2007－HPC－112 (10)
2007／9／9

ユーザ透過に利用可能な 耐故障・高性能マルチリンク Ethernet 結合システム

岡本高幸[†] 三浦信一[†] 朴泰祐[†,††]
塙敏博[††] 佐藤三久[†,††]

我々はこれまで，Gigabit Ethernet を複数同時に用いることで高バンド幅と耐故障性の両者を実現する PC クラスタ向けネットワーク，RI2N システムを提案・実装してきた．しかし，専用の API を用いるユーザレベルライブラリであったため，アプリケーションプログラムの再コンパイルを必要とした．そこで本論文では，RI2N のシステムレベル実装である RI2N/DRV を提案する．本システムは仮想的な Ethernet デバイスとして機能し，既存の TCP/IP レイヤをユーザ透過に利用できる．

本システムを Linux 上に実装し，Gigabit Ethernet を 2 リンクを利用した環境で評価を行った．スループットとしては最大で 229MB/s が得られ，これまでの実装システムと同様に，故障時にも冗長リンクを使って通信を継続することができることを確認した．また，本システムの内部や TCP/IP のパラメータチューニングを行い，TCP/IP をマルチリンク上で用いた場合の性能向上を試みた．

User-transparent Ethernet multilink bonding system for fault-tolerance and high performance

TAKAYUKI OKAMOTO,[†] SHIN'ICHI MIURA,[†] TAISUKE BOKU,[†,††]
TOSHIHIRO HANAWA[††] and MITSUHISA SATO[†,††]

We have already developed an implementation of RI2N as a user-level communication library. However, it requires to modify and to recompile existing applications written for TCP/IP.

In this paper, we propose a user-transparent implementation of RI2N called RI2N/DRV. We implemented RI2N/DRV on Linux and evaluated the performance and fault-tolerance with dual link Gigabit Ethernet. It was shown that the bandwidth is 229MB/s and our system can keep the communication on the network link failure to provide a high reliability on the system. Then we tried to improve the performance of RI2N/DRV with TCP/IP by tuning some parameters on RI2N/DRV and TCP stack.

1. はじめに

クラスタ向きの高性能・高信頼性のネットワークである InfiniBand や Myrinet，また，Gigabit Ethernet（以下，GbE）の次世代となる 10Gigabit Ethernet 等はその価格がまだ高価であり，小中規模の PC クラスタへ導入するにはハードルが高い．そのため，現在も PC クラスタの多くでは GbE が利用されている．その一方で，近年のマルチコアプロセッサの普及により，PC クラスタにおいてもマルチコア，マルチソケットが標準になりつつある．これにより，各ノードの計算能力は飛躍的に向上しており，それらをつなぐネットワークの性能に対する要求も高くなっている．

そこで我々は GbE を複数本同時に利用することによって高いバンド幅と耐故障性を実現する RI2N（Redundant Interconnection with Inexpensive Network）を提案・実装してきた[1]．正常時にはデータのストライピングによって通信バンド幅を向上し，リンクが故障した場合には冗長なリンクを利用して通信を継続することができる．しかし，これまでの実装システム RI2N/UDP[2] は UDP/IP を利用したユーザレベルライブラリであったため，ソースの改変やリコンパイルが必要であった．そこで，本論文ではシステムレベルの実装によってユーザ透過に利用できる新たなシステムを提案する．本システムは Linux kernel 2.6 における仮想的な Ethernet デバイスとして動作し，既存の TCP/IP レイヤをそのまま利用して高バンド幅化と耐故障性を実現することを目指す．本論文では，まず Ethernet のマルチリンクを用いたシステムについて関連する研究を紹介する．そして，RI2N/DRV の設計について述べ，実環境での評価について述べる．また，性能改善の手法の一つとして行った各種パラメータのチューニングについて述べる．

† 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
†† 筑波大学計算科学研究センター
Center for Computational Sciences, University of Tsukuba

2. 関連研究：マルチリンク Ethernet の利用

2.1 Linux Channel Bonding

Link Aggregation (IEEE802.3ad)[3] のリンク選択アルゴリズムを拡張し，NIC のハードウェアに依存せずにマルチリンクを利用するドライバとして Linux Channel Bonding[4] がある．IP などの上位層からは 1 つの Ethernet デバイスと等価に扱うことができる．

