修 士 論 文

TCP/IP 輻輳制御の性能改善に関する研究

指導教官　篠田 陽一　助教授

北陸先端科学技術大学院大学
情報科学研究科情報システム学専攻

大嶋 健司

1997 年 2 月 14 日

要 旨

本稿では,TCP/IP におけるパケット輻輳制御について従来の実装における問題点を指摘する. 次に, 途中経路の状態を観測するために妨げとなっている受信側の遅延確認応答を送信側から制御する方式の提案をおこなう. RTT の比較実験によりこの方式の影響を評価し, データ転送性能のさらなる向上のための手法を示す.

目 次

第 1 章

はじめに

1.1 研究の背景

Transport Control Protocol(以下 TCP)[23] はインターネットでのデータ転送の主要な部分を担っている. これは TCP がコネクション指向であり, 信頼性のあるストリーム配送を実現しているからである.

TCP では Jacobson らによる研究によって効率の良いパケットの流量制御が実現されている. これは受信側が告知する受信バッファの空き容量であるウィンドウサイズを元にして, 送信側が送信量を調節するものである. また受信側が返す受信確認が, データの到達を保証すると同時に, 送信するタイミングにも用いられ, ネットワークにながれるデータの流量を自動的に調整することが出来る. また TCP では, パケットが往復する時間 (RoundTripTime) について観測し, 輻輳を発生させないような再送 timeout 時間の決定にも用いられている.

データを受信した確認をすぐに送り返すと, 確認をするだけのパケットが発生し, データ転送能力を低下させてしまう. そこで, 確認の応答を遅延させて, 複数の確認を 1 つにまとめることで TCP ヘッダによるオーバーヘッドを低減する. しかしこの確認応答の遅延が流量制御や輻輳制御に悪影響を与えていることがしばしば指摘されている.

1.2 目的

本研究の目的は従来の TCP における流量制御, 輻輳制御の検証をし, 性能低下の大きな原因となっている受信側の確認応答の遅延を送信側から制御する方法を提案して, 性能の改善を計ることである.

また,RTT 計測における遅延確認応答による影響をなくすことでネットワークの状態変動を正確に計測し, 流量制御に利用する方法を提案する.

そして, 従来の輻輳制御アルゴリズムや広域ネットワークに対する影響を考察し, 新たな流量制御の設計を行う.

1.3 論文の構成

まず,2 章では TCP における流量制御, 輻輳制御アルゴリズムについての説明を行なう. 3 章では従来の実装で性能向上の弊害であった遅延確認応答 (Delayed ACK) について, その問題点を明らかにする. 4 章ではその Delayed ACK を送信側から制御する CDL フラグを TCP のヘッダに新たに設定し, その制御方法の設計を行なう. 5 章では CDL フラグによる制御の効果をデータ転送の比較実験より確認する. 6 章では CDL フラグの利用によって実現された正確な RTT 計測によって, タイマの粒度を上げて, ネットワークの状況を更に詳しく把握する方法について取り上げる. 7 章ではこれらの方式が従来の TCP の輻輳制御や広域ネットワークに与える影響について考察する. 8 章で論文のまとめと今後の課題について述べる.

第 2 章

TCP について

2.1 TCP

TCP(Transmission Control Protocol)[23] は信頼性を保証したコネクション指向のプロトコルである. インターネットのトランスポートレイヤでは重要、かつ一番利用されているプロトコルであるである.

TCP では, ベースとなるデータリンクレイヤのシステムに関する仮定をほとんど行っていないので, さまざまなパケット配送システムの上で柔軟に対応することが出来るようになっている. たとえば低速のダイアルアップ回線からローカルエリアのネットワーク (LAN) だけでなく, 広域エリアネットワーク (WAN) や超高速のネットワークシステムに対しても, そのことをユーザ側が意識することなく利用することが出来る.

2.1.1 受信確認応答

TCP はデータの連続性や順序付けを保証し, 信頼性のあるストリーム転送サービスを実現するために, 受信確認応答 (Acknowledgement:以下 ACK) を用いている.

TCP ではデータストリームをオクテット, またはバイトの列とみなしている. 転送のためにデータをセグメントごとに分割し,TCP のヘッダ情報を付加して, 1 つの IP データグラムのパケットとしてデータリンクへ流している.

パケットの順序をコネクションのそれぞれの方向において決定するために TCP では 32bit からなるシーケンス番号を用いている. シーケンス番号はデータの最初からのバイト

数にコネクションを確立した際に任意の値で設定した初期番号 (Initial Sequence Number) を加えたものである.

送信側は TCP ヘッダに, これから送信するパケットのシーケンス番号, これまでに相手からの受信を確認したデータの最大のシーケンス番号である確認応答番号 (Acknowledge Number) を記録して送信する。受信側は今までに送信したデータのシーケンス番号と, 相手から送られて来る受信確認の ACK 番号を比較することで, 相手に送信したデータグラムがきちんと到着したか, 確認をすることが出来るようになる.

TCP はたいていは IP 層の上で運用されているが, IP 層自身ではパケットがきちんと届いたか保証する機構がない. そこでデータの順序付け, 重複部分の切捨て, 失われた部分の再送要求の機能を TCP が受け持つことになる.

2.1.2 流量制御

ここでは効率の良いデータ転送をするためのフロー制御を実現方法について述べる.

受信側がその時点での受信出来るバッファの空き容量をウィンドウという. それぞれのパケットの TCP ヘッダにはウィンドウのサイズ (バイト数) を告知している.

受信側からの ACK を確認するにしたがって, ウインドウはシーケンス番号の上を進んでいくことになる. これをスライディングウィンドウ方式と呼ぶ.

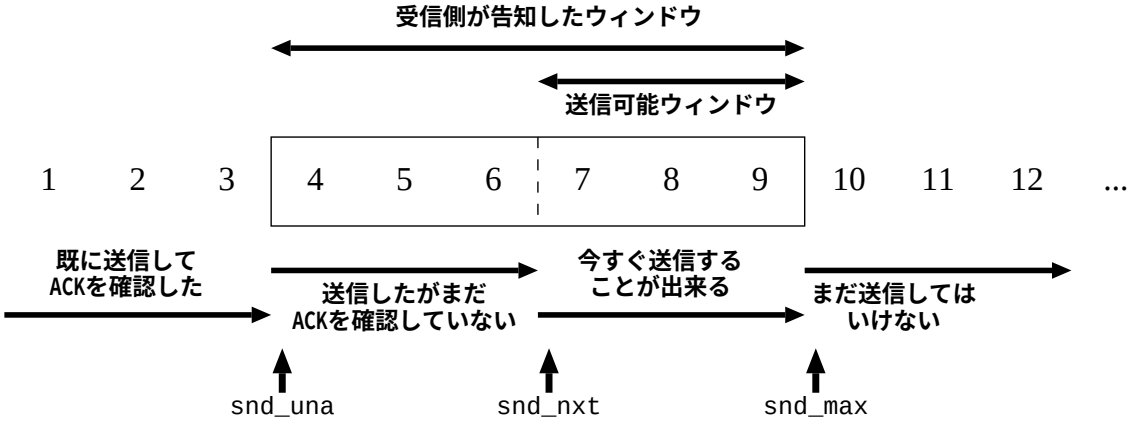


図 2.1: スライディングウィンドウ方式

このスライディングウィンドウ方式では 3 つのポインタの扱いが重要になる. 1 つ目のポインタ (snd_una) はウィンドウの一番左側で, まだ送信されたがまだ ACK がされてい

ないデータの一番始めである. 2 つ目 (snd_nxt) はウィンドウの右側で,ACK が受信されなくてもすぐに送信することができるデータの始めである. 3 つ目は (snd_max) ウィンドウのさらに右側で, 送信することが出来る最大のデータである. スライディングウィンドウ方式にしたがって送信側は, ACK 番号から受信側が告知したウィンドウサイズを加えた分までのデータグラムを送信することが許される.

2.1.3 Self-clocking メカニズムと輻輳

TCP の Self-clocking メカニズムはデータフロー制御の根底にあるものである.

接続の状態が安定している場合には, ウィンドウ一杯までデータの転送を続けることが出来る. その状態ではネットワーク上からパケットを 1 つ転送し終った時を, 新たにパケットを 1 つだけ送信することが出来る印であると見ることが出来る. このことから受信側からの ACK は, 新たな送信を決定するものとみなすことができる.

この考えは同時に輻輳制御の条件を満たすことが出来る. ネットワーク上において, 通信帯域以上のパケットが回線に流れ込んだ場合には, 一旦手前のルータの送信キューのメモリにパケットが格納される. 大抵は FIFO で順番に回線に流されるようになっているが, キューのメモリ容量以上にパケットが流れ込んだ場合は, そのルータはパケットを破棄してしまう. これが輻輳の発生である.

