WIDE Technical-Report in 2011

IPv6のみで構成されたIaaSシステム用のIPv4-IPv6変換ソフトウェアおよびその運用システムの実装と評価

wide-tr-cloud-map646-eval-00.pdf

WIDE Project : http://www.wide.ad.jp/

Title: IPv6 のみで構成された IaaS システム用の IPv4-IPv6 変換ソフトウェアおよびその運用システムの実装と評価
Author(s): 石田 渉, 島 慶一
Date: 2011-12-01

IPv6のみで構成されたIaaSシステム用のIPv4-IPv6変換ソフトウェアおよびその運用システムの実装と評価

石田 渉[†]　　島 慶一[‡]
[†]東京大学　　[‡]IIJ イノベーションインスティテュート

インターネット運用は IPv4 のみの運用から IPv4/IPv6 のデュアルスタック運用に徐々に移行しつつある。運用負荷を考えるとシングルスタックでの運用が望ましいものの、長く残ると考えられる IPv4-only ユーザー、今後現れるであろう IPv6-only ユーザーの両方への対応は必須である。本論文では IaaS のような仮想計算機資源提供環境を IPv6 のみで運用しつつ、ユーザーへはデュアルスタックでの接続サービス提供を可能にするための IPv4-IPv6 変換ソフトウェアとその運用モデルを提案する。IaaS サービスを用いたサーバー運用では、ひとつの仮想計算機にひとつの IPv4 アドレスが事実上一対一で割り当てられるという前提のもとに、高速、スケーラブル、耐障害性にすぐれたシステム設計を提案し、評価を行った。

Implementation and Evaluation of IPv4-IPv6 Translator Software and its Operation Architecture Designed for IPv6-only IaaS Backend Systems

Wataru Ishida[†]　　Keiichi Shima[‡]
[†]The University of Tokyo　[‡]IIJ Innovation Institute Inc.

Internet operation is migrating from IPv4-only operation to IPv4/IPv6 dual stack operation. From the viewpoint of the operation cost, it is best if the operation can be done only with a single protocol, either IPv4 or IPv6. However, it is mandatory to accept IPv4-only users who will remain for a long time, and IPv6-only users who will appear in the future as IPv6 deployment progresses. In this paper, we will provide IPv4/IPv6 header translation software and its operation model designed for the IaaS system operated with IPv6-only network. We assume that a global IPv4 address and IPv6 virtual machine are mapped one-to-one when the virtual machine is used as a server. With this assumption, we designed high-performance, scalable, and redundant IaaS system, and evaluated the system.

1　はじめに

計算機の速度は年々上昇を続けているものの、単一の計算機の性能向上にだけ頼った方式には限界が見え始めている。近年はネットワークで接続された計算機を統合して利用する、グリッドあるいはクラウドと呼ばれる技術に注目が集まっており、その利用方法やアプリケーションモデル、資源の効率的な運用方法などの研究が盛んである。また、計算機の資源をより細かい粒度で活用するために、ひとつの計算機を仮想的に複数の計算機資源に分割して運用する仮想計算機技術も成熟してきており、今後物理計算機の台数をはるかに越える多数の仮想計算機を用いた計算機ネットワークが構築されると考えられている。

計算機をネットワークに接続する場合、これまではIPv4が利用されてきた。しかしながら、IPv4 アドレスの枯渇に伴い、IPv6 への移行が現実のものとなってきている。ただし、IPv4 が枯渇した後も、それまでに分配された IPv4 アドレスが利用できなくなるわけではなく、今後長期間に渡って IPv4/IPv6 のデュアルスタック環境が運用されていくものと考えられている。

計算機資源のネットワーク設定を管理する場合、今後はデュアルスタックでの運用が必須となる。しかしながら、IPv4 と IPv6 は、そのアドレス空間の大きさの差を除いてほぼ同じ機能を提供しているにも関わらず相互に互換性がないため、運用者は類似したネットワークプロト

