システムソフトウェアと
オペレーティング・システム 91－3
（2002．8．21）

プロトコルスタックをバイパスする Myrinet用ソケット通信機構

中井 良一† 日下部 茂†† 荒木 啓二郎††

九州大学大学院システム情報科学府†
九州大学大学院システム情報科学研究院††

近年，ネットワークハードウェアの性能向上に伴い，データ通信の際のソフトウェア処理のオーバヘッドが相対的に高価になってきている．また，ソフトウェアオーバヘッドを低減した高速通信ハードウェア向けの通信ライブラリも存在するが，これらのライブラリは一般的に用いられているソケット APIとは異なる APIを採用している．ソケットを用いたアプリケーションはそのままでは利用することが出来ず，専用の APIで書き換えなければならなかった．本論文では，高速な通信ハードウェアとして Myrinetを用いた環境で，ソケット APIと互換性のある APIを持ち，かつ通常のプロトコルスタックをバイパスする事で低ソフトウェアオーバーヘッドを実現したソケット通信機構について述べる．

Bypassing Protocol Stack in Socket Communication on Myrinet

Ryoichi Nakai, Shigeru Kusakabe, and Keijiro Araki

Graduate School Information Science and Electrical Engineering, Kyushu University

Recent performance improvement of hardwares makes software overheads more outstanding. Although communication libraries for high performance networks with reduced software overheads have been proposed, their interfaces are different from socket interface generally used. Thus, existing applications using socket interface are unable to use these libraries without rewriting by using interfaces specific to such libraries. In this paper, we propose a socket communication mechanism for Myrinet, which has general socket interface and low software overheads by bypassing the existing protocol stack.

1 はじめに

近年，Myricom 社の Myrinet[1] のようなギガビットクラスの高速通信ハードウェアが比較的安価に手に入るようになった．このような高速通信ハードウェアは，複数の計算機を接続して並列・分散処理を行うクラスタシステムにおける計算機間の通信に広く用いられている．

しかしながら，高速通信ハードウェアを導入するだけでそのハードウェアの持つ高い性能を全て引き出すことができるというわけではない．通信を行う際には，データが通信路中に存在する通信遅延時間に加えて，データが計算機のメモリと通信路の間を移動する際に生じるハードウェアオーバヘッドと，データに対して通信のための処理を加える際に生じるソフトウェアオーバヘッドが発生するためである．

通信ハードウェアの性能向上により，通信媒体そのものによる通信遅延時間は飛躍的に短縮されてきている．また，マイクロプロセッサやデータバスのような計算機ハードウェアの性能向上によってハードウェアオーバヘッドも短縮されてきている．このため，ソフトウェアオーバヘッドによる遅延時間が相対的に大きくなり，近年では通信におけるボトルネックとなってしまっている．

そこで，ソフトウェアオーバヘッドを低減するためにユーザレベル高速通信ライブラリ (AM[2]，FM[3]，PM[4] など) が提案され，開発・実用化されてきた．しかし，これらユーザレベル高速通信ライブラリはそれぞれ API やセマンティクスが異なっており，従来の TCP/IP とソケットを用いて作成されたプログラムはそのままでは使用することが出来ず，新たにユーザレベル高速通信ライブラリ用の API を用いてプログラムを書き直す必要があった．つまり，プログラマは新たにユーザレベル高速通信ライブラリ用に API を習得しなおし，その API を用いてプログラムを書き直す必要に迫られ，大きな開発コストを要することになる．

本論文ではオペレーティングシステムのプロトコルスタックをバイパスするソケット通信機構 [5] について述べる．このソケット通信機構は高速通信ハードウェア Myrinet 用のソケット通信機構で、従来の TCP/IP におけるソケット API と互換性のある API を持つ．そのため，ソケット API を用いた既存の分散アプリケーションを変更無しに Myrinet クラスタ上で実行できる．さらに Myrinet の特徴を生かして不要なソフトウェア処理を削減することでソフトウェアオーバヘッドを低減している．また，このソケット通信機構は複数の通信ハードウェアを併用する環境での使用を想定しており，Myrinet 以外の通信ハードウェアで接続された計算機との通信の際は通常のオペレーティングシステムのプロトコルスタックを用いて通信を行うことが出来る．以降，本論文中ではこのソケット通信機構を高速ソケット通信機構と表記する．

