IPv6 における能動的経路選択に関する研究

2003年3月

佐賀大学大学院工学系研究科
システム生産科学専攻

大谷　誠

目 次

第 1 章　はじめに **1**
1.1　研究の背景 1
1.2　本研究の概要 2
1.3　本論文の構成 3

第 2 章　次世代インターネットプロトコル IPv6 **4**
2.1　インターネットの歴史的外観 4
2.2　IPv6 の特徴 9
2.2.1　IPv6 ヘッダ 10
2.2.2　拡張ヘッダ 13
2.3　本章のまとめ 14

第 3 章　経路選択アプリケーションの実現 **15**
3.1　経路選択アプリケーションの概要 15
3.2　ルーティングヘッダによる経路選択機能 15
3.2.1　ルーティングヘッダのフォーマット 16
3.2.1.1　従来のルーティングヘッダとの変更点 17
3.2.1.2　ルーティングヘッダの処理 18
3.2.2　IPv4 におけるソースルーティング 20
3.3　経路選択アプリケーションの実装 21
3.3.1　送信経路の選択 21
3.3.2　受信経路の選択 22
3.4　本章のまとめと議論 24

第 4 章　品質選択支援システム **27**
4.1　品質選択支援システムの機能 27
4.2　品質選択支援システムの実装 28
4.3　品質選択支援システムの動作 29
4.3.1　FTP クライアントの拡張 29
4.4　本章のまとめと議論 31

第 5 章　検証実験 **33**
5.1　経路選択の有用性 33
5.2　ルーティングヘッダのオーバーヘッド 36
5.3　本章のまとめと議論 38

第 6 章　考察 **40**
6.1　本研究の位置 40
6.2　他の研究との関連 42

第 7 章　おわりに **45**
7.1　本論文のまとめ 45

謝辞 **47**

参考文献 **48**

研究発表 **50**

付録 A　Routing Header **54**
A.1　Routing Header 54
A.2　Responding to Packets Carrying Routing Headers 59

付録 B　Advanced Sockets API for IPv6 Routing Header Option **60**
B.1　Routing Header Option 60
B.1.1　inet6_rth_space 62
B.1.2　inet6_rth_init 62
B.1.3　inet6_rth_add 63
B.1.4　inet6_rth_reverse 63
B.1.5　inet6_rth_segments 63
B.1.6　inet6_rth_getaddr 63
B.2　Examples using the inet6_rth_XXX() functions 64
B.2.1　Sending a Routing Header 64
B.2.2　Receiving Routing Headers 68

図 目 次

1.1 ユーザによる通信回線の選択 2

2.1 IPv6 ヘッダフォーマット 10
2.2 次ヘッダフィールドによるヘッダの識別 12

3.1 ルーティングヘッダのフォーマット 16
3.2 ルーティングヘッダのフォーマット (変更前) 18
3.3 ソースルーティングオプションのフォーマット (厳密) 20
3.4 ソースルーティングオプションのフォーマット (非厳密) 20
3.5 データ受信経路設定時の動作 25
3.6 FTP の経路指定例 25

4.1 ネットワークの構成例 29
4.2 品質選択支援システムを用いた通信例 30

5.1 実験ネットワークの構成図 34
5.2 スループットの比較 34
5.3 ルーティングヘッダが付加された IPv6 パケット 35
5.4 100Mbps でのデータ受信 37
5.5 100Mbps でのデータ送信 37
5.6 10Mbps でのデータ受信 38
5.7 10Mbps でのデータ送信 39

6.1 ポリシーの管理されたネットワークでの利用 43

表 目 次

3.1 ソースルーティング処理でのヘッダの書き換え 18
3.2 putroute コマンドの仕様 22
3.3 getroute コマンドの仕様 23
3.4 経路設定に関する転送パラメータコード 23

4.1 品質選択支援システムの設定例 28
4.2 品質選択支援システムのためのコマンド拡張 31

5.1 経路指定数の違いによる転送時間の変化 36

第 1 章

はじめに

1.1 研究の背景

近年、インターネットは社会基盤として、企業や学校、一般家庭に急速に普及してきた。この普及とともに、ユーザのインターネットの利用目的が多様化し、WWWやメール、インターネット電話やビデオ会議などにも日常的に利用されるようになった。このようにインターネットが日常的に利用されるにつれ、現在のインターネットの問題点がいくつか指摘されるようになった。たとえば、アドレス空間の枯渇、セキュリティの確保、NAT(Network Address Translation) のようなアドレス変換技術などによって利用を制限されるピア・ツー・ピア通信などである。またこれらの問題点とともに、インターネットを使用するユーザからの、セキュリティや通信帯域の保証などといった、通信品質に対する要望も生じ始めている。

それ故近い将来のインターネットにおいては、インターネットの利用目的に応じた通信品質を選択する方法の実現が重要な課題と考えられる。ここでの通信品質とは、帯域、セキュリティ、コスト、信頼性などの様々なものが考えられる。

一方、プロバイダなどから提供される通信回線が、帯域やセキュリティのレベルなどの違いにより多様化し、ユーザがその通信回線を選択できるような環境が整いつつある。

1.2 本研究の概要

本研究では、このような背景から、通信品質を選択する方法の 1 つとして、インターネットを使用するユーザが、インターネットの途中の経路を指定することにより、利用目的に応じた通信の品質を選択できる方法を提案する。

この方法で、通信先までに存在する品質の異なる回線の中から、使用したい品質の回線を経由するように経路を指定することで、通信品質の選択を実現する。この方法は、たとえば、通常はこの方法を使用せずにルーティングプロトコルなどに経路を任せて通信をしているが、サイズの大きなファイルを FTP でダウンロードするために広帯域な衛星回線を使用したいと言ったように、ある特定の通信に対して臨時的に通信品質を変更したい場合に有用であると考える (図 1.1)。

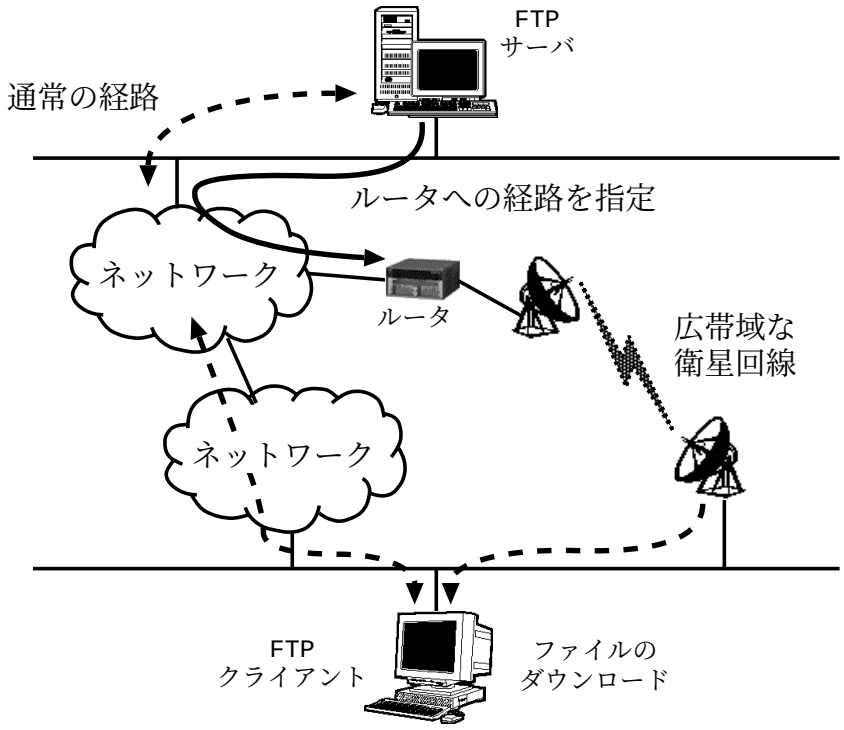


図 1.1: ユーザによる通信回線の選択

ここで経路の指定方法として、次世代のインターネットプロトコルである IPv6 (Internet Protocol version 6) の拡張ヘッダであるルーティングヘッダを使用した。このルーティングヘッダは、回線の始点ノード (たとえば図 1.1 のルータ) の IP アド

レスを指定することで経路指定が可能である。また相手先への通信経路を厳密に指定する必要がないため、容易に通信回線の選択を実現することが可能である。

この品質選択の方法を、(1) アプリケーションに実装、そして、(2) 検証を行った。また品質選択の支援を目的として、(3) ユーザに通信回線の情報を提供し品質選択の支援を行う、“品質選択支援システム” の試作を行った。

(1) においては、IPv6 におけるルーティングヘッダを利用した品質選択の枠組みを FTP アプリケーション (クライアントおよびサーバ) に実装し、アプリケーションの利用者に提供する。

(2) においては、アプリケーションが正確に経路の選択が実現されていることを、パケットのモニタリングを行い確認するとともに、帯域の異なる通信経路を持つ実験ネットワークを構築し、経路選択の有用性を検証した。また、ルーティングヘッダを使用した際のオーバーヘッドについての検証も行った。

(3) においては、ネットワークの管理者が把握している範囲の品質情報をユーザに提供するものである。これによってユーザが、ネットワークの構成を詳細に把握することなく品質選択が可能となる。

1.3 本論文の構成

本論文の以降の章では、本研究の具体的な内容について述べる。

第 2 章では、次世代インターネットプロトコル IPv6 の概要について述べる。第 3 章では、品質選択の枠組みを実装した品質選択アプリケーションの実現について述べる。第 4 章では、品質選択支援システムの概要について述べる。

第 5 章では、品質選択アプリケーションの検証について述べ、6 章を考察とし、7 章をまとめとする。

第 2 章

次世代インターネットプロトコル IPv6

2.1 インターネットの歴史的外観

現在のインターネットの仕組みを支えている IPv4 (Internet Protocol vertion 4) などのインターネットプロトコル群は、1970 年代に開発された。それから 30 年あまり経った現在、インターネットは、社会基盤として、企業や教育機関、一般家庭に急速に普及し、WWW やメール、インターネット電話、ビデオ会議など、日常的に使用されるようになってきた。このような急速なインターネットの普及によって、IPv4 が設計された当時には予測できなかった状況の変化が起こり始めた [1]。

- 通信機器の高機能、高速化

  IPv4 が設計された当時、ルータやエンドノードなどの通信機器が利用できるメモリ量は限られており非常に小さなものであった。そこで、プロトコルの処理効率を上げ、メモリの使用量をできるかぎり抑えるような実装が行われてきた。ところが現在、プロセッサの性能は飛躍的に向上し、利用可能なメモリの増設も容易になった。これに伴い、提供できるサービスも増やせるようになった。

- 回線の高速・広域化

当初、コンピュータ間の通信に使用できる回線は低速で、56Kbps、または64Kbps の専用回線が主流であった。TCP/IP によるコンピュータ間相互接続が行われた 1980 年代始めの ARPANET (Advanced Research Projects Agency NETwork) では、56Kbps のデジタル専用回線が使われていた。1986 年からは、米国のスーパーコンピュータセンター間を相互接続する NSFNET (National Science Foundation NETwork) がインターネットの発展の主舞台となり、45Mbps の T3 回線による基幹網の構築と、1.5Mbps の T1 回線による各組織との接続が進められてきた。さらに 1990 年代に入ると ATM/OC-3 (155Mbps) 回線による接続が実現され、ATM/OC-12 (622Mbps) 回線によるインターネット接続も行われるようになってきた。近年では GigabitEternet (1Gbps) の登場している。また一般家庭にも、CATV 回線や ADSL 技術によって高速なネットワークが急速に普及し始めている。

- トラフィック特性の変化

  TCP/IP の設計当時、インターネットで使用されるアプリケーションは、文字ベースの情報交換を行うものが大半であった。事実、1980 年代のインターネットは、主にファイル転送や電子メールの交換のために利用されていた。これらの古典的なサービスを提供するためのプロトコル (FTP, SMTP, Telnet など) は、トランスポート層プロトコルとして TCP を使い、クライアントとサーバ間でのインタラクティブなデータ転送を前提として設計されることが多い。したがって、これらの古典的なサービスでは、少量のデータが比較的長い時間にわたって転送されるといったトラフィックが主流であった。ところが、1990 年代になると、UDP を用いたインターネット電話やビデオ会議のような音声や映像データの転送が開始され、さらには、どちらかといえば大さなデータをいくつも連続して取得する WWW などをサービスも始められた。その結果、大量のデータが比較的短時間で転送されるトラフィックが主流になってきた。

- スケーラビリティの考慮

  TCP/IP が設計された時代には、インターネットに接続されているコンピュータの数は少なく、将来的にも数万台程度の接続しか予想しえなかった。1980 年代始めの ARPAnet 主としてホスト・コンピュータが接続されており、せいぜ

い1000台程度までしか増えないと思われていた。ところが、1980年代後半になってSunに代表される安価なワークステーションが開発され、インターネットへの接続台数が急増した。1990年代になると、パーソナル・コンピュータがインターネットに接続されるようになり、しかも米国だけでなく世界各地のインターネットが相互接続されるようになった。これにより、インターネットに接続されているコンピュータは、飛躍的に増加した。すなわち、TCP/IPに設計時点で考えたものよりも、はるかに高いスケーラビリティが必要となったのである。このために現在、不都合が表れ始めている。特に、IPのアドレス空間の枯渇が大きな議論をまき起こすこととなった。

- クライアントの急激な増加

  1990年代のインターネットの急激な発展は、インターネットのサービスを利用するクライアントの急激な増加と捉えることができる。逆にいえば、1台のサーバの提供するサービスが、膨大な数のクライアントから利用される可能性があるということである。事実、サーバに対するトラフィックの集中が従来より頻繁にみられるようになってきた。その結果、これまでのネットワークのボトルネックであったトラフィック特性が場合によってはサーバのボトルネックになりかねない状況になっている。

- 移動ホストの出現

  TCP/IPが設計されたころ、インターネットに接続されるシステムは汎用機やミニコンピュータ、ワークステーションといった固定的にインターネットに接続されるシステムばかりであった。しかし、1980年代の後半からPC関連の技術が急速に進化し、ラップトップPCなどの移動ホストが接続されるようになってきた。このようなシステムの登場により、インターネット環境においてもサービス、接続の確立といった面でのPlug & Play機能が求められるようになってきた。

- アドレス空間の枯渇

  IPv4では、通信相手の識別を行うために、32bitsで表されるアドレスを使用する。32bitsでは、約42億のアドレスを使用することが可能であるが、IPv4

アドレスの割り当ては、当初ネットワークトポロジに対してランダムに行われていたことや、アドレスクラスという概念による煩雑な割り当てを行っていたことも影響し、アドレス空間の枯渇といった問題が、危惧されるようになってきた。その他にも、煩雑な割り当てによる経路制御の複雑化、NAT(Network Address Translation) のようなアドレス変換技術などによって利用を制限されるピア・ツー・ピア通信など、いくつかの問題が発生し始めている。

このような上記の IPv4 の問題を解決し、新たなインターネットへの要求に対応するために、新たに提案されたプロトコルが次世代インターネットプロトコルである IPv6(Internet Protocol version 6) である。

まずこの IPv6 が提案されるにあたり、次世代のインターネットプロトコルが備えるべき機能を決定するために、1993 年 12 月に RFC1550 “IP: Next Generation (IPng) White Paper Solicitation” [2] が発行された。この RFC は、次世代インターネットプロトコルの選定過程で考慮すべき中心要素を、ネットワーク関係者に求めるものであり、この RFC に対して、セキュリティや大企業ユーザ、ケーブルテレビ業界などから多くの意見が寄せられた。

これらの意見を受けて RFC1726 “Technical Criteria for Choosing IP The Next Generation (IPng)”[3] においての評価作業で、使用すべき基準が 17 項目定められた。そのいくつかを以下に述べる。

- 規模

  少なくとも 10^{12} 個のエンドシステムと 10^{9} 個のネットワークを識別し、個別にアドレスを指定するとが可能でなければならない。

- 移行

  IPv4 から堅牢な移行計画を提示できなければならない。

- 安全な運用

  セキュリティ的に安全なネットワーク層がなければならない。

- 固有名

全てのIP層オブジェクトに対して大域的名称、偏在的名称、インターネット全体に対して、ユニークな名称を割り当てることができなければならない。

- 拡張性

  拡張可能でなければならず、将来のインターネットサービスに対するニーズを満たせるように成長できなければならない。また成長の過程において同一ネットワーク上にいくつものバージョンが共存できなければならない。

これらの基準をもとにして、1995年1月にRFC1752 “The Recommendation for the IP Next Genereation Protocol”[4] として3つの提案が公開された。これにはRFC1726で提示された評価基準に基づいた評価も記述されている。この3つの提案はそれぞれCATNIP (Common Architecture for the Next Generation Internet Protocol)、SIPP (Simple Internet Protocol Plus)、TUBA (TCP/UDP with Bigger Address) と呼ばれた。

