マルチメディア通信と分散処理 82－24
（1997. 4. 25）

TCPデッドロック問題の解決策の提案

村山 公保　　山口 英
yukio-m@is.aist-nara.ac.jp　*suguru@is.aist-nara.ac.jp*

奈良先端科学技術大学院大学情報科学研究科

インターネットで最も重要なトランスポート・プロトコルであるTCPでは確認応答処理に不備がありデッドロックが発生することがある．デッドロック状態が発生すると，利用可能な帯域に関係なく間欠的にしかセグメントが転送されずスループットが極端に悪化する．
TCPのデッドロックの解決法は，これまでにいくつか提案されているが，従来の方法ではTCPのデータ転送モデルを無視しているため，デッドロックが解決する代わりにトラヒックが増加するという問題がある．
本研究では，TCPのデータ転送モデルを重視し，送信ホストが受信ホストの遅延確認応答処理を完全に制御することで，トラヒックの増加なしにデッドロックを解決できる方法を提案する．

A proposal for the solution of the TCP deadlock problem

Yukio Murayama　　Suguru Yamaguchi
yukio-m@is.aist-nara.ac.jp　*suguru@is.aist-nara.ac.jp*

Graduate School of Information Science, Nara Institute of Science and Technology.

Since TCP is the most important transport protocol in the Internet, it should be strongly encouraged to improve drawbacks of the TCP. One of major drawbacks is deadlock. In some circumstances, TCP may occur deadlock. Regardless the available bandwidth for the TCP connection in deadlock, the data over the connection are transfered intermittently.
There has been some methods proposed for solving this TCP deadlock problem. Some of them can be a solution for this problem. However, they add more traffic extremely, thus, networks cannot be used effectively in terms of the end-to-end throughput.
We propose yet another solution for this problem. TCP deadlocks can be eliminated with our method and this solution adds little traffic more, because the TCP sender can controls the receiver's processing of the delayed ACK perfectly.

1 はじめに

インターネットの発達とともに，インターネットのプロトコルに潜在していた問題が浮き彫りになってきた．インターネットの急速な拡大に伴い，IPアドレスが枯渇して初めてIPv4はスケーラビリティに問題があるという重大な欠点が見つかった．この問題に対処するため，IPの次のバージョンであるIPv6に関する研究・標準化作業が進められた [1]．IPv6は当面の問題であるIPアドレスの枯渇問題に対する解決だけではなく，今までインターネットを運用してきた経験と反省を踏まえて，不満を一掃できるような仕様を標準化した．具体的にはルータの処理を簡略化する工夫やプラグ&プレイ機能，セキュリティ，品質制御 (QoS) などの機能をオプションではなく，標準で対応するようになっている．これまでインターネットでは，プロトコルのオプションの機能は必ずしも実装されない事があり，それがインターネットを運用していく上でしばしば問題を引きおこしたからである [2][3]．

インターネットで利用されているトランスポート・

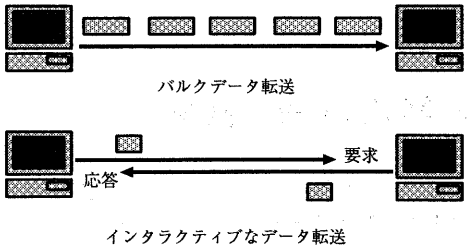

図 1: データ転送の 2 つの形式

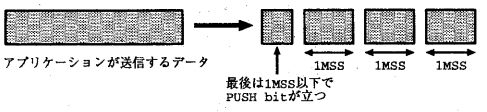

図 2: アプリケーションのデータと PUSH ビットの関係

サービスの代表的なものの一つである TCP にも問題がある事はわかっており，その中のいくつかについてはオプションとして提案されているものがある [4][5]. しかしそれらは IPv6 上でも特に必須の機能とは定義されておらず，IPv4 と全く同一の TCP が利用されようとしている．これでは IPv6 上でも TCP の問題点がそのまま残る事になり，IP が v4 から v6 になったとしても現在のインターネットで見られるパフォーマンスの問題点は必ずしも改善されるとは限らない．このため，現在の TCP をできる限り改善し，その結果を標準に反映させる活動を積極的に行う必要がある．

