TCP 詳説

西田 佳史

（(株）ソニーコンピュータサイエンス研究所）

1999 年 12 月 14 日

Internet Week 99　パシフィコ横浜

（社）日本ネットワークインフォメーションセンター編

目次

1　概要

TCP には、通信経路の性質やネットワークの混雑状況を推測しながら通信を行う手法が採用されるなど、様々な環境に適応できるよう柔軟に設計されています。

この講演では、広域ネットワークで信頼性の高い通信を実現する要素技術の 1 つである TCP の基本設計や転送効率を高めるために導入されたアルゴリズムの紹介、輻輳制御機構の仕組みについて説明します。

また、Windows98 や Linux、Solaris などで実装が始まっている広帯域、高遅延ネットワークに対応する新しい機能も紹介します。

2　TCP 基本機能

TCP(Transmission Control Protocol)とは、トランスポート層プロトコルで転送の制御を行います。IP の上位プロトコルに位置しており、トランスポート層プロトコルには TCP のほかに UDP もあります。UDP はシンプルなプロトコルであるため、アプリケーションの方で通信の状態を細かく制御しなければなりません。これに対し、TCP は高信頼性サービスを提供します。自分自身で細かく速度を調節する機能があるため、アプリケーションが通信の状態を把握する必要はありません。UDP は転送レートを制御する機能を持っていないため、音声とか映像とかの情報を流すのに適していると言われています。これに対し、TCP はファイル転送に最適と言われているプロトコルです。

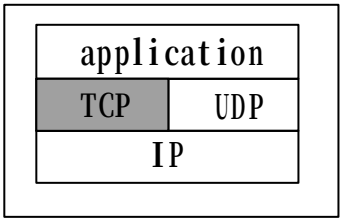

図 1：プロトコル階層図

TCP の仕様は、基本スペックが RFC793 で、輻輳制御アルゴリズムが RFC2581 で定義されています。TCP の特徴は、①非構造化ストリーム、②全二重通信、③コネクション指向、④高信頼性サービスの 4 つが挙げ

られます。

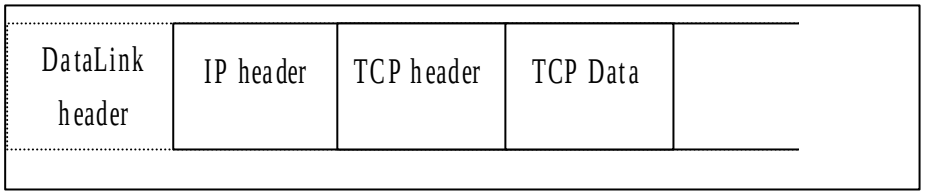

図２：ヘッダフォーマット

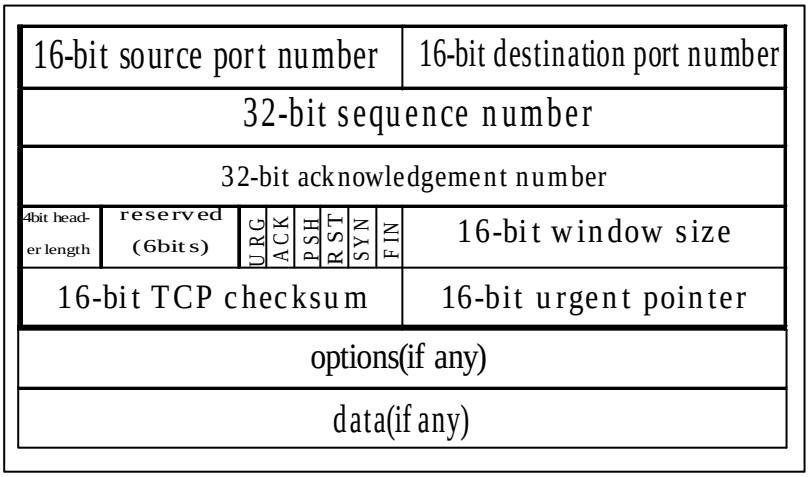

図３：パケットフォーマット

2.1 非構造化ストリーム

非構造化ストリームとは TCP が扱うデータが構造を持たないということです。単なるビット列と見なし、送信者が送ったストリーム列をそのまま受信者へ渡します。その際のデータ構造の解釈は上位層に任せているわけです。

また、TCP は通信経路に最適なセグメント長を推測し、送信者が送ったストリーム列を通信経路に最適な大きさで、アプリケーション構造を全く考慮せずに分割します。

このほか、UNIX のファイルシステムとは、UNIX のファイルシステムがビットストリームでファイルの中に特別な構造を持っていないという点で TCP との親和性が高いという理由で、UNIX への実装を通じて TCP/IP の実装が広まっていきました。

2.2 全二重化通信

全二重化通信とは、通信する両者が同時にデータストリームを送受信できることです。それは Piggyback を使うことによってさらに効率を高められます。

Piggyback は通信する両者が同時にデータストリームを送信しあっているなかで、一方から送信情報に対するフィードバック情報を逆方向のデータストリームを使って伝達する仕組みです。

このほか、片方向だけのデータストリームを停止でき、半二重通信の状態を作り出すことも可能な設計になっています。

2.3 コネクション指向

2.3.1 ネゴシエート

コネクション指向は、データの送信の前に送信側と受信側がネゴシエートして、受信側の能力や通信経路を把握し、効率の高い通信を実現する仕組みです。

このネゴシエートによって受信側が最大のウィンドウサイズを送信側に通知することで、送信側は受信用のバッファがどの程度用意されているかを把握できます。

また、イーサネットの場合、1 度に送ることができるパケットの固まりは 1500 オクテット、光ファイバの場合、4096 オクテットなど接続されたインタフェースの MTU などを通知して、通信経路に最適な最大パケット長 MSS(Max Segment Size：最適パケット長)を決定するのに利用します。

2.3.2 バーチャルサーキット

また、ネゴシエートによって、たとえば、送信者と受信者の間の仮想のパイプを作るように送信者と受信者の間の通信をセットアップします。この仮想のパイプをバーチャルサーキットといいます。アプリケーション側は、バーチャルサーキットの中にデータを投げ入れさえすれば、データを必ず届けてもらえます。

2.3.3 コネクションの確立

コネクションでは IP アドレスとポート番号のペアで相手側を識別します。このため、1 つの Web サーバに対して複数の TCP コネクションを確立するということも可能になります。

コネクションの識別情報はすべての TCP パケットに含まれています。IP

アドレスとポート番号のペアで認識します。これがコネクションの端点となります。

コネクションは、3Way handshake といわれる方法で確立します。

3Way handshake は、①クライアント側から「確立要求(SYN)」を送信、②サーバ側は「確立要求(SYN)」と「確立要求の確認応答(ACK)」を Piggyback によって同時に送信、③クライアント側からの「確立要求の確認応答(ACK)」の送信、の手順を踏みます。SYN と ACK は TCP のヘッダフォーマットのフラグに含まれています。SYN フラグをオンにして通信すれば、そのパケットは確立要求をしたことになります。

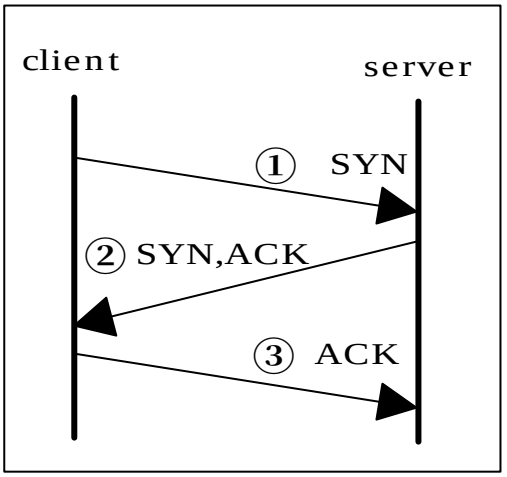

図 4：3Way handshake

2.3.4 コネクションの終了

コネクションの終了はハーフクローズという方法で行います。ハーフクローズでは、コネクションを片方ずつ閉じます。このため、4 つのパケットが伝送されることになります。

最初にクローズ要求を出す側を Active Close といいます。Active Close する側は最初にコネクションの終了要求 FIN を送ります。それに対して、FIN を受け取った側は確認応答 ACK を送信します。Active Close する側は、最後の FIN を受信して最後の ACK を送った後、一定時間待ってコネクションを終了します。この待ち時間は MSL(Max Segment Lifetime：最大セグメント生存時間)の 2 倍の時間、つまり 2MSL です。一般的にはサーバの負担を軽くするため、通常はクライアントが Active Close するのがよいといわれています。