2 Myrinet

この章では高速通信ハードウェアの一つであり、本研究において通信ハードウェアとして用いた Myrinet，および Myrinet を制御するためのドライバについて説明する．

2.1 Myrinet

Myrinet は高バンド幅，低遅延なパケット交換方式の高性能通信ハードウェアであり，同一クラスタ内の計算機同士を接続するためのネットワークとして広く用いられている．

Myrinet は，NIC(Network Interface Card)，全二重のリンク，カットスルー・ルーティングを行うスイッチから構成される．

Myrinet-NIC は NIC 内にプロセッサとローカルメモリを備えており，このプロセッサに独自のファームウェアを実行させることによって独自のデータ転送方式を実装することが可能である．

Myrinet は，heartbeat によって全てのリンク先の計算機が動作しているかを継続的にチェックし，フロー制御とエラー制御によってメッセージの配送とメッセージの配送順序を保証する．また，Myrinet-NIC が CRC チェックを行うので信頼性も高く，Myrinet を用いた通信では通信相手の物理層レベルまで確実にメッセージを届けることが保証されている．

2.2 Myrinet ドライバ

Myrinet ドライバは Myrinet-NIC を制御し，Myrinet によるデータ転送を行う．

高速ソケット通信機構では通信時に発生するデータコピーの回数を削減している．そのコピー回数の削減のため，データを受信する前にカーネルがあらかじめデータ受信領域を指定することができるドライバが必要であったので，Myrinet 用のドライバと

して九州大学大学院システム情報科学府の中島らが開発した Myrinet ドライバを使用している．

この Myrinet ドライバは Myrinet-NIC に独自のファームウェアを実行させ，Myrinet によるデータ通信を制御する．

Myrinet ドライバがメインメモリと Myrinet-NIC 間のデータコピーを行う際は，Myrinet-NIC のローカルメモリ上に存在する送受信領域管理表によってメインメモリの送信領域と受信領域を把握し，その領域情報に従って DMA 転送を行う．

Myrinet ドライバによるデータ送受信の様子を図 1 に示す. まず，送信側計算機は送信データのアドレスを，受信側計算機は受信用バッファのアドレスを Myrinet-NIC 上の送受信領域管理表に登録する．その後送信データは DMA 転送により Myrinet-NIC に転送され，Myrinet リンクを通して受信側計算機の Myrinet-NIC へと転送される．さらにこのデータは先に登録された受信バッファへ DMA 転送される．

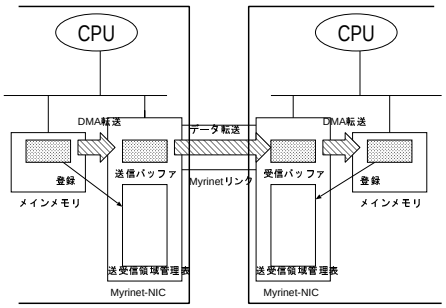

図 1: Myrinet ドライバによるデータ送受信の様子

3　設計

この章ではソケット通信によるソフトウェアオーバーヘッドの要因を明らかにし，高速ソケット通信機構で実現したソフトウェアオーバーヘッドの削減方法について述べる．

3.1　ソフトウェアオーバーヘッドの要因

3.1.1　信頼性の保証

一般的に用いられている TCP/IP による通信は，信頼性の低い通信路でも利用されることが想定されており，ソフトウェア側で信頼性を高めるため，様々な処理を行っている．具体的には、

