システムソフトウェアと
オペレーティング・システム 85－1
（２０００．８．４）

# SANにおけるsocket通信の高速化手法

石嵜　透
伊藤　雅典，岸本　光弘
(株) 富士通研究所　コンピュータシステム研究所　ソフトウェア研究部
{tooru,marc,hiro.kishimoto}@flab.fujitsu.co.jp

本論文ではSAN上のsocket通信オーバヘッド削減のために，従来のTCP,UDP/IPではなく，低オーバヘッドなプロトコルを透過的に導入したVI socketの提案を行う．VI socketはSTREAMS機構を利用したカーネルモジュールで，モジュール部とドライバ部から成る．ドライバ部は，STREAMSのTPIとSANのVIAインタフェースのマッピングをしており，従来に比べデータコピーやエラー制御のコストを削減し，低オーバヘッド通信を実現している．モジュール部は，通信相手がSANの内外であるかに応じてプロトコル切り替えをしており，透過的に切り替えることでSAN上の全てのsocketアプリケーションを低オーバヘッド化可能としている．プロトタイプを設計・実装し現在は評価中である．1BのRTTで従来の36%の性能改善を予測している．

# Improving socket communication performance on System Area Network

Tooru Ishizaki
Masanori Itoh, Mitsuhiro Kishimoto
Software Laboratory Computer Systems Laboratories Fujitsu Laboratories Ltd.
{tooru,marc,kiss}@flab.fujitsu.co.jp

In this paper, for cutting down overhead of all socket communication on SAN, we introduce low overhead protocol in place of TCP,UDP/IP. VI socket which consists of module component and driver component is kernel module using STREAMS sub system. Driver component is mapping protocol between TPI and VIA interface, that is low overhead by cutting off data copy and error handling costs. Module component is protocol selector between low overhead protocol and usual TCP,UDP/IP, which enables applications to be not need of change, for introducing VI socket. We designed and implemented VI socket for proving efficacy. We estimate cutting down 36% overhead on 1B Round Trip Time.

## 1 序論

ECやデジタルメディアなどネットワークを介して多種多様な情報を交換する現在の情報サービス分野では，データサーバを集中的に一元管理して低コストかつ高速・高信頼なシステムを構築するのが最近の傾向である (図1).

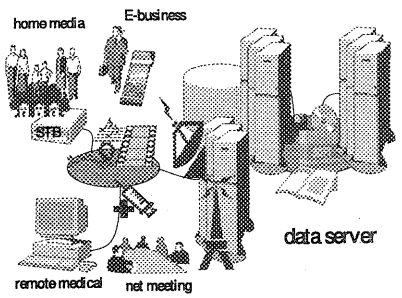


図 1 情報サービス

このようなシステムでは，計算機 (ノード) を複数接続したクラスタが用いられることが多い．その理由として，複数ノードであることを利用して高信頼，高可用なシステムを構築可能なことが挙げられる．また，構成部品をコモディティ化することで，最新技術を採り入れた低コストなシステムを容易に構築可能なことも理由の一つである．

ノード間のデータ交換には，従来では Ethernet などの一般的な LAN 技術が使われていた．しかし最近では，特に高速通信で性能ボトルネックな TCP，UDP/IP などの通信オーバヘッドを削減するために，SAN(System Area Network) という高速ノード間接続技術が使われるようになった．

ところが，十分に SAN の性能を引き出して性能向上するには，従来の TCP, UDP/IP を通る socket インタフェース [1] ではなく，SAN の API を使用しなければならなかった．アドホックな改良はいくつか試みられたが，API が独特であり改良の度に大きな開発コストが必要となった．最近では VIA[2] といった SAN の標準的な通信アーキテクチャの提唱により API 共通化は行われつつあるが，socket インタフェースからのセマンティクスの差は大きい．