本研究では既存の TCP の性能に関する問題の中で明確な解決策が打ち出されていない TCP デッドロック問題の解決法を提案する．さらに，この提案を IETF の活動を通じて，標準化することをねらう．本稿では，デッドロック問題の概要，これまでに提案されてきた解決案とその問題点，さらに，本研究で提案する解決法とその優位性について述べる．

2 TCP のデータ転送モデル

TCP ではアプリケーションで通信が行われる場合には 2 つの転送モデルがあると考えて設計されている [6]. 一つはバルクデータ転送で，もう一つはインタラクティブなデータ転送である．それぞれのデータ転送の様子を図 1に示す．

バルクデータ転送の場合にはデータが最大セグメント長 (MSS : Maximum Segment Size) に区切られて大量に片方向に流れる．このとき，送信されるアプリケーションのデータは図 2に示すような形で送信される．データが MSS より大きい場合は MSS 単位に分割され，それぞれ一つの IP データグラムとして送信される．

インタラクティブなデータ転送では一固まりのデータを送信したら送信を停止し応答を待つ．そのデータを受信したホストはそのデータを処理して返事となるデータを送信する．データが MSS より大きい場合には，バルクデータ転送と全く同じ扱いになり MSS 単位に分割されて転送される (本稿ではこれもバルクデータ転送と呼ぶ)．ただし，データの最後を含むセグメントでは図 2 に示すように PUSH ビットが立てられる．このビットはそのセグメント中に，アプリケーションに処理してほしいデータ単位が含まれていることを意味し，PUSH ビットが立っている場合は途中で無用なバッファリング処理が行われない事になっている．なお PUSH ビットの設定機構は TCP では必ず実装しなければならなず，PUSH ビットのための API (Application Programming Interface) を用意するか，さもなければ TCP モジュールが自動的に PUSH ビットを設定しなければならない [7]. TCP モジュールが自動的に PUSH ビットを設定する場合は，送信バッファに格納されている最後の未送信データの送出時に設定される．

バルクとインタラクティブの 2 種類のデータ転送を効率良く行うために，TCP ではデータの送信処理や受信時の確認応答 (ACK : Acknowledgment) の処理に工夫が施されている．

送信時には Nagle のアルゴリズム [8] とスロー・スタート [9] という二つのアルゴリズムが働く．Nagle のアルゴリズムは，パケット全体に占めるデータの割合が小さいパケットの送信を抑制しネットワークの利用効率を向上させるための仕組みである．実際には 1MSS 未満のパケットを小さなパケットと定義し，確認応答が来ていない小さなパケットがある限り小さなパケットを送信しない様に処理する．このため，パケットが往復する間に最大 1 つしか小さなパケットを送信しなくなり，ネットワークの利用効率を向上させる事ができる．また，スロー・スタートは新たなデータ送信がネットワークへの急激な負荷となることを防ぐためにデータの送信量を調節する機構である．輻輳ウィンドウ (cwnd : Congestion Window) を定義し，通信開始時，およびタイムアウトによる再送の場合は初期値として輻輳ウィンドウを 1MSS に設定し，確認応答毎に 1MSS づつ増やしていく．実際にデータ転送に使用されるウィンドウの大きさは受信ホストから通知されるウィンドウと輻輳ウィンドウの小さい方の値になる．

TCP の確認応答処理では，遅延確認応答 (Delayed ACK) の機構を実装する事が強く奨励されている [7]. そして，実際にほとんどの TCP/IP プロトコルスタックで実装されている．遅延確認応答を実装する場合には，遅延時間を 0.5 秒以内[1]にしなければならない

