修士論文　　2002年度 (平成14年度)

動的に代替可能なTCP制御機構に関する研究

慶應義塾大学 政策・メディア研究科

氏名：田原 裕市郎

修士論文要旨 2002 年度 (平成 14 年度)

動的に代替可能なTCP制御機構に関する研究

現在、インターネットに接続されているエンドノードであ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。インターネットは、通信に伴うハードウェアの特性をソフトウェアが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、現在のインターネットで利用されるコンピュータの通信環境の格差はインターネットの基本システム設計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 TCP のチューニング研究が数多く行われている。しかし、それらの研究成果は優れた結果が得られても、遅々として普及が進まない。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 チューニング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、既存のチューニング技術を活用すると同時に汎用性を維持し、効率的に転送を行う機構を提案し実装した。この機構により単一のホストで複数の TCP チューニング技術を利用できるようになり、利用上で高い柔軟性を付与することができた。

これにより、TCP チューニング技術が普及することを妨げる一要因を解決できた。

キーワード
1. インターネット、2.TCP, 3. チューニング, 4. 通信環境, 5. 通信効率

慶應義塾大学 政策・メディア研究科

田原 裕市郎

Abstract of Bachelor's Thesis

Academic Year 2002

The Research of Alternative TCP Control

The spread of the Internet was realized by absorption of hardware characteristics by software. However, the current communication environment has changed rapidly, compared to when the basic design of the Internet was created. Thus, there is a large difference between the theoretical and measured value of the communication efficiency.

To overcome some situations that cause inefficiency, there are many research that adds a new mechanism to the TCP. Though many TCP tuning are very effective, they are not popular. This is because the TCP tuning are focused on especial environments.

In this research, a effective TCP behavior selection mechanism is designed and implemented. By creating a mechanism to select the most effective TCP tuning value, high quality communication can be realized without reducing the versatility of TCP. Using this mechanism, multiple TCP tuning technique can be used within a single host. This realizes high flexibility and versatility. The results of this research solves one of the problem that restrict the spread of the TCP tuning technology.

Keywords :
1.Internet, 2.TCP, 3.tuning, 4.communication environment, 5. efficiency

Keio University, Graduate School of Media and Governance

Yuichiro Tahara

目 次

図 目 次

表 目 次

第1章　序論

1.1　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の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。インターネットを取り巻くローエンドなシステムとハイエンドなシステム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。

1.2　インターネットにおける TCP の役割

TCP はインターネットにおいて最も重要な技術の１つである。インターネットの中間ノードはパケット交換方式のデータ送受を行う。この方式は、多数の利用者がネットワーク資源を同時に利用できる点で優れている。しかし、ネットワークが許容できる以上のデータ量がネットワークに流れた場合、中間ノードによってデータが破棄される可能性がある。そのため TCP は送信側と受信側のノードで動作し、ネットワークアプリケーションに対して、低信頼性のパケット交換ネットワーク上に仮想のコネクション型通信路を提供する。TCP は、中間ノードで

失われたデータを検知し、再送処理を行うことで信頼性の高い通信を提供できる。

インターネット上で、データの欠損無く通信をする場合は、ほとんどの場合において TCP を利用している。TCP 技術が改良されることは、インターネット利用者がより高品質のネットワークサービスを享受できることを意味する。

インターネットは、互換性の無い通信機器上でも、TCP/IP 通信方式を計算機上に提供し、ハードウェアの差異を吸収することで実現されている。

IP は通信到達性を提供する仕組みであり、通信効率にはあまり関与しない。通信効率に大きく関わる部分は TCP によって提供されている。しかし、TCP のインターフェース部は通信実現のために規定されているが、通信効率に関わる具体的な内部の計算アルゴリズムは、多くの場合実装に依存している。そのため、TCP の通信効率における問題点は、設計の段階で想定したネットワーク環境と大きく関っている。実際利用される環境と設計時に想定した環境に大きな違いある場合、問題が発生する。

1.3 本論文の目的

上で述べたように、TCP にはその通信環境に合わせたパフォーマンスチューニングが出来ていない。本論文では、TCP チューニングの現状をまとめ、それぞれの長短を論じる。さらに、それらの欠点を補う機構を提案し、そのシステムの設計を述べる。最後に、本研究が提案するシステムの実装と評価を行う。

1.4 用語の定義

TCP チューニング

本論文では、TCP チューニングを次に挙げる条件を満たすものとして定義する。

- 帯域などのネットワークのリソースを可能な限り無駄なく利用する。
- 過剰なパケット送出に起因する可避な輻輳を起こさない。
- ネットワークのリソースを占有することで他のホストに損害を与えない。

以上を元に議論を進めることとする。

1.5 本論文の構成

第 2 章では、既存の TCP が持つ技術的な問題点について述べる。第 3 章では、2 章で述べた問題点に対する様々なチューニング技術を紹介すると供に、既存のチューニング技術の問題点を明らかにする。第 4 章では、本論文が提案するシステムの設計を述べる。第 5 章では、第 4 章で定めた設計要件に基づき実装を行い、それを評価する。第 6 章では、本論文の結論を述べる。

第2章　TCPが持つ性能上の問題点

本章では、TCP が持つ性能上の問題点を明らかにする。また、TCP をプロトコルレベルで変更することで問題が解決された事項を述べる。

2.1　TCP 設計における問題点と既存の解決案

これまでの多くの TCP 実装は、ネットワークの通信速度が、計算機内部の計算処理やデータ複製に比べて遅いことを前提に設計されている。しかし、通信機器の急速な発達によって、この前提では整合性のとれない部分が多くなってきた。また、通信機器の広帯域化は、計算機自体の処理能力も高性能化が求められる。このような高性能な通信機器が出現した一方、既存の通信機器（モデムや 10Mpbs のイーサネット）の利用率は未だに高い。これは、ネットワークインフラの整備が進んでいないこととインターネットで利用されるアプリケーションプログラムに広帯域回線を求めるものが少ないことに原因がある。そのため、現在のインターネットは低速・低帯域から超高速・広帯域のものまで様々な種類の通信機器が混在している。利用者数でも低速・低帯域の回線利用者が多数であるため、大きな帯域差と遅延差に最適化された通信機構の開発や利用は進んでいない。

既存の TCP では、以下に述べるような問題が知られている。また、いくつかの問題に対しては解決案が提示されている。

2.1.1　広帯域通信における問題

基本仕様の TCP では、TCP ヘッダのウィンドウサイズフィールドは 16bit 長しかなく、最大で 65535byte までしかウィンドウサイズを指定できない。しかし、ウィンドウスケールオプション [2] を利用することにより、ウィンドウサイズフィールドには、ウィンドウサイズを直接指定するのではなく、ウィンドウサイズを 65535byte の何倍とするのかというウィンドウスケール値を指定できる。この値も 16bit で表せるため、ウィンドウサイズの上限は 4GB になる。ただし、この場合はウィンドウサイズは一単位が　65535byte となりより細かな範囲の指定はできなくなる。

これは、広帯域･高遅延なネットワーク利用に向いている。ウィンドウサイズの上限が 65535byte

では、広帯域回線が十分に空いている状態でもスループットは頭打ちになる。また、高遅延ネットワークの場合は確認応答の受信時間が大きいため待ち時間が増大し、ウィンドウサイズが小さいと通信性能は格段に悪くなる。

タイムスタンプオプションは、送信側 TCP が現在のタイマカウンタ値を送り、受信側は確認応答と共にタイムスタンプ値を返送する。送信側 TCP は、タイムスタンプからデータパケットを送出してから確認応答を受信するまでの時間を計測できる。これは 2 つの目的がある。1 つは全てのデータパケットで RTT を計測できる点である。多くの実装では RTT を 1 ウィンドウあたり 1 回のみ計測する。この方式を用いた場合ウィンドウサイズが大きくなるにつれて RTT 値の精度は低下する。

もう一つの目的は、シーケンス番号周回問題の解決である。TCP は上位層から渡されたデータ 1byte ごとに先頭を 0 として最大 32bit-1 のシーケンス番号を付与する。番号は使い切った段階で再び 0 に戻る。しかし、高速なネットワークでは 32bit のシーケンス番号はすぐに周回してしまう。そのため、データをシーケンス番号とタイムスタンプの組み合わせで区別する。この解決法は PAWS アルゴリズム [2] と呼ばれている。

2.1.2 再転送に関する問題

TCP は、送信したデータが受信者に届かなかった場合、再転送を行う。通常、送信者はデータの損失を受信者からの確認応答までの時間が RTT に比べて十分に長い場合に行う。その再転送を行うためにタイマ機構が用いられている。効率的なデータ転送には、より正しい RTT の測定とが前提となる。しかし、高遅延な通信路では回線が空いていても RTT が大きくなるため、効率的な再送処理は困難である。

