Ethernetマルチリンクによる PC クラスタ向け高バンド幅・耐故障ネットワーク RI2N/UDP

岡本高幸[†] 三浦信一[†] 朴泰祐[†]
佐藤三久[†] 高橋大介[†]

PC クラスタはその対価格性能比の高さから高性能計算分野で広く用いられている．PC クラスタのネットワークとしては Gigabit Ethernet が用いられることが一般的であるが，その一方で特に大規模化を考えた場合，その信頼性は必ずしも高くない．ハードウェアの完全な故障以外にもネットワークスイッチの一時的な不具合などが問題となる．これを解決する手段として，冗長リンクを利用することで Ethernet を使って高バンド幅で耐故障性のあるネットワークを構築する RI2N というシステムが提案された．本論文では RI2N の高バンド幅と耐故障性をユーザレベルで実現することを目的とし，その実装システムとして UDP/IP を利用した RI2N/UDP を設計・実装する．実装したシステムのネットワーク性能と耐故障性について評価し，Gigabit Ethernet リンク 2 本を使用して最大 246 MB/s のバンド幅と 2 本のうちのいずれかが故障していても通信を継続する耐故障性が実現できていることを確認した．

RI2N/UDP: High Bandwidth and Fault-tolerant Network for PC-cluster Based on Multi-link Ethernet

Takayuki Okamoto,[†] Shin'ichi Miura,[†] Taisuke Boku,[†]
Mitsuhisa Sato[†] and Daisuke Takahashi[†]

The PC-cluster has been used for high performance computing with its high performance/cost ratio. To save the cost, Gigabit Ethernet is used mainly for intercommunication network. However, the reliability of Ethernet is not quite high because of not only the hardware failure but also tentative error on network switches. To solve this problem, we have proposed an interconnection network system based on multi-link Ethernet named RI2N. In this paper, we develop a user level implementation of RI2N with UDP/IP named RI2N/UDP. Through the evaluation on the performance and fault-tolerance, it is shown that the bandwidth on dual-link Gigabit Ethernet is 246 MB/s and our system can keep the computation on the network link failure to provide a high reliability on the system.

1. はじめに

クラスタの性能を大きく左右する要素として，ノード間をつなぐネットワークの性能があげられる．Myrinet [1] や Infiniband [2] などの System Area Network（以下，SAN）と呼ばれるネットワークは，高バンド幅と低レイテンシのための設計がなされており，大規模な並列数値計算を行うことを目的する HPC クラスタで利用されている[3]．しかし，一方でこれらのネットワーク製品は一般への需要が少なく，非常に高価であるという欠点がある．そのため，特に高性能化・大規模化を必要としない多くの PC クラスタではネットワークとして Gigabit Ethernet（以下，GbE）を利用している．GbE は SAN に比べてバンド幅やレイテンシなどの性能面では劣るものの，一般に広く普及しているため導入のコストが非常に小さい．

一方，クラスタ向けのネットワークとしては性能だけでなく信頼性が高いことも重要である．計算の最終段階でのネットワークエラーによってすべての計算が無駄になる可能性もあり，特に長時間の並列処理を要求するアプリケーションではネットワークの信頼性が問題となる．そのため，クラスタの通信における耐故障性を向上させる研究が行われている[4)~6)]．ネットワークはコンピュータの筐体から外に出ている部分であり，ケーブルが抜ける，断線するといった故障の発生しやすい部分である．また，ケーブルだけでなくそれを束ねるスイッチが故障する場合もある．Ethernet

† 筑波大学大学院システム情報工学研究科
Graduate School of Systems and Information Engineering, University of Tsukuba

スイッチの故障はネットワークの集約点での故障であるためそこに接続されたノードすべてに被害が及ぶ．特に Ethernet スイッチでは，そのプロトコル上の特性から，通信の集中などによって著しく性能が低下し，ハードウェア的には故障していなくてもリセットなどの措置が必要となる一時的な故障（soft failure）が多い．このように，ネットワークの故障はノード内での故障に比べて比較的起こりやすく，そこに耐故障性を持たせることは，クラスタ全体の耐故障性を向上させるうえで重要である．

そこで，ネットワークを冗長化することによって高性能化と信頼性の向上を同時に実現する手法として RI2N（Redundant Interconnection with Inexpensive Network）[7] が提案された．これは，複数の Ethernet リンクを束ねて 1 つの論理ネットワークとしてユーザプログラムに提供することにより，リンクが正常な間はそれらを同時に利用してバンド幅の向上を行い，また一部のリンクに故障が発生した場合にも他のリンクを利用して安定した通信を提供し続けることのできるネットワークである．RI2N の実装システムとしては，高バンド幅化のためのプロトタイプが作成された[7]．しかし，作成されたプロトタイプは TCP/IP のマルチストリーム上に RI2N ストリームを構築する実装方法であったため，故障の検出が行えないなどの問題が発生し故障時の性能が低下していた．

そこで我々は，耐故障性に重点を置いた RI2N のユーザレベル実装システムとして RI2N/UDP を作成する．本論文では，まず RI2N/UDP の設計について述べ，次に実装段階での種々の問題について述べる．そして，RI2N/UDP の耐故障性とネットワークとしての性能について評価と考察を行う．

2. RI2N/UDP

2.1 概　　要

RI2N/UDP は UDP/IP を用いた RI2N の実装システムである．Ethernet のマルチリンクを利用して高バンド幅で耐故障性のあるネットワークをユーザプログラムに対して提供する．高バンド幅化は複数のリンクを同時に利用することによって実現する．ユーザプログラムから与えられたデータを細かなパケットに分割して複数のリンクに送出することで，単位時間に送信できるパケットの数をリンク数倍に増加させることができる．また，RI2N ではマルチリンクの一部が故障したとしても残りの正常なリンクを使って通信を継続する耐故障機能を提供する．