コルを二重に管理する必要がある。

運用管理者としては、サービスに必要なバックエンドネットワークをシングルスタック運用しつつ、外に見える部分はデュアルスタックとして公開することで、運用コストを抑えつつ、今後のインターネットの進化に備えることができるのが理想である。本論文では、仮想計算機資源を提供するシステム (IaaS、Infrastructure As A Service) を対象とし、そのバックエンドを IPv6 のみによるシングルスタック運用としつつ、外部に公開されるサービス部分を IPv4-IPv6 変換ソフトウェアを用いてデュアルスタック化するシステムを提案する。IPv4-IPv6 変換ソフトウェアは特に新しい技術ではなく、IPv6 が提案された初期から存在する。しかしながら、一般的な変換ソフトウェアはひとつの IPv4 アドレスを複数の IPv6 ノードで共有することを前提としており、アドレスの対応状態の管理、セッションの管理などが複雑であり、また通信状態を保持管理しなければならないという特性上、スケーラビリティや耐障害性が問題となる。提案システムでは、対象を IaaS に限定することによって、アドレス変換を一対多から一対一に制限している。これにより、現実的な IaaS サービスを提供しつつ、既存の一対多 IPv4-IPv6 変換ソフトウェアが抱える問題を解決し、シングルスタックでのバックエンド運用を可能としている。

2 関連技術

IPv6 の仕様から IPv4 との後方互換性を省くという決断がなされた時点から、すでに両プロトコル間での相互接続問題を解決するための技術の検討は始まっていた。これらの技術は大きく三つにわけることができる。一つ目はアプリケーション層で IPv4 ノードと IPv6 ノードの相互接続性を確保する方法、二つ目はトランスポート層でデータを中継する方法、最後が IP 層でヘッダ変換を行う方法である。

アプリケーション層での接続性の確保はアプリケーション毎に設計された代理サーバーを用いて実現する。IPv4 と IPv6 網の境界に代理サーバーを設置し、特定のアプリケーション (たとえばウェブサービスなど) の要求を受け付ける。代理サーバは受信した要求を解釈し、そこから新たに本来のサーバーへ向けた要求を作成し、適切な IP プロトコルで再送信する。要求に対する応答も、同様の手順で本来の要求元のクライアントへ転送される。代理サーバーはアプリケーションプロトコルを完全に把握しているため、三つの方式の中では最も細かい制御が可能な仕組みとなる。反面、アプリケーションプロトコルが代理サーバーを経由しても問題ないように設計されている必要がある。また、アプリケーション毎に代理サーバーの開発が必要になるなどの短所がある。アプリケーションによっては残る二つの方式で対応可能な場合も多く、本方式は複雑な中間処理を実現したい場合の特殊対応として利用する場合が多い。

トランスポート層での中継では、IPv4 と IPv6 の境界に位置する中継サーバーが、socket レベルで接続を終端、再接続することで両プロトコル間を橋渡しする。代表的な実装として SOCKS64[1] や Transport Relay Translator[2] が挙げられる。SOCKS64 では、SOCKS64 ライブラリを利用することによって、DNS の名前解決とデータ送受信機能を差し替え、通信が SOCKS64 中継サーバーを経由するように調整される。SOCKS64 ライブラリを新たにリンクする必要があるという点で、クライアントの変更が必要になるという短所はあるものの、アプリケーション的には SOCKS64 を意識せず、通常の通信手順を踏むだけで自動的に IPv4/IPv6 変換が行われる。Transport Relay Translator の方は追加ライブラリを必要としない。その代わり、IPv6 ノードが IPv4 ノードと通信する場合には、宛先アドレスとして IPv6 アドレスの一部に IPv4 アドレスが埋め込まれた特殊なアドレス空間を用いる。中継サーバーは、前述の特殊な IPv6 アドレス空間宛ての接続を終端し、IPv6 アドレス中に埋め込まれた IPv4 アドレスを抜き出す。その上で、抜き出された IPv4 アドレス宛に、同じデータを再送信する。特殊な IPv6 アドレスは、通常は手動で計算することはなく、DNS64[3] のような、専用の DNS サーバーを運用することで対応する。

IP 層でのヘッダ変換は、IPv4 における NAT 技術と同等の技術である。この技術は nat64[4] として標準化されている。この方式では、中継サーバーが通信を終端する必要がない。中継サーバーに辿り着いた IP パケットは、事前に定められたアドレス変換規則に則って始点アドレスと終点アドレスが書き換えられる。また同時に IPv4 と IPv6 の間のヘッダ変換も行われる。中継サーバーの機能がトランスポート層にあるのか、IP 層にあるのかが異なるだけで、本質的には二つ目の方法と技術的な差は存在しない。

