特別研究報告

題目

IPv6 ネットワークにおけるエニーキャスト通信実現のためのプロトコル設計と実装

指導教官

宮原 秀夫 教授

報告者

土居 聡

平成 14 年 2 月 19 日

大阪大学 基礎工学部 情報科学科

平成 13 年度 特別研究報告

IPv6 ネットワークにおけるエニーキャスト通信実現のためのプロトコル設計と実装

土居 聡

内容梗概

IPv6 の持つ新しい機能の 1 つとして、エニーキャスト通信機能がある。エニーキャスト通信を実現するために規定されているエニーキャストアドレスは、特定の機能 (サービス) に対して割り当てられるアドレスであり、クライアント側はエニーキャストアドレスを指定しさえすれば、対応する機能を提供する複数のサーバの中から 1 台の適切なサーバが自動的に選ばれ、そのサーバに対して通信することが可能となる。しかし、エニーキャスト通信については方式が規定されているのみで、その実装、実現方法について解決すべき問題が多く存在する。そこで本報告では、エニーキャストアドレスを利用する場合の制限事項を明らかにし、その解決のための手法を設計している。また、設計案をエンドホストに実装し、既存のアプリケーションがプログラムを変更することなく、エニーキャストアドレスを利用できることを示している。さらに、エニーキャストアドレスの経路制御について述べ、エニーキャストアドレスを用いた新たなサーバ負荷分散モデルを示し、実装評価によってその有効性を明らかにしている。

主な用語

IPv6, エニーキャストアドレス, 負荷分散, ICMPv6, Neighbor Discovery

目 次

1 はじめに **7**

2 エニーキャストアドレスを用いた通信 **10**

2.1 IPv6 におけるアドレスの種類 10

2.2 エニーキャストアドレスの応用例 13

2.3 エニーキャスト通信における制限事項 14

2.4 エニーキャスト通信の利用上の問題点 17

3 エニーキャストアドレスによる通信のためのアドレス解決モデルの設計と実装 **18**

3.1 提案方式の目的と概要 18

3.2 AARP によるアドレス解決 19

3.3 AARP 中間ライブラリ 21

4 エニーキャストアドレスのための端末選択制御機構の提案 **23**

4.1 近隣探索によるサブネット内での端末選択 24

4.2 グローバルスコープにおける最適な端末の選択 26

5 AARP の実装と評価 **29**

5.1 AARP の実装 29

5.2 TCP 通信の評価 33

5.3 UDP 通信の評価 35

6 エニーキャスト経路制御によるサーバ負荷分散モデルの評価 **38**

6.1 サーバの負荷に応じた Neighbor Cache 更新手法 38

6.1.1 自サーバの負荷情報のみを利用する場合 38

6.1.2 他サーバの unsolicited NA を利用する場合 39

6.2 経路制御を用いたサーバ負荷分散モデルの性能評価 40

6.2.1 シミュレーションによる負荷分散モデルの性能評価 40

7　まとめと今後の課題 **45**

謝辞 **46**

参考文献 **47**

図 目 次

1 ユニキャストアドレス 11
2 マルチキャストアドレス 11
3 エニーキャストアドレス 12
4 最適サーバの自動選択 13
5 送信元アドレスとして使えないエニーキャストアドレス 15
6 TCP 3 ウェイハンドシェーク 16
7 経路の変化による通信の断絶 16
8 本報告で提案するアドレス解決手法 19
9 Anycast Address Resolving Protocol 20
10 client initiate — プローブパケット方式 20
11 server initiate — piggyback 方式 21
12 AARP ライブラリを組み込んだプロトコルスタック 22
13 エニーキャストアドレスの割り当て方法 24
14 Neighbor Cache の設定方法 25
15 unsolicited NA による Neighbor Cache の更新 27
16 アドレス解決を行う中間ライブラリ 30
17 ICMPv6 Echo Request/Reply によるアドレス解決 31
18 `aarpd` 32
19 `aarpd` を共有する方法 33
20 ネットワーク構成 34
21 `telnet` の結果 34
22 `ftp` の結果 34
23 ネットワーク構成 (DNS サーバがある場合) 36
24 DNS 名前解決の結果 36
25 提案するサーバ負荷分散モデル 41
26 unsolicited NA による処理サーバの変化 41

27 セグメント内の全ノードの Neighbor Cache の変化 42
28 $M/M/2$ 待ち行列システム 43
29 $M/M/1 \times 2$ 待ち行列システム 43
30 シミュレーション結果 44

表 目 次

1　各アドレスタイプの通信形態 12

1 はじめに

近年、インターネットは爆発的に普及し、WWW (World Wide Web) や E-mail に代表される気軽なコミュニケーション手段として浸透してきた。現在のインターネットを支える技術であり、広く利用されているインターネットプロトコルである IPv4 (Internet Protocol version 4) [1] は 1970 年代に設計されたものであり、インターネットの普及に伴い現在に至るまで多くの拡張がなされてきた。

しかし、開発された当時十分であると考えられていた IPv4 のアドレス空間の利用限界に到達しつつある。IPv4 では 32 ビットでアドレスを表現するため、すべての空間を利用しても 43 億個のアドレスしか使うことができない。今後、携帯端末や家電、自動車等の IP 端末化が進んだ場合、IP アドレスが枯渇することは容易に想像できる。一方、アドレス不足を解決する手法として、CIDR (Classless Inter-Domain Routing) [2] や NAT (Network Address Translator) [3] などの技術が現在用いられているが、いずれも一時的な措置であり、アドレス不足の問題を根本的に解決することはできない。

このため 1990 年代前半より、アドレス枯渇問題を解決するための新たなインターネットプロトコル (IP) の策定が、IETF (The Internet Engineering Task Force) を中心として検討された結果、IPv6 (Internet Protocol version 6) [4] として標準化され、現在実用化がなされている。IPv6 では、アドレス空間は 128 ビットに拡大され、膨大な数の IP アドレスを割り当てることが可能である。また、IPv6 ではアドレス空間の拡張だけでなく、IPv4 の柔軟性を保ちつつ問題点を改善し、さらにセキュリティやリアルタイム通信、移動端末へのサポートなど、次世代ネットワークのための新しい機能が導入されている。

これらの新機能の一つとして、エニーキャストアドレス通信機能がある。エニーキャストアドレスとは、特定の機能に対して割り当てることのできるアドレスで、利用者はエニーキャストアドレスを指定した通信を行うことにより、実際のサービスがどの端末から提供されるかを知る必要なく最適なサーバからサービスを受けることが可能となる。例えば、プロキシや DNS などの各サービスごとに既知のエニーキャストアドレスを決めておけば、あらかじめオペレーティングシステム上で設定しておくことができるため、従来のネットワーク管理者によるホストごとの設定は必要なくなる。

このように、エニーキャストアドレスは機能的に興味深い側面を持った技術ではあるが、その利用には、エニーキャストアドレスとユニキャストアドレスの区別ができない、エニーキャストアドレスはパケットの送信元アドレスとして利用できない、通信中の相手が変化する可能性がある、といったエニーキャストアドレスを用いた通信特有の問題があり、今すぐ利用可能なものとはなっていない。

そこで本報告では、エニーキャストアドレスを用いた通信を実現するための新たなプロトコルを提案、実装し、その効果を示す。はじめに、エニーキャストアドレスを用いることによる利点と、現状における問題点について明らかにし、既存のアプリケーションのままではエニーキャストアドレスを用いた通信が困難であることを示す。そして、これらの技術課題を解決するための手法として、SOCKS-based IPv6 / IPv4 Gateway [5, 6] 機構を参考にしたアドレス解決モデルを提案する。提案モデルでは、エニーキャストアドレスを端末固有のユニキャストアドレスに変換するために新たに AARP (Anycast Address Resolving Protocol) を定義する。また提案モデルは、 SOCKS-based IPv6 / IPv4 Gateway 機構での実装方法と同様の共有ライブラリ置換手法を利用することで、既存のアプリケーションのコードを修正することなく、エニーキャストアドレスを利用可能にする。さらに提案モデルを実装し、実機を用いた評価により、従来の方式では困難であった状況下における、エニーキャストアドレスを利用した通信が円滑に行われることを示す。

次に、エニーキャスト通信を行う際に必要となる、エニーキャストアドレスパケットの経路制御機構について述べる。はじめに、同一エニーキャストアドレスを持つ複数の端末から適切な端末へパケットを送出するための、経路制御メカニズムについて説明し、経路情報の更新手法を示す。次に、経路制御の適用例として、複数サーバによるエニーキャストアドレスを用いた負荷分散モデルを提案する。シミュレーションを用いた性能評価により、提案モデルがより理想に近いサーバ負荷分散を実現できることを示し、さらに実装評価によって提案方式の実現性を明らかにする。

以下、まず 2 章では、エニーキャストアドレスの利点および応用例について説明し、さらにエニーキャストアドレスを用いた通信を実現するために必要な技術課題について述べる。特に、現状におけるエニーキャストアドレス通信の問題点について明らかにする。次に 3 章

では、エニーキャストアドレスを用いた通信を行うために必要となる、エニーキャストアドレスからユニキャストアドレスへの変換について説明し、アドレス解決モデルを新たに提案し、既存のアプリケーションのコードを改変することなく、エニーキャストアドレスを利用するための実装方法を示す。さらに、4 章ではエニーキャストアドレスの経路制御機構を示す。5 章では、AARP を用いたアドレス解決手法を FreeBSD 上に実装し、実機を用いた評価結果を示す。また、6 章ではエニーキャストアドレスの経路制御機構を示し、その適用例として複数サーバの負荷分散モデルを提案する。さらにシミュレーションおよび実装評価により提案モデルの有効性を検証する。最後に 7 章で、まとめと今後の課題を述べる。

2　エニーキャストアドレスを用いた通信

本報告では、エニーキャストアドレスを用いた通信、エニーキャスト通信を取り扱う。そこで本章ではまず、エニーキャストアドレスの性質を示し、期待されるサービスについて述べる。さらに、エニーキャスト通信実現のために解決すべき技術課題について明らかにする。

2.1　IPv6 におけるアドレスの種類

IPv6 ではアドレスに以下の 3 つのタイプが定義されている。なお、アドレスは端末ではなく、端末が持つ各インターフェース (NIC など) に割り当てられ、一般に、各インターフェースには複数のアドレスが割り当てられる。

1. ユニキャストアドレス

   IPv6 上で一意に定まる固有のアドレス。宛先にユニキャストアドレスを指定したパケットは、ユニキャストアドレスに対応するインターフェースにのみ届けられる (図 1)。
2. マルチキャストアドレス

   ユニキャストアドレスとは異なり、複数の異なるインターフェースに同一のアドレスを割り当てることができる。そして、同じマルチキャストアドレスを持つインターフェースによってグループが構成される。マルチキャストアドレスを宛先とするパケットは、そのグループに属するすべてのインターフェースに届けられる (図 2)。
3. エニーキャストアドレス

   同一の機能を提供する複数の端末のインターフェースに割り当てられるアドレス。エニーキャストアドレスを指定したパケットは、同一の機能を提供する端末のうち、「最適な」端末のインターフェースへと届けられる。どのインターフェースに送信されるかは、経路制御メカニズムによって決定される (図 3)。