この3案とも、いくつかの問題点を抱えていたが、SIPPが128ビットのIPアドレスを採用することや、その他の問題に対応することを条件に、この案が採用され、最終勧告では、はこの修正版SIPPを中心とし、TUBA案が提唱する自動設定機能とIPv4からの移行上の考慮、本研究で用いている “ルーティングヘッダ” などを盛り込んだ内容となり、これをIPv6と呼ぶこととなった。

現在IPv6の仕様はRFC2460 “Internet Protocol, Version 6 (IPv6) Specificaton” [5] などとして公開され、各研究機関や企業において活発に研究、製品開発が進められている。

特に日本は、このIPv6の研究、実験を盛んに進めており、政府の「e-japan 重点計画」においても、“2005年 (平成17年) までにすべての国民が、場所を問わず、自分の望む情報の入手・処理・発信を安全・迅速・簡単に行えるIPv6が実装されたインターネット環境を実現する ” と述べられており、現在、IPv6対応機器を導入する企業に対して、財政優遇策や低利融資制度などの拡大を行うことも検討されている。このような背景からIPv6は今後、インターネットの主要なプロトコルとなり、現在のIPv4ネットワークがIPv6へと移行していくことは、ほぼ間違いはないといえる。

2.2 IPv6 の特徴

上記に述べたように、世界的規模で使用されている IPv4 を見直す発端となった原因は、アドレス空間の枯渇への危惧からであるが、IPv6 では、アドレス空間の拡大だけでなく、セキュリティ、リアルタイム通信、モビリティへの対応など様々な拡張が行われている。以下にその特徴を示す。

- ヘッダの簡略化

  IPv6 のヘッダは、IPv4 で有効活用できなかったフィールドが削除され、簡略化が行われている (図 2.1)。また経路上でのフラグメント機能の削除なども行われ、高速な転送が可能となっている。

- アドレス空間の拡大

  IPv4 のアドレス空間の枯渇に伴い、IPv6 ではアドレス長を 128 ビットに拡張した。これにより、アドレス空間の制約を受けずにアドレスを付加することができ、今後インターネットへの接続が予想される携帯機器なども容易にネットワークへ接続できると考えられる。

- 経路の集約を考慮したアドレス構造

  IPv6 のアドレス構造には、階層構造を意識した割り当てを行い、経路制御を効率的に行うことを目指している。現時点では、Aggregatable Global Unicast Address と呼ばれるアドレス体系が定義されている。

- セキュリティ機構

  IPv6 では、パケットの認証や暗号化の機構が組み込まれており、インターネットの弱点といわれているセキュリティの強化がなされている。必要に応じてアプリケーションが通信セキュリティを確保でき、盗聴、改竄、なりすましなどを防止することが可能となる。

- Plug & Play 機能

  ホストをネットワーク接続する際、アドレス等の設定を自動で行う機能を標準で装備している。これによりホストをネットワークへ接続するだけですぐに通信を行うことが可能となる。

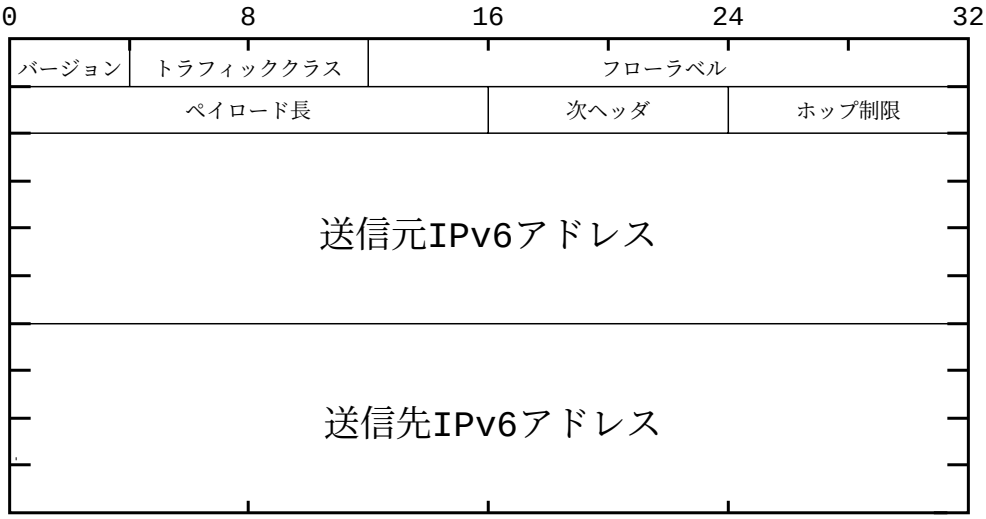


図 2.1: IPv6 ヘッダフォーマット

- リアルタイム通信への対応

  VoD (Video on Demand) やビデオ会議システムなど、リアルタイム性が要求されるトラフィックに対応するため、IPv6 ではフローラベルやトラフィッククラスといった情報がヘッダに付加される。これらの情報により IP 層のみでフローの検出が可能となり、高速な処理が可能となる。

- 拡張ヘッダ

  IPv6 の機能拡張は、それぞれ個別のヘッダとして定義され、IPv6 ヘッダと別に扱い、送信元から送信先までの経路中で処理を簡略化できる。

2.2.1 IPv6 ヘッダ

IPv6 のヘッダフォーマットは、上記に示したように IPv4 に比べて簡略化されている。以下にそのヘッダフォーマットを示す。

- バージョン

  IP のバージョンを表すフィールドで、フィールド長は 4 ビット、値は 6 である。唯一このフィールドだけが IPv4 と場所及び用途が同じである。

- トラフィッククラス

IP のパケットに対して “区分化サービス (Differentiated Service: Diffserv)” などを行うためのフィールドであり、サイズは 8 ビットである。これは IPv4 の TOS(Type of Servive) と同様な機能を提供するために提案されたものである。しかしながら現状においてこのフィールドを用いた具体的なサービスはあまり提案されていない。

また FRC2460 においてトラフィッククラスに対する挙動は以下のように定義されている。

- ノード中の IPv6 サービスを提供するサービスインターフェイスはトラフィッククラスのビット情報をそのまま上位プロトコルに渡さなければならない。
- トラフィッククラスのビットを利用するノードは、その利用目的においてパケットの生成、転送、受取りの段階で値を変更しても良い。用途不明な場合や必要としない場合は、無変更、または無視しなければならない。
- 上位プロトコルは、送信元が同じパケットのトラフィッククラスのビットの値が同じであると仮定しなければならない。

- フローラベル

  このフィールドは、通常時とは異なるサービス品質を要求する場合や、リアルタイム通信を行う際に、パケットに対する特別な処理を送信元で示す際に用いるものである。

  IPv6 仕様の第 1 版となる RFC1883[6] ではこのフィールドは 24 ビットだったが、トラフィッククラスフィールドの拡張のため 20 ビットとなった。IPv6 において特別な扱いをしなくてはならないリアルタイム通信などのサービスではこのフィールドを使ってフローの識別などを行う。フローラベルをサポートしないノードは 0 指定し、受信時は無視する。また、生成には乱数を用いる。フローの識別は送信元アドレスと 0 以外のフローラベルによって識別される。

  このフィールドの利用方法については、トラフィッククラスフィールドと同様に研究が続いている段階にある。よって今後変更される可能性もある。

、

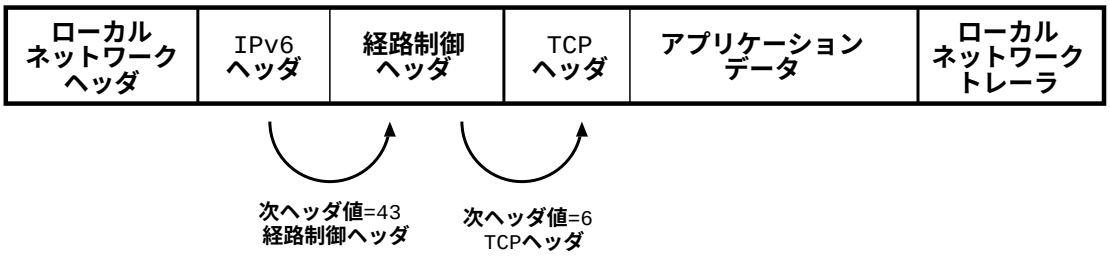


図 2.2: 次ヘッダフィールドによるヘッダの識別

- ペイロード長

  このフィールドは 16 ビットでオクテット単位で表したペイロード長を示している。このペイロード長は、IPv6 の基本ヘッダより後ろの部分を示しているため、拡張ヘッダや TCP や UDP などの上位層のプロトコルのヘッダサイズも含まれている。

- 次ヘッダ

  このフィールドは IPv6 ヘッダの直後に続くヘッダの種類を示すフィールドで、フィールド長は 8 ビットである。使用される値は IPv4 のフィールドと同じもので IPv6 固有なものなども追加されている。またこの次ヘッダに相当するものが拡張ヘッダの中にも存在し、図 2.2 のように、次に続くヘッダの識別子を順に記述する。

- ホップ制限

  このフィールドは、パケットがノードを通過するごとに 1 減少し、この値が 0 になると、このパケットは破棄される。

- 送信元 IPv6 アドレス

  このフィールドはパケットの送信元の IP アドレスを示しており、フィールド長は 128 ビットである。

- 送信先 IPv6 アドレス

このフィールドは、パケットの受取手のアドレスを示している。ここで示されている IP アドレスが必ずしも最終的なアドレスではなく、拡張ヘッダであるルーティングヘッダが使用されている場合は、次に到達すべき中継ノードを示している場合もある。

2.2.2 拡張ヘッダ

IPv6 では、IPv4 ヘッダの中で用いられていたオプションのフィールドを、拡張ヘッダとして定義し直すことにより構造を単純化している。これによって不必要な処理に起因するオーバーヘッドによってパケットの処理速度の低下を招くことはない。また構造が単純化しているにも関わらず、IPv4 より複雑な要求にも対応できるように設計されている。

以下にこの拡張ヘッダについて示す。

- オプションヘッダ (中継点、終点)

  中継点オプションヘッダ、終点オプションヘッダは、中継点のみまたは、終点でのみ作用するオプション情報を記述するヘッダである。

- ルーティングヘッダ

  このヘッダは、始点から終点への経路上のパケットが経由するノードのリストを含んでおり、パケットの通信経路を制御したいときに用いる。このルーティングヘッダを用いることで、アプリケーションからの能動的な経路制御が実現可能である。これについては第 3 章以降において詳しく説明する。

- 断片ヘッダ

  送信したいデータが巨大で、経路の MTU (Muximum Transmission Unit, 最大伝送単位) に収まらない場合、このヘッダを用いてパケットを分割することによりデータの送信可能にするためのヘッダである。

- 認証ヘッダ

  このヘッダは RFC2404 (Authentication Header)[7] で定義されており、現在提案されている認証ヘッダの機能は、パケット全体の完全性およびデータの認

証を付加するものである。また繰り返し行われる攻撃に対しての保護機能も提供している。

- 暗号ペイロードヘッダ

  このヘッダは、RFC2402 (Encapsulationg Security Payload)[8] で定義されており機密性、認証、整合性などを実現するよう設計されている。

- 次ヘッダなし

  ヘッダの次ヘッダフィールドの値が 59 のとき、そのヘッダがデータの最後となり、この後に続くヘッダは何も存在しない。このようなことからこのヘッダは、次ヘッダなし (No Next Header) と呼ばれている。

2.3 本章のまとめ

本章では、現在のインターネットの歴史的外観と、次世代インターネットプロトコル IPv6 の特徴について述べた。次章では、IPv6 の拡張ヘッダであるルーティングヘッダを利用した経路選択アプリケーションについて述べる。

第 3 章

経路選択アプリケーションの実現

3.1 経路選択アプリケーションの概要

本章では、実際に作成した経路選択アプリケーションの概要について述べる。

複数の通信経路の中からユーザが最適な経路を能動的に選択できる機能をアプリケーションに実装した。この経路選択の機能は、IPv6 の拡張ヘッダであるルーティングヘッダを用いて実現した。

この方法では、経由したいノード (ルータなど) の IPv6 アドレス (以下、アドレスと表記する) を、ルーティングヘッダを用いて指定し、パケットに付加して送り出すことによって経路を選択する。

本研究では、FTP アプリケーションにこの経路選択の機能を実現した。この FTP アプリケーションは、FTP における内部コマンドと転送パラメータコードを拡張することによって経路選択を実現している。

3.2 ルーティングヘッダによる経路選択機能

ルーティングヘッダは、IPv6 の拡張ヘッダとして定義されている。このヘッダの解釈と動作は、IPv6 の完全実装おいて必須である。よって通常のアプリケーションは、このヘッダを使用した経路選択を実現できる。

図 3.1 にこのルーティングヘッダのフォーマットを示す。

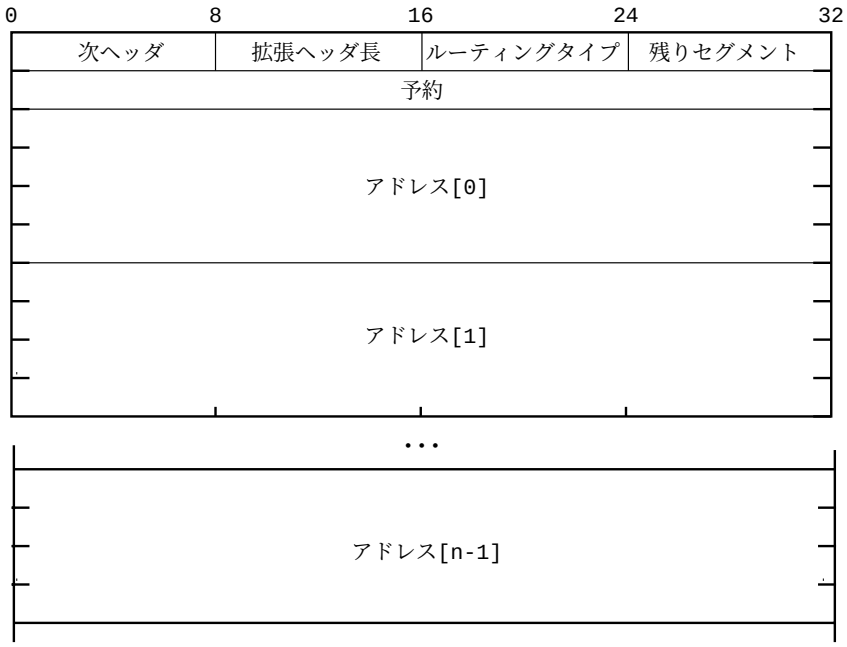


図 3.1: ルーティングヘッダのフォーマット

このヘッダに、経路上に存在するノードのアドレスを記述することで、通信の際にパケットが経由するノードを指定することができる。ヘッダ上での指定方法は、最初に経由するノードのアドレスを IPv6 の基本ヘッダの相手先アドレスとして記述し、その他の経由ノードと通信先のアドレスを経由する順に、図 3.1 に示した Address[1] の部分から順に記述する。

このヘッダを使用するための枠組みをアプリケーションに実装することによって、ユーザが通信品質を能動的に選択することが可能となる。

3.2.1 ルーティングヘッダのフォーマット

以下にルーティングヘッダの詳細について述べる。

- 次ヘッダ (Next Header)

  最初の次ヘッダフィールドには、ルーティングヘッダの直後にくる拡張ヘッダ、または、上位層プロトコルのヘッダの番号が格納される。

- 拡張ヘッダ長 (Hdr Ext Len)

  このフィールドには、拡張ヘッダ長が格納される。8 オクテットを単位としたときの拡張ヘッダ長を表し、かつ拡張ヘッダの最初の 8 オクテットを含まない。

- ルーティングタイプ (Routing Type)

  5 オクテット目以降のフォーマットを指定する識別子が定義される。今回、品質選択アプリケーションにおいて使用したのものは、タイプ 0 のルーティングヘッダである。

- 残りセグメント (Segments Left)

  ソースルーティングを処理してきて、拡張ヘッダ内にあるアドレスリストに示されているノードのうち、まだ通過していないノードがいくつあるかを表す。このフィールドは、ルータでソースルーティングを処理を行うたびに減らされる。残りセグメントの値が 0 になったら、何の意味ももたなくなる。

ルーティングヘッダの 5 オクテット目以降は、ルーティングタイプごとに定義された書式にしたがって用意される。図 3.1 はタイプ 0 の書式を表している。

- 予約 (Reserved)

  最初の 4 オクテットは予約フィールドであり、すべて 0 を設定しておく。ルーティングヘッダの処理では、単純に無視する。

- アドレス

  経由するルータのアドレスを順に記述するためのフィールドである。

3.2.1.1 従来のルーティングヘッダとの変更点

RFC1883 で公開されていたルーティングヘッダ (図 3.2) から、厳密/非厳密ビットマスクが削除された。これは厳密 (strict) 経路制御の必要性が疑問視されたこと、及びこのマスクにより経由できるノードに制限 (23 ノード) が生じるという 2 つの理由からである。これによりルーティングは、すべて非厳密ルーティング (指定したホストを順に通過するのみ) として扱われる。

