博士論文　　平成２８年度（２０１６年度）

IP (Internet Protocol) を用いた低遅延分散アーキテクチャ

慶應義塾大学大学院 政策・メディア研究科

松谷 健史

概要

クラウドの普及と Ajax や WebSocket を利用した画面の自動更新に対応するサービスの普及によって，アプリケーションの高速化が求められるようになり，データセンタ内においてネットワークスループットの向上だけでなく，低遅延通信が要求されている．例えば，memcached (Facebook や Twitter で利用されている) などの分散メモリキャッシュサーバや、iSCSI などのストレージサービスでは，高速化のために低遅延通信が求められている．

しかし、データセンタで使われている IP ネットワークは接続性に優れているが，InfiniBand や独自の専用ネットワークと比べて通信遅延が大きく，これらのサービスの処理性能を低下させている．低遅延通信には，RDMA が利用されるが，機構的に DMA リードによるオーバーヘッドが生じる．

本研究では，DMA リードを廃することで，RDMA より通信遅延を少なくする IP-NUMA を提案する．IP-NUMA は，バークレイソケット API に代わり，ノード間の一部のメモリを NUMA アーキテクチャによって共有メモリ化し，データ送信時に DMA リードに代わって CPU ライトを使い，PC 上で生じる通信遅延を削減する手法である．IP-NUMA NIC を用いたメモリ書き込みによる通信 （PC+FPGA，Linux 上で構築）と，市販 NIC を用いたバークレイソケットによる通信を ping-pong プログラムを使って分析したところ，通信遅延を 90% 削減した．片道通信遅延は，1.081 μs で，RDMA を用いた場合と比べて，11% 以上削減することができた．

また，L3 スイッチの IP パケット転送に必要な処理と最小転送遅延をあきらかにした．FIBNIC は，パケット転送遅延を削減するため，本研究では全工程を最少ステージ数でパイプライン化し，パケットバッファを用いない IP パケット転送手法である．本手法を市販の FPGA ボードに実装したところ，41 万経路の IPv4 パケットの転送遅延が 976 ns で一定であることが確認された．これは市販のギガビット L3 スイッチにて 64 バイトフレームを計測したときの 20〜25% に相当する．

本手法である IP-NUMA と FIBNIC を memcached に適用し，見積もりしたところ，ローカルの IP ネットワークにおいて，バークレイソケットと比べ，4 倍のトランザクション性能となった．コモディティネットワークを使用しながら，クラウドや HPC などのインターコネクトとして IP プロトコルの適用範囲が広がることが期待できる．

キーワード: データセンタ, 低遅延通信, RDMA, IP, memcached, FPGA.

Abstract

Due to the migration of services to the cloud and automatic update of a computer screen using Ajax and WebSockets, and the demands of faster applications, improvement in network throughput and low latency communication is required in the data center. For example, a distributed memory cache server, such as memcached, which is used by Facebook and Twitter, or storage services, such as iSCSI, needs low-latency communication.

IP network technology, which is used in a data center, is superior in longevity. However, compared with InfiniBand or a special purpose network, the IP network takes a long time to communicate, causing lower performance of processing in which low-delay communication is required for these services. Even when RDMA is used in low-latency communication, overhead is caused by the mechanism for DMA read.

In this research, we propose IP-NUMA to reduce the communication delay compared to RDMA by removing the DMA read. IP-NUMA is a technique which takes the place of Berkeley sockets, and reduces communication delay on PCs, by sharing parts of memory among nodes by using a NUMA architecture and creating a flexible IP protocol as an interconnect. A ping-pong program writing memories using IP-NUMA (implemented on PC+FPGA with Linux) reduced communication delay by 90% compared to Berkeley sockets. One-way communication delay could be reduced by 11% or more 1.081μs as compared with the case of using RDMA.

We clarified the process and the minimum packet forwarding delay for IP packet forwarding of L3 switch. In order to reduce the packet forwarding delay, FIBNIC is pipelined in the minimum number of stages. It is an IP packet forwarding method that does not use packet buffers. When we implemented this method on a commercial FPGA board, we confirmed that the forwarding delay even with 410,000 IPv4 routes is constant. Our latency corresponds to 20$\sim$25% of what when we measured 64 byte frame with a commercial Gigabit L3 switch.

When we applied FIBNIC and IP-NUMA, to memcached, they were 4 times as efficient in transaction rates on a local IP network compared to Berkeley sockets. An application scope of IP protocol as an interconnect such as cloud or HPC by using commodity network will be expected.

Keywords: Data-center, Low-latency, RDMA, IP, memcached, FPGA.

目次

第 1 章　序論　1
1.1　IP ネットワークの速度向上とバークレイソケットによる制限 1
1.2　データセンタ内における低遅延通信の必要性 3
1.2.1　ストレージサービスと低遅延通信 3
1.2.2　メモリキャッシュサーバと低遅延通信 5
1.3　本研究が解決する問題 6
1.4　本論文の構成 7

第 2 章　関連技術と研究　9
2.1　既存 NIC に専用ドライバを用いた高速・低遅延通信 9
2.1.1　Intel DPDK 9
2.1.2　netmap 10
2.2　RDMA 対応 NIC による低遅延通信 10
2.2.1　iWARP/RoCE 12
2.3　PCI Express を用いた低遅延通信 12
2.4　まとめ 12

第 3 章　低遅延分散アーキテクチャ　14
3.1　概要 14
3.2　通信遅延の分析 16
3.2.1　ソフトウェア処理 16
3.2.2　ハードウェア処理 17
3.2.2.1　PCI Express バスの遅延計測 20
3.2.2.2　DMA リード・ライト 20
3.2.2.3　CPU リード 21
3.2.2.4　CPU ライトと Write-Combining 22
3.2.2.5　PCI Express コマンドの検討 22
3.2.2.6　パケット送受信時のハードウェア遅延 23
3.3　IP-NUMA の設計 24
3.3.1　ページとページマップテーブル 25
3.3.2　TLP over IP 27
3.3.3　データ転送処理 27
3.3.4　データ保護 28
3.3.5　再送処理と UDP プロトコル 29

第 4 章　IP-NUMA の実装と評価　31
4.1　システムの実装 31

4.1.1 リクエスタ 32
4.1.2 サーバ 32
4.2 通信遅延の評価 32
4.2.1 バークレイソケットによる ping-pong 33
4.2.2 IP-NUMA による ping-pong 33
4.2.3 計測 35
4.3 考察 37
4.3.1 CPU 負荷 37
4.3.2 データ転送 38
4.3.3 通信遅延の改善 38
4.3.4 アプリケーションへの適用 38
4.3.5 データセンタでの適用 39

第 5 章 IP 転送遅延の解析と FIBNIC による検証 40
5.1 ネットワーク網の通信遅延の削減 40
5.1.1 既存の問題 41
5.1.1.1 フレームの書き換え 41
5.1.1.2 フレームの配送 42
5.1.1.3 論理回路からみた IP パケット転送処理 42
5.1.1.4 主要回路 42
5.1.1.5 PHY による遅延 43
5.1.1.6 転送遅延の試算 43
5.1.2 パイプラインによる IP パケット転送 44
5.1.2.1 フレーム書き換えのパイプライン化と先頭タギング . . 44
5.1.2.2 IP CHECKSUM 45
5.1.2.3 Frame Check Sequence(FCS) 46
5.1.2.4 FIFO によるフレーム転送 46
5.2 システムの実装 48
5.2.1 FIBNIC 48
5.2.2 概要 48
5.2.3 IP Lookup 探索 50
5.2.4 IPv6 対応 51
5.2.4.1 デュアルスタック 51
5.3 性能評価 52
5.3.1 評価システムの構成と計測結果 52
5.4 考察 54

第 6 章 アプリケーションによる評価 55
6.1 memcached の概要 55
6.2 memcached の機能とプロトコル 56
6.3 データベースキャッシュとしての memcached 58
6.4 通信パケットの解析 59
6.5 基本性能計測 61
6.6 性能見積 62
6.6.1 ラック内接続 65
6.6.2 ラック間接続 66
6.7 考察 68

6.7.1 本手法が有効なアプリケーション 68

第 7 章 結論 70
7.1 本論文のまとめ 70
7.2 結論 71
7.3 本研究が社会に与える影響 72
7.4 今後の課題 72
7.4.1 IP-NUMA を用いた受信時のポーリング 72
7.4.2 バークレイソケット ラッパの対応 72
7.4.3 L3 スイッチのポート数と FIFO 73

謝辞 75

参考文献 76

付録 A 本研究に関する業績リスト 80
A.1 学会発表，論文，活動 80
A.1.1 国際学会発表 80
A.1.2 論文誌 80
A.1.3 国内コンペティション 81
A.1.4 作品 81

図目次

1.1 転送速度の比較 2
1.2 データベースサーバ構成例 6
1.3 キャッシュサーバの構成例 6

2.1 一般の NIC とドライバによる送受信データの流れ 10
2.2 RDMA 対応 NIC による送受信データの流れ 11

3.1 バークレイソケットモデル 15
3.2 本研究手法 (NUMA) モデル 15
3.3 想定ネットワーク 16
3.4 既存のパケット通信処理の流れ 17
3.5 NIC 送信時の処理フロー例 18
3.6 RDMA NIC 送信時の処理フロー例 19
3.7 NUMA NIC 送信時の処理フロー 23
3.8 送受信遅延の比較 24
3.9 処理フロー 25
3.10 ページとページマップテーブル 26
3.11 ページマップテーブル詳細 (上) と PC ID テーブル詳細 (下) 27
3.12 パケット形式 28
3.13 メモリマップ 29
3.14 共有領域テーブル 30

4.1 IP-NUMA の概要構成図 33
4.2 バークレイソケットによる ping-pong 34
4.3 IP-NUMA による ping-pong 35
4.4 ping-pong プログラムによるループ/秒の比較 (10 ギガ) 36

5.1 フレーム書き換え・ACL に必要なアクセス項目 41
5.2 イーサネットを構成する主要回路 43
5.3 送受信 PHY 間の遅延
上が PORT2 の GMII Transmit Enable 信号
下が PORT1 の GMII Receive Data Valid 信号 44
5.4 フレーム受信と転送 45
5.5 パイプライン処理 46
5.6 主要回路モジュールと FIFO（4 ポート構成例） 47
5.7 転送回路への複数入力 47
5.8 モジュールダイアグラム 49

5.9 IP Lookup 51
5.10 フレームサイズ毎の遅延 53

6.1 アプリケーションとキャッシューサーバの構成例 56
6.2 set コマンド発行時の送受信パケット 57
6.3 get コマンド発行時の送受信パケット 58
6.4 データ読み込み (キャッシュミス時) 59
6.5 データ読み込み (キャッシュヒット時) 59
6.6 データ更新 60
6.7 評価環境の構成 60
6.8 ベンチマーク用コマンド 61
6.9 パケットキャプチャ用コマンド 61
6.10 Wireshark によるパケット解析 (SET コマンド問い合わせ) 62
6.11 ラック内接続の想定構成 64
6.12 ラック間接続の想定構成 64
6.13 ラック内接続における処理能力の比較 66
6.14 ラック内接続における処理時間内訳 (データ長 16 バイト) 66
6.15 ラック間接続における処理能力の比較 67
6.16 ラック間接続における処理時間内訳 (データ長 16 バイト) 68
6.17 アプリケーション処理時間と処理性能比（ラック内接続） 69
6.18 アプリケーション処理時間と処理性能比（ラック間接続） 69

表目次

1.1 ペリフェラルバスの速度推移 3
1.2 ストレージバスの速度推移 3
1.3 イーサネットの速度推移 4
1.4 メモリインターフェイスの速度推移 4
1.5 イーサネットの PPS (Packet Per Second) と 1 パケットの伝送遅延 . . 4
1.6 接続形態によるトランザション処理時間 (SET コマンド ×1, データ長 16 バイト) 7
1.7 IP パケット転送遅延 (IPv4 64 バイトフレーム) 8

3.1 実験環境 20
3.2 DMA リード処理時間 21
3.3 DMA ライト処理数 21
3.4 memcpy 処理時間 22

4.1 IP-NUMA 10G NIC 実装環境 31
4.2 IP-NUMA ギガビット NIC 実装環境 32
4.3 10G 計測時の仕様 36
4.4 ギガビット 計測時の仕様 36
4.5 Round Trip Time 37
4.6 片道遅延 39

5.1 実装環境 49
5.2 64 バイトフレーム (IPv4) 52
5.3 1518 バイトフレーム (IPv4) 53
5.4 IPv4 41 万経路 (入力 1 ポート，出力 3 ポート) 54
5.5 IPv6 32 経路 (入力 1 ポート，出力 1 ポート) 54

6.1 memcached の主要コマンド 57
6.2 PC の仕様 60
6.3 memcached コマンドの実行時間計測 (単位 ns) 63
6.4 適用手法と速度別の GET コマンドの実行時間 (単位 ns) 63
6.5 往復遅延時間 64
6.6 ラック内・間接続時の GET コマンド実行時間 (単位 ns) 65
6.7 ラック内接続における処理能力の向上 67
6.8 ラック間接続における処理能力の向上 67

第 1 章

序論

本章では，まず本研究の背景となる IP ネットワークの進化における問題とデータセンタ内の低遅延通信要求について述べ，次に本研究の解決する問題を定義する．また，本論文の構成についても述べる．

1.1 IP ネットワークの速度向上とバークレイソケットによる制限

インターネットが一般に普及されるまでの 1990 年代までは，コンピュータネットワークには様々な仕様が存在した．物理層では，Local Talk，トークンリング [1]，イーサネット [2], FDDI, ATM．通信プロトコルでは Apple Talk, IPX/SPX, NetBIOS, TCP/IP, SNA などである．しかし，インターネットの普及によって，ネットワークの規模に関わらずイーサネットと IP プロトコルが標準的に使われるようになった．ネットワーク仕様が標準化されたことによりハードウェアおよびソフトウェアが広く使われるようになり，イーサネットは大幅な速度向上を果たしている．

1980 年代以降のイーサネットと以下にあげる PC 用の各種バスインターフェイスの転送速度比較を図 1.1 に示す．縦軸は対数表記である．

ペリフェラルバスについての詳細を表 1.1 に示す．転送速度は双方向の転送帯域の合計で，カッコ内は片方向での転送帯域である．ISA/PCI/PCI-X バスは，半二重通信，PCI Express バスは，全二重通信が可能である．仕様上，32 レーンまで定義されているが，市販で利用されているのは 16 レーンまでで，実質的に制約されているのでそちらのデータを示す．ストレージバスについての詳細は 表 1.2，イーサネットについては 表 1.3，メモリインターフェイスについては 表 1.4 に示し，転送速度はシングルチャンネル時のもので，仕様上の最大のものを記述しているが，製品として市場にリリースされていない場合

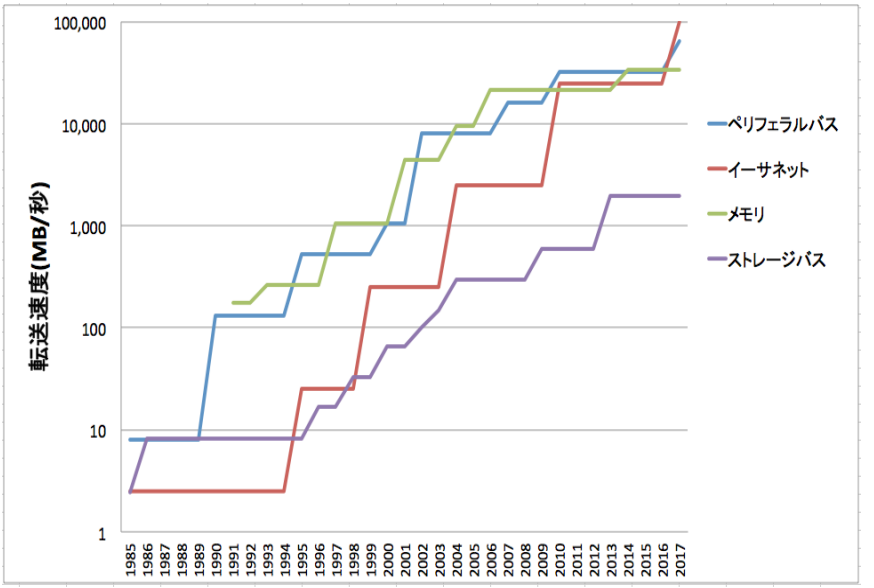


図 1.1 **転送速度の比較**

がある．

図 1.1 より，初期のイーサネットの速度は他のバスインターフェイスと比べ著しく遅いが，インターネットが普及しはじめた 1990 年代頃より改善され，2010 年代ではペリフェラルバスと遜色のない転送速度まで高速化され，テラビットを超える仕様も計画 [3] されている．イーサネットはネットワークの規格で，PC 向けのバスインターフェイスのように PC のハードウェア仕様に制限されないためである．

一方，イーサネットの速度向上とともに時間あたりの処理するパケット数が増加し，ネットワーク性能を十分に生かすことが困難になってきている．例えば， 40G イーサネットでは表 1.5 で示すように，秒間最大 59.5MPPS のパケットが発生し 1 パケットの伝送遅延は，16.8 ns となる．このことより，1 パケットのソフトウェア処理時間は，5GHz の CPU において 84 サイクル以内で処理することが求められる．

しかし，ソフトウェアにおけるネットワークプログラミングの基本モデルはバークレイソケットによって定義され，イーサネットの初期時代から 30 年以上に渡って利用されてきた [4] プロトコルスタックと API をソフトウェアで実装するバークレイソケットのモデルがネットワーク高速化の制約となってきている．例えば，Linux カーネル 3.5 の環境においての Brandeburg の計測によると，パケットの送信と受信処理では 5,722 ns[5]（内訳はドライバが 1,172 ns，アプリケーションが 356 ns，プロトコルスタックが 4,194 ns）のソフトウェア処理時間が必要とされ，高速 IP ネットワークの妨げになってきている．

表 1.1 **ペリフェラルバスの速度推移**

仕様策定	規格	バスまたはレーン	周波数	転送速度/秒	双方向通信
1981 年	ISA	8 ビット	8.33MHz	8MB	半二重
1990 年	PCI 1.0	32 ビット	33MHz	133MB	半二重
1995 年	PCI 2.1	64 ビット	64MHz	533MB	半二重
2000 年	PCI-X 1.0	64 ビット	133MHz	1.06GB	半二重
2002 年	PCI Expresss 1.0	16 レーン	2.5GHz	8GB (4GB)	全二重
2007 年	PCI Expresss 2.0	16 レーン	5.0GHz	16GB (8GB)	全二重
2010 年	PCI Expresss 3.0	16 レーン	8.0GHz	32GB (16GB)	全二重
2017 年	PCI Expresss 4.0	16 レーン	16.0GHz	64GB (32GB)	全二重

表 1.2 **ストレージバスの速度推移**

仕様策定	規格	バス	転送速度/秒	双方向通信
1980 年	ST-506	1 ビット	0.625MB	半二重
1985 年	ESDI	1 ビット	2.44MB	半二重
1986 年	IDE	16 ビット	8.3MB	半二重
1996 年	Enhanced IDE	16 ビット	16.7MB	半二重
1998 年	Ultra ATA/33	16 ビット	33MB	半二重
2000 年	Ultra ATA/66	16 ビット	66MB	半二重
2002 年	Ultra ATA/100	16 ビット	100MB	半二重
2003 年	SATA 1.0 (1.5Gbps)	シリアル	150MB	半二重
2004 年	SATA 2.0 (3Gbps)	シリアル	300MB	半二重
2009 年	SATA 3.0 (6Gbps)	シリアル	600MB	半二重
2013 年	SATA 3.2 (16Gbps)	シリアル	1969MB	半二重

1.2 データセンタ内における低遅延通信の必要性

クラウドの普及によって，データセンタ内においてネットワークスループットの向上だけでなく，低遅延通信が要求されている．本節では，低遅延通信を必要としているストレージサービスとメモリキャッシュサーバについてそれぞれ述べる．

1.2.1 ストレージサービスと低遅延通信

ストレージデバイスとしてハードディスクが主流だった頃は，アクセス遅延が大きいため，ネットワークの通信遅延が問題になることは少なかった．しかし，SSD や NVM Express (NVMe) などのフラッシュメモリベースのストレージデバイスが普及するにつれて，デバイスのアクセス遅延が大幅に削減され，通信遅延から生じるストレージサービ

表 1.3 **イーサネットの速度推移**

仕様策定	規格	速度	転送速度/秒	双方向通信
1983 年	IEEE802.3 10BASE5	10Mbps	2.5MB (1.25MB)	全二重
1995 年	IEEE802.3u 10BASE-TX	100Mbps	25.0MB (12.5MB)	全二重
1999 年	IEEE802.3ab 1000BASE-T	1Gbps	250.0MB (125MB)	全二重
2004 年	IEEE802.3ak 10GBASE	10Gbps	2.5GB (1.25GB)	全二重
2010 年	IEEE802.3ba-2010 100GBASE	100Gbps	25.0GB (12.5GB)	全二重
2017 年	IEEEP802.3bs 400GBASE	400Gbps	100.0GB (50.0GB)	全二重

表 1.4 **メモリインターフェイスの速度推移**

仕様策定	規格	バス幅	バス周波数	転送速度/秒	双方向通信
1991 年	FPM DRAM	64 ビット	66MHz	176MB	半二重
1993 年	EDO DRAM	64 ビット	66MHz	264MB	半二重
1997 年	SDR-SDRAM (PC133)	64 ビット	133MHz	1.066GB	半二重
2001 年	DDR-SDRAM (PC-4400)	64 ビット	275MHz	4.4GB	半二重
2004 年	DDR2-SDRAM (PC2-9600)	64 ビット	600MHz	9.6GB	半二重
2006 年	DDR3-SDRAM (PC3-21333)	64 ビット	1333MHz	21.3GB	半二重
2014 年	DDR4-SDRAM (PC4-34100)	64 ビット	2132MHz	34.1GB	半二重

表 1.5 **イーサネットの PPS (Packet Per Second) と 1 パケットの伝送遅延**

	64 バイトパケット		1518 バイトパケット	
速度	PPS	時間/Packet	PPS	時間/Packet
1Gbps	1,488,095	672.0 (ns)	81,274	12,304.0 (ns)
10Gbps	14,880,952	67.2 (ns)	812,743	1,230.4 (ns)
40Gbps	59,523,810	16.8 (ns)	3,250,975	307.6 (ns)
100Gbps	148,809,524	6.7 (ns)	8,127,438	123.0 (ns)
400Gbps	595,238,095	1.7 (ns)	32,509,753	30.8 (ns)

スの性能低下が問題になってきた．

Qiumin Xu 氏の計測によると，SATA ハードディスクの読み込み遅延は，5.3 ms[6]，NVMe ストレージの読み込み遅延は，110 μs で，ハードディスクと比べて，数百分の 1 に削減されている．

ネットワークの通信遅延を削減する技術として，Remote Direct Memory Access[7] (RDMA) がよく知られ，それに対応した NIC とソフトウェアが使われる．例えば，SCSI プロトコルを RDMA に対応させた，iSCSI Extension for RDMA (iSER)，SCSI RDMA Protocol (SRP) や，NFS を RDMA に対応させた，NFS/RDMA などがあげられる．

近年では，Persistent memory[8] に対応した様々な製品がリリースされてきている．

NVMe よりも優れた低遅延アクセス性能 (数十 ns) を持ち，PC のメモリスロットに直接装着可能な不揮発性高速メモリである NVDIMM があげられる．これらをインメモリデータベースなどで利用することにより，通信遅延に関してもより一層の削減を求められてきている．

1.2.2 メモリキャッシュサーバと低遅延通信

図 1.2 は，1990 年代のウェブサービスのサーバ構成例である．アプリケーションは，ユーザがクリックしたときにのみ実行され，それ以外での負荷は少ない．このときの処理の流れを以下に示す．

① アプリケーションサーバがデータベースサーバへ問い合わせる
② データベースサーバがハードディスクへアクセスする
③ データベースサーバがアプリーションサーバへ該当レコードを返す

ハードディスクはアクセス時間が ms 単位のため，ネットワークに対する通信遅延時間の削減要求はそれほど多くなかった．

2000 年代になると，Ajax[9] や WebSocket[10] などの双方向通信を用いた，画面の自動更新に対応するサービスが一般的になってきた．

インターネット利用者の増加と双方向通信により，頻繁にデータベースへの問い合わせが増え，アプリケーションの負荷が増大し，ハードディスクへの頻繁なアクセスが発生するこれまでの方式では対応しきれなくなった．このため，図 1.3 で示す例のように，データベースサーバとは別にインメモリベースの memcached などを分散メモリキャッシュサーバとして用いる構成が使われるようになった．このときの処理の流れを以下に示す．