また TCP では基本的に、最初にデータ送信を開始する場合、RTT を計測する事なしに適切な再送タイムアウト値（RTO）を 3 秒に設定することが RFC1122[1] で推奨されている。RTT が十分大きな通信環境では、これが原因となり極端な性能劣化が発生する。なぜなら、`RTO < RTT` の場合に送信者はパケットの ACK を受け取る前に不必要にパケットが再送され、通信帯域に十分な余裕があるとしても、送信者が送出量を増大させるのに時間がかかる。そのため、RTT が大きな通信路では適切な RTO の設定が必要となる。

そして、多くの TCP 実装では、確認応答の数に応じてウィンドウサイズを増加させるため、高遅延ネットワークの場合、ウィンドウサイズの増加がゆるやかになる。ネットワークの混雑によって遅延が発生している場合は適切な挙動であるが、国際線などの長距離ネットワークによる高遅延の場合は最適なウィンドウサイズに収束するまでに時間がかかり通信効率が悪くなる。そのため、広帯域・高遅延のネットワークの場合は、確認応答数ではなく確認応答されたデータパケット数に応じてウィンドウサイズを増加させるようにアルゴリズムを変更する提案

がなされた。

2.1.3 パケット喪失率が高い通信路における問題

基本的なウィンドウ機構では、すでに届いたパケットでも、それ以前のパケットの確認がとれていないために、ウィンドウ内にあるパケットの再送を行ってしまう。無線ネットワークのような、パケット喪失率が高いネットワークで効率の良い通信を行うために、TCP の累積確認応答アルゴリズムを拡張し、受信側は喪失パケットに関する詳細な情報を送信側に返送するオプションが提案された。このオプションによって喪失パケットの情報が取り出せるため送信者は既に届いたパケットの再送を行う必要がなくなる。その機構は SACK と呼ばれる。

2.2 計算機処理能力における問題点

ネットワーク機器が広帯域化しただけでは、広帯域通信環境を有効利用することはできない。インターネットアーキテクチャーではネットワークのエンドノードに任されている処理が多いため通信に関わる計算処理コストは大きい。現在の高性能な PC でも 100Mbps の通信を行い続けると、内部転送バス占有率が高くなってしまう。

巨大なデータを通信でやり取りする場合には、巨大な受信バッファが必要になる。例えば、10Gbps の通信デバイスを持ち、10GB のメインメモリを搭載した PC を考えると、仮にすべてのメモリを転送バッファに当てたとしても、10Gbps の転送速度では 8 秒で溢れてしまう。この話は極端な例であるが、通信の高速・広帯域が進むにつれて計算機の能力は無視できなくなる。具体的には、搭載メモリによって OS は TCP の挙動を変化させるような仕組みが有効に成り得る。

第3章　TCPチューニング技術

第2章では、プロトコルレベルのTCP改変について述べたが、TCPの性能を向上させるにはプロトコルを変えなくても可能である。プロトコルに依らないTCP改変は高い互換性を保持できるところが利点である。本章では、その主な手法を取り上げる。

3.1　アプリケーションレベルの性能改善

3.1.1　NLANR FTP

NLANR FTP[3]は、使用者がFTPクライアントで動的にバッファサイズを設定可能なようにアプリケーションの改変を行う。FTPプロトコルを拡張し、クラアントはサーバに対してバッファサイズの変更を要求する制御を行う。サーバ側では、要求に従い送信バッファを大きくして、クラアントのGETコマンド発行時に通常よりも高い転送速度で送出を行う。

NLANR FTPでは、TCPが利用するバッファサイズを使用者が転送を行う時点で判断する必要がある。

3.1.2　drsFTP

drsFTP[6]（Dynamic Right-Sizing FTP）は、基本的に前項のNLANR FTPと同様に、FTPのコントロールコマンドを拡張しクライアントとサーバ間でバッファサイズを調整し合う。両者の違いは、NLANR FTPがデータコネクションの初期化時のみにバッファサイズを調整するのに対して、drsFTPではデータ転送中もバッファサイズの調整を続ける。

3.1.3　Enable

Enable[5]は、クライアントとサーバ間での計測結果を元に、TCPのバッファチューニングを施す方式である。FTPサーバなどのコンテンツサーバ上でEnableデーモンが稼働する。Enableデーモンは、Enable対応クライアントとの通信結果を記録しデータベースに記録する。同一ホストから再び通信要求がなされた場合、前回の通信結果を元にバッファサイズを設定する。

Enableはアプリケーションに対して、主に以下の3つの要素を提供する。

1. Enableサーバ
2. Enableサーバと Enable対応クライアント間の通信プロトコル
3. Enableサーバへ問い合わせを行うための API

Enableサーバは、EU DataGrid Project[4] の環境に合わせて設計されている。これは、物理学などの科学計算を分散処理し中央データベースサーバを介して演算を行うもので、特定のクライアントとサーバ間でサイズの大きいデータのやりとりが頻繁に行われる。Enableは、前回の通信結果が利用されるので、このように密に結びついたホスト間の通信では効果を上げることができる。

3.1.4 まとめ

第3.1節で紹介した技術は、いずれも送受信バッファによって転送改善を計っている。これを次の表3.1にまとめる。

チューニング手法	バッファサイズ変更	チューニングサイド	変更主体
NLANR FTP	静的	サーバ側	手動
DRS FTP	動的	両エンド	手動
Enable	静的	サーバ側	自動

表 3.1: アプリケーションレベルバッファチューニング

上図3.1において、DRS FTPのみがデータ送信途中でバッファサイズを変更できる。

チューニングサイドは、データの送信側と受信側のどちらをチューニングできるのかを表している。NLANR FTPと Enableは送信側、つまり FTPサーバ側でバッファチューニングを行う。対して、DRS FTPは FTPのコントロールセッションでクライアントとサーバがバッファサイズを調整しあうため、双方でバッファのリサイズが行われる。

変更主体は、バッファサイズを決定する主体を表している。NLANR FTPと DRS FTPでは、FTPクライアントを操作するユーザがバッファサイズを指定する必要がある。Enableは、過去の通信記録を元に FTPサーバが自動的にバッファサイズを決定する。

3.2 TCP 実装レベルの性能改善

3.2.1 バッファチューニング

多くの OS 実装では、socket バッファの標準値が広帯域なネットワークリンクで十分な性能が発揮できないことが知られている。この問題に対して、動的にバッファの上限値を変化させる [7] ことで、転送速度を上げることができる。しかし、最適な値を設定するには、ネットワークパスで利用できる帯域を知る必要があるなどの問題がある。

3.2.2 TCPVegas

TCPVegas[8] は RTT を正確に測定し積極的に利用することによって輻輳制御を行う仕組みである。測定中で最小の RTT 値を輻輳が無い場合の RTT とし、現在のウィンドウサイズをこの最小 RTT で割った値をスループットの期待値とする。現在の RTT で送信バイト数を割った物を実際のスループットとし、その誤差（Diff）で輻輳制御を行う。TCPVegeas は 2 つの閾値 α と β を設定する。Diff ＜ α の場合は輻輳ウィンドウを線形的に増加させ、α ＜ Diff ＜ β の場合は変化させず、Diff ＞ β の場合は減少させる。TCPVegas は輻輳制御を行い、線形的に輻輳ウィンドウを変化させるため効率的な転送を行えるが、急激な輻輳が起こった場合に、輻輳制御が機能しない等の問題点がある。

混在環境における問題

TCPVegas を旧来からの TCP と混在させると、Vegas 側が性能上の不利を受けるという問題 [9] がよく知られている。

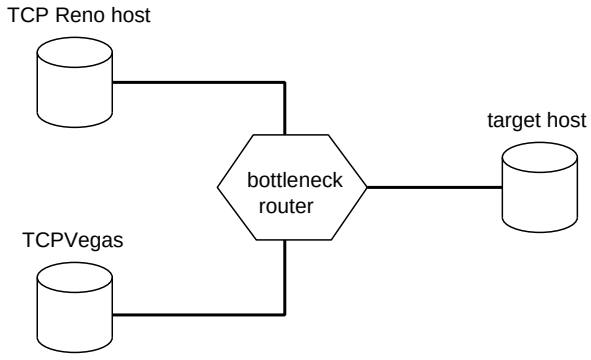

図 3.1: TCPVegas と TCP Reno の混在環境例

上図 3.1 は、最も単純な TCPVegas ホストと TCP Reno ホストの混在環境を表している。

bottleneck router で輻輳が発生した時に、各ホストは次のように振る舞う。TCPVegas ホストは、ルータバッファの占有率が高くなると、それに伴い送信量を減少させる。TCP Reno ホストは、パケットロスを検知するまで送信量を減らさない。そのため、ルータバッファを使い切るまでウィンドウサイズを増加させる。よって、TCPVegas は、旧来型の TCP 搭載ホストとの混在環境では、転送性能の面において一方的に不利益を被る。この問題が、TCPVegas 普及を妨げる一因となっている。