図 3.2: ルーティングヘッダのフォーマット (変更前)

3.2.1.2 ルーティングヘッダの処理

複数の途中経路が指定された場合は、ヘッダを書き換えながら処理が進められる。表 3.1 は始点ホスト S から終点ホスト D に到達するまでに、途中経路として A、B、C の 3 つのホストを経由する場合の処理において、ヘッダがどのように書き換えられていくかを示したものである [9]。

ホップ	IPv6 ヘッダ		ルーティングヘッダ				
	始点	終点	Hdr Ext Len	Segments Left	Address[1]	Address[2]	Address[3]
S → A	S	A	6	3	B	C	D
A → B	S	B	6	2	A	C	D
B → C	S	C	6	1	A	B	D
C → D	S	D	6	0	A	B	C

表 3.1: ソースルーティング処理でのヘッダの書き換え

最初の行は、始点ホスト S で用意される IP データグラムの各ヘッダ値である。始点ホスト S では、通常の IP データグラム転送処理を行う。A に到達するまでのノードでは、ルーティングヘッダに関する処理は不要である。

ノード A に到達すると以下の処理を行う。

1. 拡張ヘッダの残りセグメント (Segments Left) の値をチェックする。この値は 0 ではないので、ソースルーティング処理を行う。

2. 拡張ヘッダの値から、ルーティングヘッダに格納されているアドレスの個数を得る。これを n とする。拡張ヘッダ長は 8 オクテット単位としており、IPv6 アドレスは 16 オクテットから構成されるため、アドレスの個数は、

$$n = HdrExtLen/2$$

   で得られる。表 3.1 の例では、n は 3 である。残りセグメントの値が n よりも大きければ IP データグラムを破棄し、ICMP Parameter Problem(Code 0) メッセージを生成する。

3. 残りセグメントの値を 1 つ減らす。

4. 次に、

$$i = n - Segments\_Left$$

   を計算し、次に到着すべきアドレスを得る。A では、$i = 1$ となる。

5. ルーティングヘッダの Address[i] と、IPv6 ヘッダの終点アドレスを入れ換える。A では、Address[1] と終点アドレスを入れ換える。

6. 次に IPv6 ヘッダのホップリミットフィールドの値が 1 であれば、ホップリミットの定義にしたがって IP データグラムを破棄し、ICMP Time Exceeded(Hop Limit Exceeded) メッセージを生成して始点ホスト S に返送する。ホップリミットフィールドの値が 1 でなければ、ホップリミットを 1 減らす。

7. 最後に新しい送信先に転送するためのモジュールにパケットを渡す。

この処理をルーティングヘッダで指定された経由ノードで順次実行していけば、ソースルーティングが実現できる。

3.2.2 IPv4 におけるソースルーティング

IPv6 のルーティングヘッダとほぼ同等な機能を提供する IPv4 にも存在した。これはソースルーティングオプションと呼ばれるオプションである。しかしながらこのオプションは、IPv4 の必須の機能ではないことも影響し、実際にはあまり使用されてはいない。

IPv4 におけるソースルーティングオプションにも、初期の IPv6 ルーティングヘッダと同じように厳密な経路制御 (図 3.3) と非厳密な経路制御 (図 3.4) が存在する。しかし初期の IPv6 ルーティングヘッダと違い、IPv4 のソースルーティングオプションは厳密、非厳密が別のオプションとして定義される。以下にこのオプションのフォーマットを示す。

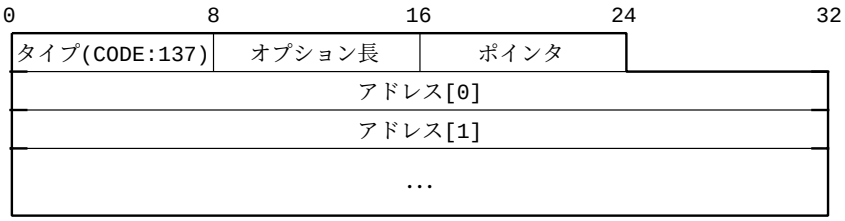


図 3.3: ソースルーティングオプションのフォーマット (厳密)

図 3.4: ソースルーティングオプションのフォーマット (非厳密)

- タイプ (Option type)

  オプションのタイプを示している。

- オプション長 (Option Length)

  ソースルーティングオプションの長さを示している。

- ポインタ (Pointer)

  次に処理されるべきアドレスに対するポインタである。IPv6 のソースルーティングヘッダにおける残りセグメント (Segments Left) と同様の働きをしている。

3.3 経路選択アプリケーションの実装

経路選択可能な FTP アプリケーションの開発には、KAME Project が公開する IPv6 プロトコルスタックを使用し、OS は FreeBSD を用いた。

FTP は通信を行う場合に制御コネクションおよびデータコネクションの 2 つのコネクションを用いる。またこの 2 つのコネクションに対して、それぞれ送信および受信の合計 4 つのアクションが存在する。そのため、4 つの通信経路に対して、経路選択を可能とした。

3.3.1 送信経路の選択

送信経路の選択に対するユーザインタフェースとして、FTP クライアントの内部コマンド “putroute” を定義した。このコマンドは、FTP クライアントから FTP サーバに送信するパケットの通信経路の選択、閲覧を行うコマンドである。

表 3.2 に “putroute” コマンドの仕様を示す。“putroute” コマンドにおける引数の “ctl” および “data” はそれぞれコントロールコネクション、データコネクションの送信経路に関するアクションを示している。これらの引数の指定が無い場合は、2 つのコネクションの両方を指定したものとする。

“経路” は経由するノードの名前を、@を区切り文字として順に記述する。例えばルータ “v6router1” 、 “v6router2” 、 “v6router4” を経由して通信を行う場合は、

経由ノードの指定方法

“@v6router1@v6router2@v6router4”

と記述する。引数の “on” および “off” は、それぞれ設定した経路を一時的に有効、無効にするものである。指定がない場合は、設定している経路情報を表示する。

表 3.2: putroute コマンドの仕様

コマンド			動作および状態		
コマンド	引数 1	引数 2	動作	制御	データ
putroute	-	-	表示	○	○
putroute	(経路)	-	設定	○	○
putroute	ctl	-	表示	○	-
putroute	data	-	表示	-	○
putroute	off	-	設定	○	○
putroute	on	-	設定	○	○
putroute	ctl	(経路)	設定	○	-
putroute	ctl	off	設定	○	-
putroute	ctl	on	設定	○	-
putroute	data	(経路)	設定	-	○
putroute	data	off	設定	-	○
putroute	data	on	設定	-	○
help	putroute	-	コマンドヘルプ		

FTP クライアントは、ユーザから指定された通信経路の情報から、ルーティングヘッダをパケットに付加し、データを送り出す。

3.3.2 受信経路の選択

受信経路の選択に対するユーザインタフェースとして、送信時と同様に内部コマンド “getroute” を定義した。このコマンドは、FTP クライアントが FTP サーバから受信するパケットの経路選択、閲覧を行うコマンドである。表 3.3 に “getroute” コマンドの仕様を示す。このコマンドにより、制御コネクションおよびデータコネクションのパケットの受信経路の選択、経路情報の参照が可能となる。

しかし、受信時におけるパケットは FTP サーバより送信されるため、受信の経路は FTP サーバがデータ送信時にルーティングヘッダによって指定しなければならない。そのため FTP クライアントにおいて “getroute” コマンドを用いてユーザから指定された経路情報を、FTP サーバに送信する必要がある。

そこで本アプリケーションでは、FTP において使用される転送パラメータコード

表 3.3: getroute コマンドの仕様

コマンド			動作および状態		
コマンド	引数 1	引数 2	動作	制御	データ
getroute	-	-	表示	○	○
getroute	(経路)	-	設定	○	○
getroute	ctl	-	表示	○	-
getroute	data	-	表示	-	○
getroute	off	-	設定	○	○
getroute	on	-	設定	○	○
getroute	ctl	(経路)	設定	○	-
getroute	ctl	off	設定	○	-
getroute	ctl	on	設定	○	-
getroute	data	(経路)	設定	-	○
getroute	data	off	設定	-	○
getroute	data	on	設定	-	○
help	getroute	-	コマンドヘルプ		

を拡張し、“CROUTE” および “DROUTE” の 2 つの転送パラメータコードを定義した。表 3.4 にこのコマンドの仕様を示す。

表 3.4: 経路設定に関する転送パラメータコード

コード	引数	経路動作内容
CROUTE	SHOW	制御コネクションの表示
CROUTE	ON	制御コネクションの制御 ON
CROUTE	OFF	制御コネクションの制御 OFF
CROUTE	(経路)	制御コネクションの設定
DROUTE	SHOW	データコネクションの表示
DROUTE	ON	データコネクションの制御 ON
DROUTE	OFF	データコネクションの制御 OFF
DROUTE	(経路)	データコネクションの設定
HELP	CROUTE	コマンドヘルプ内容の要求
HELP	DROUTE	コマンドヘルプ内容の要求

“CROUTE” は、制御コネクションの受信経路に関するパラメータであり、“DROUTE”

は、データコネクションの受信経路に関するパラメータである。引数 “SHOW” は設定されている経路情報の参照、 “ON” および “OFF” は一時的な経路指定の有効、無効を設定するものである。 “経路” は、内部コマンドおいて経路を選択する形式と同様に “@” を用いて記述する。

FTP サーバ上では、クライアントから送信された転送パラメータコードをもとに経路の設定を行い、その結果を FTP クライアントに応答する。

この応答のために、FTP プロトコルで定義されている応答コードを利用し、経路選択に関するパラメータに対して、経路情報の設定の成功および失敗の 2 つを応答した。経路情報の設定が成功した場合は応答コードとして “200” および、設定した経路情報をメッセージとして FTP クライアントに送信する。経路設定に失敗した場合はコードとして “501” および、設定に失敗した経路情報をメッセージとして FTP クライアントに送信する。

図 3.5 にデータ受信における経路設定の際のクライアントおよびサーバの動作を示す。ユーザが “getroute” コマンドで受信経路の設定を行うと (図 3.5 ①)、FTP クライアントは、その経路情報を転送パラメータコードを用いて FTP サーバに送信する (図 3.5 ②)。最後に FTP サーバは、その経路設定の結果を応答コードを用いて FTP クライアントに応答する (図 3.5 ③)。

また図 3.6 は、実際に上記のコマンドを用い、データコネクションにおける受信の経路を指定した際の FTP の出力例である。この例では、データの受信時における経路のみ “v6router1” 及び “v6router2” の 2 つのルータを経由するように指定している。

3.4 本章のまとめと議論

本章では、複数の通信経路の中からユーザが最適な経路を能動的に選択可能な機能を実装したアプリケーションについて述べた。この経路選択の機能は、IPv6 の拡張ヘッダであるルーティングヘッダを用いて実現した。

このアプリケーションでは、経路を選択する際に、通信回線の始点ノードを明示的に指定する必要があるため、通信経路の品質の把握していない一般のネットワーク利用者に対しては、ユーザが望む品質の経路をユーザに提示するなどの支援を行

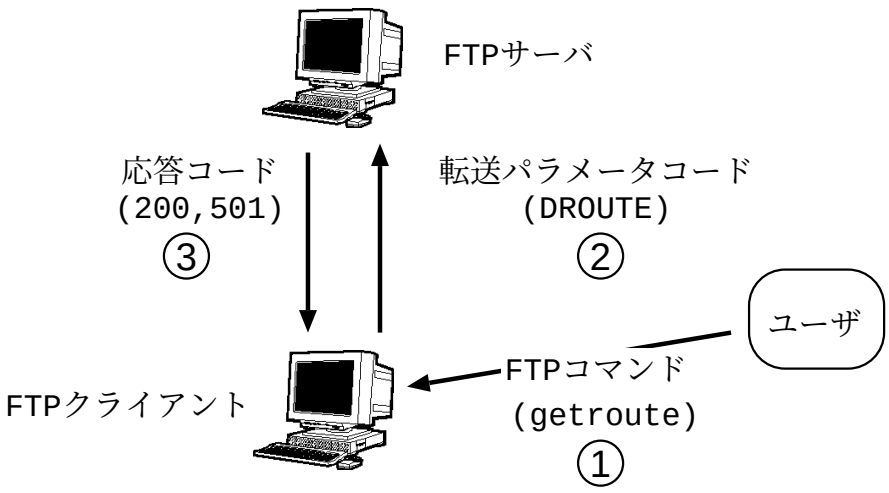


図 3.5: データ受信経路設定時の動作

```
% ftp v6host1
Connected to v6host1
220 v6host1 FTP server (Version 6.00) ready.
Name (otani):
331 Password required for user
Password:
230 User otani logged in.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> getroute data @v6router1@v6router2
200 DROUTE set data route ->
          '@v6router1@v6router2' command ok.
ftp>
```

図 3.6: FTP の経路指定例

う必要がある。そこで、この支援を行う品質選択支援システムの開発も行った。次章では、この品質選択支援システムについて述べる。

第 4 章

品質選択支援システム

経路選択アプリケーションで、ユーザは能動的に経路の選択が可能となった。しかしこの方法は、経路を選択する際に、通信回線の始点ノードを明示的に指定する必要があるため、通信経路の品質の把握していない一般のネットワーク利用者に対しては、ユーザが望む品質の経路をユーザに提示するなどの支援を行う必要がある。そこで、ユーザに対する品質選択の支援を目的とした “品質選択支援システム” の試作もあわせて行った。この品質選択支援システムは、ネットワークの管理者が把握している範囲の品質情報をユーザに提供するものである。ユーザは、品質選択支援システムを利用することにより、ネットワークの構成を把握することなく通信品質の選択が可能となる。

本章では、この品質選択支援システムの機能や、実装の概要について述べる。

4.1 品質選択支援システムの機能

品質選択支援システムは、通信回線の品質を把握していないユーザに対して、通信回線の品質情報を提供し、通信回線の選択の支援を行うシステムである。現バージョンの品質選択支援システムでは、ネットワーク管理者が把握している範囲の品質情報を、管理者自身が静的に設定する。設定内容は、通信元および通信先のアドレス、回線品質、その回線に至るまでの経路、アプリケーションの種類といった情

報である。ユーザは、経路選択アプリケーションを使用し、経路選択支援システムから提供される品質情報の中で、自分の利用目的にあった品質を選択することにより、通信品質を選択する。

4.2 品質選択支援システムの実装

品質選択支援システムを、FreeBSD 上で開発を行った。品質選択支援システムは、経路選択アプリケーションと TCP/IP を用いて情報の交換を行い、以下の手順で動作する。(1) 経路選択アプリケーションは、品質情報を取得するために、送信元および送信先アドレス、アプリケーションの情報を、品質選択支援システムに送信する。(2) 品質選択支援システムは、使用可能な品質の情報を経路選択アプリケーションに応答する。(3) その中からユーザが利用したい通信品質を選択すると品質選択アプリケーションは、それを品質支援システムに送信する。(4) 品質選択支援システムは、その通信品質を満たす経路を応答する。

品質選択支援システムの設定例を以下に示す。例えば、図 4.1 のようなネットワークが存在した場合、ネットワーク管理者は表 4.1 に示すような品質情報を品質選択支援システムに設定する。また通信元および通信先には、ネットワークアドレスを記述することが可能である。

表 4.1: 品質選択支援システムの設定例

通信元	通信先	品質	経路	アプリケーション
2001:200:160:1::/64	2001:200:160:2::/64	broadband	@router1@router2	FTP
2001:200:160:1::/64	2001:200:160:2::/64	secure	@router3	SMTP

4.3 節ではユーザが通信品質を選択した際のシステムの動作を、4.3.1 節では経路選択アプリケーションの拡張について述べる。

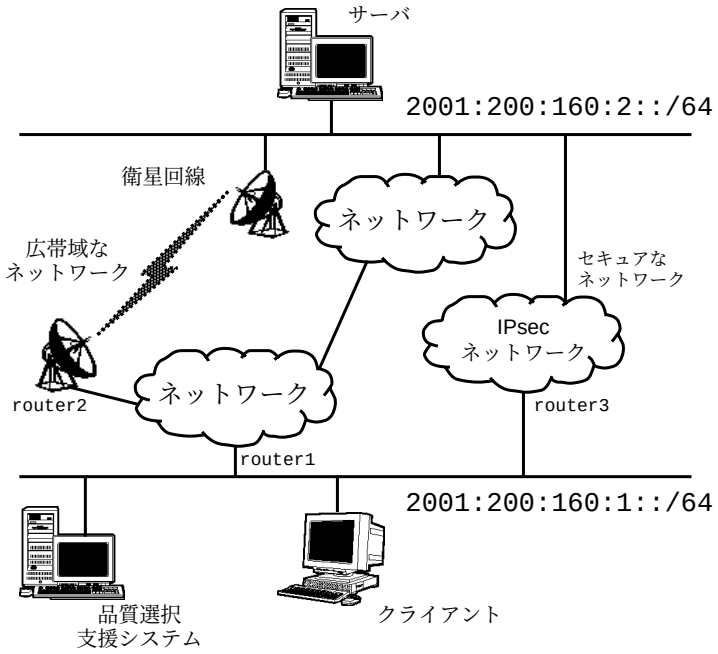


図 4.1: ネットワークの構成例

4.3 品質選択支援システムの動作

図 4.2 に品質選択支援システムを用いて通信を行った例を示す。この例では、“host1” から “host3” に対して FTP を行う際に、送信経路は高速な通信経路、受信経路は広帯域の通信経路を通るような経路を品質選択支援システムに問い合わせて、通信を行った例である。まず FTP クライアントは、ユーザから指定された品質の要求と、通信先の情報、アプリケーション情報を品質選択支援システムに送信する (図 4.2 ①)。次に品質選択支援システムは、要求された品質に合わせた経路情報を応答する (図 4.2 ②)。FTP クライアントは、受け取った経路情報を使用して、通信を行う (図 4.2 ③)。

