第 7 部

IP Version 6

第 1 章

V6 分科会

V6 分科会は、(1) IPv6 環境を構築するためのソフトウェアを複数実装し相互通信性を検証すること、(2) 6bone という実験基盤を構築し運用によって問題点を発見し解決することを目標として 1996 年度を活動した。成果として複数の IPv6 の実装と 6bone が得られた。

本章では 1996 年度の具体的な活動内容と成果について報告する。まず 1.1節では、仕様を検討した結果や仕様をもとにした実装、および、その実装の検証について説明する。次に、IPv6 の普及のために取り組んだ啓蒙活動に関して 1.2節で述べる。WIDE プロジェクトでは多数の IPv6 コードを開発しているが、その 1 つである Hydrangea の実装について 1.3節で触れる。1.4節では、IPv6 の経路制御プロトコルの 1 つである RIPng を概説する。さらに、我々が提案する新しい経路 MTU 探索を方式について 1.5節で説明する。最後に 1.6節において、IPv4 から IPv6 への移行を支援する 3 種類のトランスレータを解説しその評価を示す。

1.1 仕様の検討、および実装とその検証

インターネットの研究では、標準仕様に従って実装すると同時に、他の実装と実際に通信して相互接続性を確認することが重要である。このため V6 分科会では、標準仕様をもとに参加者がそれぞれ独自に開発している実装の接続試験を実施してそれらの相互接続性を検証するとともに、実装過程や接続試験中に出たさまざまな問題について検討から実装へのフィードバックを繰り返すことにより、実装のレベルアップを図っている。以下では、本年度 V6 分科会で実施した仕様の検討、および実装とその検証について報告する。

1.1.1 仕様の検討

V6 分科会では、実装過程や接続試験中に出たさまざまな問題について、「検討」から「その結果を実装にフィードバック」という一連のサイクルを繰り返すことにより、実装のレベルアップを短期間のうちに効率よく図ってきた。以下に、本年度 V6 分科会で議論した主な内容を示す。

- 始点アドレス選択問題

- 送出インターフェイス選択問題
- 経路 MTU 探索問題
- IPv4 / IPv6 変換問題

以下では、これら上記の重要な問題のうちの始点アドレス選択問題について報告する。

始点アドレス選択問題

IPv6 と IPv4 の大きく異なる特徴の 1 つとして、IPv6 ではプレフィックス (サブネット) の異なる複数個の IP アドレスを 1 つのインターフェイスに設定できることが挙げられる。しかし、この機能によって新たに始点アドレス選択問題が発生した。この問題について考えるため、まずリンクローカルとグローバルの 2 つのアドレスを持つホスト 1 が同一ネットワークに接続しているホスト 2 と通信する場合について考えてみよう。この場合、ホスト 1 は始点アドレスとしてリンクローカル/ グローバルのいずれを用いてもよい。同一ネットワーク内の通信であるため始点アドレスとしてリンクローカル / グローバルのどちらのアドレスが使われていようと、ホスト 2 からホスト 1 へ応答を返すことが可能だからである。

しかし、ホスト 1 とホスト 2 が異なるネットワークに接続している場合、ホスト 1 は正しい始点アドレスを選択しないとホスト 2 と通信できない。まずホスト 1 が始点アドレスとしてリンクローカル・アドレスを用いたとする。この場合、始点アドレスがリンクローカル・アドレスであるパケットは中継経路上の途中のルータで捨てられてしまうので、ホスト 1 はホスト 2 と通信できない。仮にこのパケットがホスト 2 まで届いたとしても、ホスト 2 からホスト 1 へ応答を返すときの終点アドレスがリンクローカル・アドレスになるので、ルータを通過できない。次にホスト 1 が始点アドレスとしてグローバル・アドレスを用いたとする。この場合、終点アドレスにもホスト 2 のグローバル・アドレスを用いていれば、パケットはルータを通過してホスト 2 に届けられる。そしてホスト 2 からホスト 1 への応答も同様にルータを通過して届けられる。このようにホスト 1 が異なるネットワークに接続するホスト 2 とルータ経由で通信する場合、ホスト 1 が選択すべき正しい始点アドレスはグローバル・アドレスであるといえる。

以上を V6 分科会で議論した結果、「スコープをノードローカル、リンクローカル、グローバルに分類し、始点アドレスと終点アドレスのスコープは一致しなければならない」との結論に達し、以下をインターネット・ドラフトとして IETF に提案した。

- draft-yamamoto-wideipv6-comm-model-00.txt

1.1.2 実装とその検証

V6 分科会では、昨年度分科会を設立 (1995 年 8 月設立) して以降、参加者がそれぞれ独自に開発している実装を持ち寄って接続試験を繰り返し、相互接続性を検証してきた。その結果、昨年度は以下に示す基本的な通信機能を実装するとともに、その相互接続性を確認した。

昨年度 V6 分科会で実装した機能

- ヘッダ処理を含む基本的なパケットの送受信機能
- ホスト用の近隣探索機能
- ICMP 機能
- 転送機能
- TCP/UDP およびソケット

そこで本年度は次の段階として、広域分散ネットワークで IPv6 を運用するとき必要となる以下の機能を実装するとともに、接続試験を実施し相互接続性を検証した。

本年度 V6 分科会で実装した機能

- 経路制御プロトコル (RIPng)
- ルータ用の近隣探索機能
- 分割・再構成機能
- トンネル機能

本年度 V6 分科会で実施した相互接続試験

- 第 3 回 V6 分科会ワークショップ (1996 年 5 月 27 日〜30 日の 4 日間)
- WIDE 秋期合宿 (1996 年 9 月 9 日〜12 日の 4 日間)
- 第 4 回 V6 分科会ワークショップ (1996 年 10 月 16 日〜19 日の 4 日間)
- 第 5 回 V6 分科会ワークショップ (1997 年 1 月 23 日〜25 日の 3 日)
- WIDE 春期合宿 (1997 年 3 月 16 日〜19 日の 4 日間)

本年度 V6 分科会で実施した相互接続試験の参加者

- 奈良先端科学技術大学院大学、東京大学、慶應義塾大学、大阪大学、株式会社日立製作所、株式会社東芝、富士通株式会社、日本電気株式会社、ソニー株式会社　(順不同)

以下では、これら上記機能の接続試験のうちの RIPng 相互接続試験について報告する。

RIPng 相互接続試験

各参加者の PC ルータで route6d を起動した後、以下の 3 項目について検証した。

- 各ルータ間で経路情報を交換するとき、RIPng の正しいパケット書式を使っていること
- 交換した経路情報が互いの経路表に正しく反映されること
- 交換した経路情報にしたがって正しい経路にパケットが中継されること

その結果、RIPngが正常に動作していることを確認するとともに、相互接続性についても確認した。

1.1.3 IOL IPv6 通信試験への参加

ニューハンプシャ大学 IOL(InterOperability Lab.) では、1996 年 2 月以降、定期的に世界各国から開発者が集まり IPv6 接続試験が開催されている。本年度は第 2 回 (1996 年 6 月 17 日~21 日)、第 3 回 (1996 年 12 月 2 日~6 日) が実施され、V6 分科会からは 4 つの実装の代表者が参加した。第 2 回、第 3 回の参加組織を合わせて以下に示す。

- WIDE プロジェクト
  - 奈良先端科学技術大学院大学
  - 慶應義塾大学
  - 株式会社 日立製作所
  - 富士通株式会社
- Bay Networks, Inc.
- Cisco Systems, Inc.
- Digital Equipment Corp.
- FTP Software, Inc.
- Hewlett Packard / SICS (The Swedish Institute of Computer Science)
- IBM Corp.
- INRIA (Institut National de Recherche en Informatique et en Automatique)
- Mentat, Inc.
- 住友電気工業株式会社
- Sun Microsystems, Inc.

- Telebit Inc.
- Xerox / Palo Alto Research Center

V6 分科会の実装はいずれもホスト / ルータどちらとしても動作可能であり、IOL でも両方の試験を実施した。

IOL で実施された試験項目は、(1) 基本スペックの動作確認、(2) 参加組織間の相互接続試験の 2 つに分けられる。

(1) 基本スペックの動作確認

これは、IOL の試験機と被試験機を対向に接続して試験パケットを送受信し、被試験機が RFC、ドラフトを忠実に実装しているか調べる試験 (適合試験) である。試験機では IOL が開発した試験パケット生成/ 観測プログラムを動作させる。この試験プログラムはネットワーク層以下を試験対象としている。試験項目の概略を次に示す。

- IPv6 ヘッダの処理
  - 基本ヘッダ
  - オプションヘッダ
  - 未定義ヘッダ
- 近隣探索プロトコル
  - アドレス検索 (近隣要請 / 近隣通知)
    - 状態遷移
    - デフォルト・ルータ・リストの取り扱い
  - ルータ要請 / ルータ通知
    - ルータ / プレフィックスの発見 (ホストの実装)
    - ルータ要請パケットの取り扱い (ルータの実装)
  - 向け直し

V6 分科会の実装はこの時点では ICMP の一部機能が未実装であったりタイマの動作不良などがあったが、おおむね安定した動作をした。

(2) 参加組織間の相互接続試験

これは参加者相互間で IPv6 に対応したアプリケーションを動作させ、相互に通信可能か調べる試験 (相互接続性試験) である。試験に用いたアプリケーションは、ping6、telnet6/telnet6d、route6d(RIPng) である。ping6、telnet6 の試験は次の 3 つの場合についてそれぞれ実施した。

- 同一リンク上に接続したホスト同士間の通信

- ルータを介したホスト同士間の通信
- IPv6 over IPv4 トンネルを介したホスト同士間の通信

V6 分科会の実装は、ping6、telnet6 についてはすべての場合について正常に動作し、他の参加者との間で正常に通信できた。また RIPng については経路情報を正しく交換できることを確認した。

1.1.4 6bone の構築、運用

V6 分科会では IPv6 広域実験ネットワーク 6bone を構築、1996 年 7 月 9 日に運用を開始した。IPv4 と分離し幹線部に専用線を利用した IPv6 専用実験ネットワークとしては世界で初めてである。6bone 構築の狙いは、以下の 2 つである。

- 実際に広域ネットワークを運用して、相互通信試験を実施し、実装 / 仕様の問題点を洗い出す
- IPv6 に関連する実験のための基盤を作る

ほぼ同時期に IETF の 6bone-WG(当時は BOF) において世界的な 6bone を構築したが、WIDE プロジェクトはアジア地域の取りまとめとして構築段階から参加し、世界的な 6bone の運用に参加している。なお、WIDE 6bone から世界の 6bone への接続には IPv6 over IPv4 トンネル技術を利用している。

運用開始当初は接続しているノードも少なく経路も少なかったので静的な経路制御が可能だったが、規模の拡大とともに経路も増加したため 1996 年 12 月~1997 年 1 月にかけて IPv6 用経路制御デーモン route6d を動作させ動的な経路制御へ移行した。

現在の WIDE 6bone のネットワーク構成を図 1.1に示す。

1.2 啓蒙活動

V6 分科会では、IPv6 の普及のためにさまざまな啓蒙活動に取り組んでいる。この節では、これまでに実施した啓蒙活動について述べる。

1.2.1 NetWorld+Interop '96 東京 IPv6 SSD

1997 年 7 月 24 日~26 日幕張メッセで開かれた NetWorld+Interop '96 (以下 N+I '96) 東京で WIDE プロジェクト IPv6 分科会が中心になり、SSD(Solution Showcase Demonstoration) のブースに IPv6 の展示を行った。出展したのは WIDE IPv6 分科会に参加している組織および IPv6 の実装を持っている組織である。IPv6 SSD の参加組織を以下に示す。