データ転送は途中経路上で一番回線速度の遅い部分が律速になる. 輻輳が発生してパケットが失われた場合は, 相手からの ACK が送られて来ないので, 送信側からこの分のパケットを再送する必要がある. しかし輻輳が回復しないままに再送のタイミングを早めてしまうと, ふたたび無駄なトラフィックを発生してしまい, 輻輳を助長してしまうことにもなる. ここで送信側はホスト毎になんらかの輻輳を制御する機構が必要となる.

受信側から来る ACK のスピードは, データを送信したときに一番遅い, ボトルネックにあたる回線で転送能力を一杯に使用した状態とみなすことが出来る. よって相手からの ACK が来るまで送信を止めることで, このボトルネックに流れ込むデータの量が抑えることにより, 輻輳を避けることが出来るというアイディアに基づいている.

2.1.4 Delayed Acknowledgement

受信側が送信する ACK のパケットは, たいていの場合は TCP と IP ヘッダしかなくデータが乗っていない,"pure ACK" である.

大抵の TCP の実装では, データを受信してもすぐには ACK を送り返さない. これは ACK の送信を一定時間まで待ってみて, もしその間に新たにデータを転送する必要が生じたらそのパケットと一緒に ACK を送って (piggyback) しまうことで, データ転送には関係のないパケットの送信を極力減らそうとするものである. これを遅延確認応答 (以下 Delayed ACK) という.

Delayed ACK のタイミングは実装によって決まるが, 大抵は`telnet` のような対話的なアプリケーションの反応に影響するので, 人間の反応速度である 200ms に設定されていることが多い. また Delayed ACK の機構を起動させないように受信側から設定することも可能である. たとえば Window System のマウスのポインタ情報のような応答に素早い反応が要求される場合には, `TCP_NODELAY` フラグを設定することで, すぐに ACK を返送することが出来る.

2.2 輻輳制御の実装

ここでは,4.3BSD TCP,4.3BSD Tahoe TCP,4.3BSD Reno TCP,4.4BSD と発展した TCP の輻輳制御の実装について説明する.

2.2.1 4.3BSD

4.3BSD 以前の TCP では, 輻輳制御として TCP タイマアルゴリズムが用いられてきた. このアルゴリズムはパケットを送信した際にタイマを設定し, ACK が返ってこない場合でも timeout 時間に達するまでは再送を行わないものである.

TCP ではパケットの往復時間 (Round Trip Time:以下 RTT) を計測している. これはパケットを送信すると同時にタイマを起動し, そのパケットに対する ACK が確認されたところでタイマの経過時間を計測する.

RTT は同時に 1 つのパケットにしか計測を行わない. そこで少ない計測値で RTT を精度良く推定するために Smoothed RTT 評価関数 (`srtt`) を定義する.

RTT の計測が出来た時に, この`srtt` は一種のローパスフィルタで更新される.

$$srtt \leftarrow \alpha\, srtt + (1-\alpha)RTT \tag{2.1}$$

αは smoothed factor であり, 通常は 0.9 が使われる. つまり新しい`srtt` は前の値を 90%使い, 10%新しい RTT の計測値が影響する.

srtt をもとに再送までの timeout 値 (Retransmission TimeOut:以下 RTO) を決定する.

$$RTO = \beta\ srtt \tag{2.2}$$

βは delay variance factor であり, 通常は 2 が使われる.

Jacobson[1] は, 式 (2.2) の推定では混雑したネットワークでの RTT の大きな変動に対応することが出来ないとして, 計測した RTT の平均偏差 (rttvar) をとり, それを RTO に反映させる方法を提案した.

$$\begin{aligned} \Delta &= RTT - srtt \\ srtt &\leftarrow srtt + g\,\Delta \\ rttvar &\leftarrow rttvar + h\,(|\Delta| - rttvar) \\ RTO &= srtt + 4\,rttvar \end{aligned} \tag{2.3}$$

(係数 gは 1/8,h は 1/4 が使用される) 式 2.3 の計算は bit シフトと加算だけで実現でき, ネットワークの状態変動にも対応できるものである.

2.2.2 4.3BSD Tahoe/Reno TCP

1988 年に発表された 4.3BSD Tahoe release では, 輻輳制御として Slow Start(スロースタート),Congestion Avoidance(輻輳回避), Fast Retransmit と呼ばれる機構が追加された. 1990 年に発表された Reno release では Fast Recovery が追加された.

Slow Start と Congestion Avoidance

送信側は ACK の確認が取れた次のバイトから, 受信側が告知したウィンドウサイズまでのデータを送信することができる. コネクションが確立した時点でいきなりウィンドウサイズ分のデータを送信してしまうと, 途中経路で輻輳を発生させてしまうかもしれない. しかしネットワークで接続された 2 つのホストにおいて, 輻輳の状態や送信できる最大のウィンドウサイズなどの情報を送信側があらかじめ知ることは不可能である.

そこで, 送信側が検知したウィンドウの大きさをあらわす変数として輻輳ウィンドウ (cwnd) を導入する. 送信側は, 受信側が告知するウィンドウサイズと輻輳ウィンドウサイ

ズとの小さいほうの分だけデータを送信する. この輻輳ウィンドウを輻輳を起こさないように徐々に増やしていくことで送信量の上限を見つけていく必要がある.

まず, コネクションを確立した直後や, タイムアウトによって再送が発生したときには,

- cwnd を 1 セグメントに設定し, その分のパケットを送信してみる.
- ACK が送られて来たら,cwnd を ACK ごとに 1 セグメント分だけ増加させていく.

これを Slow Start アルゴリズムという. つまり cwnd は 1RTT ごとに指数的に増加することになる.

さて,cwnd を増加させていって, 実際のネットワークで送信が可能な量を越えた場合には, 輻輳が発生してパケットが失われてしまう. その分の ACK が返って来ない場合は RTO まで新たなデータの送信を停止することになる. timeout が発生した時点で, 輻輳の発生を検知することが出来る.

ここで, スロースタートしきい値 (ssthresh) という変数を導入する. 輻輳による再送が発生したら,

- ssthresh を直前のcwnd の半分に設定する.
- cwnd を 1 セグメントに設定して, 失われた部分のパケットを再送し, スロースタートからやり直す.

手順を取る.

もしcwnd がssthresh を越えた場合には, 輻輳が近いことになる. そこで輻輳の発生を遅らせるために cwnd の増加を ACK が 1 つ返ってくるたびに 1/cwnd 分とする. つまり cwnd は 1RTT ごとで 1 セグメントずつ, 線形に増加することになる.

つまりcwnd とssthresh を比較することで送信量の増加の具合を指数的, 線形増加の 2 種類を使い分けることで, 輻輳が起きないように送信量を制御し, 輻輳制御を行なうものである.

Fast Retransmit と Fast Recovery

TCP では, シーケンス番号が前後して, 順番通りではないセグメントを受信した時に, 重複した ACK 番号の ACK をすぐに返送するように規定している. この重複した ACK を Duplicate ACK(以下 dup ACK) という.

この dup ACK を受け取った際には, 送信したデータの途中のパケットが失われて生じたのか, 受信側がセグメントの並び替え中であったために生じたのか, または途中の経路で ACK のパケットが重複されて生じたのを送信側は判断することが出来ない. そこで, 同じ ID の dup ACK をそのまま受け取り, 決められた回数 (`tcprexmtthres`) まで待ってみることにする. (`tcprexmtthres` は実装では 3 回) dup ACK の回数がそのしきい値を越えた場合にはデータのパケットが失われたものと判断して, 再送タイマが timeout するのを待たずに再送を行うことにする.

このように受信側のセグメントの並び替えを許容し, 輻輳による単一のパケットの喪失を早い段階で検知して, 再送するアルゴリズムを Fast Retransmit という.

Tahoe TCP では dup ACK を 3 つ受け取って輻輳が検出した際には`cwnd` を 1 セグメントに設定して, スロースタートの手続きを一からやり直していた.

しかし Reno TCP ではこの部分に改良を加えている. dup ACK を 3 つ以上受け取った段階で,

- `cwnd` を直前の半分の値に設定して, 失われた分のパケットを再送する.
- さらに dup ACK を受け取った際には, その分はネットワーク帯域の余裕と判断して,`cwnd` に 1 セグメント追加する.
- 新しい ACK で再送したデータの配送確認ができたら, `cwnd` を`ssthresh` にセットし直して, 再送直後のスループットに戻す.

という手段を取る.

このようにして輻輳を回避するアルゴリズムを Fast Recovery という. Fast Retransmit と Fast Recovery では程度の軽い輻輳が発生しても転送速度を落すことなく, 速い回復を計ることが可能になる.

2.2.3 4.4BSD TCP 以降

4.4BSD 以降の OS では物理的なネットワーク性能の向上に対応するために, いくつかの TCP options が追加して設定された.

TCP Window Scale options

TCP Window Scale option は RFC1323[26] によって規定されたもので, TCP ヘッダで告知できるウィンドウサイズを拡大するものである.

