マルチメディア通信と分散処理 80－12
グループウェア 21－12
（1997. 1. 30）

ATMネットワークに適したストリーム転送プロトコル

綾田 和晶†, 西村 浩二‡, 相原 玲二‡

† 広島大学大学院 工学研究科　‡ 広島大学 総合情報処理センター

ATMネットワークにおいてデータ通信を行う場合、転送プロトコルとして主にAAL5が用いられる。信頼性の要求されるデータ通信を行うために現時点ではTCP (Transmission Control Protocol) などの既存のストリーム転送プロトコルを使用することになる。しかし、TCPのアルゴリズムは低速から高速まで様々な物理回線を想定しているため、ATM通信を行う場合には必ずしも最適ではない。本稿では、ATM UBRサービスクラスにおいて高速なストリーム転送が行える、新しいデータ転送プロトコル (ATCP: ATM specific Transmission Control Protocol) を提案する。

A Transmission Control Protocol for ATM Networks

Kazuaki Ayada†, Kouji Nishimura‡, Reiji Aibara‡

† Graduate School of Engineering, Hiroshima University
‡ Information Processing Center, Hiroshima University

AAL5 is a popular adaptation layer protocol for data communication on ATM networks. To perform reliable transmission, connection based byte stream protocols, such as TCP (Transmission Control Protocol), must be used. Because TCP is designed for variety of physical lines, e.g., line speed, it is not the optimum protocol for ATM networks. We propose a new protocol (ATCP: ATM specific Transmission Control Protocol) which allows fast, reliable data communication on ATM UBR service class.

1 はじめに

TCPは、転送レートや遅延時間、またデータの損失や重複、到着順序狂いなどをアプリケーションが考慮しなくても信頼性のあるデータ通信が行えるよう設計された通信プロトコルであり、現在、ストリーム転送プロトコルの標準的な存在として様々な環境で利用されている。

しかし、光ファイバー技術の導入により、TCPがもともと想定していた通信速度を越えたネットワークが登場するようになってくると、確認応答の到着までにネットワークに投入することのできるデータ量（帯域・遅延積）が大きくなるため、ウィンドウサイズに制約があり、損失データに対する回復アルゴリズムも貧弱な現在のTCPでは十分な速度性能が得られない。そこで近年、高帯域、高遅延のネットワーク上でTCPのこの性能ボトルネックを解消しようと様々な改良策が考案されている [1][2][3]。

このような高速なネットワークとしては現在、ATM (Asynchronous Transfer Mode) ネットワークが最も注目されている。ATMネットワーク上でデータ通信を行う場合、通信ホスト間ではあらかじめVC（Virtual Channel；仮想チャネル）を設定する。また、ATMネットワーク上のスイッチにおいてセルの送信順序替えは禁止されており、このため受信側ではデータの到着順序が狂うことはまずないと考えられる。しかし、TCPは信頼性のないIPネットワーク上でデータを順番通りに確実に届けるために構築されたものであり、ATMネットワーク上での使用を考えた場合、TCPのアルゴリズムには無駄な部分が存在する。

また、ATMを用いたデータ通信の方法として、近年、IP層を介さず、ATMのフレーム（AAL5）にTCPのセグメントを直接埋め込む方法も検討されている。この方法ではIPアドレスを使用できないため、利用はATM専用のネットワークに限定され

るが、ルーティング機能を元来持っているATMにルーティングを任せることでIP層のオーバーヘッドを取り除くことができる。実際、IP層を介さず直接ATMにアクセスするTCPをSolaris2.4上に実装したもの（TCP-ONIP; TCP over Non Existent IP）も現在発表されている[4]。

そこで、本稿では、データの順序狂いに対して寛容な従来のTCPを改良し、廃棄セグメントの検知、再送をもっと効率良く行えるような、ATMに適したデータ転送プロトコルを提案する。

本稿では、以降、第2章においてTCPの標準的な輻輳制御方法となりつつある4つのアルゴリズムについて述べ、第3章では最近RFC2018として発表されたTCP SACKオプションのメカニズムおよびその問題点について触れる。また、第4章では本稿で提案するATCPの概要、第5章で各TCPのスループットの実測結果を報告し、終わりにまとめと今後の展望を述べる。