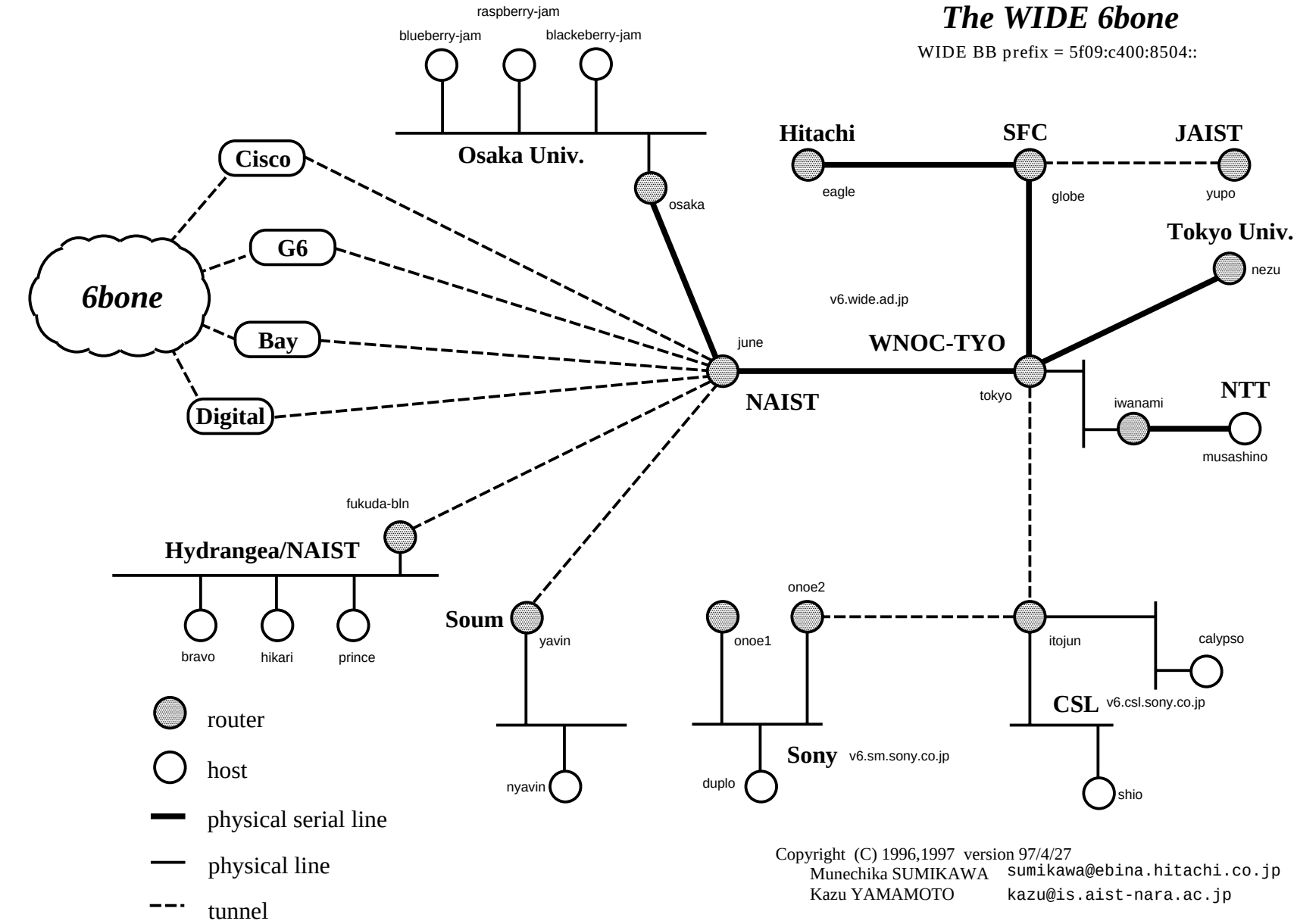

図 1.1: WIDE 6bone のネットワーク構成

Copyright (C) 1996,1997 version 97/4/27
Munechika SUMIKAWA sumikawa@ebina.hitachi.co.jp
Kazu YAMAMOTO kazu@is.aist-nara.ac.jp

- 慶應義塾大学
- 奈良先端科学技術大学院大学
- 株式会社 日立製作所
- 株式会社ソニーコンピュータサイエンス研究所
- 住友電気工業株式会社
- 富士通株式会社
- 日本ディジタルイクイップメント株式会社

IPv6 SSD の目的は、

- 一般への IPv6 の啓蒙活動
- 国内の IPv6 開発者の交流

であり、接続実験が目的ではなかった。

デモンストレーションとしては、以下を展示した。

- 6bone への接続
- 自動設定 (plug & play)
- IPv4 と IPv6 の相互接続

また、これ以外に SSD ブース脇のプレゼンテーション・コーナーにおいて、IPv6 について説明した。

これらの N+I '96 でのデモンストレーション活動を通して、以下の結果を得られた。

- N+I '96 の総入場者数 81583 人
- 配った IPv6 SSD のパンフ 2500 部
- 配ったトポロジー図 600 枚
- プレスの取材
  - 日経コミュニケーション
  - Internet User
  - DOORS
  - DOS/V MAGAZINE
  - Interface
  - OpenNetwork

1.2.2　講演

IPv6 の啓蒙活動の一貫として、以下の催しにて講演を行なった。

- 1996 年 7 月 N+I '96 東京
- 1996 年 9 月 日本サン・ユーザ・グループ・セミナー
- 1996 年 10 月 jus 秋のシンポジウム
- 1996 年 12 月 IP Meeting '96 チュートリアル
- 1996 年 3 月 Network Users '97 チュートリアル

講演では次のような内容を話した。

1. IPv6 の仕様
2. IPv6 分科会の活動
3. 世界の動向

1.2.3　雑誌への執筆

以下の雑誌に執筆し IPv6 を紹介した。

- UNIX MAGAZINE 1996 年 8 月号「転ばぬ先のセキュリティ – IPv6」
- BIT 1996 年 10 月号「NetWorld+Interop '96 の報告」
- UNIX MAGAZINE 1997 年 1、2 月号「Soraris で IPv6」
- コンピュータ&ネットワーク LAN 1996 年 9 月号「WIDE プロジェクト 次世代インターネット・プロトコル「IPv6」による実験ネットワーク「6bone」が稼動」

1.2.4　IPv6 本の翻訳

IPv6 分科会では IPv6 を広めるためには、IPv6 の心を正しく解説してある日本語の文献が必要であることを感じ、IPv6 の実装の際に参考にしていた Christian Huitema 著「IPv6 — The New Internet Protocol」を翻訳することにした。この本の以下の内容が詳しく書かれている。

- 歴史的背景

- 仕様
- これからの動向

この本の翻訳にあたり、正しく専門書を翻訳するには何をすべきかを考えた。そこで、以下を試みた。

- 内容をよく理解した者が翻訳する
- TCP/IP および今後、数十年使われる IPv6 の用語を整理 / 統一する。
- 最低でも 4 回は別の人が校正をする
- 内容が新鮮な内に出版するために、IPv6 分科会の人間が協調して翻訳する
- 標準化が進んでいる最中なので、原著の内容が古くなっている所は最新の情報に書き換える

9 月から作業にかかった翻訳作業の結果、「IPv6 — 次世代インターネット・プロトコル」は 1997 年 1 月 6 日付で、実際には 1996 年 12 月末に出版された。

1.3 Hydrangea の実装

V6 分科会では、それぞれの参加組織が独立に実装を進めた結果以下のように複数の成果を得られた。

Hydrangea 奈良先端科学技術大学院大学において BSD/OS 2.1、3.0、および、FreeBSD 2.2.1 のカーネルを拡張する形で実装された IPv6 のコード。定期的にリリースされている。

慶應義塾大学 慶應義塾大学において BSD/OS 2.1 にのカーネルを拡張する形で実装された IPv6 のコード。BSD/OS 3.0 に移植中。

日立製作所 日立製作所において BSD/OS 2.1 にのカーネルを拡張する形で実装された IPv6 のコード。

富士通 富士通において FreeBSD 2.1 にのカーネルを拡張する形で実装された IPv6 のコード。

東芝 東芝において BSD/OS 2.1 にのカーネルを拡張する形で実装された IPv6 のコード。

v6d 尾上 淳によって、ユーザ空間に実現された IPv6 のコード。

NEC NEC によって、ユーザ空間に実現された IPv6 のコード。

以下では、WIDE 内に定期的にリリースされている Hydrangea の実装について解説する。

1.3.1　プラグ&プレイ

Hydrangea はカーネルのブート中、および、PCMCIA のような抜き差し可能なカードの挿入時に、カーネル内でネットワーク・インターフェイス (以下 IF) に対し、リンクローカル・アドレスを割り当てる。Ethernet や FDDI では MAC アドレス、また対抗ネットワークでは MD5 を使った乱数を `fe80` と連結し、リンクローカル・アドレスを生成する。ただし、現在は IF の選択のために、リンクローカル・アドレスには IF 番号が組み込まれる。たとえば、MAC アドレスが 0:0:f8:1:63:17 で、IF 番号が 3 の Ethernet には、`fe80:3::f801:6317` が割り当てられる。また、各 IF において、リンクローカル・スコープの全ノード・マルチキャスト・アドレスと要請マルチキャスト・アドレスに参加する。

ループバック IF は、`fe80`、IF 番号、および、`1` を連結したリンクローカル・アドレスが割り当てられる。加えて、ループバック・アドレス (`::1`) も与えられる。さらに、ループバック IF では、ノードローカル・スコープ、および、リンクローカル・スコープの全ノード・マルチキャスト・アドレスと要請マルチキャスト・アドレスに参加する。

カーネル内における上記の IF の初期化が終った時点で、リンク内での通信が可能になる。ホストが他のリンクのノードと通信するためには、グローバル・アドレスとデフォルト経路を取得する必要がある。IPv6 ではこれらの情報を獲得するために、近隣探索プロトコル にルータ要請メッセージとルータ応答メッセージという枠組を設けてる。Hydrangea では、ルータ要請メッセージの送信をユーザ空間のコマンド (rtsol) によって実現した。よって一般には、制御がユーザ空間に移された際になんらかの方法で rtsol を起動しルータ要請メッセージを発行する。応答であるルータ応答メッセージはカーネルが受け取り、IF にグローバル・アドレスを割り当てると共に、デフォルト経路を経路表に登録する。

ルータ要請メッセージの発行を、IF の初期化後にカーネルから行なう方法も考えられる。しかし、BSD/OS では IF のハードウェアが完全に初期化されたことを知る方法がないため、初期化の修了を確実に待ってルータ要請メッセージを発行することができない。実際にこの方法を実装してみたが、コードには間違いがないと思われるのに、ルータ要請メッセージが発行されなかったり、発行されても書式が間違っていたりした。ユーザ空間に処理が移ったときは、ハードウェアの初期化から十分に時間が経っているため、確実に IF からパケットを送信できる。また、ルータやホストで挙動を変えるなどといった柔軟な処理を実現するには、IF 管理デーモンを実装する必要があると考えた。そこで、ルータ要請メッセージは、ユーザ空間から発行することに決定した。

