マルチメディア通信と分散処理 72－8
（１９９５．９．８）

# 高速トランスポートプロトコルのライブラリ形式による実装と評価

三宅 優　　加藤 聰彦　　鈴木 健二

国際電信電話（株）　研究所

〒356 埼玉県上福岡市大原 2-1-15

ローカルエリアネットワークおよび広域網の高速化により、地理的に分散された計算機間で、高速な伝送路が利用可能となる。このようなネットワークで高速通信を行うには、通信プロトコルの果たす役割が重要であるが、現在広く使用されているTCPでは、最大ウィンドウサイズの制限や輻輳回避手順の不備により、高スループットを達成するのは難しい。そこで筆者等は、XTPをベースとした高速通信プロトコルを、ライブラリ形式で実装したトランスポートライブラリを作成した。本ライブラリは、新しい通信プロトコルの実装を容易にすると共に、UNIXカーネル内のTCPプロトコルに比べ、高スループットを達成し、遅延の影響も最小限に抑えている。本稿では、トランスポートライブラリの設計方針、実装の詳細、性能評価結果について述べる。

# Implementation of High Speed Transport Protocol as User Level Library and its Evaluation

Yutaka Miyake　　Toshihiko Kato　　Kenji Suzuki

KDD R & D Laboratories

Ohara 2-1-15, Kamifukuoka, Saitama 356, JAPAN

Along with the rapid increasing of transmission speed for LAN and public networks, wide bandwidth will be available for the wide area communications. To realize high speed communication within such networks, the performance of communication protocol is important. However, the current protocols such as TCP/IP have some problems, especially in long distance and high speed networks. Therefore, it is required to use new protocols with new data transfer algorithms. In this paper, we describe an implementation method of a high speed transport protocol based on XTP by using a user level library and the results of performance evaluation. This method allows a new protocol to be developed easily and the library gives the throughput of 32 Mbits/sec over ATM network whose effective transmission speed is 36 Mbits/sec, regardless the propagation delay from 0 to 200 msec. These values of throughput are better than those of the in-kernel TCP programs.

# 1 はじめに

光通信技術の発達により、伝送路の伝送速度の高速化が著しい。ローカルエリアネットワークにおいては、FDDI、ATM LAN、FiberChannel 等の導入が進んでおり、広域網においても、高速専用線や ATM ネットワーク等の整備により、伝送速度が飛躍的に向上すると考えられる。

これらにより、地理的に分散された計算機の間で、高速な伝送路を確保できるようになるが、このような条件下で高速通信を行うには、通信プロトコルが高性能でなければならない。現在の多くの UNIX ワークステーションでは、信頼性のあるデータ転送のために TCP を実装している。しかし TCP では最大ウィンドウサイズが 64K バイトに制限されており、遅延の大きなネットワークにおいては、十分な通信速度が得られない。ウィンドウサイズを拡張するオプションも提案されているが [1]、これを実装していない OS も多く、たとえ実装していても、最大ウィンドウサイズが制限されていたり、適切な輻輳回避手順が実装されていないために [2,3]、帯域に見合ったスループットを得られないことが指摘されている。

これらの問題を解決するために、XTP [4] 等の新しいプロトコルが検討されている [5,6]。しかし、これらのプロトコルを実装している UNIX ワークステーション用 OS はほとんど無く、容易に利用できる状況ではない。そこで筆者等は、XTP をベースとした高速通信プロトコルを実現するトランスポートライブラリを、UNIX ワークステーション上に作成した。本ライブラリは、UNIX カーネル内にプロトコルを組み込む場合と比較して開発や移植が容易である。このため、多くの UNIX ベースの OS で利用することができる。また、その実装においては、高いスループットを実現するために、データコピーを回避するための送受信バッファの管理や、ユーザレベルにおける受信パケット処理の高速化を実現している。

本稿では、このトランスポートライブラリの設計方針および実装の詳細について説明すると共に、ATM LAN を用いた性能評価を行ったので、その結果を報告する。

# 2 トランスポートライブラリの基本方針

## 2.1 プロトコル構造と実装形態

図 1にトランスポートライブラリと、アプリケーションおよび下位プロトコルの関係を示す。トランスポートライブラリは、フロー制御や誤り制御等を有するコネクション指向型のデータ転送機能を提供し、アプリケーションとリンクして使用される。