- チェックサム計算
- フロー制御
- 再送制御
- 輻輳制御

といった制御が行われている。

3.1.2　データコピー

TCP/IP では送信側アプリケーションのデータ送信から受信側アプリケーションのデータ受信までに送信時に 2 回，受信時には 3 回のデータコピーが発生する．

具体的には送信時には「ユーザメモリ領域からカーネルメモリ領域へ」と、「カーネルメモリ領域から NIC のバッファへ」の 2 回、受信時には「NIC のバッファからカーネルメモリ領域へ」、「カーネルメモリ領域からソケットの受信バッファへ」、「ソケットの受信バッファからユーザメモリ領域へ」という 3 回である。

プロセッサによるデータコピーはコピー時間のオーバヘッドやメモリバンド幅の消費など、非常に高価な処理であるため，データコピーの回数がソフトウェアオーバヘッド全体に占める割合は大きい．

3.1.3　モード変更

アプリケーションがソケットを用いて通信を行う際は，オペレーティングシステムの実行モードがユーザレベルからカーネルレベルに切り替わり，カーネルでの処理が終了すると今度はカーネルレベルからユーザレベルへと切り替わる．このモード切替は比較的高価な処理であるため，ソフトウェアオーバヘッドの要因となっている．

3.2　ソフトウェアオーバーヘッドの削減

3.2.1　信頼性の保証処理の削減

高速ソケット通信機構では，通信ハードウェアとして Myrinet を用いているが，Myrinet の特徴を利用することで，信頼性を保証するためのソフトウェア処理の一部を削減することが可能である．Myrinet ではデータを確実に通信相手の物理層までは届けることができるので，再送制御を行う必要は無い。また、ハードウェアによる CRC チェックを行っているため、ソフトウェア側でチェックサム計算を行う必要も無い。高性能通信ハードウェアを持った計算機は，お互いがカットスルールーティングを行うス

