PUSHビットの有効利用によるTCPデッドロック問題の解決

村山 公保† 門林 雄基‡ 山口 英†

†奈良先端科学技術大学院大学情報科学研究科
‡大阪大学大型計算機センター研究開発部

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には，デッドロックが起きることがあるという問題がある．この状態になると，ネットワークがどんなに広帯域であったとしても，またどんなに帯域が余っていたとしても間欠的にしかセグメントが送信されずスループットが極端に悪化する．本研究では，現在のTCPのPUSHビットを拡張して利用することでデッドロック問題を解決できることを示す．またこの拡張には副作用が少なく安全な方法であることも示す．

1 はじめに

現在，インターネットは世界的な規模で急速に発展・普及している．しかし，設定や管理が難しいのが難点だと言われている．このためコンピュータをネットワークに物理的に接続するだけでインターネットを利用できる「プラグ&プレイ」の実現が望まれている．今後のインターネットの発展のためにはプラグ&プレイ技術が必要不可欠だと考えられている．次世代IPであるIPv6ではプラグ&プレイを可能にするための仕組みが仕様に採り入れられ，実現へ向けて研究が進められている[1].

現在，インターネット上のサービスの大部分はTCPで構築されている．このためTCPは最も重要なトランスポート・プロトコルと言うことができる．しかし現在のTCPはプラグ&プレイに必ずしも対応できるようになっていない．現在のTCPを使う限りコンピュータをネットワークに接続しただけではネットワークを有効に利用できない事が起こりうる．場合によっては，オペレーティングシステムのパラーメータを変更したり，アプリケーションプログラムを変更しなければ満足な通信ができない可能性もある．

本稿ではTCPをプラグ&プレイに対応させるための第一歩として，TCPデッドロック問題の解決に焦点を当てる．まずデッドロックが起こる原因を述べる．そして解決法を提案し，実装して測定した結果を示す．

A Solution of the TCP deadlock problem by making good use of PUSH bit, Yukio MURAYAMA and Youki Kadobayashi and Suguru YAMAGUCHI, Graduate School of Information Science, Nara Institute of Science and Technology

2 TCPのプラグ&プレイへの道

TCPには(1)TCPのデータ送信と確認応答がデッドロックする(2)ウィンドウサイズをアプリケーションが設定しなければならず，ウィンドウサイズの拡張オプションが有効に利用されていない(3)高帯域高遅延環境下ではスロースタートによってスループットが向上するのに時間がかかる(4)送信制御のない高速データリンクで輻輳が起きると性能が極端に悪化する，などの問題がある．これらのいくつかの問題は，システム管理者やプログラマが注意深く設定を変更すれば解決できる場合もある．しかし，一般ユーザが設定を変更するのは困難であり，また，設定の変更を誤れば大きな性能低下につながる恐れがある．

TCPのこれらの問題はインターネットがより使いやすくなっていくためには解決しなければならない課題である．解決されなければIPレベルでプラグ&プレイが実現されたとしても，簡単にはインターネットを快適に使うことができない．本研究ではこれらの問題の中の(1)TCPデッドロック問題の解決に焦点をあてる．

3 TCPデッドロック

TCPにはデータの送信処理と確認応答処理がデッドロックする問題がある．ここでのデッドロックと

は具体的には次の状態になることを意味する．

送信ホスト　確認応答(ACK)が来るまでデータを送信しない

受信ホスト　データが到着するまで確認応答を送信しない

この問題は，輻輳回避機構であるスロースタートや，シリー・ウィンドウ・シンドロームを回避するためのNagleアルゴリズム，OSのソケットとTCPの間のモジュール間の処理といった送信ホストの処理と，受信ホストの遅延確認応答(Delayed ACK)が原因で起こる．デッドロック状態は遅延確認応答を管理している200ミリ秒間隔のスロータイマが切れるまで続く．その結果，約200ミリ秒間隔でしかパケットが送信されなくなり，スループットが極端に悪化する．それぞれの問題について説明する．