2.2 実 装 方 式

一般に，このようなネットワークシステムの実装としてはデバイスドライバあるいは OS カーネルレベルでの実装（システムレベル）と，ソケットなどの API を利用したネットワークプログラミングによるライブラリとしての実装（ユーザレベル）が考えられる．システムレベル実装では，仮想ネットワークデバイスとしてユーザに提供でき，またシステムイベントを低コストでキャッチできるため，透過性が高く効率的な実装が可能である．しかし，その反面でシステムへの依存性が高くなり開発コストも大きくなるという問題がある．これに対し，ユーザレベル実装は移植性の高さという大きなメリットがあり，POSIX などの標準にあわせることによりユーザに負担をかけることなく多くのシステムで利用することができる．そのため，本研究では RI2N/UDP をユーザレベルのライブラリとして実装する．

2.3 UDP/IP の利用

HPC アプリケーションの並列処理では，ストリーム通信の高速化と信頼性向上が重要となる．Ethernet 上でのストリーム通信では多くの場合 TCP/IP が用いられる．Ethernet を用いた HPC クラスタでもストリーム通信に TCP/IP が用いられる場合が多い．しかし，一般に Ethernet 上での TCP/IP 通信では，ソケットによる固定的な channel を張るため，故障時にプログラムそのものがロックすることや，故障の検出が行えなくなることがある．これまで RI2N のプロトタイプは複数の TCP/IP チャネルを束ねた上に実装されているがそのままの拡張で TCP/IP 上に耐故障性を実現させることは難しい．そのため，UDP/IP を用いて RI2N を実装する．RI2N の耐故障性を実現するうえで最も重要なことは，ネットワークに故障が発生してもシステム自体がその影響を受けず安定して動作し続けられることである．UDP/IP はコネクションレス型であり，元々データの到着を保障していないため，ネットワークが故障しても単にパケットロスが発生するのみでシステムにはまったく影響がない．ネットワークインタフェース自体がシステムから除かれることや IP ルーティングのテーブルが変更されることなどの要因で送出先の NIC（Network Interface Card）自体が見つからなかった場合には UDP/IP でもエラーが発生する．しかし，RI2N/UDP がターゲットとしているのは主にスイッチの故障であり，そのような NIC や PC 自体の故障については適用範囲外であると考えている．

TCP/IP 上に RI2N を実装する場合，IP 層以下で

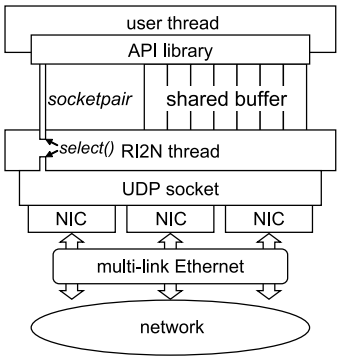


図 1 RI2N/UDP の構成
Fig. 1 Construction of RI2N/UDP.

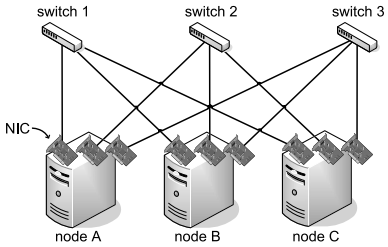


図 2 ネットワークの構成
Fig. 2 Network topology used by RI2N/UDP.

ロストしたパケットはOSカーネルのTCP実装によって自動的に再送が行われる．そのため通常のパケットロスに対して上位層のRI2Nが再送をする必要はない．しかし，故障が発生した場合にはTCP/IP上でもロストデータの再送が必要になる．これは，故障したリンクのソケットバッファ上のデータが，上位層には送信が完了しているように見えるためである．しかし，実際にはソケット自体が破棄されてしまうために送信できていない可能性がある．そのため，TCP/IP上にRI2Nを実装したとしても，耐故障性のためにデータの到達確認と再送をしなければならない．UDP/IP上での実装を考えた場合，元々UDP/IPには自動的に再送や到達確認を行う機能がないためそれらをすべてRI2N上で行うことになる．しかし，前述のようにTCP/IP上に実装したとしてもやはり同様の処理が必要になる．そのため，通常のパケットロスと故障の両方に対する再送を同様に扱ってUDP/IP上に集約して実装する方が効率的であると考えられる．

3. 設　　計

3.1 システム構成

図 1 にRI2N/UDPの構成を示す．RI2N/UDPではTCP/IPと同様にコネクションの管理やロストパケットに対する再送を行う．これらはユーザプログラムと並行して処理する必要があるためユーザプログラムとは別に新たに通信用スレッドを作成し利用する．元々はユーザプログラムであった主に計算を行うためのスレッドをユーザスレッド，RI2N/UDPの通信を管理するスレッドをRI2Nスレッドと呼ぶ．RI2Nスレッドはノード内のユーザスレッドとの間で通信を行い，また，ネットワークを介して他のノードのRI2Nスレッドとも通信を行う．

3.2 ネットワークの構成

図 2 にRI2N/UDPが想定しているネットワークの構成を示す．各ノードに複数のNICを用意してNICの冗長化を行い，それらを互いに独立したネットワークに接続する．スイッチに故障が発生するとすべての通信が停止するためリンクごとにそれぞれ別のスイッチを使用しスイッチも冗長化する．また，スイッチによって物理的に分割するだけでなく割り当てるIPアドレスもリンクごとに異なるネットワークアドレスにする．図1のとおりRI2N/UDPではUDPソケットを使って通信を行うため，NICのMACアドレスなど物理的な情報を使ってリンクを選択することができない．そこで，OSのルーティング機能を利用する．IPルーティングではNICに割り当てられたネットワークアドレスに従ってルーティングが行われるため，複数のNICをそれぞれ異なるネットワークアドレスに設定する．このようにすれば，UDPソケット上から使用するリンクを間接的に選択することができる．

3.3 RI2N プロトコル

RI2N/UDPではノード間の通信に専用のプロトコルを用いる．これをRI2Nプロトコルと呼ぶ．RI2NプロトコルはUDP上に実装したTCP[8),9)]に近いものであるが，RI2N/UDPではマルチリンクを前提としており，また故障時の性能についても考慮する必要があるため，いくつかの点でTCPとは異なっている．

3.3.1 到達確認と再送制御

RI2N/UDPではノード間のバンド幅向上を行うため宛先によって使うリンクを分けるというような静的なリンク決定は行わず，round-robinで順に使用するリンクを変更してパケットを送信する．そのため，パケットの順序が数個分ずれて受信側に到着することがシングルリンクの場合と比べて非常に多い．また，故障時やsoft failure時にはパケットを1つおきや2つおきに落としてしまう．このような状態で，古くからのTCPと同様にある番号から後のパケットをすべて再送してしまうと，すでに受信できているパケットに

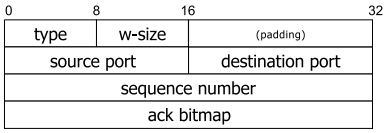


図 3 選択的確認応答パケットの構造
Fig. 3 Header of acknowledgement packet.