Linux Channel Bonding にはマルチリンクの利用の方法によって複数のモードがあるが，1 つのノードペア間でバンド幅を拡張できるのは balance-rr（round robin）というモードだけである．balance-rr モードでは利用可能なリンクを順番に選択し，各リンクで 1 パケット送信すると次のリンクに移る．このようにすることで複数のリンクを満遍なく均等に利用し，1 つの送信先に対するスループットをリンク数に比例して増加させることができる．しかし実際には，パケットの到達順序の入れ替わりにより TCP/IP では十分な性能を得ることは難しいと言われている[4]．

2.2 PM/Ethernet Network Trunking

PM/Ethernet[5] は HPC クラスタにおけるノード間通信のための軽量な通信ライブラリである．OS のプロトコルスタックを通らずに直接デバイスドライバを操作することによって，低レイテンシで高バンド幅の通信を提供している．PM/Ethernet は通信の負荷が小さく単一リンクでも高い性能を示す．さらに，複数のリンクを同時に利用する機能（PM/Ethernet Network Trunking，以下，PM/Ethernet NT）も持っている．PM/Ethernet NT はそれ自体が順序制御を行うため，あらかじめ到着順序の入れ替わりが頻繁に発生することが考慮されており，Linux Channel Bonding で TCP/IP を用いた場合のような性能低下はない．

しかし，PM/Ethernet NT は Ethernet デバイスとしてユーザ透過に利用することはできない．PM/Ethernet NT 自体がトランスポート層までをカバーしており，アプリケーションから直接呼び出されるライブラリとなっている．そのため，PM/Ethernet NT の利用にはアプリケーションプログラムの書き換え，もしくはそれを吸収する中間層が必要となる．

3. 設　　計

3.1 概　　要

現在の Ethernet のマルチリンクを利用したシステムでは，PM/Ethernet NT のように専用の API を用いなければマルチリンクの性能を十分に使い切ることができなくなっている．そこで我々は，ラウンドロビン送出の問題についてより詳細な解析を行いそれを解決することで，ユーザ透過に利用可能でより高性能な Ethernet リンク結合システム RI2N/DRV を開発する．

図 1 に RI2N/DRV のプロトコル階層上の位置を示す．RI2N/DRV は仮想的なネットワークデバイスとして動作する．IP や ARP などのデバイスドライバよりも上位のプロトコル階層に対しては通常の Ethernet デバイスと等価に振る舞う．図中では実デバイス（real device）は 1 つであるかのように描かれているが，実際には RI2N/DRV は複数の実デバイスを同時に利用する．また，上位層としては基本的に TCP/IP が利用されることを想定する．UDP や ICMP，ARP なども処理できる仕組みとなるが故障への対応の仕方や性能チューニングなどは TCP を基準として行う．

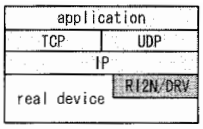

図 1　プロトコルスタック上の位置

3.2 ドライバの動作

RI2N/DRV は Ethernet デバイスとして OS に登録する．OS からは送信時に特定の送信用ハンドラが呼ばれるため，それに対応する関数を実装すれば上位層からは実体のある Ethernet デバイスと等価になる．RI2N/DRV の送信関数では，あらかじめ登録しておいた実デバイスの中からラウンドロビンでデバイスを選択して，そのデバイスの送信キューにパケットを登録する．また，このとき Ethernet ヘッダのプロトコル番号フィールドを書き換える．これは次に述べる受信側の処理のためである．

受信側で実デバイスに到着したパケットはハードウェアごとの専用ドライバによって受信され，そのまま OS のプロトコルスタックに渡される．そのため，そのままでは RI2N/DRV の受信処理を行うことができない．そこで RI2N/DRV では，利用されていない Ethernet のプロトコル番号を使って新たなプロトコルハンドラを OS に追加する．送信時に IP,ARP,RARP のヘッダをそれぞれ別のプロトコル番号に書き換えて送信することで，その番号のパケットはすべて RI2N/DRV が登録したプロトコルハンドラに渡されるようにする．そのため，このプロトコルハンドラ内で受信処理を行うことができる．受信時の処理としては，後に述べる耐故障機能のためにリンクごとのパケットカウントなどを行っている．その後，プロトコル番号を元の値に修正してプロトコルスタックにパケットを渡すことで上位層へパケットが流れていく．