以上のことから、各アドレスタイプにおける通信形態は表 1 のようになる。

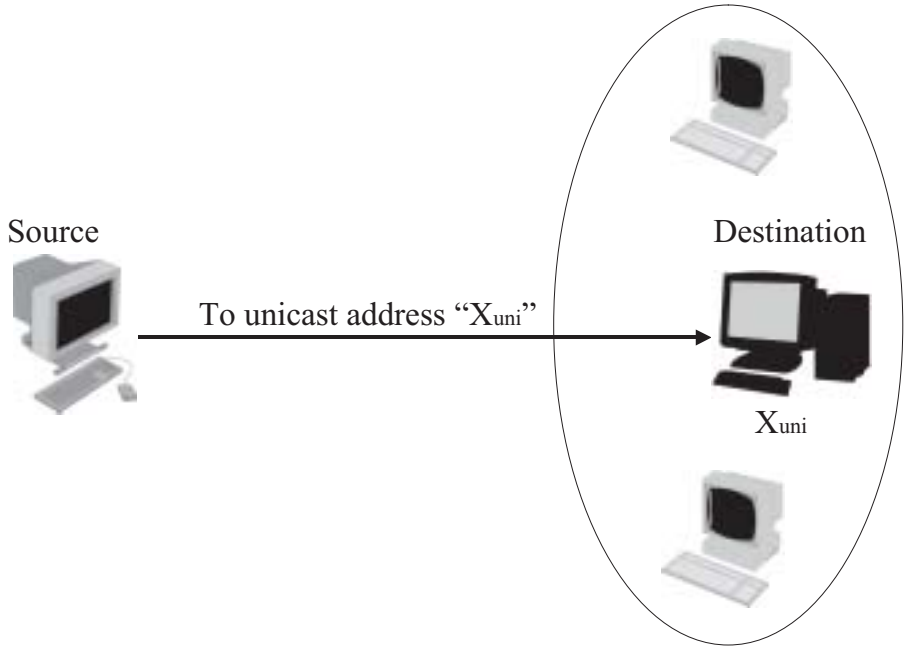

図 1: ユニキャストアドレス

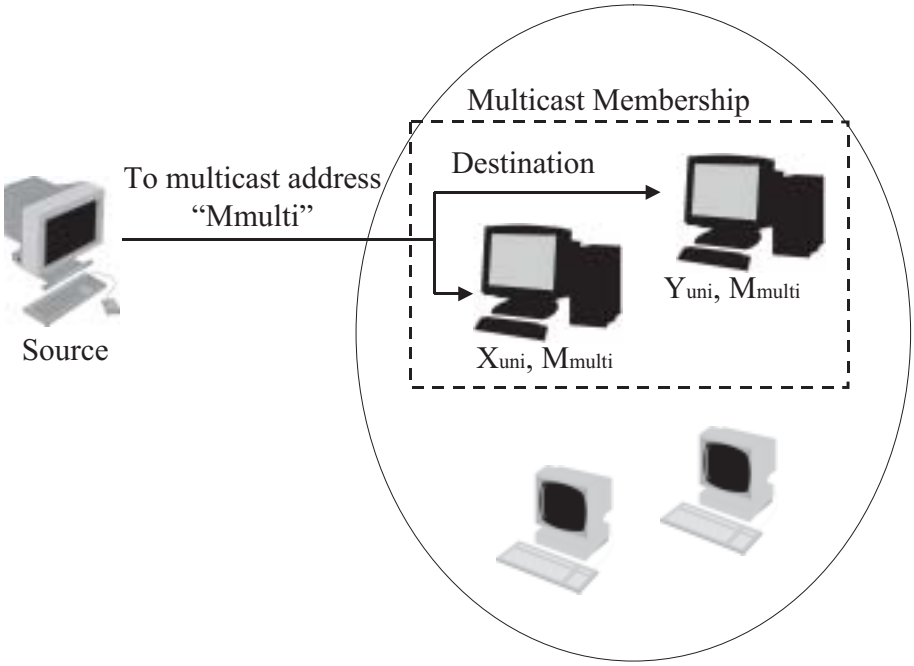

図 2: マルチキャストアドレス

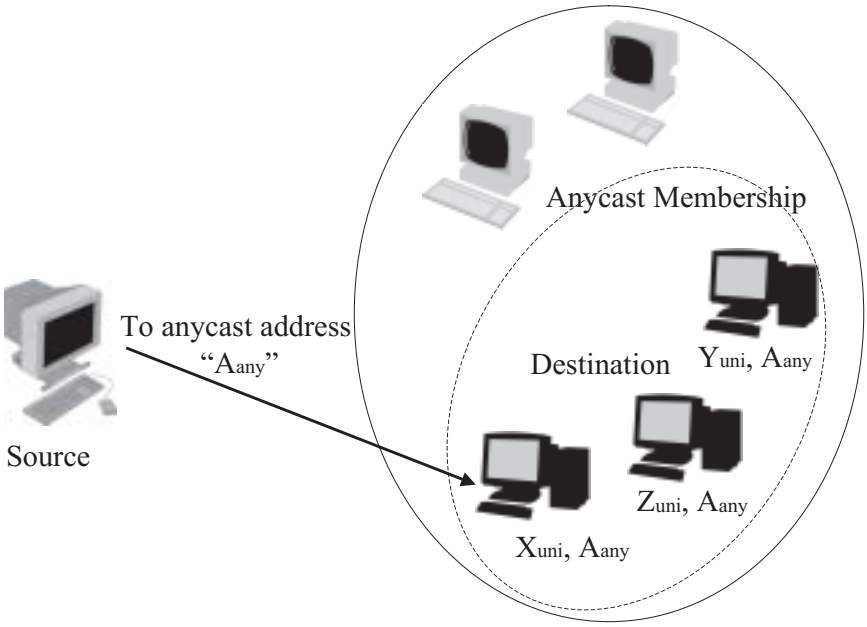

図 3: エニーキャストアドレス

表 1: 各アドレスタイプの通信形態

アドレスタイプ	通信形態	Membership	アドレス設定の対象
ユニキャストアドレス	1 対 1	1	ノード (インターフェース)
マルチキャストアドレス	1 対 多	多	グループ
エニーキャストアドレス	1 対 1	多 (ただし、実際に通信する対象となるのは、その内の「最適な」1)	機能 (サービス)

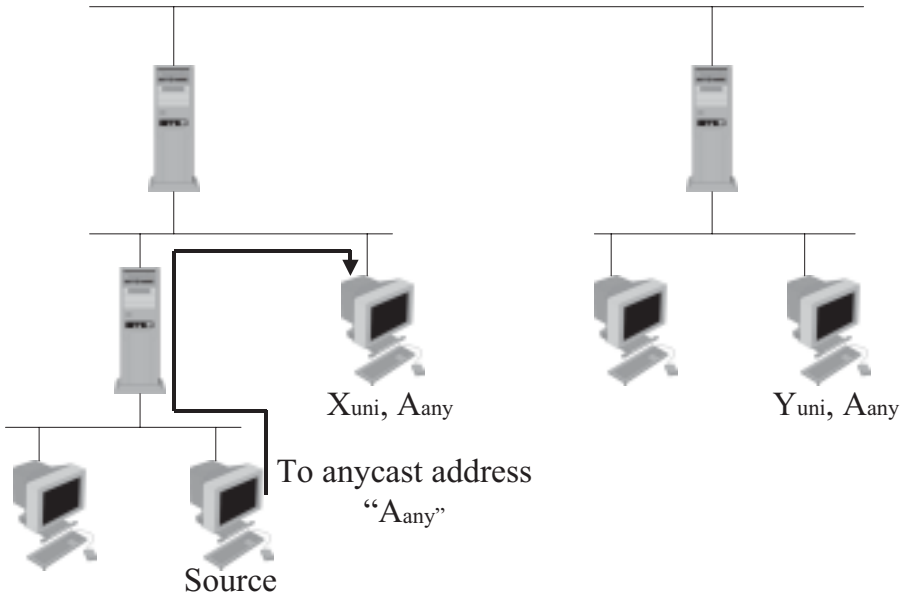

図 4: 最適サーバの自動選択

2.2 エニーキャストアドレスの応用例

複数のインターフェースが同一のエニーキャストアドレスを持つ場合、どのインターフェースにパケットを届けるかについて適当なポリシーを与れば、最適なインターフェース (たとえば、送信側から最も近い端末など) にパケットを送ることができる。また、同一のサービスを提供する端末をグループ化し、そのグループに属する端末のインターフェースにエニーキャストアドレスを割り当てることによって、サービスにアドレスをマッピングさせることができる。以下に、考えられるエニーキャストアドレスの応用例とそのメリットについて説明する。

1. 最適サーバの自動選択

   経路制御を適切に行うことで、複数サーバのうち最適なサーバと通信することができる。「最適」さの評価基準としては、経由するルータの数 (ホップ数)、サーバの負荷などが考えられる。図 4 にホップ数を評価基準とした場合にどのサーバが選択されるかを示す。

2. アドレスのサービスへのマッピング

DNS サーバやプロキシサーバといった、特定のサービスを提供するサーバに対して、そのサービスごとに特定のエニーキャストアドレスをあらかじめ定めることで、利用者は、あらかじめ決められたエニーキャストアドレスを指定するだけでサービスを受けることができる。さらに、これらのエニーキャストアドレスをアプリケーションの標準値として設定することにより、利用者は一切の設定を行うことなくサービスを利用することができる。例えば、DNS サーバにある決まったエニーキャストアドレスを割り当てておき、クライアント側でそのアドレスをオペレーティングシステム上で設定しておけば、プラグアンドプレイ時に最適な DNS サーバを利用することができる。

3. 通信の信頼性の向上とサーバの負荷分散

   経路制御機構によって、エニーキャストアドレスを宛先に持つパケットは、対応するインターフェースのうち最適なノードのインターフェースに届けられる。経路制御を負荷に応じて行えば、サーバの負荷分散も実現可能となる。

2.3 エニーキャスト通信における制限事項

エニーキャストアドレスを用いた通信を行う際、エニーキャストアドレスの特性から、注意すべき点がいくつか考えられる。エニーキャスト通信の特性として、以下が考えられる。

1. エニーキャストアドレスとユニキャストアドレスの区別ができない

   IPv6 では、エニーキャストアドレスはユニキャストアドレスのアドレス空間から選ばれるという仕様になっており、エニーキャストアドレスに対して特にアドレス空間を設定していない。このため受信端末側では、到着パケットの送信元アドレスを調べただけでは、そのアドレスがユニキャストであるか、あるいはエニーキャストであるかを知る方法はない。

2. エニーキャストアドレスはパケットの送信元アドレスとして利用できない

   宛先をエニーキャストアドレスとした場合と同様、送信元アドレスとしてエニーキャストアドレスを指定すると、パケット返送時には対応するインターフェースのどれか一つに届けられる。このとき、必ずしも実際に送信を行った端末が返信パケットを受け取るとは限らない。このため、双方向通信を行うプロトコルである TCP だけでなく、UDP

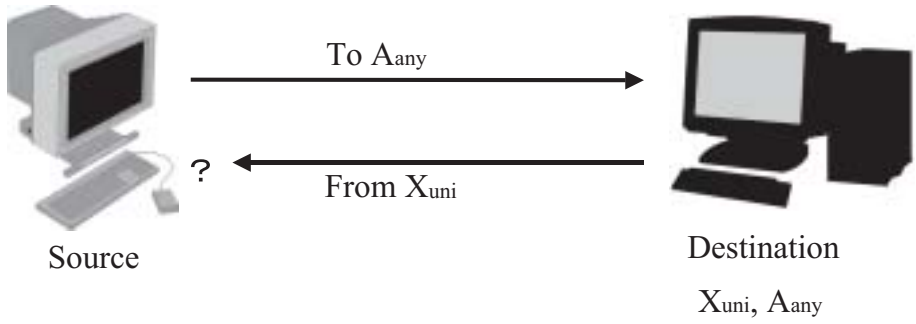

図 5: 送信元アドレスとして使えないエニーキャストアドレス

を利用したプロトコルである RTP (Real Time Protocol) [7] など、パケット到着状況を元にレート制御を行うプロトコルなどが正しく動作しない可能性があり (図 5)、そのため、エニーキャストアドレスをパケットの送信元アドレスにはできない仕様になっている [8]。特に、TCP (Transmission Control Protocol) [9] を用いたアプリケーションにおいてエニーキャストアドレスを用いた通信を行う場合、TCP コネクションを確立する際の 3 ウェイハンドシェーク時に、クライアント側で SYN セグメントの終点アドレスに指定したエニーキャストアドレスと、その SYN に対してサーバ側から送信される ACK の始点アドレスが異なるため、TCP コネクションが確立できない。そのため、現状では TCP アプリケーションにおいてエニーキャストアドレスを利用することはできない (図 6)。

3. 通信中の相手が変化する可能性がある

「最適さ」をはかる基準として考えられる、ホップ数、通信相手の負荷などは、動的に変化する。このため、最適とされるインターフェースも動的に変化する。そのため、常に最適なインターフェースへパケットが送信されることが期待されるエニーキャスト通信では、通信中の相手が変化する可能性がある。したがって、一意に固定した相手との通信ができなくなるため、通信の断絶が起こりうる (図 7)。