これらの技術は、IPv6 でのみ構成されたネットワークに接続している IPv6 ノードがインターネット上の IPv4 サーバーに接続するときの問題を解決するために提案されたものである。運用的には、ひとつ、あるいは少数のグローバル IPv4 アドレスが中継サーバーに割り当てられており、多くの IPv6 ノードがそのアドレスを共有して IPv4 インターネットへの接続を確保する形態が想定されている。しかしながら、本論文で想定しているシステムは、IPv6 で構成されたサーバー群に、IPv4 ノードがアクセスするものであり、想定された運用形態が異なっている。IPv6 サーバーを IPv4 ノードに提供するため、グローバル IPv4 アドレスを共有することはできず、IPv6 サーバーアドレスとグローバル IPv4 は一対一で対応づけられることになる。既存の技術でも、本論文で前提としている仕組みを実現することは理論的に可能であるものの、前提とする運用形態が異なるため、実際に運用できる実装は存在しない。

本論文では、IPv6 のみによる管理を前提とし、IPv4/IPv6 変換ソフトウェアを利用したデュアルスタック IaaS サービスの設計を示し、それを実現する IPv4/IPv6 変換ソフトウェアの設計および実装、またその性能評価を行う。

3 システムおよびソフトウェアの設計と実装

本節では提案する IaaS システムの全体像と、IPv4/IPv6 変換ソフトウェアの設計を述べる。

3.1 IaaS システムの設計と実装

本提案で対象としているのは、仮想計算機資源を提供する IaaS システムである。これは、複数の物理計算機で構成された資源プールから、仮想的な計算機資源を切り出し、サービス運用の部品として利用する仕組みである。サービスによって異なるが、ひとつのサービスは一般的には複数の仮想計算機を用いて構築される。たとえば、ウェブのフロントエンド、ロードバランサー、データベースなどである。あるいは、分散ストレージのようなサービスの場合は、より多くの仮想計算機がフロントエンドの裏側に配置されることも考えられる。デュアルスタックサービスを提供する場合、すべての仮想計算機に IPv4 と IPv6 の両方を割り当てても構わないが、先に述べたように二重管理の問題が発生する。実際、サービスを利用するユーザーから見えるのは、フロントエンドに位置する一部の計算機だけであり、それらの計算機にデュアルスタックでアクセスできれば、それ以外の計算機がデュアルスタックである必然性はない。サービス構築側も、内部のプロトコルがひとつであれば、ネットワーク設計が容易になると同時に、その上で動作させるサービスの開発やテスト工数が削減できる。

また、本提案では IPv4 と IPv6 を一対一で対応させることを前提としている。IPv4 アドレスを複数のサーバーで共有しないため、その消費量については議論の余地があると思われる。しかしながら、サーバーとして運用する場合は、いずれにせよ IPv4 アドレスを占有する必要があり、これまで通りのサービス内容を維持するためには避けられない消費と言える。本提案では、アドレスの割り当てをフロントエンド計算機に制限することにより、無駄な消費を抑えることにしている。

IaaS 内部ネットワークで利用するプロトコルについては、IPv4、IPv6 のどちらでも構わない。今回の実装では将来的な IPv6 への移行を考慮して、IaaS 内部ネットワークは IPv6 ネットワークのみを使う構成としている。