3.3 マルチリンクの利用

RI2N/DRV ではスループットの向上と耐故障のためにマルチリンクを利用する．Linux Channel Bonding の balance-rr と同様に複数のリンクを満遍なく利用するためラウンドロビンで使用するリンクを切り替える．また，一部のリンクに故障が発生した場合にもラウンドロビンでパケットを送出することによって，正常なリンクでは通信相手にパケットを届けることができる．そのため，上位層で TCP を利用している場合には TCP コネクションは切断されることなく維持することができる．しかし，そのままではパケット損失率が高く性能が著しく低下する．故障後にも適切なスループットを維持するためには，早期に故障リンクを検出しその

表 1 測定環境

項目	スペック
CPU	Intel Xeon 3.0GHz 1-way HT on
memory	DDR2 1024MB
kernel	linux 2.6.20
NIC	Intel PRO1000MT dual port 1000base-T (PCI-X 64bit/133MHz)
NIC driver	Intel PRO/1000 Network Driver 7.3.15-k2-NAPI
MPI	OpenMPI 1.2.3
switch	Dell Power Connect 5224 (24 ports GbE switch)

リンクの使用を停止しなければならない．

故障検出の方法としては，RI2N/UDP と同様に受信パケット数の偏りを評価する方法を用いる．この方法は，スループットが高い状態，つまり積極的に通信を行っている場合に高速に故障の検出ができるという利点がある．RI2N/DRV では受信した NIC ごとに送信元ごとのカウンタを用意して，そのカウンタ値を元に使用，不使用の切り替えを行う．これは単純な NIC 単位のカウントでは，自ノードの NIC 以外の通信経路上に故障が発生した場合に，本来通信することのできる別のノードに対してもその NIC の利用を停止してしまうためである．また，正常な状態でもバッファの挙動によって非常に短い間隔では偏りが発生する可能性がある．そのため，パケットカウンタと同時に受信時刻を記録し，最後にパケットを受信してから一定の時間は故障と判断しない．これにより故障の誤検出を削減する．また，RI2N/DRV では故障から回復したことも自動的に検出し利用を再開する．回復の検出のため，故障と判断したリンクには定期的にハートビートパケットを送信する．通信相手から送信されたハートビートパケットが受信されれば回復したと判断する．

4. 基本性能の測定

実装した RI2N/DRV の基本性能の評価を行う．RI2N/DRV はデバイスレベルでの実装であるため，TCP，UDP を用いる幅広いネットワークアプリケーションに利用可能であるが，ここでは OpenMPI[6] による MPI 上での 2 ノード間スループットを測定する．また，直接 TCP を使った通信において，通信中に意図的に故障を発生させ故障中にも冗長リンクを使って通信が継続できることを確認する．測定環境は**表 1** に示す Xeon サーバ 2 台である．

4.1 スループット

スループットの評価のためメッセージサイズを変化させた場合の ping-pong 通信性能の変化を測定する．OpenMPI には，複数チャネルの通信路を trunk してバンド幅を増加させる機能があるので，これを比較対象とする．測定結果を**図 2** に示す．以降，本論文では “OpenMPI multi” は OpenMPI のマルチリンク trunk 機能によって 2 つのリンクで通信した場合，“RI2N/DRV multi” は RI2N/DRV の機能で 2 つのリンクを利用した場合，“single” は 1 つで通信した場合を示すこととし，図中でもこの表記を用いる．

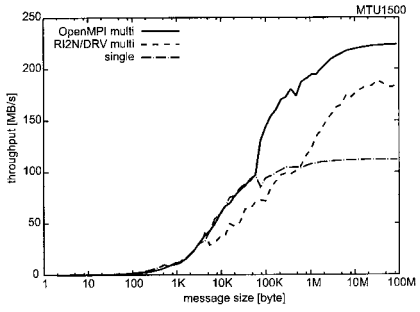

図 2 スループットの測定結果（MTU1500）

シングルリンクの最大スループットが 112MB/s，RI2N/DRV では 189MB/s，OpenMPI のマルチリンクでは 224MB/s であった．メッセージサイズを 100MB 程度まで大きくしても，RI2N/DRV のスループットは OpenMPI multi よりも 15 %ほど低く，マルチリンクの性能を十分に利用できていない．また，最大値だけでなくメッセージサイズに対するスループットの立ち上がりも OpenMPI multi に比べて遅く，メッセージサイズが 100KB から 10MB 程度までの領域で両者の差が著しい．メッセージサイズが MTU を超えたあたりから性能が急に低下しているため，複数パケットでしか起こらない問題，つまりパケット順序の入れ替わりが性能低下の原因であると考えられる．