以上のように、エニーキャストアドレスを用いた通信には、解決すべき技術課題が多く存在する。

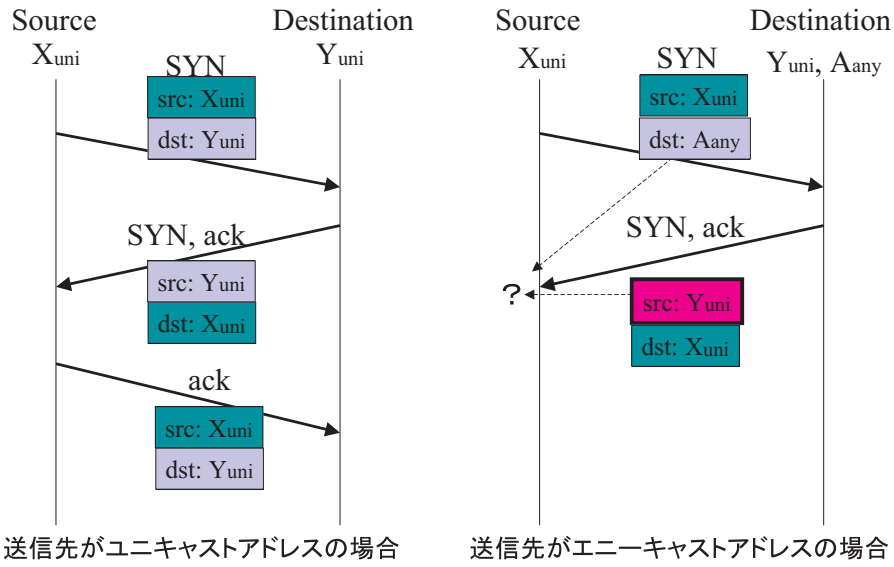

図 6: TCP 3 ウェイハンドシェーク

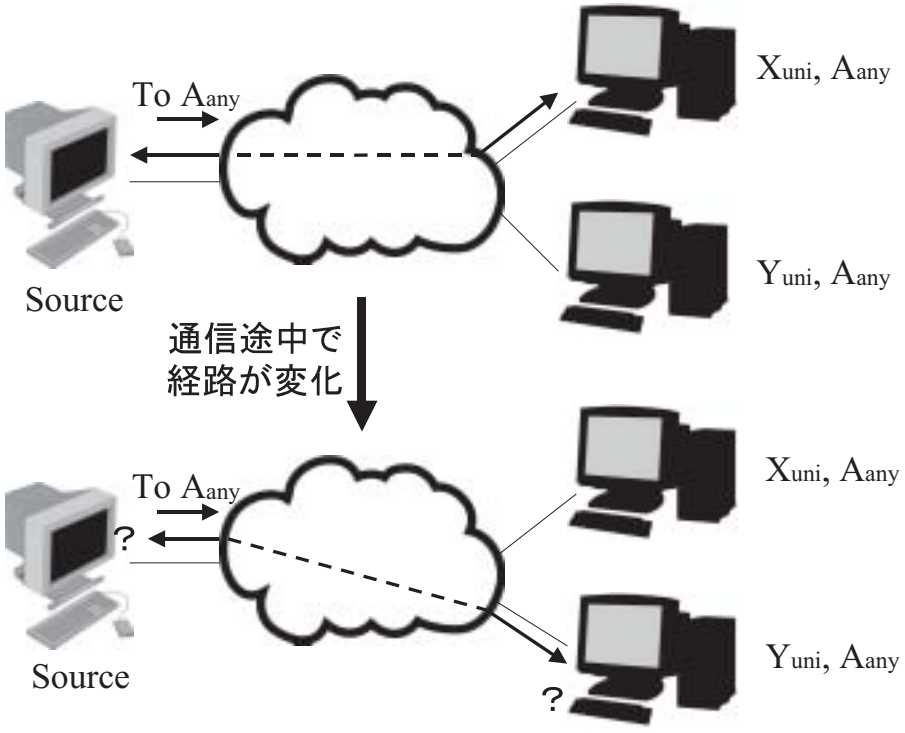

図 7: 経路の変化による通信の断絶

2.4 エニーキャスト通信の利用上の問題点

2.3 節で述べたエニーキャスト通信の制限事項から、TCP を用いたアプリケーションにおいては 3 ウェイハンドシェークによる TCP コネクションが確立できない。また、UDP を用いたアプリケーションにおいても通信中の相手が変化する可能性があるため、コネクションを保てないという重大な問題がある。

以上のことから、

- 現状では、`telnet`、`ftp` などの既存のアプリケーションでエニーキャストアドレスを用いた通信を利用することはできない

という問題点がある。

そこで本報告では、2.2 節で述べたエニーキャストアドレスを用いた通信サービスを実現するため、新たなアドレス解決モデルを提案する。提案モデルは、2.4 節で述べたエニーキャスト通信の利用上の問題点を解決するものであり、特に既存のアプリケーションに変更を加えることなくエニーキャスト通信を実現することを目的とする。

3　エニーキャストアドレスによる通信のためのアドレス解決モデルの設計と実装

本章では、2 章で明らかにしたエニーキャストアドレスを用いた通信時に問題となる点の解決策について述べる。

3.1　提案方式の目的と概要

今後、エニーキャストアドレスを用いた通信を普及させるためには、既存の枠組みを変更することなく容易に移行できることが重要となる。そのために本報告で提案する通信モデルでは、

- 既存の技術をできる限り利用する
- 既存の通信プログラムの修正を行わずにエニーキャスト通信をサポートする
- 前章で述べた、エニーキャスト通信上の問題点を解決する

ことを目標とする。

前章で述べたとおり、エニーキャストアドレスを用いた通信では、通信相手にはエニーキャストアドレスと実際のインターフェースが対応できないことから、特に TCP などの双方向通信がこのままでは利用できないという問題がある。このため、提案方式では以下の手順により、エニーキャスト通信をユニキャスト通信に変更することで、前章の問題点を解決する (図 8 参照)。

1. 通信開始時に、エニーキャストアドレスを対応するユニキャストアドレスに変換する (以下、アドレス解決と呼ぶ)。
2. 変換されたユニキャストアドレスを宛先としてパケットを送出する。

本報告では、アドレス解決の手順として新しくエニーキャストアドレス解決プロトコル (Anycast Address Resolving Protocol; 以下 AARP) を提案する。AARP は、DNS で FQDN (Fully Qualified Domain Name) を対応する IP アドレスへと変換する「名前解決」と同様、エニーキャスト通信の開始時もしくは開始前にエニーキャストアドレスをユニキャストアド

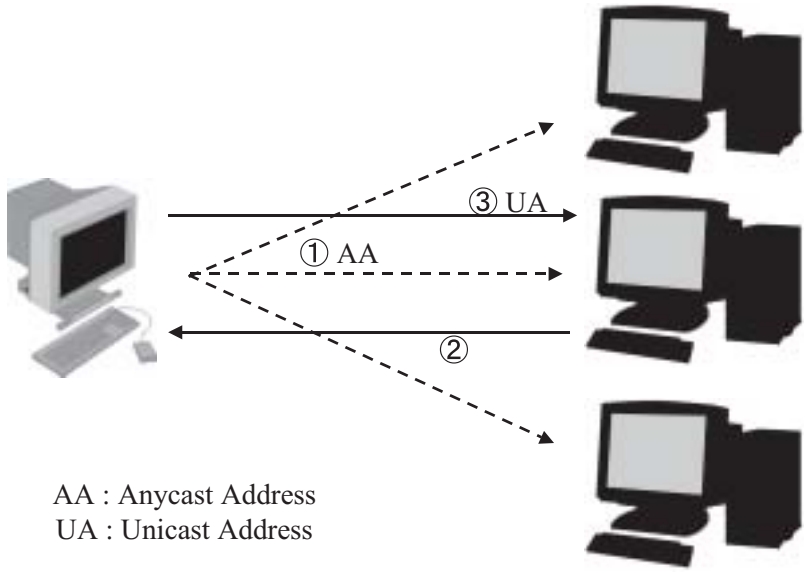

図 8: 本報告で提案するアドレス解決手法

レスへと変換する「アドレス解決」を行う新しいプロトコルである (図 9)。AARP の詳細については 3.2 節で述べる。

通信開始時にアドレス解決する単純な方法として、通信アプリケーションに実装することが考えられる。しかし、この場合はエニーキャストアドレスをサポートするためにアプリケーションを修正する必要があり、普及の大きな障害となる。そこで、本報告では通信部分の共有ライブラリをエニーキャストに対応したライブラリに置換することによって、アプリケーションのソースを改変することなくエニーキャスト通信を実現させる。

3.2 AARP によるアドレス解決

2.3 節で述べたように、エニーキャストアドレスはユニキャストアドレスのアドレス空間から選ばれるため、エニーキャストアドレスとユニキャストアドレスは区別できない。そのため、エニーキャストアドレスを対応するユニキャストアドレスに解決するためには、実際に対応する端末からのパケットを受信する必要がある。このため、アドレス解決の方法として以下の 2 つが考えられる。

1. client initiate — プローブパケット方式

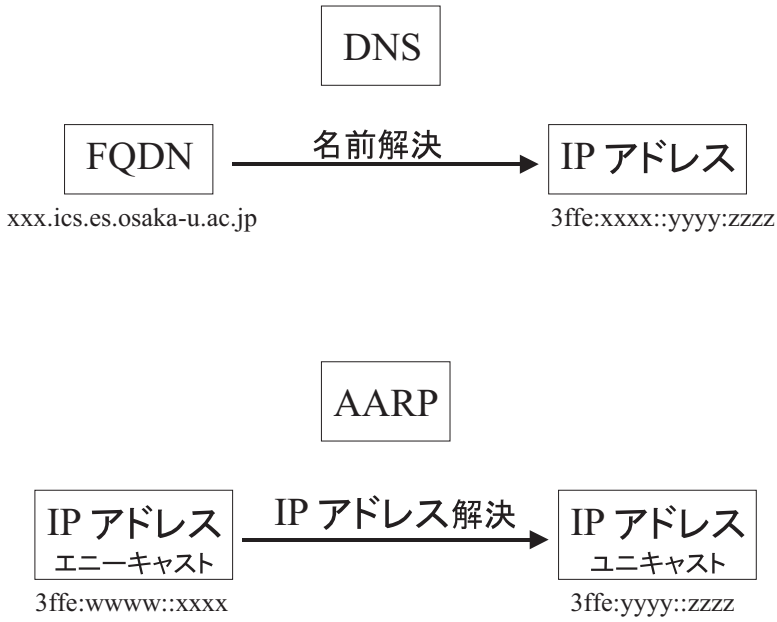

図 9: Anycast Address Resolving Protocol

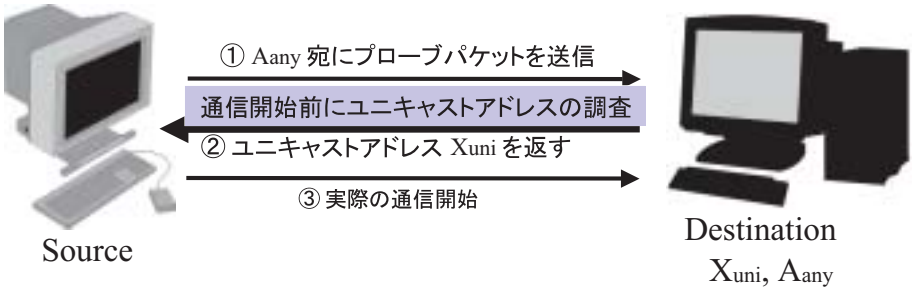

図 10: client initiate − プローブパケット方式

エニーキャストアドレスのプローブパケットを送出し、端末からの応答を受信することでユニキャストアドレスを取得する (図 10)。

2. server initiate − piggyback 方式

端末からの通信パケット上に、エニーキャストアドレスを付加させ (piggyback)、エニーキャストアドレスとユニキャストアドレスを対応させる (図 11)。

1. の方式では、プローブパケットによるトラヒックが発生し、ネットワーク資源を消費する問題がある。一方、2. ではエニーキャストアドレスを付加させるために IPv6 終点オプ

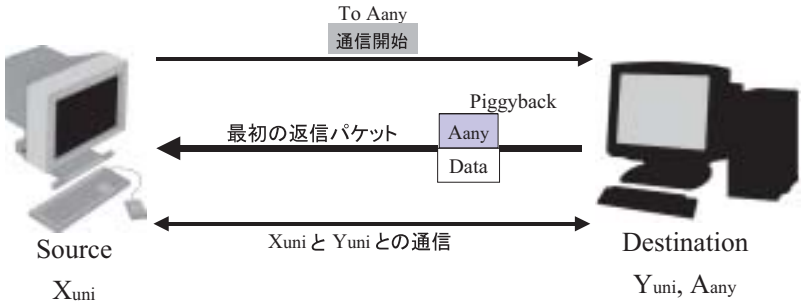

図 11: server initiate − piggyback 方式

ションヘッダを利用する [10] ことなどが必要となり、アプリケーションの修正が要求される。このため、本報告では 1. のプローブパケットを用いたアドレス解決を行う。なお、プローブパケットによるトラヒックは、エニーキャストアドレス解決のキャッシングにより軽減することが可能である。