そこで我々は VI socket の提案を行う．VI socket は，STREAMS 機構 [3][4] を利用したカーネルモジュールで，モジュール部とドライバ部から成る．ドライバ部は，SAN の標準的な通信アーキテクチャである VIA のカーネルインタフェース (VIA インタフェース) と STREAMS の TPI[3][5] のマッピングを行う．これにより，不要なプロトコル処理を削減し SAN の性能を十分引き出した低オーバヘッド通信を実現する．モジュール部は，Stream ヘッド下でプロトコル選択を行う．SAN 内の通信には低オーバヘッドプロトコル，SAN 外の通信には従来通り TCP,UDP/IP を使用する．これにより SAN 上の全ての socket アプリケーションに，手を加えることなく低オーバヘッド化を実現する．

さらに提案手法の実現性や有効性を実証するために VIsocket の設計・実装を行った．現在は評価中であるが，1B の RTT で従来の 36%の性能改善を予測している．

## 2 SAN 上の socket 通信

ここでは先ず SAN，VIA の概要について説明し，その後で，我々が採取した分析資料を元に socket 通信のオーバヘッドを明らかにする．

### 2.1 SAN

SAN は図 2 のようなクラスタ内ノードを高速結合したネットワークハードウェアで，Tandem の ServerNet[6],Myricom の Myrinet[7], 富士通の Synfinity-0[8](片方向通信で測定した bandwidth が 240MB/s) などがある．SAN の主な特徴は次の 3 つが挙げられる．

1. 異なるノードの物理メモリ間で CPU を介さずに直接データ転送が行える．
2. ハードウェアレベルで信頼性の高い通信が行える．
3. カーネルを介さずに通信処理のためのハードウェア起動が行える (ユーザレベル通信)．

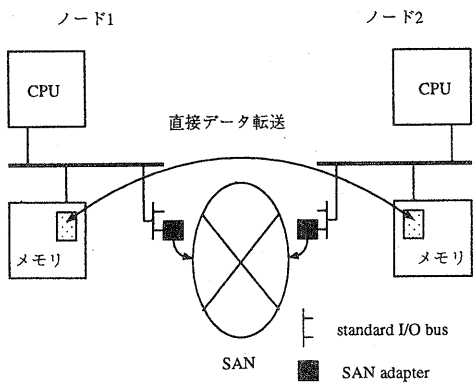


図 2 SAN

我々は，カーネルモジュールとしてプロトコル切り替えを行うので3番目の特徴は利用しない．送受信時のシステムコールのオーバヘッドは削減できないが，3.2で述べるようなSAN上の全てのアプリケーションを低オーバヘッド化できるメリットを考えると適当な選択と言える．

## 2.2 VIA

VIAはSAN上の様々な通信アーキテクチャを，統一標準化するために提唱されたアーキテクチャで，COMPAQ,Intel,Microsoftなど多くの企業がサポートしている[2]．

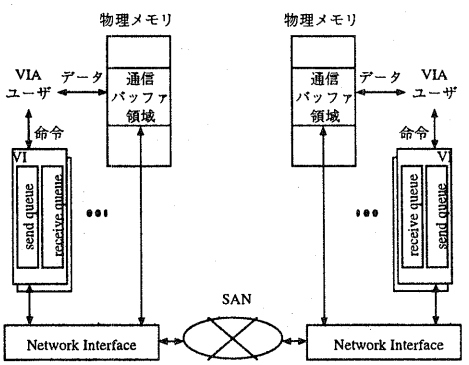


図3 VIA

VIAの構造を図3に示す．通信エンドポイントとしてVIを作成しコネクションを開設して通信を行う．VIごとに送信キュー・受信キューを持っており，送受信命令(ディスクリプタ)をキューイングして非同期に処理を行う．

VIAのカーネルインタフェースであるVIAインタフェースを利用すると次のようなメリットがある．

**・SANの直接使用による不要なオーバヘッドの削減**

通信するデータ領域をVIAの通信バッファ領域として登録することで，データコピーをせずにCPUを介さない直接データ転送を行うことができる．また，不要なソフトウェア処理をせずに，ハードウェア提供の機能を利用した信頼性のある通信が行える．