1.3.2　スコープと始点選択

Hydrangea では、1.1.1節で述べたスコープ指向の始点選択を実現している。すなわち、終点が `::1` か `ff01::1` なら、始点に `::1` を選ぶ。終点がリンクローカル・アドレス、あるいは、リンクローカル・スコープのマルチキャストであれば、始点アドレスに出力 IF のリンクローカル・アドレスを選択する。それ以外の終点に対しては、出力 IF のグローバ

ル・アドレスから終点と最長合致するグローバル・アドレスが選ばれる。この機能のおかげで、マルチホームのホストが初歩的なポリシー経路制御を実現できる。ただし、コネクションが確立された TCP にはこの規則は適応されない。

1.3.3 IPv4 のロジックからの遊離

多くの IPv6 プログラマがそうであると思うが、我々は IPv6 のコードを書く作業を BSD の IPv4 のコードを複製し IPv6 へ適応するという手順で進めてきた。しかしながら、IPv6 は IPv4 と異なる機能を提供しているため、IPv4 のロジックとはそぐわない面があることに気づいた。

端的な例としては ip_forward() が挙げられる。BSD を搭載した IPv4 ルータでは、入力された IPv4 パケットは ipintr() に届く。ipintr() は、パケットの終点が自分宛でない場合、転送のために ip_forward() を呼び出す。IPv4 ではルータにおいて、必要ならば転送時にパケットを分割する。パケット分割のコードを省略するために、ip_forward() はパケット分割機能のある ip_output() を呼び出す。最終的に ip_output() は、出力 IF の出力関数である if_output() にパケットを渡す。

しかしながら、IPv6 ではパケットを分割するのは始点ノードのみであって、ルータではパケットを分割しない。そこで、Hydrangea では ip6_forward() が直接 if_output() を呼び出すようにしている。もし、MTU の関係でパケットが転送できない場合は、パケットを破棄して ICMP パケット過大メッセージを始点に返す。

1.3.4 拡張ヘッダの保存

ICMPv6 は、もともとのパケットを無変更のまま可能な限り長く (最大 576 バイト) 格納することが仕様で定められている。たとえば UDP のポート到達不可能メッセージは、少なくとも IP6 ヘッダと UDP ヘッダ、および、その間にある拡張ヘッダをオリジナルのまま格納する必要がある。IPv4 のネットワーク・コードは、随時ネットワーク・バイト・オーダをホストの表現に直すなど IPv4 パケットに変更を加えている箇所が多い。しかし、Hydrangea の TCP を除く IPv6 ネットワーク・コードでは、ICMPv6 を考慮してパケットに変更を加えないよう留意した。

ユーザ空間に対し拡張ヘッダの内容を通知しなければならない場合があるので、ネットワーク・コードでの処理中に拡張ヘッダを削ってはならない。よって、tcp6_input()、udp6_input()、および、icmp6_input() では IPv6 ヘッダとトランスポート・ヘッダが連続していると仮定できなくなった。そこで、Hydrangea では連続を仮定している in_cksum() の代わりに、不連続にも対応できる in6_cksum() を実現した。in6_cksum() は内部で仮想ヘッダを作成するで、呼び出し側が仮想ヘッダを作る必要はなく、コードの簡略化にもなった。

1.3.5　mbuf の改良

分割不可能なヘッダ部分が最小 MTU である 576 に収まらないパケットは分割できないため、このようなパケットを使った通信は保証されていないと考えてよいと思われる。ほぼすべての場合においてトランスポート・ヘッダまでが 576 以下に収まっているであろう。よって、パケットの先頭 576 バイトが連続した領域に置かれていると、ヘッダの処理関数にとって都合がよい。

4.4BSD では、入力パケットの大きさに応じて以下のような mbuf が作成される。

(1) 1 〜 100 — 内部 mbuf が 1 つ

(2) 101 〜 208 — 内部 mbuf が 2 つ

(3) 208 〜 2048 — 外部 mbuf が 1 つ

(4) 2049 以上 — 外部 mbuf が 2 つ以上

先頭 576 バイトが連続しているかに注目して上記の分類を検討すると、(2) の場合のみが不連続である。つまり、4.4BSD の枠組では、101 〜 208 バイトのパケット処理する可能性があるため、各ネットワーク関数は `m_pullup()` を呼んで対象領域を連続にする必要があった。しかし、4.4BSD の枠組では 101 バイト以上を `m_pullup()` を使って連続にすることは不可能である。

(2) の場合が存在する理由は、メモリーが高価であった昔に仕様効率を上げるための措置である。メモリーが安価となった現在では、そのまま利用する必然性はない。そこで、Hydrangea では、各ドライバを修正し以下のような mbuf を構成する方法を採った。

- 1 〜 100 — 内部 mbuf が 1 つ
- 101 〜 2048 — 外部 mbuf が 1 つ
- 2049 以上 — 外部 mbuf が 2 つ以上

1.3.6　IP6 の再構成

4.4BSD では、mbuf のデータ部分から mbuf へのポインターへの変換 (`dtom()`) は、内部 mbuf でのみ成功する。IPv4 では断片を再構成する際に、`dtom()` を利用しているので、`m_pullup()` を呼び出し外部 mbuf を内部 mbuf へ変換しなければならない。よって、IPv4 の再構成では、先頭 100 バイト以下の領域しか連続を保証できない。

一方 TCP の再構成では、mbuf のデータ部に mbuf へのポインターを保存して、`m_pullup()` を回避している。

上記のように Hydrangea では、パケットの先頭 576 バイトを連続にする必要がある。そこで、IPv6 の再構成では、再構成キューに入ったパケットの IPv6 ヘッダ中に mbuf へのポインターを保存し、`m_pullup()` を回避した。

1.3.7 Daisy と GIF

IPv4 から IPv6 への移行には、IPv6 in IPv4 トンネルが必須である。逆に、移行後期には IPv4 in IPv6 トンネルが必要になる。さらに、さまざまなサービスを考えると IPv4 in IPv4 などにも適用できる汎用的なトンネル技術が不可欠である。Hydrangea では、汎用的なトンネル技術として入力部に Daisy、出力部に GIF(Generic InterFace) を実装した。

Daisy とは、daisy-chain になっている (IPv4 ヘッダも含む) IPv6 の拡張ヘッダを関数呼び出しによって処理していく枠組である。Daisy の特徴を以下に示す。

- 各拡張ヘッダを関数呼び出しによって処理する。
- ネットワーク層のヘッダも関数呼び出しによって処理する。このおかげで、入力キューから呼び出される割り込み関数 (`ip{,6}intr()`) とヘッダ処理関数 (`ip{,6}_input()`) を用意している。このため、一連のヘッダ解析がキューで遮断されることはない。
- 関数呼び出しで実現しているため、さまざまなデータを保存し他の関数へ受け渡せる。よって、入力 IF を特定できるため、tcpdump などでパケットを観察することも可能である。
- ネットワーク層のヘッダが、それ移行のヘッダのセマンティクスを決定すると定義している。明らかなセマンティクス違反、たえば IP ヘッダの直後に ICMPv6 ヘッダが続く場合などは、ICMP{,v6} エラーを返す。

GIF とは、トンネルを IF として実現する機構である。トンネルの設定は、gif0 などの仮想 IF に対して行なう。以下に GIF の例を示す。

```
# gifconfig gif0
gif0: gif flags=8111<UP,POINTOPOINT,PROMISC,MULTICAST>
        inet 10.0.0.2 --> 10.0.0.1 netmask 255.0.0.0
        inet6 5f09:c400:a3dd:100::2 --> 5f09:c400:a3dd:100::1 prefixlen 80
        physical address inet 163.221.11.21 --> 163.221.202.1
```

この例では、ユーザに対してあたかも IPv4 と IPv6 の対抗ネットワークが存在するように見えてる。IPv4 はローカルが `10.0.0.2`、リモートが `10.0.0.1` である。IPv6 はローカルが `5f09:c400:a3dd:100::2`、リモートが `5f09:c400:a3dd:100::1` である。この gif0 を通して出力されるパケットは、始点が `163.221.11.21`、終点が `163.221.202.1` の IPv4 パケットに格納されて送り出される。

1.4　RIPng の概要

RIPng は IPv6 の経路情報を交換するために、IPv4 用の RIP をもとに設計されたプロトコルである。IPv4 上の RIP version 2 からの大まかな変更部分を以下に挙げる。

ヘッダ形式の簡略化 — 経路エントリ中のネットワーク部の長さを示すフィールドがマスクからプレフィックス・ビット長に変更された。さらに、アドレス・ファミリと次中継点のフィールドが削除された。

次中継点フィールドの独立 — 経路エントリから削除された次中継点フィールドは、次中継点エントリとして独立した経路エントリとなった。RIPng パケット中に次中継点エントリがある場合、その後ろの各経路エントリにある経路の次中継点は次中継点エントリに示されるルータとなる。

送信間隔 — 定期送信の間隔が 15 秒〜45 秒ごとと幅を持つようになった。

1.4.1　RIPng の通信

RIPng パケットの種類は、RIP と同様に要求パケットと応答パケットとがある。応答パケットは送信するルータの経路情報を含んでおり、要求パケットへの回答や定期アナウンスなど、経路情報の伝達に用いられる。要求パケットは、他ルータへの応答パケットの要求に用いられる。これらのパケットの送受信には、マルチキャスト・グループ `ff02::9`、UDP ポート 521 が用いられる。

1.4.2　定期送信

各ルータは 30 ± 15 秒ごとにネットワークに応答パケットを送信する。このときの応答パケットの終点アドレスは ff02::9、始点アドレスは始点 IF のリンクローカル・アドレスであり、終点 UDP ポートと始点 UDP ポートはともに 521 である。またパケットがルータを経由して他のネットワークに転送されないように、送信時の中継限界数は 255 に定められている。

応答パケット生成時は、ルータは自分が持つ各経路からスプリット・ホライズンによって適切な経路を選別し、メトリックを増やして応答パケットに入れる。リンクローカル・アドレスが応答パケットに含まれることはない。応答パケットの長さは、MTU を超えてはならない。

応答パケットに含まれる経路の一部の次中継点を RIPng の始点ルータではなく他のルータにしたい場合、それらの経路の前に次中継点エントリを置く。次中継点エントリには以下に続く経路の次中継点となるルータのアドレスを指定する。ここで指定された次中継点は、次の次中継点エントリまで有効となる。次中継点として `0::0` を指定した場合、応答パケットの始点が次中継点となる。

要求パケットとそれに対する応答パケット

要求パケットは、RIPng ルータの IF がアップしたときに送信される。パケットの終点アドレスは ff02::9、始点アドレスは始点 IF のリンクローカル・アドレスであり、終点 UDP ポート、始点 UDP ポートはともに 521 である。

要求パケットを受け取った各ルータは、自分の持つ経路情報から定期送信時と同じ手順で応答パケットを生成して始点に送り返す。

上に述べた以外にも、経路情報の管理などの目的でルータ以外から要求パケットが送信されることがある。このときの終点は対象ルータのユニキャスト・アドレス、終点 UDP ポートは 521 で、始点 UDP ポートは 521 以外の番号が使われる。

このときに送られる要求パケットには、上で使われるのと同じ全経路要求と特定経路要求がある。全経路要求の場合には、定期送信と同じ手順で応答パケットを生成して始点に送り返す。特定経路要求の場合には、要求内の各経路について自分が持っているメトリックを要求パケット内に埋め込み、そのまま送り返す。もし指定された経路を知らない場合には、メトリックは無限大 (16) とする。