図 1 に提案システムの概要を示す。フロントエンドとなるノードには、実際には IPv4 アドレ

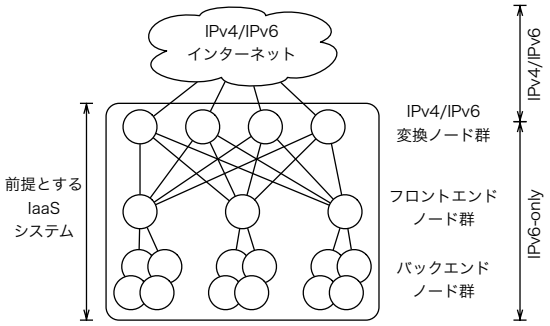

図 1: 提案システム概要図

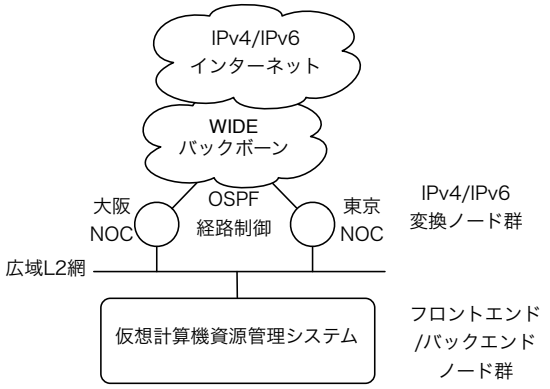

図 2: 提案システムの実構成ネットワーク図

スは設定されず、フロントエンドノードの IPv6 アドレスと、それに対応する IPv4 アドレスの情報が IPv4/IPv6 変換ノード群で共有される。アドレス変換は一対一で行われるので、各変換ノードが通信のセッション情報を保持する必要がなくなり、パケット単位で変換できる。そのため、変換ノードを複数配置することが容易となり、トラフィック増加により変換ノードの負荷が向上した場合でも、変換ノードを追加するだけで対応可能である。また、どこかの変換ノードに障害が発生した場合でも、そのノードを切り離す事で他のノード群でサービスを継続可能となる。図 2 は、我々が実際に構築運用しているネットワーク図である。

最下層に位置する仮想計算機資源管理システムは WIDE プロジェクトによって開発されている WIDE Cloud Controller (WCC)[5] を利用している。WCC は広域 L2 ネットワークを経由して東京と大阪間で接続されており、それぞれの拠点に変換ノードを配置している。変換ノードは OSPF 経路制御プロトコルを用いて、一対一対応された IPv4 アドレスの経路情報を WIDE プロジェクトが運用する L3 ネットワークに広報している。これにより、東京あるいは大阪のどちらかの変換サーバが消滅しても、適切に生き残ったサーバーにトラフィックが向かうようになっている。

3.2 変換ソフトウェアの設計と実装

IaaS システム内は IPv6 のみで運用されているため、サーバーにはすべての通信が IPv6 で行われているように見える。IPv4 クライアントからのパケットは一旦変換ノードで受信され、IPv6 に変換される。本ソフトウェアでの変換手順を図 3 に示す。

IPv6 サーバーに対応する IPv4 アドレスは変換テーブルに保持されており、クライアントから入力された IPv4 パケットの終点アドレスに応じて適切な IPv6 終点アドレスへと変換される。IPv6 パケットの始点アドレスは、元の IPv4 パケットの始点アドレス情報を保持する形で、IaaS 内で経路制御されている仮想プレフィックスの下位 32 ビットに埋め込まれる。IPv4 のアドレス空間は、IaaS システム内ではすべてこの仮想プレフィックスに含まれることになる。サーバーからの応答パケットは、上記と逆の手順で変換される。

実装は Linux や BSD で標準的に提供されている tun 仮想ネットワークデバイスを用いた。変換ノードは、IPv6 サーバーに対応付けされている IPv4 アドレスの経路を IPv4 インターネットに広告し、そのアドレス宛のパケットを一旦 tun ネットワークデバイスで受信する。受信されたパケットはユーザー空間でデバイスを開いている変換プログラムに読み込まれ、図 3 の手順に従って変換される。変換の結果生成された IPv6 パケットは、再び tun ネットワークデバイスへ書き出され、IPv6 サーバーへ配送される。逆方向も同様に、変換ノードが仮想プレフィックスの経路情報を IaaS システム内に広告し、すべての仮想プレフィックス宛のパケットを tun ネットワークデバイスで受信、変換し、元の IPv4 クライアントに再転送される。