**・非同期通信**

送受信命令をキューイングすることで，非同期に送受信処理が行える．

**・多重通信路**

VIによってエンドポイントを多重化することで，複数の独立した仮想通信路(VI接続)を確保できる．

**・ポータビリティ**

標準的な通信アーキテクチャ上のインタフェースなので，プラットホームの変更に対して修正を最小限に済ませることができる．

## 2.3 socket通信オーバヘッド

SANのように通信ハードウェアが高速になるほど，ソフトウェアによる通信オーバヘッドが性能ボトルネックとして表れる．そのほとんどがTCP/IPであることをCullerらは示している[9]．

我々はUltraSPARC-II，Solaris2.7，256MBメモリ上で，100MbEtherの1Bping pong通信のオーバヘッドを解析した．SUNのプロファイルツールgprof[10]を用いてカーネル関数プロファイルした結果を表1,2，図4,5に示す．SANではなく100MbEtherを使用しているのは，データサイズの小さい1Bping pongでは，100MbEther上のTCP，UDP/IPの方が低オーバヘッドであるからである．

片側のノードの結果を示している．RTTはTCPが226 $\mu$ s，UDPが217 $\mu$ sである．under TPIというラベルは，TPIより下位モジュールの処理オーバヘッドで，TCP or UDP，IP，割り込みを含めたDLPIドライバの処理を示している．over TPIというラベルは，TPIより上位モジュールの処理オーバヘッドで，システムコール，Streamヘッドを含めたファイルシステム内の処理を示している．ただしmutexのコストや，STREAMSメッセージ作成に使われるallocb()のコストは，全てover TPIに入っている．

TCPの場合TPIより下位モジュールの処理は，58 $\mu$ sで全体の54%を占めることがわかった．中でもtcp_rput_data()という関数の処理が7 $\mu$ s程度で最もコストが大きく，STREAMSメッセージを作成し上位モジュールへ受信データを渡す処理と，ackを返すための処理を行っていると思われる．その他，チェックサムの計算やack待ち処理としてのtimerの設定などが上位を占めることがわかった．UDPの場合は，tcp_rput_data()のようなコストの大きい処理がないためTCPの場合よりも

割合が低いが，それでもチェックサムの計算などにより39 $\mu$ sで全体の39%を占めることがわかった．

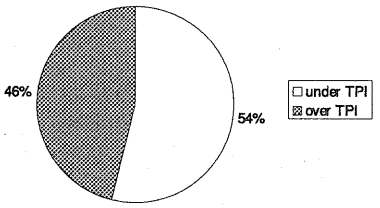


図4 TCPの通信オーバヘッド分析

表1 TCPの通信オーバヘッド分析

| | 時間 | 比率 |
|---|---|---|
| under TPI | 58us | 54% |
| over TPI | 49us | 46% |
| 全体 | 107us | |

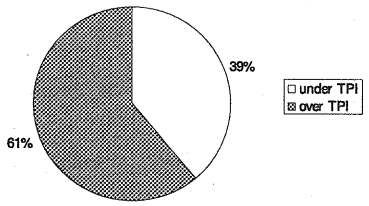


図5 UDPの通信オーバヘッド分析

表2 UDPの通信オーバヘッド分析

| | 時間 | 比率 |
|---|---|---|
| under TPI | 39us | 39% |
| over TPI | 61us | 61% |
| 全体 | 100us | |

# 3 VI socketの提案

我々は2.3で解析した結果を元にVIsocketの提案を行う．まず全体構成を説明し，その後でモジュール部，ドライバ部を個々に説明する．最後にVI socketによる性能改善見積りを示す．