4.3.1 FTP クライアントの拡張

品質選択支援システムを利用するためには、FTP クライアントにおいて、品質選択支援システムから経路情報を取得する機構を実装する必要がある。そこで、第 3.3 節で述べた “putroute”、“getroute” コマンドを、更に拡張した (表 4.2)。

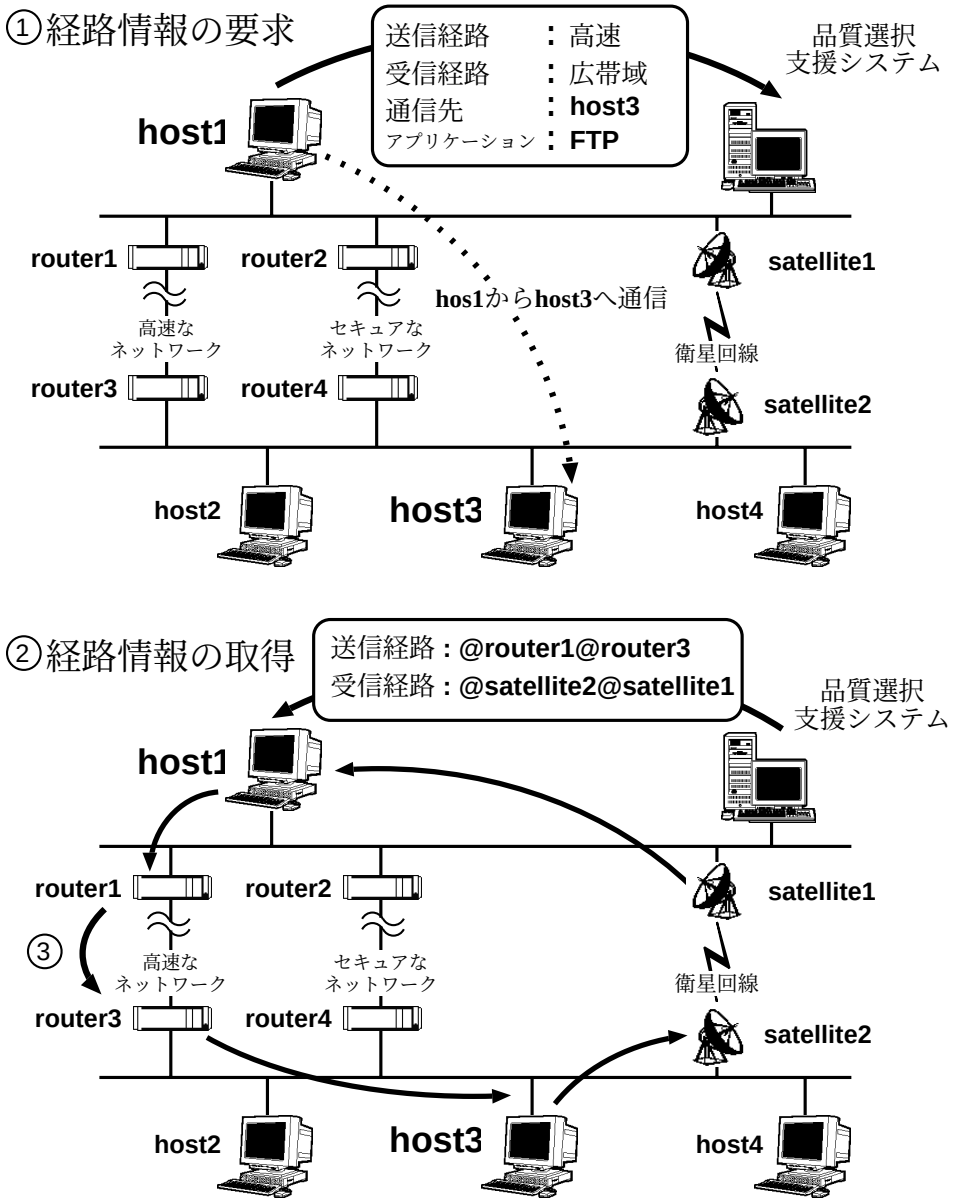


図 4.2: 品質選択支援システムを用いた通信例

品質選択支援システムは、この 2 つのコマンドに “server” 引数が渡されると、クライアントから自動的に送信される通信先ホスト、アプリケーション名の情報から適切な経路情報を判断し、その情報をクライアントに応答する。また引数に “品質” (例えば minimum delay など) を加えると、この条件に適合する経路情報を応答する。

また “qput”、“qget” コマンドは、要求した品質を満たす経路が応答された後、それを受け取った FTP クライアントが直ちに目的のファイルの送受信を行う。

表 4.2: 品質選択支援システムのためのコマンド拡張

コマンド				動作および
コマンド	引数 1	引数 2	引数 3	状態
putroute	server	-	-	設定
getroute	server	-	-	設定
putroute	server	(品質)	-	設定
getroute	server	(品質)	-	設定
putroute	ctl	server	(品質)	設定
getroute	ctl	server	(品質)	設定
putroute	data	server	(品質)	設定
getroute	data	server	(品質)	設定
qput	(品質)	(filename)	[(filename)]	設定, 送信
qget	(品質)	(filename)	[(filename)]	設定, 受信

4.4 本章のまとめと議論

品質選択アプリケーションは、経路を選択する際に通信回線の始点ノードを明示的に指定する必要があるため、通信経路の品質の把握していない一般のネットワーク利用者に対しては、ユーザが望む品質の経路をユーザに提示するなどの支援を行う必要がある。そこで、この支援を行う品質選択支援システムの開発も行った。この品質選択支援システムは、ネットワークの管理者が把握している範囲の品質情報をユーザに提供するものである。ユーザは、品質選択支援システムを利用することによりネットワークの構成を把握することなく通信品質の選択が可能となる。

このシステムは、現在、ネットワークの管理者が把握している範囲の品質情報を

静的に設定する必要があるが、他の品質管理技術と強調して動作することにより、動的な品質情報を提供することも可能である。これについては、第 6 章において述べる。

第 5 章

検証実験

第 3 章で述べた経路選択アプリケーションが正確に経路の指定が実現できていることを、パケットのモニタリングを行い確認するとともに、帯域の異なる通信経路を持つ実験ネットワークを構築し、経路選択の有用性を検証した。また、ルーティングヘッダを使用する際のオーバーヘッドについての検証も行った。

5.1 経路選択の有用性

作成したアプリケーションの経路選択の有用性を検証するために図 5.1 に示すネットワークを構築した。このネットワークは、FTP クライアントとサーバの間には 3 つのルータが存在し、通信帯域が 10Mbps および 100Mbps という 2 つの通信経路を作成している。ここでデフォルトの経路として 10Mbps の通信経路を設定し、本アプリケーションを使用して経路制御を行った場合にのみ 100Mbps の通信経路を使用可能な構成とした。

図 5.2 に、実験ネットワークにおいて、10Mbps の通信経路を使用した通信と、本システムを用いて経路指定した通信のスループットの違いを示す。試行回数はそれぞれ 10 回である。

図 5.2 が示すように、デフォルトの経路ではなく、経路選択アプリケーションを使用し広帯域の通信経路を選択した方がスループットの値が大きいことが分かる。こ

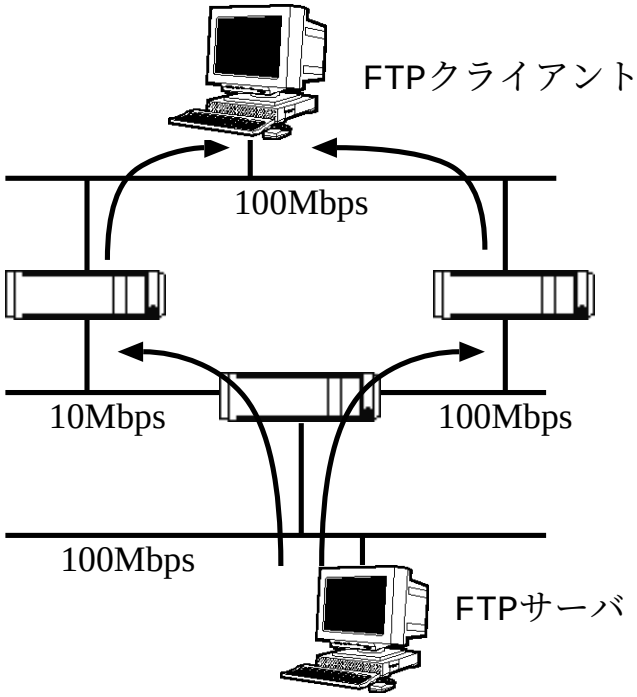


図 5.1: 実験ネットワークの構成図

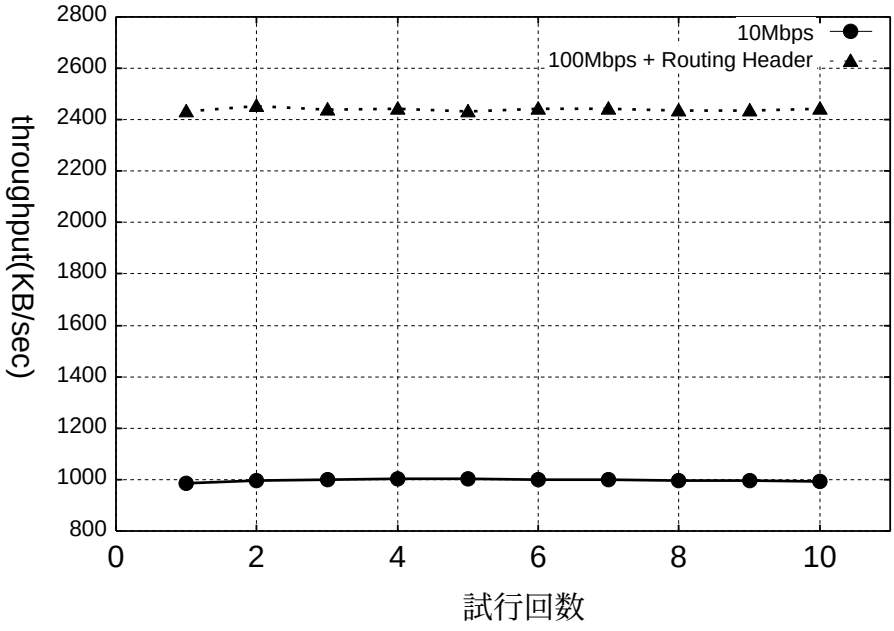


図 5.2: スループットの比較

の結果は、経路選択アプリケーションにおいて、ユーザからの経路選択が正確に機能していることを示している。

次にプロトコルアナライザによって、ヘッダ上に指定した経路を示すルーティングヘッダが付加されているかの確認を行った。図 5.3 に 10Mbps の経路を指定した場合の FTP クライアントが受信したパケットの 1 つを示す。

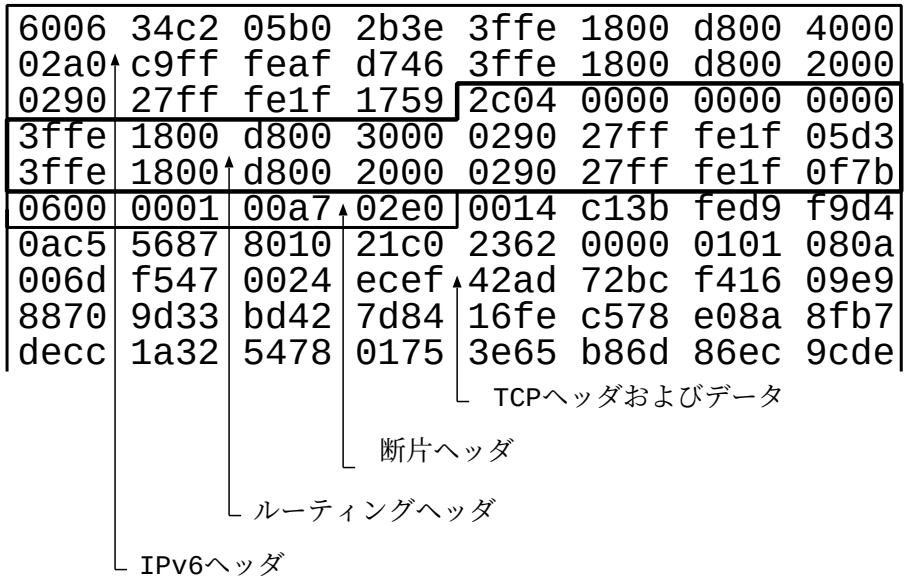


図 5.3: ルーティングヘッダが付加された IPv6 パケット

図のように、IPv6 ヘッダ、ルーティングヘッダ、断片ヘッダ、TCP ヘッダおよびデータが順に並んでいることが確認できる。

ここでルーティングヘッダの部分のみを見ると、以下のようになっている。

ルーティングヘッダの部分

```
                              2c04 0000 0000 0000
    3ffe 1800 d800 3000 0290 27ff fe1f 05d3
    3ffe 1800 d800 2000 0290 27ff fe1f 0f7b
```

まず次ヘッダとして、断片ヘッダの識別である “2c” が指定されている。またその次に、拡張ヘッダ長として”04” が指定されている。これは、8 オクテット単位で記述されるため、経由ノードが 2 つ指定されていることが分かる。そしてその次の 48

ビットの “0” は、それぞれ経路制御タイプ、中継点残数、予約を示しており、正しく指定されている。 そして残りの 256 ビットの部分は、中継ノードであるルータのアドレスを示している。

これより正しくルーティングヘッダが付加されていると確認できた。

5.2 ルーティングヘッダのオーバーヘッド

経路を指定してデータを送信する場合、ルーティングヘッダが拡張ヘッダとして IPv6 ヘッダに付加され、各ルータやエンドノードで処理される。このため、ルーティングヘッダを用いない場合と比べ、若干のオーバーヘッドが生じると考えられる。そこで本アプリケーションを用いて、通信帯域や経路指定数などの違いによる転送時間の計測を行った。

図 5.4 および表 5.1 に、100Mbps イーサネットのネットワークにおいて、50MB のデータを受信経路のみ経路制御を行って受信した際の、経路指定数の違いによる転送時間と、その平均を示す。試行回数はそれぞれ 10 回である。

表 5.1: 経路指定数の違いによる転送時間の変化

経路指定数	平均転送時間 (秒)
0	20.28
1	20.93
2	20.98
3	21.11
4	21.17

この計測結果から、ルーティングヘッダの有無が、転送時間に平均 0.65 秒、経由するルータが 1 つ増加するごとに平均 0.08 秒程度の影響しか与えていないことが分かる。よってルーティングヘッダのオーバーヘッドは、データの転送時間と比べ十分小さな値であるといえる。