3.1 スロー・スタートの問題

スロー・スタートには問題があることがわかっている[2]．通信開始時にTCPではスロースタート機構により輻輳ウィンドウが1セグメントに設定される．そのため送信ホストは1セグメントしかデータを送信できない．しかし，受信側は1セグメントのデータを受信してもさらなるデータの到着を待ち，すぐには確認応答を送信しない．その結果，最初のセグメントはかならず遅延確認応答となる．この状態は送信ホストの輻輳ウィンドウが，受信ホストのウィンドウを更新させる大きさより大きくなるまで続く*．その間スループットは極端に小さくなる．この問題は特に高速LAN環境の場合にパフォーマンスの低下の原因になりうる．

3.2 送受信バッファサイズの不整合の問題

Douglas E. Comerらが ATM ネットワークで結ばれた2つのホスト間で送受信のバッファサイズに着目してデータ転送のスループットを測定したところ，スループットが極端に小さくなる場合があることが見つかった[3]．それは，次の場合に起こる．

- 送信バッファの大きさが，受信ホストがウィンドウを更新させる大きさより小さい場合

この場合，送信ホストが送信バッファに溜っているデータをすべて送信し尽くしたとしても，受信ホストはデータの受信を待ち続ける．その結果，デッドロックが起こる．

*BSD/OS(4.4BSD Lite)では2セグメント

3.3 Nagleアルゴリズムの問題

TCPには小さなパケットがネットワークを埋め尽くすシリー・ウィンドウ・シンドロームと呼ばれる現象を回避するために，Nagleのアルゴリズムとよばれる仕組みが組み込まれている[4]．Nagleアルゴリズムは小さなデータの送信を抑制することで小さなパケットが流れることを防ぐ機構である．TCPの実装では小さなパケットとは最大セグメント長の1MSS未満のパケットと定義され，できるだけ1MSS単位でデータをまとめて送信しようとする．ただし，telnetなどの遠隔端末の利用も考慮して，確認応答待ちの送信データがない場合に限り小さなパケットを一つだけ送信できる．このNagleのアルゴリズムにより次の症状のときに送信処理がブロックされる．

- 送信バッファサイズが1MSS以下
- 送信可能なウィンドウサイズが1MSS以下

これは，ソケットとTCPのバッファ管理の矛盾などと組合わさるとさらにデッドロックが起こりやすくなる．

3.4 TCPとソケットの間の問題

送信するメッセージサイズの大きさによってはデータ転送が著しく低下する場合がある．これは，Jon Crowcroftらによって発見され，TCPのバッファ管理とソケットのバッファ管理の間に矛盾する部分があることが原因だとわかった[5]．

ソケットはユーザ・アプリケーションとTCPの間を取り持ちながらデータの受渡しをする役目を担っている．ところが，ソケットのバッファ管理とTCPのウィンドウ制御の整合がとれずに，送信が停止することがある．BSDではネットワークで送受信されるデータはmbuf(Memory Buffer)と呼ばれる構造体に格納される．ソケットとTCPは同じmbufを使って処理をするが，mbufに格納されているデータの数え方が異なっている．ソケットはデータが入っているmbufの大きさで計算するが，TCPはmbufの中に入っているデータで計算する．このため使用可能なすべてのmbufに小量のデータがのみが入っている場合，TCPはmbufにデータを格納できると判断するが，ソケット格納できないと判断する．この矛盾の影響のためTCPはバッファに余裕があると判断しているにもかかわらず，ソケットがアプリケーションからの送信要求をブロックしてしまい，結果としてTCPの送信が停止する場合がある．こ

の場合確認応答が来てTCPのバッファが開放されるまでソケットは停止することになる．そのため確認応答が遅延する場合にはデッドロック状態になる．

3.5 TCPデッドロックに関する考察

TCPのデッドロックの問題はATMなど大きなMTUのデータリンクの出現により明らかになった．そして，大きなMTUのネットワークにコンピュータをつなぐときには，コンピュータごとに送受信バッファの初期値が異なっており，デッドロックが起きる危険性がある．

しかし，逆に腕時計やICカードなど小型の機器をインターネットに接続するとこと考えると，バッファサイズが小さくなりEthernetでもデッドロックの問題が起こる可能性がある．将来，インターネットに多種多様な機器が接続されると予想すると，TCPデッドロック問題を根本的に解決することは重要な課題であることがわかる．

4 PUSHビットの拡張によるデッドロック問題の解決の提案

送信データがそれ以上来ないことを受信ホストが知っていれば，確認応答をすぐに送信することでデッドロックを避けることができる．これは送信ホストが送信を停止することを受信ホストに通知してから停止すれば実現できる．

