マルチメディア通信と分散処理　62－15
（1993. 9. 21）

信頼性のあるデータグラム通信機構

山本 学

日本アイ・ビー・エム（株）東京基礎研究所
〒２４２ 神奈川県大和市下鶴間１６２３－１４

あらまし

サーバー・クライアントモデルの利用形態の中に１つのサーバープログラムが多くのクライアントプログラムにサービスを提供するものがある。このようなアプリケーションには信頼性のあるデータグラム通信プロトコルが必要である。本稿では、この要求を満たす通信プロトコルとして、reliable User Datagram Protocol （rUDP）を定め、それをTCP上で実現するための実装手法を述べる。この実現手法の重要点はTCPのコネクションの隠蔽と流量制御である。最後にその手法に基づいて実装したプロトタイプでの実験結果について述べる。

和文キーワード　サーバークライアントモデル, データグラム通信, TCP, rUDP, 通信プロトコル

Implementation of reliable User Datagram Protocol (rUDP)

Gaku Yamamoto

Tokyo Research Laboratory IBM Japan Ltd.
1623-14 Shimotsuruma Yamato-Shi Kanagawa-Ken JAPAN

Abstract

There are several types of server-client model. One of the model is that one server program gives services to many client programs. Such appliation needs reliable datagram transmission protocol. In this paper, we show you the implementation methods of reliable User Datagram Protocol (rUDP) which is one of the reliable datagram transmission protocol. Our approach is to implement rUDP based on TCP. Key points of the method are implicit connection and flow control. Finaly we describe the results of our experiments of rUDP which we implemented.

英文 key words　Server-Client Model, Datagram Transmission, TCP, rUDP, Transmission Protocol

信頼性のあるデータグラム通信機構

山本 学

日本アイ・ビー・エム株式会社 東京基礎研究所

1 はじめに

現在、サーバー・クライアントモデルはビジネス環境に普及しはじめ、様々なサーバープログラムが多くのクライアントプログラムにサービスを提供する環境が作られつつある。ここで、サーバープログラムとクライアントプログラムの関係をその利用形態で分類すると次のようになる。

1. サーバープログラムは一つまたは少数のクライアントプログラムにサービスを提供する。クライアントプログラムはサーバープログラムを長時間ほぼ占有する。クライアントプログラムからの要求、サーバープログラムからの返事は重要であり、配送中に紛失、重複してはならない。
2. サーバープログラムは多くのクライアントプログラムにサービスを提供する。それぞれのクライアントプログラムはサーバープログラムを長時間占有することはない。クライアントプログラムからの要求、サーバープログラムからの返事は重要ではなく、配送中に紛失、重複することは許される。
3. サーバープログラムは多くのクライアントプログラムにサービスを提供する。それぞれのクライアントプログラムはサーバープログラムを長時間占有することはない。クライアントプログラムからの要求、サーバープログラムからの返事は重要であり、配送中に紛失、重複してはならない。

1の形態は、一つのクライアントが一つのサーバーを占有し、サーバープログラムとクライアントプログラムの間で会話が行なわれる。rlogin や telnet はこの形態である。SUN-RPC、DCE-RPC のコネクションベース RPC はこのような利用方法を想定している。

2の形態は、要求が処理されなくても問題が起こらない場合に利用される。SUN-RPC、DCE-RPC のデータグラムベース RPC はこのような利用方法を想定している。

3の形態は、一つのサーバープログラムが多くのクライアントプログラムからの要求を処理する。ただし、一つのクライアントプログラムがサーバープログラムを長時間占有して利用するのではなく、クライアントプログラムは、一つの要求を処理する間だけサーバープログラムを占有する。例として、データーベースサーバープログラムがある。

これらの利用形態は、それぞれを実現するために利用する通信プロトコルと密接な関係がある。例えば、1の形態では、TCP のようなコネクションベースの通信プロトコルを利用され、2の形態では UDP のようなデータの到着を保証しないコネクションレスな通信プロトコルが利用される。