トリガード・アップデート

ルータのインターフェイスのアップ/ダウンや、他ルータからのトリガード・アップデートによってルータの経路情報に変化があった場合、ルータはすみやかに応答パケットを送信する。この動作をトリガード・アップデートと呼ぶ。

トリガード・アップデート時の応答パケットには、変化した経路情報のみが含まれる。到達不能になった経路はメトリックが無限大 (16) として応答パケットに入れる。また定期送信と同様に、スプリット・ホライズンも適用される。

パケットの書式

RIPng パケットのヘッダを図 1.2 に示す。コマンドが 1 の場合は要求パケット、2 の場合は応答パケットである。バージョンは現在は 1 である。

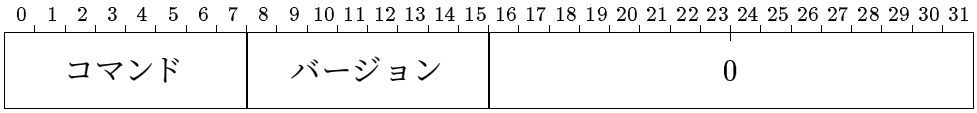

図 1.2: RIPng ヘッダ

経路エントリを図 1.3 に示す。これは要求パケットと応答パケットの場合で意味が異なる。

応答パケットの場合 各経路エントリが 1 つの経路に対応する。プレフィックスとプレフィックス長で経路を表わし、メトリックでその経路までの距離を表わす。

メトリックが `0xff` の場合、このエントリは次中継点エントリである。タグとプレフィックス長は 0 であり、プレフィックス部が次中継点アドレスとなる。

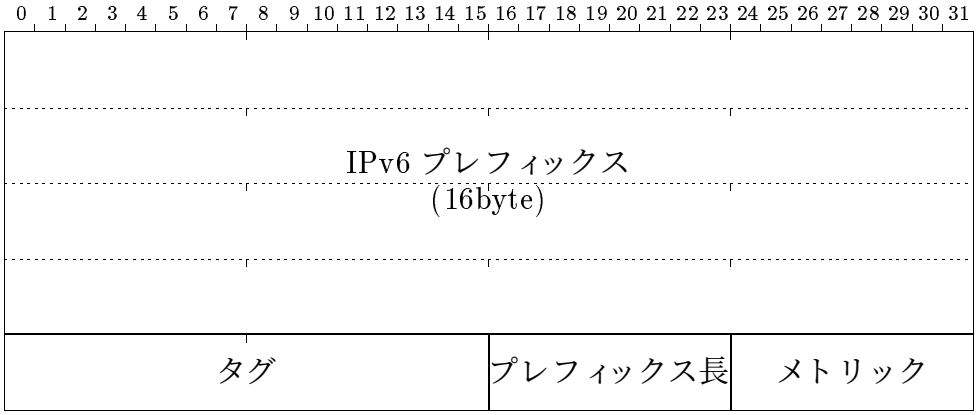

図 1.3: RIPng 経路エントリ

要求パケットの場合 通常用いられる全経路要求では、1 つの RIPng パケットに経路エントリを 1 つだけ持つ。このときの経路エントリは、プレフィックスとプレフィックス長が 0 でメトリックが無限大 (16) である。

またモニタリングなどで用いられる特定経路要求では、各エントリのプレフィックスとプレフィックス長が、メトリック要求に指定する各経路を表わす。

1.5　経路 MTU 探索

経路 MTU は、インターネットの任意の経路上で送信できる最大データ転送長であり、データ転送前にあらかじめ探索しておくことによって効率のよい通信が可能となる。しかし、IPv4 では次の理由から経路 MTU の探索機能はほとんど利用されなかった。

(1) 経路上のルータはパケットが次に出力するリンクの MTU より大きい場合は分割して転送する

(2) 正確な経路 MTU はインターネット上のすべてのホストやルータの実装を変更しなければ効率よく求まらない

(1) の経路上でのデータの分割は、任意の長さのデータが終点ホストに到達することを意味する。つまり、経路 MTU を探索しなくても終点ホストとの通信は可能であった。しかし、経路上でのデータの分割はパケットの棄却率を増加させ通信の効率を低下させることはよく知られている。

(2) は IPv4 が経路 MTU の探索機能を考慮していなかったために生じた問題である。つまり、IPv4 は、経路上のルータが始点ホストや終点ホストへ経路 MTU に関するヒントを

送信するための処理機構を実装していなかった。よって、経路 MTU の探索は始点ホストのみで探索する必要があり、正確な経路 MTU の探索には時間がかかった。経路 MTU の必要性が議論され始めてから、経路上のルータが ICMP エラーを利用して経路 MTU に関する情報を始点ホストや終点ホストに提供する仕様も定義されたが、インターネット上のすべてのルータに新しい機能を実装することは不可能であった。

これらの問題点に対して、IPv6 は以下のような解決策を選択している。

(1) 経路上のルータはリンク MTU より長いデータを破棄する

(2) ルータはデータを破棄する直前に ICMP パケット過大メッセージを始点ホストに送信する。

まず、IPv6 は通信効率の向上を理由に、経路上でのデータの分割を禁止した。始点ホストは経路 MTU より長いデータを安易に送信できなくなったので、IPv6 での 経路 MTU の探索の必要性が増した。また、(2) の ICMP パケット過大メッセージには転送するリンク MTU の値が含まれており、始点ホストはこの値を経路 MTU の探索の手がかりにできる。よって、(2) の機能を利用すればインターネット上のすべての計算機に経路 MTU のための機能を追加して実装する必要がなくなる。

以上から、実際に IPv6 では経路 MTU の探索機能の実装が推奨されている。ただ、IPv4 において経路 MTU の探索はあまり利用されなかったため、効率のよい探索方法は研究されていないと考えられる。よって、現在正確に把握されていないと考えられる経路 MTU の現状を調査し、その結果をもとに新しい探索方式である予測方式を提案した。

1.5.1 予測方式

予測方式は以下の 2 つの方針で探索する方式である。

(1) 始点ホストは、管理者が把握可能なネットワーク内の経路 (以下、既知経路と呼ぶ) では常に最適な経路 MTU で通信できるように、既存の探索方式である受動方式を利用する。

(2) 始点ホストは、未知のネットワーク内の経路 (以下、未知経路と呼ぶ) では探索時間の短い探索方式である予測探索方式を利用する。

予測方式は、より効率のよい経路 MTU の探索を行うために 2 つの探索方式を利用する。たとえば、既知経路ではビデオ・オン・デマンドなどのような高速で経路 MTU の大きいネットワークを構成している場合がある。このようなネットワークではできるだけデータの送信量を多くしたいので、最適な経路 MTU を求める探索方式のほうが有効である。よって、RFC1191、RFC1981 に記述されている既存の探索方式である受動方式を利用する。受動方式は、始点ホストのリンク MTU を経路 MTU の初期値 (以後、初期 MTU と呼ぶ) とし

て通信を開始する。そして、経路上のルータはデータを転送できないときにICMP パケット過大メッセージを始点ホストに送信する。ただ、IPv4 ではデータを分割しないようにIP ヘッダに分割禁止ビットを設定しておく必要がある。ICMP 過大メッセージを受信した始点ホストは MTU の値を小さくして、データを再送する。

(2) の未知経路では、受動方式の探索効率が悪くなる可能性がある。つまり、受動方式は常に最適な経路 MTU を求めようとするので、探索自体に時間がかかり結果的に通信効率が低下することがある。そこで、未知経路では予測探索方式[1]を利用する。予測探索方式は、あらかじめ経路 MTU を予測した MTU で探索を行う方式で、最適な経路 MTU は求まらないかもしれないが、探索時間の短縮が望める。

1.5.2　予測探索方式

予測探索方式は次のアルゴリズムで動作する。

(1) 始点ホストは、ブート時にルータから経路 MTU の初期値を獲得する

(2) 始点ホストは経路にデータを送信するとき、経路 MTU 表を調べる

- 経路 MTU 表に目的の経路の経路 MTU 値があればその MTU でデータを送信
- 経路 MTU 表に目的の経路の経路 MTU 値がなければ初期 MTU でデータを送信

(3) ルータはデータを転送できないときに、ICMP パケット過大メッセージを始点ホストに送信する

(4) ICMP パケット過大メッセージを受信した始点ホストは、経路 MTU 表の値を ICMP メッセージ内に記述されている値に更新して、より短い MTU 長でデータを送信する

まず、予測探索方式は初期 MTU の値を手に入れるために IPv6 のルータ要請とルータ通知を利用する。よって、管理者は管理しているすべてのホストに初期 MTU を設定しなくともよい。また、初期 MTU の値はできるだけ多くの経路の経路 MTU となるように設定する。

経路 MTU 表は経路ごとの経路 MTU の値を表にしたもので、各ホストが保持している。経路は通常終点アドレスで特定される。ただ、将来的に IPv6 ではフロー・ラベルが経路を特定する要素として追加されるかもしれない。経路 MTU 表の MTU の値には生存時間が決められており、一定時間経過するとその値は無効になる。

次に予測探索方式の利点と欠点を述べる。まず、利点を以下に記す。

- 探索時間が短くなる
- 初期 MTU を柔軟に設定できる

[1] 予測方式は受動方式と予測**探索**方式を組み合わせた方式であることに注意

予測探索方式は予測した経路 MTU を初期 MTU として利用する。よって、この値をうまく選択すれば、始点ホストは ICMP パケット過大メッセージを受信する回数が受動方式に比べ少なくなる。ルータが始点ホストへ ICMP パケット過大メッセージをを送信するために必要な時間は、経路 MTU を探索する時間に比例するので、ICMP パケット過大メッセージの受信回数の減少は探索時間の短縮につながる。

また、予測探索方式の有効性は、経路 MTU の分布の調査結果からも分かる。図 1.4 は奈良先端科学技術大学院大学の FDDI に継っている計算機を始点として、終点ホストに海外のホスト約 5000 台と国内のホスト約 7000 台への経路を調査した結果をまとめである。

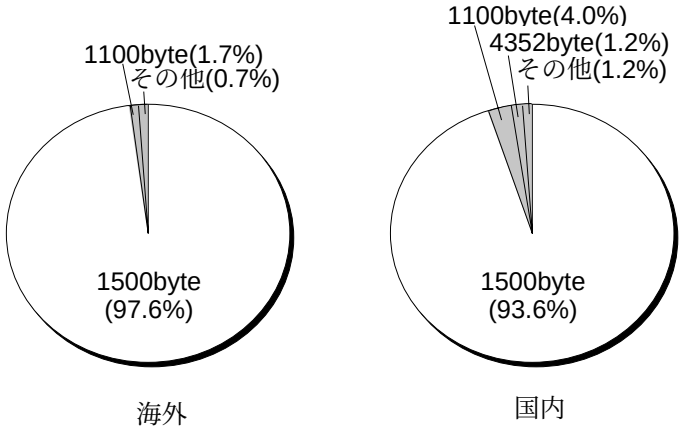

図 1.4: 経路 MTU の分布

図 1.4をみると 90%以上の経路が、1500 バイトの経路 MTU を持っていることが分かる。よって、奈良先端科学技術大学院大学では初期 MTU を 1500 バイトに設定して通信すれば予測探索方式は有効だと考えられる。