① キャッシュサーバへ問い合わせる（該当データが存在しない場合は 4 へ推移)
② キャッシュサーバがメモリへアクセスする
③ キャッシュサーバからの応答.（問い合わせ処理の完了）
④ データベースサーバへ問い合わせる
⑤ データベースサーバがハードディスクへアクセスする
⑥ データベースサーバがアプリーションサーバへ該当レコードを返す
⑦ アプリケーションサーバがキャッシュサーバへ書き込む

アプリケーションサーバは，初回のアクセスにおいて遅延が大きく負荷が重いデータベースサーバではなく，まず遅延が小さく，負荷が軽い高速なキャッシューサーバへアクセスすることで，アクセス頻度の高いアプリケーションからの要求に対応できるようにしている．インメモリベースのキャッシュサーバでは，I/O のアクセス遅延は遅くとも μs 単位である．

キャッシュサーバの導入や，データベースサーバのストレージのフラッシュ化や NVDIMM 等をはじめとする Persistent memory の導入により，一部のアプリケーションでは処理時間が大幅に削減され，ネットワークの通信時間が占める割合が増えている．

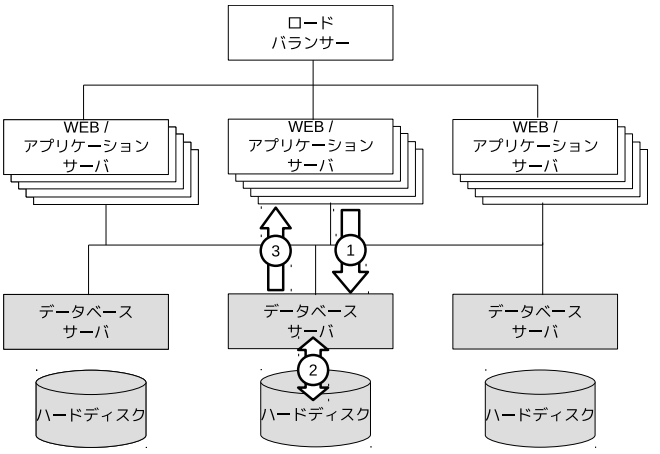


図 1.2 **データベースサーバ構成例**

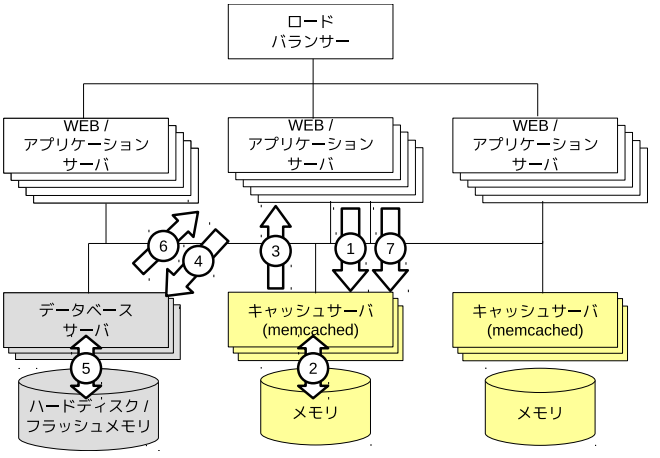


図 1.3 **キャッシュサーバの構成例**

1.3 本研究が解決する問題

前節では，IP ネットワークの高速化によって，プロトコルスタック処理をソフトウェアだけで実現するのが困難になってきていること．データセンタ内のアプリケーションにおいて，スループットの向上だけでなく，低遅延通信が要求され，一部では RDMA に対応している NIC が使われていることを述べた．

RDMA は対応の NIC と専用 API によって，低遅延通信を実現しているが，機構的にデータ送信時に DMA リードが発生し，オーバーヘッドが生じる．DMA リードはペリフェラルである NIC 側から PCI Express を経由して，メインメモリの読み込みを行う

表 1.6 **接続形態によるトランザション処理時間 (SET コマンド ×1, データ長 16 バイト)**

接続形態	トランザクション処理時間
ローカル実行 (Loopback)	11.217 (μs)
リモート実行 (10G NIC)	31.271 (μs)

が，このとき数百 ns の待ち時間が必要なためである．

本研究では，DMA リードを廃することにより，RDMA を使った場合よりも少ない低遅延で通信することを目標とする．目標値として，Mellanox 社が RDMA over Ethernet の遅延としてあげている 1.3 μs[11] があるが，PC のハードウェアを含む比較環境が異なることから，3.2.2.6 節で述べる本研究の実装環境より計測，試算した RDMA 方式での最低通信遅延である 1.213 μs を適用する．

プロトコルスタックの処理をハードウェアでオフロードすることで，今後の IP ネットワークの高速化にも対応する．

本研究の成果により，低遅延通信を要求するアプリケーションの通信処理に関するボトルネックを削減することができ，性能向上を見込める．アプリケーションの例として，ローカルとリモート実行で性能差がでるクライアントサーバ型のアプリケーションがあげられる．

例えば，分散メモリキャッシュサーバである memcached をローカル PC 上で実行した場合と，10 ギガイーサネットによって直接接続されているリモート PC 上から実行した場合のトランザクション処理の実行時間の比較を表 1.6 に示す．

リモート実行と比べて，ローカル実行ではトランザクション処理時間が 64.13% 削減されている．このようなアプリケーションは，ネットワーク通信に関する処理時間が多く占めると考えられ，本研究が問題解決に有効なアプリケーションだと考えられる．

また，PC 側 NIC の転送遅延を削減するだけでは十分ではなく，ネットワーク上の L3 スイッチで発生する IP パケット転送遅延を削減する必要がある．表 1.7 は，市販のギガビット L3 スイッチで 64 バイトパケット時の転送遅延である．IP プロトコルを使って低遅延ネットワークを設計する場合，市販の L3 スイッチは低遅延通信性能より，機能性を重視しているため，低遅延通信に関する IP ネットワークの潜在的能力についての試算が困難である．このことが今後，IP ネットワークを使った低遅延通信を導入する上での問題としてあげられる．

1.4 本論文の構成

本論文は全 7 章で構成される．第 2 章では本研究に関連する技術について述べる．第 3 章では本提案手法である IP-NUMA による低遅延分散アーキテクチャについて述べる．第 4 章では市販の FPGA 上に IP-NUMA NIC を実装し，PC 間通信遅延の削減について

表 1.7 **IP パケット転送遅延 (IPv4 64 バイトフレーム)**

L3 スイッチ名称 (I/F)	転送遅延時間
3750E-24PD (GbE)	4,976 (ns)
AX3630S-24T (GbE)	4,072 (ns)

評価する．第 5 章ではネットワーク上での遅延要因となる L3 スイッチについて，IP 転送処理に必要なプロセスを解析し最小 IP 転送遅延を試算する．また市販の FPGA ボード上 (FIBNIC) に実装し，試算との差異を検証する．第 6 章ではアプリケーション例として memcached に，IP-NUMA 手法と FIBNIC 手法を適用した場合の性能を見積もりし，評価する．最後に，第 7 章で本研究によって得られた結論を述べる．

第 2 章

関連技術と研究

本章では，低遅延通信の提案にあたり，関連技術と研究について調査した．まず，既存の NIC を用い，専用ドライバによってネットワーク I/O の性能を向上させる技術について述べ，次に，本研究が目標とする RDMA 対応 NIC と関連する技術について述べ，最後に PCI Express を用いた低遅延通信について述べる．

2.1 既存 NIC に専用ドライバを用いた高速・低遅延通信

オペレーティングシステム上で標準サポートされている NIC ドライバは，パケットの送受信時にユーザ空間とカーネル空間でのメモリーのコピーが生じ、また 1 パケット毎にシステムコールが発生するため，ネットワークのスループット向上や低遅延化の妨げになっている．

このため，既存 NIC を用いて，ネットワークの I/O 性能を向上させるために以下のドライバが使われている．

2.1.1 Intel DPDK

Intel DPDK[12] は，Linux OS 上で動作するネットワークドライバで，パケットの高速処理に用いられる割り込みをおさえたポーリングモードドライバ (PMD) とライブラリの提供によりネットワークの I/O 性能を向上する．スイッチやトラフィックモニタなどの実装に適しているが，プロトコルスタックが含まれず，バークレイソケットの代わりに専用 API を使うことから，クライアントやサーバアプリケーションからの利用には適さない．一般の NIC を用いることから，データ転送時にディスクリプタリングとデータに対する複数回の DMA 要求が必要になり，遅延要因となる．

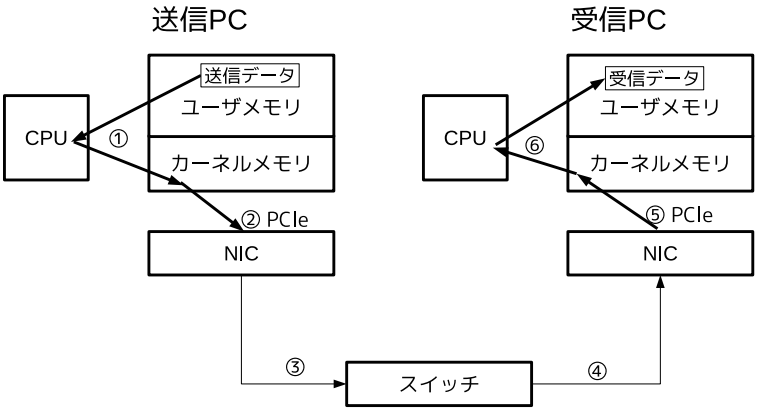


図 2.1 **一般の NIC とドライバによる送受信データの流れ**

2.1.2 netmap

netmap[13] は，FreeBSD OS に含まれ，パケットの高速処理に用いられる．割り込みをおさえたドライバの提供とパケットバッファを NIC とユーザメモリ空間で共有することで，ネットワークの I/O 性能を向上する．プロトコルスタックはカーネル標準か，ユーザ空間上で作成したものを利用することもできる．スイッチやトラフィックモニタなどの実装に適している．一般の NIC を用いることから，データ転送時にディスクリプタリングとデータに対する複数回の DMA 要求が必要になり，遅延要因となる．

2.2 RDMA 対応 NIC による低遅延通信

RDMA[7] は，RDMA コンソーシアムで策定された，ゼロコピー技術を実現する通信プロトコル．高スループット，低遅延通信を目的とし，CPU を関与することなく，ネットワーク経由でユーザメモリ領域へ DMA 転送することができる．ネットワークには低遅延スイッチが利用可能な InfiniBand を用いる．

図 2.1 は，一般の NIC とドライバを用いて 2 つの PC 間でデータの送受信を行うときのデータの流れの概要を示している．

① CPU が送信データをカーネルメモリへコピーしネットワークの各層のプロトコル処理を行う
② DMA を使って NIC にデータを転送する
③ 送信側 NIC よりネットワークへパケットを送信する
④ 受信側 NIC がネットワークよりパケットを受信する
⑤ DMA を使ってカーネルメモリへ受信データを書き込む

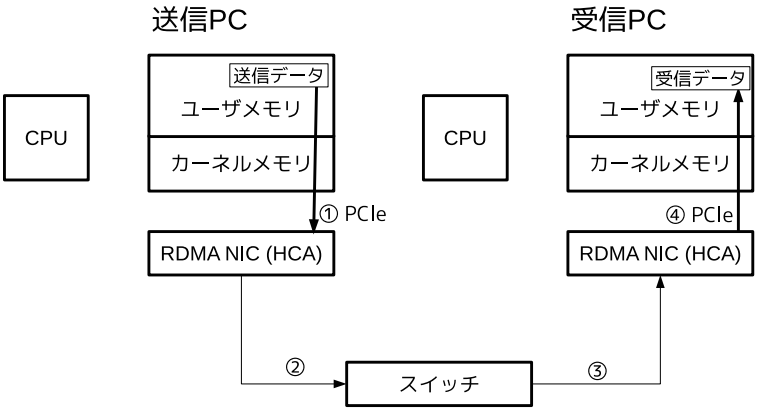


図 2.2 **RDMA 対応 NIC による送受信データの流れ**

⑥ CPU が受信データのヘッダ処理を行ったのち，データをユーザメモリへコピーする

データの送信と受信側それぞれで，データのコピーと CPU によるパケットヘッダ処理が必要になる．このことは，メモリとの間でのデータ伝送によるメモリ帯域と CPU リソースの負荷の増大をまねき，特にメモリ帯域の増大はマルチコアアーキテクチャでは，性能の低下の要因となり，また通信遅延を大きくする要因にもなっている．

次に，図 2.2 は，RDMA 対応の NIC を用いて 2 つの PC 間でデータの送受信を行うときのデータの流れの概要を示している．

① DMA を使ってユーザメモリある送信データを NIC へ転送する
② 送信側 NIC(HCA) よりネットワークへパケットを送信する
③ 受信側 NIC(HCA) がネットワークよりパケットを受信する
④ DMA を使ってユーザメモリへ受信データを書き込む

ユーザメモリ上にあるデータに対して直接 DMA 転送を行うことができるため，メモリコピーを排することができ，また CPU オフロード機能を使うとにより，ネットワークの各層のヘッダ処理をハードウェアで処理することができ，CPU の負担とともに，通信遅延を大幅に削減することが可能である．

本手法の問題は，ネットワークとして InfiniBand を用いることと，専用の API (InfiniBand Verbs) を利用する必要があること，データの送信時にオーバーヘッドが多い DMA リードを利用することである．

Mellanox 社の RDMA 対応 NIC を用いたときの DMA 動作と性能向上について，Design Guidelines for High Performance RDMA Systems[14] に述べられている．CPU と RDMA NIC 間で通信されている PCI Express コマンドを調査しており，CPU から NIC への DMA 指示に，CPU ライトが 2 回必要であることをあきらかにしている．

RDMA 利用時における，オーバーヘッドの詳細については 3.2.2 節で述べる．

RDMA では，アプリケーションの互換性のために，Sockets Direct Protocol (SDP) ライブラリを利用することで，バークレイソケット API を使うことができるが，ネイティブ API である Verbs と比べて遅延やスループットなどの性能が劣るため，低遅延通信の利用には適していない．

2.2.1 iWARP/RoCE

Internet Wide Area RDMA Protocol (iWARP[15]) は，RDMA を IP ネットワーク上で実現したプロトコルの総称で，RDMA over Converged Ethernet (RoCE[16]) は，RDMA をイーサネット上で実現するプロトコルの総称である．InfiniBand ネットワークを利用した RDMA と比べ，イーサネットを利用した RoCE は通信遅延が大きく，IP ネットワークを利用した iWARP はさらに通信遅延が大きい．

本手法の問題は，InfiniBand に依存しない点を除けば，RDMA と同様である．

2.3 PCI Express を用いた低遅延通信

PCI Express は通常，各ホストに存在するルートコンプレックスとアダプタに存在するエンドポイント間のみ通信可能で，異なるホスト間を接続するルートコンプレックス間の通信は不可能である．しかし，特定の Xeon プロセッサでサポートされる Intel Non Transparent Bridge[17] を用いることで，2 台以上のホスト間を PCI Express によって接続し，互いのメモリ空間をアクセスすることが可能になる．PCI Express を直接接続することにより，通信遅延削減には効果的であるが，PCI Express は近距離での接続向けで，専用のスイッチ機器やレーン分のケーブル数を用意する必要があり，接続範囲が限定されることが問題である．

ExpEther[18] は，PCI Express バス上の TLP をイーサネット上にカプセルする手法で，遠隔のペリフェラルを自身のそれとして利用することを目的としている．TLP のカプセリング等，実現のための要素技術は本研究と類似しているが，接続対象をペリフェラルにしていることが異なる．

2.4 まとめ

本章では，関連技術と研究について述べた．

一般の NIC を用いた手法である Intel DPDK と netmap は，ポーリングベースのドライバと専用の API を使うことで，ネットワーク I/O のスループットを向上できるが，プロトコルスタックを含まないため，スイッチやルータなどの IP パケット転送用途やトラフィックモニタ用途向けで，本研究の目的であるアプリケーションの通信ノード用途ではない．

RDMA に対応する RoCE や iWARP は，専用の NIC と API を使うことで，遅延を大幅に削減できるが，DMA リードによるオーバーヘッドが生じることが問題である．

PCI Express を用いてホスト間を接続する Non Transparent Bridge は，ネットワークが制約され接続機器や距離が限定されることが問題である．

第 3 章

低遅延分散アーキテクチャ

本章では，アプリケーション間通信において高速イーサネットの低遅延性能を生かすことができる本研究手法のアーキテクチャについて述べる．まず概要について述べ，次にアプリケーション間通信において遅延の要因になる箇所を分析したのち，IP-NUMA NIC の設計について述べる．

3.1 概要

ネットワークプログラミングの基本モデルはバークレイソケットによって定義された．バークレイソケットモデルはネットワークの初期時代から 30 年以上に渡って利用されている [4]．一方，イーサネットは 10Mbps から 100Gbps と飛躍的に進歩し，インターコネクトバス並みの速度に近づいている．クラウドサービスの普及によりデータセンタ内で，より低遅延通信が求められてきている現在，バークレイソケットモデルにのみ依存し続けると，そのオーバーヘッドによる通信遅延がアプリケーションの性能のボトルネックになる．

図 3.1 は，バークレイソケットを利用してアプリケーション間通信を行った場合のモデル例である．PC1 上のアプリケーション 1 が PC2 上のアプリケーション 2 に対して通信を行う場合，ソケット API を用い，ライブラリ，複数層のプロトコルスタック，NIC ドライバなど多くのソフトウェア処理が必要になり，多くの処理遅延を発生させる．この遅延は，特にサービス要求あたりのサーバ側アプリケーションの処理時間が短いサービス，例えば分散メモリキャッシュサーバとアプリケーション間の通信や，ファイル単位ではなくブロック単位で操作するストレージサービス等で性能劣化の要因となる．本論文は，データセンタ内におけるこうした通信の低遅延化を目指す．

プロセッサ間での低遅延通信を実現する技術として Non-Uniform Memory Access [19] (NUMA) のメモリ共有機構が用いられるが，この技術をデータセンタにそのまま適

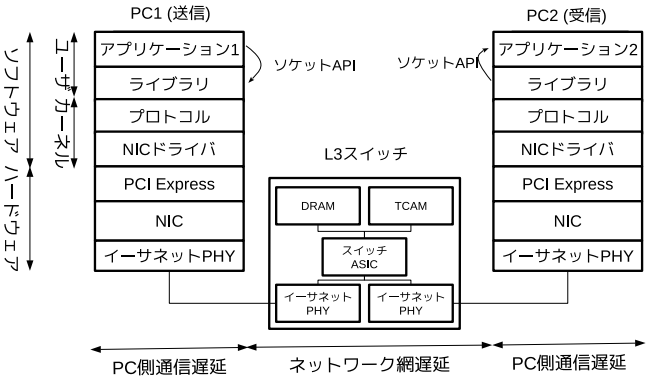


図 3.1 **バークレイソケットモデル**

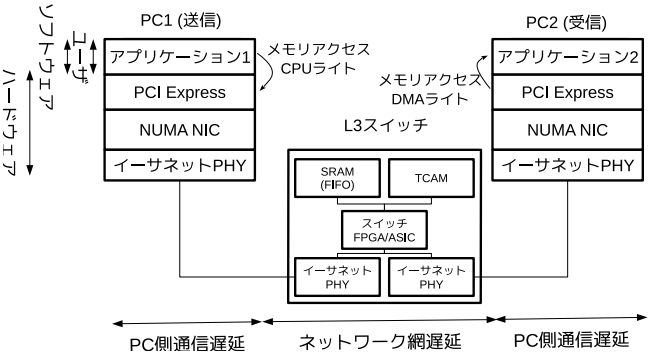


図 3.2 **本研究手法 (NUMA) モデル**

用するには専用ネットワークが必要になり非現実である．

そこで，本研究では，データセンタ内のサーバ間接続で使われている IP ネットワーク上で NUMA に極めて近い機能を実現する手法を提案する．一般的な NUMA は共有メモリを使う単一システムだが，本研究では異なるシステム同士の IP ネットワークを用いた低遅延データ通信手段として共有メモリを使う．共有メモリを用いた低遅延通信を IP ネットワーク上で実現するために，カスタム FPGA を用いた Network Interface Card (NUMA NIC) を開発して，用いる点が特徴である．

図 3.2 に本研究手法の通信モデルを示す．PC1 上のアプリケーション 1 が PC2 上のアプリケーション 2 に対して通信を行う場合，PC1 と PC2 の一部のメモリが共有されているため，PC1 側の CPU が PCI Express 空間上にある共有メモリに書き込みをすることで，NUMA NIC を経由し，パケットが送信される．また，パケットを受信した PC2 側では，NUMA NIC がメインメモリ空間上にある受信バッファに DMA ライトを行うことで，受信処理を完了する．

NUMA NIC は低遅延通信を必要とするアプリケーション向けの NIC で，OS のネットワークドライバに対応せず，またソケット API がサポートされていない．このため，

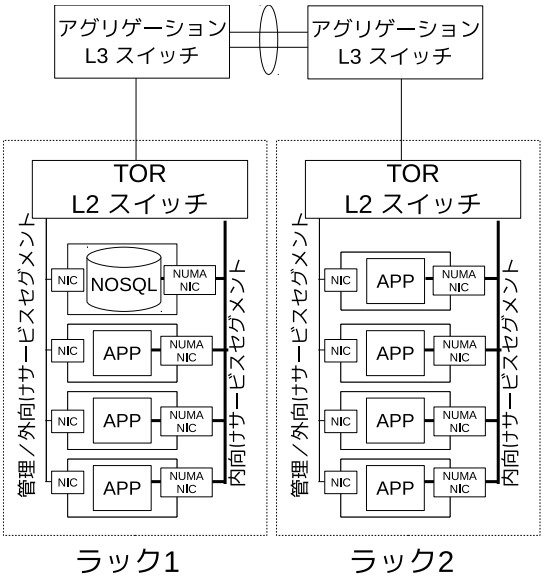


図 3.3 **想定ネットワーク**

ネットワークの管理目的で通常の NIC との共存が必要である。

図 3.3 に本論文で想定するデータセンタのネットワーク構成例を示す．ラック 1 にある NoSQL はサーバ PC，APP はクライアント PC である．それぞれの PC は,「管理／外向けサービスセグメント」と「内向けサービスセグメント」のネットワークセグメントに接続し,「管理・外向けセグメント」は，管理ネットワークと外部インターネットと接続可能なセグメントで通常の NIC を用いる．「内向けサービスセグメント」は，サーバ PC とクライアント PC 間で低遅延通信を行うためのセグメントで，本研究で提案する NUMA NIC を用いる．ラック内の PC 間接続は L2，ラック間は L3 スイッチで接続している．

3.2 通信遅延の分析

図 3.4 は PC 内におけるパケット通信処理の流れと，遅延要因を示している．ソフトウェア部とハードウェア部の処理について次に述べる．

3.2.1 ソフトウェア処理

ユーザ空間上のアプリケーションからパケットを送信すると，以下の処理が行われる (図 3.4 の番号と対応している).

① アプリケーションがソケット API 経由でライブラリを呼び出す.
② ライブラリは sys_sendto() などのシステムコールを発行する.

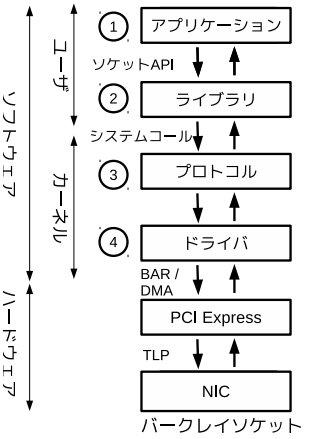


図 3.4 **既存のパケット通信処理の流れ**

③ カーネルモードで各プロトコルスタックの処理が実行され，ドライバの ndo_start_xmit() を呼び出す.

④ ドライバは PCI Express 内のメモリ空間である Base Address Register(BAR) メモリに割り当てられている Memory Mapped I/O (MMIO) を通して NIC の制御レジスタや Direct Access Memory (DMA) の操作を行う.

Linux カーネル 3.5 の環境においての Brandeburg の計測によると，パケットの送受信全体では 5,722 ns[5] 以上のソフトウェア処理時間が必要とされている．内訳はドライバが 1,172 ns，アプリケーションが 356 ns，それらを除くプロトコルスタックが 4,194 ns で，NIC とのハンドリングを行うハードウェア処理はドライバに含まれると考えられるが，DMA など CPU 処理を必要としない多くのハードウェア処理はソフトウェア処理時間に含まれない.

また，本研究が目標とする RDMA 方式による通信遅延の 1,245 ns (図 3.8 より) を大幅に超過することから，ソフトウェア処理は極力削減する必要がある.