本研究では，TCPのヘッダ中に送信が停止されることを意味するフラグを付けることでデッドロック問題の解決を提案する．このフラグは具体的には次のように処理される．

送信側 そのセグメント以後の送信を停止するとき，停止を意味するフラグを立てる．

受信側 送信の停止を意味するフラグが立っている場合，遅延なく確認応答を送る．

TCPヘッダ中に新たにフラグを設けることは，互換性に問題があり危険である．そのためTCPのPUSHビットでこのフラグを代用することを検討する．本研究ではPUSHビットを次のように使うことでデッドロックを解決する2つの方式について考える．

- デッドロック防止の機能のみ(PUSHビット本来の機能は削除)
- PUSHビット本来の機能 + デッドロック防止の機能

5 仕様や実装上のPUSHビットの意味

TCPの仕様では，PUSHビットは送信ホストが受信ホストに対して受信したセグメントを受信バッファに貯めないでアプリケーションにすぐに渡してほしい場合に立てる[6]．これは，受信したデータをバッファリングするOSで必要となる機能である．

PUSHビットの機能はC言語標準ライブラリのfflush関数に似ている．C言語の標準ライブラリを利用してデータを出力する場合，データは一旦バッファに格納される．fflush関数を実行するとその時点で貯まっているデータがすべて吐き出される．PUSHビットは同様のことをネットワークでつながれた相手のコンピュータに対して行うときに利用される．大きな違いは，fflush関数はプログラマが適宜プログラムに書くことができるが，PUSHビットはアプリケーションプログラムからは設定できないことである．PUSHビットはOSが自動的に設定する．そのため，アプリケーションプログラマはPUSHビットがどのようなときに設定されるのかを理解しながらアプリケーションを開発しなければならない．そうしなければ，送信先のホストの受信バッファにデータがいつまでもたまりデッドロックが起こりかねない．

4.4 BSD Liteの実装では，セグメントの送信時にバッファに格納されているデータをすべて送信した場合にPUSHビットを付ける．これは，アプリケーションの送信要求単位でPUSHビットが付くことを意味する．ただし，送信要求されたデータがすぐに送信されずバッファの待ち行列に入った場合は最後のデータ送信時にPUSHビットを立てる．

アプリケーションプログラマは，クライアントとサーバが対話的に処理をする場合は，相手への要求メッセージまたは応答を一回のシステムコールで送信しなければならない．複数回に分けて送信すると，Nagleアルゴリズムの影響もあり，スループットが低下する可能性がある．

しかし，4.4 BSD Liteをふくむほとんどの OSの実装では受信したデータをバッファリングしておらず，PUSHビットは有効に利用されていない．そのため，きちんとプログラムを書いていなくても問題が表面化しにくい．逆にいえば，PUSHビットの機構を必要とするOSを利用する場合，現在インターネットで正常に動作しているアプリケーションであったとしても正常に動作しないものが存在する可能性がある．

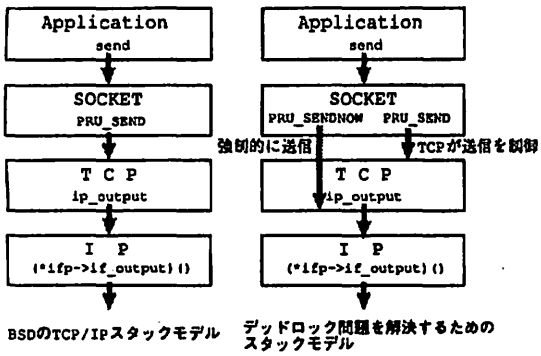

図 1: ソケットと TCP のモジュール間通信の強化

6 PUSH ビットの拡張案の実装

本研究では 4.4 BSD Lite のソースをもとにした BSD/OS 2.1 で TCP デッドロック問題の解決案を実装し検証実験を行った．4.4 BSD Lite では (1) ソケットの処理が停止するとき (2)TCP が輻輳ウィンドウが 1MSS に設定されスロースタートをするとき (3) セグメント送信後，TCP が送信可能なデータが 1MSS 以下になる場合に送信がブロックされる．この 3 つの状態のときに TCP が PUSH ビットを立ててセグメントを送信する．