しかしながら、3の形態に関しては、現在適当な通信プロトコルがなく、個々のアプリケーション開発者が独自の工夫をして実現している。本稿では、この3の利用形態に焦点を当て、アプリケーションレベルで利用する、3の形態に適した通信プロトコルを TCP 上で実現する手法を述べる。

このような通信プロトコルを TCP 上で実現する利点は、第一にポータビリティが良いこと、第二にデータの配送を保証しているために、データ配送の保証に関しては考えなくて良いことである。欠点としては、TCP 上で実現するためにオーバーヘッドがあることである。これらは、ポータビリティとパフォーマンスのトレードオフであるが、アプリケーションレベルで利用することを仮定するので、ポータビリティを優先する。

2 通信プロトコルの仕様

3の利用形態で要求される通信機能は、

- データ配送中にデータが紛失、重複しないこと。
- 一つのプログラムは、いつでも複数のプログラムからのデータを受信できること。
- データの境界を保存すること。

である。データ境界を保存するという要求は、この利用形態は rlogin や telnet のようにキャラクタをストリームで送信する利用方法ではなく、ある構造を持った要求をサーバープログラムに送信するという形態であるために必要な要求である。

本稿では、これらの要求を満たす通信プロトコルの仕様を以下のように定める。

1. 送信プログラムがデータ送信に成功した場合は、重複することなしに確実に受信プログラムで受信される。受信されない場合は、送信プログラムでエラー検出される。(信頼性)
2. アプリケーションレベルでは、プログラム間で開設されるコネクションが存在しない。(コネクションレス通信)
3. データの境界を保存する。(データグラム通信)
4. データ転送時に受信プログラムの資源がいっぱいになった場合、送信プログラムはブロックする。(流量制御)

流量制御に関する仕様は、プログラムの限られた受信バッファを考慮し効率の良いデータ転送を行なうために必要な仕様である。

本稿では、以降この仕様を満たす通信プロトコルを rUDP (reliable User Datagram Protocol) と呼ぶこととする。

3 rUDP 実装上の問題点と解決策

3.1 問題点

rUDP を実装する時の重要な問題点は、信頼性の保証と流量制御である。

我々のアプローチは TCP 上に rUDP を実装するアプローチである。したがって、二つのプログラム間の通信に

Implementation of reliable User Datagram Protocol(rUDP)
Gaku YAMAMOTO
Tokyo Research Lab. IBM Japan Ltd.

おける信頼性はTCPが保証する。しかし、TCPはコネクションベースであり、二つのプログラム間でコネクションを開設する。これに対し、rUDPはコネクションレスであるので、TCPのコネクションを隠蔽し、自動的に開設、閉鎖する仕組みが必要となる。

流量制御は、プログラムの受信バッファの残領域を考慮し、効率の良いデータ転送を行なうために必要である。TCPはデータ転送を行なう二つのプログラム間の流量制御を行なう。しかし、rUDPでは、一つのプログラムが多くのプログラムからデータを受信する。したがって、TCPが行なう二つのプログラム間の流量制御だけではなく、プログラムの全体の受信バッファの領域を考慮した流量制御が必要となる。

この流量制御とコネクションの隠蔽は密接な関係がある。例えば、TCPのコネクションを隠蔽する方法として最も簡単なものは、あるプログラムがあるプログラムにはじめてデータ転送を行なう時にTCPのコネクションを開設し、どちらか一方のプログラムが通信システムの使用終了APIを発行した時にコネクションの閉鎖を行なう方法である。

しかし、この方法では、多くのプログラムと通信を行なうプログラム（例えば、サーバープログラム）は多くのコネクションを持ち続けることになる。TCPでは一対のコネクション毎にそれを管理するための資源が確保され、コネクションが開設されている間は通信が行なわれているかに関わらず、その資源は存在し続ける。また、データ送信側のプログラムは、コネクションが確立されればデータを転送するが、受信側のプログラムはすぐにデータ受信APIを発行するとは限らない。受信処理をされないデータはTCPの受信バッファ内に蓄積されることになる。TCPでは、流量制御を行ない、一つのコネクションで費やされる受信バッファの量を規制しているが、rUDPでは一つのプログラムが多くのプログラムとの間で、コネクションを開設する。その結果、一つのプログラム内で、多くの受信バッファが消費されることが起こる。これは、サーバープログラムが一つの要求を処理する時間よりも短い時間間隔で、複数のクライアントプログラムから要求が発行された場合、破局的である。したがって、この方法では、流量制御を行なうことはできない。