下位プロトコルは、UNIX カーネル内に実装されている UDP/IP を利用している。受信パケットの分配は、UDP のポートによりカーネル内で行わせることができる。また、IP パケットの到達範囲内であれば、ルータを介した相互通信が可能である。

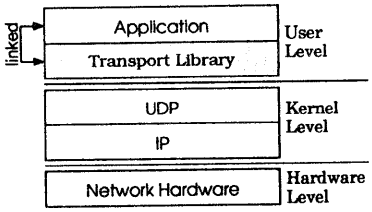


図 1: トランスポートライブラリを使用した場合のプロトコル構成

## 2.2 プロトコル機能

トランスポートライブラリのプロトコル仕様は、XTP [4,6] をベースとして、UDP/IP 上で動作させ、またユーザレベルで高速な通信を実現するための機能を持たせている。以下に、本プロトコルの主な機能を示す。

- XTP と異なり、明示的なコネクション確立フェーズを設定し、アプリケーションのバッファサイズ、UDP で使用するソケットバッファサイズやポート番号等を交渉する。
- 制御用のパケットとデータ用のパケットを受信時に処理しやすくするために、それぞれに別の UDP のポートを与える。
- フロー制御とレート制御を協調して動作させる。フロー制御は、受信バッファのオーバフローを防ぐと共に、ネットワークでの輻輳を回避するために使用する。しかし、ウィンドウサイズを大きくすると、連続してパケットを送信することになり、ルータ等での受信バッファ溢れを招く危険がある。そこで、レート制御は、一定の送信速度でパケットを送出することにより、このバッファ溢れを抑制する。この 2 つの制御機構のパラメータをネットワークの状況に応じて動的に、そして協調して変化させることにより、ネットワークの輻輳を可能な限り抑える。
- 様々な品質のネットワークに対応するため、誤り回復方法として、パケットロスが発生した時点から再度順番にパケットを送信する go-back-N 方式と、紛失したパケットのみを再送する選択再送方式の両方を実装する。
- UNIX の実装上の制限により、UDP のソケットバッファが 64K バイトまたは 53K バイト以下に制限されているものがある。このサイズは、高速で遅延の大きな通信においては、十分であるとは言えない。そこで、複数のソケットバッファを操作することにより、疑似的に受信バッファを拡大させる方法を実装する。

## 2.3 実装方針

ライブラリ形式として実装し、ユーザレベルで動作するプロトコルソフトウェアで、高速な通信を行うために、以下の方法を用いて実装を行った。

- アプリケーションは、フレームバッファなど、独自の目的のために確保したメモリ領域を利用する。そこで、以下のようなバッファ管理法を採用する。
  - トランスポートライブラリは、送受信バッファを持たず、アプリケーションがデータの送受信のために指定したメモリ領域を、そのまま使用する。これにより、トランスポートライブラリでの送受信データのメモリコピーを無くし、スループットを向上させる。
  - 送信時に複数のバッファを指定させることにより、1つのバッファに対する受信確認を受け取る前に、別のバッファのデータを送信し、データの連続転送を可能とする。
- UNIX では、ユーザレベルで動作中のプログラムは、パケットの到着を知らせる受信割り込みを受けることができない。そこで、制御用パケットに対しては、できる限りポーリングを行い、受信したパケットをすばやく処理する。
- プロトコル処理には、いくつかのタイマを実装することが要求されるが、これらの処理を可能な限り低いオーバヘッドで行うようにする。

# 3 各機能の実装

本節では、トランスポートライブラリで使用されている機能の実装方法について述べる。

## 3.1 フロー制御とレート制御機構の協調動作

フロー制御による輻輳回避は、文献 [2] による手順をベースとしている。この手順では、受信側が送信側に対して通知するウィンドウサイズとは別に、輻輳ウィンドウサイズと呼ばれるパラメータを導入し、この値により送信可能なパケットの上限を定めている。その値は、パケットロスの検出や通信状況により変化する。さらに、レート制御機構におけるパケットの送信速度を、輻輳ウィンドウサイズを用いて、以下の計算式により定める。

$$\text{送信速度} = \frac{\text{輻輳ウィンドウサイズ}}{\text{平滑化された往復遅延時間}}$$

この値に従い、送信側で送信パケットの送出間隔を調整する。

レート制御で使用するタイマは、非常に短い間隔を扱わなければならない。そこで、3.6節で述べるタイマを利用せず、パケット送出前に `gettimeofday()` により時刻を取得し、直前に送出したパケットとの間隔が、求められた送出間隔よりも短い場合は、`select()` により待つという処理を行っている。