6.1 ソケットと TCP のモジュール間通信の強化

ソケットが処理を停止することを TCP に伝えるために PRU_SENDNOW 要求を追加する．この要求を TCP が受け取った場合にはその時点で送信可能なセグメントに PUSH ビットを付けて強制的に送信する．図 1に階層間の処理のイメージを示す．これにより，ソケットバッファが処理を中断していることを受信ホストの TCP に伝えることができデッドロックが回避できる．

ソケット層では，アプリケーションからの送信要求の処理中にバッファが足りなくなり処理を中断する場合と，TCP に送信要求をする段階ですでにバッファが足りなくなっている場合に PRU_SENDNOW 要求をする．ただし，PRU_SENDNOW が連続して要求されるとシリー・ウィンドウ・シンドロームが起こる危険性があるため，PRU_SENDNOW リクエストで送信したセグメントに対する確認応答がきていない場合には TCP はこの要求を受け付けない．

6.2 TCP で送信停止が起きる場合の PUSH ビットの設定

セグメントの送信時に，次のいずれかを満たす場合には PUSH ビットを立ててセグメントを送信する．

- 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ウィンドウサイズが 1MSS 未満になるとき
- 送信ソケットバッファの残りのサイズが 1MSS 未満になるとき

7 検証実験

本研究では Ethernet を利用して検証を行った．TCP のデッドロック問題は MSS の大きさ，バッファサイズ，メッセージサイズなどの送信アルゴリズムのパラメータに関係しており，データリンクの伝送速度には関係してない．そのため，MSS の違いを考慮すれば Ethernet で実験した結果を ATM などの伝送速度の違うネットワークに当てはめて考えることができる．

つぎの 5 種類のアルゴリズムで測定を行った．

実装 1 BSD/OS 2.1 の実装のまま
実装 2 確認応答を遅延させない
実装 3 PUSH ビットが立ったセグメントが来たときは確認応答を遅延させない．
実装 4 送信がブロックされるときのみ PUSH ビットを立てる．PUSH ビットが立ったセグメントが来たときは確認応答を遅延させない
実装 5 送信がブロックされるときも PUSH ビットを立てる．PUSH ビットが立ったセグメントが来たときに確認応答を遅延させない

それぞれのアルゴリズムを分類して表 1にまとめる．実装 1 は 4.4 BSD Lite の実装そのままである．実装 2 は Kjersti Moldeklev らによって提案されたアルゴリズムである [7]．Kjersti Moldeklev らは遅延確認応答をやめれば TCP デッドロックを解決できると主張している．デッドロックの原因はすべて遅延確認応答に関係しており，確認応答を遅延させなければデッドロックは起こらなくなると考えている．実装 3. は NetBSD や FreeBSD で採用されているアルゴリズムである．一部のデッドロック問題を解消するためにこの実装が考案されたと考えられる．実装 4 と 5 が本稿で提案するアルゴリズムである．実装 5 は結果的には実装 3 と実装 4 を組み合わせたものになっている．

表 1: 評価に使用した 5 つの実装

	1	2	3	4	5
未送信データがなくなるとき PUSH ビットを立てる	○	○	○		○
送信がブロックされるときに PUSH ビットを立てる				○	○
PUSH ビットが立ったセグメントを受信しても特別な処理をしない	○				
PUSH ビットが立ったセグメントを受信したときのみ確認応答を遅延させない			○	○	○
常に確認応答を遅延させない		○		○	

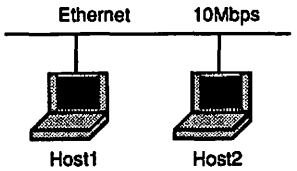

図 2: 実験環境

表 2: 測定環境と設定

CPU	Twinhead Subnote(486 33MHz)
OS	BSDI BSD/OS V2.1
Ethernet Card	3Com EtherLink III 3C589C
MTU	1500octet
MSS	1448octet (タイムスタンプオプション付き)
性能測定ツール	Netperf, DBS
送信メッセージサイズ	表記がないものは 8192byte
Netperf のデータ送信時間	10 秒