3.2.2 ハードウェア処理

ハードウェア処理は CPU が PCI Express 経由で BAR メモリ空間をアクセスするか，NIC などのペリフェラルが CPU に接続されているメインメモリ (以後，メインメモリと記述する) を PCI Express バスでアクセスすることで行われ，処理の主体は CPU とペリフェラルのいずれも考えられる.

CPU が主体となる場合のメモリアクセスを本論文では，CPU リード，CPU ライト，ペリフェラルが主体となる場合のメモリアクセスを DMA リード，DMA ライトと定義する.

PCI Express のアクセス遅延はいずれも，数十 ns から数百 ns と大きい．本節では，

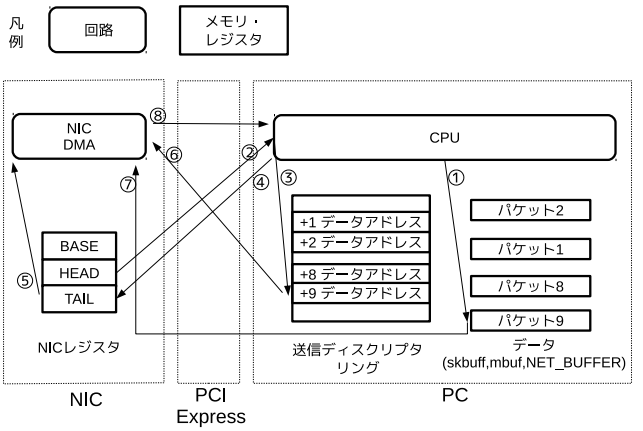


図 3.5 **NIC 送信時の処理フロー例**

既存 NIC の問題を PCI Express の遅延に着目しながら調査し，NUMA に適した PCI Express の利用手法をあきらかにする．

NIC を使って，パケット送信と受信を実現するにはデータ受け渡し処理の主体によって以下の方法がある．

- パケット送信の場合
  1. パケットデータを DMA リードする
  2. パケットデータを BAR メモリ空間へ CPU ライトする
- パケット受信の場合
  1. パケットデータをメインメモリへ DMA ライトする
  2. BAR メモリ空間よりパケットデータを CPU リードする

一般的な NIC や Remote Direct Memory Access (RDMA) ではパケットデータ送信に DMA リード，受信に DMA ライトを用いるが，パケットデータの転送以外にも，様々なところで CPU リード・ライトや DMA リード・ライトが使われている．

NIC[20] を使ってパケット送信をするときに，CPU と NIC と PCI Express 間で必要になる最小手順を，図 3.5 に示す．PC 枠内のデータは，パケットの RAW データ，送信ディスクリプタリングは送信すべきパケットの物理アドレスが記載されているリングバッファである．リングの管理情報は NIC 内のレジスタに割り当てられている．BASE は，リングの先頭アドレス，HEAD はリングの先頭で，NIC によって送信が完了した場所を示し，TAIL はリングの最後で，送信パケットを追加するときに CPU が更新する．図は，パケット 8 までの送信が完了し，あらたにパケット 9 を送信しようとしている．以下に詳細を述べる．

① 送信すべきデータを，各層のヘッダ情報を含めてパケット 9 に CPU ライトする
② HEAD を CPU リードし，リングバッファに空きがあることを確認する
③ リングバッファ 9 にパケット 9 の物理アドレスを CPU ライトする

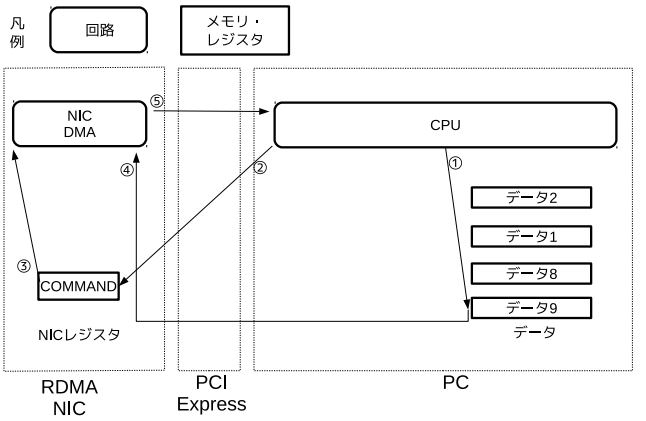


図 3.6 **RDMA NIC 送信時の処理フロー例**

④ リングバッファ 9 の位置へ TAIL を CPU ライトする
⑤ NIC が TAIL を更新したことを検知する（実行レジスタへの書き込みで検知させる場合もある）
⑥ NIC がリングバッファ 9 を DMA リードする
⑦ NIC がパケット 9 を DMA リードする
⑧ NIC が CPU へ割り込み通知する（プログラマブル）

遅延が大きい PCI Express へのアクセス回数は，CPU リードが 1 回，CPU ライトが 1 回，DMA リードが 2 回，合計 4 回となる．送信リングバッファの更新は 1 パケット毎ではなく，複数パケット分がたまってから 1 回で行うこともできるが，その場合，スループットは向上するが，通信遅延は大きくなる．

次に，RDMA に対応した NIC を使った場合の例を図 3.6 に示す．RDMA の初期化処理が完了し，通信バッファがユーザメモリ空間に割り当て済みとし，以下に詳細を述べる．

① データ 9 に CPU ライトする
② 実行レジスタに CPU ライトする
③ NIC が送信指示を検知する
④ NIC がデータ 9 を DMA リードする
⑤ NIC が CPU へ割り込み通知する（プログラマブル）

PCI Express へのアクセス回数は，CPU ライトが 1 回，DMA リードが 1 回，合計 2 回となり，通常の NIC と比べて大幅に削減されている．

今回 NUMA NIC での実装設計するにあたり，この点に関する PCI Express バスの特性を調査した．調査内容と結果を次節で述べる．

表 3.1 **実験環境**

CPU	Intel Core i7-4770 @ 3.4GHz
チップセット	Intel Z87 Express
FPGA ボード	Xilinx KC705
PC バス	PCI Express Gen2×8
SDK	Vivado 2014.4
IP Core	7Series Integrated Block for PCIe3.0 R.4

3.2.2.1 PCI Express バスの遅延計測

本節では，PCI Express バスの遅延特性を把握するために，メインメモリに対する DMA リード・ライト，BAR メモリに対する CPU リード・ライトの計測実験を行う．計測には表 3.1 の実験環境を使い，ペリフェラルである FPGA ボード上に計測用の回路を実装した．

3.2.2.2 DMA リード・ライト

DMA リードしたときの遅延計測結果を，表 3.2 に示す．表中のリンクはジェネレーション × レーン数を示し，Gen1 が PCI Express 1.0 (2.5GT/s)，Gen2 は PCI Express 2.0 (5GT/s)，Transaction Layer Protocol (TLP) は PCI Express バスの CPU 側にあるルートコンプレックスとペリフェラル側にあるエンドポイント間で用いられるプロトコル，TLP 内訳は，PCI Express バス上で発行された TLP のペイロード長 (Double Word (DW)) × TLP の数で， Double Word は 4 バイトである．

計測は 100 回行い，計測には FPGA 内の 250MHz クロックカウンタを用いた．64 ビット物理アドレスを指定して 4 バイトの DMA リードを行ったところ，Gen2×8 のときにおいて DMA リード遅延が平均で 467 ns であることが確認できた．

次に，DMA ライトしたときの処理計測結果を，表 3.3 に示す．計測は 1 秒間行い，DMA ライトの処理数をカウントした．64 ビット物理アドレスを指定して 64 バイトの DMA ライトを行ったところ，Gen2×8 のときにおいて 1 秒あたり 47,890,550 回の DMA ライトが行われた．1 回あたりの遅延に換算すると 21 ns であることが確認できた．

注意すべき点として，DMA ライトに関しては，要求が途中の経路でバッファリングされ，また要求に対する実行結果などの応答はない．このため，21 ns は実際のメモリに書き込みされるまでの遅延ではなく，PCI Express バスの占有の開始から解放されるまでの時間であり，物理メモリまでの書き込み遅延を計測したものではない．

表 3.2 **DMA リード処理時間**

リンク Gen× レーン	4 バイト DMA リード (ns)
Gen2×8	467
TLP 内訳 ペイロード × 数	1DW×1 パケット

表 3.3 **DMA ライト処理数**

リンク Gen× レーン	64 バイト DMA ライト	
	処理数/秒	遅延 (ns)/処理
Gen2×8	47,890,550	21
TLP 内訳 ペイロード × 数	16DW×1 パケット	

3.2.2.3 CPU リード

表 3.4 は，BAR メモリ空間に対する CPU リード，CPU ライト遅延の特性を調べるために，メインメモリと PCI Express 上にある BAR メモリ間でのメモリコピー処理時間を比較したものである．CPU リードは BAR メモリからメインメモリへのコピー，CPU ライトはメインメモリから BAR メモリへのコピー，WC は Write-Combining[21] を示している．

CPU リード・ライトは，計測することによるオーバーヘッドを回避するため，1 万回ループした場合の平均を用いた．メモリコピーにはカーネル内の memcpy 関数，計測には RDTSC (Read Time Stamp Counter) を用いた．

リンクのクロックやレーン数の向上に伴い，CPU リード処理時間は削減しているが，リンクの帯域に反比例にはならない．これは CPU リード要求が応答とともに逐次行われているためで，要求パケットを先行的にバースト送信できないためと考えられる．

次に 4 バイト長と比べて 64 バイト長での CPU リード処理時間が 16 倍かかる理由として，CPU リード時に TLP ペイロード長が指定できない問題があげられる．DMA リード時はペイロード長に複数 DW を指定できるが，CPU リード時のペイロード長に制約がある．

表 3.4 の最下行で示すように，4 バイトの CPU リードでは，1DW のリード要求 TLP

表 3.4 **memcpy 処理時間**

リンク Gen× レーン	4 バイト			64 バイト		
	CPU リード (ns)	CPU ライト (ns)	WC CPU ライト (ns)	CPU リード (ns)	CPU ライト (ns)	WC CPU ライト (ns)
Gen1×1	4,824	386	18	76,950	6,190	335
Gen1×4	2,983	226	11	47,299	3,192	83
Gen1×8	2,696	223	11	42,940	3,159	41
Gen2×1	2,791	223	11	43,944	3,219	167
Gen2×4	1,894	224	12	30,103	3,193	41
Gen2×8	1,731	223	11	27,134	3,220	21
TLP 内訳 ペイロード × 数	1×1	1×1	1×1	1×16	1×16	16×1

が 1 回発行されるが，64 バイトの CPU リードでは，1DW のリード TLP が 16 回発行され，非効率である．

3.2.2.4 CPU ライトと Write-Combining

CPU ライトに関しては，通常モードと Write-Combining モードが存在する．通常モードではメモリアクセスの順番が保証されるが，Write-Combining ではバッファリングされるためアクセスの順番が保証されず，バースト転送が行われる．

表 3.4 の最下行で示すように，64 バイトの CPU ライトでは，通常モードでは 1DW (4 バイト) 長ペイロードのライト TLP が 16 回発行されるが，Write-Combining モードでは 16DW (64 バイト) 長ペイロード TLP が 1 回発行される．この結果，通常 CPU ライトに対して Write-Combining モードでは大幅に処理時間を削減できる．

3.2.2.5 PCI Express コマンドの検討

ハードウェア処理ではパケット送信の 2 つの実現方法として，DMA リードと，BAR メモリ空間への CPU ライトが考えられたが，実測すると Write-Combining モードでの CPU ライトが有効である．Write-Combiling モードではアクセス順序が保証されないが，NUMA NIC での利用はパケット送信時に限定され，送信するパケット単位でメモリへ連続書き込みすることによって，影響を受けない．

DMA ライトの遅延時間については，3.2.2.2 節で述べたが，メモリまでの書き込み時

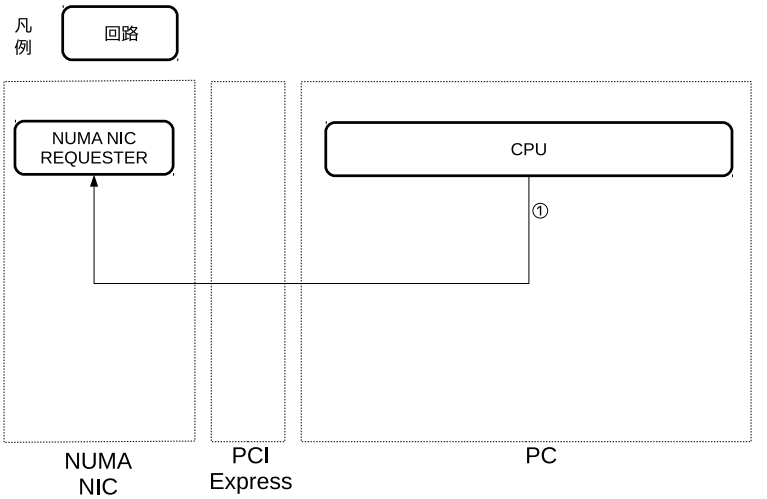


図 3.7 **NUMA NIC 送信時の処理フロー**

間ではない．CPU ライトと DMA ライトは，要求する方向が異なるだけなので，本論文では，DMA ライトの遅延時間を CPU ライトと同じ 223 ns (表 3.4) と見積もる．ペリフェラルとメモリまで往復通信が必要な，DMA リードの遅延時間は，467 ns (表 3.2 より) で，DMA ライトの遅延時間は，片道通信なので半分程度となるため，この数値は妥当と考えられる．

パケット受信の 2 つの実現方法として，DMA ライトと，CPU リードが考えられるが，DMA ライトを用いた場合，遅延が 223 ns なのに対し，CPU リードを用いた場合，表 3.4(リンク Gen2×8 の 64 バイト) より遅延が 27,134 ns と大きい．

以上のことから通信遅延を削減するために，NUMA NIC では，プロトコルスタックの処理をハードウェアにオフロードし，パケット送信では，DMA リード転送の代わりに BAR メモリへ CPU ライト，パケット受信では，メインメモリへの DMA ライトを使うことが可能な NIC が有効であると考える．

パケット送信をするときに，CPU と NIC と PCI Express 間で必要になる手順を，図 3.7 に示す．パケットの送信は，PCI Express 上にある BAR メモリ空間へ CPU ライトすることで実現する．

3.2.2.6 パケット送受信時のハードウェア遅延

本節では，前節までに計測した PCI Express バスの遅延をもとに，本研究が目標とする RDMA 方式の NIC と，本研究である NUMA 方式の NIC を使った場合の，パケット送受信遅延を見積もりし，目標とする通信遅延を設定する．

図 3.8 は，64 バイトデータの送受信するときの遅延の詳細を示す．PCI Express のリンクは Gen2×8 とする．このときの PCI Express バスとの接続は 128 ビット幅 (@250MHz) である．RDMA 方式は，3.2.2 節で述べたように，パケット送信時に CPU

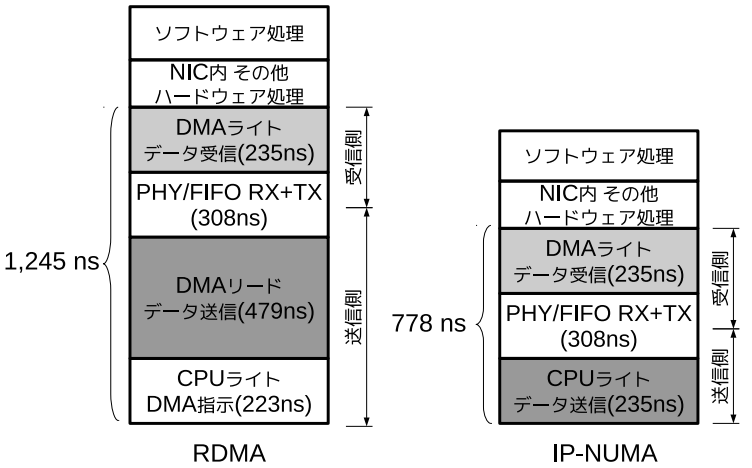


図 3.8 **送受信遅延の比較**

ライトが 1 回，DMA リードが 1 回行われる．CPU ライトは，NIC に対する DMA リード実行の指示のため 4 バイト，DMA リードは，メインメモリより 64 バイト転送が必要なため，計測値にある 4 バイト転送時と比べて 3 サイクル必要になり，

467(ns) + 4(ns) × 3(サイクル) = 479 ns となる．

PHY/FIFO RX+TX は，送受信 PHY と FIFO の遅延で，評価回路を FPGA で実装したものを，IXIA 社の IxNetwork[22] を用いて計測した．

パケット受信時に 64 バイトの DMA ライトが行われる．4 バイトの DMA ライトと比べて 3 サイクル必要になるため，

223(ns) + 4(ns) × 3(サイクル) = 235 ns となる．

RDMA 方式の場合の通信遅延合計は 1,245 ns になるが，これには NIC 上のユーザロジック回路とソフトウェアの処理による遅延時間が含まれていない．

本研究では，データ長が 64 バイトのときは 1,245 ns，4 バイトのときは 1,213 ns の数値を設定し，通信遅延を数値以下にすることを目標とする．

2.2 節で述べたように，Mellanox 社の RDMA 対応 NIC では，DMA アクセス時に 2 回の CPU ライトが必要であり，本目標数値は 1 回の CPU ライトを想定しており，目標数値として理論上の最小値を設定している．

3.3 IP-NUMA の設計

本節では，NUMA に対応した NIC の設計について述べる．IP プロトコルをインターコネクトとして利用した NUMA 構成であることから，本手法を IP-NUMA と呼ぶ．

IP-NUMA は，パケットの送信時に BAR メモリ空間への CPU ライト，パケットの受信時にメインメモリへの DMA ライトを使い，上記以外での BAR メモリ空間への CPU

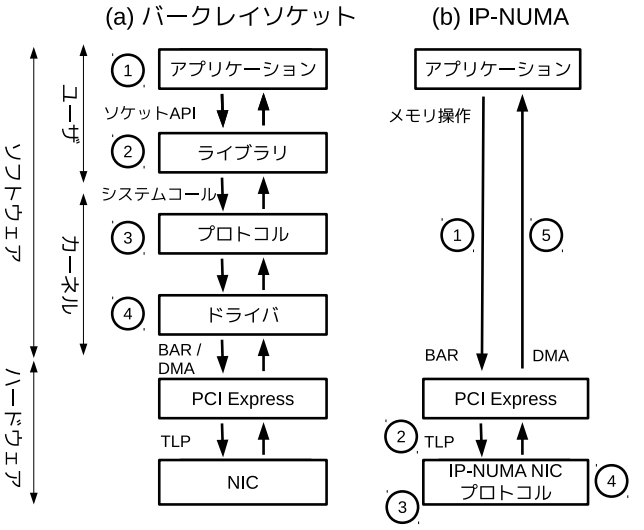


図 3.9 **処理フロー**

リード，CPU ライトを行わないことで，PCI Express バス上で発生する遅延を軽減し，プロトコル処理をハードウェアへオフロード化することで，全体の通信遅延を削減する．

図 3.9 に従来方法であるバークレイソケットと本論文で提案する IP-NUMA の違いを示す．IP-NUMA におけるパケットの送信と受信の流れについて以下に述べる．

① アプリケーションが PCI Express 内の BAR メモリ空間に CPU ライトを行う
② IP-NUMA NIC が PCI Express バス上に送られたライト要求 TLP を受信する
③ IP-NUMA NIC が IP パケットを生成し，送信する
④ 受信した IP-NUMA NIC が IP パケットを分解，デコードする
⑤ DMA ライトによって，メインメモリにデータを書き込み，受信する

本手法では，他の PC のメモリ空間を直接 CPU からアクセスでき，NUMA システムに近い機能を持つ．汎用 IP プロトコルを使うことから，本論文では以降 IP-NUMA と呼ぶ．

IP-NUMA は，メモリ空間が異なる PC 間を接続するための機構として，ローカル物理メモリアドレスと相手側 PC の IP アドレスと物理アドレスに変換するページマップテーブル，PCI Express 要求を IP パケットに変換する TLP over IP がある．

3.3.1 ページとページマップテーブル

IP-NUMA では，1 対 n での PC またはアプリケーション間通信を行う必要があるため，メモリ空間をページ単位に分割し，それぞれのページに異なる PC のメモリ空間に割り当てる．

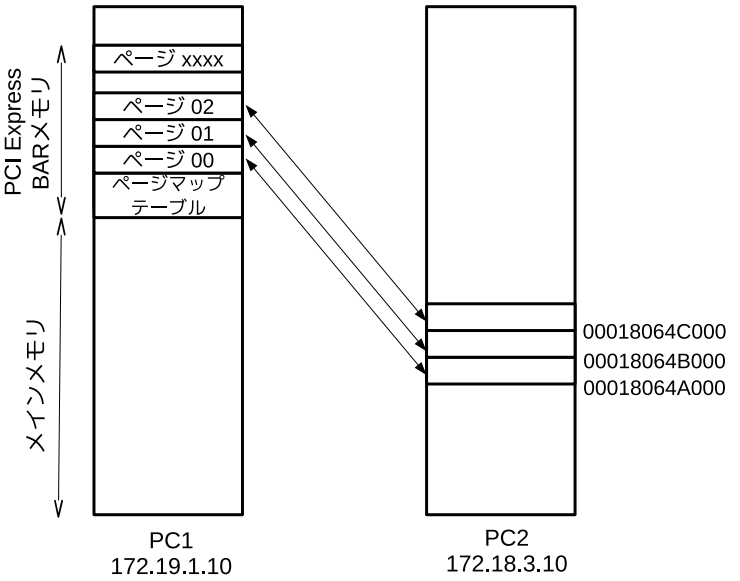


図 3.10 **ページとページマップテーブル**

ページは，MMU (Memory Management Unit) で利用される一般用語だが，本論文内では BAR メモリ内に割り当てられる 4KB ブロックを意味する．図 3.10 で示すように BAR メモリ内は，ページ領域と対応するページマップテーブルから構成される．ページは任意の他 PC の物理メモリを割り当てることができ，ページを通して他の PC のメモリ空間を直接アクセスできる．

ページマップテーブルは図 3.11 で示すように 1 ページエントリー毎に，16 ビットの相手先 PC ID，48 ビットの相手先物理アドレスを保持する．PC ID は図下で示す PC ID テーブルによって，相手先 IP アドレスと MAC アドレスに変換される．

1 ページあたり，8 バイト分のページマップテーブルが必要なため，1 ギガバイトのページ領域の場合，2 メガバイトのページマップテーブルが必要になる．

サーバ PC など同時に 2 台以上の異なる PC と接続する場合は，各ページ毎の PC ID を変えておく．図 3.11 の例では，IP-NUMA NIC の IP アドレスとサブネットマスクは 172.19.1.10/16，デフォルトゲートウェイ IP は 172.19.1.1 を示し，ページ 0 のメモリ領域をアクセスすると，別セグメント上にある PC ID2 の IP アドレス 172.18.3.10 の PC の物理アドレス 18064A000 から 18064AFFF 番地へのアクセスに変換され，ページ 100 をアクセスすると PC ID1 の 172.19.3.4 のアクセスに変換される．

PC ID テーブルには別ネットワークセグメント属する IP アドレスを登録できるが，MAC アドレスに関してはデフォルトゲートウェイを含む同一セグメント上の MAC アドレスである．IP-NUMA はデフォルトゲートウェイ以外のルーティング情報をサポートしていない．

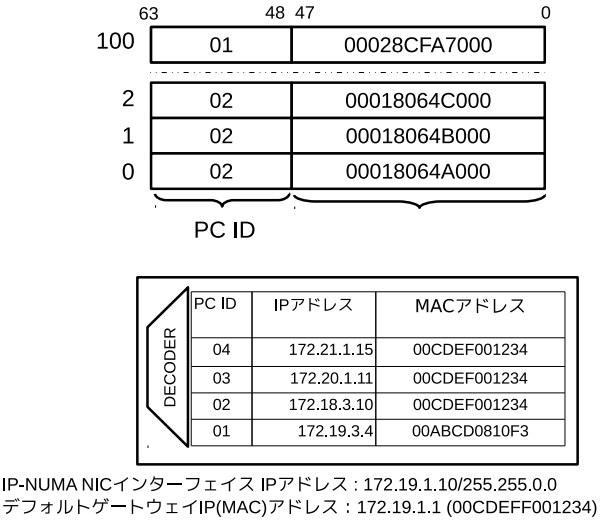


図 3.11 **ページマップテーブル詳細 (上) と PC ID テーブル詳細 (下)**

3.3.2 TLP over IP

IP-NUMA では CPU が BAR 内のページ領域をアクセスした際に PCI Express 上で発行される TLP をペイロードとし，UDP，IP，MAC ヘッダをハードウェアによって付与する．このとき，ページマップテーブルに基づいて TLP ヘッダのメモリアドレス部の書き換えを行うが，TLP ヘッダ長を 3DW から 4DW 拡張する必要がある．これは，BAR 内のページ領域は 32 ビット空間に割り当てられるため，TLP ヘッダが 32 ビットアドレス対応の 3DW で発行されるのに対し，相手先 PC が 4 ギガバイト以上のメモリ空間を持つ場合，物理アドレスが 64 ビット対応にできる 4DW ヘッダを発行する必要があるからである．

パケット形式を図 3.12 で示す．UDP プロトコルのペイロード部にマジックコードと 4DW の TLP ヘッダとライトの場合は TLP ペイロード，最後にエンドコードを格納する．

3.3.3 データ転送処理

IP-NUMA では PC 間のデータ転送を遠隔 PC に対する CPU リードでも実現できるが，PC 間の IP ネットワーク上で要求毎にパケットの往来が生じ，CPU リード処理性能が低下するため，用いない．一方，CPU ライト処理において IP-NUMA NIC は要求を一時的にバッファリングし，遠隔 PC からの応答を待つ必要がないため，イーサネットの帯域を超えない限り，CPU ライト処理性能は低下しない．

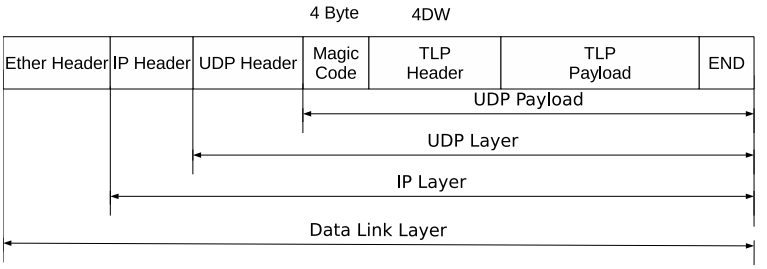


図 3.12 **パケット形式**

図 3.13 を用いて 2PC 間でデータを往復転送する手順を以下に示す.