ついても再送が行われ効率が悪い☆．そのため RI2N プロトコルでは，この性質を考慮してより大きな到着順序のずれを許容し，また到達確認応答には選択的確認応答を用いることによって再送するパケットの数を削減する．

RI2N プロトコルでは到着順序がどの程度ずれているかの確認を受信側で行う．受信したパケットはすぐに受信バッファには展開せず，パケットの順序がそろうまでは一時的な並べ替えリングバッファに保管する．そして，ある閾値を超えてずれるとそのパケットは lost したものと判断して再送の要求を行う．この到達確認応答に使用するパケットの構造を**図 3** に示す．TCP とは異なり RI2N プロトコルのパケットには，確認応答番号専用に使うフィールドを用意しない．データパケットではそのパケットのシーケンス番号を格納するために sequence number フィールドを用いるが，同じフィールドを確認応答パケットでは “次に送信することを要求するパケットのシーケンス番号”（以下，ACK 番号）を格納するために使う．また，ack bitmap フィールドは選択的確認応答のパケット列情報を格納するためのフィールドである．ACK 番号以降のパケットについて，各パケットがすでに到着したか否かを各々1 ビットを使って表現したビットマップを格納する．

3.3.2 フロー制御

TCP/IP は LAN から WAN までのさまざまなネットワーク環境を想定しているため，比較的複雑なフロー制御を実装している[10]．クラスタのネットワークプロトコルとして TCP を利用する場合，この実装が問題になることがある．たとえばいったんネットワークスループットが低下するとそれを高いものに復帰させるためのオーバヘッドが，いわゆるスロースタート問題として表面化する．また，クラスタ上で並列数値計算を行う場合，ネットワークトポロジを考慮して衝突が発生しないように通信パターンを決定する場合が少なくない．そのような場合に TCP の複雑なフロー制御はかえって全体的なネットワークスループットを低下させる可能性がある．RI2N/UDP は閉じたクラスタ環境でのネットワークとして利用されることを前提とするため，TCP/IP のような手厚いフロー制御はせず，通信端の間での単純なウインドウ制御のみを行う．パケットに window-size を格納するためのフィールド（図 3 の w-size）を用意し，これを使って通信相手に自分の受信バッファの空き容量を通知することで，送信するパケットの量を制限する．

3.3.3 故障と回復の検出

RI2N/UDP では耐故障性を実現することを最も重視している．しかし，故障がない正常状態の通信でも，スイッチでの衝突などによりパケットロスは発生する場合があり，それを補完するために前節のように再送処理を行っている．ネットワークが故障した場合でも正常に動作するリンクが 1 つ以上ある限り，この再送処理によって通信を継続することが可能である．しかし，この段階では特定のリンクが故障しているか否かを判断するわけではないため，round-robin による送信リンク選択の結果，次の送信機会でそのリンクにもパケットを送出することになる．故障したリンクへ送出したパケットはすべて消失されるため，パケットの到達確率が低下し通信効率が著しく低下する．これを回避するためには，早期に故障したリンクを検出し，そのリンクでの送信を停止しなければならない．

RI2N プロトコルでは故障の検出をパケットの受信側で行う．まず，受信側ではどのリンクからどれだけのパケットが到着したかをつねに記録しておく．送出リンクは round-robin で選択されているため，すべてのリンクが正常である場合にはどのリンクに到着するパケットの数もほぼ等しくなる．この値が相対的に見て非常に少ないリンクがあればそれは故障もしくはそれに近いレベルのパケットロスを起こしているリンクであると考えられる．受信側でこのような評価を定期的に行って受信パケット数が相対的に少ないリンクを故障と判断する．そして，故障が検出された場合にはまずそのリンクでのパケット送出を停止し検出した故障情報を通信相手に知らせる．故障情報パケットではオプション領域に故障しているリンクのアドレス情報を格納する．このパケットを受け取った側はその情報を元に，リンクの使用と停止を判断する．そして，その確認のための応答パケットを送信する．この確認応答が届くまでは故障情報パケットも一定間隔ごとに再送し，故障情報パケットがロストして相手に故障情報が伝わらなくなることを防ぐ．

故障の検出と同様に故障が回復したことの検出も重要である．RI2N プロトコルではリンクが故障から回

☆ Linux kernel 2.4 以降では選択的確認応答が実装されている．

復し再び使用できる状態になったことも自動的に検出する．これも前述の故障検知とほぼ同じ手法で行う．故障検知のために各リンクに届いたパケットの数をカウントしていることは前述のとおりであるが，故障したリンクについてもこの処理を続けていく．また，1つでも故障したリンクがある状態では一定時間ごとにすべてのリンクに対して heartbeat パケットを送出するようにしておく．通常は故障リンクに対してパケットを送出しないため受信側でいくら待っていても受信パケット数は 0 のままであるが，このようにしておけば故障から回復したリンクからは何らかのパケットが受信できることになる．そこで，故障検出の場合と同様に定期的に各リンクからの受信パケット数を確認して，すでに故障と判断されているにもかかわらず受信パケットのあるリンクは故障から回復したリンクであると判断する．この回復情報も故障情報と同様に扱い，故障情報パケットを使って相手ノードに通知する．

4. 実　　装

4.1 スレッド間の同期

RI2N/UDP を Linux 上のユーザレベルライブラリとして実装する．スレッドの実装には pthread を用いる．pthread によるスレッドプログラミングでは，スレッド間の同期をとるために mutex や条件変数などの同期機構が用いられる場合が多い．しかし，RI2N スレッドではユーザスレッドからの要求とネットワークの状況を同時に監視していなければならない．ネットワークの監視は select() システムコールによって行うため pthread の同期機構による同期と相性が悪い．そこで，RI2N/UDP では socketpair を利用することでスレッド間の同期を行う．socketpair はファイルディスクリプタとして扱うことができるため select() システムコールによる監視ができる．しかし，スレッド間のすべてのデータ転送を socketpair を経由して行うとスループットの面で不利である．そのため，同期やメタ情報の通信は socketpair で行い，実際のデータの受け渡しは通常のメモリコピーによって行う．