[1]伝統的な TCP/IP プロトコルスタックである BSD の実装では，セグメントが到着するタイミングにより 0.0〜0.2 秒の値

ことと，2つのフルサイズのセグメント (2MSS) を受信した時に遅延なく確認応答を返送しなければならないことが決められている．この遅延確認応答は小さなパケットがネットワークを埋め尽くし，ネットワークの利用効率を著しく悪化させるシリー・ウィンドウ・シンドローム (SWS : Silly Window Syndrome) と呼ばれる現象を回避するための機能の一つとして提案された [10]．また，インタラクティブなデータ転送時にピギーバックが起きやすくなったり，ホストの負荷を低減させる働きもある．ピギーバックとはデータセグメントと一緒に確認応答を転送する意味で使われる．データセグメントと確認応答を別々に送るのに比べてネットワークの使用効率もホストの負荷も減らす事ができる．

3 TCP デッドロック

2節で述べたように，TCP には Nagle アルゴリズムやスロー・スタートというデータの送信を抑制する機構と，遅延確認応答という確認応答を遅延させる機構が備えられている．しかし，これらの処理がうまく噛み合わなくなると，データ転送がデッドロック状態になる．本稿では次の状態になることを TCP デッドロックと呼ぶ．

送信ホスト: 続けて送信すべきデータがあるが，前に送信したセグメントに対する確認応答が到達するまでデータを送信しない

受信ホスト: 確認応答を送信していない受信セグメントがあるが，さらなにデータが到着しないと確認応答を送信しない

この状態は遅延確認応答の機構により，確認応答が遅延される期間だけ続く．また，このデッドロックは基本的にバルク・データ転送の時に起こる．インタラクティブなデータ転送のときもデッドロックが起きる可能性はある．しかしこれはアプリケーション・プログラムに問題があるから発生するのであり，本研究の対象外だと考える．次に，バルクデータ転送時の TCP デッドロックの詳細について説明する．

3.1 スロー・スタートの問題

スロー・スタートは短いデッドロックを起こす事がわかっている [11]．スロー・スタート開始時には輻輳ウィンドウは 1MSS である．そのため送信ホストは 1MSS のセグメントを 1 つしか送信しない．受信ホストは 1MSS のセグメントを 1 つ受信しても，遅延確認応答処理のためさらなるセグメントの到着を待ち，すぐには確認応答を送信しない．そのため，通信開始時や，タイムアウト後の再送では最初の 1 セグメントは必ず遅延確認応答となる．

になる．

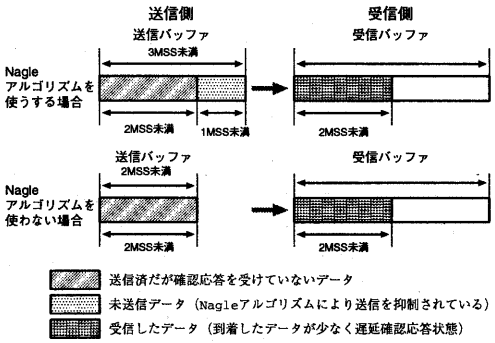

図 3: 送信バッファサイズが小さい場合のデッドロック

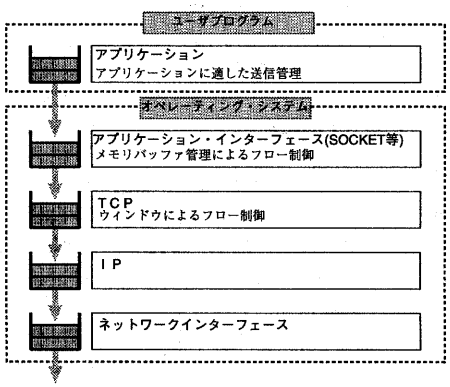

図 4: ネットワーク・モジュールの階層モデル

3.2 送信バッファが MSS に比べて十分大きくない場合

送信ホストのバッファが 3MSS より小さい場合はデッドロックが起きる場合がある [12]．厳密には Nagle アルゴリズムが動作する時は 3MSS 未満で，Nagle アルゴリズムを無効にした時は 2MSS 未満の場合である．TCP では送信したセグメントが喪失した場合に再送しなければならないため，確認応答されていないデータを送信バッファから削除する事は出来ない．そのため，送信バッファが小さいと図 3 に示すような状態になりデッドロックが起きる[2]．このデッドロックはデータ転送が終了するまで連続して発生するため，スループットが極端に悪化する．