3.2.3 初期ウィンドウサイズの増加

RFC3390[10] では実験的に TCP の初期ウィンドウサイズを RFC2001[19] で定められた 1 セグメントからおおよそ 4K バイトに増やすことの長短を調べ、実験結果を報告している。初期ウィンドウサイズを増やすことの利点は以下の 3 つである。

1. 輻輳の立ち上がり時におけるタイムアウトの待ちをなくす。
2. 多くの電子メール・Web ページ転送は 4K バイト以下であるため、小容量のデータ転送は一回で済む。
3. 最初のスロースタートフェーズの間、遅延応答確認をなくす。これは衛星通信のような高帯域大規模普及遅延ネットワークに対する利点となる。

それに対して、高輻輳下においては、パケットの欠損率を高くなる欠点が言及されている。

3.2.4 Linux 2.4 の自動バッファチューニング

Linux の version 2.4 系列より、送受信バッファの自動調節機能が搭載されている。従来の version では固定で、アプリケーションが setsockopt システムコールを発行しない限りは OS が定めた値が使われていたが、デフォルト値・最小値・最大値を定めることでアプリケーションの送受信に合わせて、デフォルト値から始まり最小値から最大値の幅でバッファサイズを変化させてくれる。この機能が有効な場合は、アプリケーションの setsockopt によるバッファサイズの指定は無視される。

3.3 関連研究

3.3.1 IETF pilc Working Group

IETF（Internet Engineering Task Force）トランスポートエリアの pilc ワーキンググループでは、特徴的な通信路での TCP の転送パフォーマンスに関して議論されている。現在、6 つの RFC や Internet Draft が発表されている。その中でも本研究を関連性が強いものを以下に 4 つ挙げる

1. End-to-end Performance Implications of Slow Links (RFC3150)
   低速通信路におけるホスト間の転送性能に関する報告

2. End-to-end Performance Implications of Links with Errors (RFC3155)
   リンク層でエラーが多い通信路におけるホスト間の転送性能に関する報告

3. TCP Performance Implications of Network Asymmetry
   ネットワークの非対称性と TCP 転送性能との関連性の報告

4. TCP over Second(2.5G) and Third(3G) Generation Wireless Networks
   次世代携帯電話をネットワークリンクとする場合の報告

まとめると上記の文書は、ある特定のネットワーク条件下に、TCP の転送性能に関して効果的に作用する TCP の拡張実装やアルゴリズムが存在することを述べている。

3.3.2 PSC Networking Research Group

PSC（Pittsburgh Supercomputing Center）の Networking Research Group ではホスト間の高速データ転送に関する研究を行っている。その中から以下にあげる論文を紹介する。

- M.Allman, S.Dawkins, D.Glover, J.Griner, D.Tran, T.Henderson, J.Heidemann, J.Touch, H.Kruse, S.Ostermann, K.Scott,and J.Semke. ”Ongoing TCP Research Related to Satellites.”
  衛星通信路を含むネットワークでの TCP に関する研究。

- J. Semke, J. Mahdavi, M. Mathis, ”Automatic TCP Buffer Tuning”, ACM SIGCOMM ’98/ Computer Communication Review, volume 28, number 4, October 1998.
  送信バッファをチューニングすることで高帯域通信路で高速転送の試みとその実験を行っている。

3.4 次世代高速 TCP 研究

3.4.1 PSockets

PSockets[13] は、本来アプリケーションが一つのソケットを利用して通信を行うものを複数のソケットを同時に利用することで転送速度を上げる仕組みである。概要は下図 3.2 のように、アプリケーションからは一つのストリームとして隠蔽化されるが、PSockets が複数の TCP/IP ソケットへとデータを振り分け、エンド間の通信を高速に行う。

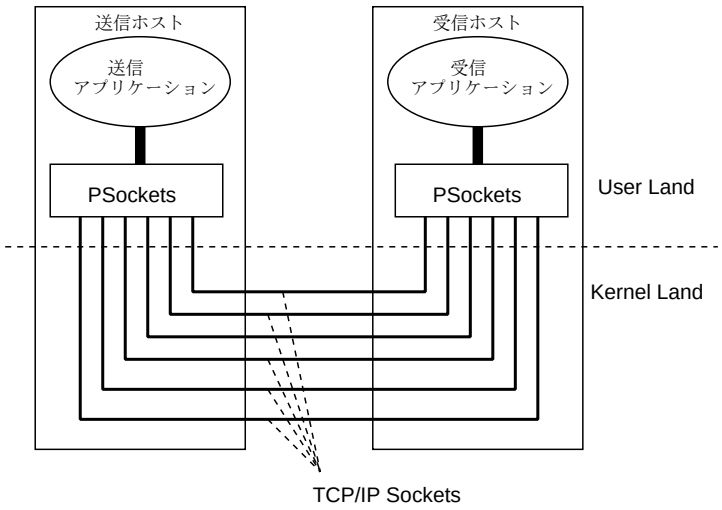

図 3.2: PSockets 概要

PSockets の長所は、複数のソケットを同時に利用するという簡単な仕組みで、通常は困難な広帯域ネットワークでの高速化を実現できる。複数ソケットで送信を行う長所は、まとめると以下の通りである。

- OS によって設定されている TCP の最大ウィンドウサイズの制限を受けない。
- 単一ソケットでは ACK 待ち時間はそのままブロック時間となるが、複数ソケットだとあるソケットが ACK を待つ間に送信を行う。
- ある TCP ストリームが輻輳回避アルゴリズムによって輻輳ウィンドウサイズを落とした場合に送信量が極端に落ちるが、その瞬間にネットワークリソースが部分的に解放されることを利用して他のソケットが送信量を増やすことができる。

上記の理由から、PSockets は効率的にデータ送出ができるとされているが、他のホストの通信を圧迫したり低速リンクでは効果が出ないなどの問題を有する。

3.4.2 Web100 Project

Web100 Project[14] は、HTTP 転送においてネットワークリソースを使い切ることを目標としている。その一環として、広帯域なリンクであっても、リソースの最大限まで利用することができる TCP の開発研究を行っている。

TCP-KIS

Web100 Project の一環として、TCP-KIS（TCP Kernel Instrumentation Set）という Project がある。これは、SNMP を利用することでネットワーク状態に関する情報をホストで収集し、その情報を元にネットワークパスの状態を推定する。これにより最適と考えられる TCP の挙動を kernel に反映させることによって TCP の効率化を目指すものである。TCP-KIS では、SNMP で収集する情報として TCP-MIB[17] を採用している。また、これを拡張し TCP-MIB 研究にフィードバックすることも目的としている。

Web100 Kernel

TCP-KIS の成果の一つとして、Web100 Kernel が公開されている。これは、Linux kernel に変更を施し TCP-KIS を実現したものである。これにより、ユーザランドから各 TCP コネクションのバッファサイズを変更できるようになっている。さらに、Web100 Kernel を操作するユーザランドプログラムのために libweb100 が公開されている。

3.4.3 WAD – TCP Tuning Daemon

WAD（Work Around Daemon）[15] は、ユーザランドに常駐するプログラムで、TCP コネクションを監視し、コネクションごとの TCP チューニングを行うシステムである。チューニングには、Web100 Kernel のバッファチューニング機能や、PSockets などを利用する。

WAD は、「netlink socket」を監視することで新たな TCP コネクションを知る。次に、WAD は wad.conf の記述に従い施すチューニングを決定する。wad.conf の記述例を論文 [15] より抜粋し、以下に掲示する。