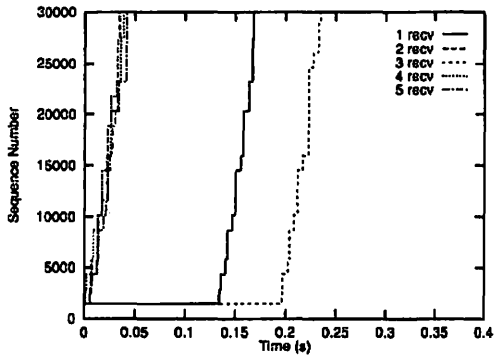

図 3: 受信データの総数の時間推移

実験に使用した環境を図 2と表 2に示す．この環境でバルクデータ転送を行い送信データの到着時刻やスループットを測定した．

8 結果

8.1 スロースタート時のデッドロックの検証

表 3: 送受信バッファサイズ (実装 1)

送信バッファ \ 受信バッファ	512	1024	2048	4096	8192	16384
512	0.01	0.01	0.01	0.01	0.01	0.01
1024	0.01	0.01	0.01	0.01	0.01	0.01
2048	1.63	2.20	3.04	0.08	0.08	0.08
4096	1.63	2.37	3.25	0.09	0.09	0.09
8192	1.63	2.37	3.98	4.92	6.48	6.37
16384	1.60	2.40	3.91	5.13	7.31	7.54

表 4: 送受信バッファサイズ (実装 2)

送信バッファ \ 受信バッファ	512	1024	2048	4096	8192	16384
512	0.40	0.40	0.41	0.41	0.40	0.41
1024	0.42	0.42	0.43	0.43	0.42	0.42
2048	1.61	2.24	3.10	3.11	3.08	3.12
4096	1.63	2.41	3.43	3.72	3.74	3.65
8192	1.62	2.37	4.59	5.81	6.56	6.59
16384	1.58	2.40	4.46	5.54	6.85	6.92

表 5: 送受信バッファサイズ (実装 3)

送信バッファ \ 受信バッファ	512	1024	2048	4096	8192	16384
512	0.39	0.40	0.41	0.41	0.40	0.41
1024	0.42	0.43	0.42	0.43	0.42	0.43
2048	1.63	2.23	0.08	0.08	0.08	0.08
4096	1.61	2.39	3.42	3.40	3.39	3.44
8192	1.61	2.38	3.79	4.98	6.45	6.39
16384	1.63	2.39	3.68	4.80	7.40	7.36

表 6: 送受信バッファサイズ (実装 4)

送信バッファ \ 受信バッファ	512	1024	2048	4096	8192	16384
512	0.40	0.41	0.40	0.41	0.41	0.40
1024	0.42	0.43	0.43	0.44	0.43	0.43
2048	1.63	2.25	3.12	3.13	3.14	3.12
4096	1.62	2.36	3.59	4.13	4.10	3.43
8192	1.61	2.38	3.91	5.28	6.16	6.04
16384	1.62	2.40	3.92	5.20	7.23	7.35

受信データ総数の推移を図3に示す．測定にはDBSを利用した[2][8]．データの送信は0.0秒に開始しており，最初のセグメントは0.0秒に受信されている．しかし，実装1，3ではスロースタートによるデッドロックが起きたため，最大約0.2秒の間データの送信が停止している．実装2，4，5ではデータ送信の停止は見られずデッドロックは起きていないことがわかる．なお実装1，3ではデッドロックの時間が異なる．これは遅延確認応答の制限時間を決めるスロータイマのタイミングの問題であり測定のたびに結果は変わる．

8.2 送受信バッファサイズごとのスループット

送受信のバッファサイズ (byte) ごとのスループット (Mbps) の関係の測定結果を実装ごとに表 3~7

表 7: 送受信バッファサイズ (実装 5)

送信バッファ \ 受信バッファ	512	1024	2048	4096	8192	16384
512	0.41	0.41	0.41	0.41	0.41	0.40
1024	0.43	0.44	0.43	0.43	0.43	0.43
2048	1.64	2.27	3.14	3.12	3.13	3.08
4096	1.62	2.39	3.45	4.60	4.54	4.58
8192	1.62	2.38	3.95	5.28	6.07	6.11
16384	1.62	2.40	3.96	5.19	7.34	7.37

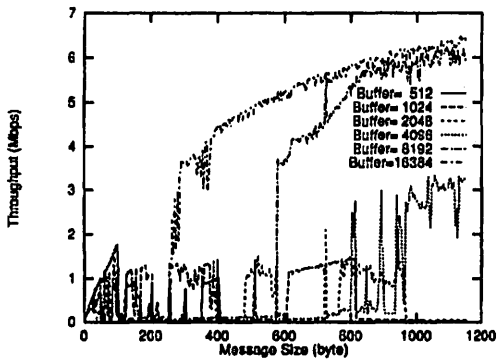

図 4: メッセージサイズとスループット (実装 1)