受信側によって告知されるウィンドウサイズは, 受信側の設定によって決定されている。(BSD 系の OS ではデフォルト値が 4096 or 8192byte である) 受信バッファを広げると, 一度に多くのパケットを受信することが出来, 転送能力を向上させることが出来る.

1 つの通信路で送信側が 1RTT 間にネットワーク上に送り出すことが出来るパケットの総量は通信路のバンド幅 [Byte/sec] と RTT[sec] の積によって決定される. これを Bandwidth-Delay Product(以下 BDP) という. この BDP の大きさを検知し, 計測した RTT からそれにあわせて送信量を制御しようというものである.

ここで, 静止通信衛星をつかった衛星リンクを例に挙げる. バンド幅 2[Mbps] で約 500[ms] の RTT を持つ衛星回線では, 1RTT 間に送信することができるパケットの総量 (BDP) は 128k[Byte] である. このような広い通信帯域と長い RTT をもつ通信路を Long Fat Network(以下:LFN) と呼ぶ. LFN に張った通信は長いパイプに見立てて,Long Fat Pipe とも呼ばれる.

ところが受信側が TCP ヘッダで告知出来るウィンドウサイズ (`th_win`) は 16 ビット表記なので,65535byte までしか伝えることが出来ない. 1 本の接続だけでこの LongFatPipe の能力を半分程度しか使うことが出来ない. このように従来の TCP のヘッダ情報では告知できるウインドウの制限によってしばしば回線帯域を 100%使い切る通信をすることが不可能になる.

TCP Window Scale options はこのような制限を無くすために設定された. Window Scale option では接続を確立, 更新する際のみに送られる. ウィンドウサイズをシフトするための値が入れることができる. 通常は 0(スケールしない) で, 最大値は 14 である. このオプションを使えばウィンドウサイズには最大の 65355×(2 の 14 乗)Byte = 約 1GB までの値を設定できるようになった.

また, 超高速なネットワークに関連したところでは 32bit のシーケンス番号があっという間に一巡してしまう問題がある. 例えば 1G[bps] の通信帯域をもつ回線にデータを 100%流すと, 30 秒足らずでシーケンス番号を使い果たし, 一意性を保つことが出来なくなる. この問題は RFC1644[28] で TCP Connection Count option を規定し, パケットを送信した時間に関係した値をこのオプションに入れることで解決している. これを PAWS (Protection

Against Wrapped Sequence Number) という.

TCP Time Stamp options

Time Stamp Option は RFC1323[26] によって規定されたもので, RTT の計測を同時に複数行なうことで, 推定の精度を上げようとするものである.

従来の RTT の計測では 1 つの接続に対して 1 つのタイマしか起動しない. この方式では, 接続を開始した直後では計測できた RTT のサンプル数が少なく, 推定した RTT には誤差が大きく含まれている. また再送が発生した時の ACK に関しては, 再送する前のパケットの ACK か, 再送した後のパケットの ACK かを区別することが出来ないので, (Karn の再送曖昧性問題 [3])RTT の更新をしないことになっている.

そこで,TCP Time Stamp option では, TCP を初期化した際に起動され, 500[ms] 毎にインクリメントされるタイマ`ts_now` を使う。パケットを送信する際にはオプションの`ts_val` フィールドに`tcp_now` を入れて送る.

Time Stamp option に対応した受信側はオプションを受け取ると, `ts_val` フィールドを読み取って, 値を一旦`ts_recent` に保存する. Delayed ACK によって複数のパケットに ACK を返した時やパケットの到着が順番通りでなかったときに ACK を返した時に RTT が少なく見積もられることのないように, 番早く到着したパケットの`ts_val` 値が`ts_recent` に入れられることに注意する. 送信側へ ACK を送り返す際にはこの`ts_recent` の値を`ts_ecr` フィールドに入れて返信する. ACK を受け取った送信側は現在の時刻`tcp_now` と`ts_ecr` フィールドの値を比較して推定する RTT の更新に用いる. ACK が返って来たパケットの数だけ RTT を更新することが出来るので同時に複数個のパケットを計測したのと同じ効果を得られる.

TCP Selective Acknowledge option

TCP Selective Acknowledge(以下 SACK) Option は受信したデータの途中のパケットが失われていた際に, その部分を送信側に伝えて, 再送信を要求できるようにするものである. SACK オプションは RFC1072[25] で TCP のオプションの 1 つとして提唱されたが, 具体的な仕様が規定されていなかったので, 実際に使われることはなかった. その後 Sally Floyd らによって詳細の実装設計が行われ, RFC2018[29] で再び規定された.

送信したパケットが途中経路で複数失われた場合にはその転送性能が極端に下がること

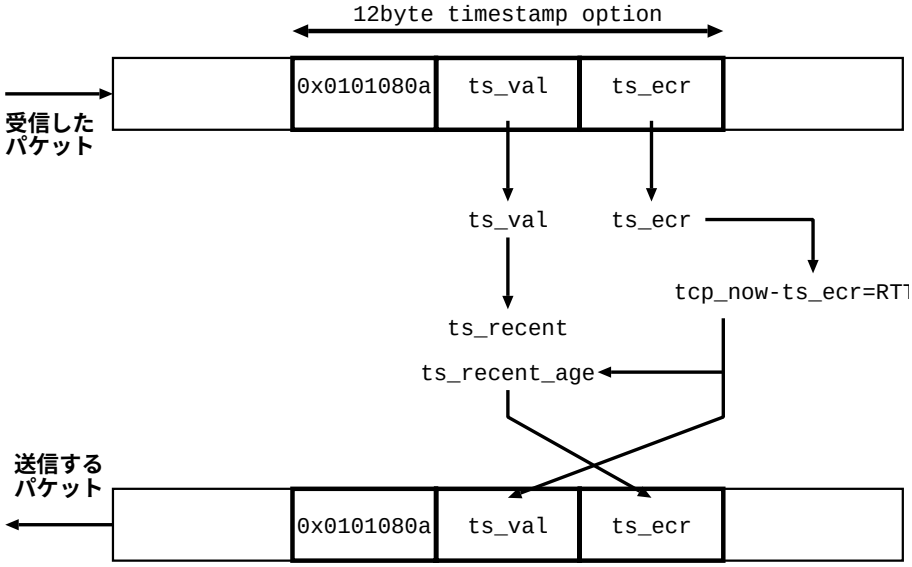


図 2.2: TCP Time Stamp Option

が知られている. ACK には連続して受信された最大のシーケンス番号しか情報がないので, Dup ACK だけでは, それまで送信したパケットの具体的にどこが失われたか, 穴の位置を判断することが出来ない. よって, 新たな番号の入った ACK が帰って来るまでは, 転送レートを下げながらパケットを 1 つずつ順番に再送せざろうえない.

そこで SACK option では受信したデータの途中のパケットが失われた際に, その穴になっているブロックの位置の右端と左端のシーケンス番号を TCP のオプションとして付けて, 送信側に送り返す. 送信側は受け取った穴のブロック部分を決めうちして再送することで, 転送性能の低下を最小限に抑えることが出来る. SACK option を利用した輻輳制御には Mattew Mathis らによる Forward Acknowledgement(FACK)[20] の研究がある.

2.2.4 TCP/Vegas

従来の TCP の輻輳制御機構に変更を加え, より高いスループットを得ることを目的とした新しいアルゴリズムに Univ. of Arisona の L.S.Brakmo らによって開発されている TCP/Vegas[7] がある.

従来の TCP のネットワークではパケットが失われたことによって輻輳の検知をしているのに加えて, TCP/Vegas では転送レートの変化を計測して輻輳の状態を判断している.

ある時点における TCP の転送性能はウィンドウサイズと配送確認 (ACK) を受信するまでの時間 (RTT) によって決定される. 途中経路のネットワーク帯域にまだ余裕がある場合には途中ルータの処理時間が短くなり, 全体の RTT も小さくなり転送レートが向上する. TCP/Vegas ではいままでに計測した RTT の中で最も短いものを Base RTT と定義し, 最大のスループットが得られる状態だと仮定した. ここで得られた理想的な転送性能 (Expected Throughput) と, 現在の RTT から得られた性能を比較して, ネットワークの帯域にまだ余裕があると判断した時には送信のウィンドウサイズを大きくし, 混雑していると判断した場合にはウィンドウサイズを小さくして, ネットワークの状態を送信量に反映させている. TCP/Vegas では理想状態と実際の状態を判断するために 2 つのしきい値を設けて, 突然の状態変化にもある程度対応できるようになっている. またこのしきい値の取り方によって転送性能が大きく左右される.

また TCP/Vegas では Slow Start の方法を従来の方法より緩やかなものにとどめ, 輻輳による影響を抑えている. Duplicate ACK の到着したときの fast retransmit にも手を加え, 従来の 3 つ目ではなく 2 つ目のタイミングで再送を開始している. 論文によれば従来の方式に比べ 3 割〜7 割の性能向上が報告されている [8] が, TCP のアルゴリズムの提案者である Van Jacobson をはじめ多くの研究者が輻輳制御機構に疑問を投げかけている. この輻輳制御に関する今後の解析, 評価が望まれる.