一つのデータを送信する毎にコネクションを開設し、そのデータ送信が終了したらコネクションを閉鎖する方法は、ある時刻でのコネクションの開設数は最大で1なので、流量制御を行なうことはできる。しかし、コネクションの開設、閉鎖のオーバーヘッドが大きく、パフォーマンスが極端に悪くなり、rUDPの重要な機能の一つである多くのプログラムとデータ転送を行なうと言う仕様を十分に満たすことができない。

3.2 解決策

この問題の解決策として、我々は次の方法をとる。同時に開設できるコネクションの最大数を制限し、その範囲内で複数開設することにする。すでにコネクションの開設数が最大数に達し、それ以上の開設要求が来たら、開設済みのコネクションの中から適当なものを選択し、それを閉鎖し、新しい開設要求を受理する。この方法により、開設されたコネクションの数はある値以下になるので、TCPのコネクションの資源と受信バッファが無制限に使用されることを防ぐことができ、多数のコネクションを考慮した全体の流量制御を行なうことができる。

また、ひとたび開設されたコネクションはしばらくの間キャッシュし、コネクションの閉鎖の必要性が生じるまで維持する。これにより、コネクションの開設、閉鎖のオーバーヘッドを小さくすることができる。

4 コネクションの隠蔽と流量制御

4.1 コネクション隠蔽の仕組み

コネクション数の制限とコネクションのキャッシュの仕組みを実現するための重要な点は、

1. コネクションの制限数を幾つにするか。
2. いつコネクションを閉鎖するか。
3. どのコネクションを閉鎖するか。
4. どのようにコネクションを開設、閉鎖するか。

である。1の問題は、パフォーマンスに大きな影響を与える。例えば、頻繁に多くのクライアントプログラムから利用されるサーバープログラムでは、コネクションの制限数が少ないと、頻繁にコネクションの開設、閉鎖が起こり、パフォーマンスが極端に悪くなる。これに対し、利用頻度の少ないサーバープログラムでは、コネクションの制限数は少なくて良い。これは、アプリケーションの性質に依存するものである。したがって、この制限数はアプリケーション開発者が決定すべきである。

コネクションの閉鎖は、コネクション開設、閉鎖のオーバーヘッドを減少させることから考えると、ひとたび開設されたコネクションはできる限り維持した方が良い。この考えからすると、コネクションを開設可能な残量が0になり、それ以上のコネクション開設の必要が生じた時に開設済みのコネクションのどれかを閉鎖することになる。しかし、この方法では、新たにコネクションを開設するときに、一旦どれかのコネクションを閉鎖し、それが完了してから新しいコネクション開設処理を行なうことになり、新しいコネクション開設に時間がかかる。したがって、コネクションの閉鎖は、コネクション開設可能な残量が少なくなってきたら、適当なコネクションを閉鎖するようにし、常に残量があるようにしておくべきである。

次に、どのコネクションを閉鎖すべきかであるが、この問題もパフォーマンスに大きな影響を与える。一つの方法としては、以前に使用してから最も時間が経過しているものを閉鎖すべきコネクションとして選択する方法がある。しかし、これも1の問題同様アプリケーションの性質に依存する。例えば、サーバープログラムに対する要求のあるものは、クライアントプログラムとの会話的な処理を必要とし、あるものは一回の要求の転送だけで終了してしまうような場合、単純に時間だけで決定できない。この問題に関する検討は今後解決すべき問題として、本稿ではこれ以上言及しない。

最後にコネクション開設、閉鎖の方法であるが、これに関しては次に述べる。

4.2 コネクション開設方法

コネクションの開設は簡単である。データの送信側プログラムからTCPのコネクション開設要求を発行すれば良い。ただし、送信側プログラム、受信側プログラムで新

たなコネクションを開設できるかをチェックする必要がある。これは次の手順で行なえば良い。

[送信側プログラム]