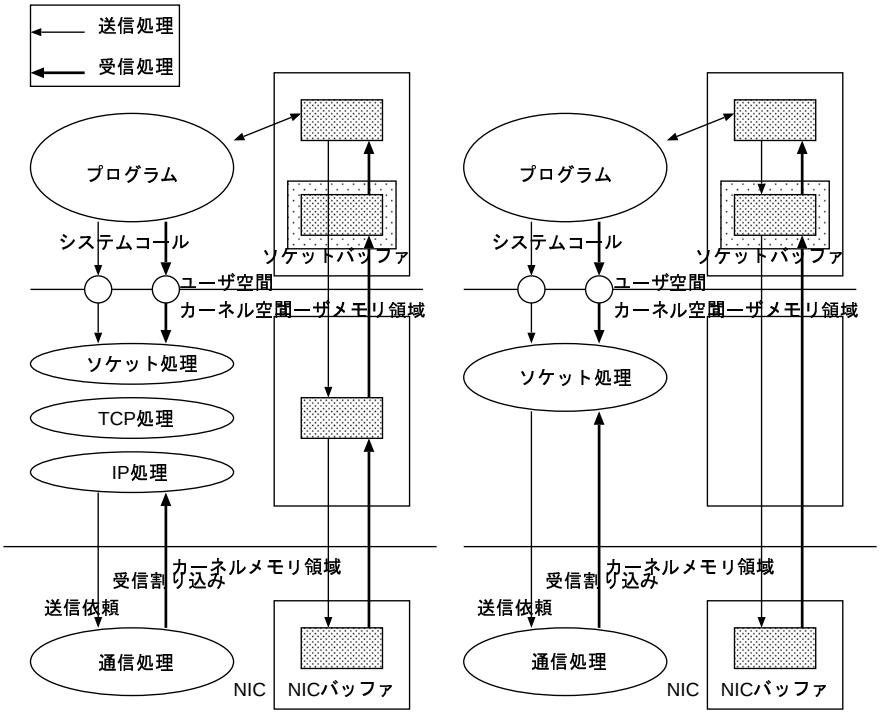

図 2: ソケット通信におけるデータ送受信の様子

イッチを介して繋がっているために輻輳もおこらない．以上の理由から，本研究における高速ソケット通信機構では再送制御，チェックサム計算，輻輳制御は行わない．

しかし，計算機の記憶領域は無限では無いため，相手ソケットの受信バッファを溢れさせないようにフロー制御は行う必要がある．

3.2.2 データコピーの削減

本研究で用いた Myrinet ドライバでは，NIC がデータを受信した際に送受信領域管理表に従ってデータを目的のバッファへとコピーするように設計されている．そのため、あらかじめソケットの受信バッファ用の領域を送受信領域管理表に登録しておくことで、データ受信の際の「NIC からカーネルメモリ領域へ」のデータコピーを行うことなく、「NIC からソケットの受信バッファへ」とデータをコピーすることが出来る．すなわち，受信の際のデータコピーを TCP/IP のときの 2/3 に削減したことになる．

一方，モード変換のオーバーヘッドに関しては削減することができない．ユーザレベル高速通信ライブラリではユーザレベルで通信ハードウェアを操作することによってモード変換のオーバーヘッドを削減しているが，高速ソケット通信機構はカーネル内での処理を行うためである．

通常の TCP/IP プロトコルスタックを用いた場合と高速ソケット通信機構を用いた場合のデータ送受信の様子を図 2 に示す．通常の TCP/IP における送受信では，送信時に二度，受信時に三度のデータコピーが発生する．また，カーネル空間ではプロトコル固有の様々な処理が行われる．一方，高速ソケット通信機構では送信時に二度のデータコピーが発生することは変わらないが，受信時に発生するデータコピーは二度に抑えられている．また，カーネル空間で行われる処理も削減されている．

3.3 通信プロトコルの振り分け

高速ソケット通信機構では Myrinet で繋がった計算機と, それ以外の通信ハードウェアで繋がった計算機へのソケット通信を区別し，カーネルがデータをそれぞれの通信ハードウェアに合った処理へと振り分ける．

Myrinet で繋がった計算機へのソケット通信では

プロトコルスタックをバイパスする高速ソケット通信機構を用い，それ以外の通信ハードウェアで繋がった相手とはオペレーティングシステムのソケット通信用プロトコルを使用する．アプリケーションはどちらの通信ハードウェアを使用するのかを意識することなく，プロトコル振り分けによって効果的なソケット通信が可能となる．

処理の振り分けはアプリケーションから与えられる通信相手のアドレス情報から判断する．

アドレス情報によるプロトコル切り替えの様子を図 3 に示す．ソケット API を通してカーネル空間にコピーされた送信データは，送信先のアドレスが Myrinet クラスタ上のものであれば高速ソケット通信機構を用いて送信が行われる．送信先のアドレスが Myrinet クラスタの外部のものであった場合は通常の TCP/IP プロトコルスタックを通して送信が行われる．

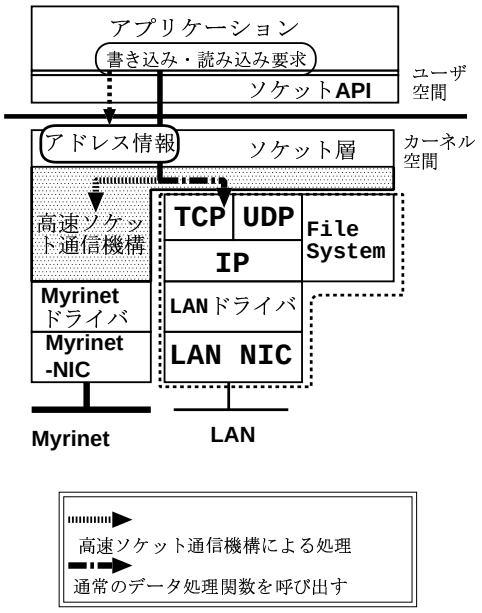

図 3: プロトコル切り替えの様子

4 実装

この章では，高速ソケット通信機構の実装に関して述べる．実装は Linux-2.4.0 カーネルに行った．

データ送受信の様子を図 4 に示す．

以降，送信操作，受信操作，受信割り込み操作についてそれぞれ説明する．

4.1 送信操作

高速ソケット通信機構の送信手順を次に示す．

1. アプリケーションから送信要求が発生．
2. アプリケーションから与えられる送信データをソケットの送信バッファへコピー．
3. 送信バッファの情報を Myrinet-NIC の送受信領域管理表に登録．
4. Myrinet ドライバを介して Myrinet に送信の契機を与える．

