次世代インターネットにおける
リンクステート型経路制御機構の実現

指導教員
村井 純

慶應義塾大学 環境情報学部
79552118
村井研究室
小原 泰弘

1999年 3月31日

概 要

インターネットにおいて通信環境と経路制御は深く関わっている。本研究では通信環境を改善するため、現状の問題点を解決した経路制御機構を設計、実装した。

インターネットのアーキテクチャでは、経路制御が重要な役割を持っている。通信の経路は、中継ノードであるルータが決定する。経路制御が正常に動作しない場合は通信経路が確立できない。また、経路制御機構の変更なくしては実現できない機能が多くある。これらのことから、経路制御に関する研究は重要である。

現在のインターネットの拡大から、インターネットプロトコルのアドレス空間が不足する事が予想されている。これを解決するため、新しいインターネットプロトコルであるIPv6が開発され、移行が進んでいる。

IPv6ネットワークにおいて、経路制御トラフィックが多い、経路収束が遅いなどの問題点が存在する。本研究では、これらを解決するためにIPv6をサポートしたOSPFv3プロトコルを実装し、評価した。

本研究によって、IPv6の経路制御の問題点が多く解決され、IPv6ネットワークにおける通信環境が改善される。これによりIPv6への移行を促進する事が予想される。IPv6を利用したネットワークは、アドレス空間の制限を考慮する必要がない事から、従来のIPv4ネットワークに比べ容易に拡大できるようになる。これらの事から、本研究はインターネットの拡大、発展を助ける。

Abstract

Routing is deeply concerned with communication environment on the Internet. To improve this environment, a routing structure which solves the present problems is designed and implemented in this research.

Routing plays important role in the Internet architecture. The route of a communication is decided by intermediate nodes, called routers. When routing doesn't work, all communications through this network cannot be made. Also, there are some functions that need changes of routing structure. Therefore, research for routing structure is important.

Insufficiency of address for Internet Protocols is caused by the rapid growth of the Internet. To solve this, a new Internet Protocol called IPv6 has been developed, and transition is in progress.

However, there are still some problems in IPv6 networks, that is much routing traffic, slow convergence, and so on. These problems are solved in this research by implementing and evaluating OSPFv3 which supports IPv6, as known as OSPFv6.

Many routing issues in IPv6 is solved, so communication environment over IPv6 is improved. This will advance transition to IPv6. Network using IPv6 is easily enlarged, because of no limitation of address space. Therefore, This research helps Internet to expand, and to develop.

目 次

第 1 章　序論 **1**

1.1 はじめに 1

1.2 本論文の構成 2

第 2 章　経路制御の現状とその問題点 **3**

2.1 現在のインターネットにおける問題点とその解決手法 3

2.1.1 IPv4 の問題点 3

2.1.2 CIDR による解決手法 3

2.1.3 NAT による解決手法 3

2.1.4 IPv6 による解決手法 4

2.2 IPv6 ネットワークにおける問題点 4

2.2.1 プロトコルの汎用性 6

2.2.2 経路の収束速度 8

2.2.3 経路制御トラフィックの量 8

2.2.4 ネットワーク管理者の意向を正確に反映できる機構の有無 . 9

2.3 本研究における解決案 10

2.3.1 IPv6 経路制御環境の問題点のまとめ 10

2.3.2 他の経路制御プロトコルの実装 10

2.3.3 OSPF の利用による RIP の問題点の解決 10

第 3 章　IPv6 をサポートした OSPF 経路制御プロトコルの設計 **11**

3.1 OSPF の概要 11

3.2 IPv6 をサポートした OSPF の設計 12

3.2.1 サブネット毎の処理からリンク毎の処理へ 12

3.2.2 トポロジの広告とアドレスの広告の分離 14

3.2.3 flooding スコープの追加 17

3.2.4 その他 17

3.3 OSPFv3 の動作概要 18

3.3.1 到達性の監視と非到達性の検知 18

3.3.2 ネットワークトポロジの広告 18

3.3.3 ネットワークアドレスの広告 19

3.3.4 広告の伝播と同期 . . . 20
3.3.5 ネットワークトポロジの計算 . . . 21
3.3.6 経路表の計算 . . . 22
3.4 OSPF データ構造体 . . . 22
3.4.1 Top Level データ構造体 . . . 22
3.4.2 Area データ構造体 . . . 23
3.4.3 Interface データ構造体 . . . 23
3.4.4 Neighbor データ構造体 . . . 24
3.5 OSPF プロトコルの必要性 . . . 25

第 4 章 OSPFD の設計 26
4.1 設計目標 . . . 26
4.2 設計概要 . . . 26
4.3 動作設計 . . . 28
4.4 Message Handler イベント群 . . . 30
4.5 LSA Handler イベント群 . . . 30
4.6 Hello Protocol イベント群 . . . 31
4.7 Interface イベント群 . . . 31
4.8 Neighbor イベント群 . . . 31
4.9 Routing Table Calculation イベント群 . . . 32

第 5 章 OSPFD の実装 33
5.1 実装目標 . . . 33
5.2 実装 . . . 33
5.3 状態遷移 . . . 33
5.4 Message Handler . . . 34
5.5 Hello Protocol . . . 37
5.6 Interface イベント . . . 37
5.7 Neighbor イベント . . . 38
5.8 LSA Handler . . . 38
5.8.1 LSA 内部表現 . . . 38
5.8.2 LSA データベース . . . 39
5.8.3 LSA Age . . . 39
5.8.4 LSA インストール . . . 40
5.9 経路表計算 . . . 41
5.9.1 Routing Table Calculation イベント . . . 41
5.9.2 Shortest Path Tree . . . 41

第 6 章　評価 **44**
6.1　評価環境 44
6.2　実装評価 46
6.3　プロトコル評価 49

第 7 章　結論 **51**
7.1　本研究の概要 51
7.2　まとめ 52
7.3　今後の課題 52

謝辞 **53**

参考文献 **55**

図 目 次

2.1 AS 間経路制御 5
2.2 AS 内経路制御 5
2.3 経路集約の例 7
2.4 RIPng の多段階層 7

3.1 OSPFv2 が正確に経路制御できない例 12
3.2 トポロジ計算結果 13
3.3 OSPFv2:OSPF メッセージヘッダフォーマット 14
3.4 OSPFv3:OSPF メッセージヘッダフォーマット 14
3.5 OSPFv2:LSA ヘッダフォーマット 14
3.6 OSPFv3:LSA ヘッダフォーマット 15
3.7 OSPFv2:Router-LSA フォーマット 15
3.8 OSPFv3:Router-LSA フォーマット 15
3.9 OSPFv2:Network-LSA フォーマット 15
3.10 OSPFv3:Network-LSA フォーマット 16
3.11 マルチアクセスネットワークにおける Adjacency の確立 21
3.12 OSPF データ構造体 22
3.13 Interface 状態遷移図 23
3.14 Neighbor 状態遷移図 24

4.1 OSPFD の構造 28
4.2 OSPFD の動作 29

5.1 Message Handler 35
5.2 LSA の入れ換え 41

6.1 評価環境 44
6.2 ルータ R1 における Neighbor の出力 47
6.3 ルータ R2 における Neighbor の出力 47
6.4 ルータ R3 における Neighbor の出力 47
6.5 ルータ R1 における経路表の出力 47
6.6 ルータ R2 における経路表の出力 48

6.7 ルータ R3 における経路表の出力 48
6.8 R3 のed1 の cost が 10 の場合のR1 の経路表 49
6.9 R2 のep0 の cost が 10 の場合のR1 の経路表 49
6.10 N1 における RIPng と OSPFD のトラフィック量 50

表 目 次

2.1 IPv4 と IPv6 において実装されている IGP 9

3.1 OSPFv2 における LSA の役割 16
3.2 OSPFv3 における LSA の役割 16

6.1 割り当てた IPv6 グローバルアドレス 45
6.2 割り当てた IPv6 リンクローカルアドレス 45

第1章　序論

1.1　はじめに

インターネットでは、通信に利用する経路の制御を中継ノードが行なっている。しかし、インターネットが大規模広域ネットワークとして成長し、構成が複雑になるにつれて、ルータの負担は大きくなっている。経路制御に問題があると、通信自体を行なう事が不可能になる。また、経路制御機構での対応でのみ実現する要求も存在する。これらの事から、複雑な構成の大規模ネットワークにおいて快適な通信を行なうために、優れた経路制御機構が必要である。

一方、インターネット基盤技術研究において、インターネットプロトコルの問題点が重要視されている。インターネットプロトコルで利用されるIPアドレスは、ホストを全世界で一意に識別する役割を果たしている。近い将来、このIPアドレスが不足する事が予想されている。

これを解決するために、IPv6と呼ばれる新しいインターネットプロトコルが開発され、世界各地で実験運用が行われている。IPv6では、膨大な数のホストがインターネットに接続した場合にも、通信環境を提供する事が可能である。インターネットの核となる部分は、IPv6に移行することが予想される。