2 4.3BSDにおけるTCPの実装

本章では、4.3BSDにおいて既に実装されており、また、TCPに一般的に導入されつつある4つの輻輳制御アルゴリズムについて簡単に説明する[5]。

2.1 スロースタート、輻輳回避

TCPは輻輳制御を行うためにスロースタートと輻輳回避という2つのモードを持つ（図1）。

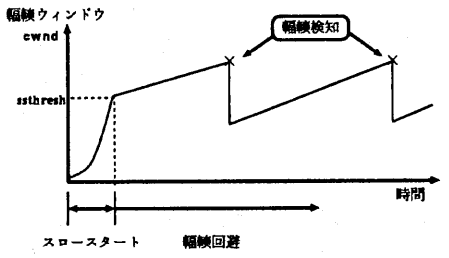

図1: TCPにおける輻輳制御

送信側はまずスロースタート・モードでデータ転送を開始し、輻輳ウィンドウの大きさがある閾値ssthreshに達するまではウィンドウを指数関数的に増やし、閾値に達すると、ウィンドウを線形増加させる輻輳回避モードへと移行する。これは、輻輳ウィンドウがある程度の大きさになるまではデータ投入量を素早く増加させ、閾値を越えると、利用可能な帯域の限界に近付いていると判断して、データ投入量の増加を抑えようとするためである。

2.2 Fast Retransmit/Fast Recovery

TCPでは累積確認応答を用いているため、ACKには常に受信側が次に受け取るべきセグメントのシーケンス番号が記されて返って来る。すなわち、あるセグメントが廃棄された後に受信側に到着したセグメントによるACKには廃棄されたセグメントのシーケンス番号と同じ値が記されている。従って、重複ACKの到着はそのセグメントが廃棄されたことの強い裏付けとなる。そこで、3つの重複ACKが返ってきた場合、Fast Retransmit/Fast Recoveryと呼ばれる次のアルゴリズムを実行する。

(1) 3つの重複ACKが返ってきたら、*ssthresh*を現在の輻輳ウィンドウ*cwnd*の半分に設定し、廃棄されたセグメントを再送する（Fast Retransmit）。そして、*cwnd*を*ssthresh*＋3セグメント（重複ACKの個数）にする。

(2) 以降、重複ACKが到着するごとにcwndを1セグメントずつ増加し、（増加したcwndによって許されるなら）受信側の告知したウィンドウサイズを越えない範囲でまだ送信していないセグメントを送る。これにより、輻輳直前の*cwnd*の半分の量のデータを常にネットワークに投入し続けようとする（Fast Recovery）。

(3) ACKが更新されたら*cwnd*を(1)で設定した*ssthresh*の値にし、これ以降、通常の輻輳回避モードに入る。

しかし、このアルゴリズムは1ウィンドウ内の1セグメントが廃棄された場合に対して最適化されたものであり、複数のセグメントが一度に廃棄されると、それらのデータはタイムアウトするまで再送できないことがある。そこで最近、TCPのこの欠点を解決すべく新しい機構が発表された。次章ではその新しい機構について触れる。

3 TCP SACKオプション

本章では、RFC2018にて提案されているTCP SACKオプションのメカニズムについて触れる[2]。

TCP SACK (Selective Acknowledgement) オプションとは、セグメントが順序通りに受信側に到着しなかった場合に、受信側が、確認応答はしていないが既に受信しているデータブロックの情報をACK セグメントに付加して送信側に送り返すことで、送信側はその情報から、受信側に到着しなかったセグメントだけを選択再送できるようにするものである。これにより、1ウィンドウ内に複数のセグメントが廃棄された場合にも、タイムアウトを待たずに複数の廃棄セグメントを検知、再送できる。