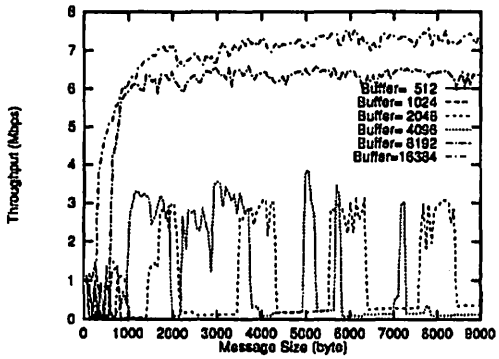

図 5: メッセージサイズとスループット (実装 1)

に示す. 測定には Netperf を利用した [9]. 表 3で 0.1Mbps 未満の部分がデッドロックが起こっている部分である. 送信バッファが 1MSS 以下の部分と, 送信バッファが小さく受信バッファが大きいというバッファサイズの不整合の部分がある. 実装 2, 実装 4, 実装 5 では 0.1Mbps 未満の部分は無くなっておりデッドロックが解消していることがわかる. 実装 3 では送信バッファが 2048byte のときのデッドロックは解消していない.

8.3 メッセージサイズとスループットの関係

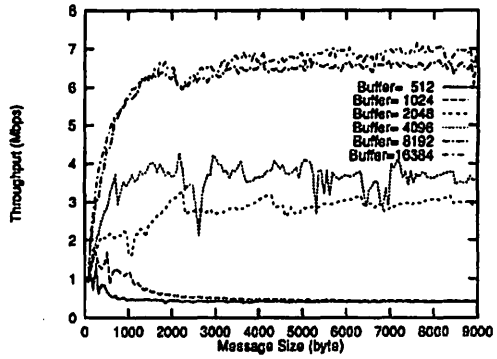

図 6: メッセージサイズとスループット (実装 2)

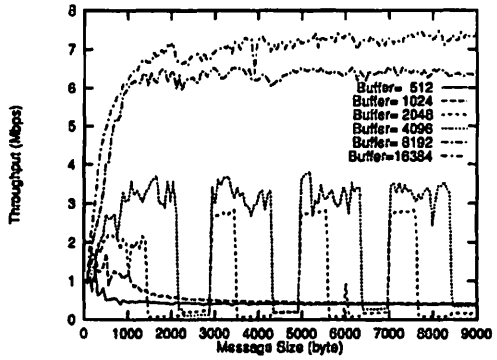

図 7: メッセージサイズとスループット (実装 3)

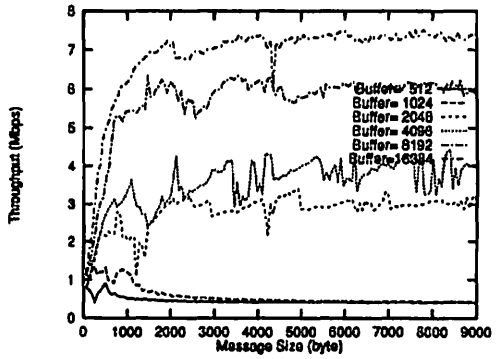

図 8: メッセージサイズとスループット (実装 4)

アプリケーションのメッセージサイズ (byte) とスループット (Mbps) の関係の測定結果を実装ごとに図 4〜9に示す. 測定には Netperf を利用した. 図 4 を見ると実装 1 ではメッセージサイズが 1024byte 以下の場合, スループット特性に極端な不連属性が見られる. しかし, この不連続性の問題は他の実装で

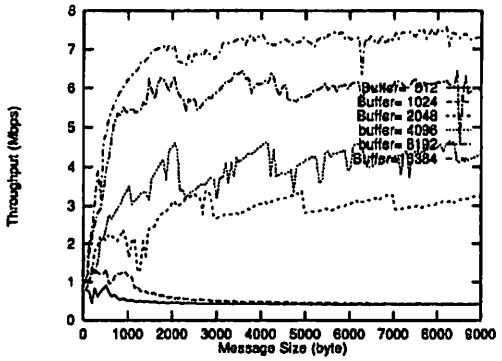

図 9: メッセージサイズとスループット (実装 5)