## 3.1 全体構成

全体構成は図6のようになる．STREAMS機構を利用した構成になっている．

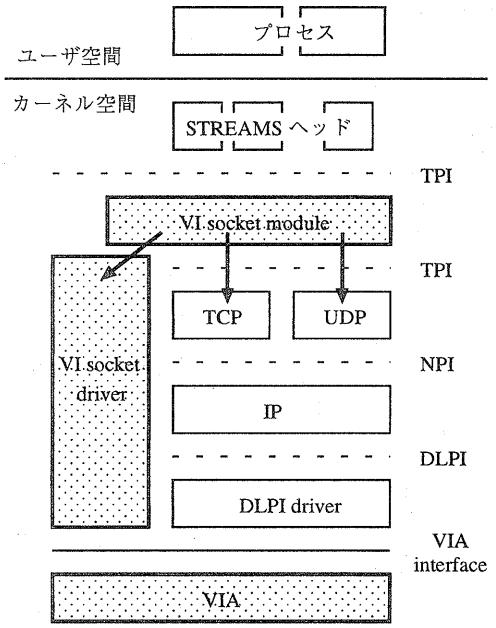


図6 VI socketの全体構成

STREAMSは，ユーザ空間のプロセスとカーネル空間のSTREAMSドライバを結ぶ全二重双方向のデータ転送路で，TPI，NPI，DLPIというインタフェースメッセージの交換でモジュール間のやりとりを行う[3][4][5].

我々はVI socketをモジュール部とドライバ部に分離する．モジュール部はプロトコル切り替えを行う．ドライバ部はTPIとVIAインタフェースとのマッピングを行う．モジュール部はSTREAMSモジュールであるが，ドライバ部は性能を考慮して，モジュール部とのやりとりを関数呼び出しで行うために，cdevドライバ[11]となっている．

## 3.2 プロトコル切り替え

SAN上のアプリケーションはSAN内だけでなくSAN外とも通信する．SAN外との通信は従来のTCP,UDP/IPを使わなければならないので，プロトコル切り替えが必要である．

我々はプロトコル切り替えをStreamヘッド下にpushしたSTREAMSモジュール(VI socketモ

ジュール部) で行う．VI socket インストール時に与える SAN の情報と，通信時の STREAMS メッセージの宛先をチェックして，SAN 内であれば低オーバヘッドプロトコルの関数を呼び出し，SAN 外であれば TCP,UDP へ STREAMS メッセージをスルーする．

ユーザ空間やファイルシステムでプロトコル切り替えを行えば更にオーバヘッドを削減できるが，以下のような問題があるため開発コスト，品質，保守性，ポータビリティなどを考慮し本方式を選択した．

**・ユーザ空間での切り替えの問題**

この方式ではシステムコールやファイルシステム (Stream ヘッド部に相当) などのオーバヘッドが削減可能である．David E.Culler らの Fast socket で用いており以下のような問題を抱えている [9]．

低オーバヘッドプロトコルを利用するためにユーザライブラリをリンクする．アプリケーションに対し透過的に切り替えるためには，従来の socket fd と同様なセマンティクスを持つ fd を実現する必要がある．そこで低オーバヘッドプロトコルを使用した通信でも，socket() や accept() 時に作られる従来のセマンティクスの fd を，shadow socket fd としアプリケーションに使用させる．このとき shadow socket は実際のエンドポイントではないので，実際のエンドポイントと結び付けるために，他の fd と区別して自分が shadow socket fd であるなどの情報をユーザライブラリに持たせる．

ところが従来の socket fd は，独立したプロセス間で交換され共有されることが可能である．もしユーザライブラリをリンクしたプロセスと，そうでないプロセスで fd 交換が行われるようなアプリケーションがあれば，管理情報を保持できないので対応不可能になる．

改善策として我々は，cdev ドライバを作成し，カーネル内で VIA を使用する方法を考えた．低オーバヘッドプロトコルを使用する場合は，cdev file を open() して fd を作成しアプリケーションに使用させる．以降この fd を使用した処理は OS により全て cdev へ渡されるので，ユーザライブラリに管理情報を保持する必要がない．

しかし実際に試作してアプリケーションを動かすと，様々な細かい機能対応洩れがあることがわかった．その原因は，socket 通信は非常に多様なオプションを設定することができ，実際にアプリケーションごとに様々な設定がされているからである．これらの機能を全て事前に調べ上げて対応するのは困難である．従来ではオプション設定の対応処理は，ほとんどファイルシステム (Stream ヘッド部に相当) 内で行われており，カーネル内で切り替えを行えば対応する必要がない．よって我々はユーザ空間での切り替えを行わないことにした．