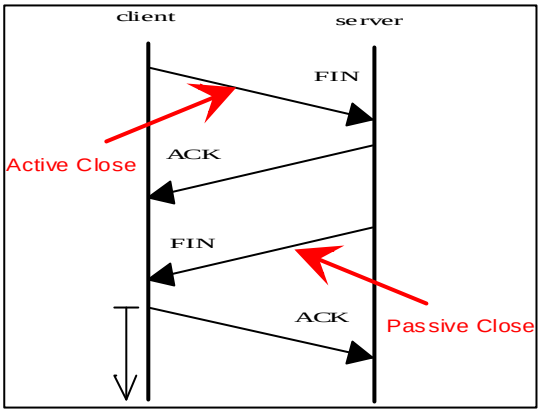

図 5：ハーフクローズ

TCP のコネクションに問題がなかった場合の典型的なコネクションの確立とコネクションの終了は図 6 のようになります。

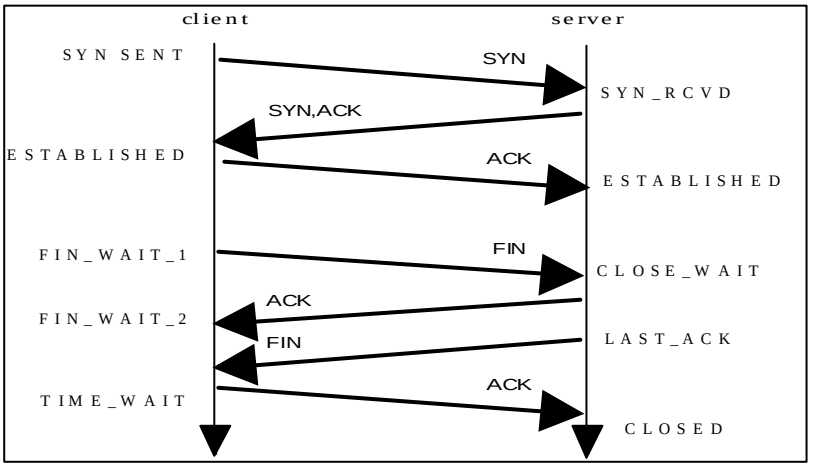

図 6：標準的なコネクション

2.4 高信頼性サービス

高信頼性サービスでは、送信したパケットが喪失した場合に、喪失したパケットを検出し、それを再送するなどの機能を提供します。それらの機能を提供するための様々な仕組みを紹介します。

2.4.1　シーケンス番号

パケットに付けられたシーケンス番号は、パケットの順番を保証し、喪失したパケットを検出するための指標になります。最初のネゴシエートで初期シーケンス番号を決定し、次に、初期シーケンス番号＋ストリーム中の位置でシーケンス番号を表現していきます。特に初期シーケンス番号はセキュリティの面からも重要な役割を果たしています。

アプリケーションが 2500 バイトのデータを TCP に渡す際に、最適なパケットサイズが 500 バイトで、初期シーケンス番号を 10000 とした場

合、各パケットのシーケンス番号は図 7 のようになります。

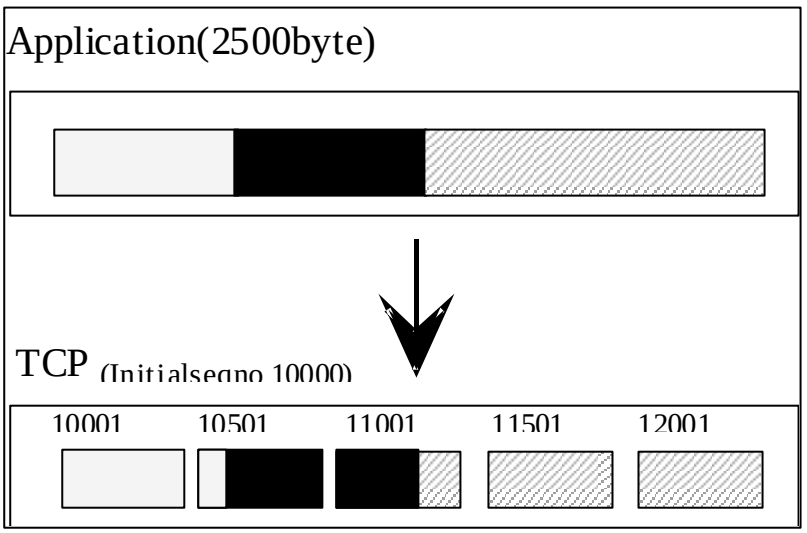

図 7：シーケンス番号

2.4.2　再送機能

喪失したパケットは再送するための機能として、TCP では ACK(ACKnowledgement：確認応答)を利用した累積確認応答方式を採用しています。

累積確認応答方式は、受信側が一定数受信したパケットのシーケンス番号が連続しているかを確認し、次に必要なパケットのシーケンス番号を送信側に通知する仕組みです。確認応答のためのパケットの送信数量を抑制するため、パケットの受信ごとに確認応答を送信せずに済む設計となっています。

累積確認応答では、受信側は「連続して到着した最大の seqno(sequence number：シーケンス番号) ＋1」の ACK を送信します。

この場合、喪失したパケットの再送要求も累積確認応答に基づきます。このため、仮に喪失したパケットが 1 つだけで、それ以降にシーケンス番号が連続している一定数のパケットがあったとしても、再送要求では喪失したパケット以降のすべてのパケットの再送を要求してしまうという欠点があります。

再送するタイミングは、①再送タイマがタイムアウトする、②重複する確認応答が一定数以上到着する、の 2 つのケースで決定され、それをきっかけに再送が開始されます。

①再送タイマのタイムアウト
パケット送信後、一定時間待っても累積確認応答が返信されない場合、送信側は再度パケットを送信します。

②重複する確認応答の到着
受信側は喪失したパケットの再送要求を累積確認応答に基づき送信します。喪失したパケットが受信側に到達しない限り、受信側は何度でも同じ確認応答を送信しますので、送信側は重複する確認応答を受け取ることになります。この確認応答が一定数以上到着したら、それに対応するパケットの再送を開始します。

2.4.3 チェックサム機構

TCP は、パケット全体が正しく受信側に届いているかの認証手段としてチェックサム機構を持っています。TCP のチェックサム機構は強力であるため、IPv6 ではチェックサムの計算をしない設計になりました。IPv4 はヘッダだけチェックサムを計算しています。

チェックサム機構はデータ部に仮想的な IP ヘッダを付加して、ヘッダ部、データ部全体でチェックサムを計算します。仮想ヘッダフォーマットは図 8 のとおりです。

<table>
<tr><td colspan="3">32bit sender IP address</td></tr>
<tr><td colspan="3">32bit receiver IP address</td></tr>
<tr><td>0</td><td>proto number</td><td>TCP segment length</td></tr>
</table>

図 8：仮想ヘッダフォーマット

2.4.4 フロー制御機構

フロー制御機構では通信状態に合わせてデータの転送速度を調節します。データの転送速度の調節は、①相手の処理速度に合わせる、②ネットワークの状態に合わせる、という 2 つの機能を持っています。

①相手の処理速度に合わせる場合
受信者が処理できない速度では転送しません。

②ネットワークの状態に合わせる場合
低速回線では低速に、高速回線では高速に通信します。それを輻輳制御といいます。特に、ネットワークが混雑している時は速度を落し、輻輳崩壊を回避するようにするのは TCP の非常に重要な技術で、現在でも様々な研究がなされています。

TCP のデータの転送速度の調節は、スライディングウィンドウ方式によって行います。

スライディングウィンドウ方式では、受信側がウィンドウサイズというパラメータを送信側に通知することで、送信側は確認応答を受け取るまでの間に連続して送信できるパケットの数をウィンドウサイズに基づき決めることができます。

ウィンドウサイズは、値を増加させることで転送レートを上げます。逆に、値を減少させることで転送レートを下げます。

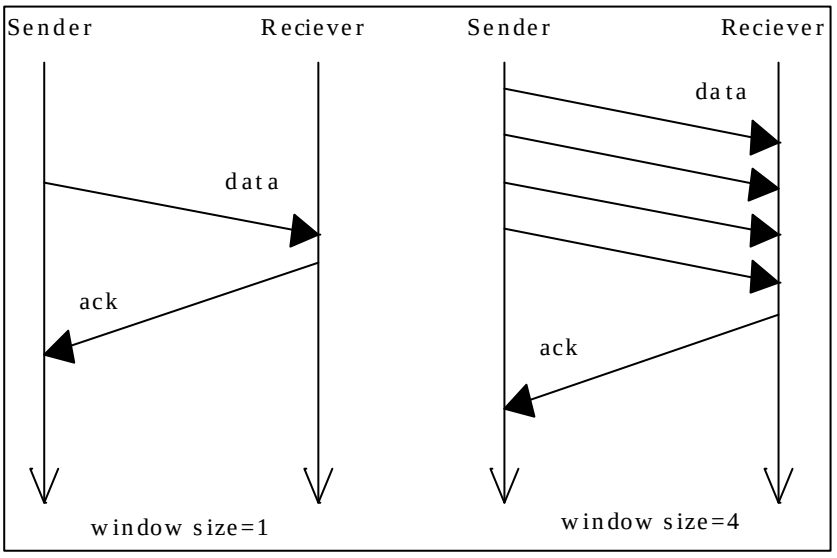

図 9：ウィンドウサイズが異なる 2 つのデータパケット転送例

図 9 の転送例のように、ウィンドウサイズが 4 の場合だと、ACK を受け取るまでに 4 つのパケットを送信できますので、ウィンドウサイズが 1 の場合よりも通信効率が高いと言えます。