IPv6ネットワークにおいて、経路制御機構の問題点が存在する。問題点は、経路制御方式に関するもの、経路制御を行なう環境に関するもの、の二つに分けることができる。

経路制御方式に関する問題点は、現在利用している経路制御プロトコルに起因する。これは、経路制御トラフィックが多い、経路収束が遅いなどの問題となる特徴を持つ。経路制御トラフィックの多さや経路収束の遅さは、通信環境の悪化に直接影響する。これを改善するためには、より問題点の少ない経路制御プロトコルを利用する事が重要である。

経路制御を行なう環境に関する問題点は、ネットワーク管理者の意向を正確に経路制御に反映する事ができない、経路制御方式を選択できないなど、ネットワークを管理する上で詳細な設定ができない事を示している。ネットワーク管理者は、快適な通信を実現するために管理するネットワークを詳細に設定することが必要である。

これらの問題点の解決手法として、本研究ではIPv6をサポートしたOSPF経路制御プロトコルを実装した。OSPFプロトコルは、経路制御方式における現状の問題点の解決および経路制御における詳細な設定を実現する。この実装の評価を

行ない証明した。

本研究によって、IPv6 ネットワークにおいて OSPF による経路制御が実現可能になった。また、OSPF による経路制御によって通信環境が改善される事を評価で示した。

IPv6 ネットワークは、アドレスの制約が無いために、従来の IPv4 ネットワークより拡大を容易に行なえる。このため、IPv6 への移行はインターネットがさらに成長し、拡大するために必要である。

これまで、経路制御方式に起因する通信環境の問題点、および詳細な経路制御の設定が不可能であることが移行の妨げとなる大きな問題点であった。

本研究により、これらの問題点が解決された。このことから、IPv6 への移行が促進され、インターネットの拡大につながる事が予想される。

1.2 本論文の構成

本論文では、2 章において本研究の現状および問題意識を述べる。3 章において OSPF for IPv6[4] の設計および既存の OSPF プロトコルである OSPF Version 2[11] からの変更を述べる。4 章において、本研究で実装した OSPFD の設計を述べ、5 章においてその実装手法を述べる。6 章において実装およびプロトコルの評価を行なった後、本論文を 7 章にまとめる。

第2章　経路制御の現状とその問題点

2.1　現在のインターネットにおける問題点とその解決手法

2.1.1　IPv4の問題点

インターネットは急激な成長を続けている。インターネットに接続されているホスト数は現在も急激な増加傾向を示している。

既存のインターネットプロトコルであるIPv4[14]は、ホストを一意に識別するためにIPアドレスと呼ばれる番号を利用するが、このアドレスの不足が問題となる事が予想されている。これをアドレスの枯渇と呼ぶ。

2.1.2　CIDRによる解決手法

現在のインターネットにおける解決手法は、CIDR(Classless Inter Domain Routing,[7])と呼ばれる技術を用いるものである。

以前はネットワークアドレスからクラスが決められ、そのクラスを元にアドレスが割り当てられていた。この割り当て手法では、割り当てられたアドレスの利用効率が悪いという問題点があった。

CIDRは、クラスという制限を取り払ったアドレス割り当てを行なう技術である。これによってアドレスの利用効率を上げ、アドレス不足を補おうとするものである。

しかし、CIDRは根本的な問題解決とはならない。アドレスのリナンバリングが困難である事から、CIDR技術以前に割り当てたアドレスの利用効率は変わらず悪い。

Christian Huitema[8]によれば、CIDR技術を用いても、2005年から2015年の間にアドレスが枯渇する。

2.1.3　NATによる解決手法

アドレス枯渇に対する局所的な解決法として、NAT(Network Address Translation,[6])が挙げられる。

NATは、アドレス変換を行なうあるードメインを決め、そのドメインからインターネットに接続するルータにおいて、アドレスの変換を行なう技術である。

アドレスはドメイン内において一意であれば良い。ドメイン外のホストと通信する場合は、ドメイン内外の接点であるルータにおいてアドレス変換を行ない、ドメイン外には、正規に割り当てられていないドメイン内のアドレスは使用されない。

しかし、NATは本節初めに前述したように局所的な解決法である。インターネットでは、マルチホームサイトと呼ばれるような、二箇所でインターネットに接続するサイトが多い。

NATはその技術特性から、マルチホームであるトポロジを利用した、最短経路を利用するような効率的な通信を行なえない。

2.1.4 IPv6による解決手法

アドレス枯渇に対する根本的解決は、情報転送を実現するインターネットプロトコルの変更無くしては不可能である。全世界で利用されているプロトコルの仕様を変更する事は困難である事から、IPv4においてアドレス枯渇に対する根本的解決は実現できない。

IPv4の問題点を解決するために、新しいインターネットプロトコルであるIPv6が開発された。IPv6では、IPv4の 2^{96} 倍のアドレス空間を持っている。このアドレス空間の広大さから、アドレス枯渇問題は解決される。

IPv6への移行はすでに進められている。6Bone[12]と呼ばれるIPv6の実験ネットワークが世界規模で運営されている。これはIPv4ネットワークを利用したトンネルで接続された仮想ネットワークとして構築されてきたが、近年IPv6の国際専用線なども施設され、拡大を続けている。

2.2 IPv6ネットワークにおける問題点

IPv6ネットワークが拡大しているにも関わらず、次世代インターネットの経路制御に関する研究の成果は公開されていない。IPv6の仕様の大枠が定められた現段階において、次世代インターネットにおける経路制御の研究、改善が成され、公開がされていない事は、インターネットの発展に関わる大きな問題である。

IPv6ネットワークにおける経路制御は、IPv4ネットワークにおける経路制御と同様に自律システム(Autonomous System、以下AS)を単位として、AS間での経路制御と、AS内での経路制御に分けられている。図2.1にAS間経路制御の概念を、図2.2にAS内経路制御の概念を述べる。経路制御ドメインをこの二レベルに分ける事によって、全世界でのスケーラブルな経路制御を実現している。

AS間での経路制御に使用されるプロトコルをEGP(Exterior Gateway Protocol)、AS内での経路制御に使用されるプロトコルをIGP(Interior Gateway Protocol)と

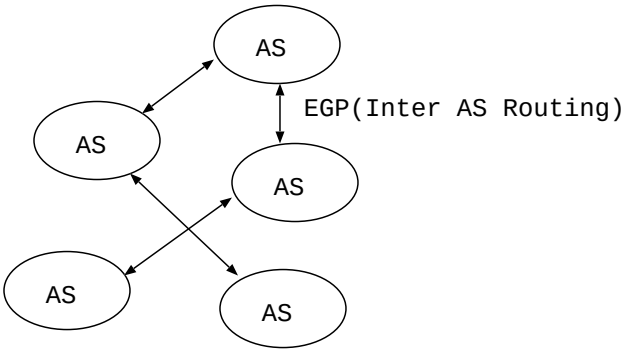

図 2.1: AS 間経路制御

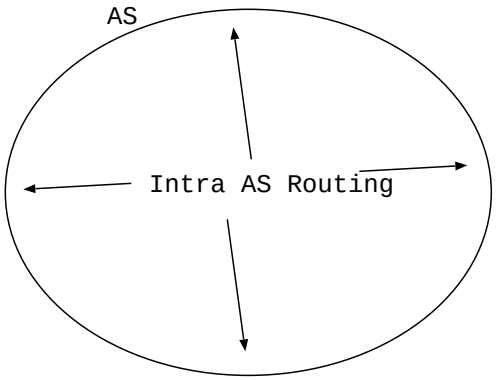

図 2.2: AS 内経路制御

呼ぶ。

現在 IPv6 ネットワークで利用可能な経路制御プロトコルは、他のプロトコルの実装の欠如から RIPng[10] のみとなっている。これは、IPv6 をサポートした RIP[5] であり、RIP における問題点を同様に持つ。

経路制御プロトコルの特性として重要視するべき事項を以下に述べる。

- プロトコルの汎用性
- 経路の収束速度
- 経路制御トラフィックの量
- ネットワーク管理者の意向を正確に反映できる機構の有無

このそれぞれに関して、IPv6 ネットワークにおける経路制御には多くの問題点が存在する。それぞれについて述べる。

2.2.1 プロトコルの汎用性

RIPng にはプロトコル設計から、「15 ホップを越えるような経路の到達性を検知できない」という問題点がある。ここから、経路制御プロトコルに最も重要だと考えられる「経路制御対象のネットワークに条件がない」という点が達成できない。

しかしながら、現在の IPv6 ネットワークは AS 内において 15 ホップを越えるような経路を持つまでに成長した。ここでは、RIPng を多段階層に分けて動作させるという解決法を取っている。

隣り合ったネットワークに隣り合ったアドレスを割り当てた場合、これらのネットワークへの経路をまとめて一つの経路として扱う事ができる。これを経路の集約と呼ぶ。

図 2.3 において、`Site B`,`Site C` は`Site A` の下流サイトであり、`Site A` からアドレスを割り当てられている。`Site A` が図 2.3 中に示すようにアドレスを割り当てたならば、`Site A` が`External` に広告する経路は、`3ffe:501:1000::/40` の一つに集約される。この例では、アドレスプレフィクスを 48 から 40 に集約して広告している。