1. すでに受信側プログラムとの間で開設済みのコネクションがあれば、それを利用する。なければ次へ進む。
2. コネクション開設可能な残量が0であれば、適当な開設済みコネクションに対し閉鎖処理を行なう。
3. TCPのコネクション開設要求を発行し、受信側プログラムからの返事を待つ。
4. 受信プログラム側からの返事が開設確認ならば開設処理を終了する。
5. 受信プログラム側からの返事が開設待ちならば、適当な時間待ってから3の処理から再度始める。

[受信側プログラム]

1. TCPのコネクション開設要求を受信したら、TCPのコネクション受理処理を行ない、コネクションを開設する。
2. コネクション開設可能な残量が0でなければコネクション開設確認を送信側プログラムに返送する。0であればコネクション開設待ちを返送する。この場合、TCPのコネクション閉鎖処理を行ない、一度開設したコネクションを閉鎖する。
3. コネクション開設可能な残量が少なければ、開設済みの適当なコネクションに対し閉鎖処理を行なう。

この手順は簡単のため、2つのプログラム間で互いにコネクション開設要求を発行した場合の処理を含めていない。以下にこの処理も含めたコネクション開設処理の状態遷移図を示す。

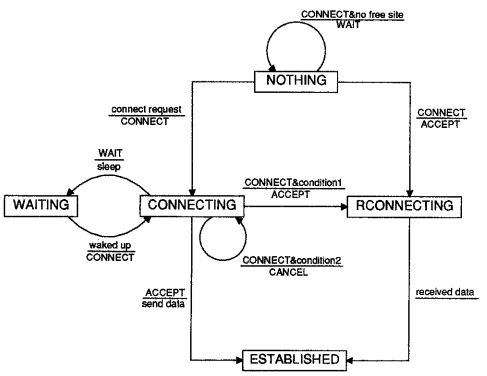

図1 コネクション開設処理の状態遷移図

NOTHING	コネクションが開設されていない状態
CONNECTING	コネクション開設要求を発行し、応答を待っている状態
RCONNECTING	コネクション開設要求を受信し、データを待っている状態
WAITING	コネクション開設要求がすぐに受理されず、しばらく待っている状態
ESTABLISHED	コネクション開設中状態

condition1は、互いにコネクション開設要求を発行した場合、自分を開設要求を受信した状態に遷移させる条件である。condition2はその逆の条件である。

4.3 コネクションの閉鎖

コネクションの閉鎖処理で重要なことは、TCPのコネクション閉鎖処理を発行するときに、TCPの受信バッファ内に未受信のデータを残してはならないことである。TCPは、コネクションが閉鎖された場合、TCPの受信バッファ内のデータを全て廃棄する。このようなデータは、送信側のプログラムでは送信は完了しているが、受信側のプログラムで廃棄されてしまうため、データを紛失することになり、信頼性の保証の仕様を満たさなくなる。このために、2つのプログラム間での同期処理と未受信データの受信処理が必要である。未受信データの受信に関しては第5章で詳しく述べるので、ここでは2つのプログラム間の同期に関して述べる。

受信バッファ内のデータの廃棄を防ぐため、プログラムがあるコネクションを閉鎖する場合、2つのことを確認する必要がある。一つは自分が送信したデータが相手プログラムの受信バッファから全て読み込まれたことの確認、一つは相手プログラムが送信したデータが受信バッファから全て読み込まれたことの確認である。これは次の手順で行なう。

閉鎖要求送信側プログラム

1. 閉鎖要求を送信し、返事が来るまで待つ。この間、このコネクションは利用してはならない。
2. 閉鎖確認の返事をTCPの受信バッファから読み込んだら、再度閉鎖要求受信プログラムに閉鎖確認を送信し、TCPのコネクションを閉鎖して終了する。