3.3 ネットワーク・モジュールの階層化の問題

アプリケーションがデータを送信する時のメッセージの大きさによってはデータ転送時のスループット

[2]受信バッファが小さい場合には受信データが 2MSS 未満でも確認応答を返す実装があり，その場合はデッドロックは起こらない．

が著しく悪化する場合がある．これはネットワーク・モジュールの階層化の問題として知られている [13]．多くのシステムでは TCP のモジュールは図 4に示すように，アプリケーション・プログラムと TCP モジュールの間に，TCP とアプリケーションを結ぶインターフェースがある．このインターフェースは OS の内部に含まれメモリバッファの管理などの機能を受け持つ．このメモリバッファに効率良くデータが格納されない場合にはデッドロックが起きる可能性がある．たとえメモリバッファが 3MSS 以上あったとしても，アプリケーションから送信要求されたデータがバッファに効率よく格納されず，結果として 3MSS 未満のデータしか格納できなかった場合は 3.2節と同じ状態になりデッドロックが起こる．

3.4 経路 MTU 探索の問題

経路 MTU 探索 (Path MTU Discovery)[14] は IPv4 ではオプションの機能であり，あまり実装されていなかった．しかし，IPv6 ではルータが MTU を越えるデータグラムのフラグメント処理をしなくなるため，経路 MTU 探索の実装が強く求められる様になった．この経路 MTU 探索は TCP に新たなデッドロック問題を引き起こす可能性がある．

TCP では MSS オプションを実装する事が必須事項となっている．一般的な実装ではコネクションの確立時にこのオプションを利用して通信するホスト間で互いに最適だと判断する MSS の値を交換する．そして両者の中の小さい方の値を MSS と決める．この場合，両者の MSS の値は同じになる．しかし，経路 MTU 探索が実装されているホストでは両者の MSS の値が同じになるとは限らない．通信開始時に交換して決定した MSS は必ずしも経路 MTU から IP や TCP のヘッダを引いた値にはなっていない．そのため，通信開始後に経路 MTU 探索で求まった MTU から計算された MSS が，MSS オプションで決められた MSS と異る場合がある．この場合，経路 MTU 探索で求まった値に MSS の値が変更される．しかし，MSS の値が変更されても新たな MSS は受信ホストには通知されないため，受信ホストは送信ホストの MSS を正しく把握しないまま遅延確認応答をしなければならなくなる．もし，送信ホストの MSS が，コネクション確立時に MSS オプションで決定された値よりも小さく変化した場合は，デッドロックが発生する可能性がある．また，大きく変化した場合は遅延確認応答が機能しなくなる可能性がある．

4 デッドロック解決案

今までにいくつかデッドロックの解決方法が提案されている．それぞれの解決案とその問題点について述べる．

4.1 今までに提案されたデッドロック解決案

i. 送信バッファを大きくする解決案

Comer らはバッファを常に MSS の 3 倍以上に設定すれば 3.2節の「送信バッファが MSS に比べて十分大きくない場合」のデッドロックを解決できると主張した [12]．これは TCP の実装を変更しなくても OS のパラメータを変更したりアプリケーション・プログラムを修正すればデッドロックを回避できるという利点がある．

ii. 遅延確認応答を止める解決案

Moldeklev らは遅延確認応答処理を止める事でデッドロック問題を解決出来ると主張した [15]．デッドロックの原因は全て遅延確認応答が関係している．そのため，確認応答を遅延させなければデッドロックは起こらない．