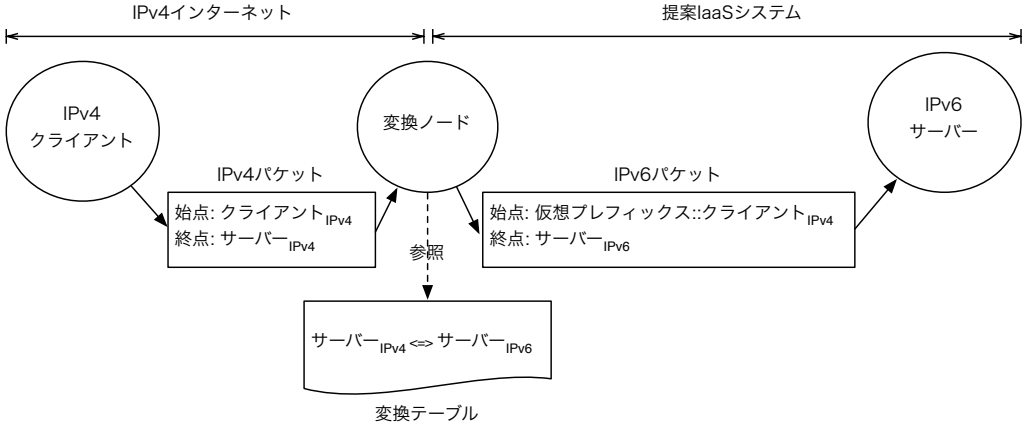

図 3: IPv4/IPv6 変換手順

最後に、本提案で用いる IPv4/IPv6 変換ソフトウェアの制限を述べる。本ソフトウェアは変換ノードでの状態を保持しない設計となっているため、変換のために状態保持が必要となる、フラグメントされた ICMP と ICMPv6 間の変換はサポートされない。また、現在のところ、ペイロード部に IPv4/IPv6 アドレスを含むパケットの変換もサポートしない。後者は実装上の制約であり、理論上は状態を保持しないまま変換可能である。

本ソフトウェアはオープンソースソフトウェア map646 として公開 [6] されており、自由に利用可能である。

4 性能計測

本節では map646 の性能を評価する。

4.1 評価環境

性能評価実験は次の環境で行った。図 3 に示した、IPv4 クライアント、変換ノード、IPv6 サーバで構成されるネットワークを実際に構築し、遅延時間の測定のために ping、クライアントサーバ間の帯域幅の測定のために iperf を用いて変換ノード上の IPv4/IPv6 変換ソフトウェアの性能を計測した。3 台の端末の詳細は表 1 の通りである。Network Interface の Realtek RTL-8169 は変換ノードの IaaS 側のインターフェイスにのみ使用した。それ以外は全て Intel 82573L を使用した。

表 1: 性能評価実験環境: 使用計算機

CPU	Core2 Duo 3.16GHz
Memory	4GB
OS	Linux 2.6.32
Network Interface	Intel 82573L Realtek RTL-8169

iperf では通信プロトコルとして TCP を使用した。比較対象として変換ソフトウェアを用いず変換ノードをルータとして扱い、クライアントと変換ノード間のネットワーク、変換ノードとサーバ間のネットワークを共に IPv4 ネットワークのみにした場合と IPv6 ネットワークのみにした場合でも計測を行った。それぞれの実験の構成を図 4 にまとめた。以降この図に従い map646 を使用した計測を構成 1、IPv4 ネットワークのみを使用した計測を構成 2、IPv6 ネットワークのみを使用した計測を構成 3 と呼ぶ。

ping、iperf の結果をそれぞれ図 5、図 6 に示す。計測は全て 5 回ずつ行い図にはその平均値を示した。

図 5 の横軸はそれぞれの構成での結果を示し、縦軸は遅延時間である。構成 1 の遅延時間は 0.172ms、構成 2 は 0.139ms、構成 3 は 0.14ms となった。