またその他の通信帯域においても同様の検証を行った。図 5.5 に同じく 100Mbps の経路を指定しデータを送信した際の結果を示す。

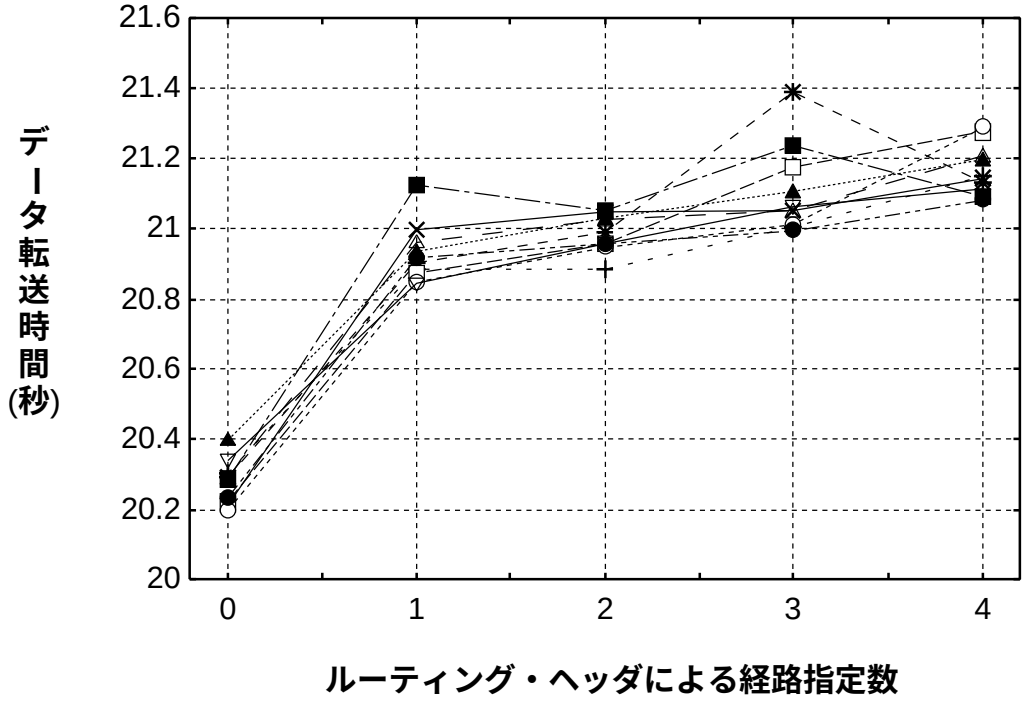


図 5.4: 100Mbps でのデータ受信

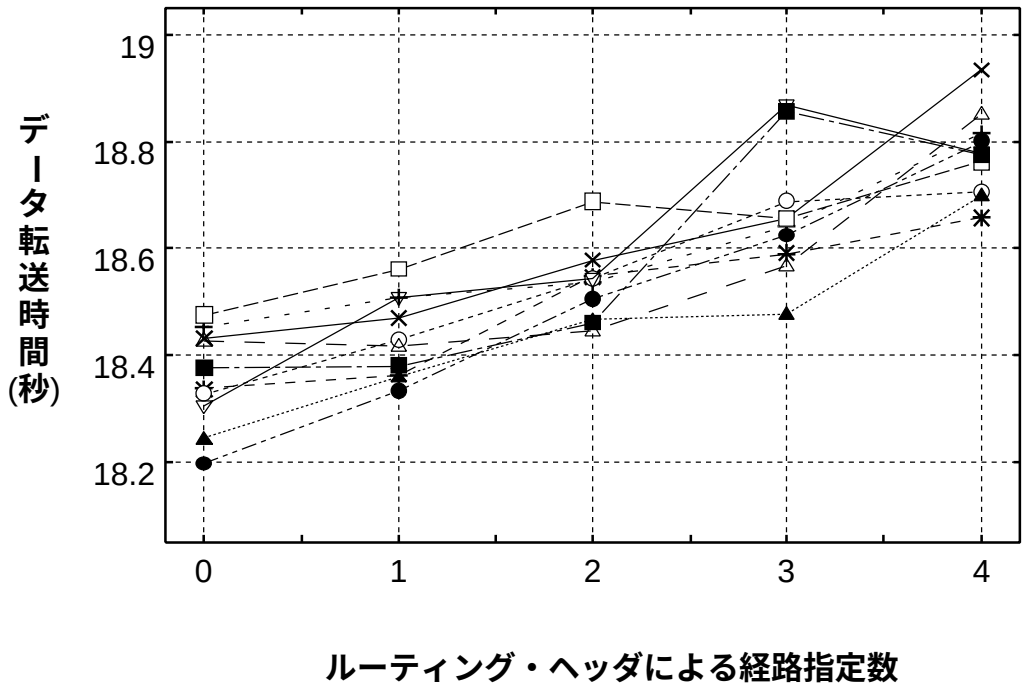


図 5.5: 100Mbps でのデータ送信

100Mbps の経路を指定した際のデータ送信においても、少し特性が異なるが受信の時と同様の結果が得られた。

また同様に、10Mbps の経路を指定した場合の検証実験も同様に行った。受信における結果を図 5.6 に、送信時における結果を図 5.7 に示す。

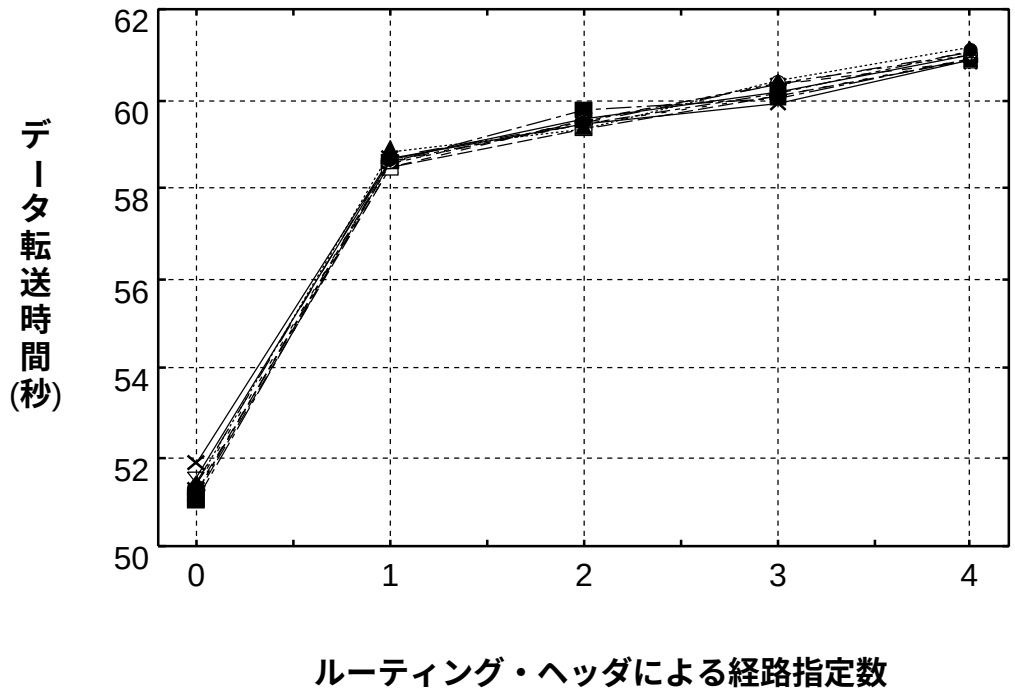


図 5.6: 10Mbps でのデータ受信

5.3 本章のまとめと議論

経路選択アプリケーションが正確に経路の指定が実現できていることを、パケットのモニタリングを行い確認するとともに、帯域の異なる通信経路を持つ実験ネットワークを構築し、経路選択の有用性を検証した。また、ルーティングヘッダを使用する際のオーバーヘッドについての検証も行った。

5.1 および 5.2 節で述べた検証により、アプリケーションからの能動的な経路選択の方法において、ルーティングヘッダを用いることは有効な手法であると考える。またルーティングヘッダの影響は、通信回線の帯域が多くなるにつれて減少するこ

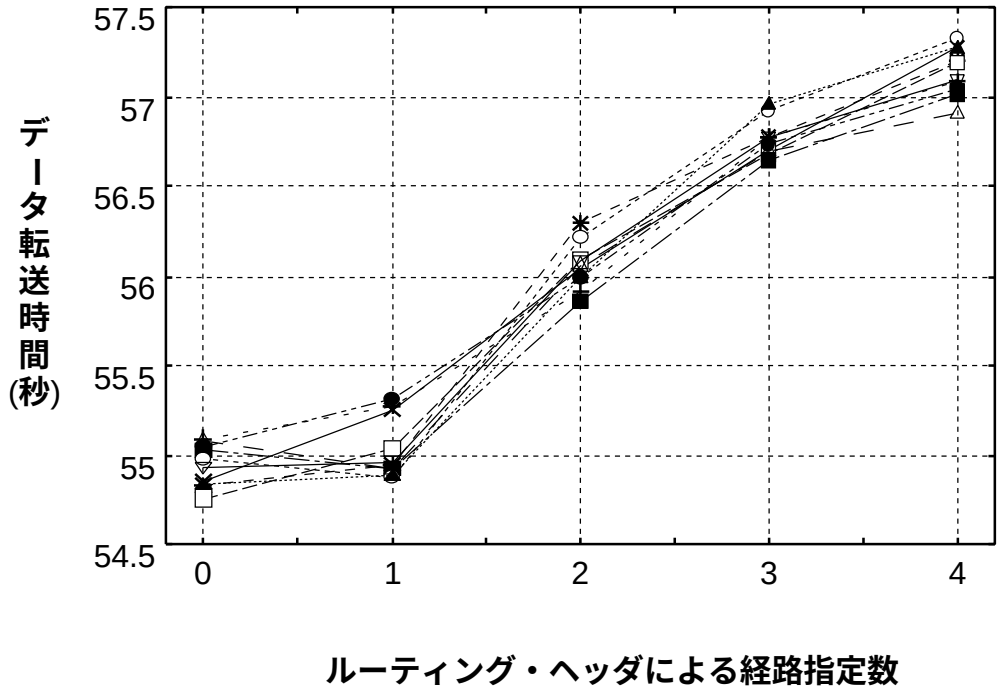


図 5.7: 10Mbps でのデータ送信

とも確認できた。これら結果から、ルーティングヘッダが通信に与える影響は、今後のネットワークの高速化も考えると十分許容できるものであると言える。

しかし今回検証を行ったネットワークは、すべて PC ルータを使用した。この PC ルータは、ルーティングヘッダを含めたパケットの処理を、すべてソフトウェアで処理している。一方、近年のネットワークの大容量化に伴い、パケットの処理の一部、またはすべてをハードウェアで行う高速ルータ専用機も登場している。このようなルータ専用機が、拡張ヘッダの付加されていないパケットをハードウェアで高速に処理し、拡張ヘッダであるルーティングヘッダを含むパケットをソフトウェアで処理するといった場合、PC ルータと転送速度等に差が生じる可能性がある。

第 6 章

考察

6.1 本研究の位置

近年、インターネットの実用性と配送の信頼性を高めるために、特定のユーザやアプリケーションのデータ配送ポリシーを選択的に保証するメカニズムに関する研究が広く進められている。そのアプローチの方法は、品質指標 (帯域、遅延/ジッタ、損失など) を制御するレベルに基づいて大きく 3 つに分類することができる [10]。

- ベストエフォート型サービス

  パケットが宛先に到着するかどうか、そして到達する場合そのタイミングなどに関して一切保証しない基本接続である。このベストエフォート型トラフィック転送においては、サービスや伝送が保証されないため、厳密には QoS サービスとはいえない (FTP などの大半のデータアプリケーションは、パフォーマンスの低下はあってもベストエフォート型で適切に動作する。しかし効率よく転送するためには、すべてのアプリケーションは、帯域幅、遅延、最小パケット損失の面である程度のネットワークの割り振りを必要とする)。

- 差別化サービス

  差別化サービスでは、トラフィックはそのサービス要件に基づいてクラスにグループ化される。それぞれのトラフィッククラスはネットワークによって

差別化され、クラスに対して設定されたQoSに応じたサービスが提供される。QoSを提供するこのスキームは、一般に「CoS」と呼ばれる。

差別化サービスを提供する技術の1つであるDiffServ(Differentiated Services)は、資源の管理ポリシーをもとに、ネットワークトラフィックを差別化しし、各クラスに帯域などのネットワーク資源を割り当てる。高いサービス品質を求めるアプリケーションには優先順位の高いクラスを割り当てることにより、大まかなQoS制御を行うものである。

DiffServ自体はサービスを保証せず、トラフィックの差別化のみを行い、あるトラフィッククラスを他のトラフィッククラスより優先的に処理することを可能にする。この理由から、このサービスは「ソフトQoS」とも呼ばれる。

差別化サービスとしては、その他に米パケッティア社のPacketShaperといったものがある。PacketShaperは、DiffServと同様にネットワークのトラフィックを監視し、パケットの優先度を管理することによって、各アプリケーションが使うネットワークの帯域を制御するものである。

このQoSスキームは、多くの帯域を必要とするデータアプリケーションに適している。基本的なネットワーク接続を常時保証するためには、ネットワーク制御トラフィックを他のデータトラフィックから差別化して優先することが重要である。

- 保証サービス

ネットワークがトラフィックフローの特定のサービス要件を満たすことを保証するためにネットワークリソースの予約を必要とするサービスである。保証サービスでは、接続パスに対してネットワークリソースを事前に予約しておく必要がある。保証サービスはネットワークからの厳密な保証を必要とするため「ハードQoS」とも呼ばれる。

単一フローごとにパスを予約する方法は、数千ものフローに同時にサービスを提供するインターネットバックボーンに対してはスケーラビリティ上の問題がある。しかし、インターネットのコアルータ内の最小限の状態情報しか必要としない集合予約は、サービスを提供するスケーラブルな手段となる。

このようなサービスを必要とするアプリケーションとしては、音声や映像などのマルチメディアアプリケーションがある。インターネット上の対話型音声アプリケーションでは、人間工学上のニーズを満たすためのサービス要件 (ラウンドトリップ遅延、最低限の帯域など) を満たせるようにリソースを予約する必要がある。

一方、本研究が提案する方法は、途中の経路を指定することにより、通信回線の品質を提供しようとするものである。一般にインターネットにおいて、あるサイトからあるサイトまでデフォルトで設定される経路が、ユーザに最適な品質を与える経路であるとは限らない。本研究の提案する方法は、このような場合に複数の経路の中からユーザ自身が経路を選択することにより、ユーザとアプリケーションにとって最適な品質の選択を可能にしようとするものである。この方法では、例えば図 6.1 のように、あるサイトからあるサイトの間に部分的に DiffServ や PacketShaper になどのよって管理された通信回線が存在した場合に、その回線を部分として含む経路を選択し、通信元から通信先までの品質を与えるのに利用することができる。すなわち本研究の提案する方法は、DiffServ などによって提供される品質の回線を部分として包含し、ユーザにピア・ツー・ピアな回線品質の提供を目指すものである。

6.2 他の研究との関連

その他に配送ポリシーを一元的に管理する COPS (Common Open Policy Service) の研究も進められている。この COPS は一元的に管理されたポリシー情報と各ノード間でポリシー制御情報を交換をするプロトコルで、ノードからの帯域予約要求などの問合せに対して、許可・拒否の判断を行うものである。本研究で試作した品質選択支援システムは、通信経路の品質情報を保持しその情報をユーザに提供する。今後は、この品質選択支援システムと COPS を連携し、配送ポリシーの管理を行うことも可能であると考えられる。また品質の選択を行う際に、利用料金の必要な回線を選択する可能性が考えられる。例えば、ある高速なネットワークを特定の時間のみ料金を支払うことによって使用可能になるといった課金のモデルが存在した場合、技術的には、このモデルに対応するためのプロトコルを品質選択支援システムに組み込むことによって対応可能であると考えられる。

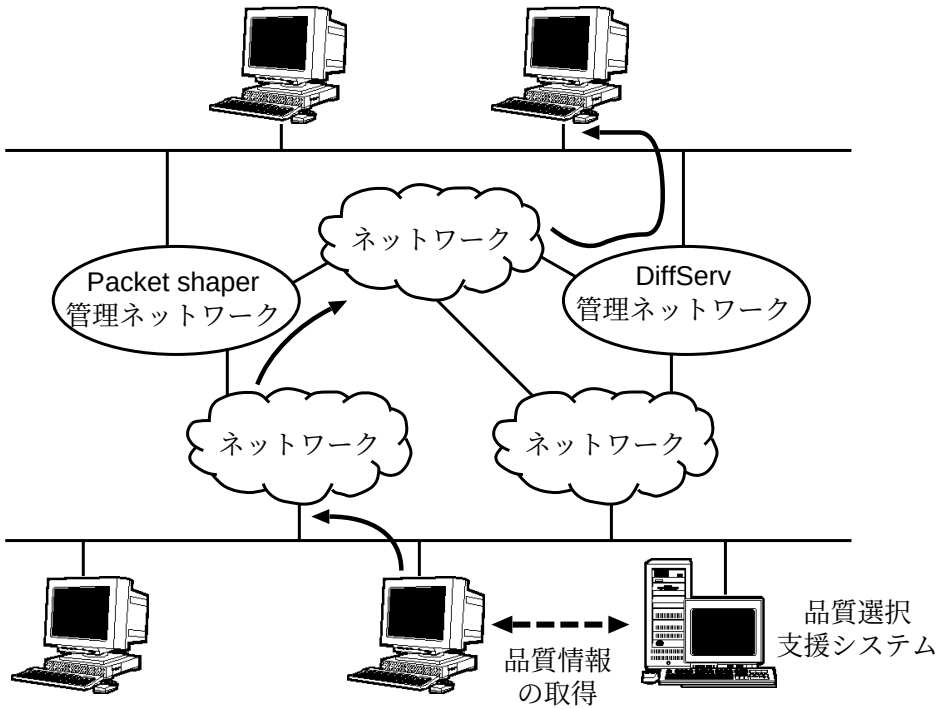


図 6.1: ポリシーの管理されたネットワークでの利用

一方、ユーザからの経路選択のアプローチとして、IETF (Internet Engineering Task Force) の SDR (Souce Demand Routing) 分科会において研究が行われ、SDRP (Souce Demand Routing Routing) と呼ばれるプロトコルが定義されている。このプロトコルは、インターネットの制御ドメイン間で、管理ポリシーに依存する経路制御を行うプロトコルである。このプロトコルは、ルーティングヘッダと同様に、経由ノードのアドレスリストを SDRP の経路制御ヘッダに記述することによって経路の制御を行うことが可能である。また SDRP の経路制御ヘッダに多くのポリシー制御の情報を含めることも可能で、より複雑な経路制御が可能である。IPv6 のルーティングヘッダは、タイプフィールドの値を指定することによって様々な経路制御方式に対応が可能になっているが、現在の所、SDRP は採用されていない。本研究では、ルーティングヘッダのタイプフィールド “0” として定義されているヘッダに経由ノードのアドレスを記述し経路指定を行う方法を用い、品質選択を実現した。

