仮想計算機におけるソケットアウトソーシングを用いたIPv4/IPv6 変換の実現

大 橋 宏 樹† 新 城 靖† 齊 藤 剛 †

この論文は, ソケットアウトソーシングという手法を用いて仮想計算機モニタ内で IPv4/IPv6 変換を行う方法を提案している. ソケットアウトソーシングは, ゲスト OS のソケット層の処理をホスト OS に移譲することでネットワーク入出力を高速化する. この論文では, ソケットアウトソーシングを高速化ではなく機能拡張に用いることで, IPv4/IPv6 変換を実現している. この時, IPv4 クライアントを動作させるために, ホスト OS 上で動作し, 仮想計算機モニタと協調して動作する DNS プロクシを用いている. 提案方式は, ホスト OS とゲスト OS ともに Linux で動作している. 10G bps のネットワークにおける実験では, 提案方式が実機上での IPv6 による通信とほぼ同等のスループットが得られた.

IPv4/IPv6 Translation Using Socket-Outsourcing in Hosted Virtual Machines

Hiroki Ohashi,† Yasushi Shinjo† and Go Saito †

This paper proposes an IPv4/IPv6 translation method using socket–outsourcing in a virtual machine monitor(VMM). Since socket–outsourcing delegates the tasks of the guest OS to the host OS at the socket layer, it accelarates network I/O. In this paper, we describes not accelerating network I/O but extending a function of a VMM that realizes IPv4/IPv6 translation using socket–outsourcing. We also implement a DNS proxy that works in the host OS together with the VMM for helping IPv4 clients in a guest OS. Our method is implemented in Linux as the host OS and a guest OS. In a 10 Gigabit network, our method yielded the same throughput as native IPv6 communication.

1. はじめに

近年, IPv4 アドレスの枯渇問題が深刻さを増している. 2011 年 2 月にインターネット全体の IP アドレス割り当てを管理する IANA で在庫ブロックが枯渇し, 2011 年 4 月に JPNIC でも枯渇により新規のアドレス割り当てが停止した. このため, IPv6 の導入に向けての動きが本格化している. 今後, インターネットにおいて IPv6 が普及していき, IPv4 を利用するホストの減少と最終的な利用停止が起こる. 本研究では, ネットワークにおいて IPv4 の運用が停止され IPv6 のみが運用されていることを想定する.

IPv6 のみが運用されているネットワークでも, IPv4 アプリケーションを使い続けなければならないことがある. このような場合, IPv4/IPv6 変換器 (IPv4/IPv6 translator) を使う方法がある[1)~4),9),11),12),16),18),19)]. IPv4/IPv6 変換器とは, IPv4 のアプリケーションと IPv6 のアプリケーションの間の通信を仲介し, 相互の通信を可能にするものである. IPv4/IPv6 変換器には, 様々な種類のものがあり, IP 層で動作するもの, アプリケーション層で動作するもの, および, アプリケーションが呼び出す API (Application Program Interface) をフックして引数を書換えるものがある.

レガシーなアプリケーションを実行するためには, 仮想計算機 (Virtual Machine, VM) を利用することが有用であることが知られている. IPv6 が主になった時には, IPv4 アプリケーションもまたレガシーなアプリケーションであり, VM で実行することが一般的になると思われる. この時, IPv4/IPv6 変換器を動作させる場所として, 次のような場所が考えられる.

- 変換専用 VM
- ゲスト OS
- ホスト OS

変換専用 VM を使う方式においては, ホスト OS は

† 筑波大学大学院 システム情報工学研究科コンピュータサイエンス専攻
Department of Computer Science, University of Tsukuba

IPv6 専用, アプリケーションが動作する VM は IPv4 専用になるという利点がある．しかしながら, 2 つの VM を用いるので通信のオーバヘッドが大きく, 性能が低いという問題がある．ゲスト OS やホスト OS で IPv4/IPv6 変換器を動作させる方式は, 変換専用 VM で動作させる方式よりも性能が高い．しかしながら, 両者ともデュアルスタック構成にして IPv4 と IPv6 が混在した環境を運用する必要がある．IPv4 と IPv6 の混在は, セキュリティや運用上の問題を引き起こす．詳しくは, 2 章で論じる．

そこで本研究では, ソケットアウトソーシングという手法を用いて仮想計算機モニタ内で IPv4/IPv6 変換を行う方法を提案する．アウトソーシングとは, 本来は, VM 上で動作するゲスト OS の処理を実機上のホスト OS に対して委譲することにより処理を高速化する手法である[7),10),21),22)]．ソケットアウトソーシングはアウトソーシングをネットワーク通信に適用した手法である．この論文の貢献は, ソケットアウトソーシングを高速化ではなく, 仮想計算機モニタの機能拡張に用い, IPv4/IPv6 変換を実現した点にある．ソケットアウトソーシングを用いると, ゲスト OS 上のプロセスが発行した IPv4 ソケットへのシステムコールの引数を VMM 内で取得することができる．VMM の内部では, IPv4 ソケットへのシステムコールの引数を IPv6 のものへ書き換え, ホスト OS に対して IPv6 ソケット関連のシステムコールを発行することで IPv4/IPv6 変換を実現する．また, IPv4 クライアントを実行するために, 名前解決の問題がある．この問題を DNS プロクシを実装し, IPv4/IPv6 変換を実装した仮想計算機モニタと協調させることにより, 解決する．

ソケットアウトソーシングにより IPv4/IPv6 変換を実現する利点は, 第 1 に, 高い性能が得られることである．10G bps のネットワークにおいて単 1 プロセッサを用いた実験では, 提案方式が実機上での IPv6 による通信とほぼ同等のスループットが得られた．第 2 の利点は, IPv4 と IPv6 が混在する場所は, 提案方式で追加する VMM と DNS プロクシに局所化され, IPv4 と IPv6 の混在による問題が生じないことである．

本論文は次のように構成される．2 章では, 本論文で想定する状況と既存方式の問題点を明確にする．3 章では, 提案方式について述べる．4 章では, 提案方式の実装について述べる．5 章では, 提案方式を評価する．6 章では, 関連研究について述べる．7 章では, 本論文についてまとめる．