① PC2 の受信領域 2 を共有領域 2 として PC1 のページ領域 1 に割り当て，PC1 の受信領域 1 を共有領域 1 として PC2 のページ領域 2 に割り当てる．受信領域は通信バッファにあたり，アプリケーション内で利用する変数領域などを共有領域として割り当てることはセキュリティ上好ましくない．共有領域の割り当てについては次節で述べる.

② PC1 の CPU が送信すべきデータをページ領域 1 に CPU ライト，PC1 の IP-NUMA NIC が物理アドレスを変換後，Memory write TLP を含んだ IP パケットを送信する.

③ PC2 の IP-NUMA NIC が受信した IP パケット内の TLP をチェックし，ルートコンプレックスに TLP を送信することで，共有領域 2 に DMA ライトが行われ，PC2 の CPU がデータを受け取る

④ PC2 の CPU が返信すべきデータをページ領域 2 に CPU ライト，PC2 の IP-NUMA NIC が物理アドレスを変換後，Memory Write TLP を含んだ IP パケットを送信する.

⑤ PC1 の IP-NUMA NIC が受信した IP パケット内の TLP をチェックし，ルートコンプレックスに TLP を送信することで，共有領域 1 に DMA ライトが行われ，PC1 の CPU がデータを受け取る

PCI Express バス上で CPU 側およびペリフェラル側 DMA のいずれでもリード処理を行わないことで，バス上で生じる遅延を削減する.

3.3.4 データ保護

IP-NUMA は，PCI Express バスより全メモリ空間を直接アクセスできるため，データを保護する仕組みが必要になる.

このため，他の PC からのアクセスを許可する共有領域テーブルを設ける．図 3.14 にあるように，テーブルは共有を許可する物理アドレスと長さ，許可する相手の IP アドレスとサブネットマスクから構成される．他の PC から許可されていないメモリ領域にアク

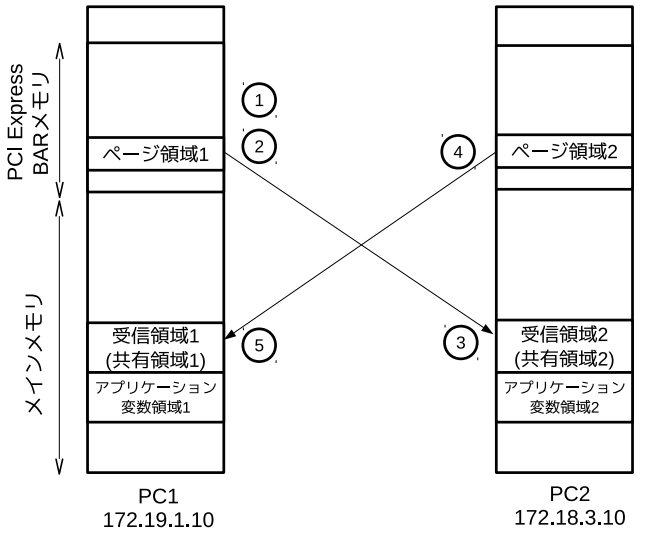


図 3.13 **メモリマップ**

セス領域があった場合，ライト要求は無視し，リード要求は 00 を返し，別途エラーコードを要求元 PC へ送信する．

図 3.14 の一番下の例では，IP アドレス AC120000/FFFF0000 (172.18.0.0/18) のアクセスを許可している．この範囲で意図しない IP アドレスからのアクセス要求があった場合，物理アドレスが 1806A0000 から 16K バイトのみ有効となり，それ以外の範囲は無視される．共有領域は動的に確保され物理アドレスの推測が困難なことから，共有領域のサイズを受信領域以上に多く取らないことがデータ保護に有効と考えられる．

3.3.5 再送処理と UDP プロトコル

IP-NUMA は UDP プロトコルを使うため，パケットの再送処理が必要になる．受信側は ACK パケット を返すことで，送信側は送信処理を完了し，送信バッファから破棄する．

IP-NUMA は，データセンタ内での利用を前提としており，光ファイバーなどのメディアによる伝播遅延が小さい．送信側が送信後に再送要求するまでのタイムアウト時間は，6.6 節の表 6.5 で示す往復遅延時間から、3〜40 μs が適切だと考えられる．

物理アドレス	長さ	IPアドレス	サブネットマスク
000300000000	00001000	C0A80300	FFFFFF00
000200000000	00001000	C0A80200	FFFFFF00
0001A96FC000	00001000	C0A80100	FFFFFF00
00018064A000	00004000	AC120000	FFFF0000

図 3.14 **共有領域テーブル**

第 4 章

IP-NUMA の実装と評価

本章では，IP-NUMA NIC を FPGA 上に実装し，アプリケーション間の通信遅延について，評価を行う．

4.1 システムの実装

我々は，IP-NUMA による PC 側通信遅延の削減を確認するため，FPGA 上に IP-NUMA の機能を持つ IP-NUMA NIC のプロトタイプを 10 ギガイーサネットとギガビットイーサネットについて作成した．10 ギガイーサネットの実装環境を表 4.1 に，ギガビットイーサネットを表 4.2 に示す．図 4.1 が NIC 内のブロックダイアグラムで，左図がリクエスタ，右図がサーバを示す．

表 4.1 **IP-NUMA 10G NIC 実装環境**

FPGA ボード	Xilinx KC705
FPGA	XC7K325T-2FFG900C
Ethernet	10GBASE-SR
PC バス	PCI Express Gen2×4 (64bit×250MHz)
SDK	Vivado 2014.4
IP Core	7 Series Integrated Block for PCIe 3.0 R.4
	10 ギガビットイーサネット PCS/PMA 5.0
ページ領域	1 GB (4KB×26,214 pages)
ページマップテーブル	2 MB (8B×26,214 pages)

表 4.2 **IP-NUMA ギガビット NIC 実装環境**

Dev Kit	LatticeECP3 Versa [23] Development Kit
FPGA	LFE3-35EA-8FN484C
Ethernet	1000BASE-T
Peripheral Bus	PCI Express Gen1x1
SDK	Diamond 2.2
SLICEs	7671 out of 23,616 (32%)

4.1.1 リクエスタ

リクエスタはアプリケーションからのメモリ要求を処理する．アプリケーションが他の PC に対してメモリ要求をし，リクエスタが送信するまでの過程を図 4.1 を使って説明する．

① アプリケーションが PCI Express 内のページ領域へ CPU ライトをする
② PCI Express からの TLP を受信し，ページマップテーブルを用いて送信先 PC の IP アドレスと物理メモリアドレスを探索する
③ メモリライト要求の IP パケットをネットワーク上へ送信する

4.1.2 サーバ

サーバはリクエスタからの要求に応じてサービスを行う．リクエスタからのパケットを受信して，DMA メモリアクセスをするまでの過程を以下に説明する．

① IP パケットを受信する
② 共有領域テーブルを用いて正当なメモリアクセス要求であるかを調べる
③ ルートコンプレックスに対して DMA ライト要求を送信する

4.2 通信遅延の評価

本節では，IP-NUMA とバークレイソケットとの通信遅延比較について述べる．評価は PC 間でデータ送受信する ping-pong プログラムを実行し，RTT を計測することで行う．通信遅延の計測を目的とするため，データの転送および処理時間の影響を受けにくくするため転送データサイズを 4 バイトとした．

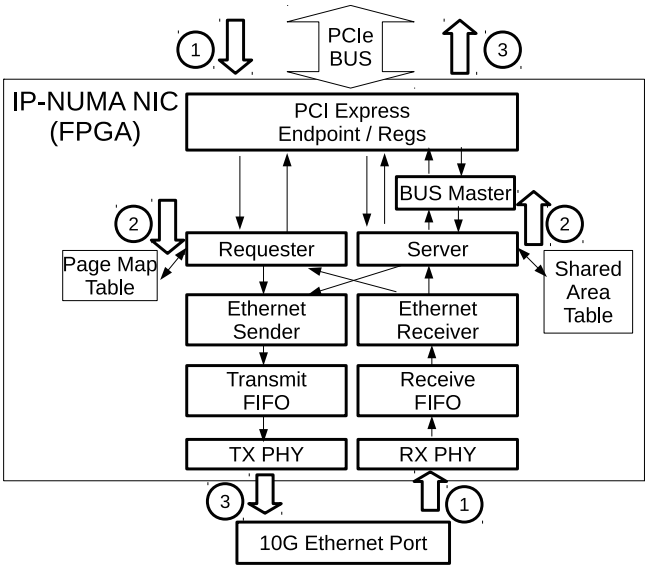


図 4.1 **IP-NUMA の概要構成図**

4.2.1 バークレイソケットによる ping-pong

ping-pong は PC 間で int 型 4 バイトのデータをカウントしながら往来を繰り返すプログラムで，プロトコルに IPv4 UDP を使う．図 4.2 はバークレイソケットを用いた ping-pong プログラムの流れを示し，PC1 はクライアント，PC2 はサーバとして事前に待機し，計測は PC1 上で行った．以下に流れを解説する．

① PC1 が変数 A の内容を 1 加算し，PC2 へ sendto 関数を使って送信する
② PC2 が recv 関数を使って，受信した値を変数 A に受信する
③ PC2 が変数 A の内容を 1 加算し，PC1 へ sendto 関数を使って送信する
④ PC1 が recv 関数を使って，受信した値を変数 A に受信する

上で示すような一連のデータ往復の処理を以後 1 ループと呼ぶ.

4.2.2 IP-NUMA による ping-pong

図 4.3 は IP-NUMA を用いた ping-pong の流れを示す．変数 RBUF は相手方からのデータを受信する受信領域で，キャッシュラインサイズである 64 バイトアライメントにて宣言し，公開する共有領域として両 PC とも設定する．変数 BAR は，BAR 内にあるページ領域を示すポインタ変数で，該当するページマップテーブルに相手型の受信領域の物理アドレスページを設定し，送信時の CPU ライト先になる．

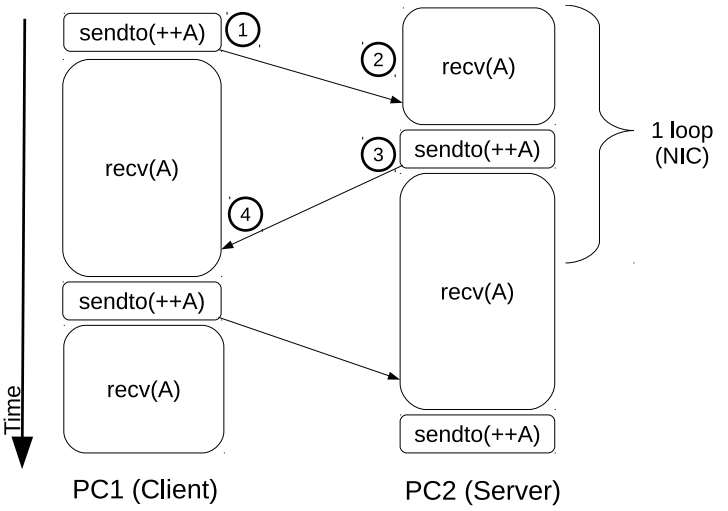


図 4.2 **バークレイソケットによる ping-pong**

IP-NUMA 通信では，通信前に互いの受信領域の物理アドレスを交換する初期化処理が必要になる．この処理には通常の NIC を用い，手順を①から③で示す．

次に，④から⑦で示す手順で 1 ループを行う．

① PC1 は受信領域 RBUF1，PC2 は受信領域 2 を確保し，PC1 が RBUF1 の物理アドレスを PC2 へ送信する

② PC2 は RBUF1 の物理アドレスを受信し，自分の受信領域 2 である RBUF2 の物理アドレスを PC1 へ送信する．PC1 の物理アドレスを BAR2 に該当するページマップテーブルに割り当てる (図中の MAP)

③ PC1 は RBUF2 の物理アドレスを受信し，PC2 の物理アドレスを BAR1 に該当するページマップテーブルに割り当てる．RBUF1 を初期化する．

④ PC1 は受信した RBUF1 の内容を変数 A にコピーし (1 回目は除く)，1 加算する．BAR1 に変数 A の内容を代入する．このとき，ライト要求 IP パケットが PC2 へ送信される．

⑤ PC2 は変数 RBUF2 の書き換えをポーリングし，データ受信を検知する．

⑥ PC2 は受信した RBUF2 の内容を変数 A にコピーし，1 加算する．BAR2 に変数 A の内容を代入する．このとき，ライト要求 IP パケットが PC1 へ送信される．

⑦ PC1 は変数 RBUF1 の書き換えをポーリングし，データ受信を検知する．

受信時に繰り返し実行される，Flush Cache Line (clflush) 命令はデータキャッシュを破棄するもので，メモリの書き換え検知までの時間を削減するために用いる．参考文献 [24] によれば，clflush を用いない場合は，メモリの書き換え検知に 999.81 μs かかり，これは，Linux カーネルの 1 tick にあたる．原因としては評価環境では DMA ライトによる書き換えを CPU 内のキャッシュ機構が検知できず，tick のタイミングで生じるコンテ

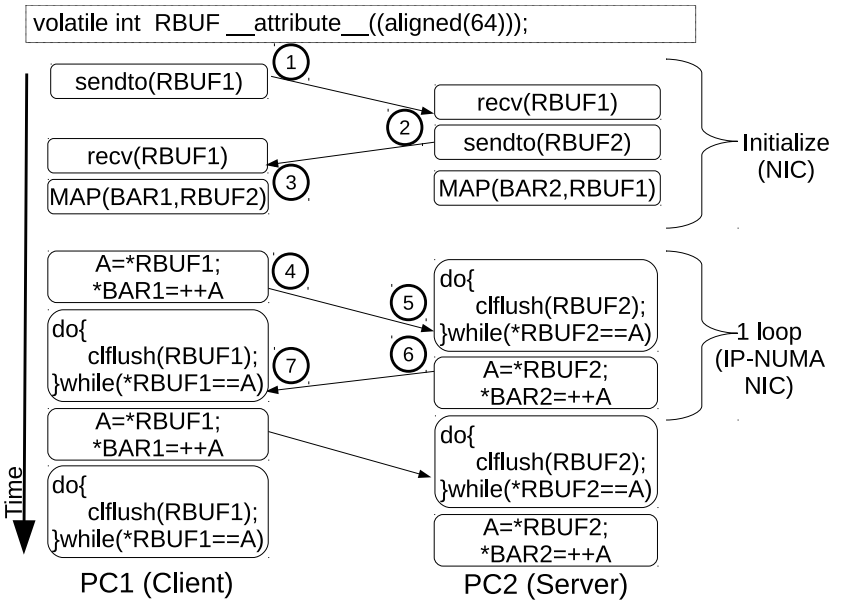


図 4.3 **IP-NUMA による ping-pong**

キストスイッチでキャッシュがクリアしたときに，正しく認識をするためと考えられる．

4.2.3 計測

10 ギガビットイーサネットの計測環境を表 4.3 に，ギガビットイーサネットの計測環境を表 4.4 に示す．実装の都合で，市販 NIC と比べて IP-NUMA NIC の PCI Express のリンク帯域が小さい．イーサネットインターフェイスの帯域よりは大きいため，スループット面での問題はないが，NIC とホストメモリ間のメモリ転送時間が大きくなるため，遅延面においては不利ではあるが，結論に影響を与えるものではない．

2 台の PC とバークレイソケット計測用の NIC，および実装した IP-NUMA NIC を用い，PC 間を光または UTP ケーブルで直接接続した．ping-pong プログラムをそれぞれ 100 万回ループ実行し，ループ前後の CPU 消化サイクルを RDTSC によって計測することで時間を算出した．

図 4.4 に 10 ギガビットでの 1 秒あたりのループ数を示す．IP-NUMA はバークレイソケットと比べてループ数が 10.44 倍となった．これは PCI Express バス上で発生する遅延を削減したことと，IP 処理の全てをハードウェアにオフロードしたことによる，CPU サイクルの削減が要因と考えられる．

表 4.5 は RTT で 100 万回ループの平均値である．10 ギガビットの IP-NUMA では RTT が 2.162 μs なので，片側遅延は約 1.081 μs になる．3.2.2.6 節より，IP-NUMA の送受信遅延は 0.778 μs と試算された．差分の 0.303 μs は，NIC 内上のユーザロジック処理と PC 上のソフトウェア処理と考えられる．

表 4.3 **10G 計測時の仕様**

CPU	Intel Core i7-4770 @ 3.4GHz × 2
NIC (バークレイソケット時)	Intel X520-SR2 (PCIe G2×8) × 2
IP-NUMA NIC	Xilinx KC-705 (PCIe G2×4) × 2
Ethernet	10GBASE-SR
OS/Kernel (PC1)	Fedora 21/3.19.7-200.fc21.x86_64
OS/Kernel (PC2)	Fedora 21/3.19.7-200.fc21.x86_64
BIOS	VT-d disable

表 4.4 **ギガビット 計測時の仕様**

CPU	Intel Core i7-4770 @ 3.4GHz × 2
NIC (バークレイソケット時)	Intel I350T2 (PCIe G2×4) × 2
IP-NUMA NIC	Lattice ECP3Versa (PCIe G1×1) × 2
Ethernet	1000BASE-T
OS	Fedora 23
Kernel	4.4.7-300.fc23.x86_64

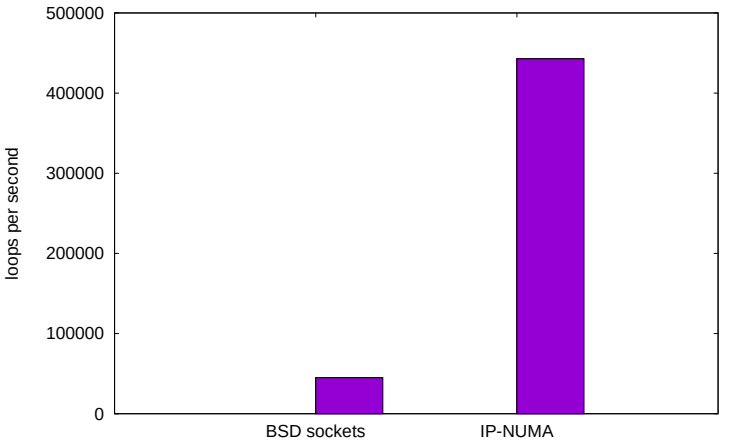


図 4.4 **ping-pong プログラムによるループ/秒の比較 (10 ギガ)**

本研究の目標通信遅延は，RMDA での試算値の 1.213 μs なので，約 11% の削減を達成できた．値には，NIC のユーザロジック処理とソフトウェア処理の時間が含められていないので，11% を超える削減が達成できたといえる．

また，この値は表 3.4 より，PCI Express メモリ空間からの 4 バイト memcpy 時間である 1.731 μs より小さい．このことから，IP-NUMA ではローカルの BAR メモリ領域から CPU リードによるメモリコピーするより，IP ネットワーク上にある他の PC に対して，CPU ライトによるメモリコピーをしたほうが遅延が小さいといえる．

表 4.5 **Round Trip Time**

手法	10 ギガビット RTT (μs)	ギガビット RTT (μs)
バークレイソケット	22.580	49.379
IP-NUMA	2.162	10.308

ギガビットの IP-NUMA では RTT が 49.379 μs なので，片側遅延は約 24.690 μs だった．

4.3 考察

IP-NUMA NIC は RDMA を用いた場合と比べて，通信遅延を 11% 以上削減することができた．

この節では IP-NUMA の特性や制限等について議論する．

4.3.1 CPU 負荷

IP-NUMA はプロトコルスタックのハードウェアオフロードなどで CPU 負荷を軽減しているが，以下の点において CPU の負荷を増やしている．

- DMA リード処理の代わりに，BAR メモリへの CPU ライトを行う
- データの受信をポーリングで検知する

汎用 NIC や RDMA では，データ送信時に DMA 転送が行われるため CPU に負担はかからない．IP-NUMA ではデータ送信を CPU ライトによって行うため，その間 CPU が占有される．このため，PCI Express バスのリンク速度を高速化することで，NIC 内にバッファリングできるので，CPU の占有時間を短くすることが有効である．例えば，PCI Express Gen3×16 では，DDR3-SDRAM 並みの転送帯域を持つので，メモリコピーに必要な時間で送信処理を完了でき，CPU を解放できると考えられる．

4.2.2 節より，IP-NUMA では受信時のポーリングによって DMA ライトを迅速に検知することができるが，CPU 資源を消費する．このため，メモリへの DMA ライト時に割り込み通知する機能を追加することによる，CPU 時間の削減が有効と考えられるが，遅延が大きくなると考えられる．RDMA ではデータ受信の完了方式をポーリングか割り込みか選択することができる．

以上のことから，送信より受信時のポーリングが CPU の負担が大きい．高速ネットワークの持つ低遅延性能を生かすためには，割り込みを使わず，ポーリングが必要である．現在の CPU はマルチコアを前提としていることから，アイドル中のコアや，サービス要求待ちのコアに通信のためのポーリング処理を負担させることが必要と考えられる．

4.3.2 データ転送

データ転送時における利用効率について検討する．IP-NUMA では図 3.12 で示すように，アクセスすべきペイロード以外に 4DW(16 バイト) の TLP ヘッダが付与されるため，この部分がデータ転送時のオーバーヘッドとなる．TLP ヘッダには未使用や値が固定されているビットが多数存在するため，TLP ヘッダ圧縮が有効である．また，TLP 単位ではなく連続したデータをバッファリングすることで，TLP ヘッダの数を削減することが可能である．この場合，遅延が大きくなることが懸念されるが，PCI Express のリンク帯域をイーサネットの物理インターフェイスの帯域より大きくさせることで，NIC とホスト側メモリ間の転送遅延をさらに下げることができるので，バッファリングによる遅延の増加を削減することができる．

また，IP を含むプロトコルスタックを全てハードウェアにオフロードしているため，イーサネットの 40/100/400G などの高速化に対してワイヤーレート対応が容易である．

今回はデスクトップ向け CPU とチップセットの組み合わせで評価したが，4.2.2 節で述べた clflush 命令によるキャッシュフラッシュが常時必要になる問題は，Intel Direct Data I/O (DDIO) [25] 機能を持つ Xeon E5 以上のサーバ向け CPU とチップセットの使用によって不要になり，遅延削減を含めた転送性能が改善される可能性がある．

転送時のセキュリティにおいては，IP は汎用的なネットワークのため，不正なアクセス要求を排除する仕組みが必要になる．不正 PC よりのアクセスを防止するために，IPSec を用いて信頼できる PC と暗号化通信を行うことが有効であると考えられる．

本実装では，含めなかったアクノリッジプロトコルのサポートがデータ転送保証をするために必要となる．

4.3.3 通信遅延の改善

4.2.3 で述べたように，計測した IP-NUMA NIC の PCI Express のリンク帯域は市販 NIC と比べて小さく，10 ギガビット版では 50%，ギガビット版では 12.5% の帯域だった．PCI Express の帯域は送信と受信側の NIC とのホストメモリ間の転送遅延に影響するため，より大きな帯域で実装できれば，通信遅延のさらなる削減が可能と考えられる．