本研究では、品質選択が可能なアプリケーションとして FTP を用いて検証を行ったが、現在のインターネットの利用状況を見ると、電子メールや、WWW (World

Wide Web) や VoIP (Voice over IP) などのストリーム系のアプリケーションなどについても考える必要がある。近年、電子メールや WWW などのアプリケーションはセキュリティ、ストリーム系のアプリケーションは帯域を必要とするようになってきており、経路による品質の選択は有効であると考えられる。実装については、個々のアプリケーションに機能を組み込むことも考えられるが、プロキシサーバを作成し、ルーティングヘッダを付加することも可能であると考える。

第 7 章

おわりに

7.1 本論文のまとめ

インターネットは社会基盤として、企業や学校、一般家庭に急速に普及してきた。この普及とともに、ユーザのインターネットの利用目的が多様化し、WWW やメール、インターネット電話やビデオ会議、などにも日常的に利用されるようになった。このようにインターネットが日常的に利用されるにつれ、現在のインターネットの問題点がいくつか指摘されるようになった。たとえば、アドレス空間の枯渇、セキュリティの確保、NAT に代表されるようなアドレス変換技術などによって利用を制限されるピア・ツー・ピア通信などである。またこれらの問題点とともに、インターネットを使用するユーザからの、セキュリティや通信帯域の保証などといった、通信品質に対する要望も生じ始めている。

よって近い将来のインターネットにおいては、インターネットの利用目的に応じた通信品質を選択する方法の実現が重要な課題と考えられる。

また、プロバイダなどから提供される通信回線が、帯域やセキュリティのレベルなどの違いにより多様化し、ユーザがその通信回線を選択できるような環境が整いつつある。

本研究では、このような背景から、通信品質を選択する方法として、インターネットを使用するユーザが、インターネットの途中の経路を指定することにより、利用

目的に応じた通信の品質を選択できる方法の提案を行った。この方法で、通信先までに存在する品質の異なる回線の中から、使用したい品質の回線を経由するように経路を指定することで、通信品質の選択を実現する。

この品質選択の方法において、次世代のインターネットプロトコルである IPv6 のルーティングヘッダを使用した。このヘッダを利用するための枠組みをアプリケーションに実装し、検証を行った。また品質選択の支援を目的として、ユーザに通信回線の情報を提供し品質選択の支援を行う、“品質選択支援システム” の開発も行った。

先に述べたように、日本では、政府の「e-japan 重点計画」においても、“ 2005 年 (平成 17 年) までにすべての国民が、場所を問わず、自分の望む情報の入手・処理・発信を安全・迅速・簡単に行える IPv6 が実装されたインターネット環境を実現する ” と述べられており、現在、IPv6 対応機器を導入する企業に対して、財政優遇策や低利融資制度などの拡大を行うことも検討されている。このような背景から IPv6 は今後、インターネットの主要なプロトコルとなり、現在の IPv4 ネットワークが IPv6 へと移行していくことは、ほぼ間違いはない。よって本研究は、今後のインターネットの発展に大いに寄与するものと期待できる。

謝辞

研究活動ならびに本論文の執筆において、多岐にわたり多大なるご指導、ご鞭撻を賜りました佐賀大学理工学部知能情報システム学科の近藤 弘樹 教授ならびに渡辺 健次 助教授に、厚くお礼申し上げます。

本論文の内容についてご指導頂いた、佐賀大学学術情報処理センターの只木 進一 教授、佐賀大学理工学部知能情報システム学科の林田 行雄 教授に心より感謝申し上げます。

研究活動において、ご指導ならびにご意見を頂きました佐賀大学理工学部知能情報システム学科の岡崎 泰久 助手、田中 久治 技官、佐賀大学学術情報処理センターの日永田 泰啓 助教授に、心より感謝申し上げます。

研究活動において、ご意見ならびにご協力を頂いた、(株) デジタルコミュニケーションズ佐賀の西村 龍一郎氏、田尻 博伸氏、溝口 正昭氏、緒方 俊彦氏に心より感謝申し上げます。

本研究を通じて、様々な議論、実験等にご協力頂いた、佐賀大学理工学部知能情報システム学科第 5 研究グループの学生の皆様に深くお礼申し上げます。

この他にも、研究活動全般において、非常に多くの方々からご協力、ご助言を戴いております。筆者の活動に関係するすべての方々に、ここに記して心より感謝の意を表します。

参考文献

[1] Christian Huitema, 村井純 監修, WIDE プロジェクト IPv6 分科会監訳, 松島英樹 訳, IPv6 次世代インターネットプロトコル, プレンティスホール出版, (Jan. 1997)

[2] S. Bradner and A. Mankin, IP: Next Generation (IPng) White Paper Solicitation, Request for Comments 1550, (Dec. 1993)

[3] C. Partridge and F. Kastenholz,Technical Criteria for Choosing IP The Next Generation (IPng), Request for Comments 1726, (Dec. 1994)

[4] S. Bradner and A. Mankin,The Recommendation for the IP Next Generation Protocol, Request for Comments 1752, (Jan. 1995)

[5] S. Deering and R. Hinden,Internet Protocol, Version 6 (IPv6) Specification, Request for Comments 2460, (Dec. 1998)

[6] S. Deering and R. Hinden, Internet Protocol, Version 6 (IPv6) Specification, Request for Comments 1883, (Dec. 1995)

[7] C. Madson and R. Glenn, The Use of HMAC-SHA-1-96 within ESP and AH, Request for Comments 2404, (Nov. 1998)

[8] S. Kent and R. Atkinson, IP Authentication Header, Request for Comments 2402, (Nov. 1998)

[9] UNIX MAGAZINE 2002 年 1 月号, ASCII, (Jun. 2002)

[10] Srinivas Vegesna 著, コムサス訳, シスコシステムズ監修, IP QoS 完全ガイド SOFT BANK, (Apr. 2002)

[11] IPng Implementations,
http://playground.sun.com/ipng/ipng-implementations.html

[12] KAME Project, http://www.kame.net

[13] Mark A.Miller, トップスタジオ訳, 宇夫陽次郎 監修, 次世代インターネットプロトコルの仕様と事例 IPv6 入門, 翔泳社, (Jun. 1999)

[14] W. Stevens and M. Thomas, Advanced Sockets API for IPv6, Request for Comments2292, (Feb. 1998)

[15] W. Stevens and M. Thomas,Advanced Sockets API for IPv6, draft-ietf-ipngwg-2292bis-08.txt, (Apr. 2003)

[16] Gilligan, et. al., Basic Socket Interface Extensions for IPv6, Request for Comments2553, (Mar. 1999)

[17] 大谷誠:“次世代インターネット・プロトコル IPv6 の実装と相互通信性の検証”, 佐賀大学理工学部情報科学科卒業論文, (Mar. 1998)

[18] 大谷誠, 田中久治, 渡辺健次, 近藤弘樹:“IPv6 のネットワーク構築とその運用”, 情報処理学会九州支部研究会, (Mar. 1999)

[19] 大谷誠, 渡辺健次, 近藤弘樹:“IPv6 におけるルーティングヘッダを用いたアプリケーションの実現”, 電気関係学会九州支部第 52 回連合大会, (Oct. 1999)

[20] 大谷誠, 津田伸秀, 渡辺健次, 近藤弘樹:“IPv6 におけるルーティングヘッダを用いたアプリケーションの実現とその検証”, 情報処理学会第 60 回全国大会, (Mar. 2000)

研究発表

●印はファースト オーサ、○印は共著を示す。

【学位論文】

●大谷誠, 渡辺健次, 近藤弘樹: “次世代インターネット・プロトコル IPv6 の実装と相互通信性の検証”, 佐賀大学理工学部情報科学科卒業論文, (1998.2).

●大谷誠, 渡辺健次, 近藤弘樹: “IPv6 ネットワーク構築とルーティングヘッダを用いた経路制御アプリケーションの実現”, 佐賀大学大学院工学系研究科情報科学専攻修士論文, (2000.2).

【学術論文】(査読付き)

○渡辺健次, 大谷誠, 田中久治, 飯盛義徳, 近藤弘樹: “ギガビットネットワークによる高精細映像を用いた遠隔講義の実践 - 映像と音声の品質とパケットロスの関係 -”, 日本教育工学会誌, Vol. 25 (Suppl.), pp. 149 - 154, (2001.9).

●大谷誠、渡辺健次、近藤弘樹: “IPv6 における能動的経路選択アプリケーションの実現と検証”, 情報処理学会論文誌, Vol. 42, No. 12, pp. 2878 - 2886, (2001.12).

○渡辺健次, 角規彦, 相森豊徳, 大谷誠, 田中久治, 岡崎泰久, 林敏浩, 近藤弘樹: “高精細映像を用いた板書型遠隔講義のための黒板画像遠隔提示システムの実現”, 教育

システム情報学会誌, Vol. 19, No. 4.

【国際会議】(査読付き)

○Kenzi Watanabe, Makoto Otani, Hisaharu Tanaka, Yoshinori Isagai, Keiko Okawa, Jiro Kokuryo, Hiroshi Esaki, Jun Murai and Hiroki Kondo: “The Next Generation Distance Classroomon Broadband Environments - A Report of the Distance Classroom Using High Quality Video/Audio Stream on the Japan Gigabit Network -”, ICCE/SchoolNet 2001, Seoul, Korea, Vol. 3, pp. 1685 - 1686, (2001.11).

● Makoto Otani, Kenzi Watanabe and Hiroki Kondo: “Implementation and verification of an application with active path selection in IPv6 environment”, APAN Shanghai 2002, Shanghai, China, Vol. 28, pp. 13-21, (2002.8).

【講演論文】(学会発表)

●大谷誠, 田中久治, 渡辺健次, 近藤弘樹: “IPv6 のネットワーク構築とその運用”, 第 13 回情報処理学会九州支部研究会講演論文集, pp. 109 - 116, (1999.3).

●大谷誠, 田中久治, 渡辺健次, 近藤弘樹: “次世代インターネットプロトコル IPv6”, 情報処理学会九州支部若手の会, (1999.7.20).

●大谷誠, 渡辺健次, 近藤弘樹: “IPv6 におけるルーティング・ヘッダを用いたアプリケーションの実現”, 平成 11 年度電気関係学会九州支部連合大会, (1999.10.2).

●大谷誠, 津田伸秀, 渡辺健次, 近藤弘樹: “IPv6 におけるルーティング・ヘッダを用いたアプリケーションの実現とその検証”, 情報処理学会第 60 会全国大会講演論文集, 3-379 - 3-380, (2000.3).

○渡辺健次, 大谷誠, 津田伸秀, 近藤弘樹: “IPv6 のルーティングヘッダを利用した能動的に経路が指定できるアプリケーションの開発”, 第 2 回インターネットにおける QoS 提供に関するワークショップ論文集, pp.180 - 196, (2000.4).

●大谷誠, 津田伸秀, 渡辺健次, 近藤弘樹: “IPv6 におけるルーティングヘッダを用いた経路制御アプリケーションの実現”, 情報処理学会九州支部若手の会セミナー 2000 講演論文集, pp. 13 - 14, (2000.7).

●大谷誠, 津田伸秀, 渡辺健次, 近藤広樹: “IPv6 における能動的な経路制御システムの実現”, 平成 12 年度電気関係学会九州支部連合大会講演論文集, p. 428, (2000.9).

○渡辺健次, 大谷誠, 田中久治, 飯盛義徳, 大川恵子, 國領二郎, 江崎浩, 村井純, 近藤弘樹: “ギガビットネットワークによる高品質映像を用いた遠隔講義”, 信学技報 (ET2000-86), Vol. 100, No. 516, pp. 71 - 77 (2000.12).

○渡辺健次, 大谷誠, 田中久治, 飯盛義徳, 大川恵子, 國領二郎, 江崎浩, 村井純, 近藤弘樹: “ギガビットネットワークによる高品質映像を用いた遠隔講義の実践”, 情報処理学会九州支部火の国情報シンポジウム 2001, pp. 9 - 16, (2001.3.8).

○林敏浩, 大谷誠, 田中久治, 秋山和範, 井上真由美, 渡辺健次, 林田行雄, 近藤弘樹: “高精細映像を用いた板書授業方式による遠隔講義”, 信学技報 (ET2000-113〜142), Vol. 100, No. 682, pp. 219 - 226, (2001.3.10).

○渡辺健次, 林敏浩, 大谷誠, 田中久治, 林田行雄, 近藤弘樹: “遠隔授業の用件と板書授業方式による遠隔講義”, 信学技報, Vol. 101, No. 101, pp. 1-8, (2001.5).

○渡辺健次, 林敏浩, 大谷誠, 田中久治, 岡崎泰久, 林田行雄, 近藤弘樹:“高精細メディアを利用した板書授業方式による遠隔講義とその検証”, 2001 年日本教育工学会第 17 回大会講演論文集, pp. 761 - 762, (2001.11).

●大谷誠, 江頭広幸, 田中久治, 渡辺健次, 近藤弘樹, 緒方俊彦, 溝口正昭, 田尻博伸, 西村龍一郎: “佐賀地域における IPv6 ネットワークの構築”, 火の国情報シンポジウム 2002, (2002.3).

○渡辺健次, 林敏浩, 大谷誠, 田中久治, 岡崎泰久, 林田行雄, 近藤弘樹: “高精細メディアによる板書授業形式の遠隔講義の構成”, 火の国情報シンポジウム 2002, (2002.3).

○角規彦, 相森豊徳, 大谷誠, 田中久治, 岡崎泰久, 林敏浩, 渡辺健次, 近藤弘樹: “板書型遠隔講義のための DVTS 静止画遠隔提示システムの開発”, 平成 14 年度電気関係学会九州支部連合大会講演論文集, p. 474, (2002.9).

○角規彦, 相森豊徳, 大谷誠, 江頭広幸, 田中久治, 岡崎泰久, 林敏浩, 渡辺健次, 近藤弘樹: “DVTS を用いた板書型遠隔講義と静止画遠隔提示システムの実現”, 電子情報通信学会教育工学研究会, 信学技報, pp. 49 - 54, (2002.12).

○中村隆敏, 山田成仙, 山下利秀, 緒方利秀, 溝口正昭, 西村龍一郎, 大谷誠, 江頭広幸, 田中久治, 渡辺健次, 近藤弘樹:“工業高校における IPv6 を用いたロボット遠隔操作の実証実験”, 情報処理学会第 65 回全国大会, (2003.3)

【研究開発】

●大谷誠, 江頭広幸: “PC–UNIX について ～FreeBSD のインストール～”, 佐賀大学情報処理センター広報第 8 号, pp. 51 - 57, (1999.3).

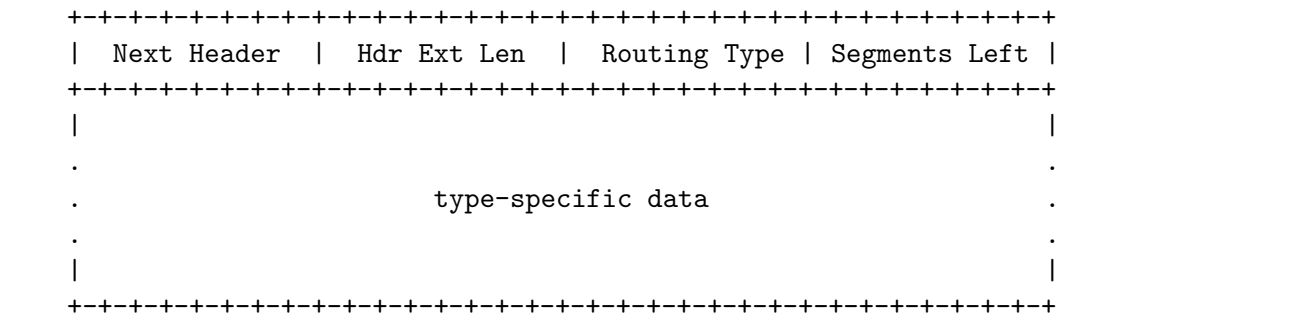
●大谷誠: “次世代のインターネットプロトコル IPv6 ～実際に使ってみよう～”, 佐賀大学学術情報処理センター広報 2 号, (2002.3).