**・ファイルシステム内での切り替えの問題**

この方法はユーザ空間での切り替えで問題になった file descriptor や多様なオプション機能への対応といった問題はない．システムコールのオーバヘッドは削減できないが，ファイルシステム内の処理を削減できるだけでもかなりの性能向上である．しかし互換性という点で大きな問題がある．Solaris はファイルシステム内のコードを公開していない．仮に許可を得て設計，実装を行ったとしても，OS のバージョンアップや他のプラットホームでの運用に小規模な修正だけで対応できることは望めない．

一方 Stream ヘッド下では，公開された十数種類程度の TPI フォーマットのメッセージを制御するだけである．よって我々は Stream ヘッド下で切り替えを行う．これにより，低オーバヘッドなプロトコルを透過的に組み込むための枠組みができたことになる．

### 3.3 TPI-VIA インタフェースマッピング

2.3 で述べた通信オーバヘッドの削減を行うには，従来よりも低オーバヘッドなプロトコル処理をしなければならない．

従来では DLPI から VIA インタフェースのマッピングを行っていた．これに対し我々は TCP, UDP/IP のモジュールを排除し，TPI と VIA インタフェースを直接マッピングすることで低オーバヘッド化を行う (VI socket ドライバ部)．TCP,UDP/IP スタックを使わない代わりに従来機能である信頼性のある全二重バイトストリーム，全二重データグラム通信 [1] を行い，バッファ管理やフロー制御を行う．

この方法には次のようなメリットがある．従来のプロトコルスタックに一切関係なく，VIA インタフェースを使用した独自プロトコル処理が行える．SAN 上の不要なデータコピーの削減，不要な

信頼性保証のための処理の削減はもとより，従来モジュールのロジック処理の最適化まで行える．

これにより，2.3 で述べた under TPI の処理は削減され，TCP，UDP でそれぞれ 58 $\mu$ s，39 $\mu$ s であった処理が 20 $\mu$ s で行える見積りになる．よって全体の通信オーバヘッドは TCP の場合で 36%，UDP の場合で 19%削減できることになる．

また，バイト数が増えるとプロトコル内で行っているデータコピーコストが増えるので，VI socket は copyin， copyout しかデータコピーを行わず，改善効果はより大きくなることが予想される．

# 4 設計と実装

提案した VI socket について，実現性や有効性を実証するために設計・実装を行った．その中でも特に性能に関連するデータコピー削減と送信データの一括処理について説明する．まず全体としてドライバ部の設計概要を説明し，その後で個々に詳細説明する．

## 4.1 ドライバ部の設計概要

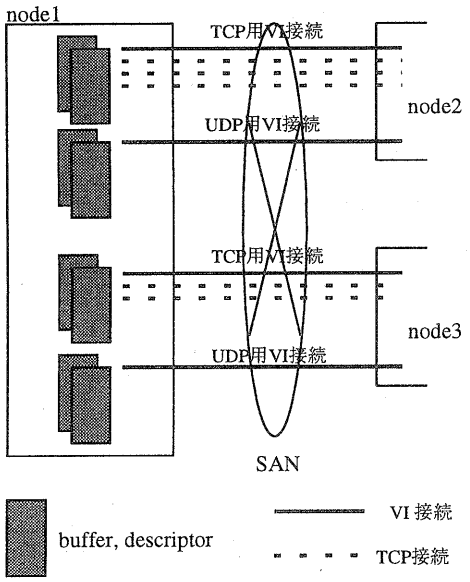


図 7 ドライバ部の設計概要

ドライバ部の設計概要は図 7 のようになる．拡張性を考えて VI を多く使わないように，リモートノード単位に TCP 用，UDP 用の VI 接続を確立し，socket 間で共用する．SAN 上に N 個のノードがあれば，各ノードで VI は 2 * (N - 1) 個用意することになる．送受信バッファ，送受信ディスクリプタなど他のリソースも同様な理由で VI 単位に用意し共用する．

socket 間で共用しているので，受信データが到着した場合は，受信通知先を判別しなければならない．従来の場合 TCP は，IP アドレスと port 番号の 4 つ組で socket を判別していたが，我々は TCP 接続時に接続識別子というインデックスを交換することによって効率的に判別を行う．送信時に相手先の接続識別子をヘッダに付加し，受信時にヘッダの接続識別子から即時に通知先の socket を判別する．