多段階層に分けた RIPng の使用では、この経路の集約を利用している。

図 2.4 では、アドレスプレフィクスの長さ 48 のレベルで三つの RIPng システムが動作している。`RIP /48` のシステムは上位にアドレスプレフィクス長 40 で集約した経路を広告する。その上位、`RIP /40` のシステムではアドレスプレフィクスの長さ 40 のレベルで一つの RIPng システムが動作している。下位のアドレスプレフィクス長 40 で広告された経路を対象に RIPng を動作させている。

このモデルでは、集約を意識したアドレス割り当てを行なっている IPv6 ネットワークの特性を利用している。IPv6 ネットワークでは、木構造を形成するように

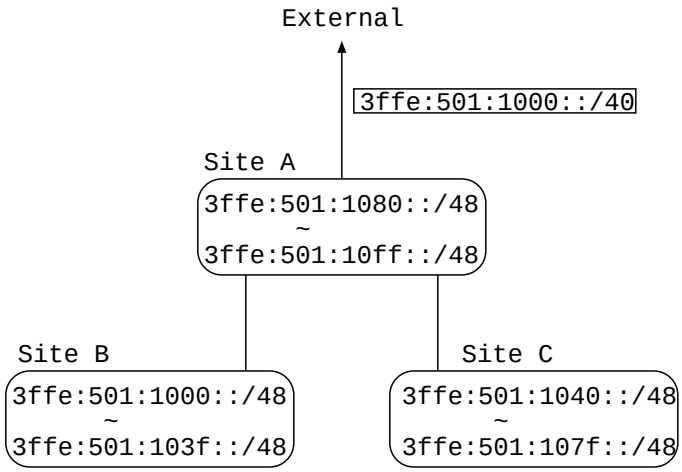

図 2.3: 経路集約の例

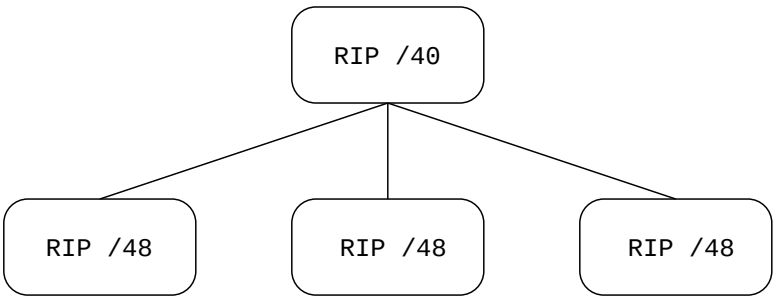

図 2.4: RIPng の多段階層

アドレスを割り当てる。この事によって経路の集約が容易になるが、現在のインターネットのトポロジはシンプルな木構造では表現できない。複雑なマルチホームを行なうネットワークトポロジにおいては、経路集約が行えず、このようなRIPngの多段階層が困難な場合が考えられる。

また、現在のこのモデルでは、一つのRIPngシステム内には15ホップを越えるような経路は存在していない。15ホップを越えたネットワークを二つのRIPngシステムに編成仕直さなくてはならないという条件は、巨大なネットワークを持つ組織にとって重大な問題である。

2.2.2 経路の収束速度

RIPでは、現在までにRIPに付け加えられたすべての機能を実装したものでも、非到達性の検知に最大120秒を要する。これは、経路の収束が迅速でない事を意味している。

RIPは、30秒毎に経路の広告を行う。この経路広告が伝播されたかどうかをRIPは確認しない。非到達性の検知に要する時間を短縮した場合、経路広告が偶然相手に到達しなかったために誤って非到達性を検知する可能性が増える。このため、RIPは経験値から、非到達性の検知を認識するのに120秒を要するプロトコルとして設計された。この場合、到達可能であるのに、偶然四つの経路広告が連続して落ちるという状況にならない限り、誤って非到達性の検知が行なわれる事はない。

これに対して、OSPFは非到達性の検知に40秒を要する。ここから、RIPはOSPFに比べ経路収束が遅いと言える。

非到達性の検知が遅い場合、到達できない終点への通信トラフィックが流れる可能性がある。これは、ネットワーク資源の有効活用ができない事を意味している。

2.2.3 経路制御トラフィックの量

多量な経路制御トラフィックを発生させるプロトコルを利用する事は、ネットワーク資源を無駄に使用する事を意味している。経路制御プロトコルが使用する帯域は、最少限にとどめられるべきである。

IPv6において唯一利用可能なIGPであるRIPngは、30秒毎に経路広告を行う。この経路広告には、広告を行うルータが持つ全経路が含まれる。それに対して、安定したネットワークでのOSPF経路制御では、30分に一度行なわれる経路の広告および10秒に一度のHelloメッセージのみである。このことから、RIPngはOSPFに比べ経路制御トラフィックが多いと言える。

ネットワーク資源を有効利用するという観点から、RIPプロトコルは経路制御トラフィックが多いという重要な問題点を持つ。これは、経路数の多い大規模なネットワークの経路を制御する場合に顕著にあらわれる。

2.2.4 ネットワーク管理者の意向を正確に反映できる機構の有無

ネットワーク管理者の意向を正確に反映するという点において、現在のIPv6経路制御環境には以下の問題点が挙げられる。

1. IGP選択肢の欠如

   ASまたはその一部のネットワークの運用ポリシを実現するためには、様々な経路制御プロトコルの選択肢を提供する環境が必要である。

   しかしながら、IPv6ネットワークにおいては、IGPの選択肢が無い。IPv4ネットワークにおけるIGP選択肢、IPv6ネットワークにおけるIGP選択肢を表2.1に比較する。

   表 2.1: IPv4とIPv6において実装されているIGP

IPv4	RIP,OSPF,(E)IGRP[9],IS-IS[3][13]
IPv6	RIP

   このように、IGPをRIPに限定するIPv6の経路制御環境では、ネットワーク管理者が運用ポリシを考慮し、IGPに利用する経路制御プロトコルを選択する事ができない。

2. RIPのネットワーク管理者の意向を反映する機構の欠如

   複雑な構成を成すネットワークにおいて、管理者の意向を正確に経路制御に反映する事が重要である。

   現在のIPv4ネットワークにおいて、回線の太さを考慮した経路決定や、運用のポリシなど、管理者の意向を経路制御に反映する事は広く行われ、大きな意味を持っている。これはOSPFプロトコルの機能を利用している。ネットワーク管理者は輻輳したリンクを調べ、OSPFのコスト設定を変更させる事によって、トラフィックを別経路へ回避させ、輻輳したリンクに流れるトラフィックを減少できる。この様にして自組織内の通信環境を改善できる。

   RIPには、管理者の意向を経路制御に反映するための機構が欠如している。

   これらの事は、IPv6経路制御環境の欠点であり、IPv4からIPv6への移行時には特に問題となる。

2.3 本研究における解決案

2.3.1 IPv6 経路制御環境の問題点のまとめ

2.2 節において、経路制御環境の重要な点およびそのそれぞれに関連した問題点を述べた。以下にその問題点をまとめる。

- RIP の欠点が経路制御に条件を与えている。
- RIP において経路の収束が遅い。
- RIP によって使用される経路制御トラフィックが多い。
- RIP において管理者の意向を経路制御に反映する機構が無い。
- RIP 以外に IGP として利用できるプロトコルが無い。

このように、問題点は主に RIP の特性から発生している。IPv6 経路制御環境において、RIP の問題点の解決が必要である。

2.3.2 他の経路制御プロトコルの実装

IPv4 では、RIP、OSPF、IS-IS、(E)IGRP 等の IGP が実装されている。IPv6 ネットワークにおいても、現在の IPv4 ネットワークにおける環境のように、さまざまな経路制御プロトコルを利用する必要がある。ここから、現在実装されていない他の経路制御プロトコルを実装する事が重要である。

2.3.3 OSPF の利用による RIP の問題点の解決

本章で述べた RIP の問題点は、IPv4 ネットワークにおいては OSPF プロトコルを経路制御プロトコルとして利用する事で解決されている。

本研究では、IPv6 ネットワークにおいても OSPF を利用する事でこれを解決する。そのため、実装する経路制御プロトコルには OSPF を取り上げた。

第3章　IPv6をサポートしたOSPF経路制御プロトコルの設計

本章では、Open Shortest Path First 経路制御プロトコル (以下 OSPF プロトコル) について述べ、さらに OSPF プロトコルの必要性を述べる。まず、OSPF の概要を述べた後、IPv6 をサポートした OSPF の変更点とその必要性を述べる。次に IPv6 をサポートした OSPF の構造と動作を述べ、最後に OSPF プロトコルの必要性を述べる。

3.1　OSPFの概要

OSPF はリンク・ステート型の経路制御プロトコルである。OSPF による経路制御では、経路制御を行なうネットワークを、Area と呼ばれる複数の経路制御ドメインに分ける事が可能である。

OSPF では、ルータとネットワークを接続点の意味を持つ「リンク」とし、そのリンク情報の広告からネットワークトポロジを計算し、経路表を計算する。リンク情報の広告は LSA(Link State Advertisement) と呼ばれる。LSA には、LSA の種類や新規性を記述するために LSA ヘッダと呼ばれる固定長、固定フォーマットのヘッダがつけられる。LSA 内の LSA ヘッダを除いた部分を LSA ボディと呼ぶ。