3.3 AARP 中間ライブラリ

本報告では、アプリケーションプログラムを修正せずにエニーキャストアドレスを用いた通信をサポートするため、SOCKS-based IPv6 / IPv4 Gateway [5] 機構の実装方法として用いられている、共有ライブラリを置き換えるための中間ライブラリを作成し、アドレス解決を行う方法を提案する。以下では、この中間ライブラリを AARP ライブラリと呼ぶ。

図 12 に AARP ライブラリを使用した場合の、送受信端末のプロトコルスタックを示す。AARP ライブラリは、アプリケーションスタックとソケットスタックの間に位置し、アプリケーションスタックで指定されたエニーキャストアドレスをユニキャストアドレスに変換し、ソケットスタックへ渡す。図 12 における通信手順は以下の通りである。

1. アプリケーション内で宛先アドレスを AA とした通信が開始されるとき、通信時にコールされる API (TCP を用いたアプリケーションであれば `connect()`) によって、通常の共有ライブラリ上の関数ではなく、AARP ライブラリ上の関数が実行される。
2. AARP ライブラリは、置き換えられた関数内で AARP によってアドレスを解決する。
3. AARP ライブラリは、得られたユニキャストアドレス UA を宛先として、本来の共有

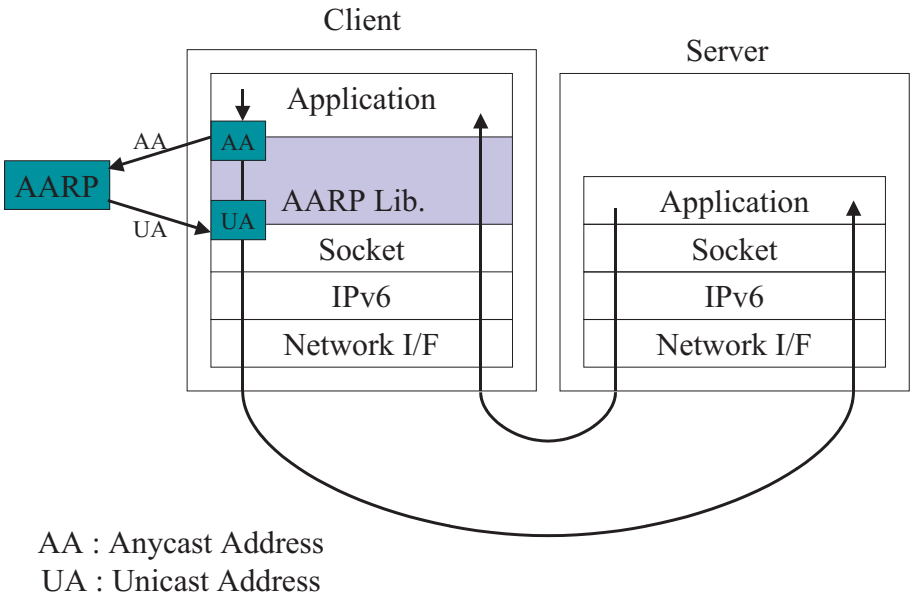

図 12: AARP ライブラリを組み込んだプロトコルスタック

ライブラリの API をコールし、通常のユニキャストアドレスに対する通信と同様の処理が行われる。

4　エニーキャストアドレスのための端末選択制御機構の提案

エニーキャストアドレスの利点として、状況に応じた最適な端末選択の実現可能性があげられる。ネットワークの状況に応じて適切な端末との通信が行えるようにするためには、エニーキャストアドレスに対する経路情報を適宜更新する必要がある。しかしながら、エニーキャストアドレスは、そのアドレスが割り当てられたインターフェースを持つノードでしか、そのアドレスがエニーキャストアドレスであることを知ることができない。したがって、このままでは、エニーキャストアドレス宛のパケットは、ユニキャストの場合と同様に既存の経路制御機構を用いて転送されることになる。そのため、ネットワークの状況に応じた動的な端末設定を実現するためには、エニーキャストアドレスに対する経路情報を動的に更新する手法が必要となる。本報告では、エニーキャストアドレスの経路制御のために、各ルータに対する経路情報の更新手法を提案する。

エニーキャストアドレスには、端末にどのようにアドレスを割り当てるかについて次の 2 つの方法がある (図 13 参照)。

1. 128 ビットすべてを用いてアドレス割り当てを行う方法
2. プレフィックス部は端末の属するサブネットのものにしたがい、インターフェース ID 部のみを用いてアドレス割り当てを行う方法

ネットワーク上の任意の場所に存在するノードに対してエニーキャストアドレスを割り当てる場合は、上記 1 のアドレス割り当てになり、同一セグメント上にのみ同じエニーキャストアドレスが割り当てられたノードが存在する場合は、上記 2 のアドレス割り当て方法になる。

エニーキャスト通信のための経路制御を考える場合、同じエニーキャストアドレスが割り当てられた複数のノードが、同一セグメント上に存在するかどうかによって経路情報の更新方法が異なるため、それぞれに分けて考える必要がある。同じエニーキャストアドレスが割り当てられたノードが同一セグメント上に存在する場合、その宛先は近隣ルータの Neighbor Cache の状態によって決定される [11]。 異なるセグメント上に存在する場合は、RIPng [12], OSPFv3 [13] などのルーティングプロトコルによって決定される。

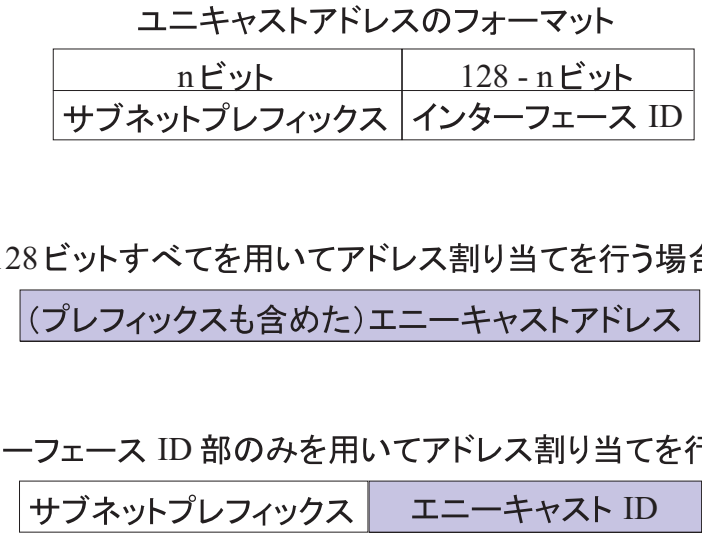

図 13: エニーキャストアドレスの割り当て方法

以下、4.1 節では近隣探索 (Neighbor Discovery) [11] によるリンク層アドレス解決について、4.2 節ではルーティングプロトコルによる経路制御について述べる。5.2 節ではさらに、エニーキャスト通信の経路制御の適用例として、エニーキャスト通信を用いた新たなサーバ負荷分散モデルを提案する。

4.1 近隣探索によるサブネット内での端末選択

IP 通信では、IP アドレスを知っていても、その IP アドレスに対応するリンク層アドレス (例えば Ethernet では MAC アドレス) を知らなければパケットを送信できない。IPv6 においては、近隣探索 [11] を用いて送信先 IP アドレスに対応するリンク層アドレスを得る。図 14 のように、近隣要請メッセージ (Neighbor Solicitation) を用いて送信先 IP アドレスに対応するリンク層アドレスを問い合わせる。問い合わせ結果は、IP アドレスとリンク層アドレスを組にしてキャッシュされる。このキャッシュを近隣キャッシュ (Neighbor Cache) と呼ぶ。これは、IPv4 における ARP (Address Resolution Protocol) テーブル [14] に対応する。

以下、送信先 IP アドレスに対応するリンク層アドレスを得る手順を示す。

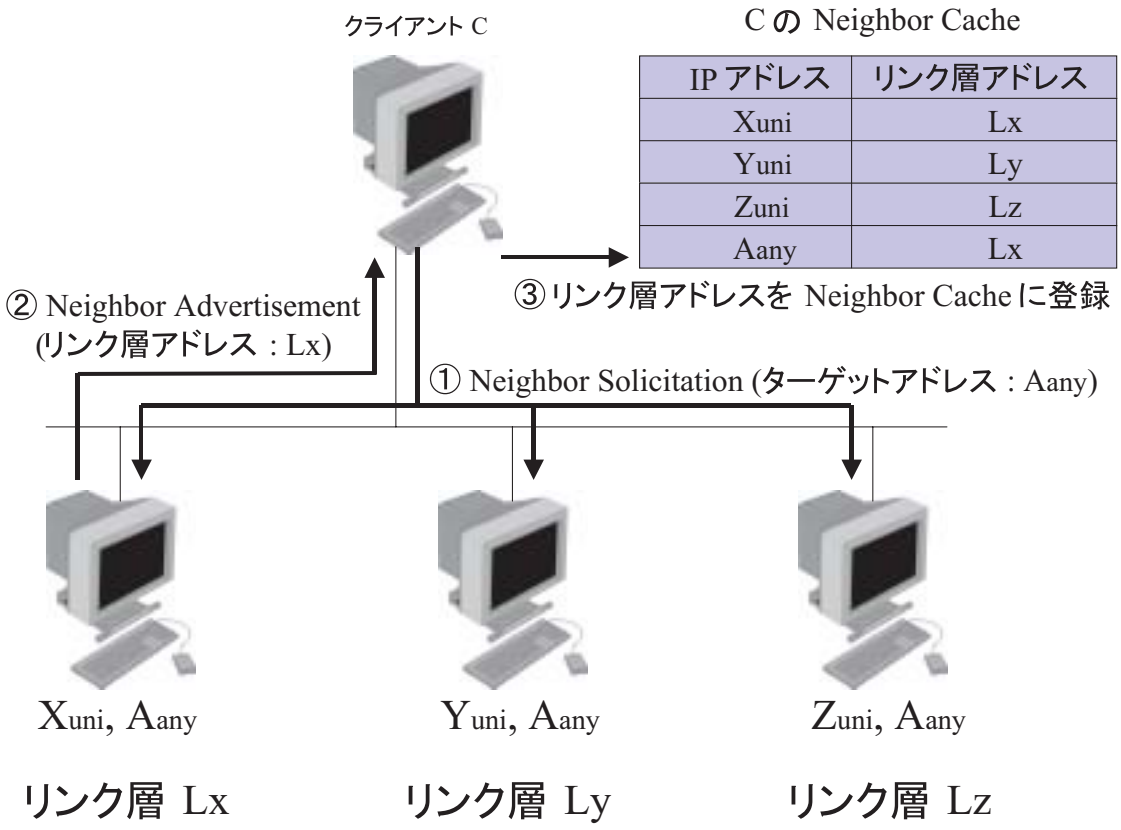

図 14: Neighbor Cache の設定方法

1. 送信先 IP アドレスに対する Neighbor Cache が存在するなら、これを参照し、リンク層アドレスを得る。
2. 該当する Neighbor Cache が存在しなければ、送信先 IP アドレスをターゲットアドレスとする近隣要請メッセージ (Neighbor Solicitation, 以下 NS) をセグメント内のすべてのノードに送信する。
3. NS を受信したノードは、ターゲットアドレスが自ノードのものであれば、NS の送信元アドレスに対し近隣広告メッセージ (Neighbor Advertisement, 以下 NA) を返す。
4. NS の送信元ノードは、NA を受信して送信先 IP アドレスのリンク層アドレスを得る。NA が複数得られた場合は最初に受信した NA のリンク層アドレスを採用する。得られたリンク層アドレスは Neighbor Cache に登録する。

同じエニーキャストアドレスを割り当てられたインターフェースを持つ複数のノードが同一セグメント上に存在する場合、そのエニーキャストアドレス宛に送信されたパケットがどのノードに届けられるかはクライアントの Neighbor Cache によって決められる。このことから、エニーキャストアドレスを持つノードがエニーキャストアドレス宛のパケットを受け取りたい場合、自発的に NA を送出して強制的にルータの Neighbor Cache を更新すればよいことになる (図 15)。この、自発的な NA を非要請近隣広告メッセージ (unsolicited Neighbor Advertisement) と呼ぶ。

通常、リンク層アドレスを取得し Neighbor Cache を更新するのは、前述の通り、NS に対する NA を受け取ったときである。これは、Neighbor Cache のエントリに登録されている情報は単に、近隣ノードの到達性、不到達性を判断するためのものであり、同一セグメント上に同じ IP アドレスを持つノードが複数存在する場合にどのノードのリンク層アドレスを登録すべきかなどは考えられていないからである。したがって、本報告では Neighbor Cache を動的に更新するために、unsolicited NA を用いることとする。

4.2 グローバルスコープにおける最適な端末の選択

IP では、送信されたパケットは複数のルータを経由して宛先端末に送られる。各ルータは、自身のルーティングテーブルの中からパケットの宛先に該当するエントリを検索し、適