2. 本研究で想定する状況と既存方式の問題点

この章では, 本研究で想定する状況を明確にし, 既存の方式では十分に対応できないことを述べる．本研究では, 次のような状況を想定する．

- ネットワークは, IPv6 だけを運用する．
- ホスト型 VMM により VM を構築し, その中で IPv4 アプリケーションを動作させる．
- IPv4 クライアントは必ずドメイン名でサーバを指定する．
- IPv4 のサーバ, および, クライアントは IPv4 固有のソケットオプションに依存しない．

このような状況において本研究では NAT-PT[18)] と同程度 IPv4/IPv6 変換を仮想計算機モニタで行う．具体的には, 次のようなことを実現する．

- VM の中で動いている IPv4 のクライアントは, 外部の IPv6 のサーバと通信できる．
- VM の中で動いている IPv4 のサーバは, 外部の IPv6 のクライアントと通信できる．

なお, この論文では, アプリケーションの通信内容の書き換えは今後の課題とし, 対象外とする．すなわち, この論文では, ALG (Application Level Gateway)[15)] のようなことは行わない．このため, たとえば通信内容に IPv4 アドレスが含まれていた場合には, 動作しない．そのようなアプリケーションとしては, FTP(File Transfer Protocol) や SIP(Session Initiation Protocol) がよく知られている．

1 章でも述べたように, このような状況で IPv4/IPv6 変換器を動作させる場所として, 既存の方式としては, 変換専用 VM, ゲスト OS, および, ホスト OS が考えられる．これら既存の方法は, 次の 2 つを同時に満たすことができない．

- 実機に近い高い性能を持つ．
- IPv4 と IPv6 が混在する場所が小さい．仮想計算機を利用する前に, IPv4 専用であった環境, および, IPv6 専用であった環境を変更しないでそのまま保つ．

変換専用 VM を用いる方法は, IPv4 と IPv6 が混在する場所が, 変換用の VM に局所化され, IPv4/IPv6 混在による問題はない．しかしながら, 通信経路が長く性能が低いという問題がある．ゲスト OS, または, ホスト OS で IPv4/IPv6 変換器を動作させると, 変換専用 VM で動作させる方法よりも通信経路が短いのでより高い性能が得られる．しかしながら, IPv4 と IPv6 がゲスト OS, または, ホスト OS で混在してしまう．IPv4 と IPv6 の混在は, 現在でもセキュリティ

上の問題を多く引き起こしている[5),6)]．本研究で想定しているように IPv6 が主になった状況においても，類似の問題が生じると予想される．たとえば，現在では IPv4 でファイアウォール等のセキュリティを向上させる仕組みがあるが IPv6 では利用できないことがある[6)]．IPv6 が主になった場合には，逆に IPv4 用のセキュリティを向上させる仕組みが保守されず利用できなくなる事態が予想される．さらに，仮想計算機を利用する前に，IPv4 専用だったゲスト OS，または，IPv6 専用だったホスト OS の環境を IPv4 と IPv6 が混在する環境に変更しなければならない．このように IPv4 専用，または IPv6 専用で安定的に動作している環境を IPv4 と IPv6 が混在する環境に変更することは，管理上大きな負担を伴う．

本研究では，ソケットアウトソーシングという手法を用いることで，高い性能を持ち，かつ，IPv4 と IPv6 が混在する場所が小さい IPv4/IPv6 変換を実現する．

3. ソケットアウトソーシングによる IPv4/IPv6 変換

この章では，ソケットアウトソーシングの概要，および，提案方式の概要と動作について述べる．

3.1 ソケットアウトソーシングの概要

ホスト型仮想計算機において，高速な入出力を実現する手法にアウトソーシングがある[7),10),21),22)]．アウトソーシングとは，ゲスト OS の高水準な処理をホスト OS に委譲することで I/O の高速化を実現している．具体的には，ゲスト OS の高水準のモジュールを置き換え，ホスト OS の機能を利用可能にする．ホスト OS とゲスト OS の間の通信は，仮想計算機に特化した RPC(Remote Procedure Call) である VMRPC(Virtual Machine RPC) という仕組みにより実装される[7)]．ゲスト OS 側で VMRPC のクライアントが，ホスト OS でサーバが動作する．

ソケットアウトソーシングは，アウトソーシングをソケットに適用し，ネットワーク I/O を高速化している[7),10),21)]．このとき，ゲスト OS 内のプロトコルスタックが VMRPC クライアントとなる．そして，ホスト OS のユーザ空間で動作しているサーバを呼び出す．このため，サーバが動作する VMM において，ゲスト OS が行ったソケットに対する操作を知ることができる．たとえば，socket システムコールや connect システムコールの引数を得ることができる．本研究では，この機能を利用し，IPv4/IPv6 変換器を実装する．

3.2 提案する IPv4/IPv6 変換の概要

本研究ではゲスト OS で動作する IPv4 プロセスが外部の IPv6 プロセスと通信するために，ソケットアウトソーシングを利用して IPv4/IPv6 変換を行う．IPv4/IPv6 変換器は，ゲスト OS 内の VMRPC のクライアント (ゲストクライアント)，VMM 内の VMRPC のサーバ (ホストサーバ)，プロトコルとアドレスの変換を行う IPv4/IPv6 変換モジュール，および，名前解決を行う DNS プロクシサーバからなる (図 1)．

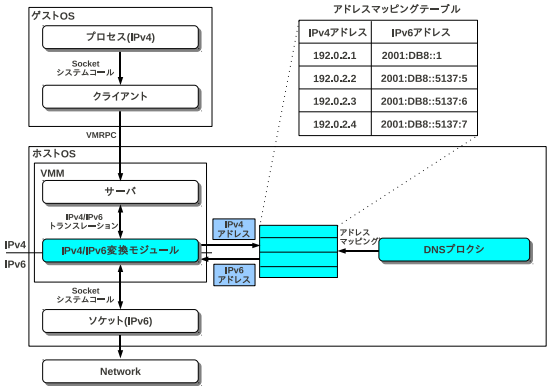

図 1 ソケットアウトソーシングによる IPv4/IPv6 変換
Fig. 1 IPv4/IPv6 Translation by socket-outsourcing