4.2 スループットのチューニング

RI2N/UDP では select() システムコールによってネットワークとユーザスレッドの監視を行う．リアルタイム性を追求するのであれば 1 回の select() システムコールでは各ファイルディスクリプタとも 1 回しか入出力を行わず，毎回 select() システムコールによってすべてのファイルディスクリプタの状況を確認して処理を行うべきである．しかし，この方法では select() システムコールを頻繁に発行しなければならず，また

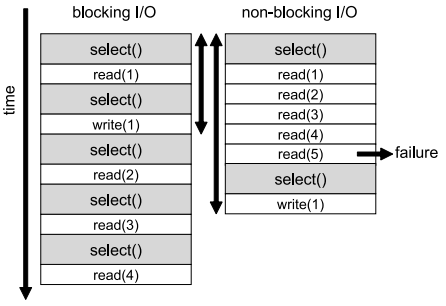


図 4 4 回の read() と 1 回の write() に要する時間
Fig. 4 The time taken for four times read() and once write().

今回使用した表 1 の環境では 1 回の select() システムコール呼び出しのコストは，fd_set の準備も含めて約 3 μs に相当する．1 回の select() に対して 1 回の read() や write() しか発行しなかった場合，select() によるオーバヘッドの割合が大きくなり GbE を束ねてその本数分に相当する性能を提供することは難しい．そのため，RI2N/UDP では非ブロッキング I/O と select() システムコールを組み合わせて利用する．非ブロッキング I/O を用いればファイルディスクリプタの状況確認と実際の入出力を同時に行うことができるため，状況確認に要する時間が隠蔽されオーバヘッドは非常に小さくなる．また，この方法では 1 つのファイルディスクリプタのみを調べるため select() よりも短時間で結果が得られる．実際に今回使用した環境で入力のないファイルディスクリプタにおいて recvmsg() システムコールに MSG_DONTWAIT オプションを付けて発行するために要する時間を測定したところ，約 0.5 μs で処理を終えていることが分かった．そこで，複数のファイルディスクリプタを監視する処理には select() システムコールを利用し，処理すべきファイルディスクリプタが決まってからの連続した入出力を制御するためには非ブロッキング通信を利用する．これによって，相対的に select() システムコールのオーバヘッドが減少しスループットが向上する．**図 4** にその様子を示す．図の左側が 1 回の read()，write() ごとに select() を発行するもので，右側のものよりも早く write() が発行できている．それに対して右側は，最初の select() で一度 read() できることが確認できた後は連続して非ブロッキングの read() を発行して 4 つ目のパケットまで受信している．図中では read(5) の失敗は他の read() システムコールと同じ時間かかるように描いているが実際には他よりも短い時間で終了する．しかし，それを考慮しなかったとしても，図のように連続してパケットを処理した方がより短時間

表 1 測定環境
Table 1 Environment for measurement.

項目	スペック
CPU	Intel Xeon 3.0 GHz 1-way（Hyper-Threading）
memory	DDR2 1024 MB
kernel	linux 2.6.17
NIC	Intel PRO1000MT dual port 1000base-T（PCI-X 64 bit/133 MHz）
NIC driver	Intel PRO/1000 Network Driver 7.0.33
switch	Dell Power Connect 5224（24 ports GbE switch）

の間に全体の処理を終えることができており，スループットの面では有利であることを意味している．

5. 評　　価

RI2N/UDP の実効スループットと耐故障性能を評価するため，ネットワークリンクが正常に動作している場合および故障している場合のスループットの変化の様子を実験によって確認する．またネットワークとしての基本的な性能であるレイテンシとバンド幅についても測定する．測定に用いた PC のスペックを**表 1** に示す．測定は，1-way Xeon マシン 2 台を 2 つの GbE リンクによって接続した環境で行った．

5.1 スループット

ネットワーク故障時にも安定して通信が続けられているか，故障と回復が正しく検出できているかを実験によって確認する．また，実効スループットについても評価する．今回は，これをスループットの変化によって確かめる．まず，1 方向のバースト転送を長時間続け 100 ms ごとに過去 100 ms 間のスループットを出力するプログラムを作成する．そしてそのプログラムを実行中に，使用している Ethernet ケーブルを抜いたり挿したりすることで意図的に故障を発生させ，故障時および回復時のスループットの変化を観測する．MTU は 6,000 byte で RI2N パケットのサイズは 5,950 byte とする．また，いずれかのリンクでパケットを 100 個受信したときに受信パケット数が 2 より少ないリンクを故障と見なすことにする．また，回復の検出に使用する heartbeat パケットの送出間隔は 3 秒間とする．

図 5 に測定結果を示す．横軸は時間，縦軸はその時点でのスループットを表している．グラフ下のバーは各リンクの状態（plugged=正常，unplugged=リンク切断）を表している．人の手による作業であるため時間はあまり正確ではないが，グラフ中の最初の時刻 t_1 で 1 つ目のケーブルを抜き，時刻 t_2 で 2 つ目のケーブルを抜いた．そして，時刻 t_3 で最初に抜いたケーブルを挿し，時刻 t_4 でもう 1 つのケーブルを挿した．

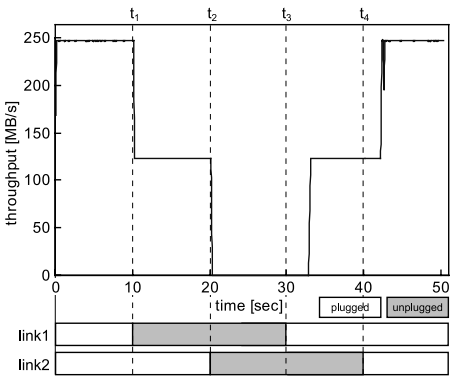


図 5 故障時のスループットの変化
Fig. 5 Throughput before and after failure.