第 3 章

従来の実装の問題点

この章では,Delayed Acknowledgement が従来の TCP の輻輳制御や RTT 計測に及ぼす影響を挙げ, 問題点を指摘する.

3.1 Delayed Acknowledgement の問題点

TCP/IP ではデータストリームをセグメントごとに分割し, ヘッダを付けて複数のパケットにして相手に転送している. 1 本の回線で双方向通信を行っている場合には, pure ACK のようなデータ部分が全くないパケットや少量のデータしか含まないパケットの送信が増えると, データを運んでいるパケットの送信を妨げ, データの転送能力を低下させる原因にもなる.

TCP ではヘッダだけによるパケットがデータの転送能力を低下させないように遅延確認応答 (Delayed Acknowledgement:以下 Delayed ACK) 機構を持っている. これは ACK の情報を一旦送信キューに溜めて, カーネルから`tcp_fasttimo` とよばれる一定時間 (200ms) ごとに呼び出されるタイマのタイミングを元にしてまとめて送信するものである. 複数の ACK をまとめることで ACK パケットの数を減らすことが出来, また, 新たに転送されるデータと一緒に送信されることも期待できる. この Delayed ACK 機構は細かいデータパケットをやりとりする`telnet` のような対話的なアプリケーションでは有効である.

3.2 計測された RTT に含まれる誤差

ここでは, パケットの往復時間を計測する RTT に含まれる変動要素を分析する.

$$
\begin{aligned}
RTT &= L + N + D + \left(\frac{P}{W}\right) \\
\Delta RTT &= \Delta N + \Delta D + \Delta\left(\frac{P}{BW}\right)
\end{aligned}
$$

L:信号の物理的転送に必要な時間 (latency)

N:経路上の途中ルータが処理に要する時間

D:受信側で Delayed ACK によって待たされた時間

P:パケットの大きさ

BW:途中経路の通信速度の最小値

RTT の分布から一番知りたいことは現在のネットワークの状況である. 両端のホストからネットワークの転送能力などの具体的な情報を得ることは機構的に不可能であるが, 一般にネットワークが混雑してくると RTT の値は大きくなる.

転送の経路が変わらない限り, L は一定であり,RTT の変動には関係がない. 途中経路上のルータの処理時間 Nとは, ルータの送信 queue のメモリに蓄えられていた時間の総計で, その瞬間のネットワークの状態を表したものである. 通信の一端から瞬間的に途中経路の通信速度の最小値を知ることは不可能であるが, それまで行なった送信の速度から推定することは出来る.

Delayed ACK で ACK パケットの返信が待たされた時間 Dは受信したときのタイミングに完全に依存している. TCP の仕様では Delayed ACK の時間を 500ms 以下の値に設定することが要求されている.(従来の BSD 系 OS の実装では 200ms) つまり変動ΔDは 0〜200ms の範囲で存在することがいえるが, Dの具体的な値を知ることは不可能である.

ネットワークの状況を知る目的から言えば, 現在の RTT 計測では Delayed ACK が大きな弊害になっている. 一般的な WAN による通信では, この Delayed ACK の分布とネットワーク負荷による遅延の分布が似通っているために, 両者の分布を分離することは不可能である. また Delayed ACK による遅延は, RTT の推定量の誤差が大きくなる方向へ生じ

ているので, それにしたがって RTO の推定量も大きくなり, 再送時間が長くなる. 輻輳が発生した場合には, 通信速度の復帰が遅れる原因にもなっている.

3.3 Self clocking 機構の遅延

TCP の送信の判断は,self-clocking 機構により受信側が告知するウィンドウサイズと ACK 番号の確認によって行なわれることは前章で説明した.

送信した分の ACK の到着が Delayed ACK 機構によって遅れると, 確認が取れるまでは, 新たなデータを送るための送信側のタイミングも遅れることになる. またその間に送信バッファに溜っていたパケットを, 一度に送信することになるので, みずからの送信が輻輳を引き起こしやすい状況を産む原因になっている.

3.4 輻輳制御での問題点

従来の実装では送信側の輻輳ウィンドウサイズの更新は ACK パケットの到着をもとにして行なわれる.

現在の輻輳制御のアルゴリズムでは, スロースタート時に ACK が 1 つ帰るたびに`cwnd` を 1 セグメントずつ増加させている. 送信した複数のパケットの確認が, Delayed ACK によってまとめられ,1 つの ACK になって返送されても, ACK の数は 1 つであるため`cwnd` は 1 セグメント分しか増えない. このため`cwnd` の増加は ACK が 1 つ 1 つ返って来る場合に比べて, 非常にゆるやかなものになってしまう.

高速な回線での通信では, データ自体による遅延が少ないために, Delayed ACK している間に多くのパケットが到着し, まとめて ACK を返されるパケットが多くなる. このために `cwnd` がなかなか増加せずに送信速度の立上りの性能を低下させる原因になっている.

第 4 章

送信側からの確認応答の制御

この章では, TCP での流量制御, 輻輳制御の障害となっている受信側の Delayed ACK を送信側から制御する Cancel Delayed ACK(以下 CDL) の実現と設計について述べる.

4.1 遅延を解除する方法の提案

前の章では Delayed ACK が RTT の計測や輻輳制御に与える影響を指摘した. ここで受信側の Delayed ACK の制御を送信側から操作する方法を提案する.

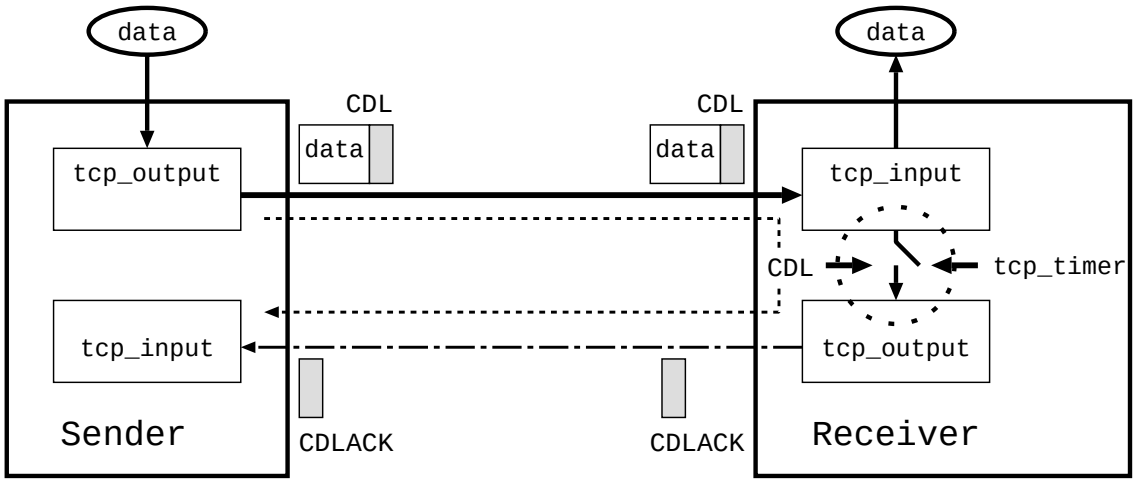


図 4.1: CDL 機構の概略

TCP のヘッダには 6 つの制御フラグ (ACK,PUSH,SYN,FIN,RST,URG) があるが, ここに送信側が, 受信側の確認応答を制御するようなフラグを追加する. この制御機構を Cancel

Delayed Acknowledgement(以下 CDL) とし, この機構を制御するフラグを CDL フラグと呼ぶ. 送信側が受信側の ACK を遅延させずに送信させたいときには送信するデータのパケットにこのフラグを立てる. このフラグを受け取った側は Delayed ACK 機構を解除してすぐに ACK を送信するようにする.

TCP のフラグの未定義の部分にこのフラグを追加したのは, ヘッダ自身のバイト数を増やすことなく制御情報を加えることが出来るからである.

4.2 CDL フラグの提案と仕様

ヘッダにおけるフラグの場所は以下の通りである.