## 3.2 誤り回復方式

go-back-N 方式は、処理負荷が軽いため、遅延が小さい場合に有効であり、選択再送方式は、遅延の大きいネットワークを利用する際に有効である。パケットロスが発生した時に、どちらの誤り回復方式を選択するかは、データの送受信を行っている計算機が独自の判断で決定できる。すなわち、受信側から送信される誤り回復要求パケットの中に、紛失パケットのシーケンス番号情報が列挙されていれば選択再送方式であり、1つしかなければ go-back-N 方式となる。送信側においても、誤り回復要求パケットの中に紛失パケットのシーケンス番号が列挙されていても、その最初のシーケンス番号から順に送り始めれば、go-back-N を選択したことになる。どちらの方式を選択するかは、アプリケーションがトランスポートライブラリを初期化する時点に決定する。

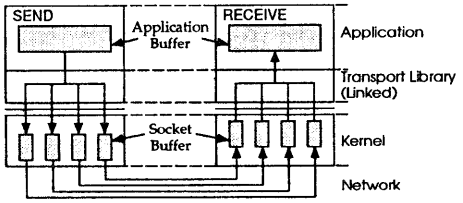


図 2: 複数のソケットバッファを用いたカーネル内送受信バッファの拡大

## 3.3 ソケットバッファサイズの拡大

UDP が利用するカーネル内の送受信バッファ(ソケットバッファ)は、`setsockopt()` システムコールにより確保することができる。この関数により確保できるバッファの最大値は、多くの UNIX の実装において、64K バイトまたは 53K バイトに制限されており、高速で遅延の大きな通信経路においてウィンドウサイズを大きくしなければならない場合では、不十分なサイズである。そこで、トランスポートライブラリでは、図 2に示されるような、一本のコネクションに対して複数のソケットバッファを操作することにより、カーネル内の送受信バッファサイズを増加させる方法を採用した。

通信プロトコルは、コネクション確立時に、この通信で使用する UDP ソケットの数と対応するポート番号を交渉し、交渉により決められた数のソケットバッファを送受信側の双方で確保する。複数のソケットを利用する場合には、送信側は送信データを各ソケットに順にデータを送信し、受信側では複数の UDP ソケットでデータ受信を待つ。これにより、ソケットバッファのオーバフローによるパケットロスの発生を減少させることが可能になる。

## 3.4 アプリケーションバッファの管理

### 3.4.1 アプリケーションバッファでのデータコピーの削減

データのコピーは、通信スループットを低下させる要素の1つである。そこで前述のように、トランスポートライブラリでは、アプリケーションが確保したバッファを直接送受信に利用する。

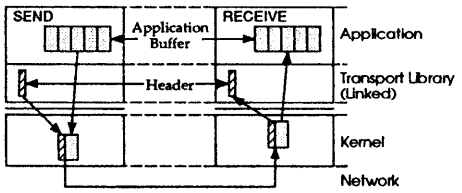

図 3: データコピーを避けたヘッダとユーザデータの操作

データ送信時には、アプリケーションはアプリケーションバッファの先頭ポインタとそのサイズを設定して送信関数を呼び出す。送信関数は、アプリケーションバッファのデータを、パケットとして送信するために最適なサイズで分割し、トランスポートライブラリが別領域に確保したヘッダと同時に、カーネル内のソケットバッファに書き込む (図3参照)。

受信側では、データの受信前にアプリケーションバッファの先頭ポインタとサイズを設定して受信関数を呼び出す。受信関数は、データパケットの到着を待ち、受信したデータをヘッダ部とデータ部に分離する。ヘッダ部は、トランスポートライブラリが確保した領域に書き込み、データ部はアプリケーションバッファに直接書き込む。なお、これらの送受信処理には、scatter read/gather write を行うシステムコール (`readv()`/`writev()`) を利用する。

### 3.4.2 アプリケーションバッファの複数化

トランスポートライブラリの処理は、アプリケーションとリンクされているため、そのアプリケーションのプロセス内で行われる。したがって、制御の流れは、アプリケーション側かトランスポートライブラリ側のどちらかにしか存在せず、これらの間で同時に別々の処理を行うことができない。