また、初期 MTU はルータに設定するので、リンクローカル・ネットワークごとに柔軟に初期 MTU を設定できる。ただし、管理者の介入を必要とする。

しかし、最適な経路 MTU が求まらない場合があるという欠点がある。予測探索方式は大きい経路 MTU をもつ未知の経路上で終点と通信するとき、初期 MTU が経路 MTU に設定されるので、最適な経路 MTU は発見できない。ただ、図 1.4の様な経路 MTU の分布だと、上述したような経路は稀であり、少なくとも奈良先端科学技術大学院大学においては現段階では無視できると考えられる。

1.5.3 経路 MTU 探索の評価

本節ではネットワークを 4 つの簡単なモデルで表現し、経路 MTU の探索方式を評価する。また、実際にネットワーク・モデルと同じ実験環境を使って探索時間を計測した。評価する探索方式は次の 3 つである。

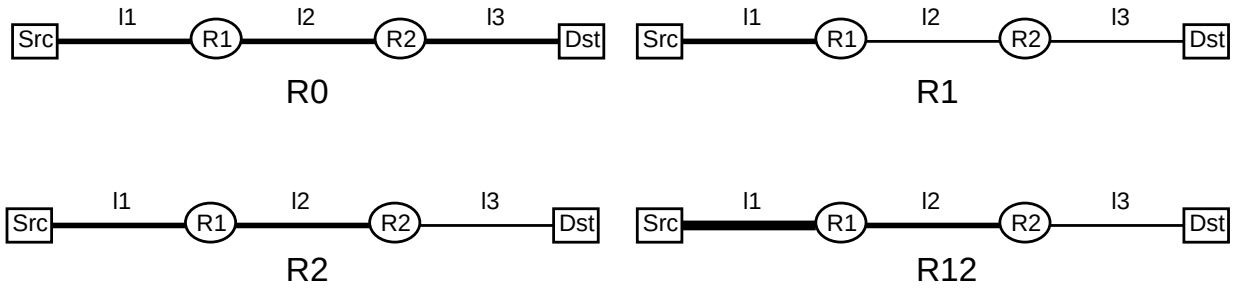

図 1.5: ネットワーク・モデル

- 受動方式
- 探査方式
- 予測探索方式

2 つめの探査方式は RFC1063 で記述されている方式である。この方式は中継点ヘッダ (IPv4 では IP オプション) を付加したデータが経路上を 1 往復することによって、正確な経路 MTU を探索する方式である。予測方式の評価は受動探索と予測探索方式の結果から推測できるので省略した。

今回のネットワーク・モデルの前提条件は以下の通りである。また、各リンクの MTU を l_nとし、予測探索方式でルータからもらっている初期 MTU を L とする

- MTU が経路上で変更されるのは 2 回とする
- ノードからノードへデータが転送される転送時間はすべて同じ t_1とする
- ノードからデータを入力する時間と出力する時間を t_2とする
- 初期 MTU はすでに設定されているとする
- すべて未知経路での探索とする

ネットワーク・モデルは、l_1、l_2、l_3の長さを変えて作成する。経路上での MTU 長の減少は経路 MTU の探索時間に影響すると考えられるので、リンクの MTU が小さくなる点に注目した 4 つのモデルを作成した。

今回作成した 4 つのネットワーク・モデルを図 1.5に示す。`Src` はデータを送信する始点を表し `Dst` は終点、R_nはルータを表している。l_1、l_2、l_3は各ノードを結ぶ通信媒体の MTU を表し、通信媒体の太さは MTU の大小を表している。

	R_0		R_{12}	
	探索時間 (ms)	経路 MTU	探索時間 (ms)	経路 MTU
A(受動探索方式)	0.117/0	l_1	12.989/6T	l_3
B(探査方式)	15.224/6T	l_1	15.282/6T	l_3
C(予測探索方式)	0.163/0	$l_1(l_1 <= L)$	13.005/6T	A($l_1 <= L$)
	0.086/0	$L(l_1 > L)$	0.132/0	$L(l_2 >= L$ & $l_3 >= L)$
			9.513/4T	$l_3(l_2 >= L$ & $l_3 < L)$
			11.95/6T	$l_3(l_2 < L$ & $l_3 < L)$

	R_1		R_2	
A(受動探索方式)	3.648/2T	l_2	9.578/4T	l_3
B(探査方式)	15.208/6T	l_2	15.214/6T	l_3
C(予測探索方式)	3.549/2T	A($l_1 <= L$)	9.446/4T	A($l_1 <= L$)
	0.132/0	$L(l_2 >= L)$	0.114/0	$L(l_3 >= L)$
	3.712/2T	$l_2(l_2 < L)$	9.687/4T	$l_3(l_3 < L)$

($T = t1 + 2t_2$とする)

表 1.1: 評価結果

4 つのモデルの名称は各モデルで MTU が小さくなるルータの番号をとって、「R_0モデル」、「R_1モデル」、「R_2モデル」、「R_{12}モデル」と名付けた。R_0モデルは経路上で一度も MTU が小さくならないことを意味している。

評価結果は表 1.1のようになった。探索時間の列の少数は実験による計測時間でスラッシュ(/) の後に、ネットワーク・モデルによる理論的な探索時間を記述している。

表 1.1から次のことが分かる

(1) 受動方式と予測探索方式のネットワーク・モデルでの探索時間はすべてのモデルで同じである

(2) 予測探索方式は初期 MTU L をうまく選択すると探索時間が短くなる

(3) 探査方式は経路上で MTU が連続的に小さくなるような経路での探索に適している。

まず、(1) の受動方式と予測探索方式のネットワーク・モデルでの最大探索時間が同じなのは、予測探索方式が受動方式を応用した探索方式だからである。つまり、初期 MTU L をリンク MTU に設定したときの予測探索方式は受動方式と全く同じ探索方式になる。よって、L の選び方によっては予測探索は探索時間が受動探索と全く同じになってしまう。

(2) は予測探索方式の利点である。予測探索方式は条件によっては、すべてのモデルにおいて経路 MTU を最短時間で探索できる。つまり、3 つの探索方式のなかで最高の性能を発揮できる。ただ、予測探索方式は探索時間の向上のために初期値以上の経路 MTU を求

められないという点を犠牲にしている。これは R_0モデルでの経路 MTU 値をみれば明らかである。

(3) は R_{12}モデルから推察できる。R_{12}モデルは、経路上で MTU が小さくなる回数が最も多いモデルで各方式の探索時間の差が最も少ない。何故なら、探査方式が経路上の MTU 長が探索時間に影響しないのに対して、受動方式や予測探索方式は経路上で MTU が小さくなる回数が増えるにつれ探索時間が長くなるからである。ただ、予測探索方式は L をうまく選択すれば探索時間を短くできる。しかし、他のモデルにおいて、L は 1 つの条件を満たせばよいのに対し、このモデルでは最短時間で探索するために、L が 2 つの条件を満たす必要がある。

実験結果とネットワーク・モデルでの評価に最も差異があるのは、R_{12}モデルである。R_{12}モデルの理論的な最大探索時間はどの探索方式でも同じである。しかし、実験結果では探査方式が他の方式より探索時間が長い。これは探査方式では経路上のルータがオプションを処理するからだと考えられる。探査方式では各ルータが中継点オプションを解析し、リンク MTU とデータグラム長を比較し、場合によっては中継点オプションのデータを更新しなければならない。それに対し、他の方式ではリンク MTU とデータグラム長との比較処理と、エラーが発生したときのみ ICMP パケット過大メッセージを作成するのみである。このことから、探査方式は経路上で MTU の小さくなる回数が大きいときの探索に有利ではあるが、中継数が増えるたびにルータの負荷も増えるので、ルータの処理を軽減しない限り、予測探索方式との探索効率の差が少なくなる。

以上のことから予測探索方式は、最適な探索ができる L の値が同じ経路がインターネット上に多く存在するとき効率よく探索できることが分かる。また、L の値はリンクローカル・ネットワークごとに設定できるので柔軟な設定が可能である。また、計測時間の値とネットワーク・モデルの理論値の上下関係もほぼ合っていることから、ネットワーク・モデルに対する各探索方法の評価は正しく、経路 MTU の探索をインターネットに導入するときの目安となるだろう。

1.5.4　経路 MTU 探索に関するまとめと評価

IPv4 であまり利用されなかった経路 MTU 探索は IPv6 で必須になった。よって、効率のよい探索方式を見つけるためにインターネット上の経路 MTU の分布を調査し、その結果をもとに新しい探索方式である予測方式を提案した。予測方式は以下の利点を持っている。

1. 2 つの異なる探索方式を柔軟に利用できる
2. 管理者が管理しているホストにのみ実装すればよい

そして、ネットワーク・モデルと実験環境を作成し、予測方式を他の探索方式と比較し評価した。その結果以下のことが分った。

1. インターネット上の多数の経路が同じ経路 MTU のとき予測方式は有効である

2. インターネット上の多数の経路において、MTU が終点に向かって小さくなっていくときは探査方式が有効になる。ただし、ルータに負荷がかかる

今後、次の問題を解決する必要がある。

- インターネットの経路 MTU の分布状況を定期的に調査し、より詳しい動向をつかむ
- 各経路 MTU の探索方式を 6bone で利用し評価する

1.6 トランスレータ

インターネットの急速な成長に伴い、IPv4 アドレスの枯渇が問題になっている。128 ビットのアドレス空間を持つ IPv6 はこの問題を解決するだけではなく、様々な新しい機能を提供している。しかし、アドレス空間が異なるため IPv4 と IPv6 の間に互換性はない。IPv4 から IPv6 へのインターネットの移行は非常に重要な問題だが、現在採択されているデュアル・スタックとトンネリングを用いた移行機構では決して十分であるとは言えない。

本節では、既存の異種プロトコルの相互接続技術について述べた後、V6 分科会で提案された 3 種のトランスレータの特徴とその内容を説明する。

1.6.1 異種プロトコルの相互接続技術

この節では、IPv4 と IPv6 のように異なるネットワーク層のプロトコルを相互接続する方法を整理する。相互接続技術には以下の種類がある。

- ネットワーク層での相互接続技術
  - デュアル・スタック
  - トンネリング
  - ヘッダ変換
- 上位層での相互接続技術

以降では、これらの技術について概説する。

デュアル・スタック

複数のネットワーク層のプロトコルを扱うことである。たとえば、市販されている多くのルータは IPv4 だけではなく、AppleTalk や Netware 等の複数のプロトコルを扱える。

本節でいうデュアル・スタックとは IPv6 の機能を持つと同時に、IPv4 の機能を併せ持つ計算機のことである。デュアル・スタック計算機は、IPv4 計算機と通信するときには IPv4 を用い、IPv6 計算機と通信する際は IPv6 を使用する (図 1.6)。

デュアル・スタックを用いることで、ローカル・ネットワーク上に IPv4 と IPv6 の計算機が混在していても、それぞれが相互に通信できる。しかし、ネットワーク的に隔離された IPv6 計算機同士、すなわち途中経路に IPv4 ルータしか存在しない IPv6 計算機同士は通信できない。このような状況下では次節のトンネリングを併用して、相互接続性を確保する。

