絶対帯域見積りに基づく輻輳回避アルゴリズム

釘本 健司† 天海 良治‡ 村上 健一郎†
NTT ソフトウェア研究所† NTT 基礎研究所‡

概要

TCP による通信での性能上の問題を解決する MAB (最小絶対帯域検出アルゴリズム) を提案する。Reno + Extension TCP による単一ストリームの通信において、エンドノードが接続されているネットワークの帯域に比べて狭い帯域のリンクが経路上に存在する場合には、輻輳によりスループットが低下する現象が見られる。本論文では、まず 従来の TCP の輻輳アルゴリズムを説明した後、この現象を解析して従来アルゴリズムの問題点を明らかにする。次にネットワークの絶対帯域を反映する最小帯域ウィンドウを新たに導入して、これに基づく輻輳制御アルゴリズムである MAB を提案する。最後に、外乱のない実験用ネットワークを構築して行なった測定の結果を通して、このアルゴリズムを実装した TCP の単一ストリームの通信における効果について述べる。

1 はじめに

近年、インターネットの世界的な広がりにともない、巨大科学と呼ばれる分野でのインターネットの利用が増えてきており、気象データベースや遺伝情報データベースのような巨大なデータベースを地理的に離れた場所から高速に相互参照するなどの目的で高速インターネットの必要性が高まっている。

高速インターネットにおいて帯域を有効に使って TCP (TransmissionControl Protocol) [1] のスループットを向上させるためには、ウィンドウサイズを増やすことが必要である。ウィンドウサイズとは確認応答 (ACK : Acknowledgement) を待たずに送信できるデータ量を表す。end-to-end で使用可能な最大の帯域を使い切るだけのパケットを送り出すには、最大ウィンドウサイズは、少なくとも伝送路の帯域と遅延の積よりも大きくなければならない。この積は DB 積 (Delay Bandwidth Product) と呼ばれる。

しかし、単一ストリーム通信において、エンドノードの帯域よりも狭い帯域のリンクが経路上に存在すると、ネットワークに他のトラフィックなどの外乱がない場合でも、伝送路の絶対帯域から期待できるはずのスループットが得られないことがある。これは、TCP はネットワークの絶対的な帯域を見積もる仕組みがないので、帯域をできる限り拡大しようとして輻輳を起こして複数のパケットが消失し、タイムアウトによる再送が起こるまでの間転送が止まるという無効な動作が繰り返されてしまうためである。この結果、スループットの著しい低下を引き起こす。

我々は、TCP のこのアノマリ (anomaly: アルゴリズムが想定していない状況下での異常動作) を解決してスループットを改善するために MAB (Minimum Absolute Bandwidth detection algorithm) と呼ぶ新しいアルゴリズムを提案する。このアルゴリズムは、TCP のコードにネットワークの絶対的な帯域を表す最小帯域ウィンドウを新たに導入し、伝送路の帯域にあった最適なウィンドウサイズを最小帯域ウィンドウに設定することにより輻輳を回避してスループットの改善をはかるものである。このアルゴリズムの実装上の利点は、(1) 送り側のアルゴリズムのみの変更で済むこと、(2) TCP のプロトコルを変える必要がないことの二点である。

本論文では、まず従来の TCP の輻輳制御アルゴリズムの種類と発展の歴史について第2節で触れ、第3節では TCP のアノマリとその原因について詳細に説明する。第4節では、TCP のアノマリを解決する MAB について述べ、第5節では従来アルゴリズムとの比較による評価を行なう。また、第6節では今後の課題について述べる。

2 従来の TCP における輻輳制御

この節では、4.3BSD TCP, 4.3BSD Tahoe TCP, 4.3BSD Reno TCP, 4.4BSD と発展してきた従来の TCP における輻輳制御について説明する。

Minimum Absolute Bandwidth Detection Algorithm — MAB —
Takeshi Kugimoto, Yoshiji Amagai, Ken-ichiro Murakami
NTT Software Laboratories, NTT Basic Research Laboratories

2.1 4.3BSD TCP 以前

4.3BSD TCP 以前の TCP から実装されていた輻輳制御アルゴリズムは、TCP タイマーアルゴリズムである。TCP では信頼性のある通信のために確認応答 (ACK) を要する。ACK が送られてこない場合には当該パケットを再送する必要があるが、再送のタイミングが早過ぎると、無駄なトラフィックを発生させてしまう。

このアルゴリズムはパケットの送信と同時にタイマを起動し、ACK が返ってこない場合でもこのタイマが切れるまでは再送を行なわないことによって輻輳を回避する。タイマの値は、計測された RTT(Round Trip Time) の値に基づいて決定される。