ゲストクライアントは，ゲスト OS 内で動作する IPv4 プロセスに対して IPv4 のソケットを提供する．このソケットに対するほとんどの処理を VMRPC によりホストサーバに移譲している．本研究では，ゲストクライアントとして，従来の高速化を目的としたものと同じものを用いている[7),10),21)]．

ホストサーバは，ゲストクライアントからの VMRPC による要求を受付るサーバである．従来の高速化を目的したホストサーバでは，ホスト OS で IPv4 ソケットを作成していた．本研究では，これを変更し，IPv4/IPv6 変換モジュールを呼び出すようにした．IPv4/IPv6 変換モジュールは，ホスト OS で IPv4 のソケットではなく IPv6 のソケットを作成する．この時，システムコールに現れる IP アドレスやその他の構造体を IPv4 のものから IPv6 のものへ変換する．

ゲスト OS 内だけで通用する IPv4 アドレスを**仮想 IPv4 アドレス**と呼ぶことにする．仮想 IPv4 アドレスは本方式において，システムコールの引数におけるアドレス構造体や DNS における要求と応答に現れる．

3.3 IPv4 サーバの動作

ゲスト OS 内で IPv4 サーバを実行する場合，IPv6 サーバと同様，ドメインを管理する DNS サーバにドメイン名と IPv6 アドレスを登録する．この時，IPv6 アドレスとしては，ホスト OS においてその VM に専用に割り当てたものを登録する．

ゲスト OS で IPv4 サーバが起動すると IPv4 ソケットを作成しようとする．すると，アウトソーシングにより，IPv4/IPv6 変換モジュールに制御が移る．このモジュールでシステムコールの引数を書換え，ホスト OS で IPv6 ソケットを作成する．このソケットに対して，IPv4 サーバは待ち受けを行う仮想 IPv4 アドレスとポート番号を設定する．すると IPv4/IPv6 変換モジュールは，IP アドレスについては仮想 IPv4 アドレスから IPv6 アドレスへの変換を行い，ポート番号についてはそのまま用いて，ホスト OS の IPv6 ソケットに設定する．

外部の IPv6 クライアントは名前解決を行うと，その VM に専用に割り当てた IPv6 アドレスを取得する．クライアントはその IPv6 アドレスとポート番号に対して接続要求を行う．

ホスト OS は，IPv6 ソケットで接続要求を受け付ける．このことは，VMM 内の IPv4/IPv6 変換モジュールに通知される．IPv4/IPv6 変換モジュールは，ホストサーバを通じて，ゲストクライアントに接続を通知する．ゲストクライアントは，IPv4 ソケットに対して接続受付の処理を行う．

IPv4 サーバは，接続された IPv4 ソケットに対してメッセージの送受信を行う．すると，そのことはゲストクライアント，ホストサーバを経由して，IPv4/IPv6 変換モジュールに伝えられる．IPv4/IPv6 変換モジュールは，システムコールの引数を変換して，ホスト OS の IPv6 ソケットに対してメッセージの送受信を行う．

3.4 IPv4 クライアントの動作

ゲスト OS 内で IPv4 クライアントを実行する場合は，クライアントはドメイン名から仮想 IPv4 アドレスを問い合わせる DNS クエリを発行する．しかし，本研究では，ネットワークとして IPv6 のみが運用されていることを前提としているので，たとえ目的のサーバが IPv4 と IPv6 の両方のアドレスを持っていたとしても，IPv6 の方のアドレスが必要である．

本研究では，この問題を解決するために，IPv4/IPv6 変換器と協調して動作する DNS プロクシを用いる．

DNS プロクシは IPv4 クライアントからの DNS 要求を受け付け，IPv6 アドレスの名前解決を行う．仮想 IPv4 アドレスを新たに生成し，IPv6 アドレスと対応付けを行って，仮想 IPv4 アドレスを含む DNS 応答を IPv4 クライアントに返す．

ゲスト OS 上のクライアントはこの仮想 IPv4 アドレスに対して接続するためのソケットを作成しようとする．すると，アウトソーシングにより，IPv4/IPv6 変換モジュールに制御が移る．IPv4/IPv6 変換モジュールは IPv6 ソケットを作成した後，IPv6 ホストへ接続する．このとき，仮想 IPv4 アドレスから先ほど対応付けた IPv6 アドレスを取得し，それを用いて接続する．

表 1 アドレスマッピングテーブルにおけるキーとバリュー
Table 1 keys and values in address mapping tables

テーブル	キー	バリュー
46 アドレスマッピングテーブル	仮想 IPv4 アドレス	IPv6 アドレス
64 アドレスマッピングテーブル	IPv6 アドレス	仮想 IPv4 アドレス

この DNS プロクシの動作は，DNS64[3] の方法と良く似ている．DNS64 との違いは，VMM 内で動作する IPv4/IPv6 変換器と協調的に動作する点にある．

4. IPv4/IPv6 変換の実装

この章では，3 章で述べた IPv4/IPv6 変換モジュール，DNS プロクシ，および，それらで用いられるアドレスマッピングテーブルについて述べる．

4.1 アドレスマッピングテーブル

アドレスマッピングテーブルとは，仮想 IPv4 アドレスと IPv6 アドレスの対応表である．VMM 内の IPv4/IPv6 変換モジュールは，これを用いて仮想 IPv4 アドレスと IPv6 アドレスの変換を行う．アドレスマッピングテーブルは **46 アドレスマッピングテーブル**と **64 アドレスマッピングテーブル**からなる (表 1)．46 アドレスマッピングテーブルではキーは仮想 IPv4 アドレス，バリューは IPv6 アドレスとなり，64 アドレスマッピングテーブルにおいてはキーは IPv6 アドレス，バリューは仮想 IPv4 アドレスとなる．

マッピングテーブルには，次のような時にエントリを追加する．

- ゲスト OS で動いている IPv4 クライアントが IPv6 サーバと接続する時．詳しくは，4.2 節で述べる．
- ゲスト OS で動いている IPv4 サーバが，accept() や recvfrom() などのシステムコールにより，IPv6 クライアントからの接続要求やメッセージを受け付けた時．詳しくは，4.3 節で述べる．

4.2 DNS プロクシ

DNS プロクシは，名前解決で IPv4 アドレス，IPv6 アドレスの取得に関わる要求と応答の変換を行う．DNS プロクシの動作を以下に示す．

(1) ゲスト OS 上の IPv4 クライアントから IPv4 アドレスを得るための要求を受信する．
(2) 受け取った要求からドメイン名を取り出す．このドメイン名から IPv6 アドレス要求を作成する．作成した IPv6 アドレス要求を予め指定さ

```
1 int socket_4to6(int domain, int type, int protocol){
2   return socket(AF_INET6, type, protocol);
3 }
```

図 2 IPv4/IPv6 変換モジュールにおける socket() システムコールの変換
Fig. 2 Translation of the system call socket() in IPv4/IPv6 Translation module

れた DNS サーバに送信する.