iii. PUSH ビットで遅延確認応答を制御する解決案

NetBSD1.1 では PUSH ビットが立ったセグメントを受信した場合，確認応答を遅延させずにすぐに送るように実装されている[3]．その結果，送信ホストのバッファで未送信データが無くなった時には確認応答が遅延せず，それが原因で生じるデッドロックは起きなくなる．

iv. 送信ホストが遅延確認応答を制御する解決案

筆者らは，送信すべきデータがあっても送信を停止しなければならない場合，それを受信ホストに伝えてから停止すればデッドロック問題を解決できると主張した [16]．デッドロックは送信ホストがデータの送信を停止したのに受信ホストが確認応答を送信しないから起きる．そのため，送信ホストのデータ転送が停止したことを受信ホストがわかっていれば確認応答を遅延させずに送信することによりデッドロックを回避する事ができる．

4.2 デッドロック解決案の問題点

i の解決案は 3.2節のデッドロック以外は解決できないという欠点がある．

ii の解決案は遅延確認応答を止めるという提案自体が問題である．そもそも遅延確認応答は SWS を防ぐための戦略であり，これを止めると SWS が発生する危険性がある．現在のインターネットでは多彩なデータリンクが用いられていることを考えると，SWS 防止は必須であり，この解決案は受け入れられるものではない．また，インタラクティブなデータ転送時にピギーバック処理が行われなくなり，無駄なトラヒックが増えたり，ホストの負荷が増えるなどの問題がある．この提案は確かにデッドロックを解消できるが，副作用が大きいという欠点がある．

[3] FreeBSD 2.1 にも実装されておりオプションで同様の処理ができる．

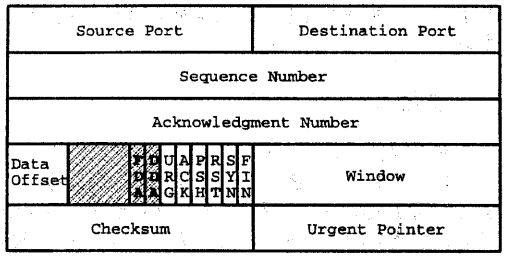

図 5: 提案する TCP のヘッダ

iii の解決案は PUSH ビットが設定されずにデッドロックが起きる場合は解決できない．基本的には PUSH ビットとデッドロックの間には完全な相関関係はなく，デッドロックが起きる時に必ず PUSH ビットが設定される分けではない．そのため，TCP デッドロックに対してほとんど何の解決にもなっていない．また，インタラクティブなデータ転送では PUSH ビットが設定されたセグメントを受信した後のデータ送信でピギーバックが行われるが，PUSH ビットが立ったセグメントの確認応答を遅延させない場合にピギーバックできなくなる．そのため，ii の提案と同様に無駄なトラヒックが増えたり，ホストの負荷が増えるなどの問題がある．

iv の解決案は TCP のデッドロックを解決できるが，PUSH ビットを利用して実装することが提案されているため，iii の実装と同じようにピギーバックできなくなるという問題がある．

5 デッドロック解決のための TCP の拡張の提案

TCP デッドロックは，受信ホストが送信ホストから明確な指示を得ずに独立して確認応答の返送処理をしている事が根本的な原因となっている．そのため，送信ホストが完全に確認応答の制御をすればデッドロックは解決できる．

本稿では 4.1節の iv の提案を拡張して，図5に示したように DDA (Don't Delay ACK) と FDA (Force Delay ACK) という 2 つの制御ビットを TCP ヘッダの予約ビット [17] に追加する事を提案する．このビットを正しく利用すれば TCP の既存の遅延確認応答と同じ処理を送信ホストが制御できるようになり，デッドロックを根本的に解決できる．また，経路 MTU 探索をする時に受信ホストが送信ホストの MSS の値について知る必要が無いため，3.4節の問題も完全に解決できる．また，ピギーバックが起こらなくなるなどの問題もない．