2.2 4.3.BSD Tahoe TCP

Tahoe TCP では、輻輳制御のために Congestion Avoidance, Slow Start, Fast Retransmit アルゴリズム [6] が 4.3BSD TCP に追加された。これらのアルゴリズムでは、輻輳ウィンドウ (cwnd) と閾値 (ssthresh) が新たに導入されている。輻輳ウィンドウは通常のウィンドウの最大値を輻輳の状況に応じて動的に制限する働きをもつ。この輻輳ウィンドウの増加のしかたは、sshtresh の値を境界として変わる。輻輳ウィンドウが ssthresh を越えているときには Congestion Avoidance アルゴリズムが働き、ssthresh 以下のときには Slow Start アルゴリズムが働く。

Congestion Avoidance は、輻輳ウィンドウ cwnd により輻輳の回避を行なうアルゴリズムである。送り側はパケットの消失を検知すると、それを輻輳の兆候とみなして輻輳ウィンドウを半分にしてパケットの送信を制限することにより、ネットワークの輻輳回避に寄与する。輻輳が検知されると ssthresh は直前の cwnd の値の半分にされる。cwnd は 1RTT ごとに 1MSS (Maximum segment size) だけ増やされる。すなわち、1RTT ごとに線形に増加することによって輻輳の発生を遅らせようとする。

Slow Start は、受信側から通知されてきたウィンドウサイズ分のパケットがネットワークに急激に送り込まれないようにするために、輻輳ウィンドウの増加を制御するアルゴリズムである。このアルゴリズムでは、コネクションが確立した直後や再送タイムアウトの直後は cwnd は 1MSS になっており、輻輳ウィンドウの値が ssthresh を越えるまでは ACK がひとつ返るごとに 1MSS ずつ cwnd が増やされる。すなわち、cwnd は 1RTT ごとに指数的に増加する。

Fast Retransmit は、同じ番号の ACK すなわち重複 ACK (duplicate ACK: dup-ACK) を TCP コード中の tcprexmtthres で決められる回数だけ受けとると、パケットが失われたものと判断して当該番号のパケットを再送タイマが切れる前に早い段階で再送することによって高スループットを保とうとするアルゴリズムである。tcprexmtthres の値は通常 3 である。

2.3 4.3BSD Reno TCP

Reno TCP は、Tahoe TCP にさらに改良を加えたものである。主な変更点は、Fast Retransmit アルゴリズムに新たに Fast Recovery アルゴリズムを組み込んだことである。Fast Recovery アルゴリズムは、Fast Retransmit が行なわれた後に働く。

Reno TCP では、dup-ACK を 3 つ受けとると、これを輻輳によるパケットの消失の証と見て、当該パケットを一つだけ再送し (Fast Retransmit)、その後に cwnd を半分の値にする。その後、重複 ACK が返る度に cwnd は 1MSS ずつ増やされる (Fast Recovery)。

Reno TCP より以前の TCP では、Fast Retransmit 後に直ちに Slow Start アルゴリズムに移り、cwnd は 1MSS にまで減じられていたのに対し、Fast Retransmit / Fast Recovery アルゴリズムでは cwnd をパケット再送直前の値の半分の値から再スタートさせ、また、dup-ACK をネットワークの帯域の余裕とみなして cwnd を増やすことにより、早く元のスループットまで回復させることを狙ったものである。

Fast Retransmit / Fast recovery アルゴリズムは、軽度の輻輳からの回復を対象としたもので、単一のパケットの消失は回復できるが、複数のパケットの消失に対するリカバリがうまく働かない [2]。

2.4 4.4BSD TCP

Reno TCP からの主な変更点は、長距離高速環境下の通信に対応できるように TCP Extension[3][4][5] を組み込んだことである。TCP Extension は、主に (1) TCP で扱える最大ウィンドウサイズの拡大するための Window Scale Option と、(2) RTT の計測精度をあげる目的で導入された Time Stamp Option からなっている。

TCP のヘッダのウィンドウサイズを格納するフィールドは 16bit であるため、4.4BSD 以前の TCP では最大のウィンドウサイズが $2^{16}(=65535)$bytes であった。いま、10ms の遅延を持つ伝送路を仮定すると、64Kbytes のウィンドウサイズでは無限の帯域を持つ伝送路であっても 6.4Mb/s のスループットしか得られない。したがって、この値は将来のギガビットクラスの広域ネットワークでは不適切な値と言える。Window Scale Option を実装した TCP では、コネクションの確立の段階でウィンドウオプションが使用可能であるかどうかを調べ、可能であれば $2^{30}(=1G)$bytes まで最大ウィンドウをとることができる。この拡張により長距離高遅延の高速ネットワークでも高いスループットを得ることができる。例えば、月との間の通信を仮