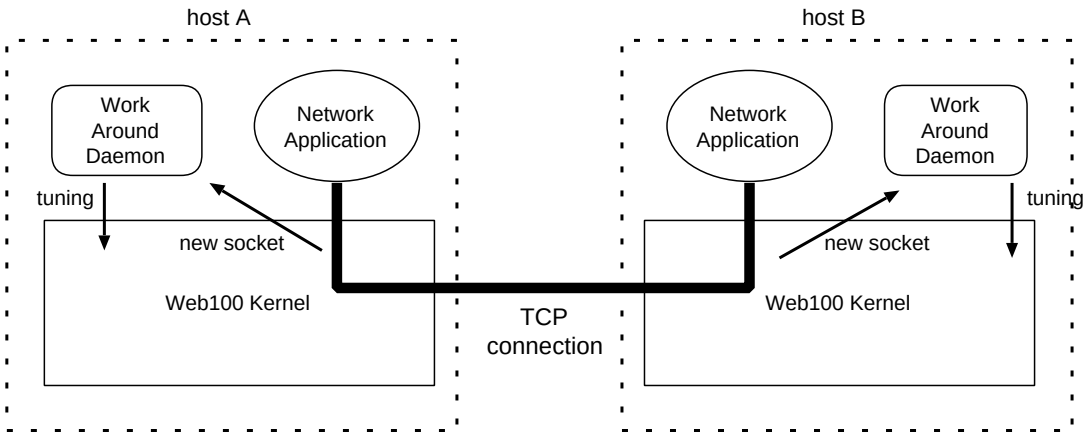

図 3.3: Work Around Daemon 概要

wad.conf の記述例

```
[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
```

記述のパラメータには、TCP コネクションを特定するものと、コネクションが条件に合う場合に行われるべきチューニング変数を併記する。

3.5 既存チューニング技術のまとめ

3.5.1 アプリケーションチューニング

アプリケーションチューニングによる TCP の性能改善は、チューニングそれ自体には OS が提供する setsockopt システムコールを利用する。そのため、多くの場合チューニング手段としてバッファチューニングを行うものに限られる。

setsockopt システムコールの SO_SNDBUF 値と SO_RCVBUF 値によって、それぞれ送信バッファと受信バッファを設定する。さらに、setsockopt システムコールの制限により、アプリケーションが起動してから TCP コネクションが成立する以前に socket バッファを設定する必要がある。この制限により、高速転送を目的とするチューニングでは想定される必要量より十分に大きなバッファ量を要求しなければならない。

また、アプリケーションによって一旦確保された socket バッファは、アプリケーションの終了まで割り当てられ、この値を動的に変化させるような改変はアプリケーションのみでは不可能である。それにより、無駄にバッファ占有が行われる可能性がある。

前提として、アプリケーションチューニングを行うにはアプリケーションそのものに改変を要する。既存のアプリケーションをチューニングするには、ソースコードレベルでチューニングを必要とする。たとえば、FTP なら ncftp のみに対する patch として提供されているものが存在する。

ユーザランドのみのチューニングでは、どのようにチューニングを施すべきかの指標となる情報をカーネル空間から得るのが困難である。その解決として、クライアントとサーバ間で通信帯域や RTT などの計測を行い、それをもとにチューニングが決定される。また、計測した情報を記録し活用するケースもある。しかし、前述の通り setsockopt システムコールは、TCP が通信を開始する以前に設定される必要があるため、目的とするデータ転送を行う以前に、計測用の通信コネクションを試みるものや、データコネクションとは別にコントロールセッションを張ることでデータベースから記録を取得する必要がある。

3.5.2 TCP 実装チューニング

既存の TCP 拡張実装は、TCP の転送性能劣化が大きなネットワーク環境や TCP の転送レートが伸びにくいネットワーク環境などの特定の通信環境に対する改善策が多く、どのような環境でも高性能を出す実装は生み出されていない。それは、TCP の基本設計の汎用性が高いことと、インターネットを取り巻く通信環境が多様な細分化を遂げているためである。

これに対して、様々な TCP の拡張実装では、ある特徴的なネットワーク環境に TCP の挙動を特化させることによってチューニングを実現している。特徴的な環境が存在する理由は、イ

ンターネットの普及に伴い、通信衛星・海底ケーブル・無線通信デバイス・高速通信機器などの様々な通信手段が利用されることに起因している。この結果、TCP/IP が利用可能が通信環境は、帯域格差・遅延格差・データ欠損率などの条件が大きく異なり、利用される通信環境に合わせることで通信効率化を行う余地が大きくなっている。ところが、ある環境に特化させた TCP 拡張では、異なる環境で通信劣化が発生する。例えば、広帯域高遅延の広域高速通信に特化させた TCP 拡張では、通信速度を向上させるために送信バッファを大きくし、輻輳ウィンドウサイズをできるだけ早い段階に閾値に近づける必要がある。そのため、ACK を待たずに多量のデータが送信される。しかし、低帯域リンクでこの TCP を用いた場合、大量のパケットロスとパケット再送により通常の TCP よりも転送速度が落ちる。

このように、環境に特化させた TCP は、特定の通信環境でしか用いることができないといった問題を抱えることとなる。

3.5.3 最近の動き

Internet2 の High Performance 研究分野 [16] では、既存の様々な TCP チューニング技術を統合することで次世代インターネット環境における TCP 性能改善を目指している。これまでの研究では、特定のアプリケーションに特化したものや特定の環境に特化したものなど、ユーザにとって使用方法が限定されるものだった。言い換えるならば、利用者はネットワーク環境や計算機環境を予め知っておく必要がある、もしくはそれに合わせてコンピュータネットワークを構築することを前提としている。しかし、WAD[15] ように状況に合わせて最適化手段を選択するような機構が開発されるようになった。これにより、これまで困難であった最適化技術の複数同時利用する使用方法が可能となった。

3.6 チューニング技術の分類

第 3 章で紹介した TCP 改良手法を実際に利用を行う観点で表 3.2 にまとめる。

3.6.1 分類項目

表 3.2 に出てくる各項目は次の通りである。

- NLANR FTP
- drsFTP
- Enable

- Linux 2.4（Linux 2.4 の自動バッファチューニング）
- TCPVegas
- RFC3390（初期ウィンドウサイズの増加）
- PSockets
- Web100

3.6.2 分類表

改善手法

如何なる手法によって TCP の転送性能を改善するのかを分類した。

プロトコルレベルの改変は、第 2 章で述べた通りである。

送受信バッファチューニングを導入するものが多い理由は大きく分けて 2 つある。第一に、改変が非常に容易な点である。アプリケーションから利用する場合は、setsockopt システムコールを利用すればよい。カーネルレベルで TCP 改変を行う場合は、カーネル内部の変数を変更するのみで済む。第二に、バッファサイズがボトルネックに成りやすい点である。TCP の転送性能に問題がある場合に、バッファサイズが必要量に満たないことが多い。そのためチューニングを施す場合は、まずバッファサイズから始めるべきである。上記の理由から、転送性能改善手法としてバッファチューニングが採用されるケースは多い。

TCP のアルゴリズム改良は、TCP プロトコルの互換性に影響の無い部分で改変が行われる。主として、輻輳ウィンドウサイズの増減に関わるアルゴリズムを改良する。

コネクション多重化するのは、PSockets の特徴である。アプリケーションレベルの高速化手法だが、よく似たものに FTP や HTTP のレジューム機能を利用したコネクション多重化がある。

実装対象

改良実装を施す箇所で分類を行った。

ユーザ空間での実装は、アプリケーションを改変するか、アプリケーションが利用するライブラリを改変することによって実現される。この方式の利点は、カーネル空間に比べて実装が容易なことである。

カーネル空間のみでの実装は、アプリケーション非依存に提供できる点が良い。反面、実装が複雑になったり、特定の OS に限定されてしまう欠点がある。

改善手法	TCP プロトコル改善	（第 2 章を参照）
	送受信バッファチューニング	NLANR FTP drsFTP Enable Linux 2.4 Web100
	TCP アルゴリズム改良	TCPVegas RFC3390
	コネクション多重化	PSockets
実装対象	ユーザ空間	NLANR FTP drsFTP Enable PSockets
	カーネル空間	Linux 2.4 TCPVegas RFC3390
	ユーザ空間とカーネル空間の双方	Web100
利用形態	バイナリレベルで固定	TCPVegas RFC3390 PSockets
	明示的に操作	NLANR FTP drsFTP
	自動処理	Linux 2.4 Enable Web100

表 3.2: TCP 改良研究の分類

Web100 は、実際にチューニングを行うカーネル部分の変更と、どのようにチューニングを行うのかをカーネルに通知するユーザランドプログラムから成る。この形式の長所は、まずユーザ空間のみの実装に比べて自由度の高いチューニングが施せる点が挙げられる。次に、どのようなチューニングが為されるのか判断を行うべき部分をユーザランドプログラムが受け持つことで細やかなチューニングが可能となる。例えば、Web100 では第 3.4.2 項で述べたように TCP-MIB データベースを参照することによってチューニングを行っている。また、Enable のように蓄積型データベースの情報を利用するチューンニング手法も可能である。このように、ユーザランドとカーネルランド双方での実装は前二者に比べて、研究分野としては高い発展性

がの望まれる。事実、第 3.4.3 項で紹介した WAD はチューニング手段の一つとして Web100 Kernel を利用しているが、これは本来、Web100 の制作者は意図していなかった利用方法である。つまり、あるチューニング技術を二次利用して、新しいチューニング技術を産み出すことができるのである。