4.3.4 アプリケーションへの適用

IP-NUMA ではメモリアクセスによって，PC 間のデータ転送が行われるよう完全に抽象化しているため，OS の種類や有無を問わず，電源さえ入っていればリクエスタ，サーバいずれも利用可能である．そのため，プログラミングモデルがバークレイソケットと大幅に異なり，現状のプロトタイプを適用するのには大幅なコードの書き換えが必要になる．このためバークレイソケット API に準ずるラッパーの作成を検討したい．

表 4.6 **片道遅延**

手法	直接接続 (μs)	ラック内接続 (μs)	ラック間接続 (200m) (μs)
バークレイソケット	11.29	11.67	13.81
IP-NUMA	1.08	1.46	3.60

一般的な NUMA 構成では，共有メモリとして利用するためスピンロック機能が必要になるが，IP-NUMA ではこれらの用途では使わないため，特にこれらの機能を用意しない．

本件研究は同じアーキテクチャの PC サーバを対象とするため，異なる CPU アーキテクチャ間でのバイトオーダー問題は発生しない．しかし，今後は External Data Representation (XDR) [26] の適用についても検討する．

IP-NUMA が有効なアプリケーションについては，6.7.1 節で述べる．

4.3.5 データセンタでの適用

評価は直接接続で行ったが，本節ではデータセンタ内での適用について検討する．

図 3.3 の例では 2 つのラックがアグリゲーションスイッチで接続している．ラック間の総ケーブル長は 200m とする．ラック内の接続は，TOR の L2 スイッチ 1 台で行われ，ラック間の接続は，TOR の L2 スイッチ 2 台とアグリケーション L3 スイッチ 2 台の合計 4 台で行われる．

表 4.5 は，RTT なので片道遅延に換算すると，バークレイソケットで 11.29 μs，IP-NUMA で 1.08 μs となる．例えば，低遅延転送を考慮しているスイッチである ARISTA の 7150 Switches の転送遅延は 380 ns[27] なので，ラック内のスイッチ転送遅延は 380 ns，ラック間のスイッチ転送遅延は 1520 ns となり，ラック間を 200m の光ファイバーケーブルで接続すると伝搬遅延はおよそ 1 μs[28] となる．表 4.6 がラック内接続とラック間接続 (ケーブル距離 200m) での片道遅延を試算したものを示す．

このことから，本手法がラック内接続において 86%，ラック間接続において 74% の遅延を削減できることを示し，データセンタ内での適用が有効であると考えられる．

第 5 章

IP 転送遅延の解析と FIBNIC による検証

市販のルータ（L3 スイッチ）の多くは，QoS や VPN などの機能性を重視し，IP 転送遅延の削減について考慮されていない．本章では，低遅延通信が必要なアプリケーションで利用するネットワーク設計をする上で重要な L3 スイッチの IP 転送遅延を調べるために，PHY（PHYsical layer Chip）の処理遅延時間と IP 転送に必要なプロセスを解析し，最小の IP 転送遅延時間を試算し，あきらかにする．

また，本手法を市販の FPGA ボード上に設計・実装した FIBNIC を使って，IP 転送遅延の試算値と実測値を比較検証する．

5.1 ネットワーク網の通信遅延の削減

本節では，ネットワーク網内における L3 スイッチの IP 転送遅延の削減について検討し設計をする．

クラウドサービスの普及により，異なるホスト間でのストレージ共有や VM マイグレーションなど IP ネットワークにおける低遅延通信が求められようになってきている．しかし，IP はインターネットのような広域通信での利用を前提としたプロトコルであり，複数パケットの ACK 応答を一度に行うなど，通信遅延に対して寛容なプロトコルであり，InfiniBand[29] のような構内通信を前提としたプロトコルと比べるとネットワーク機器の転送遅延が大きい．ギガビット L3 スイッチの転送遅延は 4〜16 μ s になる，これは光が 1.2〜4.8 キロメートル進むのに相当する時間であり，複数の L3 スイッチで接続をするデータセンタなどでは近距離接続のメリットが失われてしまう．

IP の下位にあたるイーサネット層ではフレーム転送方式としてストアアンドフォワー

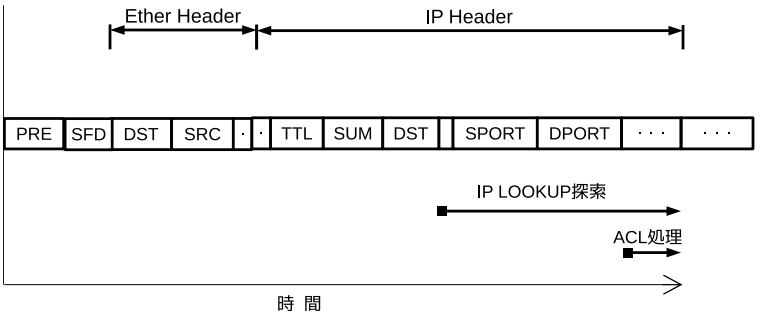


図 5.1 **フレーム書き換え・ACL に必要なアクセス項目**

ドとカットスルー [30] がある．ストアアンドフォワード方式はフレーム全体を受信してから，送信するためフレームサイズの大きさに応じて遅延が大きくなる．一方，カットスルー方式では転送に必要なヘッダ情報を受信した後に転送処理を開始するのでフレームサイズの大小に関わらず転送遅延が一定となり，また転送遅延が小さくなる．

本研究では高速 IP パケット転送の実現と，また転送遅延の要因のひとつであるパケットバッファを用いず，FIFO を組み合わせたパイプライン構成を持つ手法について提案する．

本論文では L2 と L3 を同一レイヤーにて扱うため，フレームとパケットという言葉が混在するが，フレームは L2，パケットは L3 視点で用いる．

5.1.1 既存の問題

本研究の目的は IP パケット転送時に必要な最小転送遅延をあきらかにすることである．

そのためには，各プロセスで消費される時間について把握しておく必要がある．最初に，IP パケット転送に必要なフレームの書き換えとフレームの配送，それぞれの処理について整理する．

5.1.1.1 フレームの書き換え

IP パケット転送に必要な RFC1812[31] の要件を満たすために，下記項目の参照と書き換えが必要になる．

- 参照項目
  - 送信先 MAC アドレス，送信元 MAC アドレス，フレームタイプ，Time To Live(TTL)，IP CHECKSUM，送信先 IP アドレス
  - さらに Access Control List(ACL) によって必要になる項目：送信元 IP アドレス，送信元ポート番号，送信先ポート番号
- 書き換え項目

– 送信先 MAC アドレス，送信元 MAC アドレス，TTL，IP CHECKSUM

フレームの受信から書き換え完了までは，以下の処理が必要になる．

(1) プリアンブル部および Start Frame Delimiter（SFD）を受信する
(2) 送信先 MAC アドレスではじまる項目の取得を開始する
(3) IP TTL を 1 減算する
(4) 送信先 IP アドレスまでを取得する
(5) Forwarding Information Base(FIB) より Next Hop IP アドレスを調べるため，IP Lookup を開始する
(6) ポート番号など ACL 処理に必要な項目を全て取得する
(7) ACL および TTL の評価を行い，パケット転送の許可，不許可を決定する
(8) IP Lookup の探索が終了する
(9) ARP テーブルより MAC アドレスとポート番号を探索し取得する
(10) 受信したフレームを配送プロセスへ転送開始する
(11) 送信先 MAC アドレスおよび送信元 MAC アドレスを書き換える
(12) RFC 1624[32] に基づいて IP CHECKSUM を書き換える．

必要なポートに対してフレーム出力を開始するためには特に下記の 2 つの処理が終了する必要がある．

処理 1 は，(7) である．5 タプルを条件に用いた場合は IPv4 で最小で 46，最大で 86 バイト，IPv6 で 66 バイトの読み込みが必要になる．

処理 2 は，(8) の IP Lookup の探索である．

5.1.1.2 フレームの配送

IP パケット転送用に書き換えられたフレームは，一旦パケットバッファメモリに保存される．その後，ポートへの出力準備の完了後，パケットバッファから読み込まれ，ネットワーク上に送信する．

パケットバッファがあることにより，NPU[33]（Network Processor Unit）や CPU に高負荷時に受信があったような場合や，輻輳時など一時的にパケットに対する処理がすぐに施せない状況でもパケットが失われない．

その一方，パケットバッファへの書き込み，読み込み時には各ポートとの競合が生じ，遅延の要因になる．

5.1.1.3 論理回路からみた IP パケット転送処理

5.1.1.4 主要回路

図 5.2 は，イーサネットを構成する主要回路である．PHY はネットワーク物理層のインターフェイスで，FPGA/ASIC などのディジタル回路との信号変換を行う．

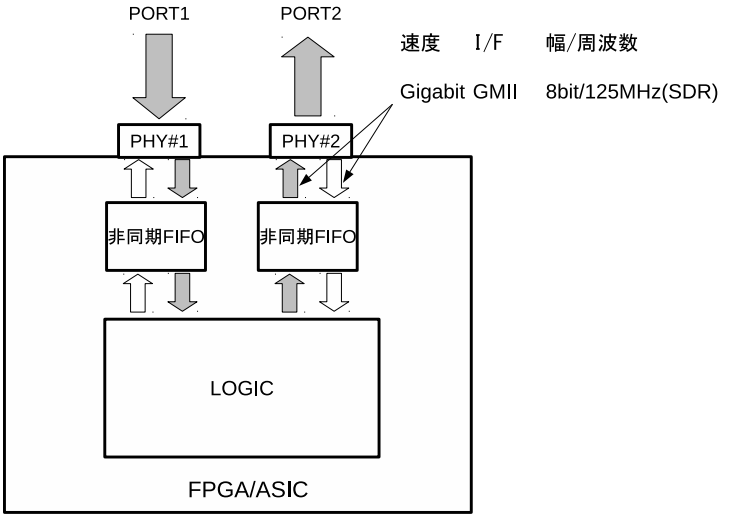


図 5.2 **イーサネットを構成する主要回路**

ディジタル回路と接続するデータ幅と周波数は図 5.2 のインターフェイス（I/F）によって決定される．PHY の基準クロックと論理回路の基準クロックは周波数あるいは位相が異なるため，非同期 FIFO を経由して接続される．転送遅延の要因としては，PHY，非同期 FIFO，論理回路内に大別できる．

5.1.1.5 PHY による遅延

PHY 部分の転送遅延を計測するために，図 5.2 のギガビットイーサネットの構成にて，PORT1 と PORT2 を長さ 10cm の UTP ケーブルで直接接続し，PHY#2 から PHY#1 へフレームを送出し，送信と受信 PHY 間の遅延時間を調べた．図 5.3 より，上の波形が PHY#2 よりの送信許可信号でプリアンブル部の送信開始で 1 になる．下の波形が PHY#1 よりの受信データ有効信号で，プリアンブル部の受信開始で 1 になる．上下の波形の差異が送信および受信 PHY 間の遅延を示し，356 ns であることが確認できた．

5.1.1.6 転送遅延の試算

次にギガビットイーサネットにおいて転送遅延の試算を行う．図 5.2 より，論理回路は 8 ビット幅/125MHz で動作し，1 サイクルで 1 バイトの転送をする．

5.1.1.1 節 (4) より，IP Lookup の開始をするためには送信先 IP アドレスが必要なため，図 5.4 より IPv4 では先頭 42 バイトを読み，42 サイクル消費する．

また，PHY の遅延は 356 ns 消費し，125MHz において 1 サイクルは 8ns なので，$42 \times 8 + 356 = 692$ ns となる．

この数値には，IP Lookup と ARP の探索時間，非同期 FIFO，同期 FIFO，データパス遅延などほとんどの論理回路の処理遅延が含まれていない．そのため，この数値以下の

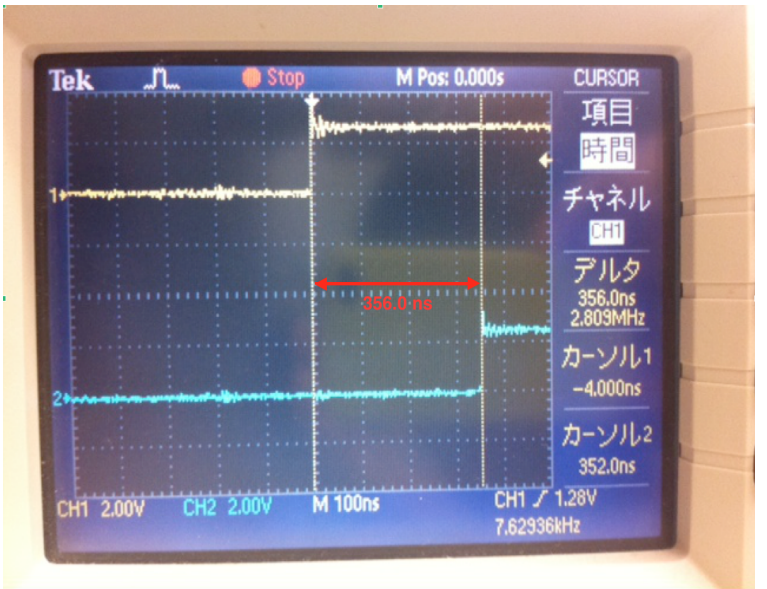


図 5.3 **送受信 PHY 間の遅延**

上が PORT2 の GMII Transmit Enable 信号

下が PORT1 の GMII Receive Data Valid 信号

時間によるパケット転送は実現不可能である．

5.1.2 パイプラインによる IP パケット転送

本研究の手法は，入力から出力までの工程のパイプライン化とパケットバッファを使わず，入出力ポート間を直接転送することによって，全体の転送遅延を削減する．

工程としては，フレームの書き換え処理とフレーム転送処理があり，それぞれの回路が FIFO によって接続される．

5.1.2.1 フレーム書き換えのパイプライン化と先頭タギング

図 5.4 は，ギガビットインターフェイスにて PORT1 から受信したフレームを書き換えてフレーム転送するプロセスを示したものである．

IP Lookup の探索時間と ACL の評価時間の上限を求めることによりプリアンブル部の受信開始から転送開始までの遅延サイクルを固定化できるので，パケット書き換え処理までの工程についてパイプライン化が可能である．

必要なパイプラインステージ数は遅延サイクル数となり，最初のステージはフレームの読み込み処理，最終ステージは転送モジュールへつながる FIFO への書き込み処理となる．

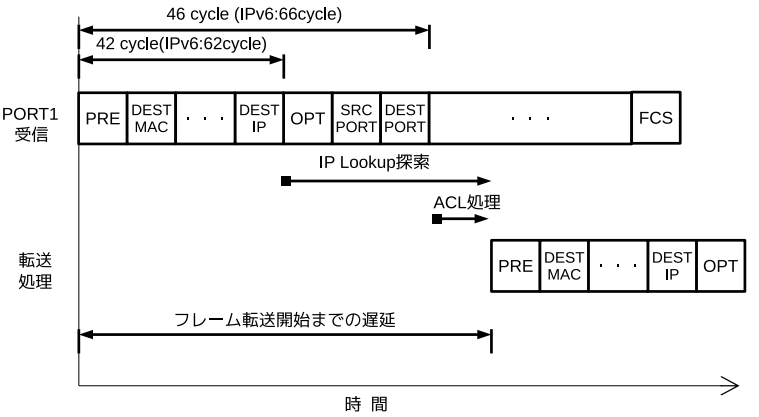


図 5.4 **フレーム受信と転送**

このとき，パケット先頭のプリアンブル部をタグで置き換える．タグには送信先のポート番号や拡張スタック用ネットワーク番号など次へ接続すべき回路の情報が記述され，後工程である転送モジュールにおいて，パケット先頭を読み込んだ時点で接続先を判別し，パイプラインストールをおこさず転送処理を開始できる．

図 5.5 にパイプライン処理によるタギングの例を示す．IP Lookup 探索時間は 20 サイクルとし，それまでの 42 サイクル（図 5.4）との合計 62 サイクルがステージ数となる．

図 5.5 の 0 ステージが読み込み処理，62 ステージが転送モジュール FIFO への書き込み処理，数値はプリアンブルを含めたフレーム位置を示し，0〜7 はプリアンブルとスタートフレームデリミタ，8 よりイーサーネットヘッダがはじまる．ステージ 0 が 62 バイト目のときは，62 ステージではフレーム先頭（プリアンブル部）にあたる．このときには既に 0 ステージの 42 バイト目に探索した IP Lookup の結果を受け取ることができるので，転送先ポート番号などをタグとして書き換えることができる．また，DST/SRC MAC，TTL，IP CHECKSUM など書き換える必要がある項目に関しても，62 ステージの該当フレーム位置にて書き換えを行う．TTL が 0 あるいは，ACL によってドロップするフレームと判断された場合は，転送用 FIFO への書き込みは行わない．

VLAN，カプセリング等の処理が必要な場合は，パイプラインのステージを増やす．例えば，62 ステージ目に VLAN の処理を挿入し，63 ステージ目を転送処理にすればよい．

5.1.2.2 IP CHECKSUM

IP パケット転送では TTL を 1 減算させるため，IP CHECKSUM を書き換える必要があるが， 後ろに計算対象の IP ヘッダがあるため，そのまま再計算を行うとパイプライン処理が乱れてしまう．そのため，TTL 減算分だけを考慮した RFC1624[32] に従い，再計算を行う．

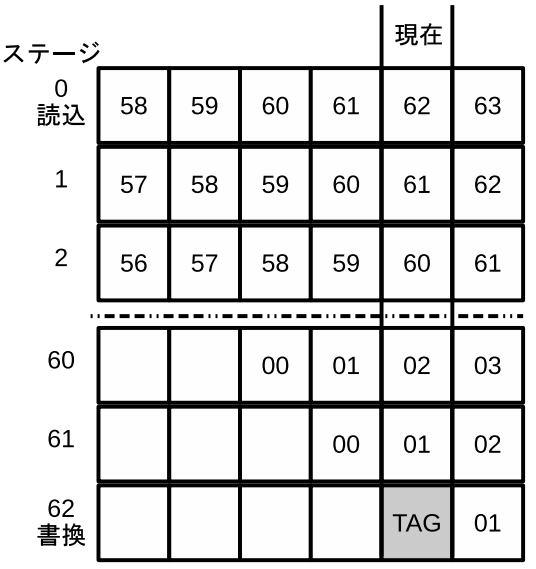


図 5.5 **パイプライン処理**

5.1.2.3 Frame Check Sequence(FCS)

FCS でエラーフレームを発見した場合，処理は続けるが該当するフレームの FCS 値が必ずエラーになるように転送処理を行う．また，ハードウェア障害などによって入力時に FCS エラーが頻繁に発生する場合に備えて，フレーム書き換え処理の入力前に 1 フレーム分受信をバッファリングして FCS が正しくない場合は破棄する迂回経路に切り替えられる必要がある．

5.1.2.4 FIFO によるフレーム転送

図 5.6 は，入力ポート毎にフレーム書き換えモジュールが FIFO を経由し，各出力ポートに対応したフレーム転送モジュールに接続されていることを示す．4 ポート構成の場合，入力ポート毎に自身を除く全てのポート宛の FIFO と ARP/ICMP 処理用 FIFO の合計 4 つの FIFO を持つことから，全体では 16 個の FIFO が必要になる．ARP/ICMP は該当するプロトコルを処理するモジュールである．

図 5.7 は，複数からポート 4 へ同時入力があった場合を示している．各 FIFO の大きさは，最大フレームサイズで 2 フレーム分のサイズを確保しておく．フレーム転送モジュールは，ポート 1 からの FIFO を読み込み，ポート 4 へ送信している．ポート 2 からの FIFO は空のため，フレーム書き換え処理からの転送は FIFO に送られる．ポート 3 からの FIFO には，既に閾値（最小フレームサイズ）を超える 1 フレームのデータが格納されているため，フレーム書き換え処理からの転送は 1 フレーム分破棄される．次に，フレーム転送モジュールが 1 フレーム分送信を終えると，ラウンドロビンにて次の要求がある

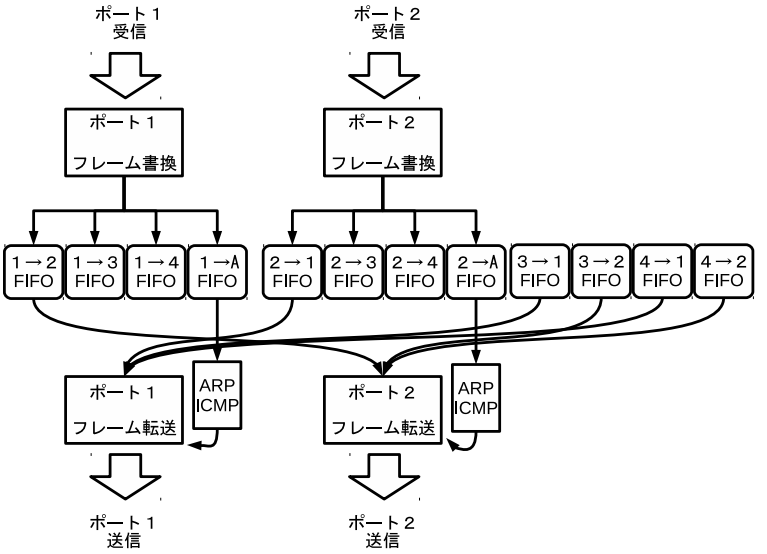


図 5.6 **主要回路モジュールと FIFO（4 ポート構成例）**

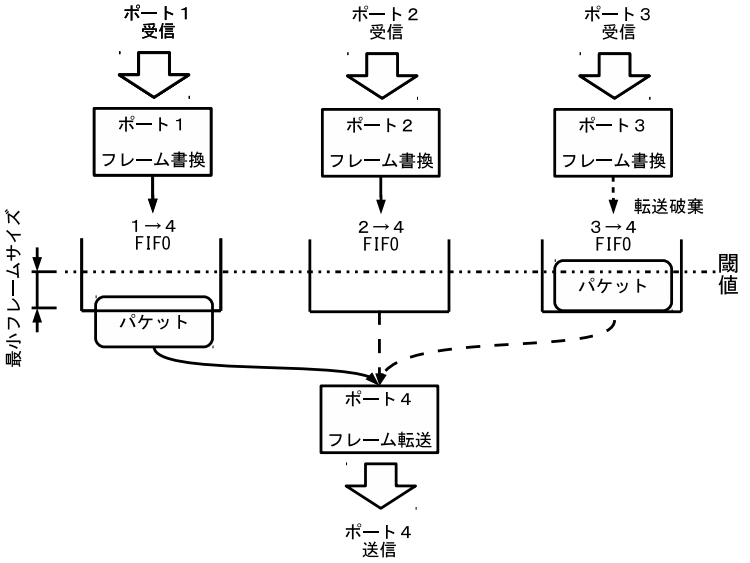


図 5.7 **転送回路への複数入力**

ポート 3 の FIFO から読み込みを開始する．

本手法では出力ポート別の入力ポート毎に FIFO を設け，最大 1 フレームしか保持しないため，以下の制約がある．

(1) ひとつのポートに出力が集中するような輻輳状態において，直ちにパケットロスが発生する．

(2) ポート数の二乗数だけの FIFO が必要になるため，ポート数を大幅に増やせない．

（1）に関して，図 5.7 のケースでひとつの送信ポートに対して，同時に 3 ポートからワイヤーレートでフレームが入力されるような輻輳状態が発生すると，それぞれの入力ポートのパケットロス率は 2 つ目のフレームより 66.6% となる．

このため輻輳発生時からパケットロスが発生するまでのフレーム数を増やす場合は，FIFO のサイズと閾値をあげる必要がある．

しかし，無限のサイズの FIFO を持つことは不可能であり，入り口でフロー制御をしない限り本質的な解決はできない．また，バッファされた分だけ転送遅延が生じる．

本方式は個々の中継ネットワーク機器内でフレームの保証をするために，転送遅延を増加させるよりも，転送遅延を少なくすることによってエンドツーエンドが ACK/NACK などで，直接フレームの保証をすることを期待している．

（2）に関しては，実装物の規模によるが，4〜16 ポート程度を想定しており，それを超える場合は多段化接続をする．

5.2 システムの実装

5.2.1 FIBNIC

本研究手法をルータシステムとして動作させるためには，OSPF や BGP などのルーティングプロトコルがサポートされる必要がある．そこで我々は，オフロード IP パケット転送機能を備えた NIC を市販で入手が用意可能な設計・実装とした．そのため，本方式の機能を兼ね備えた NIC（Network Interface Card）を市販で入手が容易な FPGA ボード上で開発した．本実装を以後，FIBNIC と呼ぶ．

FIBNIC は受信したルーティングプロトコルなどのフレームを NIC 機構にそのまま転送することで，PC ルータとして経路情報の交換などをサポートし，それ以外のフレームはハードウェアによる IP パケット転送オフローディングルータとして動作する．