Area 内のルータは LSA のデータベースを同期させ、個々のルータが独自に SPF (Shortest Path First) アルゴリズムを用いて経路表を計算する。この際にネットワークトポロジを計算するので、ループを構成する様な複雑なネットワークにおいても、経路制御ループを発生させる危険が無い。経路表の計算には、コストと呼ばれる値が用いられる。経路上のコストの総和が少いものが、経路表に記述される。

隣接ルータからの応答は OSPF 内部の Hello プロトコルにより監視され、応答の無くなったルータは即座に経路計算から外される。

LSA は一時間の有効期限を持ち、ルータは三十分毎に自分の送出した LSA を更新する。この事から、安定したネットワークでは経路制御情報が消費するネットワーク帯域が、RIP を始めとする従来のディスタンス・ベクタ型経路制御プロトコルに比べて少ない。

LSA データベースの同期に使用する経路制御トラフィックの冗長性を最小にとどめるため、各マルチアクセスネットワークにおいて代表ルータ (Designated Router、

以下 DR) およびバックアップ代表ルータ (Backup Designated Router、以下 BDR) を選出する。ネットワークの各ルータは他の全ルータとデータベースの同期を取るのではなく、全てのルータが DR に選出されたルータとデータベースの同期を取る事で、最小限のネットワークトラフィックを用いて実現する。

3.2 IPv6 をサポートした OSPF の設計

IETF の OSPF ワーキンググループから、IPv6 をサポートした OSPF の設計が提案されている [4]。この提案では、メッセージや LSA が従来の OSPF と互換ではないため、OSPF のバージョンが 2 から 3 に上がっている。基本アルゴリズムは変更されていない。

本節では、IPv6 をサポートした OSPF バージョン 3(以下 OSPFv3) について、OSPF バージョン 2(以下 OSPFv2) からの変更点およびその必要性を述べる。

3.2.1 サブネット毎の処理からリンク毎の処理へ

IPv4 では、一つの物理ネットワークには単数のネットワークアドレスが割当てられている事が前提であった。この物理ネットワークとネットワークアドレスを合わせてサブネットと呼ぶ。これに対して、IPv6 では一つの物理ネットワークをリンクと呼び、複数のネットワークアドレスが割当てられる事を明示的に許している。

OSPFv2 はサブネット毎の処理を行なっていた。OSPFv3 では IPv6 の概念の変更にともない、サブネット毎の処理をやめ、リンク毎つまり物理ネットワーク毎の処理を行なう。

OSPFv2 において、複数のネットワークアドレスが割当てられたネットワークでは、正確な経路制御ができず、いくつかの深刻な問題を引き起す。

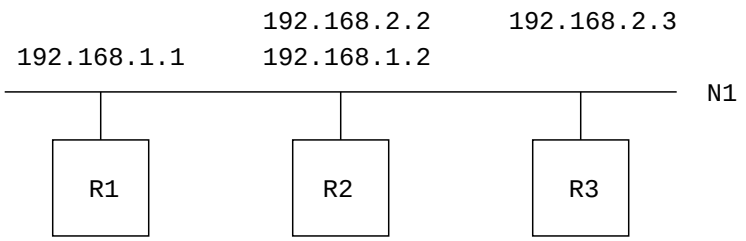

図 3.1: OSPFv2 が正確に経路制御できない例

図 3.1 を例に説明する。ネットワーク N1 にはルータ R1、R2、R3 が接続され、OSPF バージョン 2 を動作させている。各ルータはネットワーク N1 へのインターフェースを一つ持ち、図の上に示されるアドレスが割当てられている。ルータ R2 には

二つのアドレスが割当てられており、ネットワークN1 全体では 192.168.1.0/24 と 192.168.2.0/24 の二つのネットワークアドレスが割当てられている。ルータR1 とR3 が送信する OSPF メッセージは、受信されたパケットの始点アドレスが自分の接続するネットワーク上に無い事から、お互いに破棄される [11, section 8.2,(1)]。OSPF メッセージが正常にやりとりされないので、R1 またはR3 が OSPF 経路制御に参加できない。

R2 のインターフェースに二つのアドレスが割り当てられていた場合、どちらを OSPF のインターフェースアドレスとして使用するかは実装依存である。R2 が 192.168.1.2 をインターフェースのアドレスとして使用した場合、R3 は OSPF 経路制御に含まれずにトポロジ計算が行なわれる。この結果、Area 内のすべてのルータはこの部分のトポロジを図 3.2 の様に計算する。

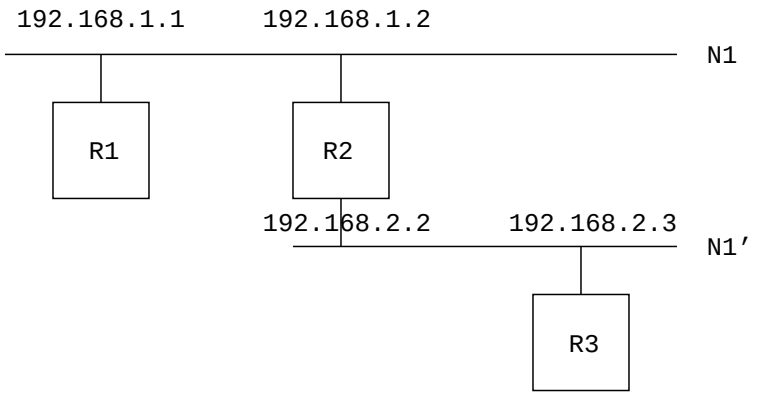

図 3.2: トポロジ計算結果

ここで、192.168.2.0/24 のネットワークはネットワークN1' として、N1 とは別のネットワークとして計算される。これは、R2 が Stub ネットワークとして広告する。このため、R3 以遠のネットワークが存在した場合、経路が計算されず到達性がなくなる。

また、R1 からR3 への IP パケットは、一番初めにR2 に送信され、ICMP redirect が使用された後、次回からはR1 からR3 に直接送信される。

このようなネットワークトポロジにおいて、OSPFv2 では経路制御が正確ではないため、無駄や経路計算の不可能が起きる。

OSPFv3 ではリンク毎のトポロジ計算を行なった後、ネットワークアドレスの計算がおこるので、この例においても正確なトポロジ計算の後、ネットワークアドレス 192.168.1.0/24、192.168.2.0/24 両方がネットワークN1 に割当てられている事を間違い無く計算できる。

3.2.2 トポロジの広告とアドレスの広告の分離

OSPFv2 は、トポロジの広告とアドレスの広告が区別されていないという欠点を持っていた。トポロジの広告は Router-LSA および Network-LSA で行なわれ、アドレスの広告は前述の二つを含む全ての LSA で行なわれていた。OSPFv3 では、トポロジ情報の広告とアドレス情報の広告が分離された。具体的には、LSA ヘッダおよび OSPF メッセージヘッダから、アドレスの意味が取り消されている。OSPFv2 における OSPF メッセージを図 3.3 に、LSA ヘッダを図 3.5 に、Router-LSA と Network-LSA をそれぞれ図 3.7、図 3.9 に示す。OSPFv3 における OSPF メッセージを図 3.4 に、LSA ヘッダを図 3.6 に、Router-LSA と Network-LSA をそれぞれ図 3.8、図 3.10 に示す。

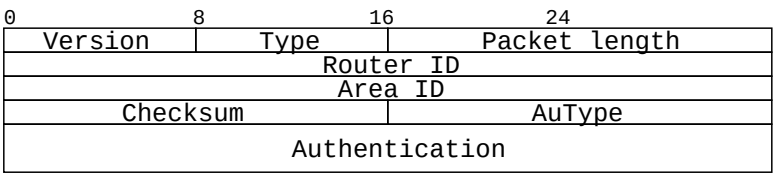

図 3.3: OSPFv2:OSPF メッセージヘッダフォーマット

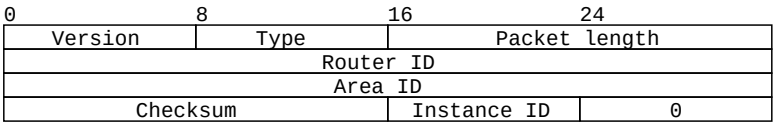

図 3.4: OSPFv3:OSPF メッセージヘッダフォーマット

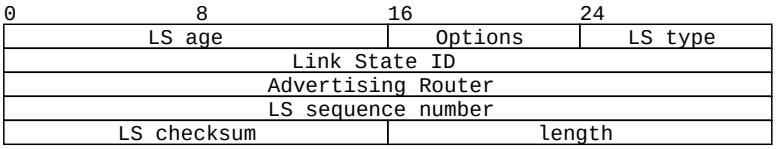

図 3.5: OSPFv2:LSA ヘッダフォーマット

OSPF では、バージョンを問わず、ネットワークのトポロジのタイプによってルータが挙動を変える。ネットワークのトポロジのタイプには、Transit ネットワークと Stub ネットワークがある。ルータが複数台接続されないネットワークは、ネットワークトポロジにおいて末端である。末端のネットワークは必要がないため、OSPFv3 においてはトポロジ計算に含まれない。このようなネットワークを Stub ネットワークと呼ぶ。ルータが複数台接続されているネットワークは、ネットワー