利用形態

チューニング技術が施された実装を利用する場合に、実際にどのような処理が必要とされるのかという視点で分類を行った。

バイナリレベルで固定のものは、ソースコードレベルで改変を行った後は、挙動に対して何らかの変更を及ぼすことはできない。チューニング技術をユーザが意識する必要がないため利便性は高いが、環境に合わせて柔軟にチューニングを行うような用途では使用できない。

NLANR FTP や drsFTP は、ユーザがバッファサイズを明示的に指定する必要がある。そのためにユーザはネットワークチューニングの知識を必要である。さらに、知識を持っている場合でも、そのネットワーク環境に最適な値を求めるのは煩雑な作業である。

チューニングの判断が自動で処理されるものは、ユーザが意識することなく利用できる。しかし、機械的な判断であるため、熟練した技術者が手動でチューニングを行うケースには及ばない場合がある。

3.7 既存研究の問題

表 3.2 で分けたように、TCP チューニング技術を転送性能の視点ではなく、利用形態から視た場合は、その特徴によりエンドユーザに提供される機能性が大きく異なる。

これまで、TCP チューニング技術の比較研究 [18] は行われてきたが、どのような形でチューニング技術が提供されるのか議論されることはなかった。また、特定のネットワーク環境に依存しているため、他のネットワーク環境での利用に関して深く言及もされていない。

第 3.4.3 項で取り上げたように、WAD では TCP コネクションを特定するトランスポートアドレスとそれに合致する場合に使用するチューニング方法の組み合わせを記述するのみで様々なチューニング技術が利用できるようになっている。このように、WAD ような機構の存在によって初めて、TCP チューニング技術を通信環境に合わせて選択するという手法が実現可能となった。

しかし、図 3.4 のように、WAD ではユーザに対するインターフェースは設定ファイルとして統一的に整理されているが、WAD が TCP コネクションに対してチューニングを施す部分のインターフェースは規定されていない。

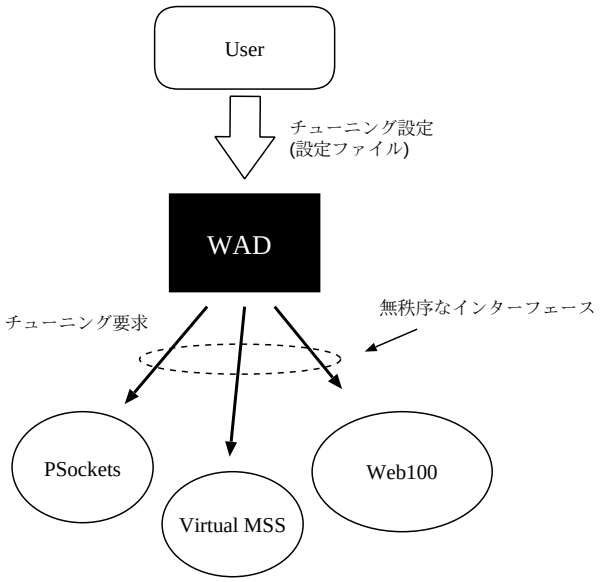

図 3.4: Work Around Daemon のインターフェース

このため、例えば PSockets というチューニング技術を使うべき状況を識るには専門知識を要する。つまり、WAD の記述方式である「TCP コネクションの識別子＋チューニング技術名称」というチューニング条件の指定では、新しいチューニング方式が増えるにつれて最適なチューニングを施すのに必要な知識が増大する。チューニングを行うのに必要な情報とは、チューニング技術の名称ではなく、そのチューニングによってどのような効果が得られるのかということである。つまり、「TCP コネクションの識別子＋高速転送」のように、チューニングによる効用によって指定されるべきである。

第4章　設計

前章では TCP の性能問題に対する既存の解決策を紹介し、それらでは未解決の部分に関して言及した。この章では、本研究による解決策を提示し、本解決策の設計について述べる。最後に本設計に必要な用件についてまとめる。

4.1　本研究による解決

TCP には、他のホストとの通信互換を提供する機能と通信効率を提供する機能の二面性がある。通信効率を担う部分は、前章までで述べたようにエンド間の通信環境によって、最適な TCP 挙動が異なる。

本研究では、まず TCP より通信効率を担う部分を切り離し、これを場合によって切り替えることを提案する。これをモジュール化し、カーネル内での扱いが容易な形式にすることを目指す。次に、アプリケーションを改変することなくユーザがこれらのモジュールを選択できる仕組みを用意する。予め定められた選択記述に従い、TCP は使用するモジュールを決定する。最後に、モジュール化機構が成立するための条件を整理し議論を行い、本研究が提示するモデルの長所と短所を整理する。

4.1.1　TCP 中の通信効率部分

第 3.2 節では、TCPVegas や RFC3390 のように TCP プロトコル互換性に依らない、通信効率を担う部分の改変について述べた。TCP には、このように転送効率を担うアルゴリムや変数演算を行う部分が存在する。標準的な TCP にも搭載されているものでは、輻輳ウィンドウ制御 [11] や高速リカバリアルゴリズム [19] などがある。

トランスポート層プロトコルである TCP は、ネットワークアプリケーションとネットワーク層プロトコルである IP との中間に位置し、双方の間でデータの入出力を行う。TCP はデータ入出力のためを送受信バッファを持っている。そのため、上位層や下位層との入出速度を調整することが可能である。この調整を行う部分が前段落で述べた転送効率を担う部分である。輻輳制御の範囲内で、送受信量に強弱を調整したり、送出タイミングを調整することでチューニングが行える。TCP が送受信量を調整する仕組みを簡単に模式化すると、図 4.1 のように

なる。

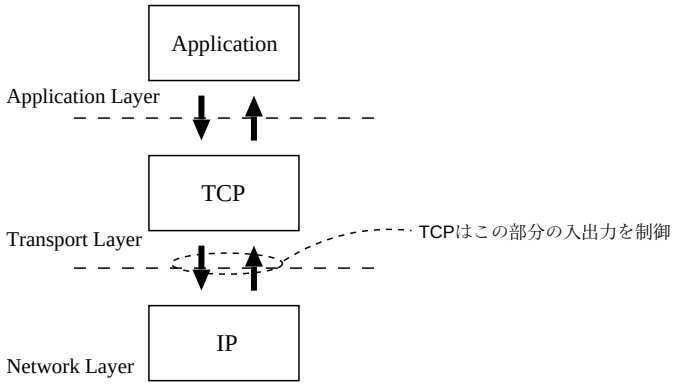

図 4.1: TCP による送受信量の調節

本研究では、このような転送効率を担う部分を TCP 本体から切り離しモジュール化を行う。モジュールとして切り離すことで図 4.2 のように TCP は基底部分とモジュール部に分離できる。さらに、第 3 章で述べたようなチューニング技術を利用したモジュールを複数用意する。複数のモジュールをモジュール群として扱い状況に合わせて切り替え可能なモデルを提供する。

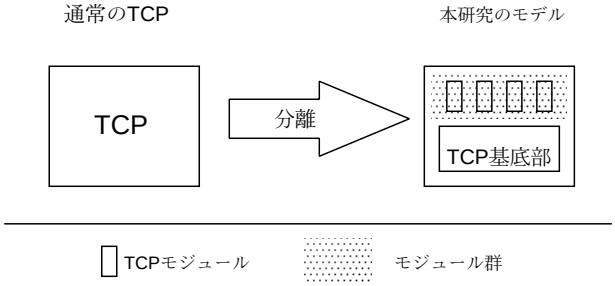

図 4.2: TCP 通信制御部分のモジュール化

4.1.2 コネクションごとのチューニング

前項で述べた通り、本研究が提案する通信効率化はユーザ空間ではなくカーネル空間で行われる。第 3.5 節で議論したように、ユーザ空間における実現方法では、チューニングが行われる主体は特定のアプリケーションとなり、ユーザにとって明示的である。そのため、どの TCP コネクションの効率化を果たすのか選択する機構は必要としない。それに対して、カーネル空間における実現方法では、すべての TCP コネクションに影響を及ぼしてしまうため特定の環境に依存し易い。そのため、アプリケーション非依存性とネットワーク環境非依存性を両立さ

せるには、TCP コネクションを選別し、チューニングを施すべきものとそうでないものの判断が可能な機構が必要となる。

本研究では、TCP コネクションとそれに応じたチューニング動作をシステムが判断するために対応表を用意する。対応表は以下の要素から成る。

- TCP コネクションを区別する記述子
- 使用するチューニング手段を区別する記述子

TCP コネクションを区別する記述子

TCP コネクションを一意に区別する記述子は、トランスポートアドレスである。トランスポートアドレスは通信元の IP アドレスとポート番号、宛先の IP アドレスとポート番号の 4 つの要素から成る。これは、第 3.4.3 項で紹介した WAD と同様である。