閉鎖要求受信側プログラム

1. 閉鎖要求をTCPの受信バッファから読み込んだら閉鎖確認を送信する。
2. 閉鎖要求送信側プログラムから閉鎖確認を受信するか、TCPのコネクションが閉鎖されるまで待つ。
3. TCPのコネクション閉鎖処理をして終了する。

この手順で、閉鎖要求送信側プログラムが閉鎖確認の返事を受信した後で再度閉鎖確認を送信するが、閉鎖要求受信側プログラムが閉鎖確認を送信した直後にTCPのコネクションを閉鎖してしまうと、閉鎖確認が閉鎖要求送信側プログラムのTCP受信バッファ内で廃棄されてしまうため、これを防ぐために必要である。

この手順では、互いに同時にコネクション閉鎖要求を発行した場合の処理を含めていない。以下にこの処理も含めたコネクション閉鎖処理の状態遷移図を示す。

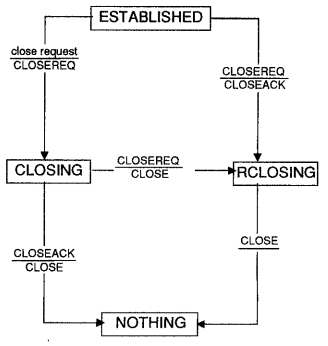

図2 コネクション閉鎖処理の状態遷移図

CLOSING	コネクション閉鎖要求を発行し、応答を待っている状態
RCLOSING	コネクション閉鎖要求を受信し、次の閉鎖確認を待っている状態

5 中間バッファとその制御

5.1 中間バッファの導入

rUDPでは、上位アプリケーションからコネクションを隠蔽するため、コネクションは必要な時に上位アプリケーションに関知されることなく開設され、閉鎖される。コネクションを閉鎖する場合は、TCPの受信バッファ内から到着済みのデータを読み込んでおかなければならない。しかし、上位アプリケーションはデータがいつ到着したかに関係なく、受信処理を行ない、到着済みのデータを上位アプリケーション領域に読み込む。上位アプリケーションの受信処理に合わせてコネクションの閉鎖処理を行なうのでは、効率の良いコネクションの開設、閉鎖を行なうことはできず、また、送信側プログラムは受信側の上位アプリケーションが到着データを読み込むまでブロックする可能性がある。

この問題を解決するために、上位アプリケーションの領域とTCPの受信バッファの間に中間バッファを設ける。TCPの受信バッファに到着したデータを、すみやかにこの中間バッファに読み込み、TCPの受信バッファをなるべく空の状態にする。アプリケーションは必要な時にこの中間バッファから上位アプリケーション領域にデータをコピーする。中間バッファを利用することにより、上位アプリケーションの動作と非同期に送信されてくるデータをすみやかにTCPの受信バッファから読み込むことができ、効率の良いコネクションの開設、閉鎖を行なうことができる。その結果、送信側プログラムがブロックすることを防ぐことができる。

それでは、この中間バッファをどのように管理すべきであろうか。一つの方法として、データがTCPの受信バッファに到着したら、そのデータを格納できる大きさのメモリを確保することが考えられる。しかし、このような方法では、中間バッファが無制限に確保されることが容易に起こりうる。また、動的にメモリを確保するとオーバーヘッドが大きくなる。

そこで、あらかじめ適当な大きさの中間バッファを確保しておき、それをうまく切り分けて使い、再利用する方法をとる。この方法では、中間バッファがいっぱいになった場合、空き領域ができるまでTCPの受信バッファから読み込むことができなくなる。最終的にデータ送信側プログラムはブロックする。しかし、これは受信側プログラムの資源の利用量を制御する流量制御である。また、中間バッファの大きさを適切なサイズにしておけばなるべく防ぐことができる。

中間バッファの大きさは、アプリケーションの性質に大きく依存する。したがって、アプリケーション開発者が中間バッファの大きさを指定できるようにしておくべきである。

5.2 中間バッファの利用方法

本節では、中間バッファの確保と再利用の方法について述べる。中間バッファの利用方法と、TCP受信バッファからの読み込み方法、上位アプリケーションの中間バッファからの読み込み方法は密接な関係がある。まず始めにTCPの受信バッファからの読み込み手順と上位アプリケーションの中間バッファからの読み込み手順を述べる。