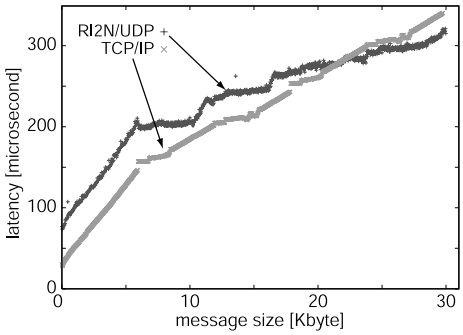


図 6 レイテンシの測定結果
Fig. 6 The result for latency.

スループットは最大で 246 MB/s であった．

5.2 レイテンシ

レイテンシはノード間でメッセージの ping-pong を 10 回行ってそれに要する時間から算出する．MTU は 6,000 byte で RI2N のパケットサイズを 5,950 byte にした場合，および TCP を用いた場合の測定を行う．

図 6 に測定結果を示す．横軸は ping-pong するメッセージの大きさ，縦軸はレイテンシである．複数回測定した結果，各回における値のばらつきがほとんどなかったため，図 6 にはそのうちの 1 回の測定結果をそのままプロットした．RI2N/UDP では最小で 72 μs，TCP では 28 μs であった．

6. 考　　察

6.1 耐故障機能

測定結果における故障および回復時のスループットの変化から，RI2N/UDP では 1 つでも正常なリンクがあれば通信を継続できていることが分かる．図 5 を見ると，時刻 t_1 でスループットが階段状に低下している．この時点で故障（ケーブルを抜くこと）を起こすため，これ以後は使用可能なリンクが 1 つになりス

ループットが低下している．しかし，故障後にも安定して 123 MB/s 程度のスループットで通信を継続している．このスループットは別途測定した TCP/IP での最大スループットと同等であり，故障検出によって設計どおりに故障リンクの使用を停止して，その状態での最大のスループットを維持できていることが分かる．

また，故障検知にかかる時間も十分に短いものであったといえる．予備実験によれば故障リンクの使用を停止せずに round-robin による送信選択から外さなかった場合のスループットは 10 MB/s 程度という非常に低い値であったが，図 5 を見る限りにおいてそのような測定点はない．これは，測定間隔の 100 ms よりも十分に短い時間内に故障が検出され 1 リンクでの動作に切り替えられたためであると考えられる．今回の実験では，パケットがいずれかのリンクに 100 個到着するごとに故障検出を行った．また，各リンクにはおよそ 1 Gbps でパケットが流れていたので，パケットサイズ 5950 byte でのパケットの到着間隔は約 50 μs である．よって，理論的にはおよそ 5 ms で故障の検出が可能であるということになる．実際に故障が発生してからそれが検出されるまでの時間を正確に測定することは困難であるため今回の測定でそれは行っていないが，heartbeat では困難な短時間での故障検出が実際に行えていることが確認できた．

回復の検出について見てみると，故障の検出ほど短時間では行われていないことが分かる．図 5 の時刻 t_3 で link1 が正常な状態に戻されているが，その後約 3 秒間スループットは 0 のままである．また，link2 を正常に戻した時刻 t_4 でも同様に，ケーブルを挿してからそれがスループットに反映されるまでに数秒程度の時間を要している．Ethernet のリンクアップ時には Auto-Negotiation など，Ethernet 自体の開始に必要な処理がありケーブルを挿した後も実際には利用できない時間がある．この時間は機器によって異なっていると考えられるが経験的には 1 秒から数秒程度である．このリンクアップ時間と heartbeat の間隔が重なって，今回のように故障の検出に 3 秒程度の時間がかかったのではないかと考えられる．もちろん回復の検出も短時間で行われることが望ましいが，故障時のように 10 分の 1 にまでスループットが低下しているわけではなく故障の検出に比べれば緊急性は低い．また，多くの場合故障からの回復には人の手による物理的な作業が必要になるため，人間の作業にかかる時間を基準とすればこの程度の間隔で回復が検出できていれば十分であると考えられる．

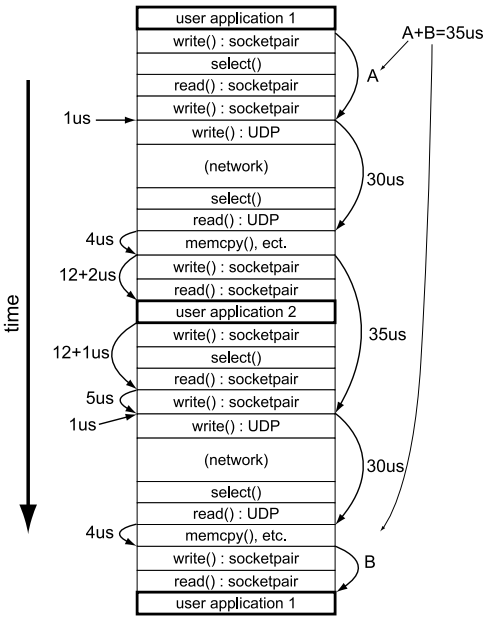


図 7 ping-pong 通信に要する時間の解析
Fig. 7 An analysis of latency.

6.2 通信性能

GbE リンク 2 本を用いたスループットの測定結果について見てみる．図 5 より，2 つのリンクが両方とも正常な場合には RI2N/UDP は最大で 246 MB/s のスループットが達成できていることを確認した．また，いずれか一方のリンクが故障している場合には最大で 123 MB/s であった．これらは GbE の理論ピーク性能（1 リンクあたり 125 MB/s）の 98% に相当する値であり，2 リンク使用時には 2 倍のスループットが実現できていることが分かる．今回の測定では 3 つ以上のリンクを束ねるような実験は行えていないが，少なくとも 2 リンクではマルチリンクによる高バンド幅化という RI2N の機能が実現できていることが分かる．

次に，レイテンシの測定結果について見てみる．片方向の通信にかかる時間は，最も短いもので TCP が 28 μs，RI2N/UDP が 72 μs であった．これらの差を検討するため，1 回の ping-pong 通信における処理時間の解析を行った．その結果を**図 7** に示す．図 7 は上から順に時間軸に沿って主に時間を消費している部分を並べ，実際に測定された消費時間を記入したものである．RI2N/UDP で要した 72 μs のうち約 35 μs が図 7 の左側に示した時間は，さらに細かく各システムコールの所要時間を測定した結果である．socketpair を 1 回利用すると write() と read() あわせておよそ 12 μs を要していた．しかし，実際の RI2N スレッド内