使用するチューニング手段を区別する記述子

前項に述べたように、本設計ではチューニングは TCP モジュール群が担っている。そのため、チューニング手段の選択はモジュールの選択によって行うことができる。本システムでは、モジュール識別子によって、これに当てる。

4.1.3 対応表による判別

本設計は、前節で述べたトランスポートアドレスとモジュール識別子から成る対応表を元に、選択すべきチューニング手段を決定する。そのためには、新しい TCP コネクションを検知し、対応表を引き、使用するモジュールを決定し通知する機構が必要となる。

4.2 システム全体像

前節までで述べたシステムの全体的な動作を示す。図 4.3 から図 4.6 までは時系列に沿って、本システムの基本的な挙動を示している。

まず図 4.3 は定常状態のシステムの様子である。前節で述べた対応表は、カーネル内部に保持される。ユーザ空間に存在する設定書き込み機構によって、ユーザ空間とカーネル空間との間に用意されたインターフェースを利用して設定が書き込まれる。

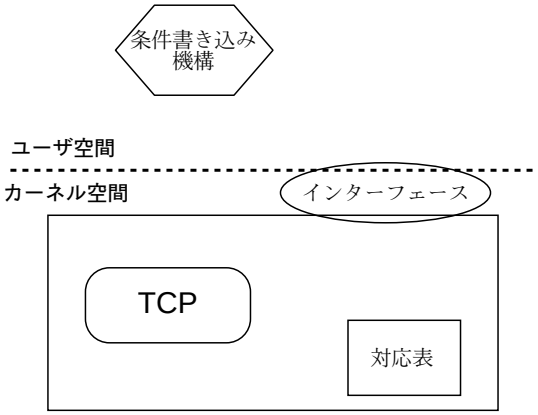

図 4.3: システム全体像 1

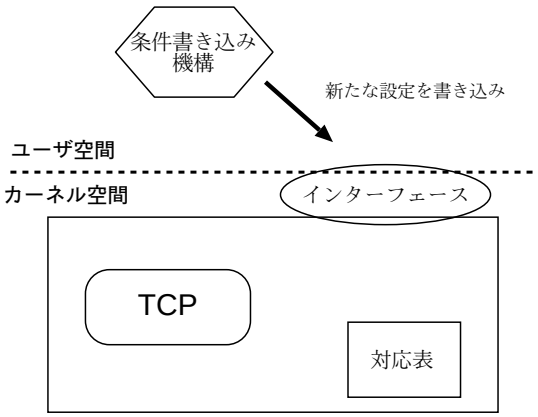

図 4.4: システム全体像 2

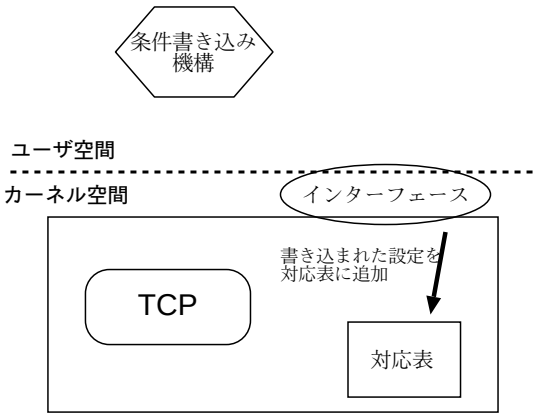

図 4.5: システム全体像 3

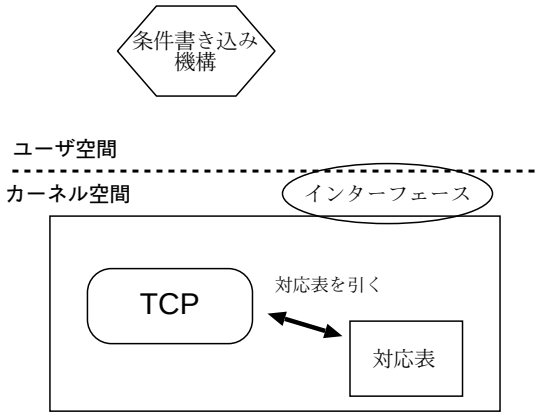

図 4.6: システム全体像 4

そして、新たな設定項目を書き込む場合は、書き込み機構がインターフェースを通して書き込む様子が図 4.4 である。通常、ユーザ空間からカーネル空間内部の変数を操作することは出来ない。そのため、書き込み機構が対応表に入力を行うためのインターフェースを用意する必要がある。

さらに、インターフェースで書き込まれた値を、カーネル空間内部の対応表に追加すると、図 4.5 となる。図 4.3 から図 4.5 まで完了することで、システムは対応表を更新できたことを意味する。

新しい TCP コネクションが発生する場合は、図 4.6 のようになる。TCP が通信先とコネクションの初期化を完了する前に、対応表を引くことで、現在行われつつある通信に合致するかどうかを調べる。合致しない場合は、通常通りの通信が行われる。合致した場合は、該当する TCP モジュールを利用して通信が行われる。

4.3 設計要件

4.3.1 想定する利用環境

本研究は、OS 環境として UNIX 互換 OS などの TCP/IP ネットワーク機能を備えるものを想定する。

さらに計算機環境として、PC のようにソフトウェアの書き換えで OS に対する変更が行えるものを前提として議論を進める。

ネットワーク環境として、インターネットに代表される TCP/IP プロトコルで作動しているネットワーク環境を、さらにはホストが扱えるネットワークデバイスとしてインターネットで一般的に利用されるものを想定する。転送速度を向上させるためにハードウェアレベルで特殊

な制御が為されているようなネットワークは想定外とする。

4.3.2 まとめ

ここでは本システムの設計要件をまとめる。
本システムに必要な要素を列挙すると次の通りである。

- 条件書き込み機構
- カーネル空間との入出力用インターフェース
- カーネル空間内部の対応表
- TCP 基底部とモジュール群

以下で上記の各要素について述べる。

条件書き込み機構

ユーザが意図するチューニング条件を対応表が定める形式に変換し、その値を書き込むアプリケーションプログラムが必要である。チューニング条件は、人間が取り扱うものであるため可読性が高いものが望まれる。これをシステムが解する記述へと変換しなければならない。

カーネル空間との入出力用インターフェース

ユーザ空間からカーネル空間内部の情報を読み書きすることは、UNIX 互換 OS では禁止されている。そのため、ユーザ空間に位置するアプリケーションプログラムが、カーネル空間内部に位置する値を変更する場合はカーネル側に変更を施す必要がある。本研究では、上記の条件書き込み機構より書き込まれる値をカーネル内部の対応表に反映するインターフェースが必要となる。

カーネル空間内部対応表

カーネル内部には、トランスポートアドレスとモジュール識別子が組となった対応表を用意する。TCP は、対応表を引いて自らの挙動を決定するため、この表は TCP より参照可能なようにしなければならない。

TCP 基底部とモジュール群

TCP 本体より通信効率化部分を切り離してモジュール化する。前述の対応表を引いて合致した場合は、指定のモジュールを使って効率化の処理を行う。表に合致しない場合は、OS が標準で提供する TCP の挙動を行う。

第5章　実装と評価

第 4 章では、本研究の設計について述べた。この章では、設計に基づき実装を行う。

5.1　実装

5.1.1　実装環境

第 4.3 節の設計要件で述べた通り本設計は UNIX 互換 OS のアーキテクチャを想定している。本実装では、UNIX 互換 OS の中でも Linux を採用した。採用理由を次ぎに列挙する。

- 一般に広く使われている UNIX 互換 OS である。
- 入手しやすくソースコードが公開されている。
- loadable kernel module アーキテクチャを持つため、カーネルに機能を追加するのが容易である。
- Proc ファイルシステムや Netlink などのカーネル空間とユーザ空間とのデータ入出力を行うための仕組みが提供されている。

本実装を行った計算機環境をまとめると表 5.1 の通りとなる。

Kernel	Linux 2.4.19
libc	glibc 2.2.5
C コンパイラ	gcc-2.95.3
CPU	Intel PentiumIII 1GHz
RAM	512MB

表 5.1: 実装を行った計算機環境

5.1.2 本研究における実装

第 4.3 節の設計要件を満たすために以下に挙げるソフトウェアの実装を行った。

- 条件書き込み機構を実現するための設定用ツール
- カーネル上で作成したユーザ空間に対するインターフェースとカーネル空間の対応表
- TCP モジュール群

設定用ツール

本実装では、チューニング条件をユーザ空間からカーネル空間に書き込むための設定用ツールとして、atconfig コマンドを作成した。次節で用意される入出力用のインターフェースを開いて設定を書き込むことが可能である。