4.2 故障時の通信状況

耐故障機能を評価するため，バースト転送中に 2 リンクのうち 1 つを意図的に故障，回復（ケーブルを抜き差し）させた場合のスループットの変化を測定する．測定結果を**図 3** に示す．MPI 上の通信であると OpenMPI の耐故障機能により別経路での通信再開が考えられるため，TCP 上でバースト転送を行うプログラムを作成し，それを使って測定を行った．スループットは 0.2 秒ごとのタイマによりアプリケーション上で取得したものである．また，故障と判断する条件としては，偏りの大きさ 10:1 以上で無受信時間が 10ms 以上であることとした．また，回復のためのハートビートの間隔は 2 秒とした．時刻 20 秒でリンクを故障させ，時刻 40 秒で復帰させた．

時刻 20 秒で故障を発生させてからそれが検出されスループットが回復するまでに 1.5 秒程度かかっている．しかし，その後は 1 リンクでの最大スループットを維持しており，冗長リンクを使って通信を継続させることができている．また，時刻 40 秒でハードウェアを回復させた後も 2 秒程度の間隔をおいて自動的にそれを検出し，スループットが 2 リンクを使った元のレベルに戻っている．故障検出に要する時間は測定するごとに大きく異なっており 10 秒程度となる場合もある．RI2N/DRV では通常のデータ通信パケットの様子から故障検出を行っているため，ある程度の数のパケットが流れないと故障検出ができない．TCP を用いている場合には，パケットロスにより急激に輻輳ウインドウ

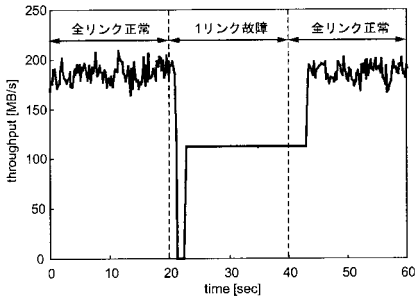

図 3　故障時のスループットの変化

サイズ（Congestion Window size，以下，CWND）が引き下げられるためにデータパケットが流れなくなり，さらに故障検出の時間を遅らせてしまうという悪循環に陥る場合がある．それに対して，回復の検出についてはハードウェアの回復から 2 秒程度でスループットが回復しておりこの値は測定回によらず概ね一定であった．このようなハートビートによる検出であれば検出までに要する最長時間を一定値に抑えることができる．そのため，故障検出の手段としてもハートビートは有効であると考えられ，今後その機能を追加していく予定である．

5. 性能改善方法の検討

5.1 ラウンドロビンの問題

マルチリンクでのラウンドロビン送出によってパケット順序の入れ替わりが起こる原因は 2 つある．一つは，スイッチのパケット処理速度や信号の伝搬遅延などにより，リンクごとにパケットの転送に要する時間が変化する“物理的な到着順序の入れ替わり”である．これは物理的に異なるハードウェアを利用しているため完全に防ぐことは難しい．しかし，HPC クラスタのように非常に短距離のネットワークでスイッチのホップ数も少ない場合には，各リンクでの通信時間の差は非常に小さくなる．TCP は 2 つまでのパケット順序の入れ替わりであれば許容するようになっており，リンクごとに 1 パケット分（MTU1500,1Gbps で 12μs）以上の差があるとは考えにくい．そのため，この問題は RI2N/DRV で TCP を用いた場合のスループットの低下にはほとんど影響していないと考えられる．

もう一つの要因は，バッファの利用によって引き起こされる“ソフトウェアによる順序の入れ替わり”である．通信経路上ではバッファを用いる場面は複数あるが，その中でも影響が最も大きいと考えられるのが Ethernet デバイスのバッファから OS のバッファにパケットをコピーする部分，およびそこからシリアライズされたキューにパケットを挿入する部分である．Intel 製 NIC ドライバ（e1000）ではこれらの処理は割り込みハンドラ内で行われており，バッファに複数のパケットがあればそれらはその順序のまま OS のキューに挿入されることになる．また，多くの GbE カードではパケット到着時に OS に対してかけるハードウェア割り込みを意図的に遅らせて，高スループット時の CPU 負荷を減らす実装がされている．しかし，このバッファ機能とマルチリンクのラウンドロビン送出が同時に用いられると，**図 4** のようにパケット順序の入れ替わりを引き起こしてしまう．実デバイスにパケットが届く順番が 101(NIC1),102(NIC2),103(NIC1)... のように送出順番通りになっていたとしても，OS の受信キューに挿入される時点では 101,103... のように順序がずれてしまう．このようなバッファによる順序のずれが本質的な問題であると考えられる．