このため、アプリケーションが送信関数 (`send()`) を呼び出すと、プロセスの制御はトランスポートライブラリに移動し、トランスポートライブラリ内でデータ送信のための処理が行われる。前述のように、トランスポートライブラリは、アプリケーションバッファをそのまま使用しており、アプリケーションは、送信関数がリターンするとそのバッファを自由に操作する。また、送信関数は、バッファ内のデータに対する受信確認が行われるまで、制御用パケットの処理を行うための制御を有する必要がある。これらの理由により、送信関数の処理を終了させるのは、受信側からの全送信データに対する受信確認を受け取った後でなければならない (図 4(a) 参照)。

大容量データ転送では、複数回送信関数を呼び出し、データを連続して送信するのが一般的である。しかし、この場合は、送信関数内で、受信側からの受信確認を待つまでに、データを何も送信しない時間帯が発生してしまい、伝送帯域を十分利用することができない。この影響は、遅延が大きいネットワーク程、大きくなる。

この問題を解決するために、トランスポートライブラリでは、アプリケーションバッファを複数用意し、受信側からの受信確認の待機中もデータを送信できる手順を導入した。図 4(b) に、その手順を示す。

この手順では、送信関数の呼び出し時に複数のアプリケーションバッファを指定する (図 4(b) では、2 つのバッファを指定している)。送信関数は、指定されたバッファを順にパケットに分割してデータを送信していく。最初のバッファの内容を送信した後も、次に送るべきデータを持つバッファが指定されているため、そのバッファの内容のデータを続けて送信する。データ送信中に、最初のバッファに対する受信確認が到着する。これにより、最初のバッファの送信が終了したとして、一旦関数を抜け、アプリケーションに制御を移す。アプリケーションは、次に送信すべきデータが格納されているバッファを用意し、そのバッファの先頭ポインタとサイズを引数として再度送信関数を呼び出す。送信関数は、以前の続きからデータの送信を再開し、そのバッファの受信確認が到着した時点でアプリケーションに制御を移す。

この手順の導入により、受信側からのデータ受信確認のパケットを待機している時間帯にもデータ送信が可能となり、伝送帯域に見合った通信スループットを達成することが可能となる。

## 3.5 制御用パケットの処理

トランスポートライブラリでは、送受信計算機間でお互いの情報を交換するために、制御用パケットを使用している。このパケットの到着を、以下のポイントにおいて確認している。

1. `select()` の代替関数を提供し、その関数をアプリケーションが呼び出した時にチェックする。
2. データの送受信関数の呼び出し時に、アプリケーションのデータを処理する前に制御用パケットの到着を確認し、到着していればそれを読み込み、処理を行ってから、送受信処理を行う。

## 3.6 タイマ

プロトコル処理には、コネクションの管理や誤り制御機構のためのタイマが必要である。このタイマ機構を実装するために、インターバルタイマを設定する `setitimer()` 関数と、`SIGALARM` シグナルを利用した。トランスポートライブラリの内部では、デフォルト値が 500 ミリ秒のインターバルタイマを動作させ、プロトコル処理で利用されている各タイマがタイムアウトしたかどうかを、この間隔でチェックしている。

トランスポートライブラリが `SIGALARM` シグナルを利用しているために、アプリケーションは、`setitimer()`、`sleep()`、`usleep()`、`fork()` 等の、このシグナルを利用する関数を直接利用できない。そこで、これらの関数と同等の動作をする代替関数をアプリケーションに提供した。