定し、月までの RTT を約2.6秒とすると、理論的には約 3Gb/s の通信まで行なえる。

Time Stamp Option は、再送タイムアウトを決定する際の RTT の計測をより高い精度で行なうために追加されたオプションである。しかし、実際の4.4BSDに実装された Time Stamp Option で用いられる時刻は 500ms の粒度であり、このオプションを使っても、再送タイムアウト値の単位は依然として500msである。

3 従来の TCP のアノマリ

本節では、Reno+Extension TCP で最大ウィンドウサイズを広く変化させた時のスループットの変化を調べる実験と、そこで観測されたアノマリとその原因について述べる。

3.1 実験環境

図1のようなネットワーク環境を構築して実験を行なった。

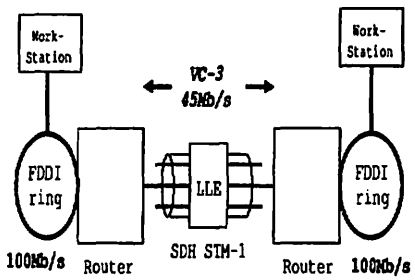

図 1: 実験環境

実験に使用した二台のワークステーションはルータの FDDI(Fiber Distribution Data Interface) リングにつながっており、最大ウィンドウサイズを 1034bytes から 1Mbytes まで変化させることができる。また、動作中の TCP コード内部の変数の変化を観測するために TCP に関するデバッグ情報を取れるようにカーネルに改造を施してある。

また、二台のルータの間は LLE(long Link Emulator: 遅延発生装置) を介して SDH STM-1(155Mb/s) でつながっている。SDH STM-1 は、仮想的に 3 本の VC-3(45Mb/s) にわかれており、このうちの一本で結んである。このリンクでは、LLE により 100ms までの伝送遅延を発生できる。

3.2 アノマリ発生時の TCP の動作解析

図2は、最大ウィンドウサイズを 1536 bytes から 524 Kbytes まで変化させたときのスループットの変化である。LLE で設定されている RTT は 10ms である。

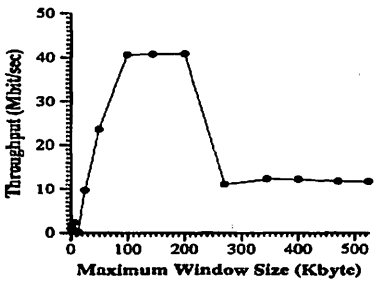

図 2: 最大ウィンドウサイズ vs スループット

最大ウィンドウサイズが 90Kbytes までの範囲ではスループットと比例しており、90Kbytes からはスループットは約 41Mb/s で飽和しはじめて、200Kbytes になるまで安定している。最適なウィンドウサイズはスループットが飽和しはじめる点であるので、図2から 90Kbytes と読みとれる。デバッグ情報によると、パケットが安定して流れている時の平均の RTT は 16 ms であり、TCP の絶対帯域が実測値の 41Mb/s 程度とすれば、DB 積は 82 Kbytes である。したがって、図から読みとれる最適な最大ウィンドウサイズと DB 積とはほぼ一致する。最大ウィンドウサイズが 200 Kbytes を越えるあたりからスループットの低下が起こり始めている。ウィンドウサイズが DB 積に対して大きすぎる場合には、スループットの低下が起こることがわかる。

図 3は、送信側で観測した、アノマリが起こっている時のシーケンス番号と ACK 番号のそれぞれの増加の様子を表している。急激な転送と転送停止が交互に行なわれるという無効な動作を繰り返している。この無効な動作の繰り返しがトータルのスループットを低下させる原因である。

図4は、図3のグラフ中の最初の転送停止前後の部分を拡大したものである。測定から得られた TCP のデバッグ情報を解析した結果とあわせ、アノマリが起こっている時の TCP の動作をこの図を用いて説明すると以下のようになる。

1. 送信側のウィンドウサイズは大きいので、実際に使用可能な帯域以上にパケットをネットワークに送り込んでしまうために輻輳が起こり、送信側のパケットの連続したシーケンス番号の複数のパケットの消失が起こる (図4の白マル)。