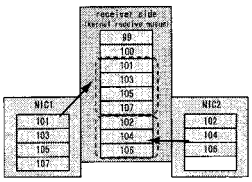

図 4　バッファによるパケット順序の入れ替わり

5.2 TCP フロー制御の振る舞い

RI2N/DRV では上位層で TCP/IP が利用されることを想定しているため，マルチリンクを用いた場合の TCP フロー制御の振る舞いについて考慮する必要がある．

TCP チャネルの受信側は，順番の入れ替わったパケットを受信するとすぐに ACK パケットを生成，送信する．送信側ではこの ACK 情報を元にフロー制御を行うため，最初の 1 回だけではなく不揃いなパケットが届くごとに毎回 ACK が送信される．送信側（ACK パケットを受信する側）では，基本的には再送タイムアウトまでパケットの再送信を行わない．ただし，ACK 番号が同じパケットを一定数以上受信した場合にはタイムアウトを待たずに再送を開始する．これは，そのACK 番号と同じシーケンス番号のパケットがネットワーク上で消失されたと判断されるためである．マルチリンク環境ではこれが原因となって大量のパケットが再送信されることが予想される．さらに，パケットロスによる CWND の引き下げも同時に行われる．古典的な Reno アルゴリズムでは CWND はそれまでの 2 分の 1 に設定されるが，この値はフロー制御アルゴリズムによって異なっている[7]．TCP フロー制御アルゴリズムには Long Fat Pipe 用や無線通信用など多くの種類があり，RI2N/DRV での通信のように Round Trip Time（以下，RTT）が短く順序入れ替えが頻発する状況に最適化されたアルゴリズムを用いることで，よりスループットが得られる可能性がある．

5.3 リンク選択方法の変更

5.1 節で述べた問題に対応するために，ラウンドロビンで 1 個ずつパケットを送出するのではなく，α 個ずつを 1 つのリンクに連続して送出する方法（以下，rr-burst(α) 方式）を考える．rr-burst(α) 方式を用いた場合，各 NIC のバッファに保存されるパケットの順番は最大 α 個連続してそろっている．OS がこれらを一

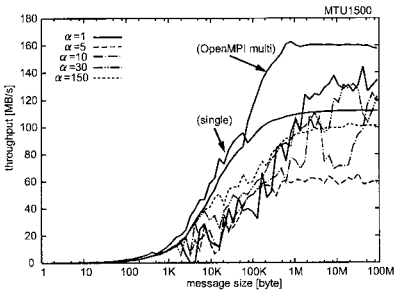

図 5　rr-burst(α) 方式でのスループット

度にキューに挿入すればその部分でパケットの順序は入れ替わらず，単純なラウンドロビンよりも順序の入れ替わりは少なくなると考えられる．

実際に RI2N/DRV のリンク選択アルゴリズムを rr-burst(α) 方式に変更してスループットを測定した．その結果のうち代表的なものを**図 5** に示す．割り込み遅延時間の長さは比較的低速な固定値，60μs に 1 回の割り込み（e1000 では RxIntDelay=58, InterruptThrottleRate=16667）となるように設定した．MTU は 1500byte とした．また，OpenMPI による 2 リンクの trunk を行った場合とシングルリンクでの結果を比較のため合わせて示した．α を 1 から 300 まで変化させてスループットの測定を行ったが，結局 $\alpha = 1$ の場合が最もスループットが高いという結果になった．$\alpha = 5$ の付近で最もスループットが低くなり，その後さらに α を大きくしていくと $\alpha = 30$ 程度で一度ピークを迎える．しかし，この付近では測定するごとに値が大きく変わる．そのため，場合によっては $\alpha = 1$ の場合を上回る場合もあるが最大値は概ね $\alpha = 1$ の場合に等しい．さらに α を大きくしていくと測定回ごとの揺れが少なくなり，$\alpha = 150$ として示したグラフに収束していく．