以上の動作で送信操作は終了である．TCP による通信ではデータが正確に届いたことを示す確認応答を受け取ることで送信成功と判断するが、高速ソケット通信機構では Myrinet ハードウェアによる確実なデータ転送と、高速ソケット通信機構のフロー制御により受信側のソケットの受信バッファへと確実にデータを送信できるため、確認応答の送信は行わない．

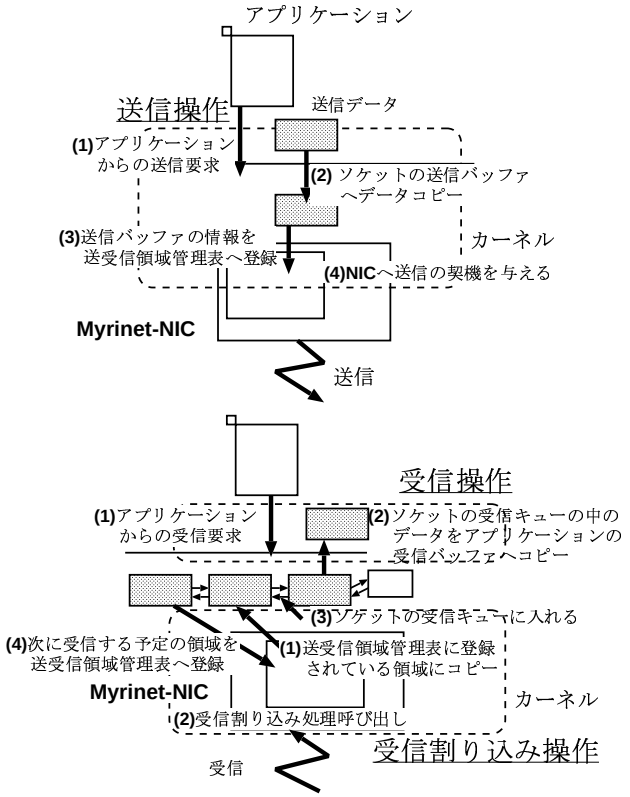

図 4: 高速ソケット通信機構によるデータ送受信の様子

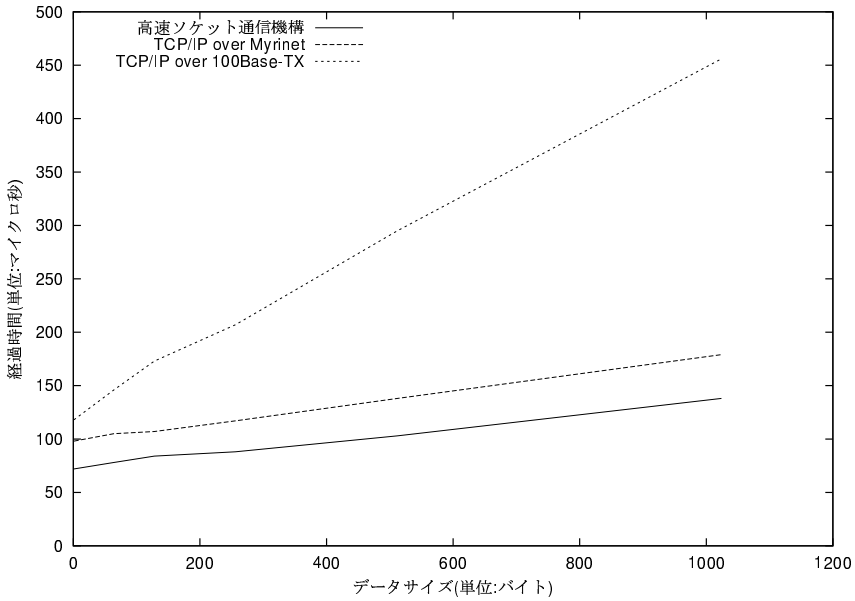

図 5: PingPongプログラムの実行結果

4.2 受信操作

高速ソケット通信機構の受信手順を次に示す.