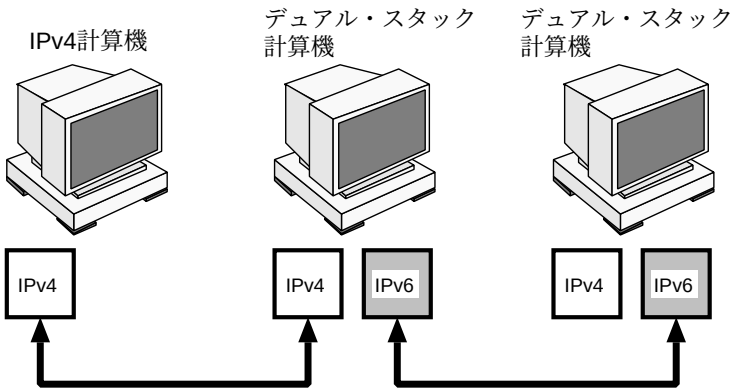

図 1.6: デュアル・スタックの概念図

トンネリング

あるネットワーク層のプロトコルを通すために、異なるネットワーク層を利用する技術をトンネリングという。トンネリングを用いれば、IPv6 パケットの前に IPv4 ヘッダを付加することにより、IPv4 ネットワークを利用して IPv6 パケットを転送できる。その結果、ネットワーク的に隔離された IPv6 の計算機同士でも通信が可能になる。(図 1.7)。

ヘッダ変換

ヘッダ変換は途中経路でパケットそのものを変換してしまう技術である。その例としては、プライベート・アドレスをグローバル・アドレスに変換する NAT[?] がある。これは IPv4 から IPv4 へのヘッダ変換である。

ヘッダ変換を用いると、IPv4 のみの計算機と IPv6 のみの計算機が通信できる (図 1.8)。しかし、異なるプロトコル同士の完全な 1 対 1 の変換は不可能であり、異なるアドレス空間を持つ IPv4 と IPv6 のアドレスの対応づけが問題となる。

上位層での相互接続技術

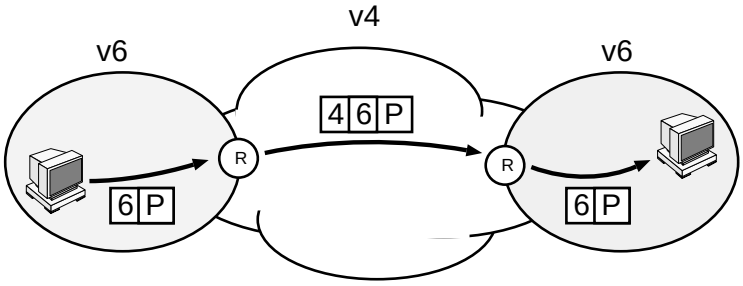

図 1.7: トンネリングの概念図

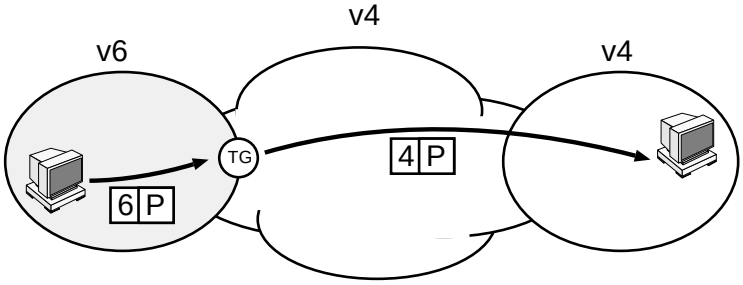

図 1.8: ヘッダ変換の概念図

途中経路のゲートウェイで、ネットワーク層よりも上位の階層で通信を一旦終端してやり、改めて終点へコネクションを確立することでネットワーク層プロトコルの差異を吸収できる。

終端する層によって以下の 2 つのゲートウェイがある。

- トランポート層ゲートウェイ
- アプリケーション層ゲートウェイ

従来は、相互接続技術としてではなく、SOCKS[?] などの防火壁 (Firewall) を構築するための技術として利用されてきた。

途中経路でコネクションを再接続するため、ヘッダ変換と同様に IPv4 と IPv6 のアドレスをどう対応づけるかが問題である。

1.6.2　現在の移行戦略とその問題点

IETF の次世代移行分科会はデュアル・スタックとトンネリングを用いて移行を進めることに決定した [?]。しかし、この 2 つの技術だけで混在環境下であらゆる計算機が相互に通信できるわけではない。

IPv4 と IPv6 の混在環境下では、次の 3 種類の計算機が存在する。

- IPv4 のみの計算機
- デュアル・スタック計算機
- IPv6 のみの計算機

それぞれの計算機が相互に通信すると表 1.2の組合せが考えられる。デュアル・スタックとトンネリングを併用するだけでは、IPv4 のみの計算機と IPv6 のみの計算機が通信できないのが分かる。移行期の前半では、IPv6 計算機はデュアル・スタックであることが義務づけてられているので、IPv4 のみの計算機と IPv6 のみの計算機が通信する状況はほとんどありえない。

しかし、移行期の後半では、IPv4 アドレスはすでに枯渇してしまっているので、たとえデュアル・スタック計算機であっても IPv4 アドレスを割り当てることはできない。この計算機は IPv6 のみの計算機と同義であり、IPv4 のみの計算機と通信できない。

1.7　3 つのトランスレータ

表 1.2の “通信不可” の部分を通信可能にする、すなわち IPv4 のみの計算機と IPv6 のみの計算機の途中経路でパケットを相互に変換する技術がトランスレータである。

表 1.2: 相互接続表

	IPv4 のみ	デュアル・スタック	IPv6 のみ
IPv4 のみ	IPv4 で通信	IPv4 で通信	通信不可
デュアル・スタック	IPv4 で通信	IPv6 で通信	IPv6 で通信
IPv6 のみ	通信不可	IPv6 で通信	IPv6 で通信

V6 分科会では 3 つのトランスレータ、TOWNS、FAITH、SOCKS64 が提案されている。ここでは、それぞれの特徴と問題点などを述べる。

1.7.1 TOWNS

TOWNS(TranslatOr With Name Server) は奈良先端科学技術大学院大学の角川宗近が提案するヘッダ変換ゲートウェイである。

これは、第 1.6.1節のヘッダ変換をもとにした技術である。TOWNS は従来のデュアル・スタックとトンネリングによる移行機構を補完する形で動作する。ヘッダ変換の際、変換する IPv4 と IPv6 の始点、終点アドレスをどう対応させるかが問題になるが、これを改良を加えたネームサーバを用いることで解決する。

TOWNS の対象モデル

TOWNS は移行期の後半に IPv4 と IPv6 の相互接続性を確保することを狙いとしている。この時期にはバックボーンをはじめとするインターネットの大部分は IPv6 に対応し、一部のサイト[2]のみがまだ IPv4 のまま残っている。IPv4 アドレスは枯渇し、IPv6 計算機の大部分は IPv6 の機能しか有してない。

この対象モデル内の 1 つの IPv4 サイトが TOWNS を導入することによって、IPv6 インターネット全体との相互接続性を確保できる。つまり、この IPv4 サイト内の全計算機は、透過的にサイト外の IPv6 計算機と通信できるようになる。

アドレスの対応づけ

IPv4 と IPv6 のヘッダを変換するためには、IPv4 と IPv6 の始点アドレスと終点アドレスが一意に変換できなければならない。しかし、IPv4 に比べて、IPv6 のアドレス空間は著しく大きいため、IPv6 アドレスを一意に IPv4 アドレスに割り当てることは無理である。

そこで、通信したい IPv6 計算機のアドレスを一時的に IPv4 アドレスに割り当てる方法を採用する。TOWNS では設定によって任意の IPv4 アドレスへの割り当てが可能だが、プライベート・アドレス `10.0.0.0/8` の利用を推奨する。これはサイト内で自由に使えるアドレス空間である [?]。この IPv4 アドレスはネームサーバを利用して自動的に割り当てる。つまり、クライアントからネームサーバへの問い合わせに応じて、IPv4 アドレスを割り当

[2]本節では「サイト」とは 1 大学や 1 企業などある程度の規模のネットワークを指す

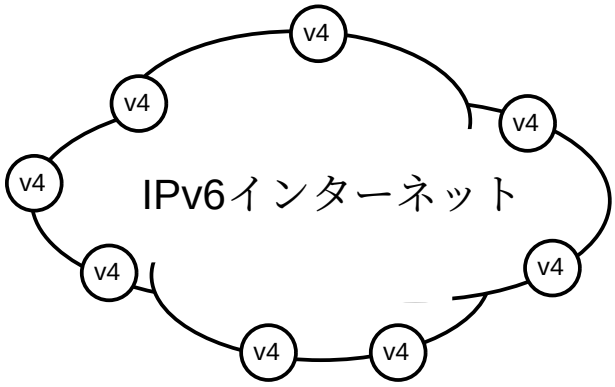

図 1.9: TOWNS の対象モデル

てていく。

逆に IPv4 から IPv6 へ対応づけは後者の方がアドレス空間が広大なので、静的な割り当てが可能である。TOWNS ではサイトが取得した IPv6 アドレスの空間の一部を任意の IPv4 アドレスに割り当てることができる。

TOWNS の実装

TOWNS のシステムは以下の 3 種類のソフトウェアから構成される (図 1.10)。

変換ゲートウェイ

IPv4 と IPv6 パケットを相互に変換する。

マップサーバ

変換のために一時的に対応させた IPv4 と IPv6 のアドレスの組を管理する。

改良ネームサーバ

IPv6 計算機に対して IPv4 アドレスを一時的に割り当て、それをクライアントに返す。

これらのサーバとゲートウェイは “変換アドレス要求”、“変換アドレス返答” という 2 つのプロトコルで互いに情報を交換する。このプロトコルは IPv4 の ICMP で実現されている。図 1.11にこの ICMP のパケット形式を示す。

1.7.2　FAITH

FAITH は奈良先端科学技術大学院大学の山本和彦と (株) シャープの島 慶一の開発によるアプリケーション層のゲートウェイである。ユーザは外部の計算機と透過的に通信している感覚を受けるが、実際には途中経路の FAITH ゲートウェイで一旦コネクションを終端する。そして、ゲートウェイ上の代理サーバを通して外部の計算機と通信する。ステル

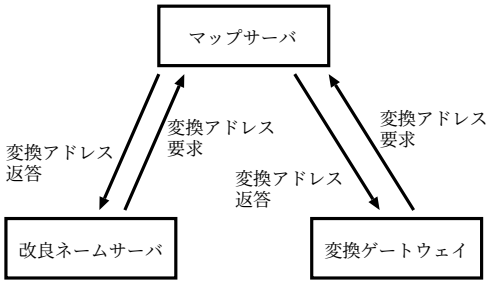

図 1.10: TOWNS のシステム構成

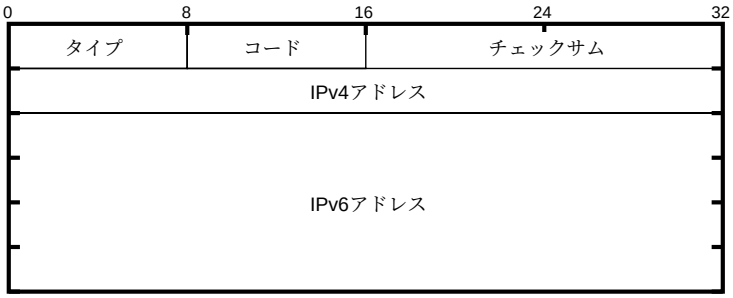

図 1.11: 変換アドレス要求・回答プロトコルのヘッダ形式

ス型の防火壁として動作しながら、トランスレータとしても動作する。FAITH はユーザ空間で実現されている。

第 1.6節で問題になった IPv4 と IPv6 のアドレスの対応づけは、TOWNS で提案した機構をそのまま使用している。つまり、改良ネームサーバを用いて一時アドレスを割り当てることで解決している。