(3) DNS サーバから IPv6 アドレスを含む応答を受信する.

(4) エラーならその旨を通知する応答を作成し, ゲスト OS 上の IPv4 クライアントに送信する.

(5) 受け取った IPv6 アドレスについて, 既にアドレスマッピングが行われているかを確認する. マッピングが存在しないならば, 仮想 IPv4 アドレスを新たに生成して登録する.

(6) マッピングテーブルから IPv6 アドレスと対応する仮想 IPv4 アドレスを取り出す. この仮想 IPv4 アドレスを含む応答を作成する. 作成した応答をゲスト OS 上の IPv4 クライアントに対して送信する.

4.3 IPv4/IPv6 変換モジュール

この節では, IPv4/IPv6 変換モジュールにおける主要な手続きの実装について述べる.

4.3.1 socket()

socket() は, プロトコルを決定するシステムコールである. ゲスト OS において socket() システムコールが実行されると, TCP/IP, および, UDP/IP のプロトコルが指定された場合, IPv4/IPv6 変換モジュールの手続きが呼ばれる☆. この手続きでは, ホスト OS の socket() システムコールを呼ぶ. この時, 引数のプロトコルを IPv4 から IPv6 に書換える (図 2).

4.3.2 bind() と connect()

bind() は, 自分自身のソケットに名前 (IP アドレスとポート番号) を付けるシステムコールである. ゲスト OS で実行された bind() システムコールの引数には, IPv4 のアドレス構造体 (struct sockaddr_in) が含まれている. IPv4/IPv6 変換モジュールでは, まずそのアドレス構造体から仮想 IPv4 アドレスを取り出し, それをマッピングテーブルを用いて IPv6 アドレスへ変換する. ただし, IPv4 のアドレスとして INADDR_ANY が指定された時には, IPv4/IPv6 変換モジュールは, その仮想計算機に割り当てられた IPv6 アドレスへ変換する. 次に, アドレス構造体からポート番号を取り出す. こうして得られた IPv6 のアドレスとポート番号を使って IPv6 のアドレス構造体 (struct sockaddr_in6) を作成する. 最後に, それを使って, ホスト OS の bind() システムコールを呼ぶ.

connect() は, 通信相手の名前を指定するシステムコールである. IPv4/IPv6 変換モジュールにおける connect() の処理は, bind() の処理とよく似ている. まず引数の IPv4 のアドレス構造体から IPv6 のアドレス構造体を作成する. そして, ホスト OS の connect() システムコールを呼ぶ.

4.3.3 sendmsg() と recvmsg()

sendmsg() は, メッセージを送信するシステムコールである. ソケットアウトソーシングでは, ゲスト OS 内の VMRPC のクライアントは send(), sendto(), sendmsg(), write() などのシステムコールを全て sendmsg() という手続きに一元化して VMM 内の VMRPC のサーバを呼び出す. IPv4/IPv6 変換モジュールでは, まず引数の IPv4 アドレス構造体を取り出し, bind() と同様に IPv4 アドレス構造体から IPv6 アドレス構造体に変換する. 次に, 新しく msghdr 構造体を作成し, 引数の msghdr 構造体のコピーする. ただし, IPv4 のアドレス構造体とそのサイズについては IPv6 のアドレス構造体へ変換する. 最後に, その新たに作成した msghdr 構造体を引数として ホスト OS の sendmsg() システムコールを呼ぶ.

recvmsg() は, メッセージを受信するシステムコールである. ソケットアウトソーシングでは, ゲスト OS 内の VMRPC のクライアントは recv(), recvfrom(), recvmsg(), read() などのシステムコールを全て recvmsg() という手続きに一元化して VMM 内の VMRPC のサーバを呼び出す. IPv4/IPv6 変換モジュールにおける recvmsg() の処理は, sendmsg() と良く似ている. 異なるのは, recvmsg() システムコールを先に呼び, その後, 結果に含まれている IPv6 のアドレス構造体を IPv4 のアドレス構造体に変換する点である.

4.3.4 accept(),getsockname(), および,getpeername()

accept() は, TCP/IP のサーバにおいてクライアントからの接続要求を受付けるシステムコールである. IPv4/IPv6 変換モジュールでは, まず IPv6 のクライアントのアドレスを受け取るために, IPv6 アドレス構造体を引数として, ホスト OS の accept() システムコールを発行する. ホスト OS へ発行した accept() システムコールからリターンした時, クライアントの IPv6 アドレスが得られるので, それをマッピングテー

☆ Unix Domain Socket の場合は, ゲスト OS 内で処理される.