1. アプリケーションから受信要求が発生.
2. ソケットの受信キューに入っているデータをアプリケーションの受信バッファへコピー.

受信操作はソケットの受信キューにあるデータをアプリケーションの受信バッファへコピーするだけである. 到着したデータをソケットの受信キューに入れるのは受信割り込み操作である.

4.3 受信割り込み操作

高速ソケット通信機構の受信割り込み処理の手順を次に示す.

1. NIC がデータの受信を検知し，送受信領域管理表にあらかじめ登録されている領域へ受信したデータをコピー.
2. 受信割り込み処理を呼ぶ.
3. 受信したデータを対象のソケット受信キューへ入れる.
4. 次にデータを受信する予定の領域を送受信領域管理表に登録.

受信割り込み処理では受信したデータをソケットの受信キューへ繋ぐ. さらに、次の受信データのためにあらたに受信領域を確保し，送受信領域管理表に登録を行う.

5 評価

高速ソケット通信機構の評価を行うため、二台の計算機間でデータの送信，受信を行う PingPongプログラムを作成し，一度の送受信にかかる時間を測定した。実験環境には CPU が PentiumIII の 1.26GHz，メイン メモリが 512MB，PCI バス 32bit33MHz の PC/AT 互換機を用い，Myrinet-NIC は 1MB のローカルメモリを搭載し，最大バンド幅が 1.2GHz のものを使用した. また，併せて通常の TCP/IP プロトコルスタックを用いて Myrinet で通信した場合，100Base-TX の Ethernet で通信した場合の測定も行った. この測定の結果を図 5 に示す。

測定の結果，データサイズが 1byte の時は高速ソケット通信機構では 72 マイクロ秒，通常の TCP/IP プロトコルスタックを用いて Myrinet で通信した場合は 98 マイクロ秒，100Base-TX の Ethernet で通信した場合は 118 マイクロ秒であった. データサイズが 1024byte の時は高速ソケット通信機構では 138 マイクロ秒，通常の TCP/IP プロトコルスタックを用いて Myrinet で通信した場合は 179 マイクロ秒，

100Base-TX の Ethernet で通信した場合は 456 マイクロ秒であった．

この結果は，TCP/IP による通信で行われている処理を削減し，データコピーの回数を削減した結果が反映されていると考えられる．Ethernet 上での通信と比較すると，データサイズが大きくなった際に差が大きくなっている．これは Myrinet が高バンド幅であるために通信するデータサイズが大きくなるほどに通信遅延時間に差が出ているためと考えられる．

今後はネットワークベンチマークソフトを用いたより包括的な性能評価を行い，次いで実アプリケーションへの適用を行う予定である．

6 関連研究

この章では高速通信ネットワーク上でソケット通信を行うための他のシステムについて述べる．

Berkley 大学の NOW(Network Of Workstation) プロジェクトの Fast Socket[6] は，通信ドライバ部分に AM ライブラリを用い，ユーザレベル通信ライブラリとしてソケット通信機構を実装している．Fast Socket はユーザライブラリであるために，例えば fork() を用いて子プロセスを生成した際に親プロセスとソケットを共有できない等，カーネル資源管理の点で問題があった．そのため，Fast Socket は利用できるアプリケーションに制限がある．

Myricom 社の Sockets-GM[1] はユーザレベル通信ライブラリである．モード変更を行わないカーネルバイパス，ゼロコピー通信などにより，高速なソケット通信を可能にしている．しかしながら，ユーザレベルでの実現であるため，やはり資源管理の点において問題を抱えている．