1.7.3　SOCKS64

SOCKS64 は (株) 富士通研究所の陣崎 明と小林伸治が提案するアプリケーション層のゲートウェイである。TOWNS と FAITH がアドレス対応管理のために DNS を改造するというアプローチを採用しているのに対して、SOCKS64 は防火壁の技術として開発された SOCKS を利用することで IPv4 と IPv6 のアドレス対応表管理を不要としていることに特徴がある。

アプリケーションのユーザは接続先を IP アドレスで直接与えることは少なく、通常は接続先を名前 (FQDN。Fully Qualified Domain Name[?]) で指定する。FQDN は DNS で IP アドレスに変換するが、防火壁環境では組織内部の DNS と組織外部の DNS とを分離して運用していることも少なくない。このため、SOCKS は接続先を FQDN のままゲートウェイに渡し IP アドレスへの変換もゲートウェイに任せる仕組みを提供している。SOCKS64 は SOCKS のこの機能を利用することで IPv4 と IPv6 のアドレス対応表管理を不要としている。

SOCKS

SOCKS[?] は米国 NEC が開発した技術で、1996 年の 3 月に RFC になった SOCKS5[?] が最新の仕様である。SOCKS5 の実装も米国 NEC から公開されている。

SOCKS は通常のソケットを SOCKS 用のライブラリで置き換える。このためクライアント側は SOCKS 対応のアプリケーションを準備する必要があるが、UNIX であればアプリケーションをライブラリを入れ替えて再コンパイルする程度の手間ですむ。PC などクライアント側では SOCKS 機能をもつアプリケーションを導入しなければならないが、最近市販されているネットワーク・アプリケーションでは SOCKS など proxy 機能をサポートする例が多くなっている。また、Windows 用には普通のアプリケーションを SOCKS に対応させる SocksCap というライブラリも提供されている。

SOCKS ライブラリはまずクライアントと SOCKS サーバを TCP/IP で接続し、かつ SOCKS サーバから本来接続すべきリモートサーバへ TCP あるいは UDP で接続する。これが成功すれば、その後はクライアントとリモートサーバが SOCKS サーバを介して透過的に接続される。

SOCKS4 まではソケットをそのまま SOCKS サーバで実現するような単純な仕様となっていたが、SOCKS5 では以下のような仕様が追加された。

- アドレス表現形式として IPv6 アドレスが追加された

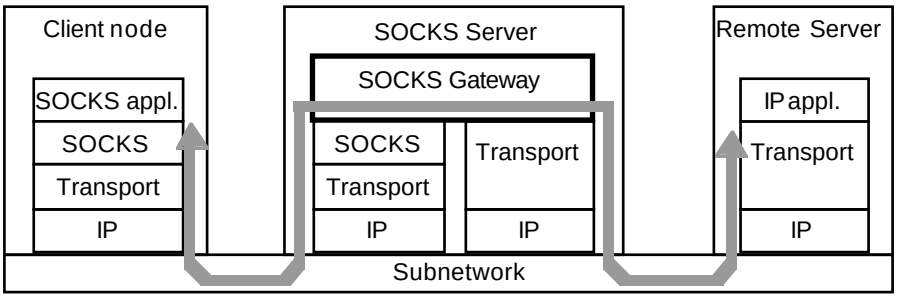

図 1.12: SOCKS による相互接続

- アドレス表現形式として FQDN が追加された
- 認証機能が追加された
- UDP サポートが追加された

ここでトランスレータに大きく関係するのはアドレス表現形式として IPv6 アドレスと FQDN が追加された点である。特に FQDN が扱えるようになった点は DNS 管理を明確にし、IP アドレスの取扱いを SOCKS サーバで一元化できることを意味する。

SOCKS5 でのアドレス表現

SOCKS Version 5 のプロトコルでは、クライアントは SOCKS サーバに対して接続先ノードへの `connect()` や `bind()` を要求する。その際、接続先ノードは図 1.13のようにアドレスタイプとアドレス (任意長) で通知する仕様になっているため、IPv4 アドレスに限らず任意のアドレスを渡すことができる。

フィールド名	VER	CMD	RSV	ATYP	DST.ADDR	DST.PORT
フィールド長	1	1	1	1	可変	2

VER	プロトコルバージョン
CMD	コマンド
RSV	リザーブ
ATYP	アドレスタイプ
DST.ADDR	接続先アドレス
DST.PORT	接続先ポート

図 1.13: SOCKS5 の接続要求フォーマット

SOCKS5 で定義されているアドレスタイプは IPv4 アドレス、IPv6 アドレス、FQDN の 3 通りである。

SOCKS64 の動作

SOCKS64 の動作を具体的に説明する。

前提として、ユーザは SOCKS5 対応のネットワーク・アプリケーションを持っている必要がある。サーバは SOCKS64 ゲートウェイを動作させておく必要がある。このサーバも IPv4 と IPv6 の両方を使っていることを除いて通常の SOCKS サーバと同じである。作成上も IPv6 対応であることによる本質的な問題はない。なおリモート・サーバ側は SOCKS とは関係ない。

まずユーザは SOCKS サーバに対して SOCKS コネクションを要求する。このとき終点アドレスは FQDN で指定する。これを受けた SOCKS サーバはまず与えられた FQDN を解決し、得たアドレスが IPv6 アドレスであれば IPv6 ソケットを、IPv4 アドレスであれば IPv4 ソケットを使ってリモート・ノードに対するコネクションを確立する。その後ユーザに対して応答を返す。

ユーザがゲートウェイに対してデータを送信すると、ゲートウェイはユーザ空間までデータを吸い上げ、宛先ノードに中継する。このとき、中継前と後で通信は完全に独立であり、アプリケーションに応じた処理が可能になるためネットワーク層で上位層のデータの検査なしで必要に応じて処理できる。

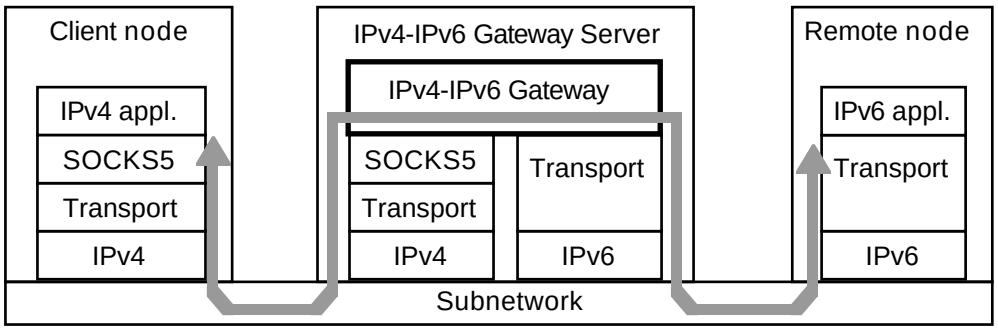

図 1.14: SOCKS64 による相互接続

以上のように IPv4 ユーザも IPv6 ユーザも終点アドレスやプロトコル意識することなく通信できる。

SOCKS64 の実装

SOCKS64 は Sun が配布している Solaris2.5 for x86 用の IPv6 スタックの実装、Release-4 を用いて実装した。使用した SOCKS5 のバージョンは socks5-beta-0.16.4-exportable である。

SOCKS バージョン 5 の実体は libsocks5.a というライブラリである。このライブラリをリンクすると `connect()` や `bind()` などの関数が SOCKS バージョン 5 のものに置き換わる。これらの関数は、引数で指定された接続先に直接接続する代わりに、設定ファイル/etc/libsocks5.conf や環境変数 SOCKS_SERVER などで指定される SOCKS サーバに対

して接続を要求する。

実装状況、制限

SOCKS64 ゲートウェイを実現するために、SOCKS5 のライブラリとサーバのプログラムを変更した。動作の確認には telnet と ftp を用いた。これらには変更を加えていない。また IPv6 に対応したアプリケーションとして、INRIA の IPv6 コードを参考にして telnet と ftp を変更し、telnet6 と ftp6 を作成した。

サーバの変更点を少なくするため、IPv4 ノードからの要求を受け付けるプロセスと IPv6 ノードからの要求を受け付けるプロセスとを別のプロセスとして動作させる仕様にした。SOCKS5 サーバ socks5 の起動時に-6 オプションを与えると IPv6 ノードからの要求を受け付けるプロセスとして動作する。このオプションを与えないと IPv4 ノードからの要求を受け付けるプロセスとして動作する。

SOCKS5 のサーバは応答を速くするためにあらかじめ `fork()` しておくなど、いくつかの動作モードを持つが、今回の変更を加えたサーバはこれらの動作モードすべてについて動作確認をしたわけではない。SOCKS サーバプロセスを 2 つ動作させる仕様にしていることも関係して、プロセス間でのロックの確保などにバグが残っている可能性もある。

なお、今回の実装では UDP には対応していない。

SOCKS64 の動作確認

4 種類のクライアントを用いて動作を確認した。動作確認に使用したノードの OS は表 1.3 の通りである。実験では、接続元のノードは SOCKS64 ゲートウェイを動作させているノードと兼用した。SOCKS サーバと IPv4 で通信するか IPv6 で通信するかは SOCKS_SERVER 環境変数に IPv4 アドレスを指定するか、あるいは IPv6 アドレスを指定するかで変更している。

ノード	OS
SOCKS64 ゲートウェイ	Solaris for x86 IPv6 Release-4
接続先 IPv4 ノード	SunOS 4.1.3_U1
接続先 IPv6 ノード	Solaris 2.5 IPv6 Release-4

表 1.3: SOCKS64 の動作確認環境

各クライアントを用いた動作確認の結果は表 1.4の通りである。

IPv4 ノードから IPv6 ノードへの ftp が△となっているのは接続はできるがディレクトリ表示やデータ転送といった操作ができないことを意味する。これは現在の SOCKS64 の実装が PORT コマンドと LPRT コマンドの相互変換 [?] をサポートしていないためである。PASV コマンドも同様に LPSV コマンドに変更する必要がある。

		接続先	
クライアント	接続元	IPv4	IPv6
telnet	IPv4	○	○
telnet6	IPv4	○	○
	IPv6	○	○

		接続先	
クライアント	接続元	IPv4	IPv6
ftp	IPv4	○	△
ftp6	IPv4	○	○
	IPv6	○	○

表 1.4: SOCKS64 動作確認結果

1.7.4　トランスレータの評価

前節までに説明した 3 つのトランスレータのうち TOWNS はヘッダ変換ゲートウェイ、FAITH と SOCKS64 はアプリケーション層ゲートウェイである。まずそれぞれの方式が持つ問題点を整理し、性能と導入コストについて比較を行った後に 3 種類のトランスレータの特徴をまとめる。

ヘッダ変換方式の問題

ヘッダ変換は高速に処理できるという特長を持つ反面、IPv4 と IPv6 のアドレス対応の問題、および FTP のように上位層でネットワークアドレスを利用するプロトコルへの対応の問題がある。TOWNS は前者については DNS の改造によるアドレス変換を採用し、後者についてはそのようなプロトコルには対応しないという方針を採っている。