フロー制御はチケット交換で行う．送信側はチケットを獲得して送信する．即時に獲得できない場合はキューイング (VI の送信キューではなく，VI socket が管理するキューへ) し非同期に送信する．受信側はデータをバッファから読みとるとチケットを増やし，piggy back もしくはチケット通知メッセージで送信側へ送る．

## 4.2 データコピー削減

従来の TCP，UDP では copyin，copyout 以外に，受信バッファから copyout 用のバッファへデータコピーを行っている (図 8)．copyin，copyout はファイルシステムの処理なので我々も削減できないコストであるが，それ以外に不要なデータコピーをしないように次のような処理をする．

VI socket ドライバ部へ送信データとして渡される STREAMS メッセージのデータバッファを，VIA インタフェースを使って通信バッファ領域として登録し，データコピーなく送信する．ただし短いデータの場合は登録コストの方が大きいので，予め用意した通信バッファ領域にコピーして送信する．

受信は，VIA インタフェースを使って，予め受信バッファを通信バッファ領域として登録しておく．受信後 esballoc() を使って，受信バッファを STREAMS メッセージのデータバッファとし Stream へッドへ渡す．データバッファは，STREAMS メッセージ解放時の callback 関数で，copyout 後再び受信バッファとして利用される．

これにより，copyin copyout 以外の不要なデータコピーを削減できたことになる．データサイズ

が大きな通信では，コピーコストが通信オーバヘッドのほとんどであるので，3 回のコピーが 2 回のコピーに減ったことは，従来の 2/3 に通信オーバヘッドを削減したことになる．

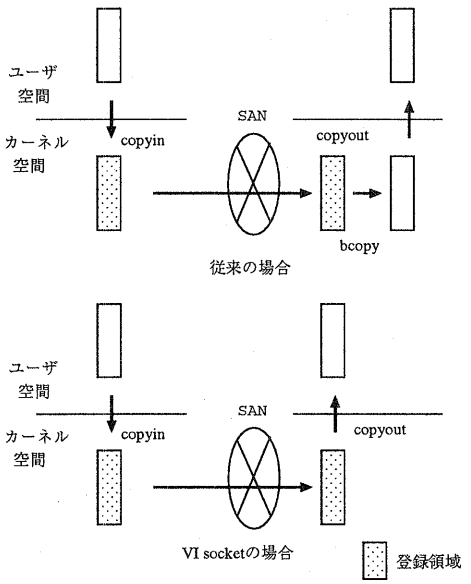


図 8 データコピー削減

### 4.3 送信データの一括処理

TCP ではフロー制御で送信データ (STREAMS メッセージ) が蓄積された場合，複数メッセージを一括して処理する場合がある．VI socket でも同様な処理を行い band 幅性能を向上させる．

そこでまず，図 9 のように送信命令 (送信ディスクリプタ) に複数バッファを指定できるようにしておく．これにより複数メッセージを改めて 1 つのバッファにデータコピーすることなく一括して送信できることになる．

さらに，VI socket ではハードウェアの処理状況に応じて効率的に転送を行う．まず，VI にキューイングされている送信ディスクリプタ数をカウントしておく．送信ディスクリプタをキューイングする前にカウンタをチェックし，一定値以上の場合はハードウェアの処理が追いついていないので，キューイングせず VI socket が制御できる状態に keep しておく．keep した送信ディスクリプタは，既にキューイングされている送信ディスクリプタが完了すると，その割り込みコンテキストでキューイングされる (図 9)．