この測定では割り込み間隔を一定値に設定したが，ドライバのデフォルトではスループットに合わせて動的に割り込み間隔を変更する設定になっている．4.1 節で示したスループットはデフォルトの設定で測定したものであったが，そちらの場合の方が OpenMPI multi と RI2N/DRV の両方でスループットが高いという結果になった．OpenMPI multi では特に 160MB/s 付近で明らかに処理能力が飽和している．これは割り込み間隔を 60μs に固定したことによる性能低下であると考えられる．しかし，それを差し引いたとしても RI2N/DRV の性能低下は大きく，また不安定な挙動を示していることから rr-burst(α) 方式によるこれ以上の性能の改善は難しいと考えられる．

5.4 TCP フロー制御アルゴリズムの変更

次に，5.2 節に示した問題について調べる．そのために，TCP のフロー制御アルゴリズム，およびフロー制御のパラメータを変化させて，それぞれが性能（ping-pong スループット）に与える影響について調査する．現在の Linux カーネルでは BIC-TCP が標準のフロー制御アルゴリズムとなっている．しかし，それ以外にも，旧来の Reno，Long Fat Pipe 向けの Scalable TCP，無線通信向けの TCP Westwood+などが用意されており，明示的にこれらのアルゴリズムを指定すれば利用することができる．現在利用可能な既存の TCP アルゴリズムを用いて RI2N/DRV のスループットをそれぞれ測定した．**図 6** に特徴的なアルゴリズムでの結果を示す．また，比較のため BIC-TCP でのシングルリンクと OpenMPI によるマルチリンクの結果も合わせて示す．

試行したフロー制御アルゴリズムの中では TCP Westwood+が最も安定して高いスループットを示した．バースト転送でのスループットでは BIC-TCP と Westwood に大きな差はないが，100KB から 10MB 程度までのメッセージサイズでは Westwood の方が高いスループットを示している．この結果は，Westwood が RTT を元に経路上のスループットを予測して CWND を決定しているためであると考えられる．RI2N/DRV では実際にパケットが消失することは少ないが，TCP の仕様により順序の入れ替わりがパケットロスのように判断される．通常はそれにより CWND が大幅に縮小されるが，Westwood ではそれがなく CWND が高いまま維持される．そのため，結果として高いスループットが得られるものと考えられる．

次に，TCP フロー制御の影響を調べるもう一つの実験として，sysctl のパラメータ net.ipv4.tcp_reordering を変化させてその影響を調べた．この値はパケット数を表しており，ACK 番号が等しいパケットがそれ以上届いた場合にパケットロスがあったと判断する基準値となっている．デフォルトでは 3 が設定されている．この値を大きくすることで，順序の入れ替わりをパケットロスと判断するまでの閾値を高くすることができ，結果として CWND の縮小を抑えることが期待される．測定結果を**図 7** に示す．

reordering パラメータを大きくしていくと最終的に図 6 の Westwood の値に収束していくという結果が得られた．reordering=15 まではいったんスループットが低下するが，その後徐々に高くなり，22 以上では変化がなくなる．今回実験を行った環境では 22 個以上の連続したパケットの順序入れ替えは発生していないため，reordering パラメータによるスループットの変化がなくなったものと考えられる．

2 つの実験によって TCP フロー制御アルゴリズムに関連するチューニングを行い，パケット順序の入れ替わりに対する対策を試みた．いずれの場合もある程度の改善はあるもののその大きさは十分とは言えず，いずれの実験でも 1 つの限界値（Westwood の値,reordering≥22 の値）に収束した．しかし，TCP のスループットはその時点での CWND と RTT によって決まる．このことから，何らかの要因で CWND が十分に大きくならず性能の低下を招いている可能性がある．また，パラメータチューニングのみでは順序不一致による ACK パケット自体を減らすことはできない．