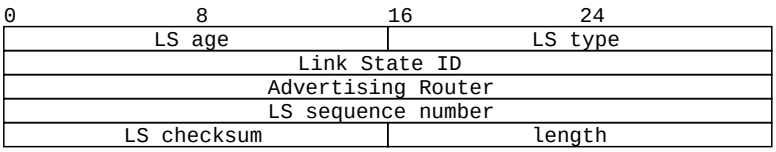

図 3.6: OSPFv3:LSA ヘッダフォーマット

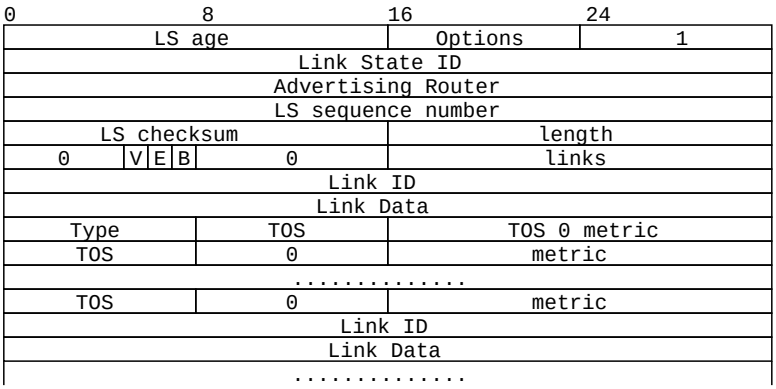

図 3.7: OSPFv2:Router-LSA フォーマット

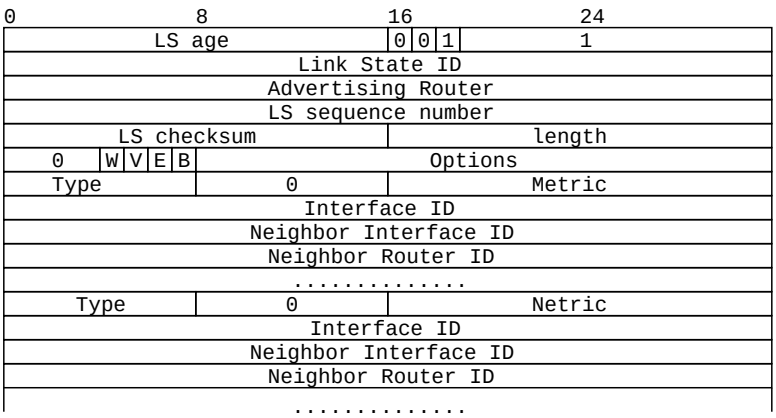

図 3.8: OSPFv3:Router-LSA フォーマット

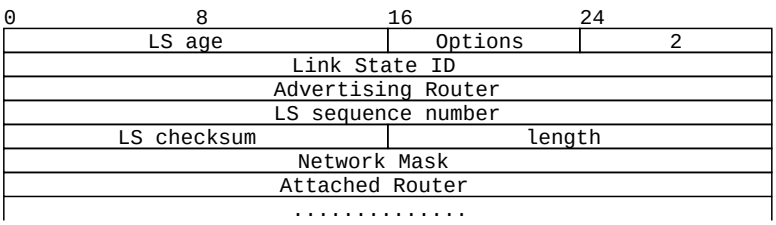

図 3.9: OSPFv2:Network-LSA フォーマット

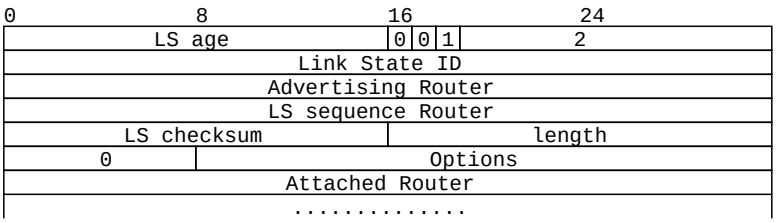

図 3.10: OSPFv3:Network-LSA フォーマット

表 3.1: OSPFv2 における LSA の役割

トポロジ広告	アドレス広告
Router-LSA	Router-LSA
Network-LSA	Network-LSA
	Type 3 Summary-LSA
	Type 4 Summary-LSA
	AS-External-LSA

クトポロジにおいてルータ間を接続する重要なネットワークであり、トポロジ計算に含まれる必要がある。このようなネットワークを Transit ネットワークと呼ぶ。

OSPFv2 における全 LSA を役割毎に表 3.1 に示す。OSPFv3 における全 LSA を役割毎に表 3.2 に示す。

OSPFv2 では、表 3.1 に示されるように、Router-LSA および Network-LSA がアドレスの広告も行なっていた。OSPFv2 では Router-LSA を使用して行なっていた Stub ネットワークの広告、および Network-LSA を使用して行なっていた Transit ネットワークの広告は、OSPFv3 では Intra-Area-Prefix-LSA を使用する。

OSPFv2 の Transit ネットワークの広告において、Router-LSA の Link-State-ID フィールドを利用していたインターフェースアドレスの広告は、OSPFv3 においては Link-LSA を利用する。このアドレスは経路表計算時に Next hop アドレスとして使用される。

表 3.2: OSPFv3 における LSA の役割

トポロジ広告	アドレス広告
Router-LSA	Link-LSA
Network-LSA	Intra-Area-Prefix-LSA
	Inter-Area-Prefix-LSA
	Inter-Area-Router-LSA
	AS-External-LSA

OSPFv2 における Type 3 Summary-LSA、Type 4 Summary-LSA は、それぞれ Inter-Area-Prefix-LSA、Inter-Area-Router-LSA と名前を変えている。実際の役割に変更は無い。

この様に、トポロジの広告とアドレスの広告を分けた OSPFv3 プロトコルは、IPv4 に特化した経路制御機構を提供する OSPFv2 と異なり、経路制御の対象となるネットワークプロトコルに独立して動作するようになった。新しいネットワークプロトコルに対応する場合は、新しい LSA を追加すればよい。ネットワークプロトコルが変わってもトポロジの計算方法には変化はないので、OSPFv3 ではどの様なネットワークプロトコルを使用していてもトポロジの計算に支障がない。

IPv4 ではアドレスの長さが 32bit であったのに対し、IPv6 では 128bit に拡張された。このため、トポロジの広告とアドレスの広告が区別されたのは重要な意味を持つ。

OSPF ではバージョン番号を問わず、各ルータが接続されたネットワークの DR のインターフェース識別子を広告内で参照する場合が多い。このため、トポロジを表現する際にアドレスを利用している OSPFv2 の方法を IPv6 に利用した場合、アドレスが四倍になった事から生じる無駄が多い。また、同じ理由で Netmask は削除され、ネットワークアドレスの長さは PrefixLength と呼ばれる上位 bit からの整数で表わされるように変更された。

OSPFv3 では、アドレスとルータ識別子、インターフェース識別子が分けられている。ルータ識別子は OSPF 経路制御ドメイン内で一意、インターフェース識別子はルータ内で一意の任意の数字とし、長さは 32bit のままである。

これは IP アドレスをルータ識別子やインターフェース識別子として利用していた OSPF バージョン 2 と異なる部分である。

3.2.3 flooding スコープの追加

LSA が伝播する事を flood と呼び、LSA が伝播される範囲の事を flooding スコープと呼ぶ。

OSPFv2 では、flooding スコープは LSA の種類と Area の設定によって決められていた。OSPFv3 では LSA type フィールドの上位 3bit を利用して、未知の LSA の操作の方法の指定と flooding スコープの指定を行なう。

これによって、新しい LSA が定義された場合の拡張性が増やされた。

3.2.4 その他

他にも、前述の変更を実現するために OSPF ヘッダと LSA ヘッダに変更が加えられている。また、Hello パケット内や LSA 内に現れる Options フィールドが 24bit に拡張されている。また、Neighbor の識別には Router-ID を利用するなど、細か

い変更が加えられている。

3.3 OSPFv3の動作概要

本節では、OSPFv3の動作概要を説明する。OSPFv3における動作概要は、OSPFv2におけるものと変わらない。

OSPFの動作は、以下のステップに分かれる。

3.3.1 到達性の監視と非到達性の検知

OSPFは、到達性の監視と非到達性の検知を実現するために、内部にHelloプロトコルを持つ。隣接ルータからの応答が一時期の間に受信され、双方向の通信が保証された事を到達性が持続していると認識する。

Helloプロトコルは、それぞれのルータとの間に双方向の通信が可能である事を確認する。ルータはHelloInterval変数毎にOSPF Helloメッセージを送信する。Helloメッセージには、過去RouterDeadInterval変数秒以内にHelloメッセージを受信した他のルータを記述する。相手のHelloメッセージ内に自分が記述されていれば、そのルータと双方向の通信が保証されたと判断する。