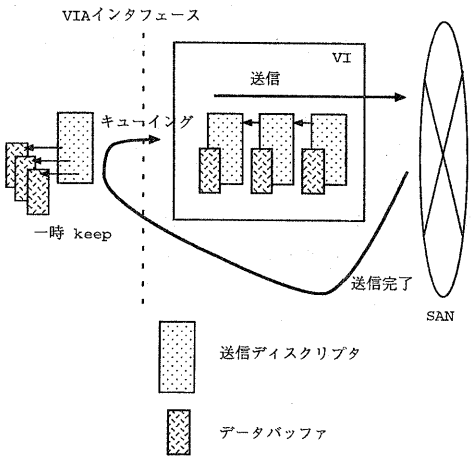


図 9 送信データの一括処理

このように，ハードウェアの処理状況に応じて送信ディスクリプタを制御できる時間を増やしておき，より多くのメッセージを一括して送信できるようにする．

## 5 関連研究

David E.Culler らは，Fast socket というユーザライブラリによるプロトコル切り替えで socket 通信の高速化を行っている [9]．ユーザライブラリであるために，3.2 で述べたような 2 つのプロセスで socket 共有できないなどの制限がある．SAN に Myrinet を使用し，UltraSPARC1 上で 60 $\mu$ s の RTT，33MB/s の最大バンド幅を実現している．

また Microsoft では，WSDP(Windows Socket Direct Path) という我々と同様なカーネル内でのプロトコル切り替えで，Windows 上の SAN の socket 通信高速化を行っている [12]．

## 6 結論

本論文では SAN 上の socket 通信オーバヘッドを透過的に削減する VIsocket の提案を行った．

VI socket ドライバ部は，SAN の VIA インタフェースと STREAMS の TPI をマッピングすることで，不要な処理を削減し低オーバヘッドなプロトコル処理を実現した．VI socket モジュール部は，Stream ヘッド下でプロトコル選択することで，全てのアプリケーションに透過的に低オーバヘッドプロトコルを導入した．

さらに提案の実現性や有効性を実証するためにVI socketの設計・実装を行った．現在は評価中であり，1BのRTTで従来の36%程度の性能改善を予測している．

一般的な通信インタフェースであるsocketを透過的に低オーバヘッド化できたことで，SANクラスタを用いた高機能・高性能なサーバシステムの構築を容易化することができた．

今後はVI socketの性能を解析し性能改善に向けた実装レベルのチューニングを行う．また従来のsocket通信機能が保証されているかどうかの検証も合わせて行う．

## 参考文献


[1]Stephen A. Rago; "UNIX System V Network Programming"; Addison-Wesley, 1993

[2]Intel Corp; "Intel Virtual Interface (VI) Architecture Developer's Guide Revision 1.0"; September 9.1998

[3]B. Goodheart, J.cox; "Magic Garden Explained: The Internals of UNIX System V Release 4 An Open Systems Design"; Prentice Hall, 1994

[4]Sun Microsystems Inc; "STREAMS Programming Guide"; Octorber 1998, http://soldc.sun.com/developer/support/driver/index.html

[5]The Open Group; "Transport Provider Interface (TPI)"; 1997, http://www.opengroup.org

[6]Robert W.Horst,David Garcia; "ServerNet SAN I/O Architecture"; Hot Interconnects V,August 1997 ,http://www.tandem.com/Library.asp

[7]Nanette J.Boden et al; "Myrinet–A Gigabit-per-Second Local-Area Network"; IEEE Micro, Feburary 1995,http://www.myri.com/research/publicat ions/index.html

[8]O.Shiraki et al; "AP-NET advanced high- performance network for scalable parallel server"; In Hot Interconnects, 1996

[9]Steven H.Rodrigues, David E. Culler; "High-Peformance Local Area Communication With Fast Sockets"; Prentice Hall, 1994

[10]Sun Microsystems, Inc. gprof manual page. Solaris 2.7 manual page.

[11]Sun Microsystems Inc; "Writing Device Drivers "; Octorber 1998, http://soldc.sun.com/developer/support/driver/index.html

[12]Microsoft Corporation. "Windows Sockets Direct Path For System Area Network"; WinHEC'99, 1999