図2に受信側からのACK セグメントに付加されるSACK オプションのフォーマットを示す。図のように、受信したが確認応答していないデータブロックの先頭シーケンス番号および最終シーケンス番号(正確には最終番号の1つ後ろの値)の組を1単位として、最も最近変化したブロックから順にTCP のヘッダ長が許す限りできるだけ多く付加する。付加するブロック数をnとすると、SACK オプションのオプション長は$8\times n+2$バイトとなるが、TCP ヘッダのオプション長の最大値は40バイトであるから$n \leq 4$となる。しかし、SACK オプションは通常、10バイト長のタイムスタンプオプション[1]とともに用いられることが多いと思われるため、実際には$n \leq 3$となる。送信側では、到着した全ての確認応答セグメントのSACK ブロックを記録しておき、Fast Recovery の際には、まだ選択確認応答されていないセグメントのうち既に選択確認応答されているセグメントよりもシーケンス番号の小さなものを再送する。

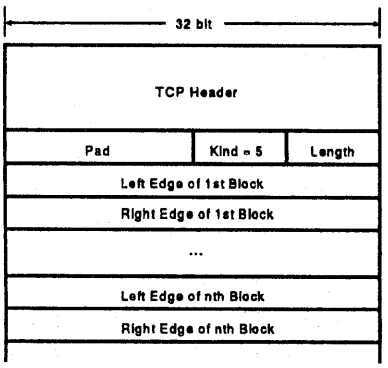

図 2: TCP SACK オプションのフォーマット

しかし、この実装では、選択確認応答されたセグメントより小さなシーケンス番号を持つセグメントを廃棄されたものと見なしているため、再送セグメントが再び廃棄された場合にはそれを検知できず、タイムアウトを待たなければならないという欠点がある。その様子を図3に示す。ただし、セグメント長 (MSS: Maximum Segment Size)、送信ウィンドウサイズはそれぞれ1000byte、15000byte であり、また図中の●、□、△はそれぞれ、送信側の送信したデータセグメントのシーケンス番号、受信側から返ってきた確認応答のACK 番号およびSACK により選択確認応答されたセグメントのシーケンス番号を表すものとする。この図では、シーケンス番号7000〜9000 (○印) の3つのセグメントが連続して廃棄された後、シーケンス番号7000の再送セグメントが再び廃棄された場合の送信側の振舞いが示されている。3個の重複ACK を受け取りFast Retransmit が行われるが、シーケンス番号7000の再送セグメントが再び廃棄されたためにACK が更新されないまま Fast Recovery を続けている (新たなセグメントを送信し続けている)。そして、受信側の告知したウィンドウサイズ分のセグメントを送信し終えた後、タイムアウトによりスロースタートモードでシーケンス番号7000のセグメントが再送されている。