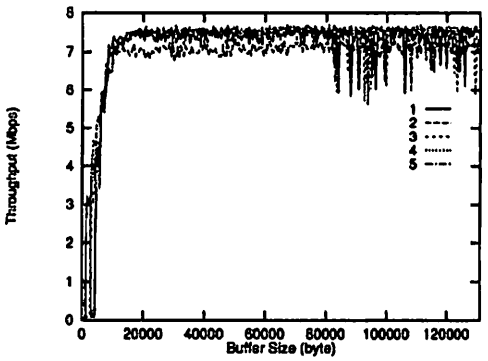

図 10: ウィンドウサイズによる影響 (スループット)

はすべて解消されている．また図5を見るとバッファが 512byte と 1024byte のときにはスループットが 0.01Mbps 程度しかでていない．また，2048byte と 4096byte のときはメッセージサイズが大きくなるにつれてスループットが 0.1Mbps 程度と 3Mbps 程度の間を周期的変動している．これらの問題は実装 2，実装 4，実装 5 ですべて解消されている．実装 3 では 512byte と 1024byte の問題のみ解消されている．

8.4 ウィンドウサイズとスループットの関係

ウィンドウサイズとスループットの関係を図10に，送信データ 1MSS あたりの送信パケット数の割合を図11に，送信データ 1MSS あたり確認応答数の割合を図12に示す．測定には Netperf を利用した．図10を見ると，バッファサイズが 10000～80000byte のとき実装 2 は他に比べてスループットが低い．図11を見ると送信セグメントの割合はどの実装もほとんど同じ結果だが，図12の確認応答の割合には違いが見られる．実装 2 は約 1 なのにたいし他の実装では

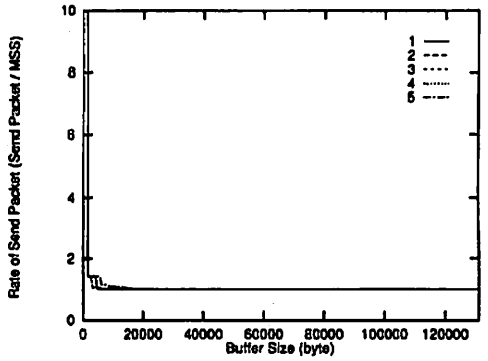

図 11: ウィンドウサイズによる影響 (送信パケット数/MSS)

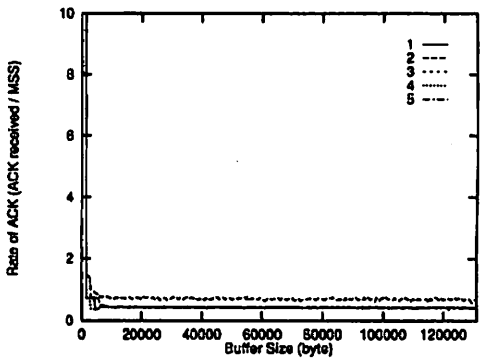

図 12: ウィンドウサイズによる影響 (確認応答数/MSS)

0.5 程度である．実装 2 は遅延確認応答をしないため確認応答の数が増え，そのオーバヘッドのためにスループットが低下していると考えられる．また，バッファサイズが 80000byte 以上の場合，実装 3，5 でスループットのふらつきが見られる．これは TCP にバッファサイズを動的に変更するアルゴリズムが備わったときに問題となる可能性がある．

9 考察

結果を表8にまとめる．また，この表に telnet による遠隔端末操作時のパケット数の評価も加える．実装 1，4 では遠隔端末の利用時には図13の左側のようにパケットが流れ，実装 2，3，5 では右側のようにパケットが流れる．telnet では小さなパケットしか流れないため，ほとんどのパケットで PUSH ビットが立つ．その結果，実装 3，5 ではデータがピギーバックされず，実装 1，4 に比べてパケットが 4/3 倍

表 8: 評価のまとめ

	1	2	3	4	5
スロースタート		○		○	○
送受信バッファサイズの不整合		○		○	○
バッファサイズが1MSS以下		○	○	○	○
ウィンドウサイズが1MSS以下		○		○	○
TCPとソケットの間のバッファ管理の不整合		○		○	○
送信パケットの効率	○	○	○	○	○
確認応答の効率	○	△	○	○	○
ウィンドウが大きい時のスループット	○	△	○	○	○
遠隔端末利用時のパケット数	○	△	△	○	△