スライディングウィンドウ方式によるデータの転送速度の調節は、受信側が最大の転送レートを決めて、送信側はその転送レートの範囲内で、ネットワークの状態を考慮して送信するパケットの数を調整します。

3　TCP 詳細機能

ここでは、TCP のタイマ機構と、そのタイマによって遅延確認応答アルゴリズムや Nagle アルゴリズムがどう動作するかを説明します。さらに、シリーウィンドウシンドロームと、SYN、ACK、FIN 以外の TCP のフラグについても紹介します。

3.1　TCP のタイマ機構

3.1.1　基本機構

TCP を普及した BSD の場合だと、TCP に実装されている基本的なタイマ機構はスロータイマとファーストタイマの 2 つと言えます。

①スロータイマ
スロータイマは遅い間隔で起動されるタイマです。BSD 実装の場合だと 500msec ごとに起動します。このときに、再送タイマ、パーシストタイマ、キープアライブタイマ、2MSL タイマなどの計算を行っています。

②ファーストタイマ
ファーストタイマは早い間隔で起動されるタイマです。BSD 実装の場合だと 200msec ごとに起動します。遅延確認応答アルゴリズム、Nagle アルゴリズムなどに利用されます。

3.1.2　再送タイマ

再送タイマは、パケット送信後、パケットの喪失などで一定時間待っても累積確認応答が返信されない場合、送信側が再度パケットを送信する場合に利用します。

パケットが転送される度にセットされます。再送タイマが終わるとパケットの再送が行われます。この場合、適切なタイムアウト値を選択することが重要です。タイムアウトが長すぎると転送効率の低下を招きます。逆にタイムアウトが短すぎると、正しく届いているにも関わらず不必要な再送が発生し、ネットワークに余分な負荷をかけます。

(1)再送タイマのタイムアウト時間

Round Trip Time から算出します。Round Trip Time とは、データを送信してから送達確認を受信するまでの時間です。長い回線の場合は、Round Trip Time は長くなりますので、長めに待ち、反対に短い回線の場合、Round Trip Time は短くなりますので、短めに再送タイマのタイムアウトを計算・設定することによって効率の良い通信を実現しています。

(2) Round Trip Time の計算

Round Trip Time の計算で難しいのは、ネットワークが混んでいると、通信中の Round Trip Time の値が変動しているためです。Round Trip Time の値が常に一定であれば計算しやすいが、ネットワークが混んでくるとなかなかパケットが届かずに ACK が遅れて返ってくるなどで Round Trip Time の値は非常に大きくなります。2 倍、3 倍になるケースもあります。

Round Trip Time の計算は、Round Trip Time の間にスロータイマが何回起動されたかをスロータイマの割り込み回数の計測によって数え、計算に役立てています。

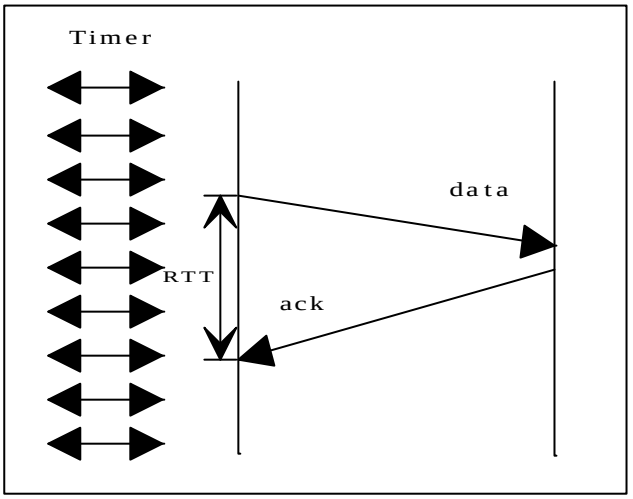

図 10：スロータイマの割り込み回数の計測

多くの実装では、ウィンドウサイズの場合だと、1 ウィンドウに 1 個だけ測定用のパケットを作り、1RTT(Round Trip Time)に 1 回計測する方式を採用しています。

Round Trip Time は変動するのでできるだけ計測値の平滑化します。

実測値を rtt、平滑化した値を srtt とした場合、平滑化した値 srtt は以下の計算式で導き出されます。

$$srtt = \alpha \times srtt + (1 - \alpha) \times rtt$$

実測値 rtt には変動幅があるため、安定した通信にするために再送を行います。α の推奨値は 0．9 といわれています。平滑化した値 srtt を利用して再送のタイムアウト時間を決めていくのが TCP の再送アルゴリズムです。

(3)再送タイマのタイムアウト値の計算

昔のアルゴリズムでは再送タイマのタイムアウト値は「rto ＝2×srtt」の計算式で決めていました。しかし、その後の研究で Round Trip Time が急速に変動することが分かり、Round Trip Time の急速な変動に対し、この計算式を採用したアルゴリズムは弱いという欠点がありました。

このため、新しいアルゴリズムとして平均偏差を採用するという提案がなされました。UNIX のカーネルの中で計算するためには計算が簡単であることが求められ、平均偏差の計算は標準偏差より簡単であったためです。新しいアルゴリズムでは再送タイマのタイムアウト値は「rto ＝平均 rtt ＋4×平均偏差」の計算式で決めます。

再送タイマには、指数バックオフという機能があります。

指数バックオフは連続して再送タイマのタイムアウトが起こる場合、タイムアウト毎にタイムアウト値を 2 倍にして送信する機能です。最大値は 64 秒と決められています。

3.1.3 パーシストタイマ(持続タイマ)

受信側が ACK で通知するウィンドウサイズがいったん 0 となった場合(つまり、データパケットの転送の中止を求められた場合)に、その後、受信側がある一定サイズのウィンドウサイズサイズを通知し、データ転送の再開を要求しても、その ACK のパケット自体が喪失してしまうと、双方が通信待ちの状態デッドロックになります。このデッドロックを防止するため、受信側がデータパケット転送の中止を求めているにも関わらず一定時間経過後に 1 バイトだけのデータパケットを転送する機能があります。パーシストタイマは、そのためのタイマの役割を果たします。パーシストタイマはタイムアウトした場合、指数バックオフ機能に基づき再送を繰り返します。

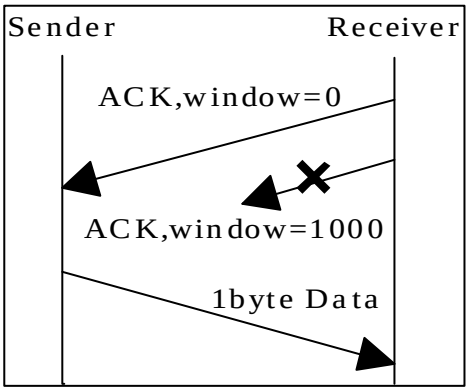

図 11：パーシストタイマ

3.1.4 キープアライブタイマ

キープアライブタイマとは、断続的な通信を想定し、通信相手からパケ

ットの転送がなくなった場合、通信相手がパケットを受信できる状態にあるかを確認する検査用パケットの転送間隔を決めるタイマです。キープアライブタイマは、2 時間毎に起動して通信相手の状態を調査します。FTP など不定期の通信を想定した設計になっています。システムダウンなどで通信相手が終了手順なしに通信を終了して通信相手からパケットの転送がなくなった場合などで有効です。

キープアライブ機能に関する議論では、デフォルトではキープアライブ機能をオフにして、ユーザーが望む場合にオンにすることが推奨されています。

キープアライブ機能が有効である代表的なケースは Web サーバです。多数の通信相手を受け付けている中で、たとえば、実際には通信相手はシステムダウンしていて TCP コネクションを使っていなくても、コネクションは確立されているかのように振る舞っているケースがあったとします。こうしたケースでは、使用されていないコネクションによって、カーネルのメモリが消費されてしまい Web サーバシステムのパフォーマンスが低下します。この場合だと、キープアライブ機能をオンにすることでパフォーマンス低下を回避できます。

逆に、パケット単位の課金システムの場合だと、キープアライブ機能がオンであれば、本来情報量に対して課金する料金体系として考案されているにも関わらず、キープアライブの度に転送されるパケットにも課金されてしまいます。

3.1.5 2MSL タイマ

2MSL タイマは、MSL(Max Segment Lifetime：最大セグメント生存時間)を表すために利用されます。MSL とは、パケットがネットワークに滞留できる最大時間のことです。MSL を越えてパケットが到着しない場合は何らかの理由でパケットが喪失したと見なすことができます。コネクションの終了時に、最初にクローズ要求を出す Active Close する側が、最後の FIN を受信して最後の ACK を送った後、2MSL(MSL の 2 倍)待機する、というケースで利用されます。

RFC793 の 2MSL の推奨値は 2 分とされていますが、多くの実装では、たとえば Solaris の場合だと 30 秒などです。

3.2 遅延確認応答アルゴリズム

遅延確認応答アルゴリズムは受信側の確認応答を遅延させる、つまり、受信側がパケットを受け取ってもすぐには ACK を返さないようにさせるアルゴリズムです。これによって、逆方向のデータ送信があれば PiggyBack するなど、ACK の数を減らし、ネットワークに不要なパケットが溢れ出す輻輳の可能性を減らすことができます。また、パケット受信割り込みの頻度を減らすことで、受信側の負荷を軽減させる狙いもあります。