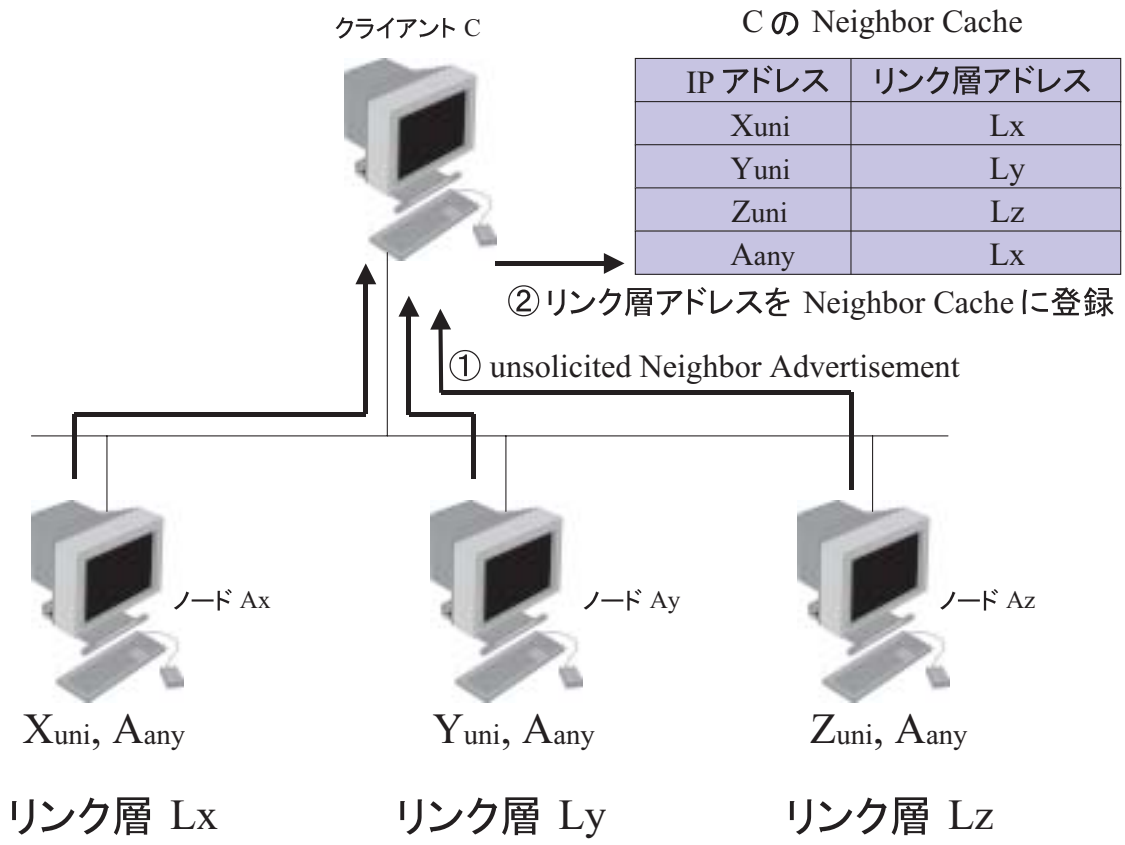

図 15: unsolicited NA による Neighbor Cache の更新

切な経路に転送する。宛先アドレスに対してルーティングテーブル内の複数のエントリが適合した場合、ルータはエントリに書かれたメトリック値 (metric) にしたがって経路を選択する。この機能については今後の課題とする。

5　AARP の実装と評価

本章では、3 章で述べた AARP ライブラリによるアドレス解決モデルを実機に実装し、評価実験を行うことによって、既存のアプリケーションのコードを改変することなくエニーキャスト通信が実現できることを示す。

以下、5.1 節では、AARP ライブラリを FreeBSD 上に実装する方法を具体的に述べる。5.2 節では、TCP を利用したアプリケーションである telnet、ftp を用いた通信が実現できていることを示し、さらに 5.3 節では UDP を利用したサービスである DNS を用いた通信が実現できていることを示す。

5.1　AARP の実装

本報告では、共有ライブラリの置換により IPv6 と IPv4 との相互通信を実現する SOCKS-based IPv6 / IPv4 Gateway [5] 機構での実装方法と同様、エニーキャストアドレス解決を実現する中間ライブラリを作成する。この方法により、図 16 のように、アプリケーション側ではエニーキャストアドレスにより通信されているように見えるが、中間ライブラリによってユニキャストアドレスに変換されて、実際はユニキャストによって通信が行われることになる。

本報告では、アドレス解決を行うプローブパケットとして、ICMPv6 [15] Echo Request および Echo Reply を用いる。これは、エニーキャストアドレス宛に Echo Request を送信し、そのエニーキャストアドレスを割り当てられているインターフェースを持つノードが自身のインターフェースに割り当てられているユニキャストアドレスを送信元とした Echo Reply を返すようにすれば、エニーキャストアドレスに対応するユニキャストアドレスが得られるためである。なお、実装で用いた FreeBSD 4.4-Release では、エニーキャストアドレス宛の Echo Request を受信したときは自身に割り当てられたユニキャストアドレスを送信元アドレスとした Echo Reply を送信するという実装になっている。

ICMPv6 Echo は、`ping6` プログラムなどで利用されている ICMPv6 の一種で、IPv6 が稼動しているノードは ICMPv6 Echo Request に対して必ず ICMPv6 Echo Reply を返さ

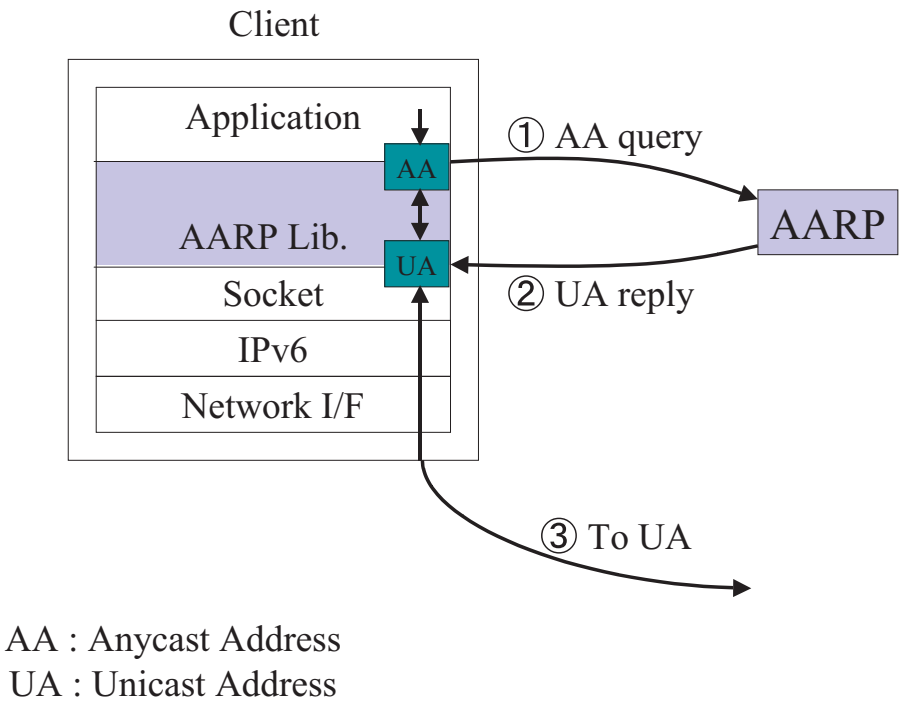

図 16: アドレス解決を行う中間ライブラリ

なければならないという通信規約がある [15]。

図17 のように、クライアント P はエニーキャストノード Q との ICMPv6 Echo Request/Reply のやり取りによって、ノード Q のユニキャストアドレスを知ることができる。

1. クライアント P からエニーキャストアドレス A_{any} を宛先として Echo Request を送信する
2. エニーキャストノード Q はパケットの送信元アドレスにエニーキャストアドレスを設定することができないため、ユニキャストアドレス Y_{uni} を送信元アドレスとした Echo Reply をノード P 宛に送信する。
3. クライアント P は、受け取った Echo Reply の送信元アドレスから、エニーキャストアドレス A_{any} に対応するユニキャストアドレスは Y_{uni} であることがわかる。

このように、ICMPv6 Echo Request/Reply を利用して通信相手のユニキャストアドレスを調べた後、得られたユニキャストアドレスを用いて通信すれば、エニーキャスト通信の問題

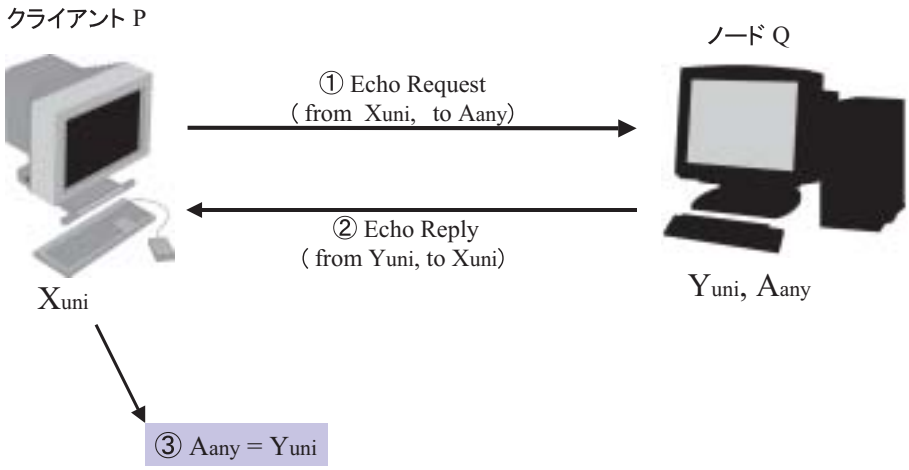

図 17: ICMPv6 Echo Request/Reply によるアドレス解決

点を回避できる。以上の手順は、API (Application Program Interface) を公開することにより実現可能であるが、ICMPv6 によるアドレス解決手法を利用するためには管理者権限が必要となる。そのため、アドレス解決部を API で提供する場合、アプリケーションが管理者権限によって動作する必要があり、既存の設定を変更する必要が生じる。また、セキュリティ上の点からも好ましくない。さらに、API による方法ではエニーキャストアドレスが見つかるたびにプローブパケットが送られるため、ネットワーク資源が浪費されることが考えられる。そこで本報告では、プローブパケットを使ったアドレス解決部分をデーモン (daemon) プログラムにより提供し、利用側にはデーモンプログラムとの対話部分の API を公開する方法を提案する。デーモンプログラムは、すでに解決済みのエニーキャストアドレスについてはキャッシングを用いて応答することによって、プローブで消費されるトラヒックを軽減することが可能である。図 18 に具体的な手順を示す。提案方式では、実際のアドレス解決を行うデーモン `aarpd` と、デーモンにアドレス解決を依頼する `aarp_resolv()` API が存在する。`aarp_resolv()` はエニーキャストアドレスを引数に取り、ユニキャストアドレスを返す API である。アプリケーションは `aarp_resolv()` API をコールし、`aarpd` にエニーキャストアドレス解決を依頼する。`aarpd` は、ICMPv6 Echo メッセージを用いてアドレスを解決し、アプリケーションにユニキャストアドレスを返す。本報告で提供する中間ライブラリを用いれば、中間ライブラリが `aarp_resolv()` によってアドレス解決を行うため、ア

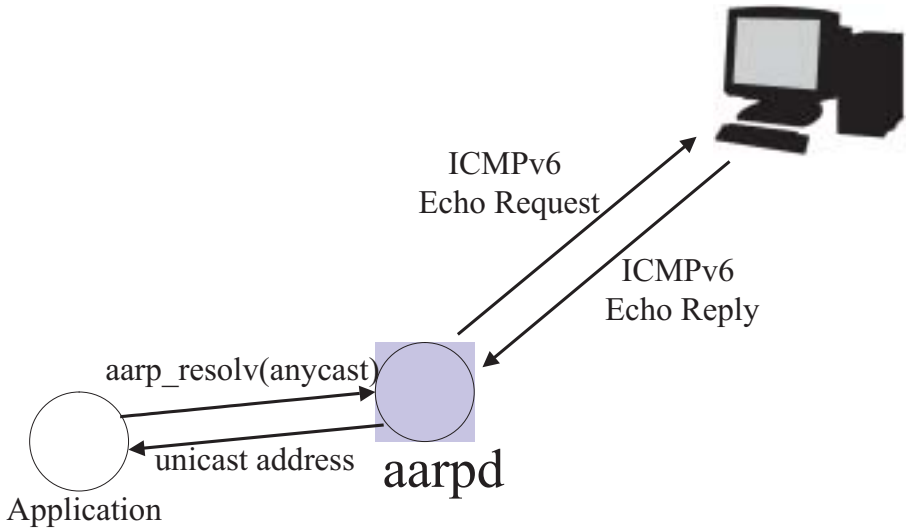

図 18: aarpd

プリケーション側でアドレス解決を行う必要はない。また、aarpd は必ずしも端末ごとに必要ではなく、例えば図 19 のように、同一セグメント内に 1 つだけ aarpd を設置すれば、さらにプローブトラヒックの量を軽減することも可能である。