# 4 性能評価

トランスポートライブラリの有効性を確認するために、その評価を行ったので、本節で述べる。

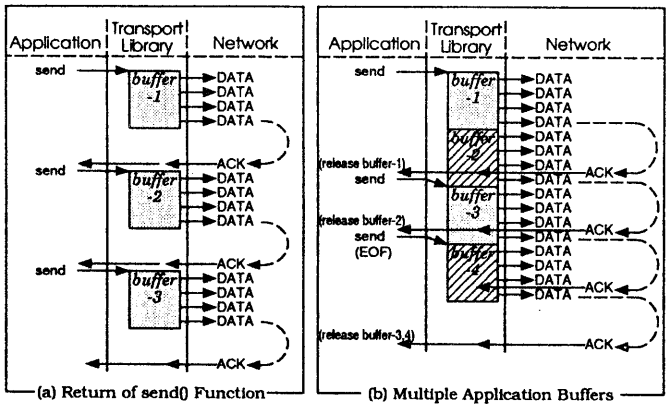


図 4: アプリケーションバッファの管理

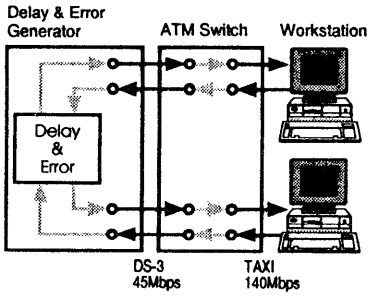


図 5: 評価に使用したシステムの構成

## 4.1 評価方法

図 5に、性能評価に使用したシステムの構成を示す。評価には、ATM LAN を使用し、ATM のスイッチに遅延とエラーを挿入することが可能な装置を接続した。

使用したワークステーションは、SPARCserver 670MP (SPARC CPU 40MHz × 2、SunOS 4.1.2) と SPARCstation 20 (SuperSPARC CPU 60MHz × 2、Solaris 2.3) である。これらのワークステーションは、ATM インタフェースにより ATM スイッチへ 140Mbps の伝送速度で接続される。遅延/エラー発生装置は、DS-3(45Mbps) で接続されている。通信を行うワークステーション間の伝送パスは、PVC の設定により必ず遅延/エラー発生装置を経由する。この伝送パスのボトルネックは、DS-3 である。DS-3 の正確な伝送速度は、44.736Mbps であるが、DS-3 のフレーミングや ATM セルのヘッダのオーバヘッドにより、ATM のペイロードで運ばれるデータの最大伝送速度は、36.864Mbps である。

トランスポートライブラリを用いて通信を行った際のスループットを計測するために、2 つのワークステーション間でメモリ上のデータを転送するプログラムを作成した。このプログラムに、トランスポートライブラリのいくつかのパラメータを操作する機能と、送受信したデータサイズと経過した時間から通信スループットを計算する機能を実装した。評価においては、フリーソフトウェアである `ttcp` を用いて、UNIX カーネル内に実装されている TCP の性能評価も行った。評価の対象とした TCP は、ウィンドウスケールオプション [1] を実装していないため、最大ウィンドウサイズは 64K バイト以下に制限されている。

| | |
|---|---|
| アプリケーションバッファサイズ | 512K バイト |
| アプリケーションバッファの数 | 1, 2, 4, 8 |
| UDP ソケットの数 | 4 |
| エラー回復方法 | 選択再送方式 |
| レート制御 | OFF |

表 1: 評価に使用したパラメータ

## 4.2 性能評価の結果

通信を行っている 2 台のワークステーション間の往復遅延時間と通信スループットの関係を調べた。トランスポートライブラリを用いて通信を行った際に使用したパラメータを表 1に示す。図 6に測定結果を示す。トランスポートライブラリを使用した測定では、送信関数に指定するアプリケーションバッファの数を、1、2、4、8 の 4 通りで測定しており、グラフ上の数はそのバッファ数を示している。`ttcp` を用いた TCP の測定では、ソケットバッファサイズを 53K バイトとし、できる限り高スループットが達成できるパラメータを選択した。

遅延を挿入していない場合では、TCP の最高スループットが 26.78Mbps であるのに対し、トランスポートライブラリでは、アプリケーションバッファの数に依存すること無く、約 32Mbps を達成している。TCP のスループットは、最大ウィンドウサイズの制限により、遅延の増加に従って急激に減少している。それに対し、トランスポートライブラリでは、アプリケーションバッファ数が 4 および