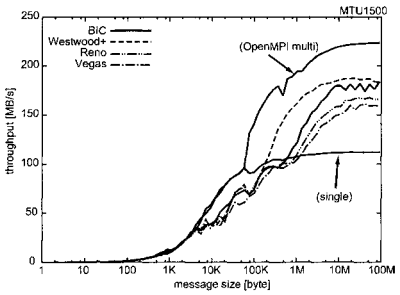

図 6 TCP フロー制御アルゴリズムによる性能の変化（RI2N/DRV）

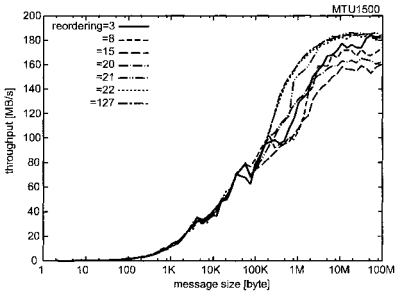

図 7 パラメータ（reordering）による性能の変化（RI2N/DRV）

そのため，この大量の ACK パケットが性能低下の原因となっている可能性もある．

5.5 性能改善の方法

今後の RI2N/DRV の性能改善の方法として以下の 3 つが考えられる．

(1) パラメータチューニングによる改善
(2) ドライバ（モジュール）でトランスポート層まで実装する
(3) ドライバ内でパケット順序の並び替えを行う

5.3 節と 5.4 節で現在設定可能ないくつかのパラメータを変更して性能改善を図った．しかし，前者は性能向上が得られず後者もその改善の幅は大きくなかった．より詳細な性能解析，問題解析によってさらに性能を改善できる可能性はあるがこの方法での抜本的な改善は難しいと考えられる．

(2) では PM/Ethernet NT と同様に，アプリケーションレベルの API まで RI2N/DRV が提供する形になればレイテンシなどについても大きく改善できる．しかし，開発のコストが大きく，また専用の API を用いることになればユーザ透過に利用できるという利点を失うことになる．

(3) は比較的容易に実装できる現実的な方法であると考えられる．この場合にも TCP を用いることを想定し，ドライバ内ではパケット順序の入れ替わりだけをある程度修正し，再送処理や輻輳回避，また，最終的な順序制御は TCP 層に任せる．ただし，この案の実装には順序番号を持ったフッタの追加，もしくは TCP のシーケンス番号を見るなどして，送信前の順序を復元する手段が必要となる．今後はまずこの 3 つめの方法を用いて RI2N/DRV の性能改善を行っていく．

6. おわりに

RI2N のシステムレベル実装として，仮想 Ethernet デバイスとして動作する RI2N/DRV を提案，実装した．性能評価においては，2 リンクの GbE を用いて最大 229MB/s のスループットが得られ，また，片方のリンクが故障した状態でも通信を継続する耐故障性を確認した．しかし，メッセージサイズに対するスループットの立ち上がりが遅く，パラメータチューニングによる性能改善を試みた．その結果，動的に設定可能なパラメータチューニングのみでは十分な性能改善は行えず，さらなる改善のためにはより詳細な問題分析が必要であることがわかった．

今後は，パケットの並べ替えをドライバ内でサポートし，マルチリンクの利用による影響をドライバ内で吸収できるようにシステムを改良していく．

謝辞 本研究の一部は科学技術振興事業団・戦略的創造研究推進事業 (CREST) の研究プロジェクト「省電力でディペンダブルな組込み並列システム向け計算プラットフォーム」による．

参考文献

1) 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).
2) Okamoto, T., Miura, S., Boku, T., Sato, M. and Takahashi, D.: RI2N/UDP: High bandwidth and fault-tolerant network for a PC-cluster based on multi-link Ethernet., *IPDPS*, IEEE, pp.1–8 (2007).
3) IEEE: IEEE 802.3ad "Link Aggregation" (2000). http://www.ieee802.org/3/ad/index.html.
4) Davis, T.: Linux Ethernet Bonding Driver. http://sourceforge.net/projects/bonding.
5) Sumimoto, S. and Kumon, K.: PM/Ehernet-kRMA: A High Performance Remote Memory Access Facility Using Multiple Gigabit Ethernet Cards, *CCGrid 2003*, pp.326–333 (2003).
6) Gabriel, E., Fagg, G., Boscilca, G., Angskun, T., Dongarra, J., Squyres, J., Sahay, V., Kambadur, P., Barrett, B., Lumsdaine, A. et al.: OpenMPI: Goals, Concept and Design of a Next Generations MPI Implementation, *11th European PVM/MPI Users Group Meeting*.
7) 剛長谷川，正幸村田：高速・高遅延ネットワークのためのトランスポート層プロトコル (トラヒック解析・制御 (1), インターネットトラヒック,TCP/IP, 性能解析・評価, ネットワークモデル及び一般), 電子情報通信学会技術研究報告. IN, 情報ネットワーク, Vol.106, No.524, pp.41–46 (20070125).