双方向の通信が保証されていた隣接ルータから、RouterDeadInterval変数秒以内にHelloメッセージを受信しなかった場合、双方向の通信が切断されたと判断する。この事は、非到達性の検知を意味する。

RouterDeadIntervalの変数はHelloIntervalの変数の4倍が推奨されている。この状態では、Helloメッセージが続けて4回落ちない限り、誤って双方向の通信が不可能であると判断する事はない。HelloIntervalは10秒が推奨されている。

Helloプロトコルによって、動的に他のOSPFルータを検出する事ができる。検出されたルータはNeighborとして表される。

3.3.2 ネットワークトポロジの広告

ネットワークトポロジの広告は、Router-LSA, Network-LSAによって行われる。各OSPFルータは、Router-LSAを広告する。Multi Accessネットワークでは、ネットワーク毎にDRが選出され、DRがそのネットワークのNetwork-LSAを広告する。DRはHello Protocolによって選出される。

Router-LSA

各OSPFルータはRouter-LSAを広告する。Router-LSA内にはTransitネットワークに接続されたインターフェースを記述する。ルータのインターフェースと

Transit ネットワークの接続は、ルータのインターフェース識別子と Transit ネットワークの DR のインターフェース識別子の組で表現される。

Network-LSA

Transit ネットワークにおいて DR に選出されたルータは、Network-LSA を広告する。Network-LSA には、ネットワークに接続されたルータが記述される。

3.3.3 ネットワークアドレスの広告

ネットワークアドレスの広告は、Link-LSA、Intra-Area-Prefix-LSA、Inter-Area-Router-LSA、Inter-Area-Prefix-LSA、AS-external-LSA によって行なわれる。

ネットワークアドレスは PrefixLength、PrefixOptions、Address Prefix の組によって表現される。

Link-LSA

各ルータは Link-LSA を広告する。Link-LSA は広告されたネットワーク以外へは flood されない。Link-LSA は主に二つの役割を持つ。

- 自分に対する next hop アドレスを他のルータに知らせる
- ネットワークアドレスを DR に知らせる

DR がネットワークに割当てられたネットワークアドレスを全て知っている保証は無く、DR が広告する Intra-Area-Prefix-LSA に記述されないネットワークアドレスは、他の OSPF ルータから到達可能だと判断されない。Link-LSA は、全てのルータのこのネットワークへのインターフェースに割当てられたネットワークアドレスを、DR に知らせる役割を持つ。

Intra-Area-Prefix-LSA

Intra-Area-Prefix-LSA は Area 内にネットワークアドレスを広告する。

ネットワークの DR に選出されたルータ、スタブネットワークを持つルータ、およびホストルートを持つルータやアドレスを割当てられたポイントトゥポイントネットワークに接続されたルータが広告する。

この LSA には参照する LSA の Type、Link State ID、Advertising Router が記述され、Network-LSA や Router-LSA に関連づけられる。

この LSA によって広告されたネットワークアドレスは経路表の計算に使用される。

Inter-Area-Router-LSA

OSPF バージョン 2 における type 4 summary-LSA と同じ意味を持つ。Area の境界ルータによって、別 Area の境界ルータのアドレスが広告される。

別 Area への経路の next hop を計算する場合に使用される。

Inter-Area-Prefix-LSA

OSPF バージョン 2 における type 3 summary-LSA と同じ意味を持つ。Area の境界ルータによって、AS 内の別 Area のネットワークアドレスが広告される。

別 Area への経路を計算する場合に使用される。

AS-External-LSA

AS 境界ルータによって、AS 外のネットワークアドレスが広告される。

AS 外への経路を計算する場合に使用される。

3.3.4 広告の伝播と同期

広告の伝播と同期のために使用する経路制御トラフィックを最小にするために、各マルチアクセスネットワーク内において、DR と BDR を選出すると前述した。

ネットワークにおけるトポロジの広告は DR が行うので、各ルータ内において DR の変更はトポロジの変更として判断される。このため、Hello プロトコル内の DR 選出アルゴリズムは、DR の変更が頻繁におきないように設計されている。

DR に選出されたルータが停止すると、広告の同期のためにそのネットワークにおいて DR の選出からやり直される。この場合、広告の同期が達成された後経路が計算されるので、そのネットワークを含む Area 内の経路収束には時間を要する。この時間を最小に留めるため、各マルチアクセスネットワークでは DR の他に BDR を選出する。

広告の伝播と同期は各ルータと DR 間のみでなく、BDR との間においてもそれぞれ行なわれる。広告の伝播と同期を行なう各ルータ間の接続を Adjacency と呼ぶ。Adjacency は DR と各ルータ、BDR と各ルータ間に確立される (図 3.11)。

DR が停止した場合、BDR が DR に選出される。BDR と各ルータの間には Adjacency が既に確立されているので、新しく DR と Adjacency を確立するために要する時間を最小に留める事ができる。

初期状態の広告の伝播と同期は、以下の手順で行なわれる。

1. 自分の持つ広告をお互いに教える。

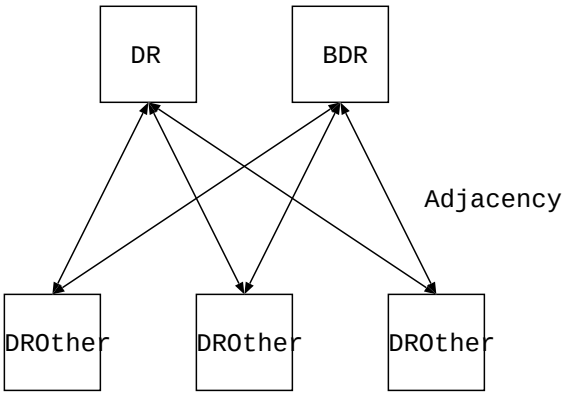

図 3.11: マルチアクセスネットワークにおける Adjacency の確立

Database Description メッセージを使用し、自分の持つ LSA データベースを全て記述する。

2. 相手が持つ広告のなかで、自分が持っていないものを要求する。

   Link State Request メッセージを使用し、Database Description メッセージによって記述された相手の LSA データベースのうち自分が持っていないものを要求する。

3. 相手から要求された広告を送信する。

   Link State Update メッセージを使用し、Link State Request メッセージによって相手から要求された LSA を送信する。

以上の手順が終了した状態を「Adjacency が確立された」と呼ぶ。Adjacency が確立された後も、広告の伝播は以下の手順で行なわれる。

1. 新しく自分が得た広告を、相手に伝播する。
2. 相手から応答を得るまで、伝播を続ける。

3.3.5 ネットワークトポロジの計算

広告されたネットワークトポロジ情報は、経路表を計算するために Shortest Path Tree を構築する際に使用される。Shortest Path Tree は計算するルータ自身を根とした、ルータとネットワークを頂点とするグラフとして表現される。

Shortest Path Tree の構築には Dijkstra のアルゴリズムを利用する。候補リストにはルートからのある終点へのパスが記載される。Shortest Path Tree は Area 毎に以下の手順で構築される。

1. Shortest Path Tree のルートを、計算するルータ自身に設定し、候補リストを

空にする。

2. Shortest Path Tree に新しく設定された頂点のすべてのリンクを調べる。リンクの到達点が、より短かいパスとして候補リストに記載されていない場合、候補リストにこの頂点へのパスを記載する。
3. 候補リストの中で一番短いパスを Shortest Path Tree の頂点に設定する。
4. 候補リストが空である場合終了する。そうでない場合、2 に戻る。

3.3.6 経路表の計算

経路表の計算は、以下の手順で実行される。

1. Area 内の経路計算
2. Area 外かつ OSPF 経路制御ドメイン内の経路計算
3. OSPF 経路制御ドメイン外の経路計算

3.4 OSPF データ構造体

OSPFv3 は、OSPFv2 と同様に、動作アルゴリズムの対象として以下のように階層化されたデータ構造を内部に持つ。OSPF データ構造体を図 3.12 に示す。

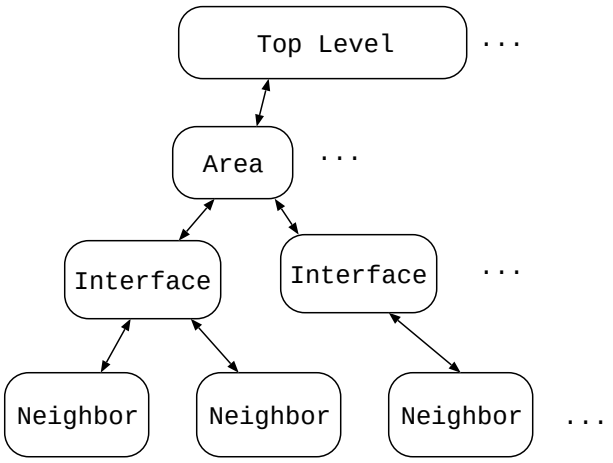

図 3.12: OSPF データ構造体

3.4.1 Top Level データ構造体

Top Level データ構造体は、OSPF ルータによって取り扱われる一つの OSPF システムを表わす。

Top Level データ構造体はこのルータが属する Area のデータ構造体を持つ。また、OSPF 外部から広告された経路情報はこの構造体内に格納される。経路制御表や Router-ID など、OSPF システム全体で一意に扱うデータはこの構造体が持つ。