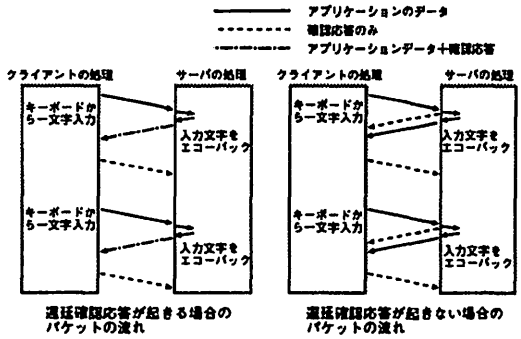

図 13: 遅延確認応答

流れることになる．4/3 倍程度ならば大きな問題とは言えないかも知れない．しかし，品質制御が実現された場合に遠隔端末は低遅延性を要求するため，他のコネクションとの競合を緩和するためにパケットの数は少ない方が望ましい．

総合的な結果は 4 の実装がもっともよい．しかしこの方法は TCP の仕様に反するためこのままでは利用できない．次世代の TCP では DDA(Don't Delay ACK) ビットなどとして制御ビットに付け加えられるべきだと考える．そうすればデッドロックの解決がたやすく実現できる．

デッドロックを解消するだけであれば確認応答の遅延を止める方法もある．ただし，確認応答の数が増えるため，ウィンドウサイズが大きくなったときにスループットの低下を招く．確認応答が増えればネットワークを無駄に使うことにもなる．ただし，悪いことばかりではない．図 4の送信バッファ8192byte, 16384byte，受信バッファ2048byte，4096byte の部分では他の実装よりもスループットが大きい．これは，ウィンドウサイズが小さいときには確認応答を早く返すため送信処理の待ち時間が小さくなっていると推測できる．逆にいえば，ウィンドウサイズが十分大きい場合には，確認応答を適宜減らすことにより，処理やパケットのオーバヘッドを減らすことができる．この結果から，送信データに対する確認応答の割合を動的に変動させることでスループットの向上が可能だと考えられる．

TCP のプロトコルを変更せずにデッドロックを回避する方法としての現実的な解は実装 5 である．これにより，デッドロック問題を解決でき，なおかつ転送効率の低下は小さい．実装 5 は TCP デッドロックを解決でき，副作用が小さいといえる．

10 まとめ

IP がプラグ&プレイを実現しただけではインターネットは使いやすくならない．TCP もさまざまな環境に適合できるようにプラグ&プレイを実現する必要がある．本稿では TCP デッドロック問題について取り上げ，PUSH ビットを拡張して利用することで解決できることを示した．

今後は，ATM などの高速ネットワークや，低速なシリアル回線，高遅延である衛星回線等でもデッドロックが解消されることを確認するとともに，Nagle アルゴリズムを OFF にした場合や，複数のデータストリームがネットワークを共有する場合に問題がないか検証する．そしてプラグ&プレイ TCP を実現するため，ダイナミックなウィンドウ制御，高帯域高遅延環境下でのスロースタートの改善などの課題に取り組む予定である．

参考文献

[1] Christian Huitema, "IPv6: The New Internet Protocol ", Prentice Hall PTR, 1996

[2] 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, 1996

[3] Douglas E. Comer and John C. Lin, "TCP Buffering And Performance Over An ATM Network" Purdue Technical Report CSD-TR 94-26, 1994

[4] John Nagle, "Congestion Control in IP/TCP Internetworks", RFC 896, January 1984

[5] Jon Crowcroft, Ian Wakeman, Zheng Wang, and Dejan Sirovica "Is Layering Harmful?", IEEE Network Magazine, vol.6, pp 20-24, January 1992

[6] J. Postel, "Transmission Control Protocol", RFC 793, September 1981

[7] 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

[8] 村山 公保, 門林 雄基, 山口 英, 山本平一,"多点間接続での TCP 性能評価システム DBS の提案" , 情報処理学会, DPS-95-07, 1996

[9] Hewlett-Packard Company, "Netperf: A Network Performance Benchmark Revision 2.1", 1995