の処理では socketpair に対して write() システムコールを発行した後，select() システムコールを発行するまでの間に約 2 μs の処理（fd_set の準備やすべての RI2N ソケットの状態確認など）を行っており，これによってコンテキスト切替えが遅れるため実際には 14 μs を要しているものと考えられる．また，RI2N スレッドはユーザスレッドの ri2n_write 命令発行時に返り値をユーザスレッドに伝えるため再度 write() システムコールを発行する．これには約 5 μs を要する．ただし，この write() に対するユーザスレッド側の read() システムコールはネットワークを利用している時間で隠蔽されるためレイテンシには影響しない．また，select() システムコールで要求の検出を行うためには 1 回あたり約 1 μs を要する．これらの値を合計すると，受信側と送信側で socketpair を 2 回通過するため $12 \times 2 + 2 + 5 + 1 = 32\,\mu$s となり gettimeofday() の精度を考えた場合，マクロに測定した socketpair によるスレッド間の通信時間とほぼ一致しているといえる．

一方，RI2N/UDP よりも下層，つまり UDP 層以下の通信で消費される時間はおおむね TCP の通信時間に等しく，実測で約 30 μs であった．これは select() システムコールによるソケットの監視を含めた時間である．さらに，これら実際の通信にかかっている時間以外にも，ユーザスレッドから受け取ったデータをネットワークに送信するまでの間に 1 μs，パケットを受信してからユーザスレッドに渡すまでの間に 4 μs を要している．後者の時間が長いのはパケットの並べ替えやバッファへのコピーなどを行う必要があるためである．そのため，RI2N/UDP のレイテンシは 1+30+4+35=70 μs になる．実際には片道の通信に 72 μs を要しているがこれは測定誤差の範囲内であると考えられる．

解析したレイテンシのうち，ネットワーク通信に要している 30 μs については UDP/IP 以下のスタックを変更しない限り削減できないものである．socketpair による通信部分 35 μs については 7 章に述べる方法で 5 μs 程度削減できる可能性がある．残りの 4+15 μs の部分は本来通信を行っている時間ではなく，構造体のサーチやメモリコピーにほとんどの時間を費やしているものと考えられる．そのためこれらも，処理の省略，検索テーブルの利用などによって削減できる可能性がある．

以上の解析の結果，RI2N/UDP の片道通信レイテンシのうち，正味の通信（UDP 通信）以外の部分が全レイテンシの半分強であるということが分かった．その主要因は，RI2N スレッドの導入によるユーザスレッドとのやりとりのための socketpair 通信コストである．これは通信処理がアプリケーションと非同期に行われる必要があることと，すべてをユーザレベル実装としたことによるオーバヘッドと考えることができる．

また，図 6 から RI2N/UDP と TCP/IP の通信性能がクロスオーバするメッセージサイズを読み取ることができる．2 つのグラフが交差するのはメッセージサイズがおよそ 21 KB の部分で，22 KB 以上では RI2N/UDP の方が短い時間で通信を終えている．このメッセージサイズは double 型のベクトル変数を想定すると 2625 個になり，これは HPC アプリケーションとしてはリーズナブルな通信データ長であると考えられる．たとえば，複数ノードでの並列化を要求するような問題では，行列計算の 1 行分のデータにせよ，重力多体問題における粒子数にせよ，数万行あるいは数十万粒子といった大量のデータを扱うことはきわめて普通である．1 万要素の倍精度浮動小数点ベクトルは 80 kbyte に相当し，RI2N/UDP を用いる価値は十分にある．また実アプリケーションの例をあげるならば，たとえば超並列クラスタ PACS-CS [11] の中心的なアプリケーションの 1 つである QCD 計算では，ノード間通信における支配的なメッセージサイズは 70 KB 程度である．また，HPL [12] において支配的な，ピボット行データを他のすべてのノードに送る作業では，平均メッセージサイズは計算に用いる行列サイズ N の 2 分の 1 であり，N は 1 万以上であることが一般的である．RI2N/UDP をすべての HPC アプリケーションに対して有効に働くシステムと断言することは難しいが，高いスループットが要求される大規模科学技術計算においては高い性能が得られる可能性は十分にあると考えられる．

7. 実装方法の改良

RI2N/UDP は 4 章でも述べたように socketpair と呼ばれる全二重型の pipe と共有メモリを併用してスレッド間の通信を行っている．これは，データの転送と同期という 2 つの目的を分けることでスループットを向上させるためであった．しかし，今回のスループットの測定によりユーザレベルでも GbE リンク 2 本程度であれば十分に駆動可能であることが分かった．このことから，データ転送と同期の 2 つの機能を socketpair のみで実現した場合にも十分なスループットが得られる可能性があると考えられる．また一方で，今回のような共有メモリを使う手法では socket API のすべての関数を RI2N/UDP 用に再実装する必要があり，

特に select() システムコールや poll() システムコールのように，通常のファイルディスクリプタと RI2N ソケットを表すファイルディスクリプタを同時に監視しなければならない場合にはその実装が非常に煩雑になる．そのため，ユーザレベル実装であったとしても，開発コストの増加や環境変化に対する可搬性の低下を招いてしまい，ユーザレベル実装であることの利点が失われてしまう．

スレッド間の通信をすべて socketpair によって行った場合，RI2N/UDP で新たに実装しなければならない関数は最小限に抑えられる．socket()，bind()，connect() などの TCP/IP に合わせて拡張されている部分については RI2N 側で再度実装する必要があるが，write()，read()，select() などの一般のファイルディスクリプタにも用いることのできる関数はそのまま socketpair に対しても発行することができるため，ユーザに提供する API library を大きく簡略化することができる．今後，RI2N/UDP 上に MPI の実装システムを移植するなど実際の使用を考えた場合にはアプリケーションを容易に移植できることが重要であり，そのような場合にはネイティブな socket API を利用できる socketpair のみによる実装の方が有利であると考えられる．また，6.2 節で述べた “返り値をユーザスレッドに渡すための write() システムコール” が不要になるため，レイテンシを約 5 μs 削減することも可能である．そのため，socketpair のみでスレッド間の通信を行うプロトタイプシステムを作成し，その実効スループットに問題がなければ今後こちらの実装方法を採用することも考えられる．