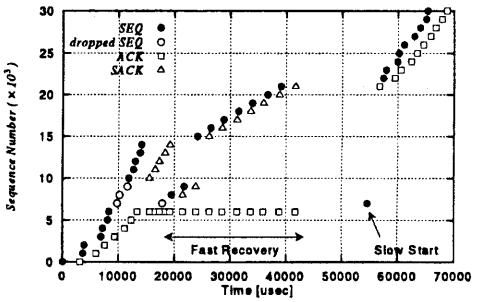

図 3: 再送されたセグメントが再び廃棄された場合のSACK TCP の振舞い

次章において、この欠点をも解消し、廃棄セグメントを受信されるまで何度でも選択再送できる、新しいアルゴリズムを提案する。

4 ATCP プロトコル

4.1 ATCP の概要

ATCP は、ATM に直接アクセスしてデータ通信を行う、ATM ネットワーク用に最適化されたストリーム通信プロトコルである。前述のように、ATM

ネットワーク上ではデータの到着順序は入れ替わらないという特徴を活かして、ATCP では TCP に次のような変更を加えた。

まず、ATCP ではセグメントヘッダに、従来の TCP で用いられるシーケンス番号（SEQ）および確認応答（ACK）のほかに、再送セグメントも含めて送信セグメントごとに 1 ずつ単調増加するセグメント番号（S_SEQ）を付加してデータを送信する。受信側では到着したセグメントごとにその S_SEQ の値を S_ACK フィールドに入れて確認応答を行う。送信側では、従来通り、確認応答セグメントの ACK フィールドが更新されていればウィンドウをスライドさせて新たなセグメントを送信するが、ACK フィールドは更新されなかったが S_ACK フィールドであるセグメントが選択的に確認応答されていれば、S_ACK フィールドの示すセグメント番号よりも小さなセグメント番号を持つデータは全て廃棄されているとわかる。従って、Fast Recovery の際には選択確認応答されなかったものだけを再送できる。また、再送セグメントにも前回の送信時とは別のセグメント番号（S_SEQ）が付けられているため、再送セグメントが廃棄された場合でもそのセグメント番号より大きな値のセグメント番号を持つセグメントが選択確認応答されれば、再送セグメントの廃棄を再び検知でき、受信されるまで何度でも再送可能である。

4.2 ATCP のセグメント・フォーマット

図 4 に ATCP で用いるセグメントヘッダのフォーマットを示す。

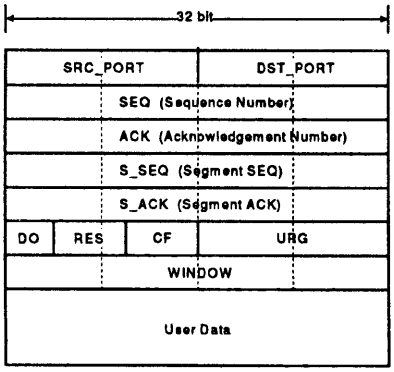

図 4: ATCP のセグメントヘッダ・フォーマット

(1) SRC_PORT (16 ビット)：送信ポート番号。

(2) DST_PORT (16 ビット)：宛先ポート番号。

(3) SEQ (32 ビット)：セグメント・データの最初の 1 バイトの順序番号が入れられる。

(4) ACK (32 ビット)：シーケンス通りに受信した最終バイトの次の順序番号が入れられる。

(5) S_SEQ (32 ビット)：送信セグメントごとに一意的に付けられるセグメント番号。

(6) S_ACK (32 ビット)：受信したセグメント番号（S_SEQ）を確認応答するフィールド。

(7) DO (4 ビット)：TCP のヘッダ長。32 ビットを 1 単位として指定する。

(8) RES (6 ビット)：将来使用する目的のフィールドで、現時点では未定義。0 を代入しておく。

(9) CF (6 ビット)：セグメントの種類を示すフィールドで、TCP と同様のフラグが定義される。

(10) URG (16 ビット)：データの先頭から何バイト目までが緊急データであるかを示す。

(11) WINDOW (32 ビット)：受信バッファの空き領域のバイト数を示す。

4.3 ATCP の輻輳制御・再送メカニズム

ATCP においても、TCP と同様にスロースタート、輻輳回避、Fast Retransmit/Fast Recovery アルゴリズムを実装すべきである。ただし、輻輳時の ACK に対する処理が従来の TCP とは少し異なり、送信側は"sndsseq"、"rcvsack"、"scount"、"maxscount"という新たな 4 つの内部変数を用いて次のような動作をする。ここで、"SEG.ACK"、"SEG.SACK"はそれぞれ確認応答セグメントの ACK、S_ACK フィールドの値であり、送信側の保持する変数"sndsseq"には自分が最も最近送信したデータセグメントに付けられたセグメント番号（S_SEQ）が、また変数"rcvsack"には受信側から送られた確認応答の SEG.SACK の値がそれぞれ記録されるものとする。その他の変数は以下の文中で説明する。

(1) 1 つでも重複 ACK が返ってきたら、直ちに *ssthresh* を現在の *cwnd* の半分の値に設定し、廃棄されたセグメントを再送する（Fast Retransmit）。そして SEG.SACK で示されたセグメントを選択確認応答した後、現在ネットワーク中にあるセグメント数 *sndsseq* − *SEG.SACK* および現在のウィンドウサイズ