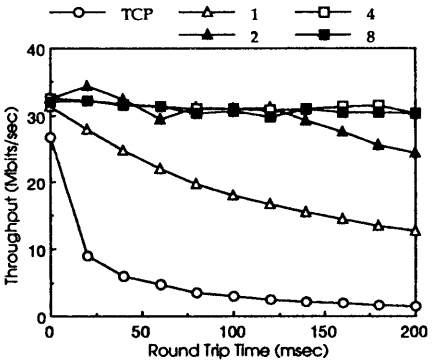


図 6: 往復遅延時間とスループットの関係

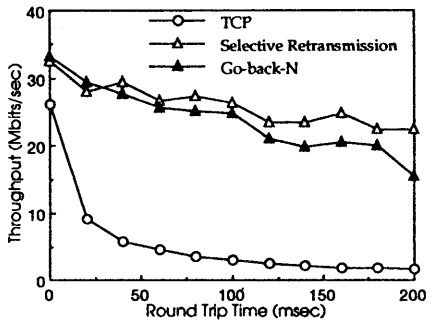


図 7: 誤りを挿入した場合の
往復遅延時間とスループットの関係

8 の時において、遅延の影響を受けること無くスループットを維持している。

アプリケーションバッファ数が 1 の時は、3.4.2節で述べたように、連続して送信関数を呼び出す場合に、受信確認を待つためにデータを送信できない時間帯が存在する。この時間帯は、往復遅延時間の増加と共に長くなるため、スループットも徐々に減少していく。アプリケーションバッファ数が 2 の場合は、往復遅延時間が約 125 ミリ秒の時点からスループットの減少が発生しているが、これは、この遅延時間以上の場合、512K バイトのバッファを 2 つ送信終了した時点で、最初のバッファの受信確認が到着しないため、データ送信できない時間帯が発生するためである。

図 7に、伝送路に $1.0 \times 10^{-6}$ の割合でランダムビットエラーを与えた時の、往復遅延時間とスループットの関係を示す。測定に使用したパラメータは、表 1と同じであるが、アプリケーションバッファの数は 4 に固定で、誤り回復方式を、go-back-N と選択再送の両方式を使用した。

選択再送による処理は、go-back-N と比較して特に負荷が高いということはなく、遅延が小さい場合では、両者にそれほどスループットの差はない。遅延の増加にともない、選択再送方式の有効性が現れている。

また、以上の結果より、本ライブラリで実装されたフロー制御や誤り制御は、遅延の増加に伴い、送受信ワークステーション間での再送要求などの交換に時間を要するようになった場合も、有効に動作することが確認された。

## 5 おわりに

本稿では、XTP をベースとした高速通信プロトコルをライブラリ形式で実装したトランスポートライブラリについて報告した。本ライブラリは、新しい通信プロトコルの実装を容易にする共に、UNIX カーネル内に実装されている TCP プロトコルに比べ、高スループットを達成し、遅延の影響も最小限に抑えている。このトランスポートライブラリを、ファイル転送プログラムである FTP に組込み、高スループットでファイル転送を行えることも既に確認している [8,9]。今後は、様々なネットワーク環境での評価を進めると共に、このライブラリを利用したアプリケーションの開発を行っていく予定である。最後に、日頃御指導頂く KDD 研究所 浦野所長に感謝します。

## 参考文献

[1] D. Borman, R. Braden, V. Jacobson, "TCP Extensions for High Performance," RFC-1323, May 1992.

[2] V. Jacobson, "Congestion Avoidance and Control, " *Proceedings of the ACM SIGCOMM Conference*, pp. 314–329, August 1988.

[3] L. S. Brakmo, S. W. O'Malley and L. L. Peterson, "TCP Vegas: New Techniques for Congestion Detection and Avoidance," *Proceedings of the ACM SIGCOMM Conference*, August 1994.

[4] "XTP Protocol Definition Revision 3.6," *Protocol Engines Inc.*, PEI 92-10, 11 January 1992.

[5] W. A. Doeringer, D. Dykeman, M. Kaiserswerth, B. W. Meister, H. Rudin and R. Williamson, "A Survey of Light-Weight Transport Protocols for High-Speed Networks," *IEEE Trans. Commun.*, Vol. 38, pp. 2025–2039, November 1990.

[6] 三宅、加藤、鈴木、"$x$-kernel 上へ実装した XTP プログラムの性能評価," 情報処理学会第 50 回全国大会、1T-7、1995.

[7] C. Partridge and S. Pink, "A Faster UDP," *IEEE Trans. on Networking*, vol. 1, pp. 429-440, August 1993.

[8] 三宅、加藤、鈴木、"高速通信プロトコルを実現するトランスポートライブラリの実装と評価," 情報処理学会第 51 回全国大会、1E-7、1995.

[9] Y. Miyake, T. Kato, K. Suzuki, "Implementation Method of High Speed Protocol as Transport Library," *Proceedings of the ICNP '95*, November 1995. (To appear.)