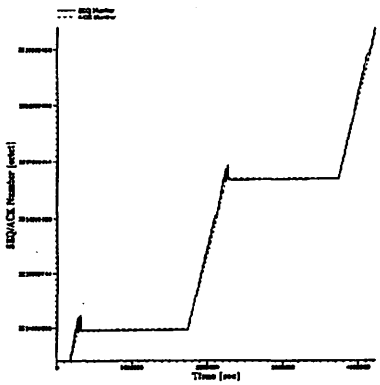

図 3: アノマリ発生時の順序番号の増加

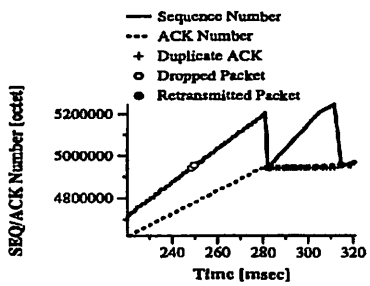

図 4: 転送停止直前

2. 送信側パケットの消失のため、受信側は期待しているものよりも大きなシーケンス番号を持つパケットを受信する。これらを受けとるたびに受信側は次に期待しているシーケンス番号をセットして直ちに送信側に ACK を返す。これらは Duplicate ACK と呼ばれる (図 4の +)。

3. 消失したパケットが送り出された直後の送信側のウィンドウは十分大きいため、Duplicate ACK により輻輳が検知される前に、先行して多くのパケットが送られる (図 4の 250 ～ 280ms)。これらのパケットの数だけ Duplicate ACK が返されることになる。

4. 送信側は、受信側から送られてきた Duplicate ACK の最初の 3 個を受けとると、消失したパケットを回復するために Fast Retransmit アルゴリズムによって当該パケットだけの再送を行なう (図 4の黒丸)。また、輻輳状態から回復した後に早くもとの転送速度に戻るように、輻輳ウィンドウをもっとも小さい値に戻すのではなく、3 個の Duplicate ACK が返った時点での輻輳ウィンドウの半分の値にされる (Fast Recovery)。

5. 送信側は、すでに送られたものの直後のシーケンス番号のパケットから送信を再開しようとする。ところが、Duplicate ACK が三つ続いた直後は、輻輳ウィンドウが小さ過ぎてパケットの送信ができない。Duplicate ACK は、ネットワークの潜在的な帯域と見做されるため、送信側では ACK が返るたびに輻輳ウィンドウが t_maxseg ずつ増やされる。こうして増加した輻輳ウィンドウが Fast Retransmit 直前の値を越えるとパケットの送信が再開され、いくつかのパケットが送られる (図 4の 305 ～ 315ms)。Fast Retransmit で再送したパケットに対する ACK が返ると、大きくなり過ぎた輻輳ウィンドウを是正するために、ssthresh + t_maxseg に減じられる。

6. 送信側から再送したパケットの次のパケットも消失しているので、再びこのパケットに対する Duplicate ACK が続く。ここで再び Fast Retransmit / Fast Recovery アルゴリズムが働く。しかし、先行して送られたパケットが一度目に比べると少ないため、輻輳ウィンドウが十分に大きくならないうちに Duplicate ACK は尽きる。そのため送信側からのパケットの送信が全く行なわれなくなり、再送タイムアウトが起こるまでの間、すなわち、1 ～ 1.5 秒間転送は完全に止まる。このためにスループットが低下する。

以上が単一ストリームでスループットが低下する現象の説明である。なお、複数ストリームの場合にも、各々のストリームでは上に説明したのと同様の現象が見られるが、どれかのストリームが転送を停止している間に、他のストリームが転送を行なうといったように、各々のストリームが交互に転送を行なうため、結果として帯域を有効に使うことになる。したがって、単一ストリームでのスループットの低下を解決することが課題となる。

4 MAB (最小絶対帯域検出アルゴリズム)

本節では、従来の TCP アノマリを解決して、単一ストリームでの安定かつ高速な通信を行なう目的で我々が新たに開発した MAB (Minimum Absolute Bandwidth detection algorithm) について説明する。

4.1 アルゴリズムの目的

MAB は、ネットワークの絶対帯域を超えないように送信を制限して輻輳を予防することで転送の self clocking が失わないようにし、帯域を効率的に使うことができるようにすることを目標とする。また同時に、この改造を施していないホストとの通信でも効果を得る