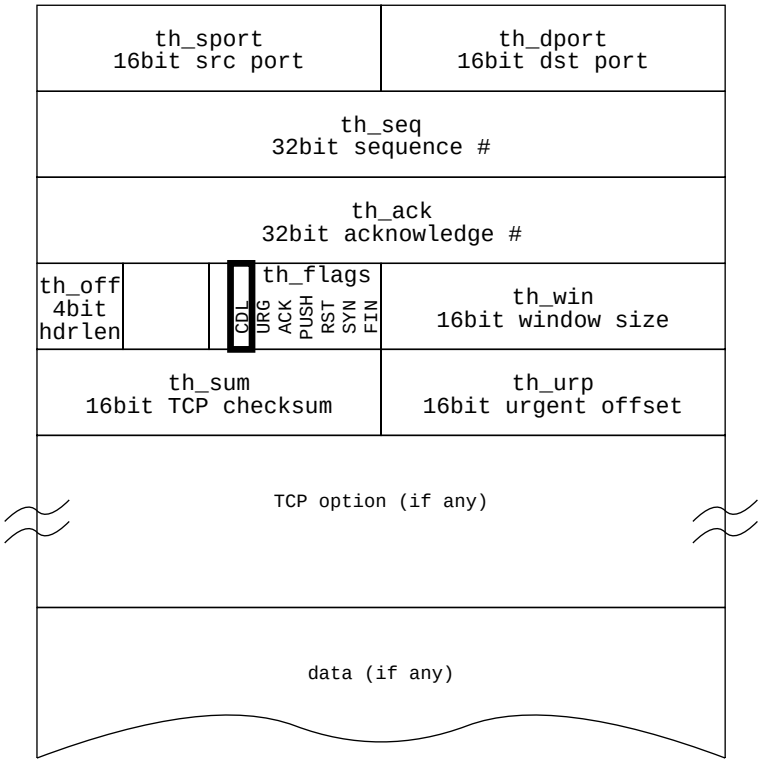


図 4.2: 提案する CDL フラグの TCP ヘッダでの位置

CDL フラグに対応するホストは次のような応答を示す.

送信側が CDL フラグを立てる 送ったデータに対する ACK を遅延なく返送してもらいたい場合には CDL フラグだけをセットする. この際, 受信側に対する ACK 番号`th_ack`

はセットするが, CDL フラグによる制御を区別するために,ACK フラグは立ててはいけない. (CDL-only フラグと呼ぶ)

受信側が CDL フラグのみがついたパケットを受け取る 受信側がデータと一緒に CDL-only フラグを受け取った場合には, 確認応答を遅延させずに送信側へ ACK を返す. その時には CDL フラグによって遅延なく返したことを示すために, CDL フラグと ACK フラグの両方をセットし,(CDLACK フラグと呼ぶ) ACK 番号`th_ack` をセットする.

もし何らかの理由 (socket バッファとのデータのやりとりが遅れている) が生じ, 遅延無しに ACK を返信できないときには,CDL フラグを立てずに普通の ACK フラグを立てたパケットを返信する.

送信側が CDLACK フラグを受信する CDL-only フラグを立てたパケットのシーケンス番号と CDLACK フラグを立てて帰って来た ACK の番号を確認することで, この応答が最小遅延で ACK が帰ってきたことを確認する.

4.3 CDL Allow Option の提案

CDL フラグを利用するためには, 送受信両方のホストともこのフラグに対応している必要がある. なぜなら送信側がつけた CDL フラグによって受信側の遅延確認応答が制御出来ることが重要であるからである. 片方のホストが従来の TCP のパケットしか受け付けることが出来ないと次のような問題が生じ, データの流れが止まってしまう.

- CDL フラグのみが付いたパケットを受信した時, 受け取ったパケットには ACK フラグが付いていないので, 送信側に ACK を送信することができない.
- CDL フラグは TCP チェックサム [24][27] の対象になっているので, 実装上の理由でこの部分を無視して計算しているホストではチェックサムエラーでパケットが破棄されてしまう.

そこで, 従来の TCP を採用しているホストと CDL を実装したホストとを見分けるために, あらたに TCP CDL-Allow options を設定した. コネクションを確立, または更新する時に SYN フラグと一緒にこの TCP CDL-Allow options を送信する.

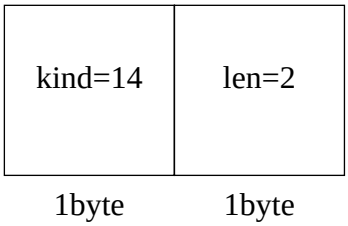


図 4.3: CDL Allow Option

CDL に対応しているホストは,CDL-Allow options によって相手のホストが CDL に対応しているかどうかを調べる. CDL に対応できるホストが CDL-Allow options を受け取った場合には, SYN に対する確認パケットを送る際に CDL-Allow options を付けて送信し, 両者が CDL フラグに対応していることを確認する.

CDL に対応していないホストが相手の場合には, CDL-Allow options は受け取ることが出来ないので破棄される. よって相手からの SYN に対する ACK パケットには CDL-Allow options は付いてこない. 相手が CDL に対応できないと判断した場合には, それ以降 CDL フラグを立てることを禁止することによって, 従来の TCP の実装にも対応できるようにしている.

なお,CDL フラグが付加されたパケットが未対応のホスト上を通ったとしても, 大抵のシステムではチェックサムを通過することは可能である. また CDL フラグ自体は無視される.

4.4 送信側の制御方針による影響

CDL フラグの導入により, TCP ヘッダのオーバヘッドを増加させることなく制御することが可能になった.

そこで全ての送信パケットに CDL フラグを付けてやると, 相手側から ACK パケットが最小遅延で帰って来るので, 反応が良くなる効果が得られるが, 受信側からの確認のためのパケットが増えるので, 反対に通信経路の資源を無駄にすることになる.

また送信側からの転送速度が非常に速い場合, 受信側のホストがパケットの中のデータをアプリケーションレイヤに渡す処理速度が律速になる場合も生じる. この処理がパケットの到着に追い付かない場合には, CDL の仕様より CDLACK が返せないので,CDL フラ

グの効果も薄れてくる. つまり CDL を利用する際には, 送信側のポリシーが重要になってくる.

3 章で述べたように Delayed ACK の影響を抑えたい時は, RTT の計測をしている時と, スロースタートの際の輻輳ウィンドウサイズを検知したい時である. よって RTT タイマの起動時とスロースタート時に CDL フラグを立てるのが一番効果的である.

次の章の実験では, 送信側が CDL フラグを付ける条件の違いによる改善点について, いくつかの条件を設定して性能の比較を行う.

第 5 章

CDL フラグの効果確認実験

この章では提案した CDL フラグ, および Delayed ACK 制御機構を実装したカーネルを用いて行った効果確認実験について述べ, 評価を行う.

5.1 TCP の転送性能評価について

現在 TCP の転送性能の評価に用いられる手法には 2 つある. シミュレーションによる方法と, 実際にデータを転送して観測する方式である.

広く利用されているネットワークシミュレータとしてはNest[17], REAL[16],SimPack3,netsim,ns[18] などが挙げられる. これらのシミュレータはノード間の転送時間や通信帯域を設定し, 簡単なネットワークを構成する. さまざまなアプリケーションを使用したときの流量分布や TCP の送信制御アルゴリズムに沿ってパケットの流れや送受信キューを計算機上でシミュレートするものである.

TCP の転送性能は, 純粋にプロトコルやアルゴリズムによる性能だけでなく, 実装方式や OS やネットワークのアーキテクチャ, データリンクの性能などに大きく左右される. 一般的なネットワークシミュレータでは OS やデータリンク層の細かい特性を再現することは困難である.

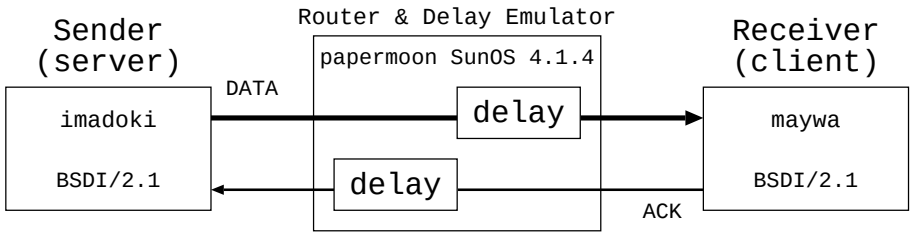
そこで実際にあるネットワーク機材やデータリンクを用いてネットワークを構成し, データを転送して, その挙動を解析することで性能を評価する方法が多く用いられている.

5.2 評価実験

この論文では, 以下のようなネットワーク模擬環境を構成して, 提案した Delayed ACK 制御機構の性能比較を行った.

5.2.1 実験環境

実験の対象となるホストには 2 台の Intel-base(Pentium) の PC を用意した. 2 台の間には帯域 10M[bps] の ethernet(10Base-T) を用いてネットワーク的に接続し, 途中にデュアルホームである (2 本の ethernet リンクを持つ)Sun WS をルータとして接続した.



この Sun WS は,delay emulator[15] を使用して, IP 層においてソフトウェア的に遅延を模擬発生させている. これは伝送遅延 (固定遅延, 変動幅), 通信帯域, パケットロス, Queue サイズをユーザ側から任意の値に設定することが出来るものである. この delay emulator 自体による処理時間は 1[ms] 以下で, 実験でのパケットの観測にはほとんど影響は及ぼさなかった. この 3 台の間には実験以外のパケットが流れることはほとんどなく, ほぼ閉鎖された環境とみなして実験を行った.

実験で想定した環境は行き 25[ms]+帰り 25[ms] の RTT が 50[ms], 帯域がどちらも 512K[bps] で, サーバからクライアントへ 1500[Bytes] の TCP/IP パケットを 10 秒間送信し, その間にながれたパケットをサーバ側から観測した.