ブルを用いて仮想 IPv4 アドレスに変換する．もしマッピングテーブルにエントリが存在しない場合, 新たに仮想 IPv4 アドレスを生成し, この仮想 IPv4 アドレスとその IPv6 アドレスのマッピングを追加する．次に, IPv6 アドレス構造体からポート番号を取り出す．最後に, ゲスト OS から渡された引数の IPv4 アドレス構造体に, プロトコルと 仮想 IPv4 アドレス, および, ポート番号を設定する．

getsockname() と getpeername() は, それぞれ自分自身, および, 通信相手のソケットの名前 (IP アドレスとポート番号) を得るシステムコールである．これらに対応した IPv4/IPv6 変換モジュール における処理は, accept() の処理と同様に IPv6 アドレスを IPv4 アドレスに変換する．

5. 評　　価

この章では, 提案方式の評価を行う．仮想計算機のゲスト OS で IPv4 アプリケーションを動作させる場合, 2 章で述べたように, 既存方式では性能面に問題があるか, または, ホスト OS 全体, または, ゲスト OS 全体で IPv4 と IPv6 が混在してしまうという問題があった．この章では, まず提案方式であるソケットアウトソーシングによる IPv4/IPv6 変換は, 高いスループット, および, 実用上問題が無い応答性能があることを示す．特に本方式のスループットは, IPv4/IPv6 変換器をゲスト OS やホスト OS で実行する方式よりも高く, また, CPU 資源の消費も少ないことを示す．次に, 提案方式では, IPv4 と IPv6 が混在する場所がごく狭い領域だけであることを示す．これにより, 提案方式では, 実機に近い高い性能を持ち, かつ, IPv4 と IPv6 が混在する場所が小さいことを同時に満たすことを示す．

5.1 性 能 評 価

性能評価として, 変換器をホスト OS で動作させた場合, ゲスト OS で動作させた場合, ソケットアウトソーシングにより VMM で変換した場合のスループット, CPU 利用率, 応答時間を測定した．なお, 既存の IPv4/IPv6 変換器として, アプリケーション層で動作する DeleGate を用いた[14]．DeleGate は多機能プロキシサーバであり, IPv4 クライアントからの接続を IPv6 サーバへ中継する, あるいは IPv6 クライアントからの接続を IPv4 サーバへ中継する機能も持つ．IPv4/IPv6 変換器の実装として IP 層で動作する ecdysis[13] と naptd[17] についても実験を行ったが, 我々の環境ではうまく動作しなかったり性能が低すぎるという問題があったので, それらについては実験結果を含めていない．また, 実機については, 仮想計算機を実行するためのホスト OS を用いた．

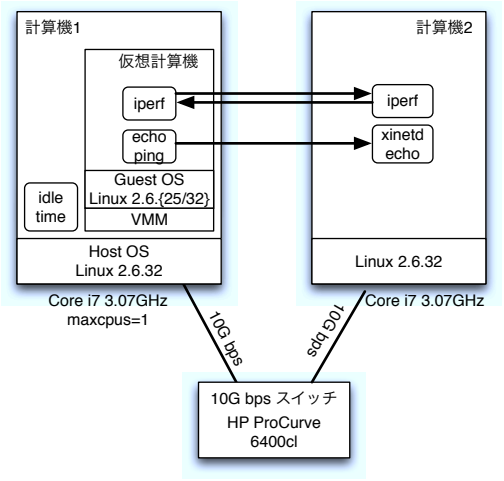

図 3 実験環境
Fig. 3 The experimental environment.

測定したのは次の通りである．

(1) 実機–IPv4[Physical–v4]
(2) 実機–IPv6[Physical–v6]
(3) 完全仮想化–IPv4[Emulation–v4]
(4) 準仮想化–IPv4[VirtIO–v4]
(5) ソケットアウトソーシング–IPv4[SOS–v4]
(6) 実機 DeleGate–v4/v6[Physical+DeleGate–v4/v6]
(7) 準仮想化+DeleGate–v4/v6(Host)[VirtIO+DeleGate–v4/v6(Host)]
(8) 準仮想化+DeleGate–v4/v6(Guest)[VirtIO+DeleGate–v4/v6(Guest)]
(9) ソケットアウトソーシングを用いた IPv4/IPv6 変換 [SOS-v4/v6]

(1)–(5) では, 参考として変換器を動作させない場合における通信性能を測定した．また, (6) は参考として, ホスト OS 上のプログラムにより, 通信を変換した場合の性能を測定した．この中で, (7) は 1 章で述べたホスト OS で変換器を実行する方式に該当する．(8) はゲスト OS で変換器を実行する方式に該当する．1 章では, この他に変換専用 VM を用いて変換する方法についても述べたが, この方式の性能は低いことは自明であるため, 今回は測定の対象に含めなかった．

5.1.1 実 験 環 境

実験環境を図 3 に示す．実験を行う計算機は, Intel Core i7 3.07GHz の CPU, Intel CX4 10G bps のイーサネットカードを備えたものを用いた．ホスト OS には Linux 2.6.32 を用い, その計算機を 2 台用意し, スイッチにより接続してネットワークを構成した．なお,

スイッチには HP ProCurve Switch 6400cl を用いた. また, 仮想計算機モニタは Linux KVM を用いた. これはソケットアウトソーシングが Linux KVM に実装されているためである. 完全仮想化では NIC として e1000 をエミュレーションし, 準仮想化では VirtIO ドライバを使用した. ゲスト OS としては, 完全仮想化, 準仮想化では Linux 2.6.32 を, ソケットアウトソーシングでは Linux 2.6.25 を用いた.

5.1.2 実験方法–スループットと CPU 利用率

この実験では, 提案手法が実機なみの高いスループットを持っていることを示す. そのために, スループットを測定するプログラム iperf を用いた.

この実験では図 3 の計算機 1 上で KVM を実行し, そのゲスト OS 上で iperf サーバ, および, クライアントを実行した. CPU 利用率を測定するために, 計算機 1 はホスト OS 起動時に maxcpus を 1 に設定し, コア数を 1 つに限定した. 計算機 2 上で iperf クライアント, および, サーバを実行した. これらの iperf サーバ–クライアント間で TCP により通信を行い, スループットを測定した. この実験で iperf は, クライアントからサーバに対して大量のメッセージを送信する.

通信のスループットが何によって決定づけられているのかを調べるために, CPU の利用率を測定した. もし CPU の利用率が低ければ, スループットはネットワークインターフェースによって決定されていることが分かる. もし, CPU 利用率が 1 に近ければ, スループットは CPU によって決定されていることが分かる. また, CPU 利用率が低いことは, 多くの仮想計算機をホスティングするときに有利である.

iperf を実行中にその処理に必要な CPU の利用率を測定するために, 低優先度で CPU 時間を消費するだけのプログラムを同時に実行した. そして, このプログラムの CPU 利用率を測定し, 1 からこれを引くことで iperf の処理に利用された CPU の利用率を算出した.