TCPの受信バッファからの読み込み

1. TCPの受信バッファに到着したデータを格納するバッファを中間バッファから適当な大きさだけ切りだす。
2. TCPの受信バッファからデータを切り出されたバッファに読み込む。
3. 切り出されたバッファを受信データキューの最後尾につなぐ。

上位アプリケーションの中間バッファの読み込み

1. 上位アプリケーションは、受信データキューの先頭からデータをコピーする。
2. コピーが終了したバッファは受信データキューから切りはなし、中間バッファの空き領域とする。

この方法では、中間バッファから切り出された領域は、切り出された順に、受信データキューにつながれていく。したがって、上位アプリケーションに読み込まれる順も切り出された順となり、中間バッファに返される順も同じである。結果として、中間バッファは先頭から順に使われ、使われた順に返されていく。この特徴を考慮した中間バッファからのデータ格納バッファの確保は次のように行なう。

バッファの確保

1. 中間バッファの使用中領域の長さを参照し、空き領域があるかを確認する。空き領域がなければ処理は行なわない。
2. 切り出すバッファの大きさを計算する。この大きさは適当なサイズを1ブロックとしたブロック単位で切り出す。もし、切り出すバッファが中間バッファの再後尾か、使用中領域の先頭を越えてしまう場合は、メモリの連続領域として確保できるまでを切り出す。
3. 中間バッファの使用中領域の直後（ただし、中間バッファの再後尾の場合は、中間バッファの先頭）から計算した大きさだけ確保する。
4. 中間バッファの使用中領域の長さに切り出したバッファの大きさを加算する。

切り出されたバッファの返却は次のように行なう。

バッファの返却

1. 中間バッファの使用中領域の先頭を返却するバッファの直後（ただし、中間バッファの再後尾の場合は、中間バッファの先頭）にする。
2. 使用中領域の長さから返却するバッファの長さを減算する。

ここで、注意すべき点は中間バッファの大きさが不適切で、その大きさに比べ、一つのデータがあまりに大きく、中間バッファを占領してしまう場合が起こり得ることである。これが起きると、そのデータが上位アプリケーションに読み込まれるまで、たとえ小さなデータでも他のコネクションのTCP受信バッファ内のデータを読み込むことができず、結果として、効率の良いコネクションの開設、閉鎖が行なえなくなる。そこで、TCP受信バッファから一度に読み込むデータの最大量を制限し、TCP受信バッファ内に残されたデータの格納は、その時点で到着してい

る他のコネクションのTCP受信バッファ内のデータを格納してから行なう。このデータは前に格納されたデータのブロックとリスト状につなぐ。格納バッファを返却する時は、リスト状につながっているデータブロックを全て返却する。これにより、一つのデータが中間バッファを占領することは無くなる。

この方法を行なうと二つの問題が生じる。一つはデータ全てがTCP受信バッファから中間バッファに格納されていないにもかかわらず、上位アプリケーションがデータを読み込むこと、一つは中間バッファからバッファを確保する順と中間バッファに返却する順が変わることである。前者に関しては、TCP受信バッファに残されたデータを上位アプリケーションの領域に直接コピーすれば良い。後者の問題は、大きな問題にはならない。我々の中間バッファの利用方法では、中間バッファの使用中領域の先頭の領域と共に中間バッファの中に位置する領域が返却されることになる。これは、使用中領域が先に返却されるだけで、中間バッファから連続する領域を確保する障害にはならない。