提案方式では、トランスポート層のプロトコルによって通信開始時に呼び出される関数が異なるため、それぞれの場合に分けて、どの関数を置き換えるべきかを考える必要がある。これは、実装を行うシステム上で稼動しているオペレーティングシステムなどによって異なる。本報告では、FreeBSD 4.4 Release 上に実装を行った。

まず、TCP 通信の場合、通信開始時の 3 ウェイハンドシェークを起動する関数である、connect() を置き換えればよい。

次に、UDP 通信であるが、データを送受信する関数である send()、sendto()、sendmsg()、recv()、recvfrom()、recvmsg() が置き換えの候補となる。このうち、send() は sendto() を、recv() は recvfrom() をそれぞれ内部で呼んでいるため、sendto()、recvfrom() を置き換えれば send()、recv() も置き換わる。したがって、sendto()、sendmsg()、recvfrom()、recvmsg() を置き換える必要がある。

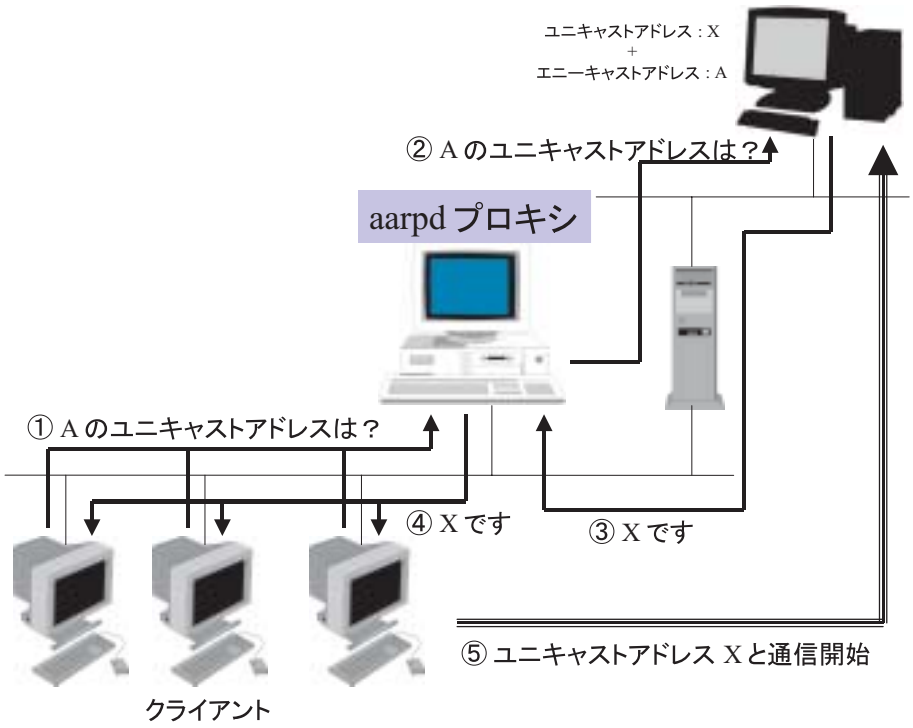

図 19: aarpd を共有する方法

5.2 TCP 通信の評価

まず、TCP を利用したアプリケーションである telnet、ftp について、図 20 のネットワーク上で、

- telnet 3ffe:fffe:2:1:ffff::1 (**エニーキャストアドレス**)
- ftp 3ffe:fffe:2:1:ffff::1 (**エニーキャストアドレス**)

をクライアントが実行したときのサーバとの接続確立までのパケットの送受信の記録を tcpdump によって出力したものを、telnet は図 21 に、ftp は図 22 に、それぞれ示す。telnet、ftp ともに、

1. ICMPv6 Echo Request/Reply により、アドレス解決をおこなう
2. 得られたユニキャストアドレスに対して実際の通信を開始する (TCP 3 ウェイハンドシェークの開始)

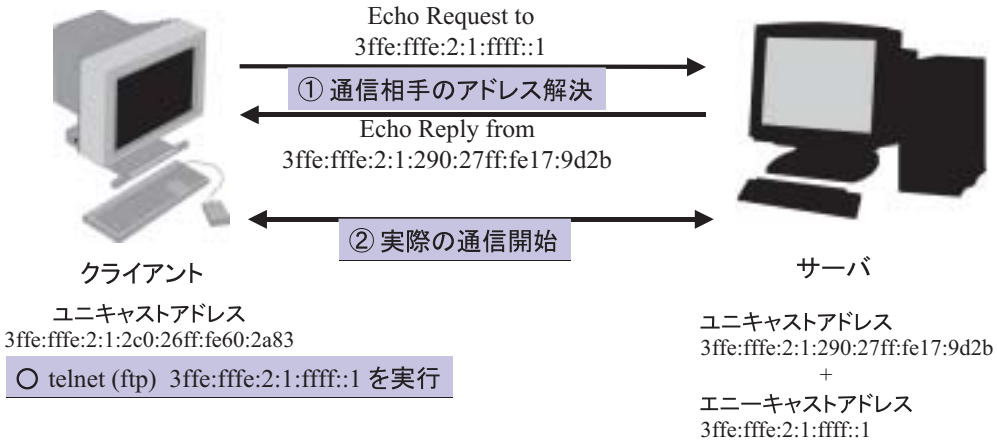

図 20: ネットワーク構成

```
06:41:12.489877 3ffe:fffe:2:1:2c0:26ff:fe60:2a83 > 3ffe:fffe:2:1:ffff::1: icmp6:  echo request
06:41:12.490032 3ffe:fffe:2:1:290:27ff:fe17:9d2b > 3ffe:fffe:2:1:2c0:26ff:fe60:2a 83: icmp6: echo reply
```

① ICMPv6 Echo Request / Reply によるエニーキャストアドレスからユニキャストアドレスへの変換

```
06:41:12.490924 3ffe:fffe:2:1:2c0:26ff:fe60:2a83.1447 > 3ffe:fffe:2:1:290:27ff:fe 17:9d2b.23: S
06:41:12.491076 3ffe:fffe:2:1:290:27ff:fe17:9d2b.23 > 3ffe:fffe:2:1:2c0:26ff:fe60 :2a83.1447: S ack
06:41:12.491126 3ffe:fffe:2:1:2c0:26ff:fe60:2a83.1447 > 3ffe:fffe:2:1:290:27ff:fe 17:9d2b.23: . ack
```

② ユニキャストアドレスへの TCP 3 ウェイハンドシェーク

図 21: `telnet` の結果

```
07:32:48.700560 3ffe:fffe:2:1:2c0:26ff:fe60:2a83 > 3ffe:fffe:2:1:ffff::1: icmp6:  echo request
07:32:48.700716 3ffe:fffe:2:1:290:27ff:fe17:9d2b > 3ffe:fffe:2:1:2c0:26ff:fe60:2a 83: icmp6: echo reply
```

① ICMPv6 Echo Request / Reply によるエニーキャストアドレスからユニキャストアドレスへの変換

```
07:32:48.701590 3ffe:fffe:2:1:2c0:26ff:fe60:2a83.1449 > 3ffe:fffe:2:1:290:27ff:fe 17:9d2b.21: S
07:32:48.701743 3ffe:fffe:2:1:290:27ff:fe17:9d2b.21 > 3ffe:fffe:2:1:2c0:26ff:fe60 :2a83.1449: S ack
07:32:48.701795 3ffe:fffe:2:1:2c0:26ff:fe60:2a83.1449 > 3ffe:fffe:2:1:290:27ff:fe 17:9d2b.21: . ack
```

② ユニキャストアドレスへの TCP 3 ウェイハンドシェーク

図 22: `ftp` の結果

という手続きを行っている。

この結果から、telnet、ftp のいずれも、ICMPv6 Echo Request/Reply によってアドレス解決を行い、ユニキャストアドレスに対して TCP 3 ウェイハンドシェークを行っていることが確認できた。

5.3 UDP 通信の評価

次に、DNS サーバにエニーキャストアドレスを割り当て、クライアントが通信を行う際のホスト名解決を行っている様子を、図 23 のネットワークでの実験によって示す。

クライアント端末の /etc/resolv.conf(FreeBSD において DNS サーバのアドレスを設定するファイル) には、

- 3ffe:fffe:2:1:ffff::5

と書いてあり、図 23 は、このクライアントが

- telnet xxx.yyy.zzz (FQDN)

を実行したときの様子を表している。その通信手順は以下の通りである。

1. 自ノードに設定してある DNS サーバのエニーキャストアドレスに対し、ICMPv6 Echo Request/Reply によるアドレス解決を行う
2. 得られた DNS サーバのユニキャストアドレスに対して、ホスト名 xxx.yyy.zzz の名前解決の問い合わせを行う。
3. 名前解決によって得られた通信相手の IP アドレスに対し、ICMPv6 Echo Request/Reply によるアドレス解決を行う
4. 得られた通信相手のアドレスに対して実際の通信を開始する

この、ホスト名解決を行ってから xxx.yyy.zzz と通信を開始するまでのパケットの送受信の記録を tcpdump による出力として、図 24 に示す。 この結果から、DNS サーバの IP アドレスとしてエニーキャストアドレスを指定したホスト名解決を実現できることが確認できた。

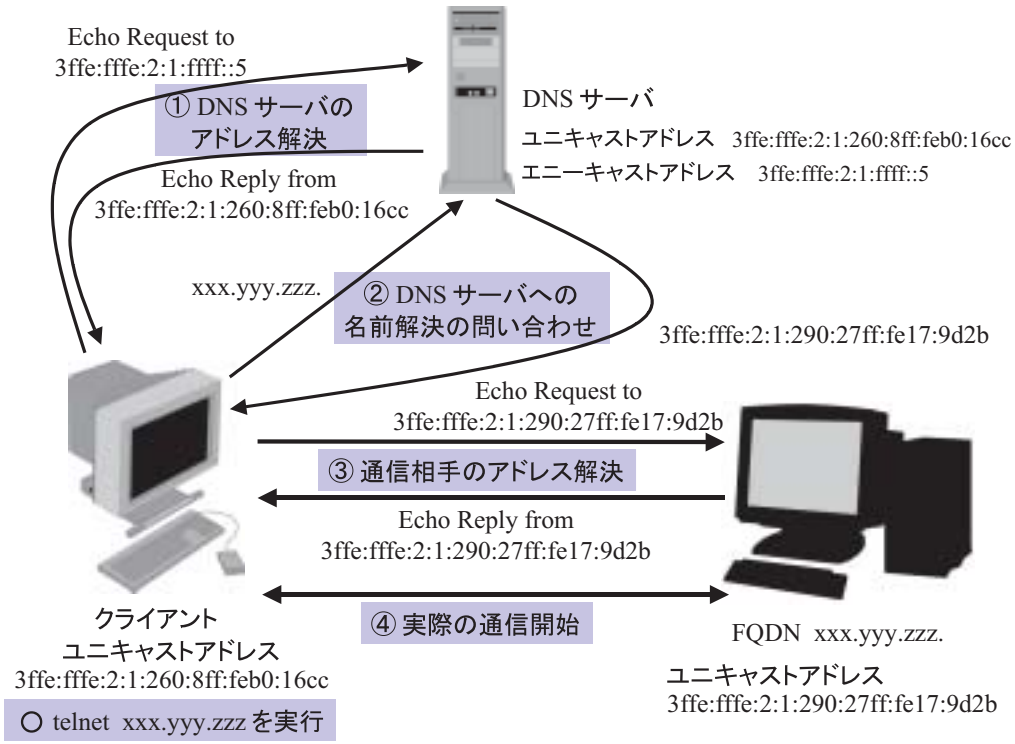

図 23: ネットワーク構成 (DNS サーバがある場合)

```
07:57:43.162472 3ffe:fffe:2:1:260:8ff:feb0:16cc > 3ffe:fffe:2:1:ffff::5: icmp6: e cho request
07:57:43.162653 3ffe:fffe:2:1:2c0:26ff:fe60:2a83 > 3ffe:fffe:2:1:260:8ff:feb0:16c c: icmp6: echo reply
```