遅延確認応答の条件は、①次のパケットの到着、②ファーストタイマの起動、です。

3.3 Nagle アルゴリズム

Nagle アルゴリズムは、小さなパケット扱う通信の場合、低速なネットワークではパケットをまとめて送信し、高速なネットワークでは素早く送信するなど、転送効率を高めるために考えられたアルゴリズムです。小さなパケットはできるだけまとめて転送するため、アプリケーションからの転送要求を遅延させます。telnet や rlogin などで有効です。遅延させる条件は、送出した小パケットの ACK を受信するまで次の小パケットを送信しない、というものです。

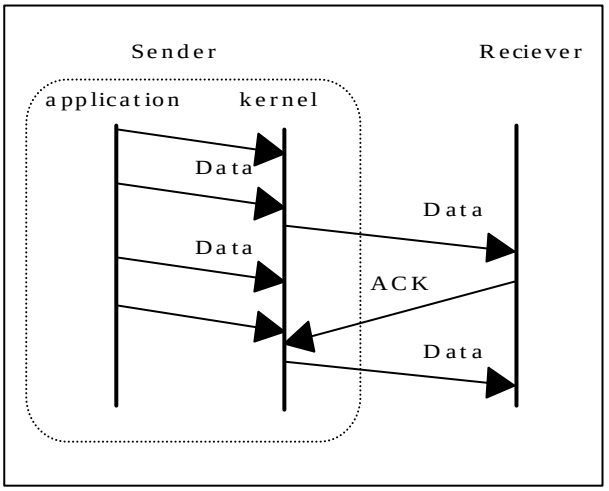

図 12：低速ネットワーク(パケットをまとめて送信)

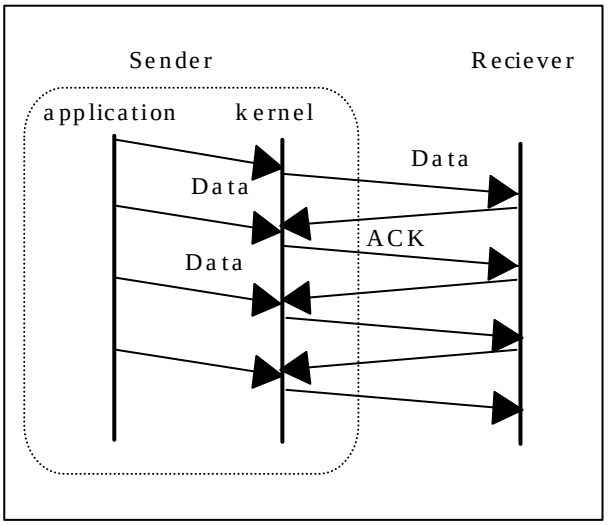

図 13：高速ネットワーク(パケットを素早く送信)

3.4 シリーウィンドウシンドローム

シリーウィンドウシンドロームは受信側が小さなウィンドウサイズを通達することにより、小さなパケットがやりとりされる現象です。この現象が生じると通信の効率が低下します。

回避方法は RFC813 で定めています。受信側については、ウィンドウサイズを通達する際の条件として、ウィンドウサイズが通信経路に最適な最大パケット長のフルサイズセグメント分空いているか、ウィンドウサイズの 2 分の 1 が空いている場合としています。また、送信側については、パケットを送出する際の条件として、①フルサイズセグメントを送出できる、②通達されたウィンドウサイズの 2 分の 1 を送出できる、③ ACK を待機しているパケットがない、としています。

3.5 TCP の通信フラグ

TCP の通信フラグは TCP ヘッダ中で指定します。6 つの指定が可能です。SYN フラグは、コネクション確立時に使用します。ACK フラグはパケットが ACK 情報を含んでいることを示します。FIN フラグはコネクション終了時に使用します。このほか、Push フラグ、RST(Reset)フラグ、緊急フラグがあります。

(1)Push フラグ

Push フラグは送信側が受信側にデータを遅延なく転送したい場合に利用します。

送信側はアプリケーションから渡されたデータのカーネルでのバッファリングを抑制し、受信側はアプリケーションへ渡すデータのカーネルでのバッファリングを抑制します。

現在ではこの機能にあまり意味がありません。現在の実装では、受信したパケットは直ちにプロセスに転送されるほか、Push フラグをアプリケーションから活用するための API も存在していないためです。

(2)RST(Reset)フラグ

RST フラグは、TCP コネクションをリセットするために使うパケットに利用します。Listen(コネクションの要求待ち)を行っていない port に対する接続要求を拒否したり、コネクションの中断やハーフオープンのコネクションを終了する場合に利用します。

RST に対する確認応答は生成されず、送信側は RST を送信すると同時に Close します。また、受信側は RST を受信すると同時に Close または Listen の状態になります。

(3)緊急フラグ

緊急フラグは、帯域外データを転送するために利用します。帯域外デー

タは、アプリケーションに緊急にデータを渡す場合に利用します。データ転送を中断したい場合などです。実際には通常のデータストリームと同様に配送されるので、正確には帯域外ではありません。アプリケーションにとって仮想的な意味で帯域外と表現されています。

(4)帯域外データの転送処理

送信側の処理は、アプリケーションから緊急にデータを送って欲しいという要求を受け取ると、送出データの中で緊急データの最後のバイトのシーケンス番号を緊急ポインタ(Urgent Pointer)にセットし、緊急フラグを立てます。この緊急フラグが立ったパケットを受け取った受信側は、アプリケーションプロセスに緊急データの到着を通知するという処理を行います。アプリケーションが緊急データまでのストリームを読み出すまで緊急モードと見なします。

4 TCP と輻輳制御

4.1 輻輳制御の重要性

(1)輻輳崩壊の危険性

インターネットのような自律的なネットワークは終端から送り込まれてくるデータの総量を全く予測することができないため、中継点でデータが溢れパケットの喪失が発生します。この状態を輻輳といいます。輻輳は悪化していく傾向があります。輻輳が起きた場合に適切なフィードバックを行わないと輻輳崩壊を招く危険性があります。たとえば、ネットワークのレスポンスが低下すると、利用者も TCP も再送要求をかけるので、ネットワークに不要なパケットが溢れ出し、それがさらにレスポンスの低下を招いてしまいます。それを回避するために、輻輳制御技術が重要とされています。

(2)輻輳制御の歴史

表 1：輻輳制御の歴史

初期インターネット	輻輳制御技術なし
1980 年初頭	輻輳崩壊の発生が指摘される
1980 年後半	エンドノード主体による輻輳制御アルゴリズムの出現
1990 年後半	中継ノード主体による輻輳制御アルゴリズムの出現
現在	エンドノードと中継ノードの協力による輻輳制御の試み

4.2 輻輳制御の難しさ

インターネットの輻輳制御は、①インターネットの状態把握が困難、②インターネットが自律的な性質を持っていること、③インターネットはモデル化が困難、の 3 つの理由で難しいと言えます。

(1)インターネットの状態把握が困難

IP は、次の 2 点で技術的に成功しています。

①様々な通信媒体の性質の抽象化によってアナログ電話回線、FDDI、ATM などの様々な通信媒体を透過的に扱えるようにした。

②中継システムに負担をかけない設計によりネットワークとネットワークの相互接続を簡素化した。

その反面、輻輳制御の観点からは、通信媒体の性質を上位プロトコルから見え難くしてしまった、中継システムの機能を制限してしまった、という欠点を持っています。このため、インターネットでは、帯域幅や利用率などの情報を取得することが困難になっており、ネットワークの許容量、混雑度がわからないため、輻輳制御が難しくなっています。

(2)インターネットは自律的

インターネット全体を制御する中央集権的な機構がないため、トラフィックに応じてユーザの振るまいを統括的に制御することができません。

(3)インターネットはモデル化が困難

インターネットはスケールが大き過ぎるためシミュレーションすることができないこと、通信媒体の構成要素、ネットワークの構成形態が多様であり典型的なネットワーク構成を特定できないこと、アプリケーションの登場などによってトラフィックパターンが全く変化してしまうなどの理由でモデル化が困難です。昨日のインターネットは今日のインターネットではないとも言えます。トラフィックパターンが全く変化した代表的な例は、Web ブラウザのモザイクの登場です。1992 年のモザイクの登場がそれまではほとんど利用されてこなかった HTTP プロトコルをインターネットの代表的なプロトコルに変えました。現在では、HTTP はインターネットのトラフィックパターンのほとんどを占めています。このため、現在ではインターネットの輻輳制御は TCP で行っていくことがコンセンサスになっています。

4.3 TCP による輻輳制御

TCP による輻輳制御は終端ノードによる自律的な制御と言えます。TCP は、通信経路を推測しながら通信経路に適した転送レートを選択する簡易なアルゴリズムを持っています。これによって、ある程度の輻輳の発生を防ぎ、また輻輳を検出した場合、転送レートを下げることによって輻輳崩壊を防ぐようにしています。