図3は、本節で述べた中間バッファの利用方法を示している。

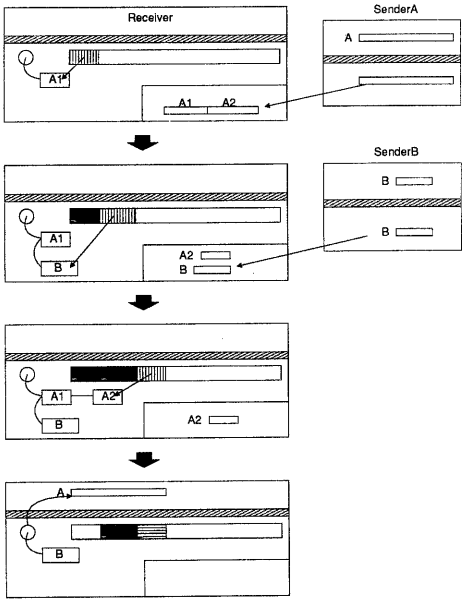

図3 中間バッファの利用法

ここでは、プログラムSenderAがデータAを送信し、プログラムSenderBがデータBを送信する。データAは大きいので、A1、A2に分割してTCP受信バッファから読み込まれる。A1、B、A2はこの順で中間バッファから領域が確保され、受信データキューに接続される。データAが上位アプリケーションに読み込まれた時にA1の領域はすぐに中間バッファに返され、中間バッファの使用中領域の先頭位置が更新される。A2は空き領域マークされる。データBが上位インタフェースに読みとられ、Bの領域が返される時にA2の領域も共に返され、中間バッファの使用中領域の先頭はA2の直後にセットされる。

6 実装とその動作観測

我々は、今までに述べた実装手法に基づきrUDPをOS/2バージョン2.0上でプロトタイプを実装した。今回は、本稿で述べた機構の確認のために実装したので、実装を行ない易い環境をもつOS/2を選択し、OS/2 TCP/IPパッケージライブラリを用い、ユーザライブラリとして実装した。rUDPでは、非同期で送信されてくるデータをTCP受信バッファから中間バッファに読み込む処理を上位アプリケーションの動作と独立して行なわなければならない。OS/2は、マルチスレッド、セマフォなどのサービスを提供しているため、このような非同期処理を行なうプログラムを比較的容易に実現できる。この実装では、開設中のコネクションからどのコネクションを閉鎖するかを選択するスケジュールとして、以前に使用されてから最も時間が経過しているコネクションを閉鎖する方法を用いた。このライブラリでは、コネクションの最大開設数と中間バッファの大きさはアプリケーション開発者が指定できる。また、アプリケーション開発者はrUDPライブラリが自動的にコネクションの閉鎖を行なうきっかけとなるコネクション開設可能な残量（最小未開設コネクション数）も指定できる。

我々は、このライブラリを用いて、簡単なテストアプリケーションを組んで、動作を観測した。

このアプリケーションは、サーバープログラムとクライアントプログラム間でデータ転送を行なうもので、クライアントプログラムが1Kバイトのデータをある時間毎に繰り返しサーバープログラムに転送する。サーバープログラムはそのデータを受信し、同じデータを返送するというものである。サーバープログラム、クライアントプログラムの中間バッファの大きさは10Kバイトであり、コネクションの最大開設数は5とした。最小未開設コネクション数は1以下になったとき、開設されているコネクションから適当なものを選択して閉鎖するように設定した。このテストでは、クライアントプログラムを複数起動し、それぞれのプログラムが独立に一つのサーバープログラムに対し、データを転送する。テストプラットフォームは、複数台の80386を搭載するPS/55を16Mbpsのトークンリング LANで接続したものである。このLANは通常業務で使用されいるが、トークンリングLANのパフォーマンスが低下する程ではない。

この実験では、以下のことが観測された。クライアントプログラムのデータ送信間隔が十分に長い時は、クライアントプログラムの数に関係なく問題なく動作した。しかし、クライアントプログラムのデータ送信間隔を1秒として、このアプリケーションを起動すると、クライアントプログラムが5個以下の場合は順調に動作するが、6個目のクライアントプログラムはほとんどデータを送信できなくなった。次にデータ送信間隔を1.5秒とすると6個のクライアントプログラムまで順調に動作し、データ送信間隔を3秒とすると8個まで順調動作した。次に、サーバープログラムのコネクション最大開設数を10とし、クライアントプログラムのデータ送信間隔を1秒とすると11個のクライアントプログラムまで対応できた。