図 6 の横軸はそれぞれの構成での結果を示

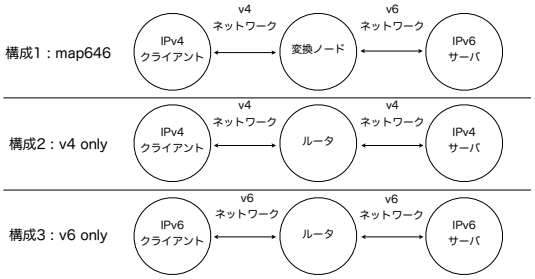

図 4: 実験構成

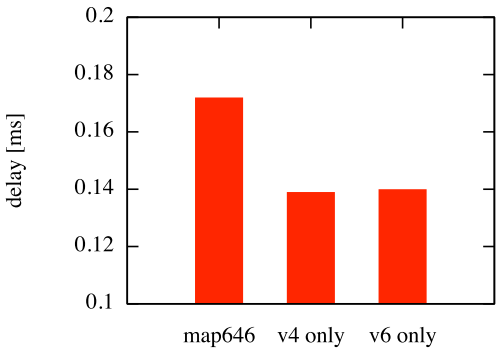

図 5: ping による遅延時間の測定

し、縦軸は帯域幅である。構成 1 の帯域幅は 572.4Mbps、構成 2 は 593.4Mbps、構成 3 は 584.2Mbps となった。

4.2 類似技術との性能比較

次に類似技術として nat64[4] を用いて、map646 との性能比較を行った。nat64 は IPv6 ネットワーク内にあるクライアントが変換ノードを通して IPv4 ネットワーク内にあるサーバにアクセスするための技術で、クライアントは IPv6 サービスプレフィックスである 64:ff9b::の下位 32 ビットにアクセスしたい IPv4 アドレスを埋め込み変換ノードへパケットを渡す。変換ノードでの IPv6 パケットから IPv4 パケットへの変換は、セッション情報の管理とポート変換によって行われるため、変換に必要な IPv4 アドレスは一つだけで済む。今回の比較実験では nat64 の実装として Ecdysis[7] を使用した。Ecdysis は nat64 のオープンソースの実装で、Fedora をベースにパッケージ化された LiveCD が配布されている。

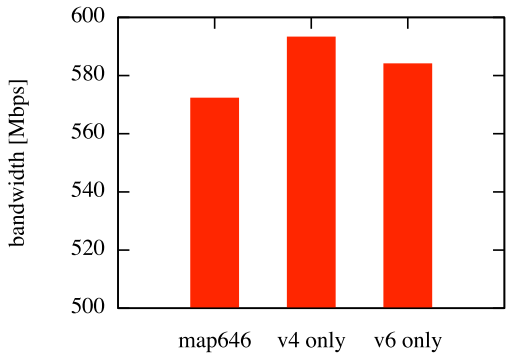

図 6: iperf による帯域幅の測定

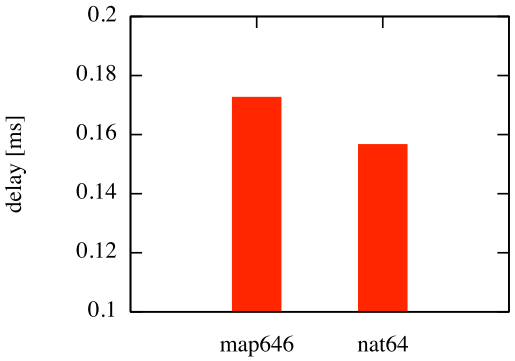

図 7: nat64 との比較: ping による遅延時間の測定

Fedora 内で nat64 はカーネルモジュールとして組み込まれている。

nat64 での想定利用環境に合わせ、IaaS システム内を IPv4 ネットワーク、IaaS システムの外を IPv6 ネットワークとして 4.1 章で行ったものと同様の計測を行った。ping、iperf の結果をそれぞれ図 7、図 8 に示す。

図 7 の横軸は map646 での変換を行った場合、nat64 を使用した場合の結果を示し、縦軸は遅延時間である。map646 を使用した際の遅延時間は 0.172ms、nat64 を使用した場合は 0.156ms となった。

図 8 の横軸は map646 での変換を行った場合、nat64 を使用した場合の結果を示し、縦軸は帯域幅である。map646 を使用した際の帯域幅は 582.4Mbps、nat64 を使用した場合は 706.4Mbps となった。