DDA (Don't Delay ACK)

DDA ビットは遅延確認応答をせずに，すぐに確認応答を返送させる働きがある．デッドロックが起きる状態の時にこのビットを立てればデッドロック状態にならずに済む．ホストは FDA ビットを次のように処理する．

送信ホスト

1. 2MSS のデータを送信する度に 1 にする[4]．
2. keep alive や window probe を送信する場合に 1 にする．
3. 送信したセグメントに対してピギーバックが期待できず[5]，かつ確認応答が来ない限り送信処理が停止する場合[6]に 1 にする．(デッドロックの回避)
4. SYN，FIN，RST が 1 のセグメントや純粋な確認応答の場合には 0 にしなければならない．ただし，FDA ビットが 1 のときはその処理をする．

受信ホスト

1. 1 のときにはそのセグメントに対応する確認応答を遅延なく返送する．
2. 0 のときには通常の遅延確認応答処理を行う．

FDA (Force Delay ACK)

このビットは，遅延確認応答の仕様で決められている「フルサイズのセグメントを 2 つ受信した場合に確認応答を送信する」という機構をキャンセルする働きがある．ホストは FDA ビットを次のように処理する．

送信ホスト

1. DDA で確認応答を正しく制御する場合，通常のデータセグメントのすべてで FDA ビットを 1 にする．
2. SYN，FIN，RST が立ったセグメントでは FDA ビットは 0 でなければならない．

受信ホスト

[4] バルクデータ転送でフルサイズのセグメントを連続して送信する場合は，1 つおきにこのビットを 1 にする．

[5] 実装に依存するが，PUSH API を持たない 4.4 BSD Lite の実装の場合は 1MSS の大きさを持つセグメントが送信される場合だと考えられる．

[6] 実装に依存するが 4.4 BSD Lite の実装の場合には次の場合に確認応答が来ない限り送信処理が停止する [16]．

- 輻輳ウィンドウの大きさが 1MSS のとき．
- セグメントの送信後に送信可能なウィンドウの残りが 1MSS 未満になるとき．
- セグメントの送信後に送信ソケットバッファの残りが 1MSS 未満になるとき．
- ソケットが管理するメモリが足りなくなり処理が停止するとき．(ただし，これを事前に知ることはできないため，Nagle のアルゴリズムを無視して TCP のバッファに格納されている未送信データを強制的に送信する機構が必要である．またこの処理が SWS を発生させることが無い様に制御する必要がある)

1. 1のときにはそのデータセグメントに対する確認応答を遅延させなければならない.
2. 0のときにはなにも処理を行わない.
3. 純粋な確認応答の場合はDDAビットの値は無視する.
4. DDAとFDAがともに1の場合はDDAを優先させる.

6 本提案のインターネットへの適用について

広く運用されているインターネットでヘッダのフォーマットを変更する事は互換性上大きな問題となる可能性がある. しかし, この提案の場合は既存のIPv4 ネットワークで実装したとしても大きな問題とはならない. 通常TCPの予約ビットは送信時に0に設定され, また受信時には無視される. かりに, 本稿で提案するDDAビットとFDAビットを実装しているホストと実装していないホストが通信を行っても, 現状のTCPの通信と全く同じように処理される. 問題があるとすれば次の3点が考えられる.

1. TCP/IPのヘッダ圧縮機能[18]を利用するネットワークを通る場合, 圧縮機能が働かないか, DDAビット, FDAビットがともに0に設定される.
2. TCPの予約ビットが1になっていると誤動作するホストまたはルータがある場合.
3. DDAとFDAに対応するビットのいずれかまたは両方に1をつけて送信するホストがある場合.

1.で未定義ビットが0に設定される場合には既存の遅延確認応答と同じ処理が行われる事になり事態はなにも変化しない. ヘッダ圧縮を利用できない場合はスループットが悪化するが, それ以外の問題はない. 2.や3.の機器がある場合は問題だがそういう機器はほとんど無いと考えられる.