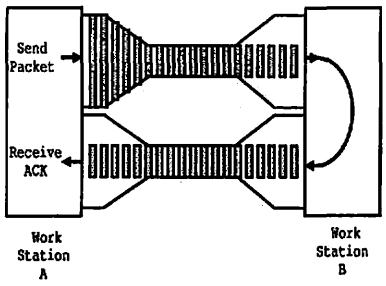

図 5: 使用可能帯域の見積り

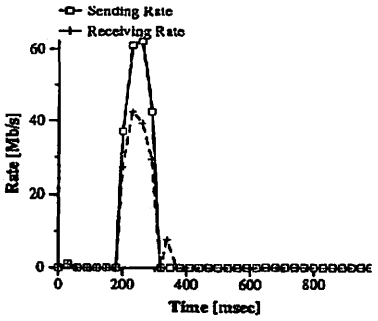

図 6: 送信レートと受信レート

ために、TCP のコードの改造は送り側の改造だけで済むようにすることと、TCP プロトコル自体には変更を加えないことを指針とした。したがって、Reno + Extension を実装したホストであればどのホストに対する通信でもアノマリ領域でのスループットの向上を期待できる。

TCP のスループットの向上をはかるには、状況に応じて適切に最大ウィンドウサイズを規制して、輻輳が起こらないようにパケットの送信を制御するアルゴリズムを組み込む必要がある。この場合の適切な最大ウィンドウサイズとは DB 積である。

最適な DB 積を求めるには、

1. end to end のネットワークの絶対帯域
2. 正確な RTT

の二つを見積もる必要がある。DB 積が求められたら、

3. DB 積を用いて輻輳を予防する。

以上が本アルゴリズムの目的である。

4.2 ネットワークの絶対帯域の見積り

これまで TCP がネットワークの絶対帯域を見積もる方法はなかった。我々は、送信側に返る ACK のスピードからその時点で使用可能な絶対帯域を見積もる手法を提案する。

図 5 に示すように、送信側から送られたパケットがネットワークを通過して受信側に届く時には、転送レートはその経路上のもっとも細いリンクの帯域まで落ちていると考えられる。受信側に到着したパケットに対して ACK が送信側に送り返されるので、ACK から計算できる単位時間あたりのパケットの量は、ネットワークを通して受信側に到着したパケットの転送レートである。この転送レートの最大値が、このコネクションにおける使用可能な最大帯域、すなわちネットワークの絶対的な帯域と考えられる。

図 6 は、図 4 に示したグラフの近辺での送信側でのパケットの送信レートと、返ってきた ACK を元に計算された転送レートを 30ms ごとにそれぞれ計算してグラフにしたものである。この実験環境でのリンクの絶対帯域は 45Mb/s であり、ACK を元に計算された転送レートはこの値以内に収まっていることがわかる。この転送レートの最大値が絶対帯域を表すと考えられる。それに対して送信レートの最大値はリンクの絶対帯域を越えている。従来の TCP アルゴリズムが絶対帯域を越えてパケットを送り込もうとしていることが、このことからも明らかである。

受信側から返ってくる ACK をもとに現在のネットワークの絶対帯域を調べる機構を TCP に追加するための準備として、関数 `swnd_timer()` と変数 `t_acked`, `t_maxacked`, `ms_period` を導入する。`swnd_timer()` は、周期 `ms_period` (ms) で呼び出されるタイマ関数であり、`t_acked` は、1 周期の間に受信側への送達が確認されたデータ量を格納する変数である。変数 `t_maxacked` には`t_acked` の最大値が格納される。`t_acked` と `t_maxacked` の単位は byte である。

これらの関数と変数を使って、ネットワークの絶対帯域を以下のようにして調べる。

1. ACK パケットが受信されると、受信パケットを処理する TCP の関数である`tcp_input()` が起動される。`tcp_input()` では、ACK パケットにより知らされた、受信側で送達を確認されたデータ量 (acked data) だけ `t_acked` を増加させる。
2. `tcp_input()` とは独立に `ms_period`(ms) ごとにタイマー関数 `swnd_timer()` が呼び出される。この時点で、`t_acked` は、最近の `ms_period`(ms) の間に送達が確認されたデータの量を表す。`t_acked` は `t_maxacked` と比較され、`t_acked` の方が大きければ`t_acked` を新しい `t_maxacked` とする。`t_maxacked` を 周期 `ms_period` で割ると、帯域を測り始めてからこれまでの間のネット

ワークの最大帯域となる。この後、t_acked は次の周期の送達確認データ量を記録するために 0 でクリアされる。

関数 swnd_timer() の呼びだし周期 ms_period は、長すぎても短かすぎても正確なものが得られない。本アルゴリズムでは、呼びだし周期ms_period を経験的にもっともよい値と思われる 30ms ごととした。