付録 A

Routing Header

A.1 Routing Header

The Routing header is used by an IPv6 source to list one or more intermediate nodes to be "visited" on the way to a packet's destination. This function is very similar to IPv4's Loose Source and Record Route option. The Routing header is identified by a Next Header value of 43 in the immediately preceding header, and has the following format:



Next Header — 8-bit selector. Identifies the type of header immediately following the Routing header. Uses the same values as the IPv4 Protocol field [RFC-1700 et seq.].

Hdr Ext Len — 8-bit unsigned integer. Length of the Routing header in 8-octet units, not including the first 8 octets.

Routing Type — 8-bit identifier of a particular Routing header variant.

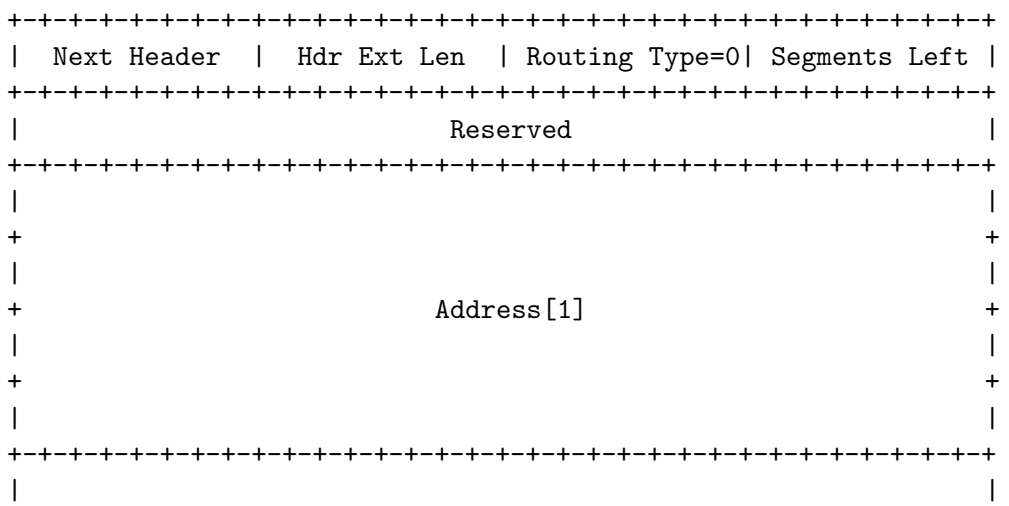
Segments Left	8-bit unsigned integer. Number of route segments remaining, i.e., number of explicitly listed intermediate nodes still to be visited before reaching the final destination.
type-specific data	Variable-length field, of format determined by the Routing Type, and of length such that the complete Routing header is an integer multiple of 8 octets long.

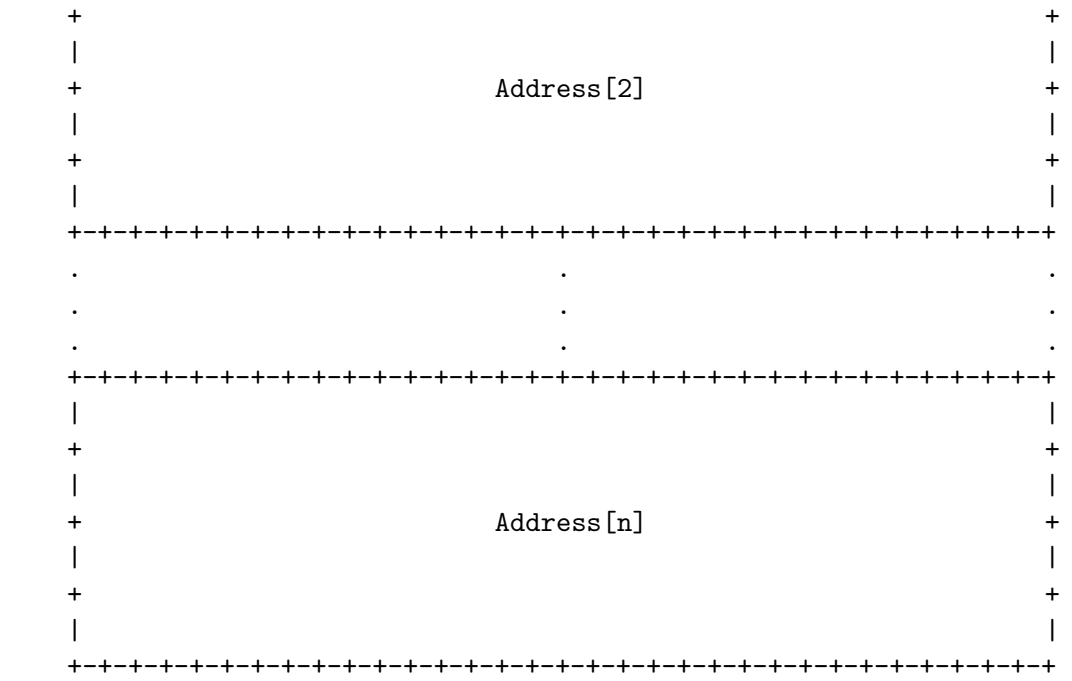
If, while processing a received packet, a node encounters a Routing header with an unrecognized Routing Type value, the required behavior of the node depends on the value of the Segments Left field, as follows:

> If Segments Left is zero, the node must ignore the Routing header and proceed to process the next header in the packet, whose type is identified by the Next Header field in the Routing header.
>
> If Segments Left is non-zero, the node must discard the packet and send an ICMP Parameter Problem, Code 0, message to the packet's Source Address, pointing to the unrecognized Routing Type.

If, after processing a Routing header of a received packet, an intermediate node determines that the packet is to be forwarded onto a link whose link MTU is less than the size of the packet, the node must discard the packet and send an ICMP Packet Too Big message to the packet's Source Address.

The Type 0 Routing header has the following format:





Next Header	8-bit selector. Identifies the type of header immediately following the Routing header. Uses the same values as the IPv4 Protocol field [RFC-1700 et seq.].
Hdr Ext Len	8-bit unsigned integer. Length of the Routing header in 8-octet units, not including the first 8 octets. For the Type 0 Routing header, Hdr Ext Len is equal to two times the number of addresses in the header.
Routing Type	0.
Segments Left	8-bit unsigned integer. Number of route segments remaining, i.e., number of explicitly listed intermediate nodes still to be visited before reaching the final destination.
Reserved	32-bit reserved field. Initialized to zero for transmission; ignored on reception.
Address[1..n]	Vector of 128-bit addresses, numbered 1 to n.

Multicast addresses must not appear in a Routing header of Type 0, or in the IPv6 Destination Address field of a packet carrying a Routing header of Type 0.

```
A Routing header is not examined or processed until it reaches the
node identified in the Destination Address field of the IPv6 header.
In that node, dispatching on the Next Header field of the immediately
preceding header causes the Routing header module to be invoked,
which, in the case of Routing Type 0, performs the following
algorithm:

if Segments Left = 0 {
   proceed to process the next header in the packet, whose type is
   identified by the Next Header field in the Routing header
}
else if Hdr Ext Len is odd {
      send an ICMP Parameter Problem, Code 0, message to the Source
      Address, pointing to the Hdr Ext Len field, and discard the
      packet
}
else {
   compute n, the number of addresses in the Routing header, by
   dividing Hdr Ext Len by 2

   if Segments Left is greater than n {
      send an ICMP Parameter Problem, Code 0, message to the Source
      Address, pointing to the Segments Left field, and discard the
      packet
   }
   else {
      decrement Segments Left by 1;
      compute i, the index of the next address to be visited in
      the address vector, by subtracting Segments Left from n

      if Address [i] or the IPv6 Destination Address is multicast {
         discard the packet
      }
      else {
         swap the IPv6 Destination Address and Address[i]

         if the IPv6 Hop Limit is less than or equal to 1 {
            send an ICMP Time Exceeded -- Hop Limit Exceeded in
            Transit message to the Source Address and discard the
            packet
         }
         else {
            decrement the Hop Limit by 1

            resubmit the packet to the IPv6 module for transmission
```

```
            to the new destination
         }
      }
   }
}
```

As an example of the effects of the above algorithm, consider the case of a source node S sending a packet to destination node D, using a Routing header to cause the packet to be routed via intermediate nodes I1, I2, and I3. The values of the relevant IPv6 header and Routing header fields on each segment of the delivery path would be as follows:

As the packet travels from S to I1:

```
     Source Address = S                  Hdr Ext Len = 6
     Destination Address = I1            Segments Left = 3
                                         Address[1] = I2
                                         Address[2] = I3
                                         Address[3] = D
```

As the packet travels from I1 to I2:

```
     Source Address = S                  Hdr Ext Len = 6
     Destination Address = I2            Segments Left = 2
                                         Address[1] = I1
                                         Address[2] = I3
                                         Address[3] = D
```

As the packet travels from I2 to I3:

```
     Source Address = S                  Hdr Ext Len = 6
     Destination Address = I3            Segments Left = 1
                                         Address[1] = I1
                                         Address[2] = I2
                                         Address[3] = D
```

As the packet travels from I3 to D:

```
     Source Address = S                  Hdr Ext Len = 6
     Destination Address = D             Segments Left = 0
                                         Address[1] = I1
                                         Address[2] = I2
                                         Address[3] = I3
```

A.2 Responding to Packets Carrying Routing Headers

When an upper-layer protocol sends one or more packets in response to a received packet that included a Routing header, the response packet(s) must not include a Routing header that was automatically derived by "reversing" the received Routing header UNLESS the integrity and authenticity of the received Source Address and Routing header have been verified (e.g., via the use of an Authentication header in the received packet). In other words, only the following kinds of packets are permitted in response to a received packet bearing a Routing header:

- Response packets that do not carry Routing headers.
- Response packets that carry Routing headers that were NOT derived by reversing the Routing header of the received packet (for example, a Routing header supplied by local configuration).
- Response packets that carry Routing headers that were derived by reversing the Routing header of the received packet IF AND ONLY IF the integrity and authenticity of the Source Address and Routing header from the received packet have been verified by the responder.

付録 B

Advanced Sockets API for IPv6 Routing Header Option

B.1 Routing Header Option

Source routing in IPv6 is accomplished by specifying a Routing header as an extension header. There can be different types of Routing headers, but IPv6 currently defines only the Type 0 Routing header [RFC-2460]. This type supports up to 127 intermediate nodes (limited by the length field in the extension header). With this maximum number of intermediate nodes, a source, and a destination, there are 128 hops.

Source routing with the IPv4 sockets API (the IP_OPTIONS socket option) requires the application to build the source route in the format that appears as the IPv4 header option, requiring intimate knowledge of the IPv4 options format. This IPv6 API, however, defines six functions that the application calls to build and examine a Routing header, and the ability to use sticky options or ancillary data to communicate this information between the application and the kernel using the IPV6_RTHDR option.

Three functions build a Routing header:

```
inet6_rth_space()    - return #bytes required for Routing header
inet6_rth_init()     - initialize buffer data for Routing header
inet6_rth_add()      - add one IPv6 address to the Routing header
```

Three functions deal with a returned Routing header:

```
inet6_rth_reverse()  - reverse a Routing header
inet6_rth_segments() - return #segments in a Routing header
```

```
  inet6_rth_getaddr()  - fetch one address from a Routing header
```

The function prototypes for these functions are defined as a result of including the <netinet/in.h>.

To receive a Routing header the application must enable the IPV6_RECVRTHDR socket option:

```
    int  on = 1;
    setsockopt(fd, IPPROTO_IPV6, IPV6_RECVRTHDR, &on, sizeof(on));
```

Each received Routing header is returned as one ancillary data object described by a cmsghdr structure with cmsg_type set to IPV6_RTHDR. When multiple Routing headers are received, multiple ancillary data objects (with cmsg_type set to IPV6_RTHDR) will be returned to the application.

To send a Routing header the application specifies it either as ancillary data in a call to sendmsg() or using setsockopt(). For the sending side, this API assumes the number of occurrences of the Routing header as described in [RFC-2460]. That is, applications can only specify at most one outgoing Routing header.

The application can remove any sticky Routing header by calling setsockopt() for IPV6_RTHDR with a zero option length.

When using ancillary data a Routing header is passed between the application and the kernel as follows: The cmsg_level member has a value of IPPROTO_IPV6 and the cmsg_type member has a value of IPV6_RTHDR. The contents of the cmsg_data[] member is implementation dependent and should not be accessed directly by the application, but should be accessed using the six functions that we are about to describe.

The following constant is defined as a result of including the <netinet/in.h>:

```
    #define IPV6_RTHDR_TYPE_0    0 /* IPv6 Routing header type 0 */
```

When a Routing header is specified, the destination address specified for connect(), sendto(), or sendmsg() is the final destination address of the datagram. The Routing header then contains the addresses of all the intermediate nodes.

B.1.1 inet6_rth_space

```
socklen_t inet6_rth_space(int type, int segments);
```

This function returns the number of bytes required to hold a Routing header of the specified type containing the specified number of segments (addresses). For an IPv6 Type 0 Routing header, the number of segments must be between 0 and 127, inclusive. The return value is just the space for the Routing header. When the application uses ancillary data it must pass the returned length to CMSG_SPACE() to determine how much memory is needed for the ancillary data object (including the cmsghdr structure).

If the return value is 0, then either the type of the Routing header is not supported by this implementation or the number of segments is invalid for this type of Routing header.

(Note: This function returns the size but does not allocate the space required for the ancillary data. This allows an application to allocate a larger buffer, if other ancillary data objects are desired, since all the ancillary data objects must be specified to sendmsg() as a single msg_control buffer.)

B.1.2 inet6_rth_init

```
void *inet6_rth_init(void *bp, socklen_t bp_len,
                                          int type, int segments);
```

This function initializes the buffer pointed to by bp to contain a Routing header of the specified type and sets ip6r_len based on the segments parameter. bp_len is only used to verify that the buffer is large enough. The ip6r_segleft field is set to zero; inet6_rth_add() will increment it.

When the application uses ancillary data the application must initialize any cmsghdr fields.

The caller must allocate the buffer and its size can be determined by calling inet6_rth_space().

Upon success the return value is the pointer to the buffer (bp), and this is then used as the first argument to the inet6_rth_add() function. Upon an error the return value is NULL.

B.1.3 inet6_rth_add

```
int inet6_rth_add(void *bp, const struct in6_addr *addr);
```

This function adds the IPv6 address pointed to by addr to the end of the Routing header being constructed.

If successful, the segleft member of the Routing Header is updated to account for the new address in the Routing header and the return value of the function is 0. Upon an error the return value of the function is -1.

B.1.4 inet6_rth_reverse

```
int inet6_rth_reverse(const void *in, void *out);
```

This function takes a Routing header extension header (pointed to by the first argument) and writes a new Routing header that sends datagrams along the reverse of that route. The function reverses the order of the addresses and sets the segleft member in the new Routing header to the number of segments. Both arguments are allowed to point to the same buffer (that is, the reversal can occur in place).

The return value of the function is 0 on success, or -1 upon an error.

B.1.5 inet6_rth_segments

```
int inet6_rth_segments(const void *bp);
```

This function returns the number of segments (addresses) contained in the Routing header described by bp. On success the return value is zero or greater. The return value of the function is -1 upon an error.

B.1.6 inet6_rth_getaddr

```
struct in6_addr *inet6_rth_getaddr(const void *bp, int index);
```

This function returns a pointer to the IPv6 address specified by index (which must have a value between 0 and one less than the value returned by inet6_rth_segments()) in the Routing header described by bp. An application should first call inet6_rth_segments() to obtain

the number of segments in the Routing header.

Upon an error the return value of the function is NULL.

B.2 Examples using the inet6_rth_XXX() functions

Here we show an example for both sending Routing headers and processing and reversing a received Routing header.

B.2.1 Sending a Routing Header

As an example of these Routing header functions defined in this document, we go through the function calls for the example on p. 17 of [RFC-2460]. The source is S, the destination is D, and the three intermediate nodes are I1, I2, and I3.

```
          S -----> I1 -----> I2 -----> I3 -----> D

src:    *    S         S         S         S   S
dst:    D   I1        I2        I3         D   D
A[1]:  I1   I2        I1        I1        I1  I1
A[2]:  I2   I3        I3        I2        I2  I2
A[3]:  I3    D         D         D        I3  I3
#seg:   3    3         2         1         0   3
```

src and dst are the source and destination IPv6 addresses in the IPv6 header. A[1], A[2], and A[3] are the three addresses in the Routing header. #seg is the Segments Left field in the Routing header.