4.3.1 TCP 輻輳制御アルゴリズムの歴史

表 2：TCP 輻輳制御アルゴリズムの歴史

88 年頃	Tahoe	スロースタートアルゴリズム、輻輳回避アルゴリズム、Fast Retransmit アルゴリズムの採用
90 年頃	Reno	Fast Recovery アルゴリズムの採用(輻輳の度合が少ない場合、転送速度を大きく落さないのが狙い)
96 年頃	NewReno	Fast Recovery アルゴリズムの修正(パケットの喪失率がやや大きい場合に対するアルゴリズムの不具合の修正)
99 年	RFC2581	Fast Recovery アルゴリズムを採用

4.3.2 TCP の輻輳制御アルゴリズム

TCP の輻輳制御アルゴリズムはネットワークの状態はよくわからないという前提で単純なアルゴリズムによる転送制御を行っています。それは、パケットが喪失しないときは、ネットワークは空いているから転送速度を上げ、また、パケットが喪失するときは、ネットワークは混んでいるから輻輳崩壊を避けるため転送速度を下げます。TCP では転送速度の上げ下げを繰り返しながら通信を行っていきます。パケット喪失が起きるまで転送速度を上げ続けますが、それによって引き起こされるパケット喪失によって通信経路の限界を判断するわけです。

転送速度の制御はパラメータである輻輳ウィンドウ cwnd のサイズを変えることで行います。輻輳ウィンドウサイズと受信側が通知したウィンドウサイズのうちウィンドウサイズの小さい値を採用して、受信側の許容量を超えず、ネットワークの状態を考慮しながら通信を行います。輻輳ウィンドウサイズの増やし方はスロースタート段階と輻輳回避段階の 2 つの通信段階に対応しており、各通信段階によって増加量は異なります。

①スロースタート段階
ACK を受け取る毎に 1 ずつウィンドウサイズを増加していくアルゴリズムに基づき、指数的にウィンドウサイズを増加させます。

②輻輳回避段階
ウィンドウサイズ分の 1 ずつ増加していくアルゴリズムに基づき、線形的にウィンドウサイズを増加させます。

この 2 つのアルゴリズムを組み合わせ、通信状態に合わせた転送レートの上げ下げを行っていくのが TCP の輻輳制御アルゴリズムです。

転送速度の制御では、まず、データパケットの喪失が起きれば、転送速

度を下げます。

その際に、重複確認応答によってデータパケット喪失を検出した場合は、輻輳ウィンドウサイズの 2 分の 1 を ssthresh(スロースタートの threshold)として登録して輻輳ウィンドウサイズを減らし、転送速度を下げます。

また、再送タイムアウトによってデータパケット喪失を検出した場合は、輻輳ウィンドウサイズを最小にします。

データパケットの喪失が起きない場合は、輻輳ウィンドウサイズがスロースタートの threshold より小さい値のときはスロースタート段階のアルゴリズムによって指数関数的にウィンドウサイズを大きくしていきます。輻輳ウィンドウサイズがスロースタートの threshold より大きくなったら、輻輳回避段階のアルゴリズムによって線形的にウィンドウサイズを大きくしていきます。

4.3.3 Fast Retransmit アルゴリズム

Fast Retransmit アルゴリズムでは、再送タイマのタイムアウトを待たずに迅速に再送パケットを転送するアルゴリズムです。

受信側からの再送を要求する確認応答パケットが重複して 3 つ到着した場合、データパケット喪失の可能性が高いと判断して、再送タイマのタイムアウトまで待機せずに当該パケットを迅速に転送します。

これは、パケットは通常、高い確率で順序通りに転送されるため、再送を要求する確認応答パケットが 3 つも到着すれば、当該パケットは喪失したと判断するものです。再送タイマのタイムアウトを待機する時間を節約し、迅速に再送することで転送効率を向上を目指します。

Fast Retransmit アルゴリズムでは、パケット再送後は、輻輳崩壊を避けるため転送速度を減少させます。

このため、Fast Retransmit アルゴリズムを採用した Tahoe のアルゴリズムの場合は、転送速度を減少させるために輻輳ウィンドウサイズを 1 に設定して、スロースタート段階のアルゴリズムによって 1 から指数関数的に転送速度を上げていくことになります。

これに対し、Reno のアルゴリズムの場合は、Fast Recovery アルゴリズムを採用したことで、輻輳ウィンドウサイズをパケット喪失検出前の 2 分の 1 に設定して、輻輳回避段階のアルゴリズムによってパケット喪失検出前の 2 分の 1 の輻輳ウィンドウサイズから線形的に転送速度を上げていきます。

4.3.4 Fast Recovery アルゴリズム

Fast Recovery アルゴリズムは 1990 年に Reno に追加されました。

Tahoe の輻輳回避アルゴリズムの問題は、Fast Retransmit アルゴリズ

ムの終了後に輻輳ウィンドウサイズを 1 にして転送速度を落しすぎることにありました。Fast Recovery アルゴリズムの目的はその転送効率を改善することにあります。その仕組みは、Fast Retransmit アルゴリズムが成功すれば輻輳は軽微だったと見なし、転送速度をパケット喪失検出前の 50%に落すというものです。

Fast Recovery アルゴリズムでは、パケット再送後は、さらに重複した再送要求の確認応答パケットを確認すると、輻輳ウィンドウサイズ cwnd を一時的に増加させることによって、新しいセグメントを送出します。

新しい確認応答 ACK が到着すると、輻輳ウィンドウサイズ cwnd をパケット喪失前の 2 分の 1 の値でスロースタートの threshold(ssthresh)にセットします。スロースタート段階には入らず、いきなり輻輳回避段階へ入るわけです。

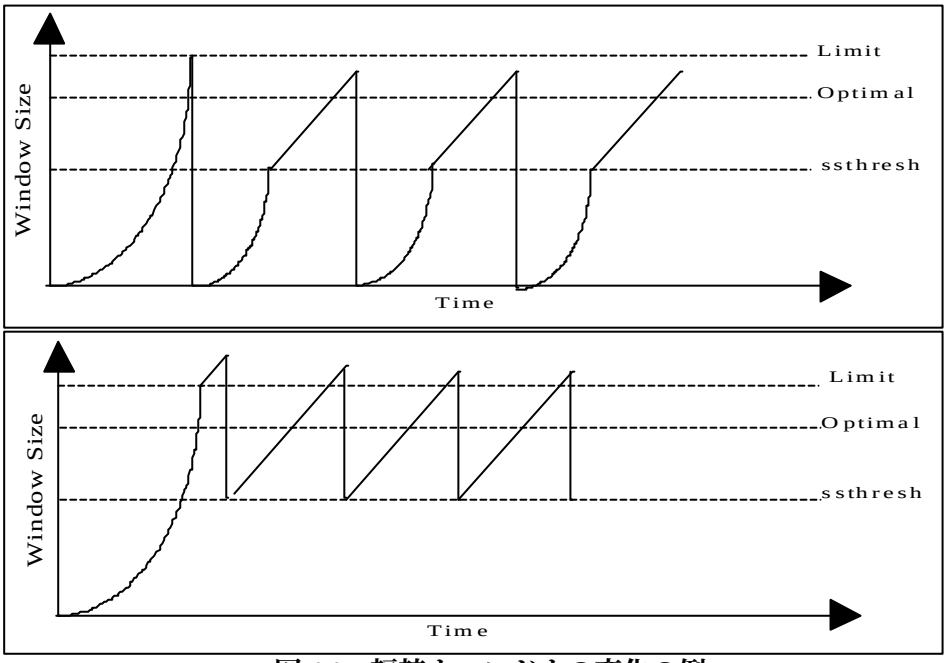

図 14：輻輳ウィンドウの変化の例

Tahoe(上)／Reno，NewReno(下)

4.3.5　TCP の輻輳制御アルゴリズムの狙い

(1)ネットワークの状態の変化に対応

TCP の輻輳制御アルゴリズムの狙いはネットワークの状態の変化に対応することにあります。つまり、輻輳が起これば転送レートを下げ、輻輳崩壊を回避します。

その際に輻輳が起こらない限り転送レートを増加させ続けます。これは、ネットワークの混雑度が常に一定でなく通信の途中で混雑が緩和した時も考慮し、転送効率を高めるという考え方に基づいています。転送速度を増加させ続けると輻輳が発生しますが、その場合はまた転送レートを下げ輻輳を回避するというフラクチエーションを繰り返します。

(2)輻輳が生じない限界近くの転送速度レートで長く通信する

TCP では、パケットが喪失しないときは、ネットワークは空いているから転送レートを上げ、また、パケットが喪失するときは、ネットワークは混んでいるから輻輳崩壊を避けるため転送レートを下げます。この転送レートの上げ方は、スロースタート段階ではウィンドウサイズを指数関数的に一気に最適値に近づけ、輻輳が生じそうな転送速度レートになると輻輳回避段階に入り線形的にゆっくりとウィンドウサイズを増加させ、転送速度レートを上げていく、というものです。このスロースタート段階と輻輳回避段階の 2 つのモードを繰り返すことによって、輻輳が生じない限界近くの転送速度レートでできるだけ長く通信ができるようにします。

(3)セルフクロッキング

TCP ではセルフクロッキングを実現できます。セルフクロッキングとは通信経路に適したウィンドウサイズで通信を続けると転送レートが自動的に制御される状態のことです。