4.3 RTT の正確な計測

従来の TCP では RTT の計測の粒度は 500ms であった。これは、500ms ごとに呼ばれる tcp_slowtimo() の中の変数 tcp_now を一ずつ増加させることによって RTT の計測を 行なっていたためである。しかし、最も小さい場合には 1ms 程度となる実際の RTT の値を考えると、最適な DB 積を求めるためにはこの精度では全く不十分である。そこで、Time Stamp Option を利用して 1ms の精度で RTT の計測ができるように、TCP コードの改造を行なった。

TCP では、Time Stamp Option を用いて RTT は以下のように計測される。Time Stamp Option が使用可能な時、パケットの送信にあたって TCP のオプションフィールド (4bytes) である TSval にシステム起動時からの時刻を表すシステム内部変数 tcp_now の値が記録される。受信側は、受けとったパケットのオプションフィールド TSval の値を ACK パケットのオプションフィールド TSecr に入れて送り返す。送信側が ACK を受信すると TSecr に記録された時刻と tcp_now の差分をとる。この差分が RTT である。tcp_now は、タイマ関数 tcp_slowtimo で 500ms ごとにカウントアップされる変数であるため、計測の粒度は 500ms である。そこで我々は tcp_now に 1ms 単位のシステム時刻を記録するように改造した。以下は、改造された関数 tcp_input() の中の RTT の計測部分のコードである。

```
tp->t_rtt=tcp_now-ts_ecr;
if(tp->t_rtt < tp->t_minrtt)
        tp->t_minrtt = tp->t_rtt;
```

tp->t_rtt, tp->t_minrtt は TCP コネクションごとに用意される変数である。tp->t_rtt には、直前に計測された RTT が記録されている。また、tp->t_minrtt は MAB のために新しく用意された変数で、RTT の最小値を記録しておくものである。

4.4 MAB の動作

ここでは MAB の動作の詳細を説明する。

最小帯域ウィンドウの決定

MAB を組み込んだ TCP では、コネクションが確立した直後は従来のアルゴリズムでの動作と全く同じように通信を開始する。

輻輳が起こってパケットの消失が起こると、Fast retransmission / Fast Recovery アルゴリズムが働き始める。これを契機として最小帯域ウィンドウを設定し、次回の輻輳を予防を試みる。具体的には、dup-ACK を表す変数tp->t_dupacks の値が tcprexmtthresh+1 となった時に最小帯域ウィンドウが設定されるようにしている。tcprexmtthres は、Fast Retransmit が行なわれるまでに受けとる ACK の数を表す変数である。

帯域を満たすのに必要十分なウィンドウの値は DB 積に等しいから、最小帯域ウィンドウには DB 積 を設定する。tp->t_maxacked を swnd_timer() の呼び出し周期ms_period で割ったものが、この輻輳が起こるまでの間に計測されたネットワークの絶対帯域である。また、RTT の最小値 t_minrtt は、伝送路の遅延を表すと考えられる。こうして得られた絶対帯域と RTT の積を最小帯域ウィンドウに設定する。また、tp->t_maxacked は次回の絶対帯域の見積りのために 0 にクリアされる。

最小帯域ウィンドウを決定する部分の TCP コードを以下に示す。

```
if( tp->t_dupacks == tcprexmtthresh+1 ){
  tp->snd_swnd =
   (tp->t_minrtt * tp->t_maxacked)/ms_period;
   tp->t_maxacked = 0;
}
```

最小帯域ウィンドウによる送信制限

決定された最小帯域ウィンドウ tp->snd_swnd を TCP パケットを送信する関数tcp_output() の中のウィンドウを決定するコードに以下のように組み込むことより、高速ネットワークの帯域と遅延にあった適正な量のパケットが送信される。

```
win = min( min(tp->snd_wnd, tp->snd_cwnd),
                                tp->snd_swnd);
```

tp->snd_wnd と tp->snd_cwnd は、それぞれ受信側から通知されてきたウィンドウ、輻輳ウィンドウである。この三つのウィンドウのうち最も小さいものが送信側のウィンドウとなる。

最小帯域ウィンドウのアップデート

他のトラフィックが存在し、複数の経路を通る可能性のある通常のネットワークで本アルゴリズムを使う

場合には、最小帯域ウィンドウのアップデートが必要となる。

たまたまネットワークが混んでいたり、経路の変動で細いリンクを通っていた場合には、ネットワークのトラフィックが減少した時や、より広い帯域の経路を通るように変わった時に設定されるべき値に比べて、絶対帯域の見積りは小さい値になってしまう。したがって、絶対帯域と RTT により計算される静的ウィンドウの値も小さいものとなる。