の半分 $(sndsseq - rcvsack)/2$ の値をそれぞれ変数 *scount*、*maxscount* に記録する。

(2) それ以降、ACKが到着するごとに、ネットワーク上から消えたセグメント数（SEG.SACK - *rcvsack*）を求め、*scount* からその値を引く。（これにより、現在ネットワーク中に流れているセグメント数が正確に分かる。）*scount* が *maxscount* より小さくなれば、受信側の告知したウィンドウサイズを越えない範囲で選択確認応答されていないセグメントおよび新たなセグメントを最大 $maxscount - scount$ 個まで送信し、輻輳直前にネットワーク中に流れていたデータの半分の量を常に流し続ける。

(3) SEG.ACK 番号が輻輳検知直前に送信していたシーケンス番号を越えたら、*cwnd* を (1) で設定した *ssthresh* の値にする。今、輻輳ウィンドウ *cwnd* は閾値 *ssthresh* に等しいので、これ以降、通常の輻輳回避モードに入る。

図5に、図3と同じ条件下でのATCPの振舞いを示す。この図より、再送セグメントが再び廃棄された場合でも、ATCPではACKが返ってくる限りタイムアウトを待たずに廃棄セグメントの再送を何度でも行えることが分かる。

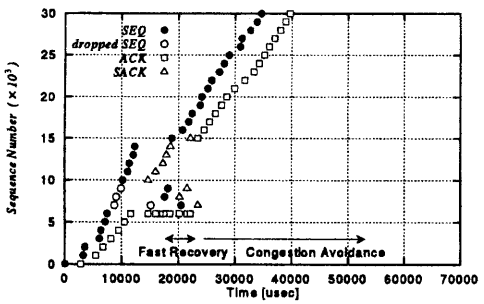

図 5: 再送されたセグメントが再び廃棄された場合のATCPの振舞い

4.4 その他の特徴

ATCPでは、受信されたデータセグメントが全て個別に選択確認応答されるため、これを利用すれば、タイムスタンプ機能を実現できる。すなわち、送信側がセグメントごとにその送信時刻を保持しておくことで、確認応答を受信する度に、現在時刻と選択確認応答されたセグメントの送信時刻からラウンドトリップ時間を計測することができる。

また、ATCPではヘッダのチェックサムフィールドを削除している。ATCPはAAL5フレームを利用しており、AAL5のチェックサム機能によりデータの妥当性は保証されているため、TCPでのチェックサム計算は冗長であると思われる。

さらに、ウィンドウサイズを表示するフィールドを32ビット長に拡大した。これにより、従来のTCPでボトルネックになっていたウィンドウサイズの制約が取り除ける。

5 実験

5.1 実験に用いた環境

SunATM™-155 SBus Card を搭載した2台のSPARC station 10（CPU clock：50MHz）をATMスイッチに接続し、今回アプリケーションとして試作したATCPのスループットを実測した。

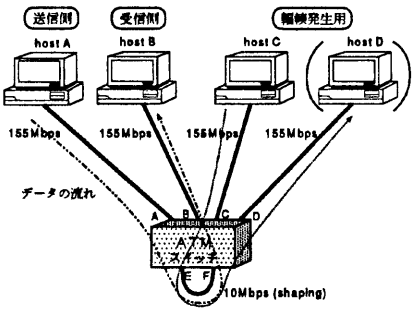

図 6: 実験ネットワーク構成

今回の実験では、遅延時間の小さなATM-LANでのデータ通信を想定して、図6に示すように通信ホスト間（A-B間）にATMスイッチを1台接続し、A-E-F-Bというポート経路でPVCを張ってAからBにデータを送信した。また、ネットワークがある程度の輻輳状態にある環境を作るために、ポートE-F間に10Mbpsのシェーピングを設定し、さらにスイッチにもう1台ATM端末（C）を接続してポートC-E-F-Dに別のPVCを張り、CからDに時々セルを送信することで、ポートE-F間で輻輳を起こさせるようにした。