① ICMPv6 Echo Request / Reply によるエニーキャストアドレスからユニキャストアドレスへの変換
(DNS サーバの IP アドレス解決)

```
07:57:43.165163 3ffe:fffe:2:1:260:8ff:feb0:16cc.3834 > 3ffe:fffe:2:1:2c0:26ff:fe6 0:2a83.53:  61090+[|domain]
07:57:43.166021 3ffe:fffe:2:1:2c0:26ff:fe60:2a83.53 > 3ffe:fffe:2:1:260:8ff:feb0: 16cc.3834:  61090*[|domain]
07:57:43.183768 3ffe:fffe:2:1:260:8ff:feb0:16cc.3836 > 3ffe:fffe:2:1:2c0:26ff:fe6 0:2a83.53:  61091+[|domain]
07:57:43.184295 3ffe:fffe:2:1:2c0:26ff:fe60:2a83.53 > 3ffe:fffe:2:1:260:8ff:feb0: 16cc.3836:  61091*[|domain]
```

② DNS への問い合わせ

```
07:57:43.187547 3ffe:fffe:2:1:260:8ff:feb0:16cc > 3ffe:fffe:2:1:290:27ff:fe17:9d2 b: icmp6: echo request
07:57:43.187703 3ffe:fffe:2:1:290:27ff:fe17:9d2b > 3ffe:fffe:2:1:260:8ff:feb0:16c c: icmp6: echo reply
```

③ ICMPv6 Echo Request / Reply によるエニーキャストアドレスからユニキャストアドレスへの変換
(通信相手の IP アドレス解決)

```
07:57:43.189940 3ffe:fffe:2:1:260:8ff:feb0:16cc.1037 > 3ffe:fffe:2:1:290:27ff:fe1 7:9d2b.23: S
07:57:43.190159 3ffe:fffe:2:1:290:27ff:fe17:9d2b.23 > 3ffe:fffe:2:1:260:8ff:feb0: 16cc.1037: S ack
07:57:43.190236 3ffe:fffe:2:1:260:8ff:feb0:16cc.1037 > 3ffe:fffe:2:1:290:27ff:fe1 7:9d2b.23: . ack
```

④ ユニキャストアドレスへの TCP 3 ウェイハンドシェーク

図 24: DNS 名前解決の結果

DNS サーバにエニーキャストアドレスを割り当て、ホスト名解決を問い合わせることができるというのは、本提案通信モデルの大きな利点である。世界中のすべての DNS サーバに、ある決まったエニーキャストアドレスを割り当てておけば、クライアント端末のオペレーティングシステム上であらかじめ DNS サーバのアドレスを設定することができる。そのため、ネットワーク管理者が DNS サーバのアドレスを設定することなく、プラグアンドプレイ時に最適な DNS サーバを利用することができるようになる。

6　エニーキャスト経路制御によるサーバ負荷分散モデルの評価

本章では、4 章で提案したエニーキャストアドレスに対する経路制御機構の適用例として、同一セグメント内に 2 台のサーバを設置した場合のエニーキャストアドレスによる負荷分散モデルを示し、シミュレーションによる評価実験を行うことで有効性を示す。

6.1　サーバの負荷に応じた Neighbor Cache 更新手法

4.1 節で述べたとおり、各サーバは unsolicited NA を送出することで動的にルータの Neighbor Cache を更新することから、各サーバが自身の負荷が小さい場合に unsolicited NA を送出すれば、エニーキャストアドレス宛のパケットは負荷が小さいサーバに届けられることになる。

unsolicited NA 送出を決定する方法として、

1. 自身のサーバの負荷情報のみを利用する場合
2. 他サーバからの unsolicited NA 情報を利用する場合

が考えられる。1 はサーバ自身の情報のみを利用するため、簡単に実装可能であるが、他のサーバからの unsolicited NA の送出状況が把握できないため、サーバが送出した unsolicited NA が他サーバの unsolicited NA によって直後に上書きされる場合など、有効に動作しない場合も考えられる。一方、2 では、他サーバの負荷情報も考慮に入れながら unsolicited NA を送出できるため効率的であるが、他からの unsolicited NA も同時に監視する必要があるため、実装が複雑になる。以下で、それぞれの方法について具体的な送出時間の決定方法を述べる。

6.1.1　自サーバの負荷情報のみを利用する場合

サーバは自身の負荷情報のみをもとに、次に unsolicited NA を送信するまでの時間 T を決める。T は次の式によって与えられるものとする。

$$T = \frac{T_{def}}{1 - \rho} \qquad (0 \leq \rho < 1) \tag{1}$$

ここで ρ は、サーバの負荷を表す値で、具体的には CPU 利用率などである。また、T_{def} は、$\rho = 0$ の場合、つまりサーバの負荷がまったくない場合に、次に unsolicited NA を送信するまでの時間である。ここでは、標準的な NA の送出間隔を用いる。

各サーバは、以下の処理を繰り返す。

1. unsolicited NA を送信する。
2. 自ノードの CPU 利用率から式 (1) を用いて T をもとめ、T 秒間待つ。

これにより、負荷の高いサーバの unsolicited NA 送出頻度は低下し、より負荷の小さなサーバが頻繁に unsolicited NA を送信するようになるため、クライアントの Neighbor Cache には、負荷の小さなサーバのアドレスが登録される割合が高くなる。したがって、エニーキャストアドレス宛のパケットは、より負荷の小さなサーバへと送信されると考えられる。

6.1.2 他サーバの unsolicited NA を利用する場合

セグメント内のすべてのノードに unsolicited NA を送信することで、他のエニーキャストサーバの送信した unsolicited NA を受信することができる。この情報を利用すれば、各エニーキャストサーバは自身の負荷情報に加え、他のサーバから送信された unsolicited NA から得られる情報をもとに、次に unsolicited NA を送信するまでの時間 T を決めることができる。T は、unsolicited NA をクライアントにのみ送信する場合に用いたのと同様、式 (1) を利用する。

ところが、他のサーバから送信された unsolicited NA 自体からは、その送信元サーバのリンク層アドレスしかわからない。わかるのはその unsolicited NA を受け取った時点でクライアントの Neighbor Cache に登録されたリンク層アドレスだけである。

そこで、セグメント内に存在するすべてのエニーキャストサーバのうち、最も負荷の小さいサーバのみが unsolicited NA を送信するようにするため、各サーバは、以下のようにして unsolicited NA を送信するかどうかを判断し、必要であれば送信する。

1. (自身も含めた) エニーキャストサーバから unsolicited NA を受信する。
2. 過去の平均 CPU 利用率から、次に NA を送信するまでの時間 T を式 (1) によりもと

める。

3. T 秒間待ち、unsolicited NA を送信する。もし待っている間に他のサーバから unsolicited NA を受信すれば unsolicited NA は送信せず、1. からの処理を繰り返す。

このように実装を行えば、T 秒おきに最も負荷の小さなサーバから unsolicited NA を送信させることができる。

6.2 経路制御を用いたサーバ負荷分散モデルの性能評価

本節では、6.1 節で提案した負荷分散モデルの有効性をシミュレーションおよび実機による評価を用いて示す。はじめに、シミュレーションにより unsolicited NA 送出時刻 T を負荷に応じて決定することで、負荷分散が行われることを示す。

6.2.1 シミュレーションによる負荷分散モデルの性能評価

6.1 節で述べた unsolicited NA を用いた Neighbor Cache 更新方式を、待ち行列を用いたシミュレーションにより評価する。

まず前提条件として、

- サービス要求は到着順 (FIFO) で処理される
- サービス要求は λ ポアソン到着とし、そのレートを λ とする
- 各サーバにおけるサービス時間は平均 $\frac{1}{\mu}$ の指数分布に従う

を仮定する。また簡単のため、ここではサーバが 2 台の場合の性能について評価する。

以上の条件により、提案方式は各サーバからの NA 送出によってサービスの処理サーバが変化する、2 つの $M/M/1$ 待ち行列によりモデル化される (図 25)。また、この時の処理サーバの変遷を図 26 (サーバの負荷情報のみを用いる場合)、図 27 (他ノードの unsolicited NA を利用する場合) にそれぞれ示す。

また、本報告では比較のため以下の 2 つのモデルの評価結果も示す。

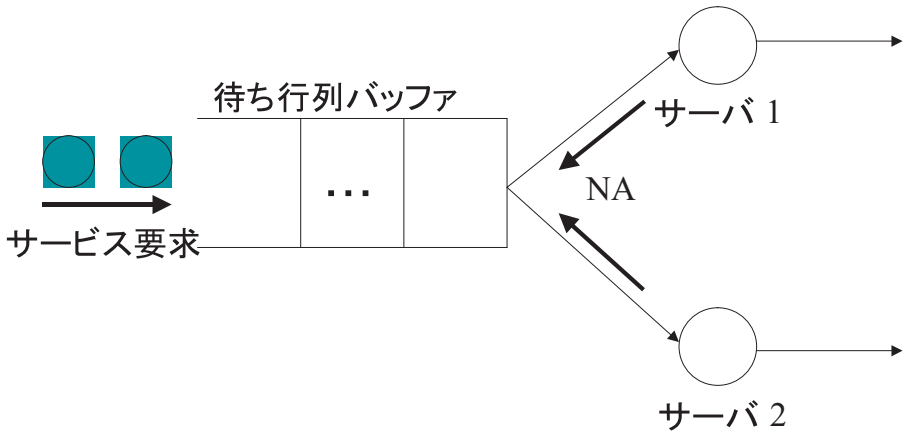

図 25: 提案するサーバ負荷分散モデル

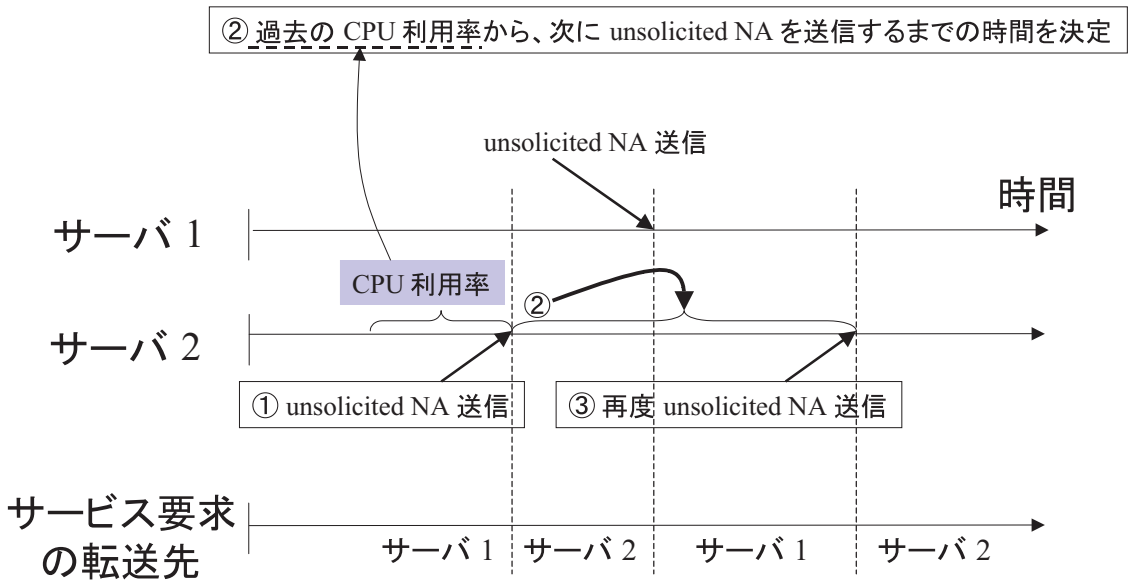

図 26: unsolicited NA による処理サーバの変化

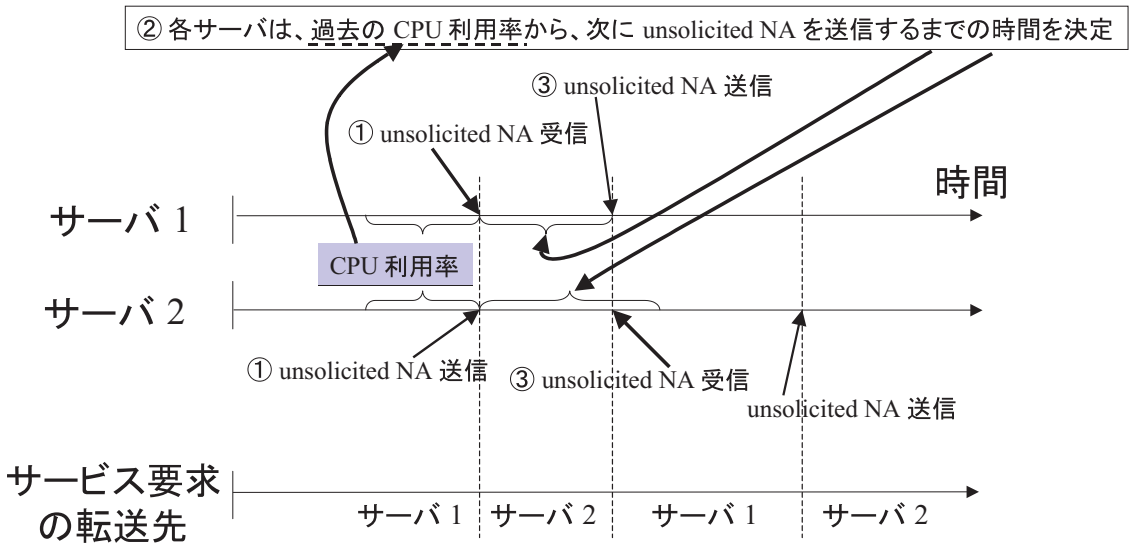

図 27: セグメント内の全ノードの Neighbor Cache の変化

- $M/M/2$ 待ち行列

  到着したサービス要求が、2 つのサーバのうち空いているサーバによって処理されるモデル (図 28) 。2 つのサーバが存在する場合に最も性能が高くなる理想的なモデルを表す。ただし、今回の場合のように 2 つのサーバが異なる端末である場合、このモデルを実現することは実際には困難であり、提案手法がどの程度 $M/M/2$ システムに近い性能が得られるかが重要となる。