これまで説明してきたアルゴリズムでは、一旦最小帯域ウィンドウが設定されると、次に輻輳が起こるまでは最小帯域ウィンドウは更新されることはない。したがって、経路の変動などで絶対帯域が増えても、最小帯域ウィンドウで設定された帯域を越えてパケットを送信することができないので、帯域を有効に使うことができなくなる。これを解決するには、最小帯域ウィンドウの値を適切なものかどうか定期的に調べ、アップデートする機構が必要である。本論文の最小帯域ウィンドウアルゴリズムでは、この機構は未実装であるため、以下にこのアルゴリズムの方針を述べるにとどめる。

1. 一定周期ごとに tp->snd_swnd の値を少しだけ大きくする。
2. RTT 後に ACK により計算される受信レートが以前より増えたら、絶対帯域が増えたと判断して tp->snd_swnd の値を大きくしたままにする。また、受信レートが変わらなかったら、tp->snd_cwnd を元の値に戻す。

5 評価

本節では、従来の Reno+Extension TCP と、MAB アルゴリズムを実装した TCP をホストに実装して測定を行なった結果を用いて、MAB の単一ストリーム通信における効果について述べる。

5.1 TCP の振舞い

図7は、Reno + Extension TCP と MAB を組み込んだ TCP のシーケンス番号の増加を表したものである。最大ウィンドウサイズはどちらも 1048576 bytes であり、RTT は LLE により 10ms に設定されている。どちらの TCP もコネクションが確立してから 0.5 秒以内に輻輳を起こして再送タイムアウトとなるまで転送が停止している。Reno + Extension では、この後に転送が再開されてからしばらくして再び転送の停止が起こり、定期的にこの動作を繰り返している。それに対し、MAB を組み込んだ TCP では順調に転送が行なわれている。測定データによると、Fast Retransmit / Fast Recovery が働く直前の t_maxacked は 163400 bytes であり、

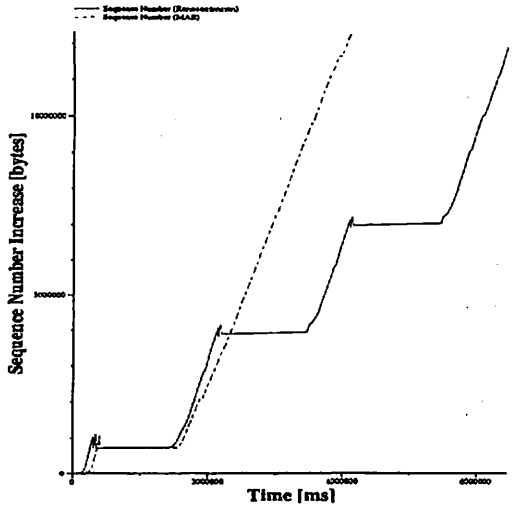

図 7: シーケンス番号の増加

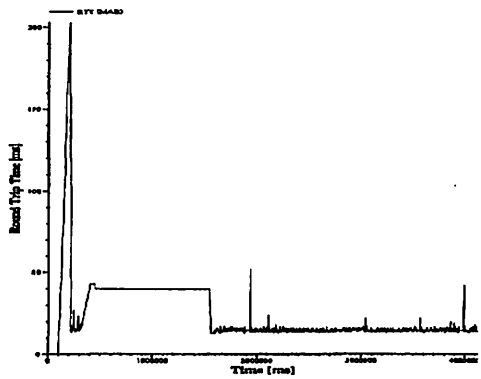

図 8: MAB での RTT の変化

t_rttmin は 14ms である。また、再送タイムアウト後に設定される最小帯域ウィンドウの値は 76253 bytes である。

図8は、MAB を組み込んだ TCP での RTT の計測値の変化である。このグラフは図7の MAB のシーケンス番号の増加に対応したものである。RTT は、はじめに輻輳の影響で急速に大きくなり、転送再開後は約 16ms に落ち着いている。
t_rttmin の値が LLE による伝送遅延の設定に一致していないのは、パケットを高速に送り出すことにより、ホストやルータの負荷が上がりバッファに溜っている時間が長くなるためであると考えられる。