5.1.3 実験結果–スループットと CPU 利用率

図 4 に仮想計算機内で iperf のサーバを動作させた場合のスループットを, 図 5 に CPU 利用率を示す. 実機–IPv4, 実機–IPv6, 完全仮想化–IPv4, 準仮想化–IPv4, ソケットアウトソーシング–IPv4 のスループットは 9.89G bps, 9.87G bps, 3.11G bps, 3.92G bps, 9.84 G bps となり, ソケットアウトソーシングを用いた場合は実機–IPv4 とほぼ同等の性能が得られた. また, 実機 DeleGate–v4/v6, 準仮想化+DeleGate–v4/v6(Host), 準仮想化+DeleGate–v4/v6(Guest), ソケットアウトソーシングによる変換

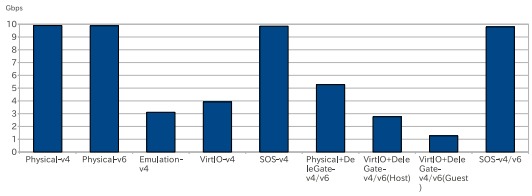

図 4 iperf サーバを仮想計算機内で実行した場合のスループット
Fig. 4 Throughput when the iperf server was exectuted in the virtual machine.

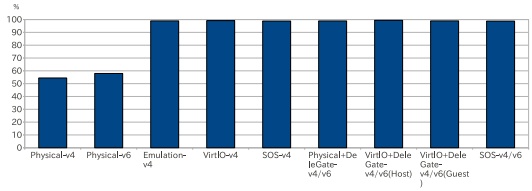

図 5 iperf サーバ動作時の CPU 利用率
Fig. 5 CPU utilization when the iperf server was executed in the virtual machine.

の場合, 5.27G bps, 2.76G bps, 1.26G bps, 9.78G bps となった. この環境では提案方式は実機なみの良いスループットが得られた.

SOS–v4/v6 は, ゲスト OS やホスト OS で変換をしている, VirtIO+DeleGate–v4/v6(Host) や VirtIO+DeleGate–v4/v6(Guest) よりも高速であった. その主な理由は, 提案方式では VMM で変換を行っているのに対して, それらがアプリケーション層で変換を行っているからである. IP 層で変換を行う ecdysis や naptd を用いれば, DeleGate を用いた方式よりも高速となることが推察される. その速度は変換を行っていない VirtIO–v4 に変換のオーバヘッドを加えたものになる. 提案方式は, 変換を行っていない VirtIO よりも高速である. 従って, 仮に IP 層での変換のオーバヘッドが 0 であったとしても, 提案方式は, IP 層で行うような変換器よりも高速であるといえる.

図 5 に示された CPU 利用率は, 実機–IPv4 が 55%, 実機–IPv6 が 58% であり, その他はどれも 100% となった. このことから, この実験では実機以外は CPU によって性能が決定されていることが分かる.

図 6 に iperf のクライアントを動作させた場合のスループットを, 図 7 に CPU 利用率を示す. 図 6 のクライアントのグラフは, 図 4 のサーバのグラフと類似している.

図 7 に示した CPU 利用率は, 実機–IPv4 が 46%, 実機–IPv6 が 48%, ソケットアウトソーシングで変換を行わない場合は 76%, 変換を行う場合は 83%であ

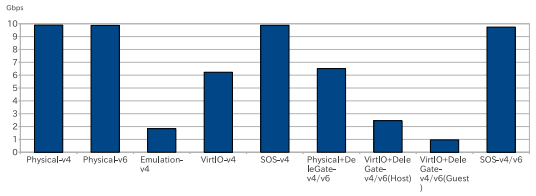

図 6 iperf クライアント動作時のスループット
Fig. 6 Throughput when the iperf client was exectuted in the virtual machine.

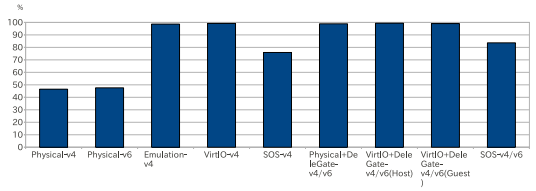

図 7 iperf クライアント動作時の CPU 利用率
Fig. 7 CPU utilization when the iperf client was executed in the virtual machine.

り, その他は 100% という結果となった. 図 7 に示したクライアントの CPU 利用率は, 実機とソケットアウトソーシングにおいて, 図 5 に示したサーバの CPU 利用率よりも低下している. その理由は, クライアントの処理, すなわちメッセージの送信処理がメッセージの受信よりも軽いからである.

また, ソケットアウトソーシングにおいて, IPv4 のものと IPv4/IPv6 変換を行うものを比較すると, 変換を行うものが約 7% 余計に消費していた. この 7% には IPv4/IPv6 の変換のオーバヘッドが含まれているが, 7% という値は変換のオーバヘッドとしては大きすぎると考えている. 現在, 他の原因がないか調査している.

以上の実験結果から, 提案方式はホスト OS における変換やゲスト OS における変換方式と比べて, スループットと CPU 利用率において優れているといえる.

5.1.4 実験方法–応答時間

この実験では提案方式が実機と遜色の無い応答性能を持っていることを示す. そのために, echoping を使って TCP の応答時間を測定した. echoping は, echo サービスを利用して応答時間を測定するプログラムである☆.

この実験では, まず計算機 2 上の xinted の内部に組み込まれている echo サーバを動作させる. これに対して計算機 1 の仮想計算機上で echoping プログラムを実行し, 応答時間を測定した. これをそれぞれの方式ごとに応答速度の測定を 30 回実施し, その平均応答速度を算出した.

☆ http://echoping.sourceforge.net/

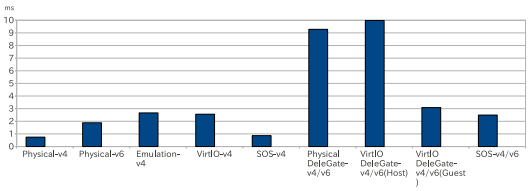

図 8 echoping による応答時間
Fig. 8 Measured latency with the echoping program.