5.2.2 実装

2 台の PC には,BSDI/2.1 OS を搭載し, 設計した CDL フラグに対応させたものを実装した.

TCP/IP のネットワークコードはもともとの Net/2 release に近いものを用いた. (Slow Start, Congestion Avoidance, Fast Retransmit, Fast Recovery, TCP Window Scale/

Timestamp options は実装されているが, TCP CC / SACK options は実装されていない.) 今回は一般的な TCP の実装との比較を行うのが目的であるため, BSDI が独自に採用した機構拡張などは用いなかった.

パケットの観測には OS に Berkeley Packet Filter(bpf-1.1)[4] を組み込んで, パケット観測ツール LBL libpcap + tcpdump を使用した. tcpdump には今回提案した CDL フラグや CDL Allow option が監視できるように改良を施した.

```
13:37:26.129717 imadoki.1027 > maywa.1060: S 18947299:18947299(0) win 8192
                 <mss 1460,nop,wscale 0,cdl-allow,timestamp>
13:37:26.184039 maywa.1060 > imadoki.1027: S 3052158394:3052158394(0)
             ack 18947300 win 8760 <mss 1460,nop,wscale 0,nop,nop,cdl-allow>
13:37:26.186071 imadoki.1027 > maywa.1060: . ack 1 win 8760
                                            <cdl-allow,timestamp 197 77079>
13:37:26.189470 imadoki.1027 > maywa.1060: D 1:1449(1448) ack 1 win 8760
                                              <nop,nop,timestamp 197 77079>
13:37:26.268132 maywa.1060 > imadoki.1027: C ack 1449 win 8760
                                              <nop,nop,timestamp 77079 197>
13:37:26.272692 imadoki.1027 > maywa.1060: D 1449:2897(1448) ack 1 win 8760
                                              <nop,nop,timestamp 197 77079>
13:37:26.275237 imadoki.1027 > maywa.1060: . 2897:4345(1448) ack 1 win 8760
                                              <nop,nop,timestamp 197 77079>
13:37:26.351237 maywa.1060 > imadoki.1027: C ack 2897 win 8760
                                              <nop,nop,timestamp 77079 197>
```

図 5.1: tcpdump によるパケット観測結果

5.3 Delayed ACK 制御機構の効果

ここでは,Delayed ACK 制御機構を単純に比較するために, CDL フラグを利用しない従来の場合と, RTT タイマで計測しているパケットにのみ CDL フラグを付けた場合とで効果の比較を行った.

また,RTT の複数計測につながる TCP Timestamp option(RFC1323) を利用した場合としなかった場合との比較も行った.

送信開始からの立上り時の転送量と転送速度の比較をそれぞれ以下のグラフに表した.

CDL フラグ	TimeStamp	転送バイト数	送信	ACK	Dup ACK	CDL	CDLACK
なし	なし	561KB	394	206	188		
なし	あり	570KB	390	205	195		
あり	なし	581KB	398	246	152	104	104
あり	あり	616KB	417	255	162	116	115

表 5.1: CDL フラグの影響比較でのパケットの種類

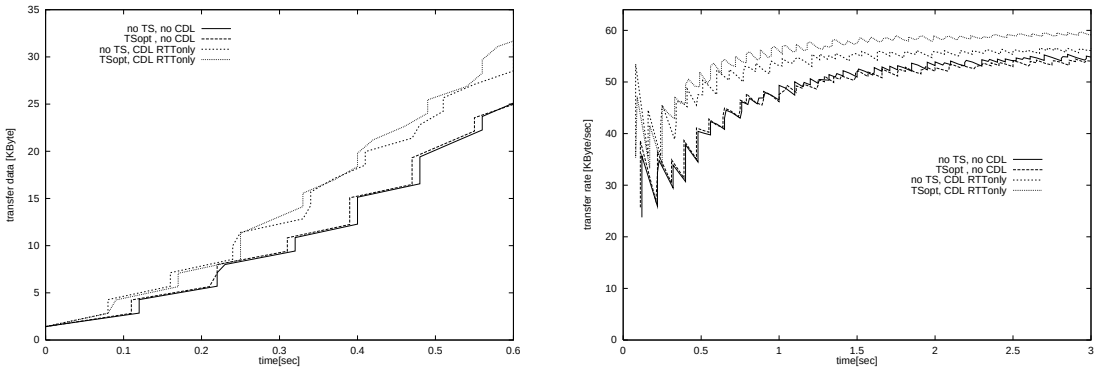


図 5.2: CDL フラグの影響比較 (RTT=25+25[ms])

これらのグラフより,CDL フラグを利用するとパケット送信の立上りの特性が良くなることが分かる.

これは送信側からの CDL フラグによる指示によって受信側の Delayed ACK が抑制されて, すぐに ACK が返送されるので, 送信したデータが最短の遅延で確認され, 次のパケットが送信できるからである. これによりこの制御機構により転送性能が向上したことが確認された.

また, ここで RTT や通信帯域は同じだが, 行きと帰りに所要する時間が違うようなアンバランスの接続を想定して, 転送実験を行った結果を次に示す. 行き 45[ms]+帰り 5[ms] の RTT が 50[ms], 帯域がどちらも 512K[bps] で, サーバからクライアントへ 1500[Bytes] の TCP/IP パケットを 10 秒間送信し, その間にながれたパケットをサーバ側から観測した.

上のグラフより, データ転送に時間がかかるような経路のほうが CDL フラグの効果がより明らかになった. これは同じ RTT と観測されても, 通信路上で ACK だけのパケット

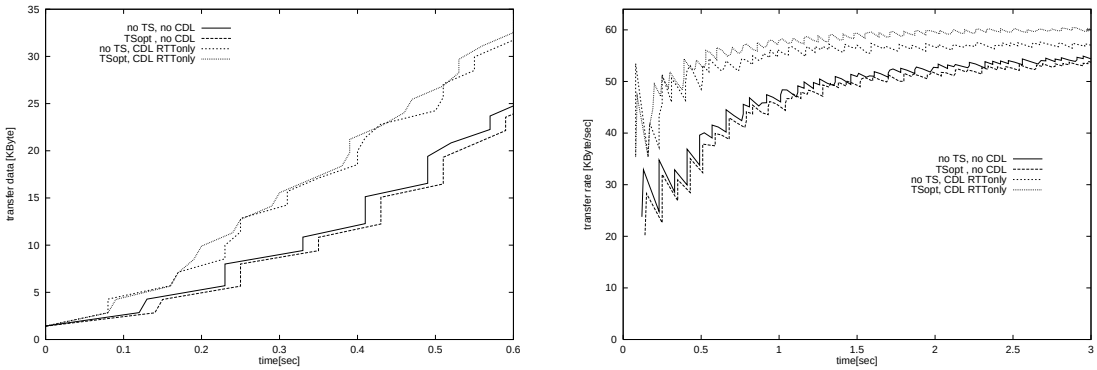


図 5.3: CDL フラグの影響比較 (RTT=45+5[ms])

が占める割合が低いので TCP/IP ヘッダによるオーバーヘッドが少ないために総合的に見て転送の性能が上がったからである.

5.4 CDL フラグの制御方針による違い

次に, 送信側が CDL フラグを付ける方針の違いによる転送性能への影響を実験により評価した.

この実験では, 送信側が次のようなアルゴリズムに基づいて, CDL フラグを立てることにし, 効果の比較を行った.

1. CDL フラグを利用しない
2. RTT 計測をしているパケットだけに設定する
3. (2)+1 つ置きごとのパケットに設定する
4. (2)+2 つ置きごとのパケットに設定する
5. (2)+ウィンドウサイズいっぱいで最後に送信するパケットに設定する
6. (2)+スロースタート時にはすべてのパケットに設定する

比較の結果のグラフは以下の通りである.