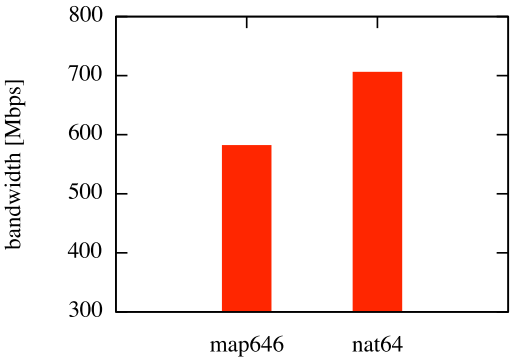

図 8: nat64 との比較: iperf による帯域幅の測定

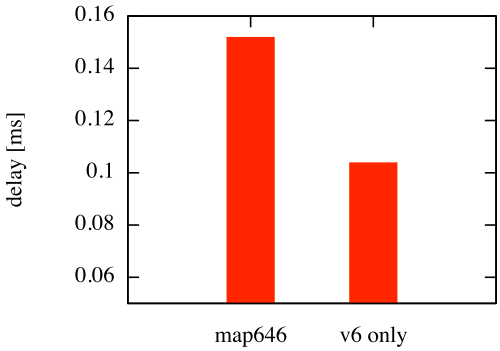

図 9: 実ネットワークでの性能計測: ping による遅延時間の測定

表 2: 実ネットワーク環境: 使用計算機

CPU	Xeon 2.33GHz * 2
Memory	4GB
OS	Linux 2.6.26
Network Interface	Broadcom NetXtreme II

4.3 実ネットワークでの性能計測

最後に 4.1 章で行ったものと同様の計測を WIDE プロジェクトが運用する WIDE Cloud 内で行った。比較対象として変換ソフトウェアを用いず IPv6 ネットワークのみを利用した場合での計測も行った。WIDE Cloud は実運用されている IaaS であるため、アドレスの付け替えが必要になる IPv4 での検証は行えなかった。ping と iperf の結果をそれぞれ図 9、図 10 に示す。図 9 の横軸は map646 での変換を行った場合、IPv6 ネットワークのみの場合を示し、縦軸は遅延時間である。map646 を使用した際の遅延時間は 0.152ms、IPv6 ネットワークのみを利用した場合は 0.104ms となった。図 10 の横軸は map646 での変換を行った場合、IPv6 ネットワークのみの場合を示し、縦軸は帯域幅である。map646 を使用した際の帯域幅は 617Mbps、IPv6 ネットワークのみを利用した場合は 927Mbps となった。WIDE Cloud 内で変換ソフトウェアが稼動しているサーバのスペックを表 2 に示す。

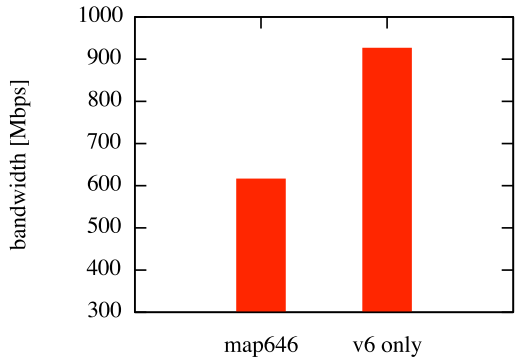

図 10: 実ネットワークでの性能計測: iperf による帯域幅の測定

4.4 考察

4.1 節と 4.3 節の遅延時間の計測より、map646 を使用する際の時間コストは 0.03-0.05ms とわかった。変換ソフトウェアではコンフィギュレーションファイルから変換ハッシュテーブルを作成し、tun インターフェイスに到着したパケットの宛先アドレスをキーとして変換ハッシュテーブルから変換先のアドレスを取得するように実装してある。今回コンフィギュレーションファイルに記述してある変換対は実験環境では実験に用いた 1 組、実ネットワークとして利用した WIDE Cloud では約 30 組であったためハッシュテーブルの検索には大した時間はかからず、得られた時間コストの大半は IP ヘッダの変換に費やされていると考えられる。ハッシュテーブルに格納する変換対が増えたときの時間コストへ

の影響の調査は今後の課題とする。

次に 4.1 節の実験では map646 を使用した際の帯域幅の減少は 4%に留まっているが、実ネットワークを用いた 4.3 節では帯域幅の減少が 34%にまで広がってしまっている。4.1 節と 4.3 節での IPv6 ネットワークのみを使用したときの結果を示す図 6 と図 10 を比較すると、4.3 節では 4.1 節の約 1.6 倍の帯域幅が得られている。この差は各環境で使用した PC の NIC の性能差と考えられ、4.1 節で map646 を使用した際の帯域幅の減少が小さかったのは NIC の性能が帯域幅のボトルネックとなっていたからと考えられる。

