送受信バッファサイズを大きくする.

本設計では, 比較的容易に実現できる 2 番目と 3 番目の方式を採用する.

3.1.1　輻輳ウィンドウサイズ

一般的に TCP の実装ではフロー制御機構としてスロースタート機構を有している. これはネットワークに突発的なパケット流量を起こさないための機構であるが, 同一セグメントにおける通信では, ウィンドウサイズが短期間で適正サイズになることを阻害している. 初期ウィンドウサイズを大きくすることで通信開始時から速い転送速度で通信できる. 特に HTTP において小さなサイズのファイルが同一のエンド間で複数のコネクションによりやりとりされる場合の有効性が RFC2414[1)] で示されている.

3.1.2　送受信バッファサイズ

TCP/IP Soket API では,setsockopt システムコールによって,TCP の送受信バッファを変更できる. 通信効率を意識したアプリケーションは, これを活用している場合が多い. しかし, 転送速度に対する要求をアプリケーションによって分類することは不完全である. 高速転送を必要とするアプリケーションとそうでないものに分類することはできる. だが, 要求通りの速度は, そのホストが置かれているネットワーク環境に左右される.

アプリケーションに, ネットワーク環境を検知し状況に応じて高速化する処理を加えることは可能である. しかし, アプリケーションプログラマがこのような煩雑な作業を考慮しなければならないのは困難が多い. また, アプリケーションごとに様々な検知機構が存在するのは無駄が多い. したがって, オペレーティングシステムによって, このような機構が存在している方が効率的で使い勝手が良い.

J. Semke, 1998[2)] では,NetBSD 1.2 において動的に送信バッファを変化させることで効率的な転送を実現できることを示している.

3.2　設計モデル

本研究の設計は, 図 2 のようになる.

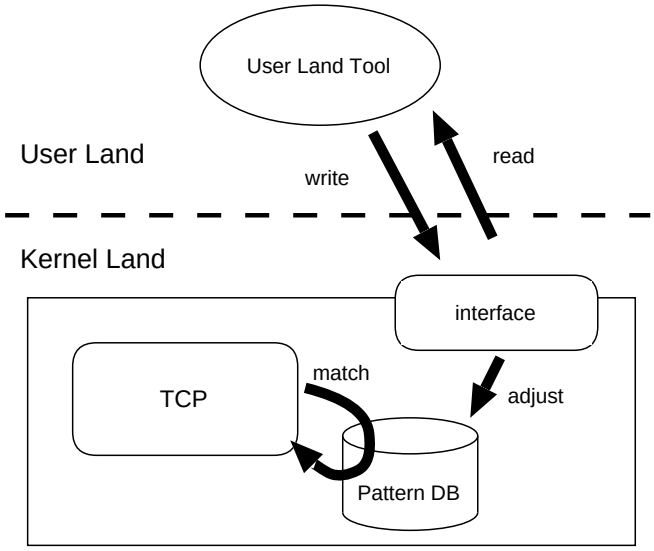

Fig. 2.　モデル図.

まず, ユーザはツールを使い, 現在の状態を知る. それを元に, 適合させたいパターンと TCP の挙動の組み合わせをツールによって書き込む. そのパターンのデータは Kernel 空間内に保持され TCP コネクションが作成される度に参照される.TCP スタックは, パターンによって挙動を変更する.

ユーザが選択できるコネクションのパターンとして以下に挙げるものを選択した.

1. Destination IP Address
2. Source IP Address
3. Destination Port Number
4. Source Port Number

このユーザランドプログラムの使用感は, ファイヤウォール設定ツールと似たような感じとなる. また, そのことによりファイヤウォールで培った知識, たとえばアプリケーションのポートレンジなどの知識がそのまま流用できる.

4 実装

本システムは, 以下に挙げる 3 つのソフトウェアの実装により成る.

1. パターンの読み出し・変更を行うユーザランドプログラム
2. ユーザ空間←→カーネル空間のインターフェース
3. TCP プロトコルスタックへの改変

4.1 実装環境

実装環境を表 1 に示す.

Table 1. 実装環境.

Kernel	Linux 2.4.17
libc	glibc 2.2.4
C コンパイラ	gcc-2.95.4

本システムは, 現在のところ Linux のカーネルアーキテクチャに強く依存している.