8. 関連研究

Ethernet のマルチリンクをクラスタネットワークの高バンド幅化に利用している研究として PM/Ethernet[13] がある．PM/Ethernet はクラスタミドルウェア SCore[14] の軽量通信ライブラリ PM の一部である．ドライバレベルの実装で複数の Ethernet リンクを 1 つのリンクとして利用することができ，高バンド幅化が実現されている．また，カーネル上の TCP スタックを通らずに直接ドライバを制御することによって TCP/IP や UDP/IP よりも低遅延な通信が可能である．しかし，マルチリンクを使った耐故障についてはまだ実装されていない．また，PM/Ethernet は SCore ミドルウェア上に構築されており，利用するには SCore ミドルウェアがクラスタ上で利用されている必要がある．これに対して，RI2N/UDP は標準的な UDP/IP ソケットのみを前提条件としており，一般的な Linux クラスタの上に容易にインストール可能な形で実現されている．

VMI（Virtual Machine Interface）[6] は RI2N/UDP に近いレイヤ（アプリケーションとソケットの間，もしくはソケットの直下のレイヤ）で通信を仮想化し，マルチリンクネットワークを利用可能にしている．下位のマルチリンクネットワークとして Ethernet だけでなく InfiniBand や Myrinet などの SAN もサポートしており，ヘテロでかつマルチリンクなネットワークを提供している．しかし，現在の実装では 1 つの通信ストリームはそのまま 1 つの物理リンクにバインドされており，耐故障機能は実装されているもののメッセージの分割送信による高バンド幅化はなされていない．

RI2N と同じくマルチリンクを利用した負荷分散，耐故障技術として Link Aggregation[15] がある．Link Aggregation は IEEE 802.3ad として標準化されており，これに対応したスイッチ間で Ethernet のマルチリンクを利用可能にするものである．また，NIC ドライバに Link Aggregation の機能を持たせることによってスイッチとノードの間にもマルチリンクを構成することも可能となっている[16]．しかし，Link Aggregation の利用にはそれに対応したスイッチが必要である．また，Link Aggregation プロトコルは 1 組のスイッチ間またはスイッチ–ノード間でのみ有効なものであり，図 2 のような複数スイッチにまたがるようなネットワークの構成は不可能である．そのため，接続先のスイッチに故障が発生すると通信の復旧は不可能になる．

通信 API の対象を一般的な socket ではなく，MPI ライブラリに限定した手法としては，LA-MPI[5] または Open MPI[17] に導入されている，マルチリンクを利用したバンド幅増強と耐故障機能があり，これらは RI2N/UDP と非常によく似た概念を実装している．これらの MPI 実装では元々シングルリンクでも，パケットの並べ替えや再送処理などを行って，通信の信頼性を高いレベルで維持する機能を持っている．これらの機能にはマルチリンクネットワークを利用するために必要とされる機能と重複する部分があるため，“MPI over RI2N/UDP” のように MPI よりも下層でマルチリンクを利用するよりも，MPI 実装の中で利用する方が効率的な実装を行うことが可能である．現在，MPI はクラスタ上の並列アプリケーション用通信インタフェースのデファクトスタンダードであるため，クラスタ上での利用において問題になることは少ないが，この方法では MPI 上の通信でしかマルチリンク

を利用することができない．RI2N/UDP では socket レベルでのユーザ API を提供しているため，MPI のような特定用途のライブラリにとらわれず，より広範なアプリケーションあるいは通信システムに適用することが可能であり，より一般的な応用が可能である．

これらの先行研究に対して，RI2N/UDP は (1) ユーザレベル実装である点や，(2) ハードウェア依存性が低くスイッチについても冗長化できる点，(3) 高バンド幅化と耐故障性を同時に実現している点，(4) 特定ライブラリあるいはミドルウェアに依存せず socket レベルでのユーザ API を提供するため幅広い応用に適用可能である点に優位性がある．

9. 終わりに

本論文では RI2N のユーザレベル実装システムとして RI2N/UDP を設計し実装と評価を行った．ネットワークの性能としては，GbE リンク 2 本の上で 246 MB/s のバンド幅を達成し，マルチリンクによるバンド幅の向上が確認できた．耐故障性能については，故障時にも安定して通信を継続できていることを確認し，また，故障によるスループットの低下も最低限に抑えられていることを確認した．しかし，レイテンシは 72 μs で TCP を用いた場合よりも大きくなっていた．これらの評価結果より，RI2N/UDP はレイテンシインテンシブなアプリケーションには適していないが，20 KB 以上の大きなメッセージでの通信が多い大規模アプリケーションでは 1 リンクのみを利用した TCP よりも高い性能が得られると考えられる．

今後は耐故障性のための機能拡張として，故障情報をユーザに通知する機能や上位のクラスタ管理システムとの連携についても検討する．また，ユーザスレッドと RI2N スレッドの間のインタフェースを改良し，より簡単に通常の TCP/IP 向けアプリケーションを移植可能にすること，およびスレッド間の遅延を低減させることを目指す．さらに，RI2N/UDP 上に MPI 実装の 1 つである MPICH [18] を移植し，実際のアプリケーションを利用した性能評価を行う．

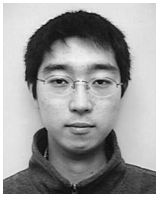
謝辞 本研究を行うにあたり，貴重な助言をいただいた CREST「メガスケールクラスタ研究チーム」のメンバに深く感謝します．本研究の一部は科学技術振興機構「戦略的創造研究推進事業（CREST）—情報社会を支える新しい高性能情報処理技術—『超低電力化技術によるディペンダブルメガスケールコンピューティング』」および文部科学省科学研究費補助（基盤研究（C）17500031）による．