これらの結果から、サーバープログラムのコネクション

の最大開設数とクライアントプログラム数、それらが送信するデータの転送頻度は密接な関係があると言える。この実験での結果をまとめると以下のようになる。

- データ送信の頻度が小さければ、サーバープログラムが対応できるクライアントプログラム数に対する、コネクションの最大開設数の影響は少ない。
- データ送信の頻度が高くなると、サーバープログラムが対応できるクライアントプログラム数に対する、コネクションの最大開設数の影響は大きい。
- サーバープログラムが対応できるクライアントプログラム数は、データ送信頻度が多くなるにつれ小さくなり、コネクションの最大開設数が多くなれば多くなる。

この原因として推測できることは、rUDP をユーザ空間のライブラリとして実装しているためのオーバーヘッドの大きく、パフォーマンスがあまり良くないこと（現在の実装では、ラウンドトリップ時間は TCP の約 1.5 倍）の影響は大きいと考えられるが、開設済みのコネクションを閉鎖し、新たなコネクションを開設するオーバーヘッドが主な原因であると思われる。この実験アプリケーションは、サーバープログラムは多くのクライアントプログラムから頻繁にデータを受信し、その返事を返さなければならない。また、今回の閉鎖すべきコネクションの選択方法は、一旦データを受信し、その返事を返すまでの間でも、そのコネクションは一旦閉鎖されることが起こる。したがって、コネクションの最大開設数を越える数のクライアントプログラムに対処するために、頻繁にコネクションの閉鎖、開設処理を行なわなければならない。

この問題を解決するためには、

1. rUDP のパフォーマンスを改善する。
2. アプリケーションに適切なコネクションの最大開設数を設定する。
3. 閉鎖すべきコネクションの選択方法をアプリケーションに適切なものとする。

などが考えられる。特に、2、3 は今後さらなる調査が必要である。パフォーマンスに関しては、現在の実装はユーザ空間のライブラリとしているためオーバーヘッドは大きいが、その中でも、特にスレッド間での同期、メモリの排他制御のためのオーバーヘッドが最も大きいと考えられる。したがって、これらの最適化は必要である。

7 まとめ

本稿では、信頼性のあるコネクションレスデータグラム通信プロトコル（rUDP）を TCP 上で実現する手法に関して述べた。実現する上で重要な点は TCP のコネクションの隠蔽と流量制御である。我々は、ある時刻で同時に開設できるコネクションの数を制限し、その範囲内でコネクションを自動的に開設、閉鎖する方法を提案し、その際に必要なコネクションの自動開設、閉鎖の仕組みを述べた。さらに、これらを効率良く行なうために TCP の受信バッファからデータを一旦蓄積するために中間バッファを導入し、そのバッファを効率良く利用する方法に関しても述べた。

我々はこの方法に基づき、OS/2 バージョン 2.0 上で実現し、その動作を観測した。その結果、コネクションの最大開設数とデータ転送の頻度、一つのプログラムが対応できるプログラム数は密接な関係があることがわかった。これらの結果からデータ転送頻度が大きいと、コネクションの閉鎖、開設処理が頻繁に起こり、そのオーバーヘッドが大きくなることが推測される。今後、さらにコネクションの最大開設数とデータ転送頻度、対応可能なプログラム数の関係を調査し、その対応策を検討する必要がある。

参考文献

[1] D.E.Comer,D.L.Stevens:Internetworking with TCP/IP vol1,2 Prentice Hall

[2] 西田 竹志:TCP/IP ネットワークプロトコルとインプリメント（株）SRC

[3] S.J.Leffler 他:UNIX 4.3BSD の設計と実装（日本語）丸善出版

[4] C.Partridge:Implememting the Reliable Data Protocol(RDP)

[5] D.R.Cheriton:The V Distributed System Communication of ACM Vol31 No3 1988

[6] A.L.Ananda 他:A Survey of Asynchronous Remote Procedure Calls

[7] A.L.Ananda 他:ASTRA-An Asynchronouse Remote Procedure Call Facility 1991 IEEE

[8] B.J.Nelson: Remote Procedure Call XEROX parc.