次の図は、設定用ツールをコマンドラインより使用した様子である。

設定用ツール使用例

```
% atconfig -s 0.0.0.0.80 -m 3
% atconfig -d 192.168.0.15.0 -m 1
% atconfig -s 0.0.0.0.20 -m 3
% atconfig -s 133.27.4.213.0 -d 133.27.193.23.0 -m 2
% atconfig -s 133.27.4.213.0 -d 133.27.193.23 -m 0
```

atconfig コマンドでは、's' オプションで送信元トランスポートアドレス、'd' オプションで宛先トランスポートアドレス、'm' オプションで使用する TCP モジュール番号を指定する。'm' オプションに 0 を指定するとエントリを消去できる。

入出力インターフェースとカーネル空間の対応表

カーネル上で作成したユーザ空間に対するインターフェースには、Linux の Proc ファイルシステムを利用した。これはメモリ上に仮想のファイルエントリが用意され、仮想ファイルに対する read システムコールや write システムコールが発行されると予め登録されたカーネル内部の関数がフックされ処理が行われる。本実装では、書き込みが行われるとユーザ空間から値を受け取り対応表に書き込まれむ。

対応表を模式化すると、表 5.2 のようになる。

フィールドの値が'0' な場合は、条件指定が無しの部分である。TCP をこの対応表を引き、4 つのフィールドである送信元 IP アドレス・送信元ポート番号・宛先 IP アドレス・宛先ポート

送信元 IP アドレス	送信元ポート番号	宛先 IP アドレス	宛先ポート番号	モジュール識別子
0	80	0	0	3
0	0	192.168.0.15	0	1
0	0	0	20	3
133.27.4.213	0	133.27.193.23.22	0	2

表 5.2: カーネル空間の対応表の一例

番号と、現在のソケットのそれとが合致するかどうか調べる。合致した場合は、モジュール識別子のフィールドに指定された番号の TCP モジュールを使用して TCP コネクションを継続する。合致しなかった場合は、OS が標準で用意している TCP の挙動で作動する。

TCP モジュール群

本実装では、TCP において出入力量を調整する演算を行う関数から TCP モジュール内部の関数を呼び出す。図 5.1 は、TCP が対応表を引き、対応するモジュールを使用して演算を行う様子である。

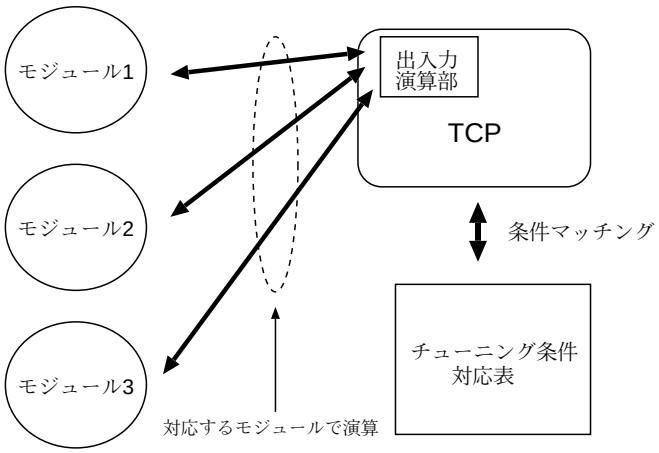

図 5.1: 演算処理にモジュールを選択

本実装によって改変された TCP は、コネクションを開始時にチューニング条件を記した対応表を引く。条件が合致するば、モジュール番号をフラグとしてセットする。

TCP が、フロー制御やリカバリ制御などの出入力制御を行うとき、通常の TCP は変数演算を行う。本実装では、演算を行う前にフラグを調べ、それにより対応する TCP モジュールを

選択する。

5.2 評価

5.2.1 本実装を利用したチューニング例

ここでは、本実装を利用するとどのようなチューニング指定が可能となるのか議論を行う。

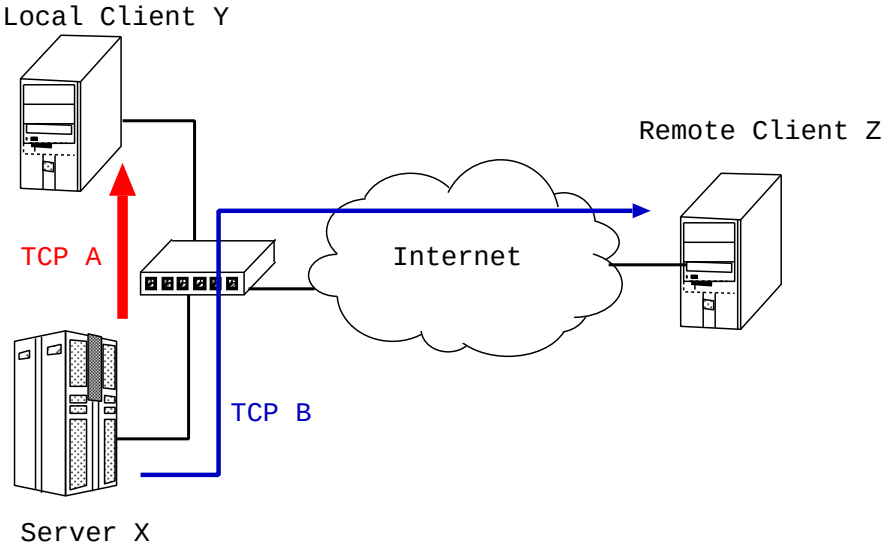

図 5.2: 宛先による TCP 挙動の切り替え

図 5.2 は、ホストマシン'Server X'（以下'X'）において本実装が動作している。今、'X' は同一サブネット内のホストマシン'Local Client Y'（以下'Y'）とインターネット経由で接続されているホストマシン'Remote Client Z'（以下'Z'）と通信を行う。'X' では、Web サーバや FTP サーバなどのクライアントの要求に対してデータを送出するタイプのサービスが動いている。

送信元 IP アドレス	送信元ポート番号	宛先 IP アドレス	宛先ポート番号	モジュール識別子
0	0	Y	0	s
0	0	Z	0	t

表 5.3: Server X 上の対応表

'X' 上の対応表が表 5.3 の通りであるとき、'X'→'Y' のコネクション'TCP A' は TCP モジュール's' を使用し、'X'→'Z' のコネクション'TCP B' は TCP モジュール't' を使用する。

モジュール識別子's' に対応する TCP モジュールを高速転送タイプに、モジュール識別子't' に対応する TCP モジュールを通常タイプに設定すれば、本実装により同一サブネット内のホストには高速通信で、インターネット経由のホストには通常の TCP で通信が行える。この時、'X' 上で動作する Web サーバなどのサービスデーモンには変更を施す必要がない。

本実装では、アプリケーションに非依存で TCP コネクションごとに個別のチューニング処理ができることが分かった。

5.2.2 WAD との比較

本実装と似た機構を持つものとして、第 3.4.3 項で紹介した WAD がある。この項では、本実装と WAD との比較を議論する。

本実装と WAD の類似点を列挙すると次の通りである。

- トランスポートアドレスとチューニング手段の組み合わせでチューニングを指定できる。
- アプリケーションに非依存である。
- 様々なチューニング手法を同一ホストで同時に利用できる。

見かけ上、WAD は第 4.3 節でまとめた設計要件を満たすことが出来る。ここでは、本実装と WAD 実装モデルの違いから両者の長短を明らかにする。

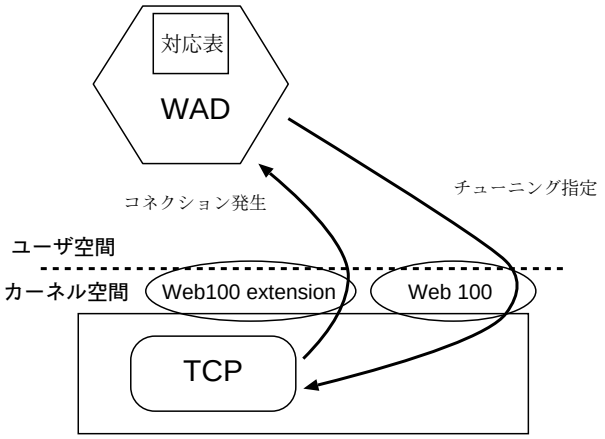

図 5.3: WAD によるチューニング選択

図 5.3 は、WAD がコネクションのチューニング手段を決定する様子である。本実装と WAD の大きな違いは、まず WAD では対応表を保持しているのはユーザ空間のアプリケーションだということである。新しい TCP コネクションが発生したとき、WAD モデルでは、まずカーネ

ル空間からユーザ空間に対して新しいコネクションの発生が通知される。そして、WAD は発生したコネクションのトランスポートアドレスなどを調べ自らが保持している対応表を引き、チューニング手段を決定する。チューニング手段を決定したら、その指定を Web100 カーネルの機能を利用して指定する。