表 5.1 に FIBNIC の実装環境を示す．FIBNIC は 4 ポートのギガイーサーネットインターフェイスを搭載している NetFPGA-1G[34] ボード上で実装した．

NetFPGA には，Stanford University が設計，開発した HDL(Hardware Description Language) モジュールおよびソフトウェアプラットフォームが提供されているが，本手法の目的に合致しないため新規に設計した．

5.2.2 概要

図 5.8 に，FIBNIC のモジュールダイアグラムを示す．

FIBNIC は PC 上の PCI デバイスとして構成され，受信パケットのフレームタイプ，プロトコル，宛先によって NIC モジュールと本手法による IPv4 および ARP/ICMP モ

表 5.1 **実装環境**

Dev Kit	NetFPGA-1G
FPGA	Virtex-II Pro 50
Ethernet	1000BASE-T × 4
Peripheral Bus	PCI
SDK	ISE 10.1SP3
Verilog sim	GPL Cver/GTKWave
SLICEs	7671 out of 23,616 (32%)
動作周波数	125MHz(Logic), 33MHz(PCI)

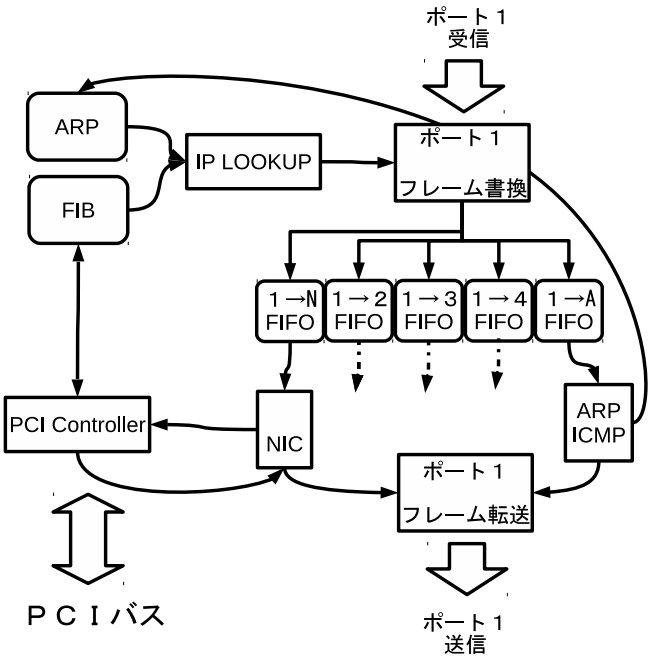


図 5.8 **モジュールダイアグラム**

ジュールに振り分けられる．

NIC モジュールは FIFO とデュアルポートメモリより構成され，PC の NIC ドライバ経由で操作する．対象となるパケットはブロードキャストパケット，NIC 宛アドレスおよび IPv4 モジュールで対象から外れたものである．

このため，ホスト側の PC ルータのソフトウェアや OS，あるいは電源が停止しても，PCI の電源さえ供給されていれば，IP パケット転送の動作を継続できるが，ルーティングプロトコルの処理が行われないため， 経路表の変更を反映することはできない．

5.2.3 IP Lookup 探索

パケットの転送先を調べる IP Lookup はルーティングの中でも時間のかかる処理で，極めて重要な役割を担う．

IP Lookup は FIB にある経路表をもとに，Longest Match 処理を行い転送先を探索する．また ARP テーブルよりローカルセグメント上の MAC アドレスも同時に調べる．

FIB の探索をハードウェアで実装する場合，ルータ専用機では TCAM を用いることが一般的であるが，TCAM は多量のハードウェアリソースを消費するため，ASIC と比べて回路規模が小さい FPGA ではあまり適さない．

一方，IP アドレスを RAM のアドレスにマッピングする手法として，文献 [35] があげられるが，今回実装する NetFPGA の SRAM は 4MB しかないためそのまま適用することはできない．

本手法では図 5.9 に示す方法で実装する．

4MB の SRAM のアドレスバスは 22 ビットになることから，IPv4 アドレスの上位 22 ビット分までを SRAM のアドレスバスに割り当てる．次に 8 ビットのデータを 2 ビット × 4 に分割し，サブネットマスクを 2 ビット拡張し，24 ビットまでのサブネットマスクへ対応させる．1 セグメントあたり 4 値を表現できるので，3 種類の Next Hop 情報を持つことができる．この値をデコードすることにより，Next Hop IP の MAC アドレスとポート番号を取得する．今回は 4 ポートルータのため問題にならないが，Next Hop 数が足りない場合は SRAM の容量を大きくすればよい．8MB にすると，セグメントあたり 4 ビット保持できるので，15 種類までの Next Hop 情報を持てる．

24 ビットの IPv4 サブネットマスクまでしか扱えないが，該当する 41 万件のサブネットが対象となる．

SRAM はデュアルポートの同期式パイプライン方式のため，PC 側から FIB をアクセスしながらでも，125MHz 動作時で 125M 回/秒の参照が可能である．

今回構成の 4 ポートギガビットイーサネットルータでは，64 バイトパケットのワイヤーレート時で約 6MPPS の負荷となり，それぞれのポートに IP Lookup 用のキャッシュ機構を採用する必要はない．

IP Lookup 処理に割り当てるパイプラインステージ数は，余裕を持って 20 とした．図 5.4 より，IP Lookup を開始するまでに 42 ステージが必要なため，合計 62 ステージのパイプラインとなる．

転送遅延の試算は，5.1.1.6 節より，$62 \times 8 + 356 = 852$ ns となる．この数値には，非同期 FIFO，同期 FIFO，データパス遅延などの論理回路の処理遅延が含まれていない．

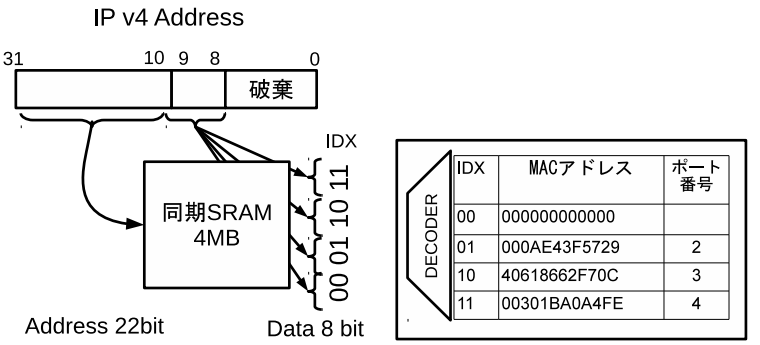


図 5.9 **IP Lookup**

5.2.4 IPv6 対応

IP プロトコルのバージョンによる遅延時間比較を行うために IPv6 版を実装した．基本的には IPv4 と同じだが，以下の点を考慮する必要がある．

(1) パイプラインステージ数
(2) IP Lookup 探索手法
(3) NDP（Neighbor Discovery Protocol）

（1）に関しては，IPv6 ではパケット内の送信先 IP アドレス位置が異なるためにパイプラインステージ数を増やす必要がある．本実装では送信先 IP アドレスの先頭 64 ビットまでを有効とし，パイプラインステージ数を 12 増やした．この値は，図 5.4 中の IPv6 と IPv4 の読み込みサイクル数の差分である 20 から後半の 64 ビットに相当する 8 バイトを減算したものである．

（2）に関しては，送信先 IP アドレスが 128 ビットに拡張されたことによる．本実装では 32 エントリーの TCAM を用いた．

（3）に関しては，IPv4 の ARP に相当するものである．

5.2.4.1 デュアルスタック

本実装では比較のため，IPv4 と IPv6 は個別に実装するが，この節ではデュアルスタック対応について述べる．

IPv4 と IPv6 が混在する場合，それぞれの回路を用意しイーサネットフレームタイプで分岐判断すればよい．このとき IPv4 のパイプラインステージ数を IPv6 のパイプラインステージ数にあわせることにより，パイプラインレジスタ等を共有化でき，回路規模の削減ができる．

表 5.2 **64 バイトフレーム (IPv4)**

対象	受信 PPS	遅延 (A) (ns)	S/F 遅延 (B)	(A)-(B)
本手法	1,488,095	976		
3750E-24PD	1,488,095	4,976	576	4,400
AX3630S-24T	1,488,095	4,072	576	3,946
PC ルータ	365,497	24,144	576	22,568

5.3 性能評価

5.3.1 評価システムの構成と計測結果

ネットワーク機器の性能評価には，ハードウェアによるトラフィック生成と遅延計測が可能な magukara[36] を用いた．magukara は回路とソフトウェアがオープンソースで公開されており，RFC2544[37] をサポートしている．また，同時に 4 ポートまでのギガビットイーサネットをサポートしている．

本手法を用いた評価システムの構成は，FIBNIC を搭載した PC ルータで，CPU に Celeron 2.53GHz，OS には Linux Kernel 3.3.7，PC ルータの構成は 2 ポート NIC に CPU に Core i5 2.8GHz，OS は Linux Kernel 3.3.7 である．

計測対象の 1 ポート目に計測器の送信ポート，計測対象の隣接する 2 ポート目に計測器の受信ポートを接続した．

表 5.2 に 64 バイトフレームを用いた測定結果，表 5.3 に 1518 バイトフレームを用いた測定結果を示す．

本手法以外ではストアアンドフォワード方式が使われているため，参考値としてストアされるパケットの転送時間を (B) として記述した．これにはイーサーネットフレームのプリアンブル部が含んでいる．

次に，フレームサイズを 64/128/256/612/1024/1518 バイトに変えた場合の遅延測定結果を図 5.10 に示す．市販 L3 スイッチはフレームサイズに対して遅延が線形的に増えるが，本手法では遅延が一定であることが確認できた．

次に，送信 1 ポート，受信 3 ポートを用いて，41 万経路時の遅延を計測した．経路表

表 5.3 **1518 バイトフレーム (IPv4)**

対象	受信 PPS	遅延 (A) (ns)	S/F 遅延 (B)	(A)-(B)
本手法	81,274	976		
3750E-24PD	81,274	18,352	12,208	6,144
AX3630S-24T	81,274	15,632	12,208	3,424
PC ルータ	59,355	101,152	12,208	88,944

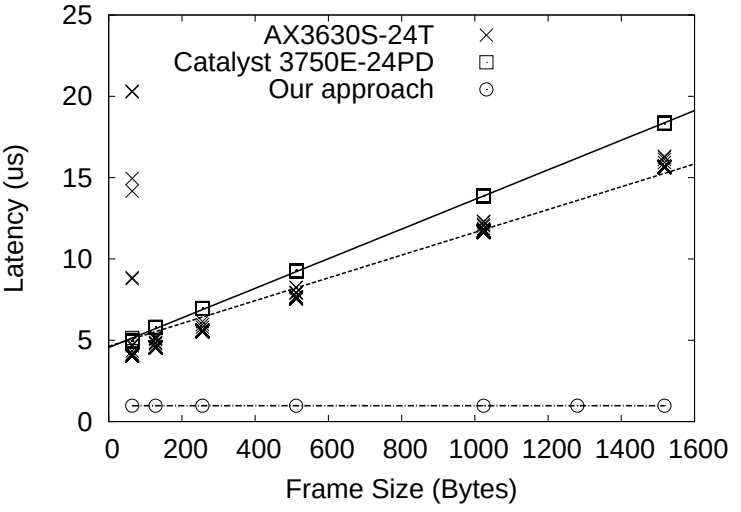


図 5.10 **フレームサイズ毎の遅延**

は IPv4 のフルルート情報 [38] から 24 ビットまでのサブネット 41 万件を取得し，Next Hop IP アドレスには乱数にて 3 ポート分のゲートウェイを登録した．転送先アドレスはホスト部を常に 1 としてサブネット 24 ビットにあたる 0.0.0〜255.255.255 までの範囲を 1 フレーム毎に 1 増加させ，ワイヤーフレームレートにて送信をした．結果を表 5.4 に示す．

表 5.5 は，IPv6 を使用したものである．フレームサイズ 80 バイト，1518 バイトにおける計測結果を示す．64 バイトで計測しなかったのは，IPv6 プロトコルヘッダの仕様によりおさまらなかったためである．

表 5.4 **IPv4 41 万経路 (入力 1 ポート，出力 3 ポート)**

フレームサイズ	遅延 (ns)
64 バイト	976
1518 バイト	976

表 5.5 **IPv6 32 経路 (入力 1 ポート，出力 1 ポート)**

フレームサイズ	遅延 (ns)
80 バイト	1,064
1518 バイト	1,064

本手法は，フレームサイズや参照する経路数に関わらず一定であることが確認できた．

5.4 考察

本手法の計測した遅延時間は，976 ns で，5.2.3 節で試算した遅延時間が 852 ns であったことから，差分の 124 ns が，非同期 FIFO と同期 FIFO 処理やデータパス遅延と考えられる．

FIBNIC はデータセンタ内で低遅延通信を要求とするアプリケーション向けの IP 転送手法であり，市販の L3 スイッチと比べて以下の点が問題である．

(1) ポリシーを使った優先制御などができない
(2) パケットバッファの代わりに小容量の FIFO を使うため，輻輳が起こると短時間にパケットロストが生じる

(1) については，優先制御をするためには一定時間パケットをバッファリングする必要があり，低遅延通信での運用は適していない．

(2) については，輻輳に対応するためには，保持すべき時間分のパケットバッファが必要になり，また，保持した時間分が最大の通信遅延となり遅延時間の増加をまねく．本手法は，FIBNIC の接続範囲がインターネットではなく遅延の小さいデータセンタ内であることを生かし，L3 スイッチでパケットのバッファリングを行わず，IP-NUMA NIC などのエンドノードが，短いタイムアウト時間にてパケットの再送機能を持つことを前提とする．

以上のことを考慮し，FIBNIC による L3 スイッチ接続は低遅延通信アプリケーションが利用する IP-NUMA NIC に限定し，一般のネットワークに接続する NIC には，市販の L3 スイッチを接続することが望ましい．

第 6 章

アプリケーションによる評価

5.1 章では，L3 スイッチの IP 転送遅延の削減，4 章では，PC 側の通信遅延の削減について述べた．本章では，これらの手法の有効性を調べるために，分散キャッシュシステムのアプリケーションである memcached に適用した場合の性能について，分析し，見積もりによって評価を行う．

6.1 memcached の概要

WEB 上のアプリケーションは，データベースサーバにデータを格納し，必要に応じてデータを要求しブラウザ上に表示する．SQL データベースサーバでは問い合わせに SQL 文などを使った高度な問い合わせが可能であるが，処理時間が長い．このため，大量のアクセスが集中するシステムではレスポンスの低下をまねく．分散型メモリキャッシュシステムは，データベースサーバへの要求と応答をインメモリデータベースに一時的にキャッシュすることで，データベースへのアクセス回数を削減し，動的なリアルタイムアプリケーションの高速化やスケーラビリティに対応する．図 6.1 に代表的な分散型メモリキャッシュシステムを用いたウェブアプリケーションの構成例を示す．

アプリケーションからの 1 回目のデータ読み込み要求時におけるデータの流れを灰色の矢印で示す．アプリケーションはデータベースサーバよりデータを入手し，ブラウザに表示する．同時にキャッシュサーバへキーと取得したデータを有効期限とともに書き込む．

2 回目以降のデータ読み込みは，キャッシュサーバにキーが存在するので，メモリからデータを読み込むことで処理を完了する．データベースサーバへの問い合わせと比べて負荷が低く，処理による遅延時間を小さくできる．このときのデータの流れを白矢印で示す．

memcached はクライアントライブラリによるサーバの分散処理に対応しているが，consistent hash に対応しておらず，サーバ障害に対して直接影響を受ける．このため，

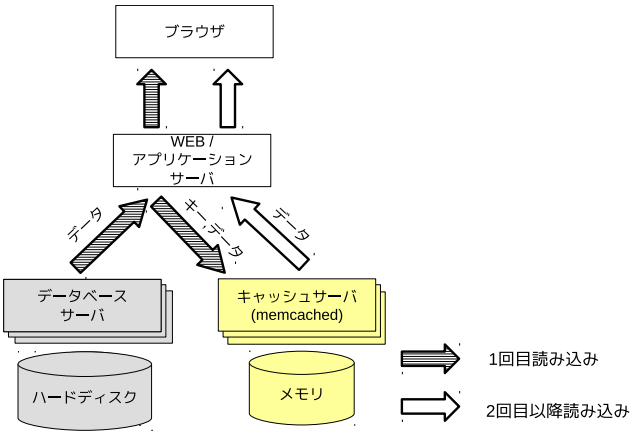


図 6.1 **アプリケーションとキャッシューサーバの構成例**

twemproxy[39] などの consistent hash に対応する Proxy サーバを用いることがある．

6.2 memcached の機能とプロトコル

本節では，代表的な分散型メモリキャッシュシステムである memcached について述べる．memcached はキーとそのデータから成り立つ極めてシンプルなデータベースである．メインメモリ上にデータを保存するインメモリ型のデータベースのため高速であるが，揮発性のため，PC をシャットダウンするとデータが消える．このため，主にキャッシュデータの保存用として用いられる．

表 6.1 に，memcached の主要コマンドと文法 [40] を示す．文法中のパラメータである ¡key¿はキーをあらわす．¡flags¿は 32 ビットの符号なし整数が指定でき，データの取得時に参照される．¡exptime¿はデータが削除されるまでの有効時間を秒数で指定し，0 を指定すると有効時間を設けないが，メモリの空き領域がなくなると削除されることがある．¡bytes¿はデータバイト長，¡data¿はデータを示す．

データの書き込みは Storage 系コマンドである set,add,repalce,append,prepend を用い，set コマンドはデータの保存，add コマンドは key が存在しない場合でのデータ保存，replace コマンドは key が既に存在する場合のデータ保存，append は key が既に存在するデータに対しての後方文字列追加，prepend は key が既に存在するデータに対しての前方文字列追加を行う．

データの取得には get コマンド，データの削除は delete コマンド，既存のデータの値を任意に加算するには incr コマンド，減算するには decr コマンド，データの有効時間を更新するには touch コマンドを用いる．

図 6.2 にクライアントが set コマンドを使って，memcached サーバにデータ保存を行うときの送受信パケットフローを示す．

表 6.1 **memcached の主要コマンド**

種別	コマンド	機能	文法
Storage	set	データの保存	< コマンド > <key> <flags> <exptime> <bytes>\r\n <data> \r\n
Storage	add	データの保存 key が存在しない	
Storage	replace	データの保存 key が存在する	
Storage	append	値への後方追加	
Storage	prepend	値への前方追加	
Retrieval	get	データの取得	get <key> \r\n
Deletion	delete	データの削除	delete <key> \r\n
Increment	incr	値への加算	incr <key> <value> \r\n
Decrement	decr	値への減算	decr <key> <value> \r\n
Touch	touch	有効時間の更新	touch <key> <exptime> \r\n

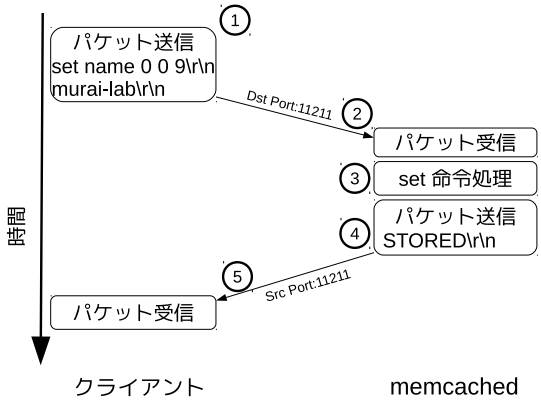


図 6.2 **set コマンド発行時の送受信パケット**

① クライアントが name をキーとして 9 文字長である murai-lab を保存する set コマンドパケットを送信する
② mecached が 1 のパケットを受信する
③ set 命令による保存処理をする
④ 正常に処理が行われたことを示す ”STORED \r\n” をクライアントへ送信する
⑤ クライアントが正常処理されたことを確認する

次に図 6.3 にクライアントが get コマンドを使って，memcached サーバよりデータを取得するときの送受信パケットフローを示す.

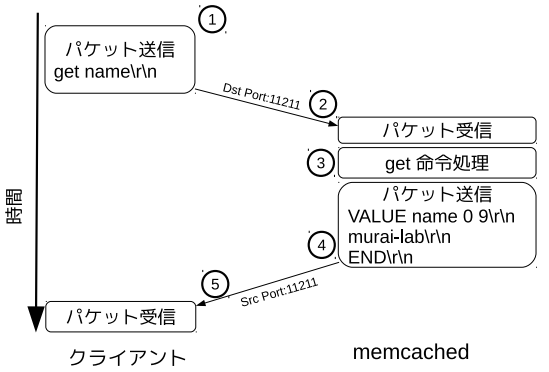


図 6.3 **get コマンド発行時の送受信パケット**

① クライアントが name をキーとして，get コマンドパケットを送信する
② mecached が 1 のパケットを受信する
③ get 命令によるデータの取得を処理する
④ 1 行目に flag とデータ長，2 行目以降にデータが入ったパケットをクライアントへ送信する
⑤ クライアントがデータを受け取れたことを確認する

6.3 データベースキャッシュとしての memcached

本節では，memcached をデータベースキャッシュとして利用する例について述べる．

図 6.4 は，アプリケーションとデータベースとキャッシュである memcached との関係を示したものである．データベースサーバへの問い合わせ処理に対して，キャッシュサーバへの処理は負荷が小さく，処理時間も極めて小さい．このため，アプリケーションからデータベースサーバへのアクセスの頻度は極力減らす必要がある．アプリケーションがデータをアクセスする手法について，例を以下に解説する．

図 6.4 は，アプリケーションがキャッシュミスとなる 1 回目のデータ読み込みを行ったときの流れを示す．1 番目に，キャッシュである memcached に対して，デーベースへの問い合わせ条件等をもとにキー生成し，get コマンドを発行し，キャッシュに存在しないことを確認する．2 番目に，データベースサーバへ問い合わせ処理を行いリザルトを取得する．3 番目に，取得したリザルトをデータとして memcached に対して set コマンドを発行することでキャッシュへ保存し，ブラウザにも結果を表示する．

図 6.5 は，アプリケーションがキャッシュヒットとなる 2 回目以降のデータ読み込みを行ったときの流れを示す．キャッシュである memcached に対して，デーベースへの問い合わせ条件等をもとにキー生成し，get コマンドを発行し，キャッシュに存在するデータ

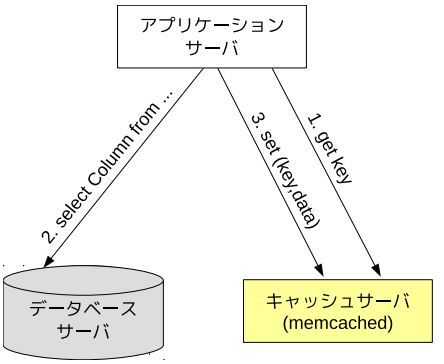


図 6.4 **データ読み込み (キャッシュミス時)**

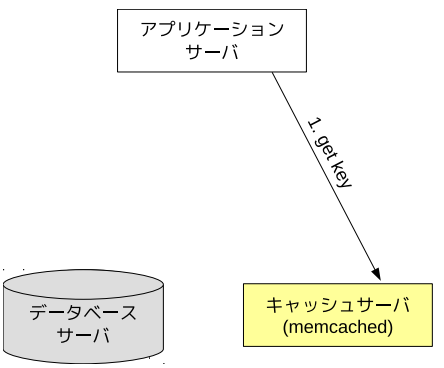


図 6.5 **データ読み込み (キャッシュヒット時)**

を取得する．取得したデータをブラウザに表示する．

図 6.6 は，アプリケーションがデータを更新したときの流れを示す．1 番目に，データベースサーバへ更新処理を行う．2 番目に，キャッシュである memcached に対して，該当するデータに関するキーを生成し，delete コマンドを発行し，キャッシュを削除する．キャッシュが削除されることにより，次にデータ読み込みが行われた場合はキャッシュミスとなり，データベースより更新されたデータが読み込まれ，キャッシュに保存される．

6.4 通信パケットの解析

本節では，アプリケーションより memcached を用いた場合の遅延時間を見積もりするために，送受信に関わるパケット処理の解析を行う．図 6.7 に，評価環境で用いる構成について示す．memcached が動作する PC と，ベンチマークプログラムが動くアプリケーション用 PC の 2 台が，ネットワークケーブルによって直接接続されている．

memcached のベンチマーク用プログラムには，クライアントライブラリである libmemcached[41] と共に提供されている memslap があるが，データ長などいくつか