The six values in the column beneath node S are the values in the Routing header specified by the sending application using sendmsg() of setsockopt(). The function calls by the sender would look like:

```
void  *extptr;
socklen_t   extlen;
struct msghdr  msg;
struct cmsghdr  *cmsgptr;
int   cmsglen;
struct sockaddr_in6  I1, I2, I3, D;

extlen = inet6_rth_space(IPV6_RTHDR_TYPE_0, 3);
cmsglen = CMSG_SPACE(extlen);
```

```
    cmsgptr = malloc(cmsglen);
    cmsgptr->cmsg_len = CMSG_LEN(extlen);
    cmsgptr->cmsg_level = IPPROTO_IPV6;
    cmsgptr->cmsg_type = IPV6_RTHDR;

    extptr = CMSG_DATA(cmsgptr);
    extptr = inet6_rth_init(extptr, extlen, IPV6_RTHDR_TYPE_0, 3);

    inet6_rth_add(extptr, &I1.sin6_addr);
    inet6_rth_add(extptr, &I2.sin6_addr);
    inet6_rth_add(extptr, &I3.sin6_addr);

    msg.msg_control = cmsgptr;
    msg.msg_controllen = cmsglen;

    /* finish filling in msg{}, msg_name = D */
    /* call sendmsg() */
```

We also assume that the source address for the socket is not specified (i.e., the asterisk in the figure).

The four columns of six values that are then shown between the five nodes are the values of the fields in the packet while the packet is in transit between the two nodes. Notice that before the packet is sent by the source node S, the source address is chosen (replacing the asterisk), I1 becomes the destination address of the datagram, the two addresses A[2] and A[3] are "shifted up", and D is moved to A[3].

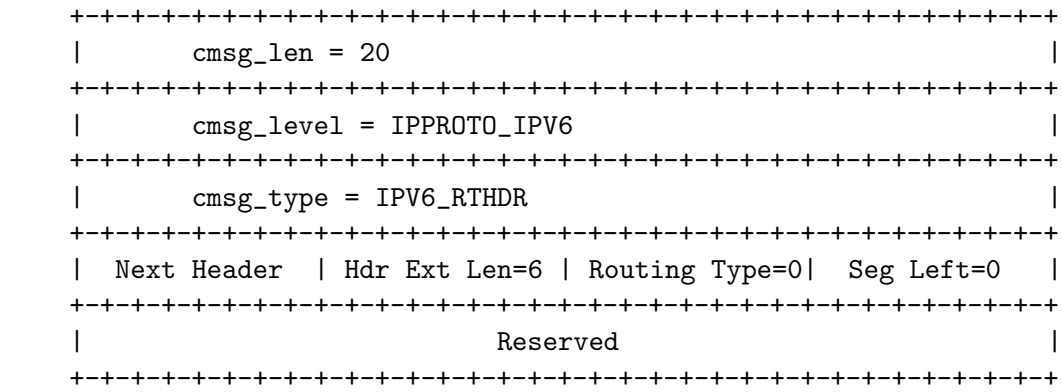
The columns of values that are shown beneath the destination node are the values returned by recvmsg(), assuming the application has enabled both the IPV6_RECVPKTINFO and IPV6_RECVRTHDR socket options. The source address is S (contained in the sockaddr_in6 structure pointed to by the msg_name member), the destination address is D (returned as an ancillary data object in an in6_pktinfo structure), and the ancillary data object specifying the Routing header will contain three addresses (I1, I2, and I3). The number of segments in the Routing header is known from the Hdr Ext Len field in the Routing header (a value of 6, indicating 3 addresses).

The return value from inet6_rth_segments() will be 3 and inet6_rth_getaddr(0) will return I1, inet6_rth_getaddr(1) will return I2, and inet6_rth_getaddr(2) will return I3,

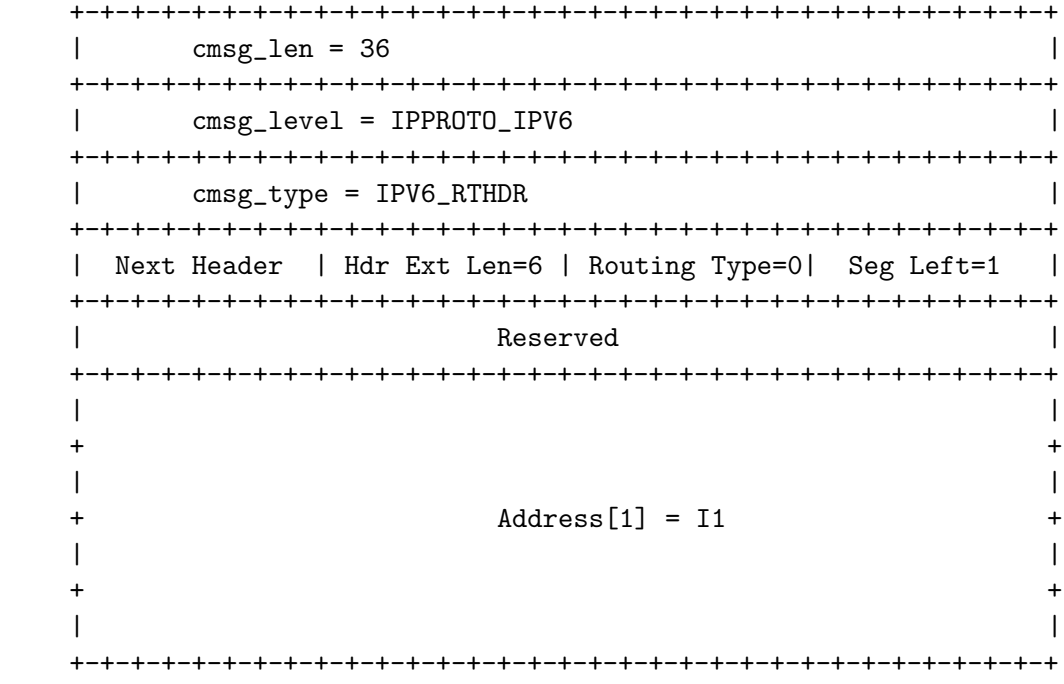
If the receiving application then calls inet6_rth_reverse(), the order of the three addresses will become I3, I2, and I1.

We can also show what an implementation might store in the ancillary data object as the Routing header is being built by the sending process. If we assume a 32-bit architecture where sizeof(struct cmsghdr) equals 12, with a desired alignment of 4-byte boundaries, then the call to inet6_rth_space(3) returns 68: 12 bytes for the cmsghdr structure and 56 bytes for the Routing header (8 + 3*16).

The call to inet6_rth_init() initializes the ancillary data object to contain a Type 0 Routing header:



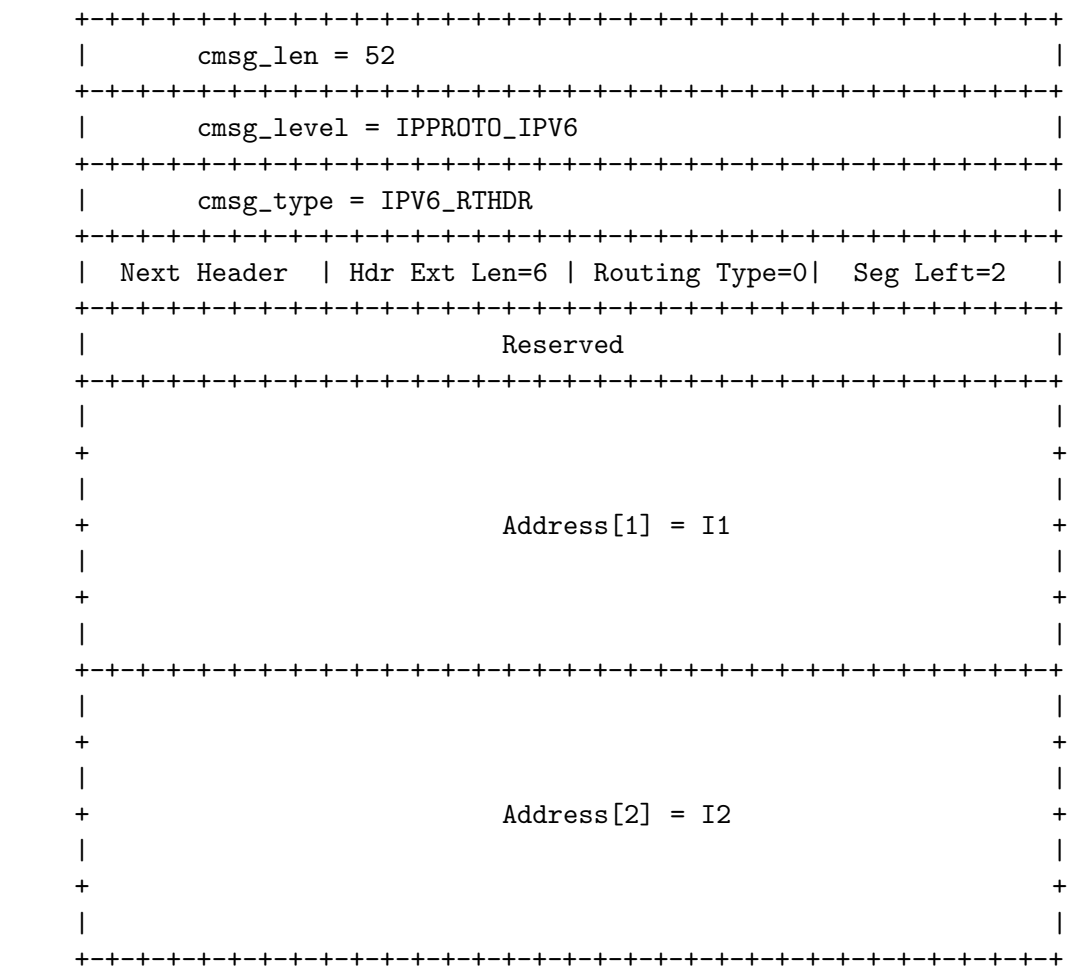
The first call to inet6_rth_add() adds I1 to the list.



cmsg_len is incremented by 16, and the Segments Left field is

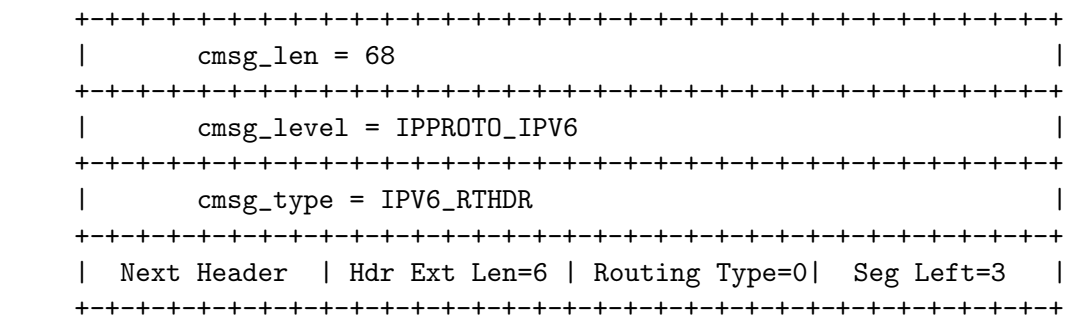
incremented by 1.

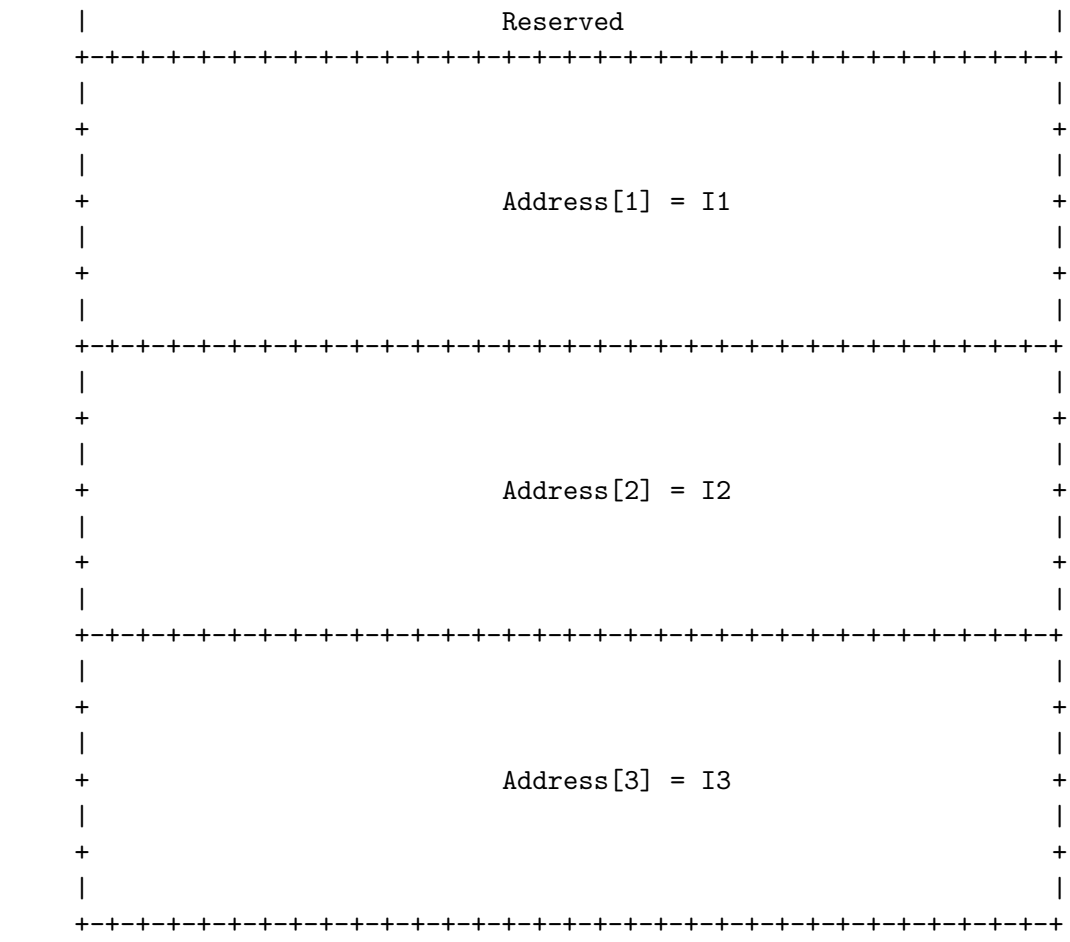
The next call to inet6_rth_add() adds I2 to the list.



cmsg_len is incremented by 16, and the Segments Left field is incremented by 1.

The last call to inet6_rth_add() adds I3 to the list.





cmsg_len is incremented by 16, and the Segments Left field is incremented by 1.

B.2.2 Receiving Routing Headers

This example assumes that the application has enabled IPV6_RECVRTHDR socket option. The application prints and reverses a source route and uses that to echo the received data.

```
struct sockaddr_in6     addr;
struct msghdr           msg;

struct iovec            iov;
struct cmsghdr          *cmsgptr;
socklen_t               cmsgspace;
void                    *extptr;
int                     extlen;
```

```
int                     segments;
int                     i;
char                    databuf[8192];

segments = 100;        /* Enough */
extlen = inet6_rth_space(IPV6_RTHDR_TYPE_0, segments);
cmsgspace = CMSG_SPACE(extlen);
cmsgptr = malloc(cmsgspace);
if (cmsgptr == NULL) {
        perror("malloc");
        exit(1);
}
extptr = CMSG_DATA(cmsgptr);

msg.msg_control = cmsgptr;
msg.msg_controllen = cmsgspace;
msg.msg_name = (struct sockaddr *)&addr;
msg.msg_namelen = sizeof (addr);
msg.msg_iov = &iov;
msg.msg_iovlen = 1;
iov.iov_base = databuf;
iov.iov_len = sizeof (databuf);
msg.msg_flags = 0;
if (recvmsg(s, &msg, 0) == -1) {
        perror("recvmsg");
        return;
}
if (msg.msg_controllen != 0 &&
    cmsgptr->cmsg_level == IPPROTO_IPV6 &&
    cmsgptr->cmsg_type == IPV6_RTHDR) {
        struct in6_addr *in6;
        char asciiname[INET6_ADDRSTRLEN];
        struct ip6_rthdr *rthdr;

        rthdr = (struct ip6_rthdr *)extptr;
        segments = inet6_rth_segments(extptr);
        printf("route (%d segments, %d left): ",
            segments, rthdr->ip6r_segleft);
        for (i = 0; i < segments; i++) {
                in6 = inet6_rth_getaddr(extptr, i);
                if (in6 == NULL)
                        printf("<NULL> ");
                else
                        printf("%s ", inet_ntop(AF_INET6,
                            (void *)in6->s6_addr,
                            asciiname, INET6_ADDRSTRLEN));
```

```
            }
            if (inet6_rth_reverse(extptr, extptr) == -1) {
                    printf("reverse failed");
                    return;
            }
    }
    iov.iov_base = databuf;
    iov.iov_len = strlen(databuf);
    if (sendmsg(s, &msg, 0) == -1)
            perror("sendmsg");
    if (cmsgptr != NULL)
            free(cmsgptr);
```

Note: The above example is a simple illustration. It skips some error checks, including those involving the MSG_TRUNC and MSG_CTRUNC flags. It also leaves some type mismatches in favor of brevity.