5.1.5 実験結果–応答時間

図 8 は各方式の平均応答時間のグラフである. 各方式の平均応答時間は, 実機–IPv4 が 0.74 ms, 実機–IPv6 が 1.88 ms, 完全仮想化–IPv4 が 2.66 ms, 準仮想化–IPv4 が 2.55 ms, ソケットアウトソーシング–IPv4 は 0.86 ms であった. また, 実機 DeleGate–v4/v6, 準仮想化+DeleGate–v4/v6(Host), 準仮想化+DeleGate–v4/v6(Guest), ソケットアウトソーシングによる変換の場合, 9.28 ms, 9.98 ms, 3.08 ms, 2.49 ms となった. レイテンシについても第 5.1.3 項で述べたスループットと同じ傾向がある. すなわち, スループットが高いものはレイテンシが小さくなっている.

SOS–v4/v6 は, ベースとしている実機–IPv6 より 0.6 ms ほど遅くなっているが, これはインターネットでの通信遅延の変動よりも小さく, 実用上問題が無いといえる. また, SOS–v4/v6 は SOS–v4 より遅延が大きくなっている. その主な原因は SOS–v4/v6 がホスト OS の IPv6 を使っているのに対して, SOS–v4 が IPv4 を使っていることによる. 図 8 に示したように, Physical–v4 と Physical–v6 では v4 のほうが高速である. 同様に, SOS–v4 と SOS–v4/v6 では SOS–v4 のほうが高速である.

以上の実験結果から, 提案方式はホスト OS における変換やゲスト OS における変換方式と比べて, 応答性能が良いといえる.

5.2 IPv4 と IPv6 の混在

提案方式において, IPv4 と IPv6 が混在するのは, 提案方式で追加する VMM 内の IPv4/IPv6 変換モジュール, DNS プロクシ, および, それらの間でデータをやり取りするためのアドレスマッピングテーブルだけである. IPv4 と IPv6 が混在するのはこのようにシステム全体の中でごく狭い領域だけである. ホスト OS は IPv6 専用であり, レガシーアプリケーションが動作するゲスト OS は IPv4 専用になる. このように提案方式では, ホスト OS もゲスト OS も IPv4 と

IPv6 は混在しないので, 2 章で述べた混在に伴う様々な問題を避けることができる. また, 仮想計算機を利用する前に IPv4 専用であった環境, および, IPv6 専用であった環境を変更しないでそのまま保つこともできる, という利点もある.

なお, 提案方式はソケットアウトソーシングを用いる. ソケットアウトソーシングでは, ゲスト OS のカーネル内にあるプロトコルスタックを置き換える必要がある. ゲスト OS において変更すべき場所はこの部分だけであり, ゲスト OS に含まれる設定ファイルやアプリケーションのバイナリを置き換える必要はない.

5.3 現在の実装の限界

本論文では, ソケットアウトソーシングを用いて IPv4/IPv6 変換を実現することを提案し, その実装を示した. 現在の実装では, アプリケーションの通信内容の書き換えを行っていない. すなわち, ALG のようなことは行っていない. このため, 通信内容に IPv4 アドレスが含まれるようなプロトコルには対応することができない. 今後, ALG を実装したいと考えている. それには, 4.3 節で述べた recvmsg() や sendmsg() において, 通信内容を書き換える必要がある.

現在の実装では, ソケットオプションには対応していない. その理由の 1 つは, IPv4 と IPv6 は別のプロトコルであり, IPv4 専用のオプションや IPv6 専用のオプションが存在するために, 完全に対応させることができないからである. たとえば, IPv4 の IP オプションヘッダを扱うものは IPv4 専用であり, IPv6 の拡張ヘッダを扱うものとは対応しない. 今後は, 要求に応じて IPv4 と IPv6 で対応可能なソケットオプションについても変換したいと考えている.

6. 関 連 研 究

TCP レイヤにおける IPv4/IPv6 変換手法として, Transport Realy Translator (TRT) 方式がある. IPv6-to-IPv4 変換器として, BSD 系 OS 固有のインターフェースである faith デバイス, faithd サーバ, 及び DNS プロクシ totd の協調動作により変換を行う[9]. この手法は IPv4 サーバを IPv6 に公開するものであり, IPv4 クライアントを IPv6 サーバに接続させることはできない. これに対して, 本研究では, サーバ, クライアントのいずれにも対応できる.

IP 層における IPv4/IPv6 変換技術として Network Address Translation-Porotocol Translation(NAT-PT)[18] と DNS-Application Level Gateway(DNS-ALG)[16] の協調動作によるものが存在する. NAT-PT は IPv4 ネットワークと IPv6 ネットワークの境界に位置し, 通過するパケットの IP ヘッダを書換えを行う. 本研究では, VMM において変換を行う点が異なる.

VMM を用いて既存 IPv4 Web システムを IPv6 化する手法が提案されている[20]. この手法では, VMware ESXi Server 上に IPv6 に対応する Web プロクシサーバ, DNS サーバを稼働させ, ネットワークに追加することで, 既存 IPv4 Web システムをデュアルスタック化する. この方法は 1 で述べた専用 VM を使う方法に相当する. この方法では DeleGate を用いて, IPv4/IPv6 変換を実現している. この方法と比較して, 本研究の特徴は VMM で変換していることとソケットアウトソーシングにより高い性能が得られていることである.

ソケットシステムコールレベルでの IPv4/IPv6 変換手法として, Bump-in-API (BIA)[11] がある. BIA はソケット API と TCP スタックの間で変換器を動作させる. 本研究は BIA の 1 つの実装としても位置づけることができる. BIA の実装では, 動的リンクライブラリ (dynamic link library, DLL) を置き換えるものがある[8]. しかしながら, この方法は, セキュリティ上, 動的リンクライブラリの置き換えを許されていない場合や静的にリンクされたアプリケーションでは利用することができない. 本研究は, 静的にリンクされたアプリケーションであっても変換できる. BIA をカーネル内で実装することも考えられる. この方法と比較して本研究の特徴は, レガシーのアプリケーションを VMM 内で動作させる場合に高い性能が得られる点, および変換器の開発がホスト OS 上で行えるため容易である点にある.