また, IPv6の場合はまだ実験段階であり, 本格的に運用される前にこの提案が標準となれば何の問題もなくTCPのデッドロックが解決される事になる. 特にIPv6では経路MTU探索が頻繁に行われるようになるため本提案はその問題を解決するための有効な手段となる.

7 今後の展開

どんなに有効な提案に思われても, それが絵に書いた餅の状態ではインターネットでは評価されない. 実装して, 実際に使用して評価しなければそれが本当に有効な提案なのか判断できないからである. 今後, この仕様を実装し, TCPのあらゆるデッドロックが防止されるか, また, 実装上の注意点がないか調査を行う予定である. また, 提案内容をInternet Draft化しIETFで議論を行い, IPv6時代のTCPのInternet標準となるように活動を展開する計画である.

8 まとめ

本稿では, インターネットのトランスポート・プロトコルであるTCPのデッドロック問題に焦点を当てて解決法について提案した. 送信ホストが確認応答の処理を制御できるため, デッドロックの問題を根本的に解決できる. この方法はTCPの転送モデルをもとにして考案してありTCPのピギーバック機構やPUSH機構との相性がよくなるように設計した.

TCPのヘッダの予約ビットを利用するが, IPv6では運用される前に標準化すれば問題は全くなく, 早急な標準化活動が必要である.

参考文献

[1] S. Deering, R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", RFC 1883, Jan 1996
[2] Christian Huitema, "IPv6: The New Internet Protocol", Prentice Hall PTR, 1996
[3] Christian Huitema, 村井純監訳, "IPv6: 次世代インターネット・プロトコル", (株)プレンティスホール出版, 1997
[4] V. Jacobson and R. Braden and D. Borman, "TCP Extensions for High Performance", RFC 1323, May 1992
[5] M. Mathis, J. Mahdavi, S. Floyd, "TCP Selective Acknowledgment Options", RFC 2018, October 1996
[6] W. Richard Stevens, "TCP/IP Illustrated, Volume1 The Protocols", Addison-Wesley Publishing Company, 1995
[7] R. Braden, "Requirements for Internet Hosts — Communication Layers", RFC 1122, October 1989.
[8] John Nagle, "Congestion Control in IP/TCP Internetworks", RFC 896, January 1984
[9] W. Stevens, "TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms", RFC2001, Jan 1997.
[10] David D. Clark, "Window and Acknowledgment Strategy in TCP", RFC 813, July 1982
[11] 村山 公保, 門林 雄基, 山口 英, "TCP性能評価システムDBSの構築", インターネットコンファレンス, 日本ソフトウェア科学会研究会資料シリーズNo.3, pp 39-44, 1996
[12] Douglas E. Comer and John C. Lin, "TCP Buffering And Performance Over An ATM Network", Purdue Technical Report CSD-TR 94-26, 1994
[13] Jon Crowcroft, Ian Wakeman, Zheng Wang, and Dejan Sirovica "Is Layering Harmful?", IEEE Network Magazine, vol.6, pp 20-24, January 1992
[14] J. Mogul, S. Deering, "Path MTU Discovery", RFC1191, November 1990.
[15] Kjersti Moldeklev and Per Gunningberg, "How a Large ATM MTU Causes Deadlocks in TCP Data Transfers", IEEE/ACM Transactions on Networking Vol.3, No.4, pp.409-422, August 1995
[16] 村山 公保, 門林 雄基, 山口 英, "PUSHビットの有効利用によるTCPデッドロック問題の解決", マルチメディア通信と分散処理ワークショップ, 情処ワークショップ論文集 Vol. 96, No.1, pp 229-236, 1996
[17] J. Postel, "Transmission Control Protocol", RFC 793, September 1981
[18] V. Jacobson, "Compressing TCP/IP Headers for Low-Speed Serial Links", RFC1144, February 1990