CDL 方針	TimeStamp	転送バイト数	送信	ACK	Dup ACK	CDL	CDLACK
CDL なし	なし	568KB	390	205	185		
RTT のみ	なし	581KB	398	240	158	104	104
1 つおき	なし	583KB	399	253	146	227	227
2 つおき	なし	585KB	401	276	125	175	175
3 つおき	なし	591KB	405	250	155	128	128
SS 全部	なし	626KB	429	333	96	231	230
RTT のみ	あり	616KB	417	255	162	266	262

表 5.2: CDL フラグの方針比較でのパケットの種類

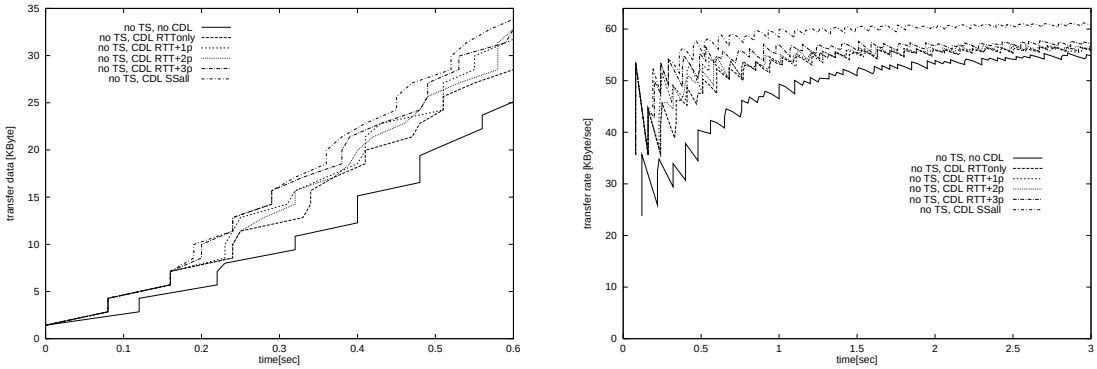


図 5.4: 方針の違いによる影響 (RTT=25+25[ms])

このグラフから, CDL フラグを利用した全ての方針で,CDL フラグを利用しなかったものより転送性能が改善されていることが確認できた. また CDL フラグの方針による違いは, RTT 計測時だけ CDL フラグを付けるよりも, それに条件を付加してフラグを利用したほうが性能が良くなることが分かった. 特にスロースタート時にすべて CDL フラグを利用したものは立上り時, 総合性能とも優れていることが分かった.

これは, 現在実装されているスロースタートアルゴリズムが, 輻輳ウィンドウサイズが不明であるときには帰って来た ACK を元にして送信ウィンドウサイズを広げる仕様になっているため, ACK の量が増える程速くウィンドウサイズが広がるからである.

またこの比較でも, 行きと帰りに所要する時間が違うようなアンバランスの接続を想定して, 転送実験を行った.

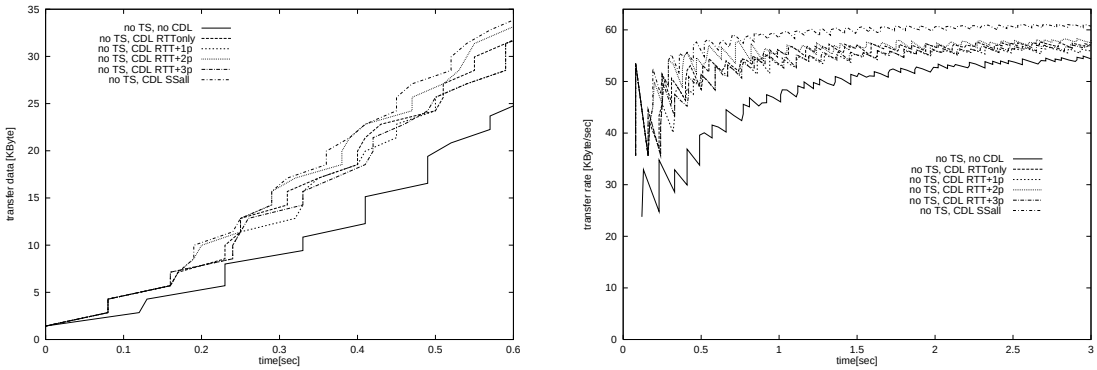


図 5.5: 方針の違いによる影響 (RTT=45+5[ms])

こちらのほうも,CDL フラグを付ける方針によって, 立上りの特性が違うことが確認された.

第 6 章

RTT 計測の改善

この章では従来の TCP の実装において, 送信制御を行う上での障害になっている, タイマの粒度について取り上げる. 提案した CDL 制御方式によって改良を施した場合の効果や問題点について述べる.

6.1 タイマの粒度について

TCP の従来の実装では,RTT は 500ms の粒度で計測されている. これはカーネル内で 500ms ごとに `tcp_slowtimo()` が呼び出され, タイマの計測値を 1 つインクリメントすることによって実現されている.

RTT を推定する smoothed RTT 評価関数 (`t_srtt`) では RTT タイマの 8 倍 (`TCP_RTT_SCALE`) のスケーリングを行うことで多数得られた計測値から精度を上げて利用することが出来る. これらは固定小数点による整数演算で実現されており, RTT の計測,RTO の推定が少ない処理ステップで実行できるようになっている.

しかし近年では高速ネットワークが整備され, データパケット自体の転送所要時間がどんどん短くなっている. 1500byte のパケットでも T1 回線 (回線帯域 1.5Mbps) では 8ms, T3 回線 (45Mbps) では 270μs しかかからない. LAN 環境などでは数 ms で相手ホストまで到達してしまうし, 世界中でも経路自体の latency が 500ms を越えるような経路はまれである.

2 章で述べたように, 送信スルーレートと RTT から Bandwidth Delay Product(BDP) を計算し, その変化からネットワークの状況を検知して, 送信制御に反映させる研究があ

る. 例えば Jain らによる CARD[21] はわずかな時間経過の間では BDP が一定であると仮定し, 経路上の一番帯域の狭い部分のスループットの変化に応じて RTT も変化するという考えに基づいている. しかし従来のタイマによって計測され, 推定された RTT では, これらの輻輳制御を行なう場合には精度が不十分であることが指摘されている.

また輻輳制御で最大通信帯域を検知するためには, 輻輳を発生させて一旦パケットを落させる必要がある. パケットの損失が輻輳検知を知る手段なので一度 timeout を発生させることになる. この timeout に見積もられる時間が長いほうに誤差が生じると, パケットの送信が再開されるまでの時間が余計にかかる. このことが立上りの送信性能を下げる大きな一因になっている.

接続の初期段階では, 計測された RTT の数が少なく, 推定値に含まれる誤差が大きくなることは自明である. そこで Delayed ACK による誤差を最低限にするだけでなく, タイマの粒度を上げることも必要になる.

6.2 システムクロックの導入

そこで, ここでは従来の `tcp_slowtimo()` ではなく, マシンのシステムクロックを利用したより粒度の高いタイマの導入を検討する. このタイマで計測することで, 精度の高い RTT 推定を行うことができ, 再送 timeout に必要な時間を最低限に抑えることが可能になる.

BSD 系 OS には `microtime()` と呼ばれる, マシンのシステムクロックが得られるμsオーダのタイマがある. また,`softclock()` と呼ばれる優先度の低い割込みハンドラがあり, 任意の時間にソフトウェア割込みを発生させることが出来る. これらを利用して,4 章で提案した CDL フラグと併用して RTT を ms オーダで計測し, 正確な推定を行う. またパケットの再送タイムアウトに利用して, 従来の `tcp_timer()` が expire するよりも粒度の細かい再送タイミングを実現することができる.

6.3 CDL フラグとの併用

3.2 章では Delayed ACK による計測した RTT への影響を説明した. この影響を導入した CDL 機構にて抑制することにより, CDL フラグを付けて CDLACK フラグで返送されたパケットの計測結果はネットワークの状態をより的確に表していることが自明である.

従来の BDP に着目した流量制御の研究では Delayed ACK の影響を打ち消すことができなかったためにネットワークの状況を知ることが出来ず, 思わしい結果が得られなかった. ここで採用した粒度の高いタイマと CDL フラグを協調して使用することでより正確なネットワークの状況計測が可能になり, 再送時間の正確な決定, 流量制御, 輻輳制御へのフィードバックが期待できる. ネットワークの混雑状況と実際に送信できる帯域がどのような関係にあるか, また, 非線形である関数に対してどのような制御アルゴリズムが有効であるかを検証する必要がある.

第 7 章

考察

この章では, 提案した CDL フラグが現在実装されている TCP の輻輳制御メカニズムや広域ネットワークの研究にどのような与える影響について考察する.

7.1 RTT 測定,RTO 推定への影響

今までの送信制御では Delayed ACK の遅延によって送信タイミングが遅れる方向へ伸ばされていた。送信側が提案した CDL フラグを使うことで, 受信側の CDLACK が確認できれば, そのパケットは最小遅延で往復して来たことが保証されている. 途中経路での latency が最小の RTT により推定できればネットワーク負荷による処理時間の分布が明らかになり、ネットワークの状態をより明確に知ることが出来る.

Van Jacobson の研究 [1] による RTO の推定式は実際のトラフィック計測を元にして係数が決定された。平均偏差を加える際の係数は最初は 2 倍であったが、その後の研究 [2] で 4 倍にしたほうが最適であることが分かり、変更が施された。再送時間の決定はコネクションの接続初期の転送性能を大きく左右するので, その後の研究でも改良をするこころみがいくつか行なわれている.

今回の CDL フラグを使用すれば, Delayed ACK による分散の影響を取り除くことが出来るようになったので, 再送 timeout 時間 (RTO) の決定式自体にも大きく影響すると思われる。たとえば,

$$RTO = srtt + 4\,rttvar$$

から,

$$RTO_{(CDL)} = srtt_{(CDL)} + 4\, rttvar_{(CDL)} + \Delta D$$

のように変更することで, Delayed ACK の変動による RTO の超過を小さくすることが出来る. CDL フラグを利用して RTT 計測をしたパケットと従来方式による計測したパケットとでは, RTT 推定の方式の変更を切替えるような必要も生じるであろう.