4.2 手法

4.2.1 プロトコルスタックへの改変

ウィンドウサイズの初期部への改変は, データの送信を行う側, 多くのネットワークプログラムではサーバ側への変更である. さらに送信バッファも同様である.

対して, 受信バッファへの改変は, データを受信する側, 主にクライアント側となる.

図 3 は TCP のスリーウェイハンドシェイク時にカーネル内部で呼ばれる関数を表したものである. まず,TCP 層では tcp_v4_connect() 関数が呼ばれ,SYN パケットを送出する. その後,TCP の状態を SYN_SENT に移行して, tcp_rcv_stats_process() 関数を呼び出し,SYN:ACK を待つ.SYN:ACK を受け取ると tcp_init_metrics() を呼び出し, 通信設定を行う.

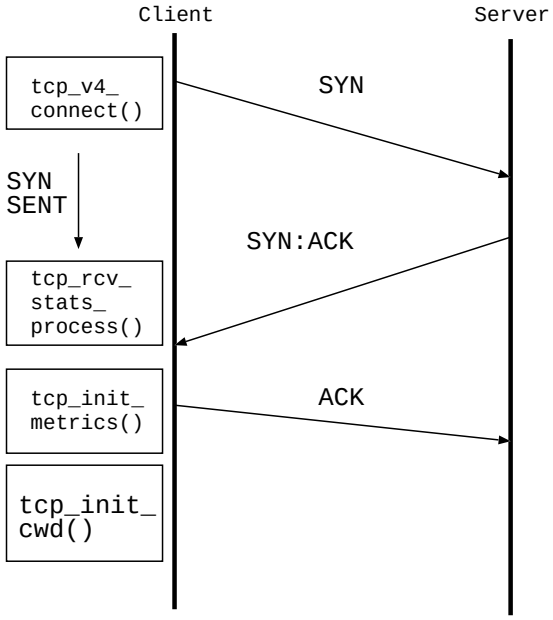

Fig. 3.　コネクション確立時の関数.

tcp_init_metrics() は初期ウィンドウを設定するために,tcp_init_cwd() を呼び出す.

本実装は, 送信先 IP アドレスがローカルエリアの場合に tcp_init_cwd() 関数ではなく, 初期ウィンドウサイズを通信先によって広告された最大ウィンドウサイズに設定する.

また, 送受信バッファは, カーネル内部で tcp_setsockopt() 関数を呼び出すことで, システムコールの setsockopt() と同じように変更できる.

4.2.2　ユーザ空間←→カーネル空間のインターフェース

本実装では,Linux の procfs 機構を用いることで, ユーザ空間とカーネル空間のインターフェースを容易に実現した.

5　検証

5.1　既存システムとの比較

本稿では, 送信先 IP アドレスをもとに TCP の初期ウィンドウサイズを切り替える実装を行った. これは, すなわちコネクションごとに TCP の挙動を変化させたことを意味する.

最近の UNIX 実装でも,TCP のパラメータは動的に変化させることができる. 例えば, FreeBSD や Linux は sysctl によって,Sun Microsystems の Solaris では/dev/tcp によって行える. しかし, それらはカーネル全体に対する変更であって, コネクションごとに変更を行うことはできない. インターネットのホストは, 同時に多地点と通信する場合が多い. 閉じたネットワークに対するファイルサーバのような用途ならば, 既存システムで十分であるが, 多くのホストは多様なネットワーク環境で動作する. 特に, 移動体通信環境では, 本システムは高い効果を発揮する.

5.2　今後の課題

TCP の挙動を変化させるには送信・受信バッファや輻輳ウィンドウサイズの他に,TCP Vegas[3] のようにアルゴリズムを改変し効率的な転送を行う研究もある. 今後, そのような TCP アルゴリズムを動的に切り替えられるようにすることを目指す.

参考文献

1) M. Allman,S. Floyd,C. Partridge, 'Increasing TCP's Initial Window', RFC2414, September 1998

2) J. Semke, J. Mahdavi, M. Mathis, 'Automatic TCP Buffer Tuning', ACM SIGCOMM '98/ Computer Communication Review volume 28, number 4, October 1998.

3) L. Brakmo, S. O'Malley, and L. Peterson, 'TCP Vegas: New techniques for congestion detection and avoidance', ACM SIGCOMM '94 Symposium, Aug. 1994