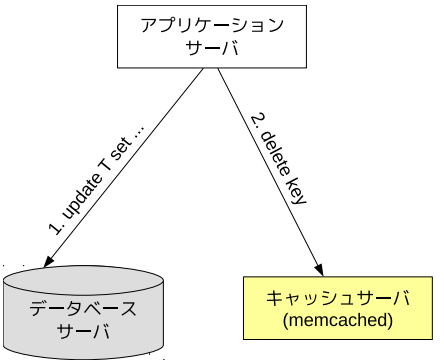


図 6.6 **データ更新**

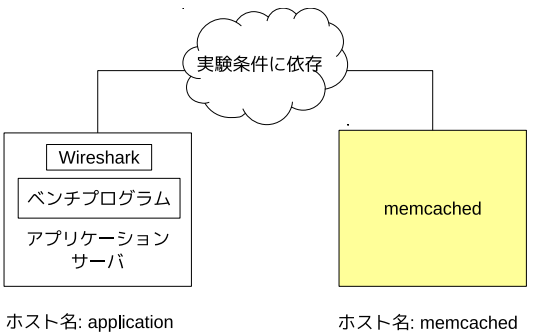


図 6.7 **評価環境の構成**

表 6.2 **PC の仕様**

CPU	Intel Core i7-4770 @ 3.4GHz × 2
Giga NIC	Intel I350T2 (PCIe G2×4) × 2, 1000BASE-T
10G NIC	Intel X520-SR2 (PCIe G2×8) × 2, 10GBASE-SR
OS/Kernel	Fedora 23/4.4.7-300.fc23.x86_64

の条件に関するパラメータを変更できない．今回は，memcached 向けのベンチマーク用プログラムである mcb[42] を用いる．mcb はデータ長などが変更可能で，memcached プロトコルの中から set，get，add の 3 種類のコマンドに対応することができる．

通信パケットの解析には，アプリケーション用 PC 上で tcpdump と wireshark を使う．PC と NIC の仕様詳細を表 6.2 に示す．

解析するパケットは，アプリケーションから SET コマンドを 10,000 回発行することで生成した．アプリケーション用 PC と memcached 用 PC 間の通信が高速でなおかつ低遅延だと，パケットキャプチャ時にパケットロスが発生するため，Giga NIC と L2 スイッチハブを経由した．このときのベンチマークコマンドを図 6.8 に，パケットキャプチャコマンドを図 6.9 に示す．

```
$ mcb -c set -a IP アドレス -p 11211 -t 1 -n 10000 -l 16  -T UDP
```

図 6.8 **ベンチマーク用コマンド**

```
$ tcpdump -w set_l16.pcap port 11211 -i enp3s0f0
```

図 6.9 **パケットキャプチャ用コマンド**

ポート番号 11211 で通信パケットをフィルタしたところ，UDP パケット 20,000 個をキャプチャできた．6.2 節の図 6.2 で示したように，SET コマンドは 1 個の要求と，1 個の応答から成り立つ．このことからアプリケーションからの 10,000 回の set パケット送信に対して，memcached 側から 10,000 回の応答パケットが送信されたと考えられる．

wireshark で解析したパケットの流れを図 6.10 に示す．丸の中の示された数字は，wireshark が受信したパケット番号で奇数番号はアプリケーションから送信した要求パケット，偶数番号は memcached から送信された応答パケットをあらわしている．左側の時間はアプリケーション PC の wireshark 上で取得したパケット送信時または受信時の時間である．以下に，時系列にパケットの流れを解説する．

1. 0 マイクロ秒に 1 回目の set パケット送信
2. 258 マイクロ秒に 1 回目の stored パケットを受信
3. 279 マイクロ秒に 2 回目の set パケットを送信
4. 528 マイクロ秒に 2 回目の stored パケットを受信
5. 1.849183 秒に 10000 回目の set パケットを送信
6. 1.849384 秒に 10000 回目の stored パケットを受信

wireshark 上での最後のパケットの受信は 1.849384 秒であるが，ベンチマークソフト上での実行時間は，1.849612 秒を示していた．これは，パケットキャプチャによる時間取得とベンチマークアプリケーションによる時間取得による誤差と考えられ 0.012% と小さいため，計測ではベンチマークアプリケーションの実行時間を採用する．

6.5 基本性能計測

前節では，memcached を用いるアプリケーションにおいて 1 コマンド要求に対して，PC 間で 1 パケットの往来が必要であることがわかった．本節では，性能見積もりを行うのにあたって，通信遅延が一番小さくなる条件でアプリケーションの基本性能を計測する．PC 間はスイッチを経由せず，ネットワークケーブルで直接接続した．

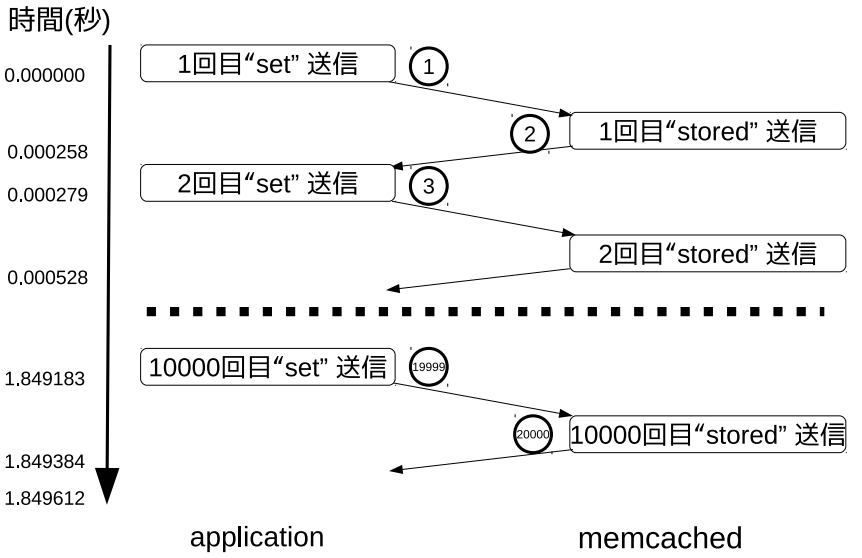


図 6.10 **Wireshark によるパケット解析 (SET コマンド問い合わせ)**

計測の条件は, コマンドに SET,GET,ADD の 3 種類, データ長は 16,32,64,128,256,512 バイトの 6 種類，通信プロトコルは IPv4 UDP で，ネットワークインターフェイスにはギガビット，10 ギガビットと参考データとしてループバック接続の 3 種類．ループバック接続は memcached 上でアプリケーションを実行し，接続先 IP アドレスを 127.0.0.1 に設定した.

計測は前節と同じく 10,000 回単位でコマンドを繰り返し実行したものを，さらに 10 回繰り返したものの中から 最小時間の値を採用した.

表 6.3 は，計測結果からコマンド 1 回あたりの実行時間に換算したものである．SET 命令において，データ長 16 バイトと 512 バイトのときの実行時間を比較すると，ギガビットインターフェイスでは 1.23 倍，10 ギガビットインターフェイスでは 1.16 倍，ループバックインターフェイスでは 1.03 倍となり，インターフェイス速度の制約がないループバックインターフェイスでは，データ長による実行時間の影響がそれほど大きくない．それに対して，ネットワークインターフェイスでは速度が遅くなるに従ってデータ長による実行時間の影響を受ける割合が多くなることが確認できた.

SET,GET,ADD の命令の違いによる実行性能差はそれほど多くないことがわかった．以後，特に指定がない限り，利用使用頻度が高い GET 命令の数値を用いることにする.

6.6 性能見積

本節では，前節で得られた memcached コマンドの実行時間計測値と 5.3.1 節で計測した L3 スイッチ転送遅延と 4.2.3 節で計測した PC 側通信遅延より，memcached にそれ

表 6.3 **memcached コマンドの実行時間計測 (単位 ns)**

接続手法と速度 (ケーブル)	命令	データ長 (バイト)					
		16	32	64	128	256	512
バークレイソケット ギガビット (1000BASE-T 1m)	SET	51,251	51,376	51,516	51,990	52,596	63,290
	GET	50,546	50,567	50,679	51,092	51,777	62,786
	ADD	51,296	51,233	51,382	51,844	52,435	63,159
バークレイソケット 10 ギガビット (10GBASE-SR 1m)	SET	30,156	30,250	30,984	31,504	31,949	34,937
	GET	29,200	29,521	29,852	30,037	30,622	34,452
	ADD	30,282	30,448	30,739	31,586	31,967	35,102
ループバック (ローカル実行)	SET	10,817	10,957	10,956	10,895	11,003	11,163
	GET	9,405	9,418	9,454	9,443	9,434	9,671
	ADD	10,720	10,526	10,617	10,693	10,805	11,099

表 6.4 **適用手法と速度別の GET コマンドの実行時間 (単位 ns)**

適用手法と速度 (ケーブル)	区分	データ長 (バイト)					
		16	32	64	128	256	512
1G ソケット	実測	50,546	50,567	50,679	51,092	51,777	62,786
1G IP-NUMA	見積	11,475	11,496	11,608	12,021	12,706	23,715
10G ソケット	実測	29,200	29,521	29,852	30,037	30,622	34,452
ローカル	実測	9,405	9,418	9,454	9,443	9,434	9,671

ぞれの手法を適用した場合の性能について見積もりを行う．

表 6.4 は，PC 側通信手法にバークレイソケットと IP-NUMA を用いた場合の memcached 実行時間の実測と見積もり値で，実測値は表 6.3 の GET 命令欄より引用している．

表 4.5 より，UDP プロトコルを用いた PC 間の 1 往復通信にかかる時間は，ギガビットイーサネットにおいてバークレイソケットでは 49,379 ns，IP-NUMA は 10,308 ns だった．このことは，IP-NUMA を利用した場合，バークレイソケットを使用した場合と比べて，39,071 ns 削減できることを示している．

実測値からそれぞれに対応する上記遅延分を差し引いたものを見積もりとして ”1G IP-NUMA” に記載した．10 ギガビットイーサネットに関しては，表 4.5 の RTT 計測をしたときの Linux カーネルバージョンが memcached 計測時のものと異なり参考にならないと考えられるため見積もりしない．

次に，L3 スイッチに関する通信遅延について検討する．L3 スイッチを介したネットワーク構成はラック内での接続とラック間での接続の構成を想定した．

ラック内での構成を図 6.11 に示す．PC 間は 1 台の L3 スイッチと合計 20m のケーブルで接続されている．

ラック間での構成を図 6.12 に示す．PC 間は 4 台の L3 スイッチと合計 200m のケーブルで接続されている．

このときに生じる L3 スイッチの転送遅延とケーブルの伝搬遅延を含む往復通信遅延を

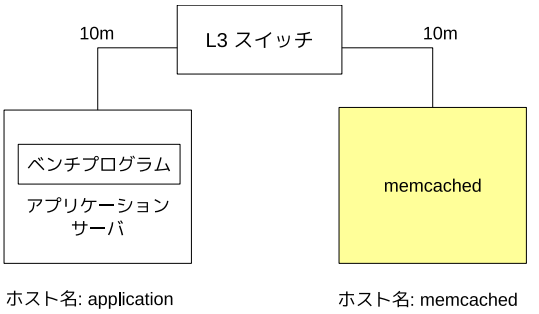


図 6.11　**ラック内接続の想定構成**

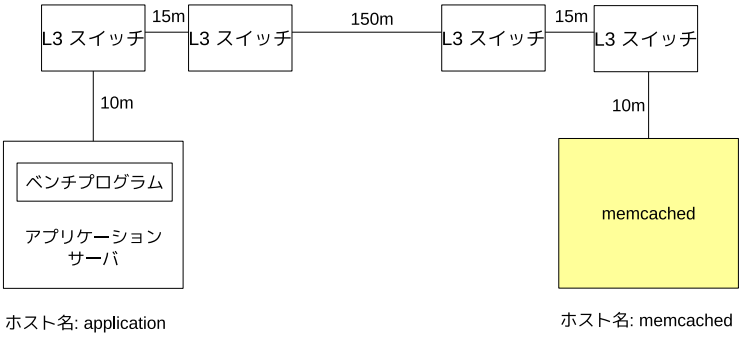


図 6.12　**ラック間接続の想定構成**

表 6.5　**往復遅延時間**

接続種別	接続形態	伝搬遅延 (ns)	スイッチ遅延 (ns)	遅延合計 (ns)
ラック内	AX3630S-24T×1+ ケーブル長 20m	200	8,144	8,344
	FIBNIC×1+ ケーブル長 20m	200	1,952	2,152
ラック間	AX3630S-24T×4+ ケーブル長 200m	2,000	32,576	34,576
	FIBNIC×4+ ケーブル長 200m	2,000	7,808	9,808

表 6.5 に示す．

市販スイッチである AX3630S-24T の転送遅延は，5.3.1 節の表 5.2 より，64 バイトパケットの転送遅延は 4,072 ns，本手法である FIBNIC は 976 ns である．これを往復遅延に換算すると，それぞれ 8,144 ns と 1,952 ns となる．1518 バイトパケットの転送遅延は，表 5.3 より，AX3630S-24T は 16,632 ns と 64 バイトサイズと比べて遅延時間が大きくなるが，本手法では 976 ns とパケットサイズに影響を受けることなく遅延時間が一定である．AX3630S-24T では，パケットサイズによって遅延時間が変化するが，本見積もりでは，最小値である 64 バイトパケットでの遅延時間を適用する．実際のパケット長は 64 バイトを超えるため，AX3630S-24T による遅延時間は実際より少なく見積もることになり，本手法である FIBNIC の性能向上を実際より少なく見積もっている．

表 6.6 **ラック内・間接続時の GET コマンド実行時間 (単位 ns)**

接続種別	区分	PC 側通信	L3 スイッチ	データ長 (バイト)					
				16	32	64	128	256	512
ローカル	実測	ソケット	−	9,405	9,418	9,454	9,443	9,434	9,671
直接	実測	ソケット	−	50,546	50,567	50,679	51,092	51,777	62,786
	見積	IP-NUMA	−	11,475	11,496	11,608	12,021	12,706	23,715
ラック内	見積	ソケット	AX3630S-24T	58,890	58,911	59,023	59,436	60,121	71,130
		ソケット	FIBNIC	52,698	52,719	52,831	53,244	53,929	64,938
		IP-NUMA	AX3630S-24T	19,819	19,840	19,952	20,365	21,050	32,059
		IP-NUMA	FIBNIC	13,627	13,648	13,760	14,173	14,858	25,867
ラック間	見積	ソケット	AX3630S-24T	85,122	85,143	85,255	85,668	86,353	97,362
		ソケット	FIBNIC	60,354	60,375	60,487	60,900	61,585	72,594
		IP-NUMA	AX3630S-24T	46,051	46,072	46,184	46,597	47,282	58,291
		IP-NUMA	FIBNIC	21,283	21,304	21,416	21,829	22,514	33,523

ラック内，ラック間の接続において，本手法を組み合わせた場合の実行時間 (単位 ns) の比較を表 6.6 に示す．PC 側通信には，バークレイソケットまたは本手法である IP-NUMA，L3 スイッチには，市販スイッチである AX3630S-24 または本手法である FIBNIC を用いる．

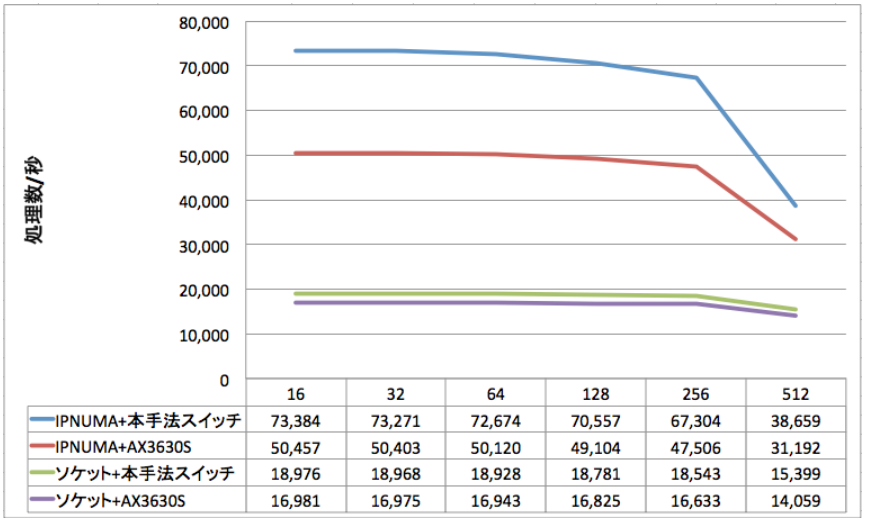
6.6.1 ラック内接続

図 6.13 は，表 6.6 のラック内に関する部分を抽出し，コマンド実行時間を 1 秒あたりの処理数に換算したものである．L3 スイッチを AX3630S から本手法の FIBNIC に変えてもそれほど，処理性能があがっていない．これは，L3 スイッチによる遅延が全体の遅延に対して多くを占めていないためと考えられる．バークレイソケットを IP-NUMA に変えた場合は，処理性能が大きくあがっている．IP-NUMA NIC を使うことにより，遅延が大きく削減できたためと考えられる．

データ長 16 バイトについて，処理時間の内訳明細を記したものを図 6.14 に示す．

通信網遅延は，L3 スイッチ遅延，PC 側通信遅延は，バークレイソケットまたは IP-NUMA による遅延，アプリケーション遅延は，表 6.3 よりギガビットの GET コマンド実行時間である 50.546 μs と，表 4.5 のギガビットの RTT である 49.379 μs との差分である 1.167 μs から算出した．表 4.5 は，単純な UDP 受信と UDP 送信などの通信処理以外に，1 個の変数の 1 加算処理時間が含まれているが，計測に影響を与えるほどでないごくわずかな時間と考えられるので無視をしている．”ソケット +AX3630S”の遅延の詳細では，PC 側通信遅延が多くを占めるのに対して，通信網遅延は全体の 2 割にも満たない．これは，先ほど述べたこととも合致する．

表 6.7 は，PC 側通信にバークレイソケット，L3 スイッチに AX3630S-24T を使用したときを 100% として，条件を変えた場合の処理能力をあらわしている．データ長が 16 バイトだと，432% の処理能力となったが，データ長が 512 バイトだと，275% の処理能



	16	32	64	128	256	512
IPNUMA+本手法スイッチ	73,384	73,271	72,674	70,557	67,304	38,659
IPNUMA+AX3630S	50,457	50,403	50,120	49,104	47,506	31,192
ソケット+本手法スイッチ	18,976	18,968	18,928	18,781	18,543	15,399
ソケット+AX3630S	16,981	16,975	16,943	16,825	16,633	14,059

図 6.13　**ラック内接続における処理能力の比較**

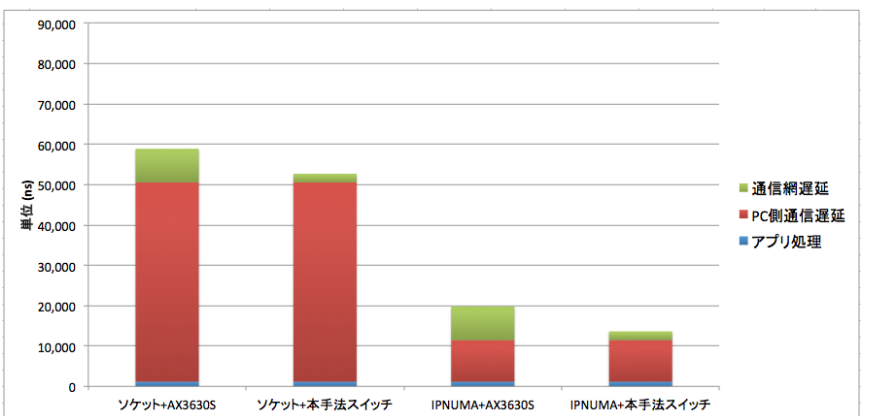


図 6.14　**ラック内接続における処理時間内訳 (データ長 16 バイト)**

力になるのは，データ転送遅延が多くを占めるからである．表 6.3 のデータ長 16 と 512 バイトの計測結果で示しているように，イーサネットのインターフェイスがギガビットから 10 ギガビットに高速化すると，データ長の大きさによる遅延時間の差は縮まっている．

6.6.2　ラック間接続

図 6.15 は，表 6.6 のラック間に関する部分を抽出し，コマンド実行時間を 1 秒あたりの処理数に換算したものである．L3 スイッチを AX3630S から本手法の FIBNIC に変えた場合も，バークレイソケットを IP-NUMA に変えた場合もいずれも処理性能があがっている．

表 6.7　**ラック内接続における処理能力の向上**

PC 側通信	L3 スイッチ	データ長 (バイト)					
		16	32	64	128	256	512
ソケット	AX3630S-24T	100%	100%	100%	100%	100%	100%
ソケット	FIBNIC	112%	112%	112%	112%	111%	110%
IP-NUMA	AX3630S-24T	297%	297%	296%	292%	286%	222%
IP-NUMA	FIBNIC	432%	432%	429%	419%	405%	275%

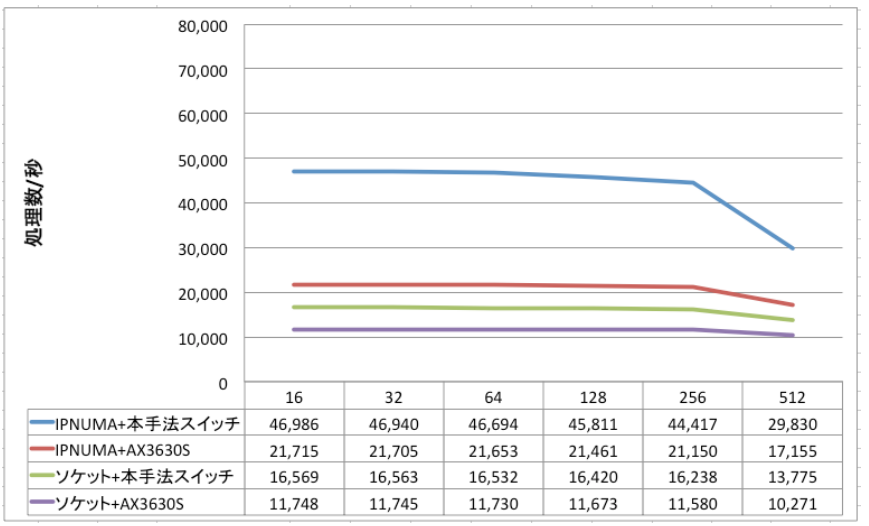


図 6.15　**ラック間接続における処理能力の比較**

表 6.8　**ラック間接続における処理能力の向上**

PC 側通信	L3 スイッチ	データ長 (バイト)					
		16	32	64	128	256	512
ソケット	AX3630S-24T	100%	100%	100%	100%	100%	100%
ソケット	FIBNIC	141%	141%	141%	141%	140%	134%
IP-NUMA	AX3630S-24T	185%	185%	185%	184%	183%	167%
IP-NUMA	FIBNIC	400%	400%	398%	392%	384%	290%

データ長 16 バイトについて，処理時間の内訳明細を記したものを図 6.16 に示す．

”ソケット +AX3630S”の遅延の詳細では，PC 側通信遅延および，通信網遅延のいずれも全体の半分近くを占めている．これは，先ほど述べたこととも合致する．

表 6.8 は，PC 側通信にバークレイソケット，L3 スイッチに AX3630S-24T を使用したときを 100% として，条件を変えた場合の処理能力をあらわしている．ラック内接続の場合と比べると，L3 スイッチによる IP 転送遅延が増える分，処理能力の向上が全体的に少し削減している．

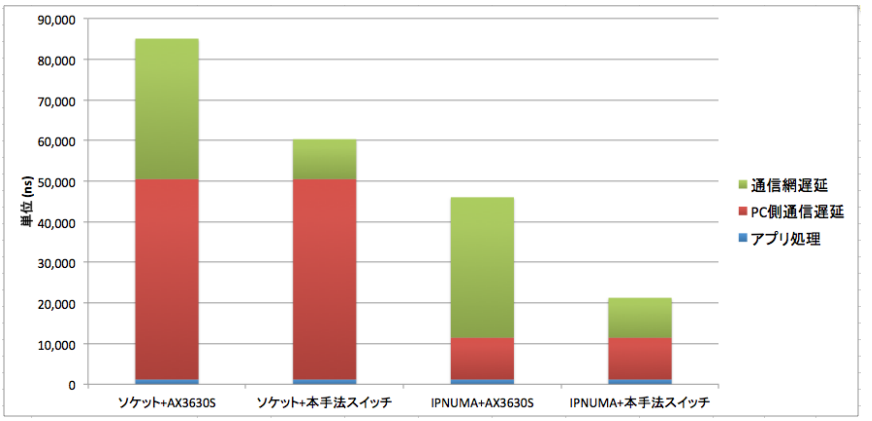


図 6.16 **ラック間接続における処理時間内訳 (データ長 16 バイト)**

6.7 考察

ラック内接続など，途中の経路にあるスイッチの段数が少ない場合は，L3 スイッチの低遅延化による性能向上は微増であるが，スイッチの段数に関わらず，PC 側の通信遅延の削減による性能向上は有効であることがわかった．