TCP はウィンドウサイズで転送レートを制限しています。ウィンドウサイズは、1 度に送ることができるパケット数を決定します。しかし、セルフクロッキングによりウィンドウサイズが 100 の場合、100 個のパケットを一気には転送せずに少しずつ間隔を空けてバーストを起こさないように転送することができます。連続して送信されたパケットの間隔は、通信経路を転送している間に、通信経路中で最もデータ転送容量の小さい通信経路をパケットが通り抜けるときに生じた間隔にだんだん近づくようになります。セルフクロッキングは、この現象を利用します。

送信側は、確認応答パケットの到着を引き金に新しいデータパケットを送信するという単純なアルゴリズムを使うだけで、最小容量の通信経路の通り抜けで生じた間隔でパケットを送信することになります。この間隔は、通信経路中の最小容量の部分をパケットが円滑に通り抜けられる間隔なので、その通信経路に最適なパケット転送の間隔を自動的に実現したことになります。複雑な転送レート制御機構は必要ないわけです。

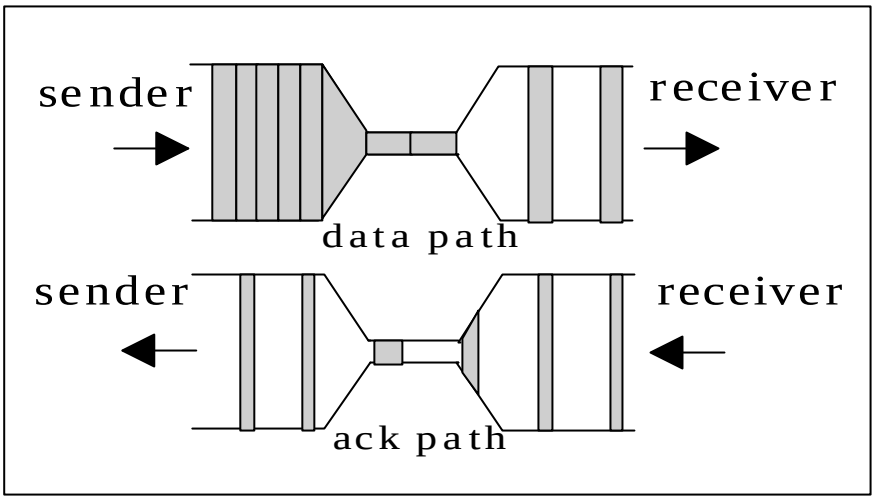

図 15：セルフクロッキング

5　採用されつつある新しい機能

ここでは、Windows98、新しい Linux、Solaris の新バージョンなどで採用されつつある機能について説明します。

5.1　広帯域、高遅延ネットワークへの対応

(1)広帯域、高遅延ネットワークにおける TCP の問題

(a)ウィンドウサイズの不足

TCP の転送性能はウィンドウサイズとデータが送信してから送達確認を受信するまでの時間 RTT(Round Trip Time)で決まります(TCP の転送性能=ウィンドウサイズ／RTT)。

ただ、帯域 2Mbps、RTT0.5 秒の衛星通信回線などの広帯域、高遅延ネットワークの場合、その帯域全部を使い切るウィンドウサイズは 512000 バイトであるにも関わらず、RFC793 で決められている TCP の最大ウィンドウサイズは 65535 バイトのため、最大でもその帯域の 12%しか利用できないという問題があります。

(b)RTT の計測が不正確になる

これまでの実装では RTT の計測はウィンドウサイズに対して 1 回だけだったので、ウィンドウサイズが増加すれば RTT の計測頻度が減少するという問題があります。ウィンドウサイズが 1000 の場合、RTT の計

測は 1000 回に 1 回しか行われないため、正確性は非常に低くなります。

(c)シーケンス番号の周回

シーケンス番号は 2 の 32 乗までつけられますが、2 の 32 乗までのパケットを一度に送ることができるネットワークがあった場合、ネットワーク中にパケットが滞留している間に、同じシーケンス番号を持つパケットが送信されてしまいます。受信側が同じシーケンス番号を持つパケットを受け取っていまい、パケットの識別ができなくなるという問題が発生します。

これらの問題を解決するため、ウィンドウスケールオプションとタイムスタンプオプションを利用します。

(2)ウィンドウスケールオプション

ウィンドウスケールオプションは TCP がネゴシエートするときに TCP のオプションを使って大きなウィンドウサイズを指定できるようにしたものです。ウィンドウサイズの値をビットシフトするスケール値を TCP のヘッダ中に指定します。たとえば、スケール値が x の場合はウィンドウサイズの値を x ビットシフト(2 の x 乗)した値をウィンドウサイズとして使うように指定したことになります。スケールとして指定できる最大のシフト値は 14 までと決まっているので、ウィンドウスケールオプションを使った場合の最大のウィンドウサイズは、

65535 バイト×2^{14}＝1073725440 バイト

となります(65535 バイトは RFC793 の最大ウィンドウサイズ)。

広帯域、高遅延ネットワークで不足するウィンドウサイズを補うことができるわけです。

ウィンドウスケールオプションは 3way handshake 中に指定します。3way handshake での指定は次のような手順を踏みます。まず、クライアント側は、クライアント側で使うウィンドウスケールオプションのスケール値を確立要求(SYN)と一緒に通知します。これに対し、サーバ側は、サーバ側で使うウィンドウスケールオプションのスケール値を、サーバ側の確立要求(SYN)とクライアント側の確立要求(SYN)に対する確認応答(ACK)を Piggyback するときに通知します。それに対するクライアント側の確認応答(ACK)によってネゴシエーションは成立します。コネクションの途中でスケール値は変更できません。

(3)タイムスタンプオプション

タイムスタンプオプションは、パケットの中に現在の時刻をタイムスタンプとして記録することで、これまで 1 ウィンドウに 1 個しかプローブできなかった RTT(Round Trip Time)の計測を、すべてのパケットでできるようにしたものです。

まず、送信側はデータパケットにタイムスタンプ値を付けて送信します。受信側は確認応答パケットにこのデータパケットに入っているタイムスタンプ値を付けて送り返します。送信側で現在の時刻と確認応答パケッ

トの中のタイムスタンプ値の差を計算することで、すべてのパケットで RTT が計測できるわけです。

タイムスタンプオプションを用いることで、RTT の計測頻度と RTT の精度が向上します。また、タイムスタンプ値とシーケンス番号のペアで周回を検出し、シーケンス番号周回によってパケット識別ができなくなるという問題の発生を防止します。

5.2 Path MTU discovery

Path MTU discovery は、すべてのデータパケットに DF(Don't Fragment)ビットをセットして通信するという方式です。この方式を用いることで、現在の通信経路に最適で可能な限り最大なパケット長を把握することができ、通信効率を向上させます。たとえば、MSS(Max Segment Size：最適パケット長)2000 で DF ビットをセットしてパケットを送信した結果、中継ノードから DF 通知が返ってきたら、MSS2000 はその通信経路で利用できないことが分かります。逆に受信側の確認応答 ACK が返ってきたら MSS2000 で通信できたということになります。RFC1191 では、通信経路のセグメントのパケット長のプローブは 10 分と推奨されています。

5.3 SACK

これまでの TCP は累積確認応答方式によってパケットの送達を確認していました。その場合は、具体的にどのパケットが送達しているのか、までは確認できないという限界がありました。SACK(Selective Acknowledgement：選択確認応答)オプションを用いることで、具体的にどのパケットが送達しているかを詳細に通知することが可能になります。SACK は、RFC2018 で規定されていて、TCP のヘッダにオプションのフィールドをさらに追加することによって利用できるようになります。

SACK には SACK Permitted Option と SACK Option の 2 つのオプションがあります。

SACK Permitted Option は、3way handshake 時の negotiate で通信相手に SACK option に対応していることを通知する機能です。確立要求の SYN フラグがセットされたパケット以外では利用できません。

また、SACK Option は、TCP のヘッダの中に、到着したパケットのシーケンス番号を格納します。

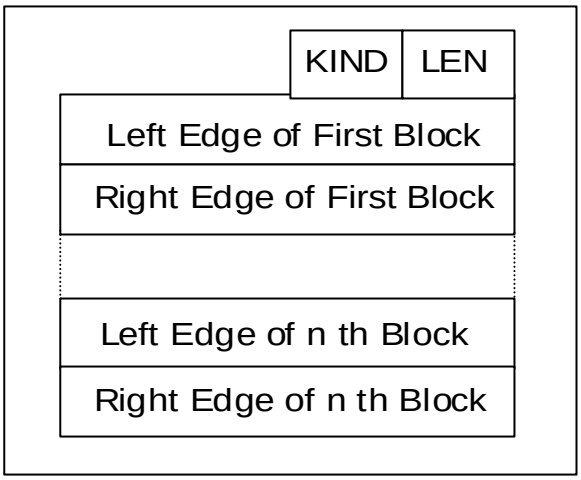

図 16：SACK Option のフォーマット

SACK Option のフォーマットは、最大 4 ブロックに到着したパケットのシーケンス番号を格納します。新しく受信したパケットのシーケンス番号の左端と右端を最初のブロックに、その前に受信したパケットのシーケンス番号の左端と右端を次のブロックに、と 4 ブロック分まで指定できます。