アプリケーション層ゲートウェイ方式の問題

アプリケーション層ゲートウェイ方式は機能的にみるとほぼ完全といってよいが、性能と実現方式に問題がある。

まず性能はネットワーク規模にもよるが、かなり厳しい問題である。たとえば SOCKS は通常 1 コネクションについて 1 個のプロセスを起動する。仮に 10Mbps Ethernet の帯域をすべて使って平均 1000Bytes のパケットが SOCKS 経由で流れるとすると、最高 500 コネクション (1000000 / 1000 / 2) が確立する可能性がある。500 プロセスが同時に動作するという状況は通常の UNIX ワークステーションならほとんど崩壊状態である。このような状態では遅延が大きくなるだけでなく、遅延のばらつきも増えることが予想されるので、いわゆるリアルタイム通信では致命的となることも考えられる。

性能問題についてはソフトウェアの作りを工夫して (いわばチューニングによって) ある程度性能を出そうというアプローチや、複数のマシンを使って負荷分散しようとするアプローチが代表的である。ソフトウェア・チューニング方式の代表的なものは、SOCKS の現在の実装にも一部組み込まれているようにスレッドなどの技術を使ってプロセス数を抑える方法である。WWW サーバでも同じ方法で性能を向上させる試みがある。但しこの方法はあくまでもチューニングであって、抜本的な性能改善とはならない。

一方、負荷分散は WWW サーバなどでよく用いられている方法で、本来IP 通信は相互に

独立であることを利用する。代表的なのは DNS による負荷分散で、Shuffle Address(SA) という概念を用いる。サーバ・グループに代表番号としての FQDN を割り当て、その FQDN をアドレス検索するときにクラスタ内の異なるマシンの IP アドレスを平均的に割り当てる。

以上のアプローチは 1.5Mbps の T1 が主流であるうちはある程度有効だが、近い将来 ATM など 100Mbps 以上の高速な広域ネットワークが普及すると対応困難になると考えられる。アプリケーション層ゲートウェイ方式によってトランスレータなどのアプリケーション層ゲートウェイを構築していくためには、現在の 10 倍〜100 倍という高速ネットワークに対応できるアプリケーション層ゲートウェイサーバ実現技術が重要な鍵となるだろう。

もう 1 つの実現方式の問題は性能と実現の容易さのトレードオフである。SIT などネットワーク層の機能はノードにおいてはカーネル空間で実装されてきたし、専用のリアルタイム OS が動作するルータでもサポートできた。カーネル空間でのパケット処理は SOCKS のようなユーザ空間の処理に比べると格段に効率がよい。しかし、アプリケーション層ゲートウェイ方式では基本的にセッション単位にコンテキストを分ける必要があるため、低いレベルでサポートするのは面倒である。

すなわち、UNIX のように複数の処理コンテキストをタイム・シェアリングで実行できるアプリケーション動作環境では容易にアプリケーション層ゲートウェイ方式をサポートできるが、性能的な限界が予測される、一方そうでない環境では実現が困難であるというジレンマがある。今後 100Mbps 台のネットワークが普及していくことを考えると、既存アーキテクチャでの性能限界を明らかにし、将来に向けて実現性と性能の両面を解決可能なアーキテクチャが必要である。

性能の評価

TOWNS と FAITH の変換速度を測定するために、図 1.15の実験環境を構築した。IPv4 計算機 A から TCP を用いて 1000 バイトの IPv4 パケットを 10000 個送出し、ヘッダ変換ゲートウェイを用いて IPv6 計算機 B と通信するのに要した時間を計算した。

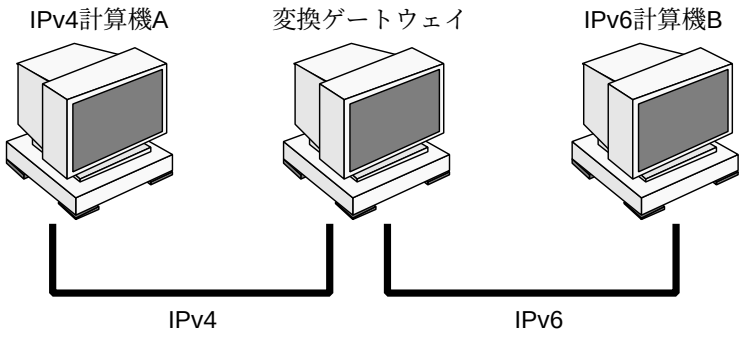

図 1.15: 速度比較の実験環境

また、高負荷時の変換遅延を調べるため、ゲートウェイ上にプロセスを 1 つ、プロセス

を 2 つ、それぞれ実行したときの時間を調べた。

参考のために単純に IPv4 の転送、IPv6 の転送に要した時間も計測した。この 2 つの時間の平均値がヘッダ変換の速度の上限だと考えられる。SOCKS64 はまだ開発中のため、変換速度を評価できなかった。しかし、その動作原理を考えると、FAITH と同程度の転送速度であると考えられる。

この速度表を表 1.5に挙げる

表 1.5: 各トランスレータの速度比較

	負荷なし (秒)	負荷 1(秒)	負荷 2(秒)
IPv4	15.71	16.21	16.26
IPv6	16.28	16.97	17.19
TOWNS	17.30	17.73	17.88
FAITH	17.73	32.82	50.03

TOWNS は負荷の有無に関わらず、単純転送にかなり近い速度で動作することが分かる。FAITH については負荷がない状態では TOWNS と同程度の速度で動作するが、負荷が上がるにしたがって、変換時間が増大することがわかる。

実際の運用を考えると、TOWNS は代理サーバを必要としないが、FAITH はゲートウェイ上でコネクションの数だけ代理サーバを起動するので負荷が増大する。よって、TOWNS と FAITH の速度比は大きくなると予想される。

3 つのトランスレータの比較

3 つのトランスレータの特徴を以下に示し、比較表を表 1.6に示す。

TOWNS

- パケットの変換が速い。
- ゲートウェイ上でサーバを立ち上げる必要はない。
- プロトコルに依存しないでヘッダ変換できるので、未知のプロトコルにも基本的には対応できる。しかし、階層違反のプロトコルには対応が困難である。
- アドレス対応表を管理する必要がある。
- 導入しなければならないシステムは、改良ネームサーバ、マップサーバ、ヘッダ変換ゲートウェイである。
- クライアントには一切変更しなくてよい。
- 外部ノードから内部ノードへのネットワーク透過性を実現できる。

FAITH

- パケットの転送が遅い。
- ゲートウェイ上にコネクションの数だけサーバをに立ち上げるため負荷が集中する。
- プロトコルごとに専用のサーバが必要となるが、階層違反のプロトコルにも対応できる。
- アドレス対応表を管理する必要がある。
- 導入しなければならないシステムは、改良ネームサーバ、マップサーバ、FAITH ゲートウェイとその上で動作するプロトコルごとの転送サーバである。
- クライアントには一切変更しなくてよい。
- コネクションを (ユーザには感じさせないが) 終端するので、きめの細かいアクセス制御ができる。
- 外部ノードから内部ノードへのネットワーク透過性を実現できる。

SOCKS64

- パケットの転送が遅い。
- ゲートウェイ上にコネクションの数だけサーバをに立ち上げるため負荷が集中する。
- プロトコルごとに専用のサーバが必要となるが、階層違反のプロトコルにも対応できる。
- アドレス対応表は不要。
- コネクションを (ユーザには感じさせないが) 終端するので、きめの細かいアクセス制御ができる。
- 導入しなければならないシステムは、SOCKS64 ゲートウェイとその上で動作するプロトコルごとの転送サーバである。
- クライアントで使用したいアプリケーションを SOCKS 対応にする必要がある。
- 外部ノードから内部ノードへのネットワーク透過性を実現できている。しかし、外部ノードのアプリケーションも SOCKS 対応にしなければならない。
- すでに SOCKS5 を導入しているサイトではクライアントの変更は不要である。

以上の比較検討より、高速性においては TOWNS が優れてるといえる。ユーザ空間でヘッダを変換する FAITH や SOCKS64 に比べると、カーネル空間でヘッダ変換を実現する TOWNS は高速に動作する。また、プロトコルごとに代理サーバを用意しなければな

表 1.6: 各種トランスレータの比較

	導入コスト	階層違反プロトコル	転送
TOWNS	低い	困難	速い
FAITH	低い	可能	遅い
SOCKS64	高い	可能	遅い

	アクセス制御	アドレス対応表	プロトコルごとのサーバ
TOWNS	弱い	必要	不要
FAITH	強い	必要	必要
SOCKS64	強い	不要	利用可能

らない FAITH に比べ、TOWNS は代理サーバが不要で、未知のプロトコルも変換できる。SOCKS64 はプロトコル毎に代理サーバを用意することもできるが、そのような変換が特に必要の無いプロトコルはそのまま通すことができる。

企業等の防火壁内のサイトで限られたプロトコルしか通す必要がなければ、防火壁の役割も兼ねている FAITH か SOCKS64 が便利である。すでに SOCKS5 を導入しているサイトならば SOCKS5 ゲートウェイを SOCKS64 に対応させるだけでよいので、導入コストが 3 つの中で最も低いといえる。SOCKS5 を導入していないサイトでは FAITH の方が導入コストが低い。

TOWNS と FAITH はいずれもアドレス対応管理に改良した DNS を用いているので、両者の親和性は高いといえる。そこで、TOWNS と FAITH を協調動作させることによって、柔軟性と高速性を兼ね備えた新しいトランスレータが考えられる。たとえば、パケット中の TCP データを解読、変換しなければならない ftp の制御コネクションは FAITH が変換し、高速性の要求される ftp のデータ・コネクションは TOWNS が担当するなどである。この融合トランスレータは、今後の課題として議論、研究を行なっていく。

1.7.5　トランスレータのまとめ

IPv6 の普及にともない IPv4 と IPv6 の相互接続技術が重要になってくると考えられるが、デュアル・スタックとトンネリングだけでは不十分である。V6 分科会ではそれらを補う技術としてトランスレータが必要であると考え、それぞれ特徴を持つ 3 種類のトランスレータを開発した。サイトごとの使用目的に応じて適切なトランスレータを選択できる。

1.8 V6 分科会のまとめ

1996 年度の V6 分科会の活動内容は広範囲に及び、多くの成果を得られた。IPv6 のコードは、カーネル空間で 5 種類、ユーザ空間で 2 種類が独立に実装された。これらの実装は、V6 分科会内のみならず、IOL において他の実装を交えた相互接続性試験において、優秀な性能を示した。また、V6 分科会が構築した WIDE 6bone は、高速シリアル・リンクを使った IPv6 のバックボーンとしては世界初である。世界規模の 6bone の構築にも最初から参加し、重要な役割を果たしている。RIPng に関しても複数の実装を得られ、これらは実際に WIDE 6bone で用いられている。経路 MTU 探索方式に関しては、効率のよい方式を考案検討した。さらに、IPv4 から IPv6 への移行後期を見据えたトランスレータを 3 つ考案し、それらの特徴を明らかにした。

1997 年度は、1996 年度の延長上にある研究を引続き行なっていく。さらなる機能の実装や、経路 MTU 探索方式のさらなる検討が必要である。トランスレータに関しては、実装をより高度に改良していく予定である。また、新しいアドレス・アーキテクチャに追従し、移動機能の実現も目指したい。さらに、WHIPS 分科会と協調して、IPv6 での IPsec を実現し相互接続性を検証していく必要もある。