前節で用いたアプリケーションの通信処理を除く，サーバ上でのアプリケーション実行時間は，1.167 μs であった．本節では，アプリケーション処理時間と処理性能の関係についてあきらかにするため，試算を行う．

6.7.1 本手法が有効なアプリケーション

4.2.3 節より，10 ギガビットイーサネットにおいて，バークレイソケットに対して IP-NUMA を使った PC 側通信遅延は，22.580 μs から 2.162 μs に削減できた．アプリケーションの処理時間を x にしたとき，処理性能比との関係を式であらわすと，

(x + 22.580) ÷ (x + 2.162) = 処理性能比 になる．

また，ギガビットイーサネットでは PC 側通信遅延が 49.379 μs から 10.308 μs に削減されたので，

(x + 49.379) ÷ (x + 10.308) = 処理性能比 となる．

次に，ギガビットイーサネットにてラック内接続した場合について試算したものを，図 6.17 に示す．横軸がアプリケーションの処理時間，縦軸が処理性能比で，バークレイソケットと AX3630S-24T の組み合わせを 1.0 としている．

IP-NUMA と FIBNIC を用いた組み合わせでは，アプリケーション時間が 80μs であれば，1.5 に近い性能比を示し，500μs であれば 1.09 程度の性能比が見込める．

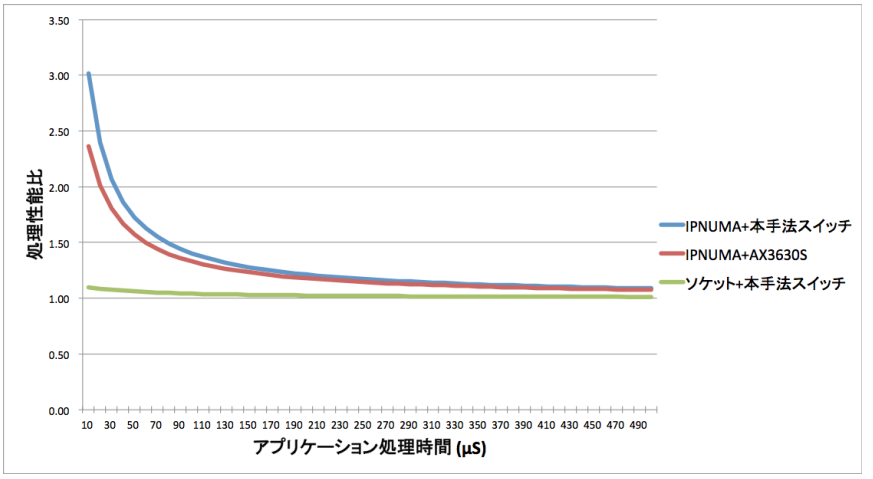


図 6.17　**アプリケーション処理時間と処理性能比（ラック内接続）**

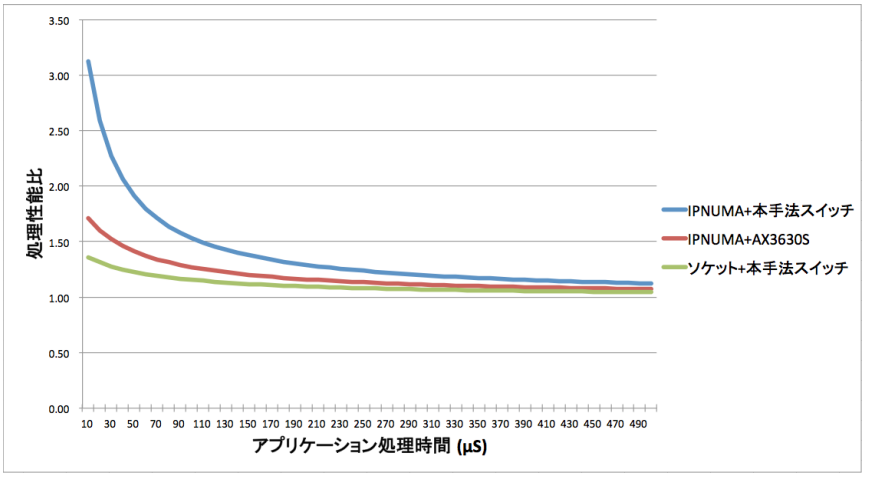


図 6.18　**アプリケーション処理時間と処理性能比（ラック間接続）**

同様にラック間接続した場合について試算したものを，図 6.18 に示す．

IP-NUMA と FIBNIC の組み合わせでは，アプリケーション時間が 110μs であれば，1.5 に近い性能比を示し，500μs であれば 1.12 程度の性能比が見込める．

バークレイソケットを使った場合でも，L3 スイッチを FIBNIC に変えることで，処理性能向上の効果があることがわかった．

以上のことから，実行時間が数百 μs 程度のアプリケーションであれば，性能向上に有効であるといえる．

第 7 章

結論

7.1 本論文のまとめ

クラウドサービスの普及により，アプリケーションの処理はデータセンタ内の膨大なサーバに集中するようになり，サーバ間を接続するネットワークではスループットだけでなく，低遅延通信が求められている．このために，既存技術では RDMA NIC が使われている．

本研究では，RDMA よりも低遅延通信が可能な IP-NUMA 方式を提案し，市販 FPGA 上に IP-NUMA NIC を実装し評価した．

RDMA NIC の最小通信遅延を見積もるため，NIC 内のユーザ回路処理時間を除いた，RDMA に必要な，PCI Express コマンドと PHY の処理遅延を FPGA 上の計測回路で調査したところ，片道通信遅延に 1.213 μs 以上必要であることがわかった．

RDMA ではデータ送信時に行われる DMA リードに機構上オーバーヘッドが生じるため，DMA リードを用いず，NUMA による分散共有メモリを使った CPU ライトによるデータ転送が可能な IP-NUMA モデルを提案した．

市販の 10 ギガビット FPGA ボード上で評価したところ，ping-pong による 2PC 間の通信において，バークレイソケットと比べて時間あたりのループが 10.44 倍，RTT で 90% 削減し，2.16 μs だった．

片道通信遅延では，1.08 μs となり，RDMA と比べて 11% 以上の通信遅延を削減することができた．

次に，ルータのパケット転送時における最低遅延をあきらかにするために，ギガビット PHY の送受信遅延の計測をしたところ，356 ns だった．IPv4 のパケット転送に必要な，送信先 IP アドレスの取得には，ギガビットの GMII において 42 サイクルかかり，ユーザ回路処理遅延とキューイング遅延を除いた，全体の転送遅延は，692 ns であることがわかった．

そして，本研究では，パケットバッファの代わりに，小容量 FIFO をフルメッシュ上に接続しパイプライン化することによって，転送遅延を削減する手法である FIBNIC を提案した．

市販の 4 ポートギガビット FPGA ボード上で評価したところ，フレームサイズに関わらず転送遅延は 976 ns だった．市販の L3 スイッチとの比較では 64 バイトフレームにおいて 76% の遅延時間の削減，1518 バイトフレームでは 93% の削減にあたる．

特にスループットを優先とするトラフィックは，フレーム長が大きく転送遅延の大幅な低下が期待できる．

本手法である IP-NUMA と FIBNIC の有効性を調べるために，データセンタ内で memcached を利用した場合を想定し，FIBNIC と IP-NUMA それぞれの手法を適用した場合の性能見積もりをした．

まず，ラック内での接続を想定した L3 スイッチを 1 台経由する 20m 距離の IP ネットワークにおいて，市販 L3 スイッチに対して FIBNIC を適用した場合は，1.12 倍のトランザクション性能，PC 側通信手法にバークレイソケットに対して IP-NUMA を適用した場合は，2.97 倍のトランザクション性能，FIBNIC と IP-NUMA の両方を適用した場合は，4.32 倍のトランザクション性能となった．ラック内接続では IP-NUMA による性能向上は有効といえるが，FIBNIC による性能向上はあまり有効といえない．

次に，ラック間での接続を想定した L3 スイッチを 4 台経由する 200m 距離の IP ネットワークにおいて，市販 L3 スイッチに対して FIBNIC を適用した場合は，1.41 倍のトランザクション性能，PC 側通信手法にバークレイソケットに対して IP-NUMA を適用した場合は，1.85 倍のトランザクション性能，FIBNIC と IP-NUMA の両方を適用した場合は，4.0 倍のトランザクション性能となった．このことから，ラック間接続では FIBNIC または IP-NUMA のいずれも性能向上に有効だった．

7.2 結論

本研究では，コモディティネットワークである IP プロトコルを使って，キャッシュサーバなど低遅延通信を要件とするアプリケーションの性能向上するために，通信遅延を削減する PC 側通信手法である IP-NUMA と L3 スイッチ向け IP 転送手法である FIBNIC を提案し，その有効性を検証した．

その結果，一般の NIC や RDMA 対応 NIC がパケット送信時に行っている，DMA リードを廃止し，CPU ライトを用いることが通信遅延の削減に有効であることがあきらかになった．その他の遅延要因としては，プロトコルスタックなどのソフトウェア処理時間があげられるが，プロトコルスタックをハードウェア上で処理することが有効であった．

キャッシュサーバを用いたアプリケーションの評価では，PC 側の通信遅延削減がサーバの性能向上に大きく影響することがわかった．

また，ネットワーク網における遅延が大きい（スイッチの段数が多い）場合は低遅延

スイッチの導入だけでもキャッシュサーバの性能向上に効果があることがあきらかになった．

パケットバッファを使わない FIBNIC による IP 転送は，IP-NUMA を使った低遅延通信が必要なアプリケーションには有効だが，そうでないアプリケーションでは，パケットの再送が多くなり非効率になると考えられる．FIBNIC は低遅延アプリケーションが使う IP-NUMA NIC のみの接続に限定し，通常の NIC の接続には，市販のスイッチを使うことが適切である．

本手法により，RDMA 対応 NIC より，低遅延通信が可能になる．クラウドや HPC などのインターコネクトとして IP プロトコルの適用範囲が広がることが期待できる．

7.3 本研究が社会に与える影響

本研究は以下のような影響を社会に与えると考えられる．

1. 現状の低遅延通信サービスで用いられている RDMA で提供されているサービス（HPC のインターコネクトや iSCSI など）をより高速化することができる
2. データセンタ内におけるインメモリやフラッシュメモリベースのデータベースサービスの性能向上が見込まれる
3. IP プロトコルスタックのハードウェア化によって，100 ギガを超える高速イーサネットの仕様上のスループットと低遅延性能をアプリケーション上から活用できる

7.4 今後の課題

7.4.1 IP-NUMA を用いた受信時のポーリング

本手法は，データ送信に DMA リードを用いないため，データ送信時に CPU が解放されない．そのため，PCI Express バスのリンク速度をできるだけあげハードウェア内でバッファリングし CPU を早めに解放する手段である．また，データ受信時にメモリに対するポーリングが発生するため，CPU 負荷を軽減するために，割り込み処理が必要になる．

しかし，割り込みに依存せず，ポーリングを主体とする手法は Intel DPDK や netmap などでも使われており，高スループット，低遅延通信に欠かせない技術になっている．

7.4.2 バークレイソケット ラッパの対応

本手法では，アプリケーション間通信にノード間の共有メモリ空間を用いたメモリアクセスを用いるため，バークレイソケットアプリケーションから利用することはできない．

これらのアプリケーションに対応する場合は，バークレイソケット API を IP-NUMA によるメモリアクセスへ変換するラッパの開発が求められる．

7.4.3 L3 スイッチのポート数と FIFO

本手法は，ポート数の二乗にあたる数の小容量 FIFO が必要になるため，4〜16 ポート程度の利用を想定している．それを大幅に超えるポート数が必要な場合，多段化接続が必要になる．L3 ネットワークで接続されたデータセンタの遅延時間の多くを占めるものは，途中を経由するスイッチである．

謝辞

学部入学以来長きにわたりご指導いただき，またご多忙の中，本研究にも多くの御助言と励ましをいただきました，慶應義塾大学環境情報学部 村井純教授に心より感謝いたします．また，修士課程より本研究の基となるコンピュータアーキテクチャに関して，アドバイスをいただいた，慶應義塾大学環境情報学部 ロドニーバンミーター准教授に深く感謝いたします．本論文の指導をはじめとする，様々なプロジェクトに関して研究サポートをいただいた，慶應義塾大学環境情報学部 中村修教授に深く感謝いたします．ご多忙の中，技術面・ストーリ面において論文のご指導をいただいた，慶應義塾大学理工学部 天野英晴教授に深く感謝いたします．

気の遠くなる膨大な条件付再録の論文やその後について，適切な指導をしていただいた，慶應義塾大学環境情報学部 三次仁教授に深く感謝いたします．

日頃より研究活動等の様々なことについてご指導をいただきました，慶應義塾大学環境情報学部 楠本博之教授，慶應義塾大学環境情報学部 植原啓介准教授をはじめとする maui-staff の皆様に感謝いたします．皆様の暖かな支えのおかげで，本研究を無事に纏め上げることができました．

学部時代から多くのプロジェクトへ共に参加し SFC で最も長い時間を共に過ごした慶應義塾大学 空閑洋平博士に感謝いたします．本研究でも共に実現した多くのプロジェクトの成果が生かされております．

公聴会の発表について細かくアドバイスしてくださいました，慶應義塾大学 佐藤雅明博士に深く感謝いたします．論文の技術面についてアドバイスしてくださいました，慶應義塾大学理工学部 松谷宏紀専任講師に深く感謝いたします．いろいろな事柄をこなし，いつも協力してくれる，慶應義塾大学理工学部 徳差雄太氏に深く感謝いたします．

毎月，高速 PC ルータ研究会にて様々な意見やアイディアを交換させていただいた定例メンバーの NTT コミュニケーションズ小原泰弘博士，東京大学 浅井大史博士，国立天文台 大江将史博士に深く感謝いたします．

博士取得について心配していただき，また論文執筆についても様々な協力をしていただいた，株式会社 IIJ イノベーションインスティテュート 田崎創博士に深く感謝いたします．

慶應義塾大学村井研究室の諸氏とは，長い SFC 生活を共に過ごし，本研究の実証実験へのご支援やアドバイスを多数いただきました．刺激的な時間を共に過ごせたことは大

事な財産です．ここに改めて感謝いたします．また長期にわたる研究活動を様々な面で支えていただいた，村井研究室の秘書の皆様に感謝いたします．他にも本論文執筆にあたり，大勢の皆様から励ましの御言葉をいただきましたことに深く感謝いたします．

Xilinx 社のユニバーシティプログラム取得についてご協力いただいた東京エレクトロンデバイスの河端麻紀子氏に深く感謝いたします．常に最新の FPGA 開発ツールが使える環境だったことは研究にとって大変なプラスでした．

学部時代より長きにわたり英語のご指導をいただいた田辺節子氏に深く感謝いたします．

最後に長期にわたる研究生活を忍耐強く支えてくれた，父 松谷卓ニ，母 松谷純子に感謝の意を捧げます．

参考文献

[1] Ieee standards for local area networks: Token ring access method and physical layer specifications. *IEEE Std 802.5-1989*, 1989.

[2] Ieee standards for local area networks: Carrier sense multiple access with collision detection (csma/cd) access method and physical layer specifications. *ANSI/IEEE Std 802.3-1985*, 1985.

[3] Ethernet Alliance. The 2016 Ethernet Roadmap. `http://www.ethernetalliance.org/roadmap/`. (Accessed 29th January 2015).

[4] W. Richard Stevens. *UNIX Network Programming*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1990.

[5] Jesse Brandeburg. Reducing Network Latency. Linux Plumbers Conference, pp. 11–11, San diego, CA, USA, 2012.

[6] Qiumin Xu, Huzefa Siyamwala, Mrinmoy Ghosh, Tameesh Suri, Manu Awasthi, Zvika Guz, Anahita Shayesteh, and Vijay Balakrishnan. Performance analysis of nvme ssds and their implication on real world databases. In *Proceedings of the 8th ACM International Systems and Storage Conference*, SYSTOR '15, pp. 6:1–6:11, New York, NY, USA, 2015. ACM.

[7] RDMA Consortium. `http://www.rdmaconsortium.org/`. (Accessed 29th January 2015).

[8] Satish M. Thatte. Persistent memory: A storage architecture for object-oriented database systems. In *Proceedings on the 1986 International Workshop on Object-oriented Database Systems*, OODS '86, pp. 148–159, Los Alamitos, CA, USA, 1986. IEEE Computer Society Press.

[9] Ajax: A New Approach to Web Applications. `http://adaptivepath.org/ideas/ajax-new-approach-web-applications/`. (Accessed 29th April 2016).

[10] I. Fette and A. Melnikov. The WebSocket Protocol. RFC 6455 (Proposed Standard), December 2011.

[11] Mellanox Technologies. RDMA over Converged Ethernet (RoCE) - An Efficient, Low-cost, Zero Copy Implementation. `http://www.mellanox.com/page/products_dyn?product_family=79&mtag=roce`. (Accessed 29th April 2016).

[12] Intel Corporation. Intel DPDK - Data Plane Development Kit. http://dpdk.org/.

[13] L. Rizzo and M. Landi. netmap: Memory Mapped Access to Network Devices. *SIGCOMM CCR*, Vol. 41, No. 4, pp. 422–423, Aug. 2011.

[14] Anuj Kalia, Michael Kaminsky, and David G. Andersen. Design guidelines for high performance rdma systems. In *2016 USENIX Annual Technical Conference (USENIX ATC 16)*, pp. 437–450, Denver, CO, June 2016. USENIX Association.

[15] RDMA Consortium. iWARP protocol specification. http://www.rdmaconsortium.org/. (Accessed 29th January 2015).

[16] InfiniBand Trade Association. About RoCE. http://www.infinibandta.org/content/pages.php?pg=about_us_RoCE. (Accessed 29th April 2016).

[17] Intel Corporation. Xeon Processor C5500/C3500 Series Non-Transparent Bridge. http://www.intel.co.jp/content/dam/www/public/us/en/documents/white-papers/xeon-c5500-c3500-non-transparent-bridge-paper.pdf, 2010. (Accessed 29th April 2016).

[18] 飛鷹洋一, 樋口淳一, 下西英之, 鈴木順, 柳町成行, 吉川隆士, 岩田淳. B-6-57 イーサネットを用いたシステム仮想化技術 ExpressEther の提案：(1) システム概要とアーキテクチャ (B-6. ネットワークシステム, 一般講演). 電子情報通信学会ソサイエティ大会講演論文集, Vol. 2006, No. 2, p. 57, sep 2006.

[19] William J. Bolosky, Michael L. Scott, Robert P. Fitzgerald, Robert J. Fowler, and Alan L. Cox. NUMA policies and their relation to memory architecture. *SIGARCH Comput. Archit. News*, Vol. 19, No. 2, pp. 212–221, apr 1991.

[20] Intel Corporation. Transmit Descriptor Ring. In *Intel 82599 10 GbE Controller Datasheet*, pp. 348–350, 2014.

[21] Intel Corporation. Write Combining Memory Implementation Guidelines. 1998.

[22] Ixia. IxNetwork: Network Topology Testing and Traffic Analysis. http://prod.ixia.org/products/ixnetwork. (Accessed 29th April 2016).

[23] Lattice Semiconductor Corporation. LatticeECP3 Versa Development Kit, 2013.

[24] Takeshi Matsuya, Yohei Kuga, Hideaki Yoshifuji, Rodney Van Meter, and Jun Murai. IP-NUMA for Low-latency Communication. In *Proceedings of The Ninth International Conference on Future Internet Technologies*, CFI '14, pp. 13:1–13:4, New York, NY, USA, 2014. ACM.

[25] Intel Corporation. In *Intel Data Direct I/O Technology (Intel DDIO): A Primer*, 2012.

[26] M. Eisler. XDR: External Data Representation Standard. RFC 4506 (INTERNET STANDARD), may 2006.

[27] Arista Networks, Inc. Arista 7150 Series. https://www.arista.com/en/products/7150-series. (Accessed 23th May 2015).

[28] Information technology – Generic cabling for customer premises, 2002.

[29] InfiniBand Trade Association. http://www.infinibandta.org/. (Accessed 29th April 2016).

[30] N.J. Boden, D. Cohen, R.E. Felderman, A.E. Kulawik, C.L. Seitz, J.N. Seizovic, and Wen-King Su. Myrinet: a gigabit-per-second local area network. *Micro, IEEE*, Vol. 15, No. 1, pp. 29–36, Feb 1995.

[31] F. Baker. Requirements for IP Version 4 Routers. RFC 1812 (Proposed Standard), jun 1995. Updated by RFC 2644.

[32] A. Rijsinghani. Computation of the Internet Checksum via Incremental Update. RFC 1624 (Informational), may 1994.

[33] 河合栄治, 門林雄基, 山口英. ネットワークプロセッサ技術の研究開発動向. 情報処理学会論文誌コンピューティングシステム（ACS）, Vol. 45, No. 1, pp. 54–65, jan 2004.

[34] J. Naous, G. Gibb, S. Bolouki, and N. McKeown. Netfpga: reusable router architecture for experimental research. In *Proceedings of PRESTO '08*, pp. 1–7, Aug. 2008.

[35] P. Gupta, S. Lin, and N. McKeown. Routing lookups in hardware at memory access speeds. In *INFOCOM '98. Seventeenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, Vol. 3, pp. 1240–1247 vol.3, Mar 1998.

[36] 空閑洋平, 松谷健史. FPGA Network Tester MAGUKARA. https://github.com/Murailab-arch/magukara.

[37] S. Bradner and J. McQuaid. Benchmarking Methodology for Network Interconnect Devices. RFC 2544 (Informational), mar 1999. Updated by RFC 6201.

[38] University of Oregon. University of Oregon Route Views Project,. http://www.routeviews.org. (Accessed 29th January 2015).

[39] Manju Rajashekhar and Lin Yang. twemproxy (nutcracker). https://github.com/twitter/twemproxy. (Accessed 29th April 2016).

[40] Danga Interactive. memcached protocol. https://github.com/memcached/memcached/blob/master/doc/protocol.txt. (Accessed 29th April 2016).

[41] Brian Aker. libMemcached. http://libmemcached.org/libMemcached.html. (Accessed 29th April 2016).

[42] Hironobu Suzuki. simple memcached benchmark. https://github.com/s-hironobu/mcb. (Accessed 29th April 2016).

[43] 松谷健史, 空閑洋平, ロドニーバンミーター, 鈴木茂哉, 吉藤英明, 村井純. 全行程のパイプライン化による低遅延 IP パケット転送方式 (スイッチング方式, 特集ヒト・モノ・データをつなげるインターネットアーキテクチャ論文). 電子情報通信学会論文誌. B, 通信, Vol. 96, No. 10, pp. 1095–1103, oct 2013.

[44] 松谷健史, 田崎創, 空閑洋平, 吉藤英明, ロドニーバンミーター, 村井純. FPGA ベース IP-NUMA による低遅延 IP 通信方式 (特集理論・実践に立脚したインターネット

アーキテクチャ論文特集). 電子情報通信学会論文誌. B, 通信, Vol. 98, No. 10, pp. 1115–1126, oct 2015.

[45] オープンルータ・コンペティション 2012,. http://www.interop.jp/2012/orc/.

付録 A

本研究に関する業績リスト

A.1 学会発表，論文，活動

A.1.1 国際学会発表

- CFI2014 (In Proceedings of The Ninth International Conference on Future Internet Technologies 2014) [24]
  “IP-NUMA for low-latency communication“
- APsys2015 Poster (6th ACM SIGOPS Asia-Pacific Workshop on Systems 2015)
  “FPGA based low-latency IPv6 Route Lookup Using Dynamic XOR Table“
- RFID-TA2015 Poster (6th Annual IEEE International Conference on RFID Technology and Applications 2015)
  “Compact Linux Platform using Soft Processor for Software Defined Radio”.

A.1.2 論文誌

- 電子情報通信学会 (査読あり 2013) [43]
  “全行程のパイプライン化による低遅延 パケット転送方式 “
- KEIO SFC JOURNAL (招待論文 2014)
  “ FPGA を使った PC アクセラレーション “
- 電子情報通信学会 (査読あり 2015) [44]
  “FPGA ベース IP-NUMA による低遅延 IP 通信方式”.

A.1.3 国内コンペティション

- オープンルータコンペティション グランプリ受賞 (インターロップ 2012) [45]
  “FIB offloading NIC“.

A.1.4 作品

- Magukara X
  “MagukaraX: FPGA based network tester for 10G ethernet“
- IP-NUMA
  “IP-NUMA: FPGA based NIC for 10G ethernet“