3.4.2 Area データ構造体

Area データ構造体は、この Area に接続されている Interface データ構造体を持つ。また、Area 内に広告される経路情報および経路制御情報は、この構造体内に格納される。

トポロジ情報の計算および結果の格納など、Area 毎に行なわれる動作はこの構造体を対象とする。

3.4.3 Interface データ構造体

この Interface を通じて通信可能な他のルータを、Neighbor データ構造体として持つ。また後述の Hello プロトコルのように、インターフェース毎に行われる動作は Interface データ構造体を対象とする。Interface データ構造体は状態とイベントを持つ。

Interface の状態遷移図を図 3.13 に示す。

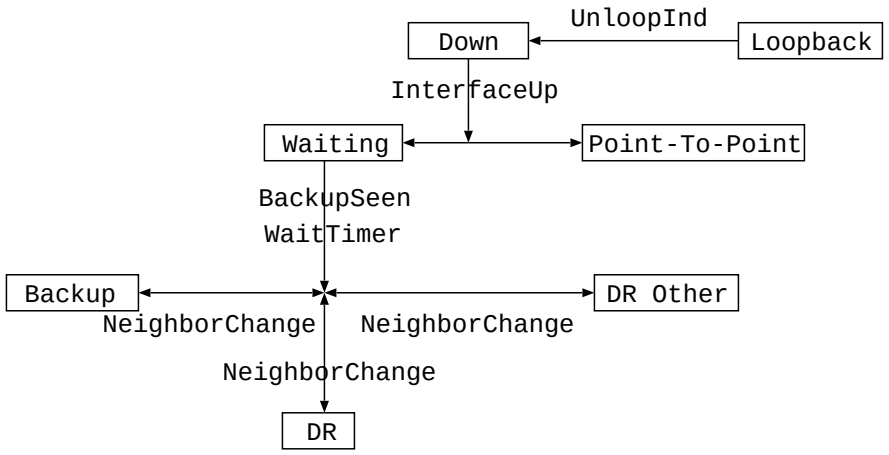

図 3.13: Interface 状態遷移図

ルータ自身がこのネットワークにおいてどのような役割を担っているかは、Interface 構造体の状態に記述される。ルータ自身がこのネットワークにおいて DR、BDR、そのどちらでも無い場合、Interface 構造体はそれぞれ DR、Backup、DR Other の状態となる。

3.4.4 Neighbor データ構造体

OSPF ルータ内において、隣接する他のルータは Neighbor データ構造体として記述される。隣接ルータは Hello プロトコルによって動的に検出される。経路情報および経路制御情報の同期は、Neighbor データ構造体を対象に行なわれる。Neighbor データ構造体は状態とイベントを持つ。

Neighbor の状態遷移図を図 3.14 に示す。

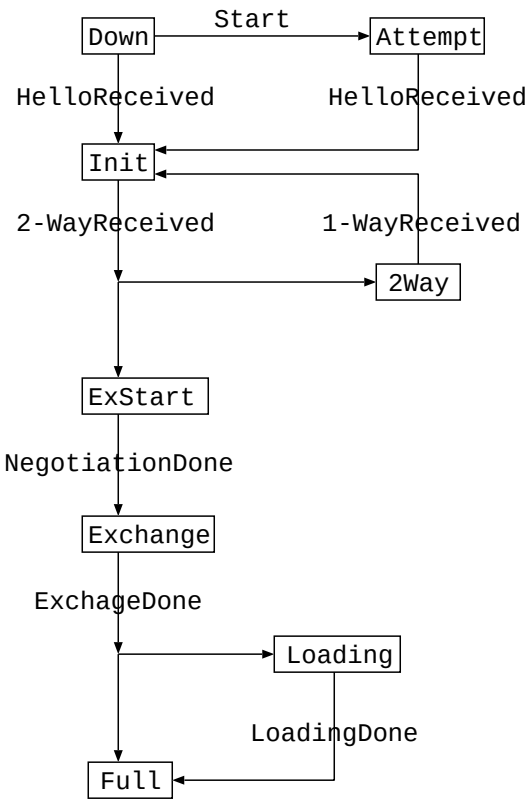

図 3.14: Neighbor 状態遷移図

Neighbor 状態遷移は二段階に分ける事ができる。

第一段階では、Hello プロトコルによって Neighbor との双方向の通信を保証する。第一段階の終わりでは、Neighbor との間に経路制御情報の同期を取る必要があるのかを検証する。同期を取る必要が無い場合、2Way 状態へ進み、第二段階へは進まない。同期を取る必要がある場合、ExStart 状態へ進み、第二段階の状態遷移を開始する。

第二段階では、Neighbor との間に経路制御情報の同期を取る。同期が完全に取れた場合、Full 状態へ進む。

3.5 OSPFプロトコルの必要性

2.2 節において、経路制御には以下の事が重要だと述べた。そのそれぞれにおいて、OSPF プロトコルが実現しているかを述べる。

- プロトコルの汎用性

  OSPF では、ポイントトゥポイント、ブロードキャストネットワーク、ノンブロードキャスト・マルチアクセスネットワーク、ポイントトゥマルチポイント等さまざまなネットワークをサポートしている。ポイントトゥポイントの場合にはアドレスの付けられていないネットワークでも経路制御が可能である。また、広域ネットワークにおける経路制御を実現するため、Area という概念を利用している。

  また、3.2.1 に述べた理由から、OSPFv2 の実装である GateD よりも汎用性が高い。この汎用性の高さは、アドレスの広告とトポロジの広告を分けている。

- 経路の収束速度

  RIP の問題点であった非到達性の検知にかかる時間は、OSPF では 40 秒である。これは、現在の IPv6 ネットワークで使用されている RIP の 120 秒と比較して迅速である。OSPF の導入によって、経路の非到達性の検知に要する時間が現状の約三分の一に短縮される。

- 経路制御トラフィックの量

  OSPF では、経路制御トラフィックを最少限に留めるために、DR の選出や LSA ヘッダのみを含むメッセージなど、様々な工夫がなされている。安定したネットワークでは、OSPF が使用する経路制御トラフィックは Hello プロトコルおよび三十分に一度の LSA の更新のみであるので、経路制御トラフィックは最少限に留められている。

- ネットワーク管理者の意向を正確に反映できる機構の有無

  OSPF では、cost 変数の設定を利用する事によって、ネットワーク管理者の意向を経路制御に反映する機構を持つ。インターフェースに設定する cost 変数は 1 から 65535 の値を取る事ができ、管理者の意向を正確に反映できる。

第4章　OSPFDの設計

4.1　設計目標

前章で述べたように、OSPF は、経路制御を行う場合に重要となる経路制御プロトコルの特性を多く持っている。

この特性を最適に利用することを設計の目標とした。このため、本研究における設計は、前章で述べたプロトコルの設計を最も忠実に再現する形で行なった。

しかし、タスクとタイマの関係から、プロトコルの設計に示されていない部分では、イベントとしては別だが構造体としては同一といった設計を行なった方が効率が良いもの等があった。これら設計の工夫を本章各節で述べる。

4.2　設計概要

Zebra[2] という経路制御ソフトウェアで、経路制御ソフトウェアに必要である、プロトコル間における経路情報の交換等、多くの機能の実装を念頭に置いた設計が成されている。簡潔で多機能であるライブラリを持ち、それぞれが一つの経路制御プロトコルを実装した経路制御デーモンの集りであるが、OSPF プロトコルの実装を持たなかった。本実装 (以下 OSPFD) は Zebra 内の経路制御デーモンとして実装した。これによって、以下を実現した。

- 重複する実装を減らす

  経路制御デーモンでは、静的な設定や他の経路制御プロトコルとの相互干渉が必要になる場合が多い。ここから、一つのルータ内に複数の経路制御デーモンを起動する可能性がある。複数の経路制御デーモン内において別々に利用される同様の機能を、一つの関数として実現することによって、ルータ内のメモリ資源の有効活用を実現することができる。

- 類似した設定インターフェースを提供し、設定を容易にする

  同一のライブラリを利用することによって、類似した設定インターフェースを提供することができる。これは、管理者の利用環境を改善に継がる。また、将来一つの設定インターフェースから、サ

ポートするすべての経路制御プロトコルの設定を行なう機能が追加される。

これらの事から、OSPF プロトコル変数の設定インターフェースやリスト機能等、OSPF を実装するために必要な基本的な機能は、Zebra のライブラリを用いた。

OSPF を実装する為に必要となる、いくつかの基本的な機能を以下に述べる [11, section 4.4]。

1. Timers

   OSPF では、多数のタイマを利用する。オペレーティングシステムのタイマ機能ではこれら多くのタイマを実現する事は不可能である。Zebra には、維持するために必要な処理の少ない優れたタイマが、ライブラリとして提供されている。これは、タスク機能の一部として実現されている。OSPFD ではこれを利用する。

2. IP multicast

   OSPF プロトコルは、無駄なトラフィックを増やさないために IP マルチキャストを利用する。これを利用するため、マルチキャストその他多くの IPv6 の仕様を実現している KAME[15] パッケージを利用する。本実装は、KAME パッケージの機能を入れたオペレーティングシステム上で行なう。