表 3 はシーケンス番号 5000 から 8500 のパケットを MSS500 バイトで送信したとき、シーケンス番号 5500、同 6500、同 7500 のパケットが喪失したケースで、受信側がどのような SACK Option のフォーマットを作るかを示したものです。

表 3：SACK Option の例

Trigger Segment	ACK	1st block		2nd block		3rd block	
		Left	Right	Left	Right	Left	Right
5000	5500						
5500(lost)							
6000	5500	6000	6500				
6500(lost)							
7000	5500	7000	7500	6000	6500		
7500(lost)							
8000	5500	8000	8500	7000	7500	6000	6500
8500(lost)							

表 3 中の Trigger Segment 項目はシーケンス番号 5000 から 8500 のパケットを MSS500 バイトで送信したときの各セグメントを、ACK 項目は受信側が累積確認応答方式によって生成する ACK を示しました。こ

の SACK Option のフォーマットから、送信側は累積確認応答によってシーケンス番号 5500 以降のパケットの再送要求のほかに SACK Option によって具体的にシーケンス番号 7000、同 6000、同 5000 のパケットが送達していることを確認できます。

5.4 現在の実装状況

現在の実装状況はビッツバーグスーパーコンピューティングセンタ(Pittuburgh Supercomuting center)で調査しています。それによりますと、Path MTU discovery、ウィンドウスケールオプションとタイムスタンプオプション、SACK のすべてに対応している OS は、Windows98、Solaris2．6、Solaris7 です。

また、Path MTU discovery、ウィンドウスケールオプションとタイムスタンプオプションの 2 つに対応している OS は、FreeBSD3.3 です。

さらに、Path MTU discovery だけに対応している OS は、Windows95、WindowsNT3.5、WindowsNT4.0 です。

一方、どれにも対応していない OS は SunOS4.1 となっています。

http://www.psc.edu/networking/perf_tune.html に詳しい情報があります。

6 TCP とセキュリティ

過去の TCP に対する典型的な攻撃は Sequence number attack と SYN flood Attack の 2 つです。適切な処置は、標準化されつつある IPsec を採用して通信するとかなりの効果があります。また、適切な Filtering を施すことでも効果を上げられますが、Web サーバのような不特定多数のコネクションを受け入れなければならないケースでは難しいとされています。

6.1 Sequence number attack

Sequence number attack は TCP コネクションに使われるシーケンス番号を生成するアルゴリズムを推測し、偽造パケットを送り込み偽の TCP コネクションを確立する攻撃方法です。IP スプーフィングという技術を使ってアドレスを偽り、確立要求 SYN フラグをセットした偽造パケットを送り込みます。アドレスを偽っているため SYN の確認応答「SYN,ACK」を受け取ることはできませんが、このコネクションに使われるシーケンス番号生成アルゴリズムを推測しているため、あたかも通信しているかのように装い「SYN,ACK」に対する ACK を送り込みます。それによってコネクションが確立されてしまいますので、その通信によってコマンドなどを送り込まれる危険性があります。

この攻撃を防ぐためにはシーケンス番号生成アルゴリズムを複雑にし推測されないように修正する必要があります。古い実装では、1 秒毎のカウンタの増加、コネクション毎の固定値の加算など変数からシーケンス番号を生成していました。このような単純なアルゴリズムは、連続してコネクションの確立を試みられた場合、容易にアルゴリズムを推測されてしまいます。

このため、4 マイクロ秒毎のカウンタの増加や、src adr、src port、dst adr、dst port などの情報をハッシュした変数からシーケンス番号を生成するなどの推測され難いアルゴリズムの実装が推奨されます。

6.2 SYN flood attack

SYN flood attack はサービス妨害(DoS)の一種です。攻撃目標に偽の確立要求 SYN を送ると、攻撃目標とされた側は half open して SYN の確認応答「SYN,ACK」を返信します。この「SYN,ACK」に対する ACK を送信しないと攻撃目標とされた側は half open のまま ACK の到着を待ちます。偽の確立要求 SYN を大量に送りつけて、half open のコネクションを大量に作らせて、新しいコネクションを受け付けられない状態にするというものです。それによって、コネクション要求キューを溢れさせたり、half open のコネクションを大量に作らせることでメモリの消費させたりして、サービスを妨害します。

これに対する終端システムでの対策は、通常 90 秒や 60 秒と設定されている 3way handshake 時のタイムアウトをたとえば 6 秒などと短くします。これによって、half open のコネクションをタイムアウトによって閉じていきます。

また、コネクション要求を受け付ける queue を 100 や 1000 など大きくすることでコネクション要求キューが溢れ出すことを防ぎます。

実装上では、half open のコネクションに割り当てるメモリを減らし、half open のコネクションが増加してもメモリが消費されない機能にすることで、SYN flood attack から防衛するというテクニックがあります。このような対策に加え、中継システムでの対策も可能です。

中継システムで、信頼できるアドレス以外からのアクセスを拒絶するようにフィルタをかけるというものです。ただ、IP スプーフィングという技術を使って信頼できるアドレスからのコネクションのように見せかける攻撃もあるで完璧ではありません。

さらに、SYN がセットされたパケットの転送レートを制限するという方法があります。これによって、SYN がセットされた大量のパケットがバースト的に送られてきても、転送レートの制御で一度に受けつけることができる許容量を減らし、新たなコネクション要求キューが溢れ出すことを防止する狙いがあります。

7　これからの技術動向

ここでは、実装はまだ実験的にしか行われていませんが、TCP の輻輳制御の技術動向として有効で今後着目されそうな新技術を説明します。

①Explicit Congestion Notification
②Initial Large Window
③TCPVegas
④NewReno
⑤Rate-Halving
⑥TCPfriendly

の 6 つです。

7.1　Explicit Congestion Notification

Explicit Congestion Notification(ECN)は中継ノードによる輻輳制御方式です。RFC2481 で規定されています。TCP は終端ノードで輻輳制御を行いますが、中継ノードによる輻輳制御を組み合わせることで新たな輻輳制御が可能になると言われています。

Explicit Congestion Notification は、具体的には次のような働きをします。

まず、送信側と受信側で通信している際、中継ルータにキューが溢れてきた場合、中継ルータはネットワークが輻輳してきたと判断して送信側が送ってきたパケットに輻輳の発生を知らせる Congestion Experience(CE)ビットをセットし受信側へ送ります。

受信側はその CE ビットによってそのパケットが輻輳が発生しているネットワークを通過したと判断します。そして、送信側に転送レートを下げるようにフィードバックをかけるため、ECN echo ビットをセットした確認応答パケットを送信側に送ります。

同確認応答パケットを受け取った送信側は ECN echo ビットによって輻輳発生を知ったので輻輳ウィンドウサイズを減少させるとともに、確認応答パケットに Congestion Window Reduce(CWR)ビットをセットし受信側に送信します。Congestion Window Reduce ビットは ECN echo に従い輻輳ウィンドウサイズを下げたので ECN echo の送信停止を求めるフラグです。

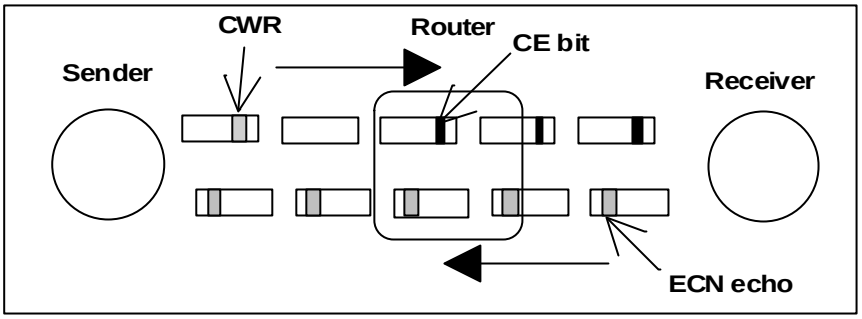

図 17：ECN の仕組み

Explicit Congestion Notification が通信で利用できるかのネゴシエーションは、3way handshake 中に指定します。クライアント側が確立要求(SYN)と一緒に ECN echo ビットと Congestion Window Reduce ビットをセットしサーバ側へ通知します。これに対しサーバ側は確立要求(SYN)とクライアント側の確立要求(SYN)に対する確認応答(ACK)を Piggyback すると同時に ECN echo ビットだけをセットし通知します。それに対するクライアント側の確認応答(ACK)によってネゴシエーションは成立します。

7.2 Large Initial Window

Large Initial Window は RFC2414 で規定されています。

Large Initial Window は、輻輳ウィンドウサイズの最小値を大きくすることで転送効率を高めるものです。従来の輻輳ウィンドウサイズの最小値は 1MSS(Max Segment Size：最適パケット長)ですが、これを 4MSS にしようというものです。輻輳ウィンドウサイズの最小値が 1MSS だと最初の転送の 1 パケットに対し必ず遅延確認応答アルゴリズムにより確認応答が遅延されて送られるため転送効率が悪いと言えます。