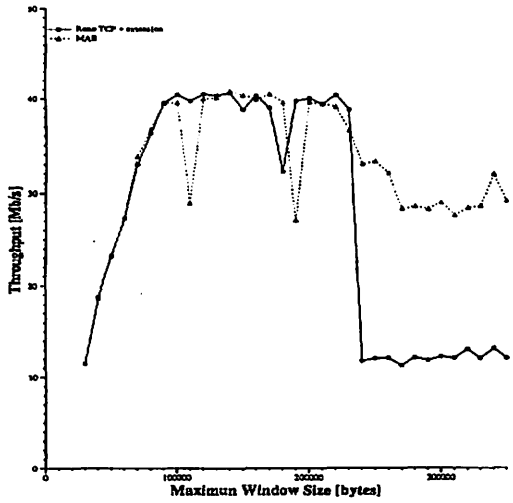

図 9: MAB vs Reno+Extention

5.2 スループット

図 9は、Reno+Extension と MAB の両方で 30 K bytes から 350 Kbytes まで最大ウィンドウサイズを変えてスループットを測定したものである。各々の測定点では、16 K bytes のパケットを 2000 回、総計 31.25Mbytes のデータ転送を行なっている。

どちらの TCP も最大ウィンドウサイズが 220 Kbytes 以下の領域では、ほぼ同様のスループット曲線が得られる。アノマリの発生し始める 220 Kbytes 以降の領域では、Reno+Extensino TCP が約 12Mb/s のスループットしか得られないのに対して、最小帯域ウィンドウを組み込んだ TCP では平均して 30Mb/s のスループットが得られる。

MAB ではネットワークの帯域を見積もるために、はじめの一回だけは Reno + Extension TCP と同様に輻輳が起こるまでパケットを送り込む。そのため、この領域では転送の停止が一度は起こる。したがって、スループットはネットワークの絶対帯域までは到達できないが、一回の TCP コネクションでのデータ転送量が多くなるにつれ、スループットはネットワークの絶対帯域に近付く。

6 おわりに

長距離高速インターネットにおいて、TCP のスループットを向上させる目的で過度にバッファサイズを増大させると、既存 TCP アルゴリズムが想定していないアノマリを発生させることを示し、その動作の詳細を説明した。また、このアノマリを回避するために、ACK によるネットワークの絶対帯域見積りに基づいた MAB を提案し、このアルゴリズムを実装した TCP を使った実験によりその効果を示した。

今後は、今回の実験では未実装である、最小帯域ウィンドウのアップデート機構の組み込みを行なう予定である。また、本アルゴリズムの実装に際して、ms_period(ms) の周期のタイマを増設したが、この改造によりホストシステムにどの程度の負荷がかかるかについても検討を行なう。さらに、このアルゴリズムが実際のインターネットで使われた場合の、他の TCP トラフィックとの親和性を検証して、実ネットワークの使用した場合にも有効であることを確認する予定である。

参考文献

[1] J. Postel: Transmission Control Protocol, RFC793, 1981

[2] K. Fall, S. Floyd: Simulation-based Comparisons of Tahoe, Reno, and SACK TCP, Computer Communications Review, July 1996

[3] V. Jacobson, R. Braden: TCP Extensions for Long-Delay Paths, RFC1072, 1988.

[4] V. Jacobson, R. Braden, L. Zhang: TCP Extension for High-Speed Paths, RFC1185, 1990.

[5] V. Jacobson, R. Braden, D. Borman: TCP Extensions for High Performance, RFC1323, 1992.

[6] V. Jacobson: Congestion Avoidance and Control, SIGCOMM'88, pp.314-329, 1988

[7] V. Jacobson: modified TCP congestion avoidance algorithm, end2end-interest mailing-list, 1990.

[8] Gary R. Wright, W. Richard Stevens: TCP/IP Illustrated, Volume 2, 1995

[9] Takeshi Kugimoto, Ken-ichoro Murakami, Yoshiji Amagai, Atsuko Oka: An experience with a Long Fat Pipe, The 9th ICOIN, 1994

[10] 天海良治、村上健一郎、釘本健司、岡敦子、伊藤正樹、後藤滋樹、伊藤光恭: 長距離超高速インターネットの特性解析, マルチメディア通信と分散処理研究会

[11] Yoshiji Amagai, Ken-ichiro Murakami, Takeshi Kugimoto, Atsuko Oka: An analysis of Behavior in a High-speed Long-distance Internet, The 2nd JAIN Consortium Symposium

[12] L. Kleinrock : The Latency/Bandwidth Tradeoff in Gigabit Networks, IEEE Communication Magazine, Vol.30, No.4, pp.36-40, 1992