WAD では、TCP コネクションが発生する度に、カーネル空間からユーザ空間への問い合わせとユーザ空間からカーネル空間への問い合わせが行われる。対して、本実装ではユーザ空間とカーネル空間での値の受け渡しは対応表を更新する時のみである。ユーザ空間とカーネル空間の間で行われる値の受け渡しは、OS のオーバーヘッドが大きい。Web サーバなどの同時に複数のコネクションが発生するホストでは、計算機の負荷が高くなる。

このようにオーバーヘッドが大きい方式を WAD が採用しているのは理由が存在する。WAD では、そのホストで保持している対応表の他に、リモートホストにあるデータベースに蓄積された統計情報をもチューニングを行う際の情報として利用できる。つまり、WAD はチューニング条件として利用される情報が自動的に切り替わる、つまり更新頻度が多いためユーザ空間に情報を保持している。

上記をまとめると表 5.4 の通りとなる。

	コネクションごとの最適化	チューニング技術の抽象化	処理の負荷	チューニング条件の自由度
本実装	○	○	○	×
WAD	○	△	×	○

表 5.4: 本実装と WAD の比較

第 4.3 節の設計要件の範囲内では、本実装の方が優れている。しかし、WAD の方が発展性が高い。

第6章　結論

本章では、結論として本研究のまとめと今後の課題について述べる。

6.1　まとめ

本研究では、TCP チューニング技術の利用が進まない点を明らかにし、その問題を解決するため議論を行った。

通信環境における格差が増大することにより、その環境に特化することで転送性能を向上させる余地が生まれた。しかし通信環境に特化することで高性能化を果たしたチューニング技術では、損害無く利用できる通信環境が限定されてしまう。さらに通信環境に合わせて利用すべきチューニング技術を選択するのは困難である。

そのため本論文では、既存のチューニング技術を分類し、それよって得られる転送性能上の効用によって再構成を行った。そしてチューニング技術を TCP モジュールという形で提供し、アプリケーション非依存な TCP チューニングが行えるようにした。さらに既存のチューニング技術を、TCP コネクションごとに TCP モジュール選択が可能な機構を提案し実装した。この機構により、通信帯域や通信先ホスト、ネットワークアプリケーションの形態などネットワークを取り巻く環境に合わせて TCP の挙動を切り替えることが可能となった。加えて単一のホスト上で複数のチューニング技術を利用者がより分かりやすく利用できるようになった。

よって、TCP チューニング技術の普及を妨げる一要因を解決できた。

6.2　今後の課題

表 5.4 では、本実装は WAD と比べ、チューニング手段を判断する部分での自由度が低かった。

本研究では、チューニング手段を選択する処理においてエンドホスト内で完結していたが、図 6.1 のように、WAD や Enable で採用されている外部のデータベースから情報を取得する方式により発展性が高まる。

また、第 3.1.2 項で紹介した drsFTP のようにデータ転送を行う TCP コネクションとは別にコントロールセッションでお互いのバッファ状態を相互に通信し送出量を調整するモデルもあ

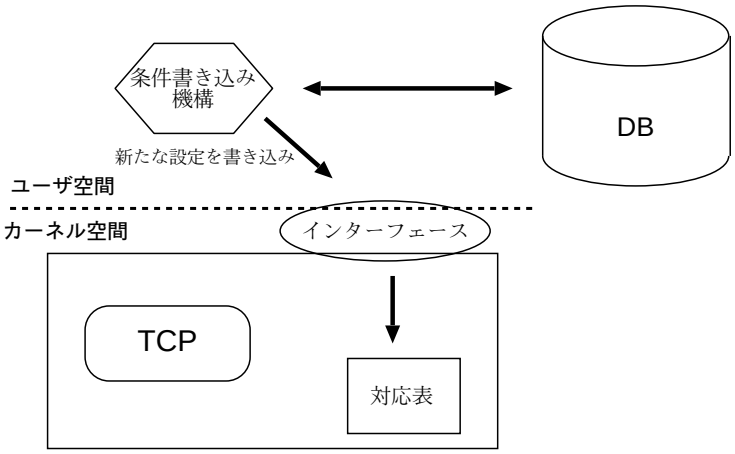

図 6.1: 外部データベースを利用したモデル

る。アプリケーション非依存なチューニング選択機構で同様の効果を得るには、両エンドにおいて常駐プログラムが動作しお互いのバッファ量を相互に通信するモデルが考えられる。

謝辞

本研究を進めるにあたり、主査である慶應義塾大学環境情報学部教授村井純博士に感謝します。また副査である慶應義塾大学環境情報学部助教授中村修博士と東京大学大学院情報理工学系研究科助教授江崎浩博士に感謝します。

絶えず指導して下さった慶應義塾大学環境情報学部助教授楠本博之博士と慶應義塾大学環境情報学部専任講師南正樹氏に感謝します。また、常に近くで助言を下さった総務省通信総合研究所の杉浦一徳氏、慶應義塾大学政策・メディア研究科小川晃通氏をはじめとする STREAM 研究軍団の諸氏に感謝します。論文執筆を陰で支えてくれた臼井健氏、本波友行氏、三屋光史朗氏、小畠元氏に感謝します。論文執筆者を惜しみない尽力と炊き出しで支えてくれた徳田・村井・楠本・中村・南合同研究室のすべての先輩方と後輩のみなさんに感謝します。

苦楽を共有し、励まし合い、時には足を引っ張りあった、海崎良氏、豊野剛氏、三川荘子氏、小川浩司氏、西村祐貴氏、菅沢延彦氏、石原知洋氏、若山史郎氏、梅染充男氏、村瀬正名氏、石井かおり氏、青木崇行氏、桐原幸彦氏、松宮健太氏、間博人氏に感謝します。

素晴らしい OS の実装を公開し、全世界のコンピュータ科学研究者に寄与している、すべての Linux 開発者に深い敬意を表します。

以上をもって謝辞とします。

参考文献

[1] R. Braden, 'Requirements for Internet Hosts – Communication Layers', RFC1122, October 1989

[2] V. Jacobson,R. Braden,D. Borman, 'TCP Extensions for High Performance', RFC1323, May 1992

[3] NLAR, 'Auto Tuning Enabled FTP Client And Server', URL:http://dast.nlanr.net/Projects/Autobuf/

[4] EU DataGrid Project, URL:http://www.eu-datagrid.org/

[5] Brian L. Tierney, Dan Gunter, Jason Lee, Martin Stoufer, 'Enabling Network-Aware Applications', Proceedings of IEEE HPDC- 10'01, August 2001

[6] Mark K. Gardner, Wu-chun Feng, Mike Fisk, 'Dynamic Right-Sizing in FTP (drsFTP): Enhancing Grid Performance in User-Space', IEEE Symposium on High-Performance Distributed Computing HPDC-11, July 2002

[7] Jeffrey Semke, Jamshid Mahdavi, and Matthew Mathis, 'Automatic TCP Buffer Tuning', Computer Communications Review, a publication of ACM SIGCOMM, volume 28, number 4 October 1998

[8] Lawrence S. Brakmo,Sean W. O'Malley,Larry L.Peterson 'TCP Vegas: New techniques for congetion detection and avoidance', Proceeding of ACM SIGCOMM '94, May 1994

[9] J. Mo, R. J. La,V. Anantharam, and J.Walrand, 'Aalysis and comparison of TCP reno and vegas' Proceedings of INFOCOM '99, March 1999.

[10] M. Allman,S. Floyd,C. Partridge, 'Increasing TCP's Initial Window', RFC3390, October 2002

[11] M. Allman, 'TCP Congestion Control', RFC2581, April 1999

[12] W. Stevens, 'TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms', RFC2001, January 1997

[13] H. Sivakumar, S. Bailey, R.L. Grossman, 'PSockets: The Case for Application-level Network String for Data Intensive Application using High-Speed Wide Area Networks', IEEE Computer Society Press Article No. 37 January 2000

[14] Web100 Project URL:http://www.web100.org/

[15] Tom Dunigan, Matt Mathis, Brian Tierney, 'A TCP Tuning Daemon', SC2002, July29, 2002

[16] Internet2 End to End Performance Initiative, URL:http://e2epi.internet2.edu/

[17] K. McCloghrie, 'SNMPv2 Management Information Base for the Transmission Control Protocol using SMIv2', November 1996 RFC2012

[18] Eric Weigle, 'A Comparison of TCP Automatic Tuning Techniques for Distributed Computing', IEEE Symposium on High-Performance Distributed Computing, February 2002.

[19] W. Stevens, 'TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms', RFC2001, January 1997