5.2 実験結果

実験ではまず、ネットワークに一切負荷をかけず、155Mbpsのネットワーク帯域を全て使える状

態で、作成した各 TCP のスループットを計測した。ウィンドウサイズは 64Kbyte、最大セグメント長は 9Kbyte に設定した。結果を図 7に示す。図には Solaris 2.4 用に実装された TCP-ONIP（TCP Over Non-existent IP）[4] および Solaris 2.4 において Classical IP over ATM を用いた従来の TCP のスループットも示してある。この図より、IP 層を除去して ATM に直接アクセスすると速度性能が向上することが分かる。

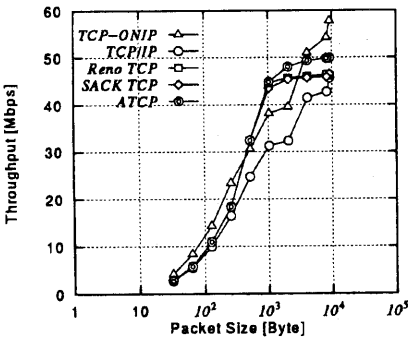

図 7: スループット測定結果

次に、ポート E-F 間に 10Mbps のシェーピングを設定し、さらにネットワークにあらかじめ負荷をかけた状態でスループットを測定した。送信データの最大セグメント長は 9Kbyte とし、また、負荷トラフィックは 64Kbyte の AAL5 フレームを連続して 3 回送信した後、一定時間休むというパターンを用いた。結果を図 8に示す。

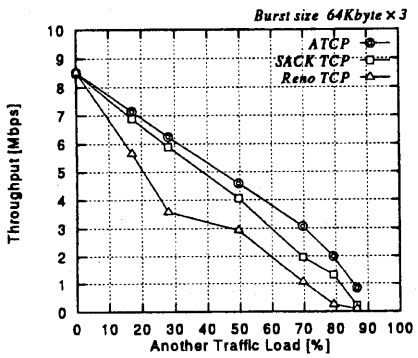

図 8: スループット測定結果（輻輳時）

図より、バースト性の高いトラフィックが他に存在する場合、Reno-TCP ではスループットがかなり低下することが分かる。これは、バーストトラフィックによって 1 ウィンドウ内の複数のセグメントが一度に破棄されたため、タイムアウトを頻繁に繰り返しているものと思われる。また、ATCP と SACK TCP において、ネットワーク負荷が大きくなればなるほど ATCP の方が良い結果が得られているが、これは頻繁に発生するバーストトラフィックにより再送セグメントが再び廃棄されてしまう確率が高くなっており、確認応答が返ってくる限り廃棄セグメントを何度でも再送できる ATCP が有利に働いているためだと思われる。

6 まとめ

本稿では、ATM ネットワーク用に最適化されたストリーム転送プロトコルを提案し、それを実装した。IP 層を取り除くことによりデータ転送効率は向上した。また、損失セグメントに対する回復アルゴリズムは従来の TCP よりも ATCP の方が優れていることが分かった。今後は、高帯域・高遅延の OLU ネットワーク上で実験を行い、高遅延環境で ATCP がどの程度スループットを維持できるか測定する予定である。

参考文献

[1] D.Borman, R.Braden, and V.Jacobson. "*TCP Extensions for High Performance*", RFC1323, Internet Engineering Task Force, May 1992.

[2] Matthew Mathis, Jamshid Mahdavi, Sally Floyd, and Allyn Romanow. "*TCP Selective Acknowledgement Options*", RFC2018, Internet Engineering Task Force, October 1996.

[3] 釘本 健司、天海 良治、村上 健一郎. "絶対帯域見積りに基づく輻輳回避アルゴリズム", 情報処理学会 マルチメディア通信と分散処理ワークショップ, pp.245–pp.252, October 1996.

[4] H. Afifi, D. Bonjour, O. Elloumi. "*TCP over Non Existent IP for ATM Networks.*", (Available via ftp://ftp.rennes.enst-bretagne.fr /pub/reseau/afifi/tcp-onipv0.9.tar.)

[5] W. Richard Stevens. "*TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms*", draft-stevens-tcpca-spec-01.txt, Internet-Draft. March 1996.