これに対し、Large Initial Window を採用しで輻輳ウィンドウサイズの最小値を 4MSS とすれば、最初の転送に遅延確認応答が適用されないという利点があります。また、最初の転送はスロースタート段階ですので輻輳ウィンドウサイズは指数関数的に増加します。このため、データが送信してから送達確認を受信するまでの時間 RTT(Round Trip Time)が長い広帯域・高遅延ネットワークでは、輻輳ウィンドウサイズの最小値を大きくしておくと、輻輳ウィンドウサイズの指数関数的な増加が早くなると利点もあります。さらに、細かいデータ転送が多い HTTP の場合なら、1RTT で小さなファイルを一気に送信できオーバーヘッドが少なくなります。

しかし、その一方で、転送のバースト性が全体的に増加する可能性も指摘されています。転送のバースト性が全体的に増加することが広域のイ

ンターネットでどのように挙動するか予測できないとの意見もあり、様々な議論が交わされています。現時点ではインターネット標準技術として採用されるかは分かりません。

7.3 TCPVegas

TCPVegas アルゴリズムは、アリゾナ大の Brakmo 氏らによって開発されたアルゴリズムです。

従来の TCP の輻輳制御アルゴリズムは、ネットワークの状態はよくわからないということを前提とした単純なアルゴリズムです。パケットが喪失しないときは、ネットワークは空いているから転送速度を上げ続け、パケットが喪失したら、ネットワークの輻輳崩壊を避けるため転送速度を下げるなど、パケットが喪失しない状態とパケットが喪失する状態を繰り返しながら転送速度を増加させていました。

これに対し、TCPVegas は、より詳細な通信経路の状態の推測するため、混み具合に応じた転送速度の調節を試みるアルゴリズムです。

TCPVegas では、データ転送をしながら実際の転送性能である Actual Throughput とこれまでの転送の結果から推測した転送性能である Expected Throughput を 2 つのスループットを計測します。そして、Actual Throughput と Expected Throughput を比較しながら、輻輳制御を行っていきます。Actual Throughput が Expected Throughput の値を下回り期待したほど転送効率が良くなかった場合、TCPVegas はネットワークが混んできたと判断して転送速度を落します。また、Actual Throughput が Expected Throughput の値を上回り期待以上に転送効率が良かった場合は、TCPVegas は、ネットワークが空いてきたと判断して転送速度を上げます。Actual Throughput と Expected Throughput の値にほとんど差がない場合はウィンドウサイズは最適な値で通信状態が非常に安定していると判断して、転送速度を維持し、安定的にウィンドウサイズを保持していきます。

ただ、実際にはインターネットのトラフィックはこのように理想的な動きはせずにスループットも激しく変化するパターンの方が一般的なので、TCPVegas アルゴリズムはあまり適応できないのではという意見が多数を占めています。TCPVegas アルゴリズムをさらに研究し効率の良い推測アルゴリズムを開発できればインターネット標準になる可能性もありますが、TCPVegas を積極的に推進していく動きは現在のところ盛んではありません。

7.4 NewReno

NewReno は、マサチューセッツ工科大学の Hoe 氏らのアイデアがベースとなった TCP の新しい輻輳制御アルゴリズムです。Fast Retransmit アルゴリズム、Fast Recovery アルゴリズムを変更し、1RTT 内に複数のパケット喪失がある場合の転送性能の改善を狙いました。RFC2582 で規定されています。

Reno のアルゴリズムは、Fast Retransmit アルゴリズムが誤作動し、

再送タイマのタイムアウト待ちによって転送性能が劣化する場合があります。

これに対し、NewReno では、また、Fast Retransmit アルゴリズムの誤作動を防止し、再送タイマのタイムアウトを待たずに複数パケットを再送するように改善しました。

ただ、逆に Reno の Fast Retransmit アルゴリズムの誤作動によって結果的に輻輳崩壊が避けられていたという考え方もあり、NewReno で Fast Retransmit アルゴリズムの誤作動問題を解決してしまうとインターネットの輻輳崩壊が発生しやすくなるという意見があります。

このため、NewReno がインターネットの標準技術として採用されるかは微妙な段階です。

7.5 Rate Halving

Rate Halving は、NewReno を考案したマサチューセッツ工科大学の Hoe 氏のアイデアを元にビッツバーグスーパーコンピューティングセンターの Mathis 氏らが提案しました。

Rate Halving では、パケット喪失検出後の処理として従来の Fast recovery アルゴリズムが輻輳ウィンドウサイズを 2 分の 1 に減らすのに対し、確認応答 ACK を送信するタイミングを 2 分の 1 にした方がスムーズに輻輳から回復できるのではないかという考え方に基づきます。

Rate Halving では、輻輳から回復していくとき、ACK がセットされた 2 つのパケットを受け取るごとに 1 つのデータパケットを転送することで、データパケットの転送間隔を通常の 2 倍にします。輻輳ウィンドウサイズを 2 分の 1 に一気に減らす従来の方法よりも TCP のセルフクロッキングを崩さずに転送レートをなめらかに下げる方が、結果的に輻輳制御に有効だという研究です。

Rate Halving はまだ研究段階であるため、実装は広がっていません。

7.6 TCPfriendly

TCPfriendly は ACIRI の S．Floyd 氏らによって提案されました。現在、活発な研究が行われています。

現在のネットワークでは、TCP には輻輳制御があるが UDP にはないため、回線が混むと UDP が帯域を占領してしまうという問題があります。このため、UDP のための輻輳制御の指標を作った方がよいとして TCPfriendly という考え方が生まれました。

TCPfriendly では、UDP を TCP の通信モデルに合わせるというものです。TCP の通信モデルは、バンド幅と TCP の転送速度とパケットロス率の関係を、バンド幅に対する TCP の転送速度とパケットロス率の関係式として、以下のような計算式に表すことができます。

$$bandwidth = \frac{1.3MTU}{RTT\sqrt{Loss}}$$

この計算式に従っている UDP flow は TCPfriendly と言えます。

TCPfriendly は、アプリケーションのプログラマーがプログラムを設計するときに、転送レートの調節は適切に行って下さいという設計への指標になるほか、ルータなどで UDP flow がこの計算式に従っているかを判別し、輻輳崩壊の原因となる UDP flow を強制的に排除するというようなアルゴリズムに適用することも可能です。

7.7 TCP の今後

標準化や実装が普及すると見られれているのは、SACK、ECN です。NewReno はいくつかの実装に取り入れられて、オン、オフ機能が加わられた形で普及していく可能性があります。

また、輻輳制御アルゴリズムの改善では、Rate halving や TCPVegas がありますが、TCPVegas にはもっと研究が必要で、むしろ、研究段階の Rate halving の方が優勢と見られています。

新たな技術との組み合わせとしては、CBQ(クラスベースキューイング)や Diffserv などのアイデアが生まれていることから、中継ノードでキューイングする機構との組み合わせて、データパケットによっては素早く転送したり、ゆっくり転送したりと差別化できるようになる可能性もあり、そのような研究が進んでいくと予想されています。

さらに、輻輳制御の方向性では、TCP だけが輻輳制御を行うのではなく、エンドシステム全体の輻輳制御技術へと発展していく可能性もあります。TCP friendly の活発な研究では、中継ルータとの連携による UDP の輻輳制御を目指していますし、TCP の輻輳制御機構を分離して UDP もその輻輳制御機構を使えるようにする試みとして Congestion Manager もあります。それらが TCP/IP のプロトコルスタックの標準化を決める IETF（Internet Engineering Task Force：インターネット技術検討部会）で議論されています。

7.8 さらに詳しく知りたい方のために

TCP に関連する主な RFC(Request For Comments)は以下のとおりです。

- ●RFC793　基本仕様
- ●RFC813　Silly Window Syndrome
- ●RFC1122　Host Requirement(アルゴリズムの指標を示す)

●RFC1323　Extention for high performance

●RFC2414　Large Initial Window

●RFC2418　ECN

●RFC2581　Congestion Control(Reno の Congestion Control アルゴリズムを記述)

●RFC2582　NewReno algorithm

IETF の TCP に関するワーキンググループは、以下の 4 つのグループがあります。

●TCP Implementation (tcpimpl)

●TCP Over Satellite (tcpsat)

●Performance Implications of Link Characteristics (pilc)

●Endpoint Congestion Manager

TCP Implementation では、TCP 実装でのバグの調査や実装のための指標づくりを行ってきました。バグ調査などの一連の作業が終了したため閉会するという動きも出ています。

TCP Over Satellite は衛星通信などの広帯域・高遅延ネットワークでの TCP の通信性能の向上を目指して研究活動を行ってきました。衛星通信の問題点が調査できたので、ワイヤレスなどを扱う Performance Implications of Link Characteristics に活動を移管する動きがあります。

Performance Implications of Link Characteristics では、ワイヤレスなどリンクする通信媒体の性質に合わせた TCP の通信プロトコルの考え方を議論しています。

Endpoint Congestion Manager は新しく設置されたワーキンググループです。TCP の輻輳制御機構を分離して TCP や UDP、あるいはこれらのトランスポートプロトコルに替わるかもしれない将来のトランスポートプロトコルでもその輻輳制御機構を使えるようにする仕組みづくりを試みています。