Duke 大学の Trapeze/IP[7] は Myrinet 上のソケット通信機構である．ユーザレベル通信ライブラリではなく，通信はカーネルを介して行われる．チェックサムオフローディング，scatter/gather DMA，ゼロコピー通信等により，高速なソケット通信を可能にしている．Trapeze/IP はユーザレベル通信ライブラリではないので，Fast Socket のようにカーネル資源管理に関しての問題はない．

富士通研究所の VI Socket[8] は下位の通信システムとして VIA[9] を用いている．VIA は様々な通信ハードウェアに対応するために抽象度が高くなっているため，環境への対応度は高いものの，Myrinet 上に限定した通信では本論文のソケット通信機構の方が高速に動作すると考えている．

7 おわりに

本論文では，Myrinet 上での通信においてオペレーティングシステムのプロトコルスタックをバイパスする事でソフトウェアオーバヘッドを削減するソケット通信機構の実装を行った．この高速ソケット通信機構では Myrinet の特徴を生かして通信時に不要な処理を省き，データコピーの回数を減少させることでソフトウェアオーバヘッドの削減を実現した．また，計算機が Myrinet 以外の通信ハードウェアで接続された相手と通信を行う場合のプロトコル選択について述べた．Myrinet で接続された計算機に対してはソフトウェアオーバヘッドの少ない高速ソケット通信機構で通信を行い，それ以外の通信ハードウェアで接続された計算機に対しては通常のオペレーティングシステムのプロトコルスタックを用いて通信を行う．このためアプリケーションは通信路を意識することなく従来通りのソケット API を用いて通信を行うことが出来る．また，高速ソケット通信機構はカーネルレベルで実装されているためにアプリケーションの再リンクの必要がない．高速ソケット通信機構の性能は単純なデータの送受信を行う PingPong プログラムでは Myrinet 上で通常の TCP/IP プロトコルスタックを用いた場合より高い，という結果が得られた。今後はネットワークベンチマークソフトを用いた性能評価，実アプリケーションへの適用を行う予定である．

8 謝辞

実験環境の一部の利用に関して、九州大学大学院システム情報科学研究院の雨宮真人研究室と村上和彰研究室に感謝いたします。

本研究の一部は、通信・放送機構の創造的情報通信技術研究開発推進制度に係る研究開発課題「次世代型インテリジェント・マルチメディア情報通信網の基盤技術に関する研究」による。

参考文献

[1] Myricom,inc.(http://www.myricom.com).

[2] http://now.cs.berkeley.edu/AM/active_messages.html

[3] http://www-csag.ucsd.edu/projects/comm/fm.html

[4] 手塚宏史, 堀敦史, 石川裕 ; ”ワークステーションクラスタ用通信ライブラリ PM の設計と実装”; JSPP’96, Information Processing Society of Japan, pp.41-48, 1996.

[5] 奥 裕治, 日下部 茂 ; ”高性能通信ハードウェア向け高速ソケット通信機構” ; 火の国情報シンポジウム 2002, pp.369-376, 2002.

[6] Steven H. Rodrigues, Thomas E. Anderson, and David E. Culler; ”High Performance Local Area Communication With Fast Sockets”; In Proceedings of the USENIX 1997 Annual Technical Conference, 1997.

[7] Andrew Gallatin, Jeff Chase, and Ken Yocum ; ”Trapeze/IP: TCP/IP at Near-Gigabit Speeds”, 1999 USENIX Technical Conference (Freenix Track), June 1999.

[8] 石嵜透, 伊藤雅典, 岸本光弘 ; ”SAN における socket 通信の高速化手法”, 情報処理学会研究報告, NO.01-Os-85, pp.1-8 2000.

[9] Intel Corp; ”Intel Virtual Interface (VI) Architecture Developer’s Guide Revision 1.0”; September 1998.