7. お わ り に

この論文では, ソケットアウトソーシングという手法を用いて仮想計算機モニタ内で IPv4/IPv6 変換を行う方法を提案した. ソケットアウトソーシングは, 元々はゲスト OS のソケット層の処理をホスト OS に移譲することでネットワーク入出力を高速化する手法として提案されたものである. この論文では, ソケットアウトソーシングを高速化ではなく機能拡張に用いることで, IPv4/IPv6 変換を実現した. 提案方式では, ゲスト OS 内で行われたソケット関連のシステムコールがそのレベルで VMM において取得できる. VMM では, IPv4 のアドレスやプロトコルを IPv6 のものに変換してホスト OS に対してシステムコールを発行する. IPv4 クライアントを動作させるために, ホスト OS 上で動作し, 仮想計算機モニタと協調して動作する DNS プロクシを用いている.

提案方式は, ホスト OS とゲスト OS ともに Linux

で動作している. 10G bps のネットワークにおいて単 1 プロセッサを用いた実験では, 提案方式が実機上での IPv6 による通信とほぼ同等のスループットが得られた. IPv4/IPv6 変換をゲスト OS やホスト OS で行う方式として, 本方式は高い性能が得られた. また, 本提案方式は IPv6 が混在する場所は, 提案方式で追加する VMM と DNS プロクシに局所化されるという利点が有る. すなわち, IPv4 と IPv6 の混在による問題が生じないことはなく, また仮想計算機を利用する前に安定的に動作している IPv4/IPv6 専用環境を変更する必要がない.

今後の課題は, ALG (Application Level Gateway) を実装して FTP や SIP 等の通信内容に IPv4 アドレスが含まれるようなプロトコルに対応することである. また, ソケットオプションや IPv6 拡張ヘッダ等の機能を利用可能にしていきたいと考えている.

謝辞

本研究の一部は, 総務省戦略的情報通信研究開発制度 (SCOPE) の支援を受けて行われた.

参 考 文 献

1) Atwood, J. W., Das, K. C. and Jiang, X. S.: IPv4/IPv6 Translation, *Proceedings of the Linux Symposium*, pp. 34–43 (2003).
2) Bagnulo, M., Matthews, P. and van Beijnum, I.: Stateful NAT64: Network Address and Protocol Translation from IPv6 Clients to IPv4 Servers, RFC 6146 (2011).
3) Bagnulo, M., Sullivan, A., Matthews, P. and van Beijnum, I.: DNS64: DNS Extensions for Network Address Translation from IPv6 Clients to IPv4 Servers, RFC 6147 (2011).
4) Bao, C., Huitema, C., Bagnulo, M., Boucadair, M. and Li, X.: IPv6 Addressing of IPv4/IPv6 Translators, RFC 6052 (2010).
5) Caicedo, C., Joshi, J. and Tuladhar, S.: IPv6 Security Challenges, *IEEE Computer*, Vol. 42, No. 2, pp. 36–42 (2009).
6) Davies, E., Krishnan, S. and Savola, P.: IPv6 Transition/Co-existence Security Considerations, RFC 4942 (2007).
7) Eiraku, H., Shinjo, Y., Pu, C., Koh, Y. and Kato, K.: Fast networking with socket-outsourcing in hosted virtual machine environments, *Proceedings of the 2009 ACM symposium on Applied Computing*, SAC '09, ACM, pp. 310–317 (2009).
8) Engelen, A.: BIAsed–transparent IPv4–to–IPv6 API translator (2003). `http://biased.sourceforge.net/libbiased.pdf`.
9) Hagino, J. and Yamamoto, K.: An IPv6-to-IPv4 Transport Relay Translator, RFC 3142 (2001).
10) Koh, Y., Pu, C., Shinjo, Y., Eiraku, H., Saito, G. and Nobori, D.: Improving Virtualized Windows Network Performance by Delegating Network Processing, *Proceedings of the 2009 Eighth IEEE International Symposium on Network Computing and Applications*, IEEE Computer Society, pp. 203–210 (2009).
11) Lee, S., Shin, M.-K., Kim, Y.-J., Nordmark, E. and Durand, A.: Dual Stack Hosts Using "Bump-in-the-API" (BIA), RFC 3338 (2002).
12) Li, X., Bao, C. and Baker, F.: IP/ICMP Translation Algorithm, RFC 6145 (2011).
13) Perreault, S., Dionne, J.-P. and Blanchet, M.: Ecdysis: Open-Source DNS64 and NAT64, Asia BSD Conference (2010).
14) Sato, Y. and Hamazaki, Y.: DeleGate: A general purpose application level gateway, *Worldwide Computing and Its Applications*, pp. 426–441 (1997).
15) Srisuresh, P. and Holdrege, M.: IP Network Address Translator (NAT) Terminology and Considerations, RFC 2663 (1999).
16) Srisuresh, P., Tsirtsis, G., Akkiraju, P. and Heffernan, A.: DNS extensions to Network Address Translators (DNS_ALG), RFC 2694 (1999).
17) Tomicki, L.: naptd: Network Address Translation, Protocol Translation IPv4/IPv6 (2011). `http://tomicki.net/naptd.php`.
18) Tsirtsis, G. and Srisuresh, P.: Network Address Translation - Protocol Translation (NAT-PT), RFC 2766 (2000). Obsoleted by RFC 4966, updated by RFC 3152.
19) Tsuchiya, K., Higuchi, H. and Atarashi, Y.: Dual Stack Hosts using the "Bump-In-the-Stack" Technique (BIS), RFC 2767 (2000).
20) 高宮紀明, 三上博英: 仮想環境を利用した既存 IPv4 Web システムの IPv6 対応, 情報処理学会創立 50 周年記念（第 72 回）全国大会 (2010).
21) 齊藤剛, 新城靖, 榮樂英樹, 佐藤聡, 中井央, 板野肯三: 仮想計算機におけるアウトソーシングのためのゲスト–ホスト間 RPC, 第 20 回コンピュータシステムシンポジウム, ポスター・デモセッション (2008).
22) 豊岡拓, 新城靖, 齊藤剛: ホスト型仮想計算機環境におけるファイル入出力の VFS アウトソーシングによる高速化, コンピュータシステムシンポジウム論文集, Vol. 21, No. 13, pp. 33–40 (2009).