参考文献

1) Myricom: Myrinet. http://www.myri.com/
2) InfiniBand Trade Association: InfiniBand. http://www.infinibandta.org/
3) van der Steen, A.J. and Dongarra, J.J.: Overview of Recent Supercomputers. http://www.top500.org/orsc/
4) Aulwes, R.T., Daniel, D.J., Desai, N.N., Graham, R.L., Risinger, L.D., Sukalski, M.W. and Taylor, M.A.: Network Fault Tolerance in LA-MPI., *PVM/MPI*, pp.344–351 (2003).
5) Graham, R., Choi, S., Daniel, D., Desai, N., Minnich, R., Rasmussen, C., Risinger, L. and Sukalski, M.: A Network-Failure-Tolerant Message-Passing System for Terascale Clusters, *International Journal of Parallel Programming*, Vol.31, No.4, pp.285–303 (2003).
6) Pakin, S. and Pant, A.: VMI 2.0: A dynamically reconfigurable messaging layer for availability, usability, and management, *SAN-1 Workshop* (*in conjunction with HPCA*).
7) Miura, S., Boku, T., Sato, M. and Takahashi, D.: RI2N — Interconnection Network System for Clusters with Wide-Bandwidth and Fault-Tolerancy Based on Multiple Links., *ISHPC*, pp.342–351 (2003).
8) Dunigan, T. and Fowler, F.: A TCP-over-UDP test harness (2002). Technical report, Oak Ridge National Laboratory, Oak Ridge, ORNL/TM-2002/76.
9) Horman, S.: iproxy: Running TCP services over UDP. linux.conf.au (2002).
10) RFC 2001: TCP Slow Start, Congestion Avoidance, Fast Retransmit and Fast Recovery Algorithms (1997). http://www.ietf.org/rfc/rfc2001.txt
11) Boku, T., Sato, M., Ukawa, A., Takahashi, D., Sumimoto, S., Kumon, K., Moriyama, T. and Shimizu, M.: PACS-CS: A Large-Scale Bandwidth-Aware PC Cluster for Scientific Computations, *Proc. 6th IEEE International Symposium on Cluster Computing and the Grid* (*CCGRID'06*), Vol.00, pp.233–240 (2006).
12) Petitet, A., Whaley, R., Dongarra, J. and Cleary, A.: HPL-A Portable Implementation of the High-Performance Linpack Benchmark for Distributed-Memory Computers, *Innovative Computing Laboratory*, Computer Science Department, University of Tennessee, Knoxville, TN (2004).
13) Sumimoto, S. and Kumon, K.: PM/Ehernet-kRMA: A High Performance Remote Memory Access Facility Using Multiple Gigabit Ether-

net Cards, *CCGrid 2003*, pp.326–333 (2003).
14) PC Cluster Consortium: SCore Cluster System Software. http://www.pccluster.org/
15) IEEE: IEEE 802.3ad “Link Aggregation” (2000).
16) Davis, T.: Linux Ethernet Bonding Driver.
17) Gabriel, E., Fagg, G., Bosilca, G., Angskun, T., Dongarra, J., Squyres, J., Sahay, V., Kambadur, P., Barrett, B., Lumsdaine, A., et al.: Open MPI: Goals, concept, and design of a next generation MPI implementation, *Proc. 11th European PVM/MPI Users' Group Meeting*, pp.97–104 (2004).
18) Gropp, W., Lusk, E., Doss, N. and Skjellum, A.: A high-performance, portable implementation of the MPI Message-Passing Interface standard, *Parallel Computing*, Vol.22, No.6, pp.789–828 (1996).

(平成 18 年 10 月 10 日受付)
(平成 19 年 2 月 1 日採録)



岡本 高幸（学生会員）

昭和 58 年生．平成 18 年筑波大学第三学群情報学類卒業．現在，同大学大学院システム情報工学研究科在学中．クラスタおよび分散コンピューティングに関する研究に従事．

三浦 信一（学生会員）

昭和 54 年生．平成 14 年千歳科学技術大学光科学部光応用システム学科卒業．平成 16 年筑波大学大学院理工学研究科修士課程修了．現在，筑波大学大学院システム情報工学研究科在学中．クラスタコンピューティング等に関する研究に従事．

朴 泰祐（正会員）

昭和 35 年生．昭和 59 年慶應義塾大学工学部電気工学科卒業．平成 2 年同大学大学院理工学研究科電気工学専攻後期博士課程修了．工学博士．昭和 63 年慶應義塾大学理工学部物理学科助手．平成 4 年筑波大学電子・情報工学系講師，平成 7 年同助教授，平成 16 年同大学大学院システム情報工学系助教授，平成 17 年同教授，現在に至る．超並列計算機アーキテクチャ，ハイパフォーマンスコンピューティング，クラスタコンピューティング，グリッドに関する研究に従事．平成 14 年度および平成 15 年度情報処理学会論文賞受賞．日本応用数理学会，IEEECS 各会員．

佐藤 三久（正会員）

昭和 34 年生．昭和 57 年東京大学理学部情報科学科卒業．昭和 61 年同大学大学院理学系研究科博士課程中退．同年新技術事業団後藤磁束量子情報プロジェクトに参加．平成 3 年通産省電子技術総合研究所入所．平成 8 年新情報処理開発機構並列分散システムパフォーマンス研究室室長．平成 13 年より，筑波大学システム情報工学研究科教授．同大学計算科学研究センター勤務．理学博士．並列処理アーキテクチャ，言語およびコンパイラ，計算機性能評価技術，グリッドコンピューティング等の研究に従事．IEEE，日本応用数理学会各会員．

高橋　大介（正会員）

昭和 45 年生．平成 3 年呉工業高等専門学校電気工学科卒業．平成 5 年豊橋技術科学大学工学部情報工学課程卒業．平成 7 年同大学大学院工学研究科情報工学専攻修士課程修了．平成 9 年東京大学大学院理学系研究科情報科学専攻博士課程中退．同年同大学大型計算機センター助手．平成 11 年同大学情報基盤センター助手．平成 12 年埼玉大学大学院理工学研究科助手．平成 13 年筑波大学電子・情報工学系講師．平成 16 年筑波大学大学院システム情報工学研究科講師．博士（理学）．並列数値計算アルゴリズムに関する研究に従事．平成 10 年度情報処理学会山下記念研究賞，平成 10 年度，平成 15 年度情報処理学会論文賞各受賞．日本応用数理学会，ACM，IEEE，SIAM 各会員．