7.2 輻輳制御に与える影響

CDL を使用すれことで, 送信側からある程度 ACK の返送を強制することが出来る. そのため Delayed ACK がない分, より速い段階でより正確に輻輳を検知することが可能になる。また CDL の比較実験の結果より, アンバランスの経路である程, CDL 制御の効果が現れることも特筆すべきことである. CDL フラグの利用によってネットワークの変化を正確に検知できるようになったので, 輻輳ウィンドウの更新方法を非線形にするなどの改良を加える必要があるといえる.

7.3 トラフィック特性への影響

CDL フラグを用いて,ACK の返送を強制して増やした場合, 実験の例では ACK のトラフィックが 1〜2 割程度増えている.

CDL フラグによる制御は,ACK パケットを強制的に送信させるので, 常にデータパケットとの帯域の奪い合いになる危険性がある. このことは物理層の選択によっても影響が大きく違って来る. 1 つの通信チャネルを共有して交信するイーサネットのような場合ではホストが CDL フラグを受け取って, すぐに ACK のパケットを送り出した場合には, 次に到着したデータパケットと衝突して, 回線帯域を占有していなくても輻輳発生状態にしてしまうかもしれない. 反対に 2 本の通信路を往復で別々に使用する経路を取る場合は、データ転送に及ぼす ACK のパケットの影響が少ないと考えられるので、CDL フラグの積極的な使用による改善が望めると思われる。

このことにより送信側の CDL フラグの運用は慎重なものであることが望まれる. あくまでも交信の過渡状態にあたる, 帯域を占有していない, 使用できる帯域が確定できない時期だけに限定される必要がある。

また,CDL フラグを使用すれば ACK の self-clocking 機構を接続の初期から利用することが出来る。このことにより,Delayed ACK によってバースト的に送信されるホストとは異なった送信特性になると思われる。データがまとまってではなく時間的に等間隔に分散して送られるのでボトルネックになっている回線の直前のルータの queueing には負荷が少なくする効果が期待できる。反対にルーティングを担当しているルータでは同一経路のパケットがバラバラにやって来るので処理を多くし、選択できる複数の経路に分散されてしまい,RTT の分散状況を悪化させてしまう可能性もある. このことも考慮に入れて CDL フラグの運用は慎重にする必要がある.

第 8 章

まとめ

ネットワーク上で信頼性のあるストリーム配送を実現している TCP の役割はインターネットの発展とともにさらに重要なものになっている。ネットワークの規模の拡大や、データリンクの性能の向上にともない、ユーザの爆発的な増加と大量のデータ通信の運用は TCP における流量制御、輻輳制御をよりいっそう重要なものにしている。

本論文では, 従来の TCP における流量制御、輻輳制御アルゴリズムの検証を通じて, 転送性能の低下の原因となっていた受信側の遅延確認応答の問題点を指摘した。この影響を抑制するために送信側から制御する機構を提案し, 実装を行なった. 新しく提案した制御フラグを送信側が設定することで TCP ヘッダの容量を増やすことなく制御できるようになった。性能比較実験を通じて, 遅延確認応答を適度に制御することによって転送性能の向上が確認された。

この制御機構との併用で, パケットの往復時間を受信側の遅延による影響を受けずに, ネットワークの状況をより正確に計測することが可能になった。またこの機構が従来の再送時間推定や輻輳制御のアルゴリズムにどのような影響を与えるかを考察した。

今後の課題としては, 正確に計測したネットワークの状況をいかに輻輳制御に反映させるか考察し、適切な制御方法の設計を行なう。さまざまな状況を想定したネットワーク転送実験を通じて輻輳やネットワークの状態変動に強い流量制御の開発を行なう。

謝辞

本研究を進めるにあたり終始ご指導下さいました篠田陽一助教授に心から御礼申し上げます。

本研究に対して有意義な議論をして頂いた篠田研究室の各メンバーの方々に深く感謝致します。

参考文献

[1] Van Jacobson Congestion Avoidance and Control Proceedings of ACM SIGCOMM '88, pp.314-329, 1988

[2] Van Jacobson Berkeley TCP Evolution from 4.3-Tahoe and 4.3 Reno Proceedings of the 18th Internet Engineering Task Force, p.365, University of Britich Columbia, Vancouver, B.C, 1990

[3] Phil Karn, Craig Partridge Improving Round-Trip Time Estimates in Reliable Transport Protocols Proceedings of ACM SIGCOMM '87, pp.2-7, 1987

[4] S. McCanne, Van Jacobson The BSD Packet Filter: A New Architecture for User-Level Packet Capture Proceedings of 1993 Winter USENIX Conference, pp.259-269, 1993 http://ftp.ee.lbl.gov/papers/bpf-usenix93.ps.Z

[5] W. Richard Stevens TCP/IP Illustrated Volume 1, The Implementation Addison-Wesley Publishing Company, 1994

[6] Gary. R. Wright, W. Richard Stevens TCP/IP Illustrated Volume 2, The Implementation Addison-Wesley Publishing Company, 1995

[7] L.S.Brakmo, S.W.Omalley, L.L.Peterson TCP Vegas: New Techniques for Congestion Detection and Avoidance Proceedings of ACM SIGCOMM '94, pp.24-35, 1994

[8] Jong Suk Ahn, Peter B. Danzig, Zhen Liu, Limin Yan Evaluation of TCP Vegas: Emulation and Experiment http://excalibur.use.edu/research/vegas/doc/vegas.html, 1993

[9] W. Richard Stevens TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms ftp://rs.internic.net/drafts/draft-stevens-tcpca-spec-01.txt, Internet Draft, 1996

[10] Kjersti Moldeklev , Per Gunningberg How a Large ATM MTU Causes Deadlocks in TCP Data Transfers IEEE/ACM Transactions on Networking, Vol.3 No.4, Aug. 1995, pp.409-422, 1995

[11] Sally Floyd TCP and successive fast retransmits ftp://ftp.ee.lbl.gov/papers/fastretrans.ps, Technical Report, 1995

[12] Kevin Fall, Sally Floyd Comparisons of Tahoe, Reno, and Sack TCP ftp://ftp.ee.lbl.gov/papers/comparison.ps, Technoical Report, 1996

[13] Vern Paxson Empirically-Derived Analytic Models of Wide-Area TCP Connections IEEE/ACM Transactions on Networking, Vol.2 No.4, Aug. 1994, pp.316-336, 1994

[14] Will E. Leland, Murad A. Taqqu, Walter Willinger, Daniel V. Willson On ther Self-Similar Nature of Ethernet Traffic IEEE/ACM Transactions on Networking, Vol.2 No.1, Feb. 1994, pp.1-15, 1994

[15] Elliot L. Yan The Design and the Implementation of an Emulated WAN http://excalibur.use.edu/research/vegas/doc/delayemu.html, 1994

[16] S. Keshav REAL : a Network Simulator Report 88/472, Computer Science Department, University of California at Berkeley, 1988

[17] D. Bacon, A. Dupuy, J. Schwartz, Y. Yemimi Nest: a Network Simulation and Prototyping Tool Proceedings of Winter 1988, USENIX Conference 1988, pp. 17-78

[18] Sally Floyd Simulator Tests http://www-nrg.ee.lbl.gov/ns/test.ps, May 1996

[19] Sally Floyd, Van Jacobson Random Early Detection Gateways for Congestion Avoidance IEEE/ACM Transactions on Networking, Aug. 1993,

[20] Matthew Mathis, Jamshid Mahdavi Forward Acknowledgement : Refining TCP Congestion Control Proceedings of ACM SIGCOMM '96, pp.281-291, 1996

[21] R.Jain A Delay-Based Approach for Congestion Avoidance in Interconnected Heterogeneous Computer Network Computer Communication Review, Vol.19, no. 5, pp. 56-71, Oct. 1989

[22] Ashok Erramilli, Onuttom Narayan, Walter Willinger Experimental Queueing Analysis with Long-Range Dependent Packet Traffic IEEE/ACM Transactions on Networking, Vol.4 No.2, Apr. 1996, pp.209-223, 1996

[23] J. Postel Transmission Control Protocol Request For Comments 793, 1981

[24] R. Braden, D.Borman, C. Partridge Computing the Internet checksum Request For Comments 1071, 1988

[25] R. Braden, Van Jacobson TCP extensions for long-delay paths Request For Comments 1072, 1988

[26] D. Borman, R. Braden, Van Jacobson TCP Extensions for High Performance Request For Comments 1323, 1992

[27] A. Rijsinghani Computing of the Internet checksum via Incremental Update Request For Comments 1071, 1988

[28] Bob Braden T/TCP – TCP Extensions for Transactions Functional Specification Request For Comments 1644, 1994

[29] M. Mathis, J. Mahdavi, Sally Floyd, A. Romanow TCP Selective Acknowledgment Options Request For Comments 2018, 1996