最後に 4.2 節の計測により、類似技術である nat64 に比べ map646 は帯域幅では 18%、遅延時間では 0.016ms パフォーマンスが劣ることがわかった。これは map646 がユーザ空間で動くプロセスとして実装されているのに対し、今回使用した nat64 は Linux のカーネルモジュールとして実装されてためと考えられる。

5　結論

仮想化技術の進歩により、より多くのサーバーを運用されるようになりつつある。今後 IPv4/IPv6 のデュアルスタックインターネットが普及していくことが予想され、サーバー側も両プロトコルに対応する必要があるものの、運用コストを考えると通信プロトコルはひとつに集約できた方が望ましい。本提案では、サービスバックエンドの構築を IPv6 のみのシングルスタックで実現し、利用者に公開するポイントのみを IPv4/IPv6 変換機構を用いてデュアルスタック化する IaaS システムを提案し、そのために必要な IPv4/IPv6 変換ソフトウェアの設計、実装、評価を行った。IPv4 アドレスと IPv6 サーバーの対応を一対一に限定することにより、変換ソフトウェアが保持すべき状態をなくし、負荷状況に応じて柔軟に変換ノードを増減でき、またひとつの変換ノードに障害が発生しても、自動的に他の変換ノードがサービスを継続できるスケーラブルかつ耐障害性の高いシステムが実現できた。変換ソフトウェアの実装は tun 仮想ネットワークデバイスを用いてユーザー空間プログラムで実現されている。カーネルモジュールとして実装されている変換ソフトウェアと比較した結果、34%の性能低下が認められたため、今後は高速化を目指す改良を進め、ネットワークの本来の性能を生かしきるような改良を進めていく。

6　今後の課題

性能をより向上されるために map646 をカーネルモジュールとして実装していく。またハッシュテーブルに格納する変換対が増えたときのパフォーマンスへの影響を調査していく。また今回は TCP での性能評価しか行わなかったが、UDP を使用した際の評価、ウェブサーバへのアクセスなど実アプリケーションを使用した際の評価も行っていく。さらに変換ノードを通過するパケットの統計的な情報を得たいという IaaS の管理面からの要求に応えるための機能を map646 上に実装し、さらに運用が容易な IaaS システムの実現を目指したい。

謝辞

本研究を進めるにあたり、東京大学の関谷勇司博士、および WIDE プロジェクトの WIDE クラウドワーキンググループの研究員の方々から多大なご協力およびご指導をいただきました。また、本 IaaS システムの利用者として多くの問題点を発見し、その解決に貢献してくださった WIDE プロジェクト研究員の方々に感謝いたします。

参考文献

[1] Hiroshi Kitamura. *SOCKS-based IPv6/IPv4 Gateway Mechanism*. IETF, April 2001. RFC3089.

[2] Jun ichiro itojun Hagino and Kazu Yamamoto. *An IPv6-to-IPv4 Transport Relay Translator*. IETF, June 2001. RFC3142.

[3] Marcelo Bagnulo, Andrew Sullivan, Philip Matthews, and Iljitsch van Beijnum. *DNS64: DNS Extensions for Network Address Translation from IPv6 Clients to IPv4 Servers*. IETF, April 2011. RFC6147.

[4] Marcelo Bagnulo, Philip Matthews, and Iljitsch van Beijnum. *Stateful NAT64: Network Address and Protocol Translation from IPv6 Clients to IPv4 Servers*. IETF, April 2011. RFC6146.

[5] WIDE project. WIDE Cloud Controller, August 2011. http://wcc.wide.ad.jp/.

[6] Keiichi Shima and Wataru Ishida. map646: Mapping between IPv6 and IPv4 and vice versa, August 2011. https://github.com/keiichishima/map646/.

[7] Viagénie. Ecdysis: open-source implementation of a NAT64 gateway, August 2011. http://ecdysis.viagenie.ca/.