- 2 つの $M/M/1$ 待ち行列 ($M/M/1 \times 2$)

  到着したサービス要求が、2 つの待ち行列に対してランダムに振り分けられるモデル (図 29)。これは、負荷情報をまったく考慮しないモデルで、DNS のラウンドロビンによる負荷分散がこのモデルに相当する。

以上の 4 つの待ち行列システム、

- $M/M/2$ 待ち行列システム
- $M/M/1 \times 2$ 待ち行列システム
- 提案方式 (負荷情報のみの場合)

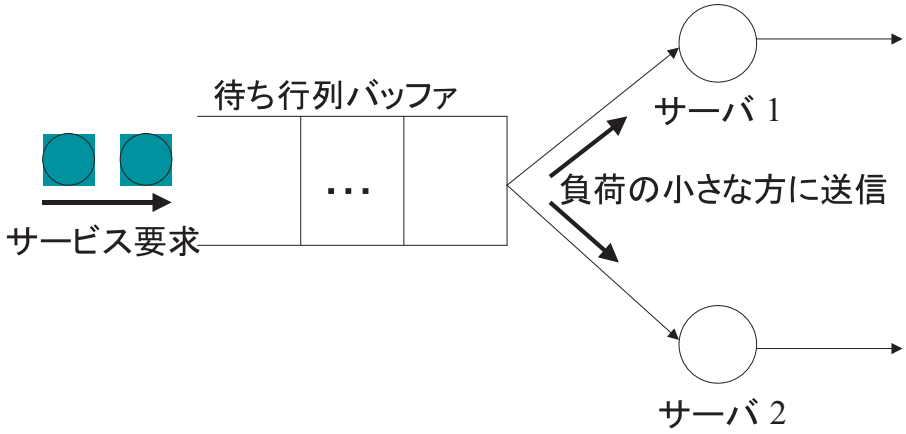

図 28: $M/M/2$ 待ち行列システム

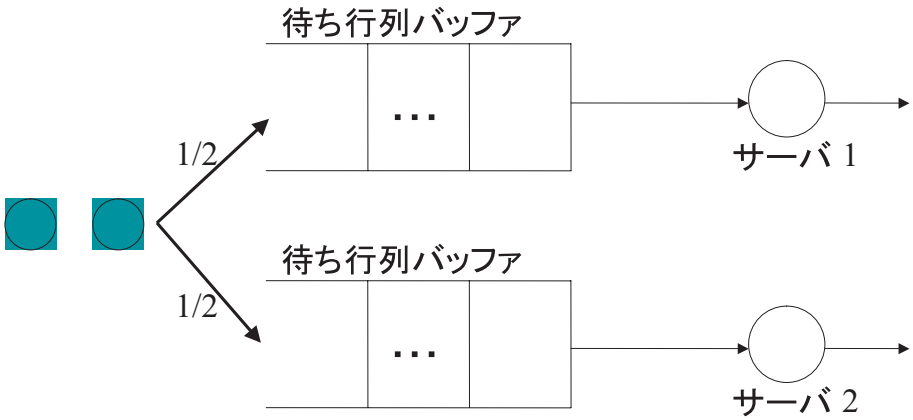

図 29: $M/M/1 \times 2$ 待ち行列システム

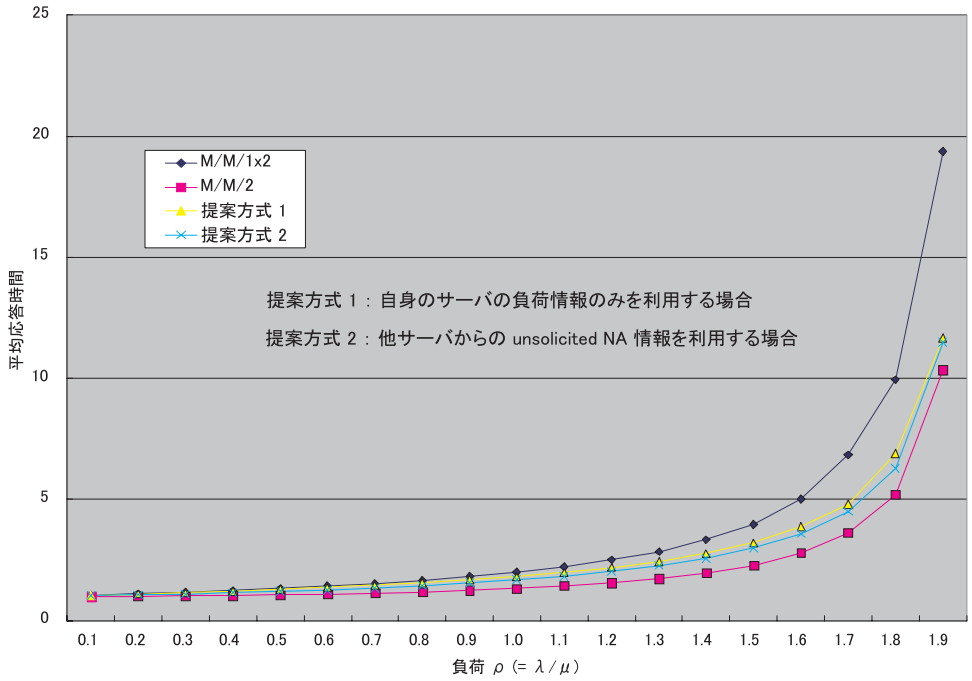

図 30: シミュレーション結果

- 提案方式 (他サーバの unsolicited NA を利用する場合)

を、待ち行列シミュレーションプログラムを用いて評価した。シミュレーションは、負荷 ρ $(= \frac{\lambda}{\mu})$ を 0.1 から 1.9 まで 0.1 刻みで変えて行い、それぞれの場合における平均応答時間を記録した。その結果を図 30 に示す。図 30 より、提案方式を用いた場合の平均応答時間は、サーバの負荷を考慮しない $M/M/1 \times 2$ 待ち行列システムよりも短くなり、$M/M/2$ 待ち行列システムの値に近づいている。このことから、提案方式の有効性が確認できる。また、他サーバの unsolicited NA を利用することで、さらに応答時間が改善されていることが分かる。しかしながら、その改善の割合はごく限られたものであることから、実装が単純なサーバ負荷情報のみを利用する場合も有効であることがいえる。ただし、unsolicited NA を利用する場合は、その利用方法によってさらなる性能向上が期待できる。これについては今後の課題である。なお、本提案方式の実装もすでに完了しており、今後は実機上での評価実験を行っていく予定である。

7　まとめと今後の課題

本報告では、IPv6 で新たに導入されたエニーキャスト機能について、その問題点に関する考察を行った。その結果、既存アプリケーションはエニーキャストアドレスを利用した通信ができないという問題があることを示した。そこでこの問題点を解決するため、AARP という、エニーキャストアドレスをユニキャストアドレスへ変換し、実際の通信は変換後のユニキャストアドレスを用いて行う、新たなプロトコルを提案した。また、実装方法を示し、既存のアプリケーションプログラムを変えることなくエニーキャストアドレスを用いた通信を行うことができることも明らかにした。さらに、エニーキャストアドレスを用いた通信の利用分野の 1 つとして考えられる、サーバの負荷分散への応用を主眼として、エニーキャスト通信のための経路制御に関する考察を行い、新たな負荷分散方式を提案し、その有効性をシミュレーションによって確認した。本提案方式の実装もすでに完了しており、今後実機上での評価を進めていく予定である。

本報告で実験に用いたネットワークは非常に小さな構成であるため、エニーキャストアドレスを広域ネットワーク上で利用した場合にどのような問題が起こるかについての実験は行っていない。また、ICMPv6 Echo Request/Reply を通信前にやり取りするためのメッセージ量の把握や、それを軽減するためにキャッシュを行う場合のキャッシュの保持時間などは、実際のネットワーク上でその影響を計測し、明らかにしていかなければならない。また、unsolicited NA による Neighbor Cache 更新手法に関しても、unsolicited NA を送信する間隔を決定するパラメータ設定方法を明らかにし、提案方式を端末に実装し実際のネットワーク上で計測を行い、その有効性を確かめる必要がある。

謝辞

本報告を終えるにあたり、御指導、御教授を頂いた宮原秀夫教授に深く感謝致します。また、本報告において終始直接御指導頂いた村田正幸教授、日頃から熱心なご指導をして頂いた日本電気株式会社の北村浩氏、大阪市立大学の阿多信吾助手に深く感謝致します。

並びに日本電気株式会社ネットワークス開発研究所の皆様には熱心なご指導を授かり心からお礼申し上げます。

また、日頃から適切な助言を頂いた大阪大学サイバーメディアセンターの下條真司教授、大阪府立看護大学の菅野正嗣助教授、大阪大学大学院基礎工学研究科の若宮直紀講師、大阪大学サイバーメディアセンターの大崎博之助手、長谷川剛助手、奈良先端科学技術大学院大学の奥田剛助手、経済学部の荒川伸一助手、国際公共政策研究科の植田和憲助手に心から感謝致します。

最後に、御協力頂いた宮原研究室および村田研究室の皆様に心からお礼申し上げます。

参考文献

[1] J. Postel, "Internet Protocol," RFC791, Sept. 1981.

[2] V. Fuller, T. Li, J. Yu, K. Varadhan, "Classless Inter-Domain Routing (CIDR): an Address Assignment and Aggregation Strategy," RFC1519, Sept. 1993.

[3] K. Egevang, P. Francis, "The IP Network Address Translator (NAT)," RFC1631, May 1994.

[4] S. Deering, R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification," RFC2460, Dec. 1998.

[5] H. Kitamura, "A SOCKS-based IPv6/IPv4 Gateway Mechanism," RFC3089, Apr. 2001.

[6] H. Kitamura, "Transition to the IPv6 network enviroment by using the SOCKS-based IPv6/IPv4 Translator," in *INET99*, Jan. 1999.

[7] H. Schulzrinne, S. Casner, R. Frederick, V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications," RFC1889, Jan. 1996.

[8] Jun-ichiro itojun Hagino, K. Ettikan, "An analysis of IPv6 anycast," draft-itojun-ipv6-anycast-analysis-02, Aug. 2001.

[9] J. Postel, "Transmission Control Protocol," RFC793, Sept. 1981.

[10] J. Bound, P. Roque, "IPv6 Anycasting Service: Minimum requirements for end nodes," draft-bound-anycast-00.txt (expired), Nov. 1987.

[11] T. Narten, E. Nordmark, W. Simpson, "Neighbor Discovery for IP Version 6 (IPv6)," RFC2461, Dec. 1998.

[12] G. Malkin, R. Minnear, "RIPng for IPv6," RFC2080, Jan. 1997.

[13] R. Coltun, D. Ferguson, "OSPF for IPv6," RFC2740, Dec. 1999.

[14] David C. Plummer, "An Ethernet Address Resolution Protocol – or – Converting Network Protocol Addresses to 48.bit Ethernet Address for Transmission on Ethernet Hardware," RFC826, Nov. 1982.

[15] A. Conta, S. Deering, "Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification," RFC2463, Dec. 1998.