3. Variable-length subnet support

   IPv6 では、既存の IPv4 と同じように、ネットワークを任意の大きさで分割することができる。これは現在の経路制御に不可欠な機能である。KAME パッケージの機能を入れたオペレーティングシステムが持つ、同一機能を利用する。

4. IP supernetting support

   ネットワークを任意の大きさで集約する IPv6 の機能。これも、Variable-length subnet support と同様に、KAME パッケージの機能を利用する。

5. Lower-level protocol support

   イーサネットプロトコルのような、ネットワーク層の下位層のプロトコル。これは、オペレーティングシステムの機能を利用する。

6. List manipulation primitives

   OSPF は、LSA をリストとして扱う。これは、Zebra のライブラリに含まれるリスト機能を利用する。

7. Tasking support

タスクは、Zebra のライブラリに含まれるものを利用した。OSPF のタスクには、即座に実行されるものと次のタスクとしてスケジュールされるものの二通りある。次のタスクとしてスケジュールする場合は Zebra ライブラリのタスク機能を用い、即座に実行されるものは Zebra のライブラリを用いずに関数を呼び出す事で実現する。

OSPFD の構造を図 4.1 にあらわす。

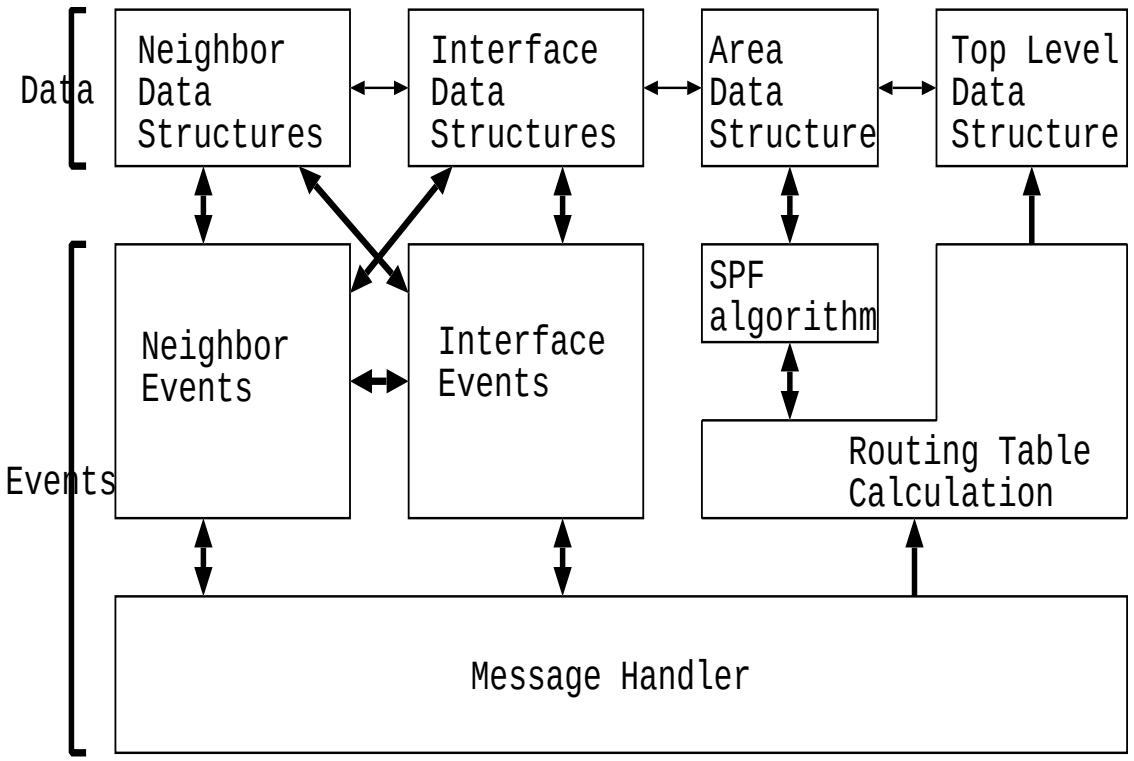

図 4.1: OSPFD の構造

OSPFD は、データ構造体とイベントから成る。データ構造体の詳細では、それぞれのデータ構造体が図 3.12 のような構造を成している。イベント群は、処理する対象となるデータ毎に分かれている。しかし、メッセージと LSA を処理するルーチンは他のさまざまなデータを同時に処理するので、OSPF の中核として一つのイベント群として設計した。

4.3 動作設計

OSPFD 内において、イベントは関数として実現される。動作対象となるデータ構造体が引数として渡される。

一つのイベントキューを持ち、さまざまなイベントが発生した順に実行される。Zebra のライブラリの設計から、タイマは発火時にこのイベントキューに含められるイベントとし、Message Handler は受信時には即座に実行されるイベント、送信時にはイベントキューから実行されるイベントとした。

OSPFD はイベント単位のマルチタスクとして設計した。このため、一つのイベント中にタイムアウト機能を必要とするような動作を行なう場合、イベント内でタイムアウト機能を実現すると、他のイベントが待たされたりタイマが狂う場合がある。このようなイベントはメッセージの受信、送信時に多く発生する。このため、タイムアウト後に実行する動作はイベントから切り離し、Message Hander イベント群として設計した。

OSPFD の動作設計を図 4.2 に示す。

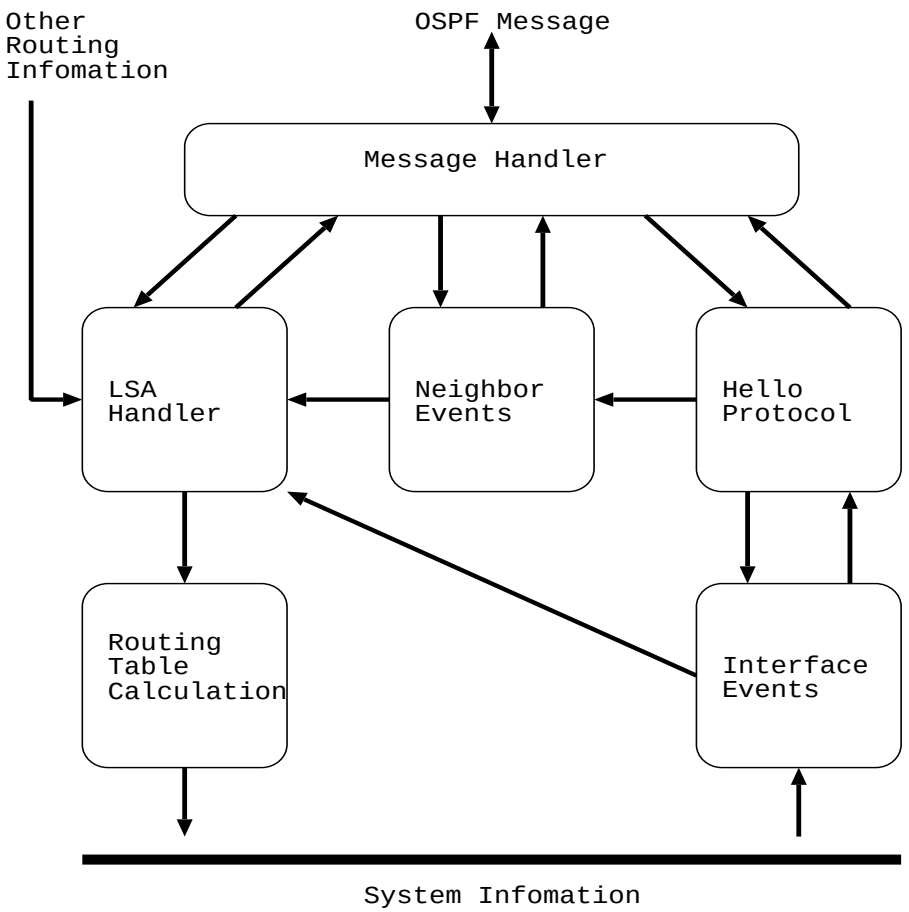

図 4.2: OSPFD の動作

4.4 Message Handler イベント群

Message Handler は受信したメッセージから適当なイベントを起こす。他のイベントからのメッセージ送信要求はイベントとして Message Handler が行なうため、Message Handler 自身もイベントになっている。Message Handler は Hello Protocol イベント、Neighbor イベント、LSA Handler イベントを呼ぶ。

メッセージは、前後の関係を重要とする。この場合、以下の機能が必要となる。

- 送られてきたメッセージの内容を一定期間記憶しておく。
- 既に送ったメッセージを再送する。

これらの機能を実現するために、メッセージを格納する構造が必要である。メッセージは送信される範囲を持つので、それぞれの機能を実現するために、送信される範囲に適したデータ構造体内に格納する。メッセージには、以下の送受信範囲のどちらかが適用される。

1. 同じインターフェースに接続された全ての Neighbor に対する送受信
2. 同じインターフェースに接続された個別の Neighbor に対する送受信

1,2 を実現するために、Message Handler はそれぞれ Interface データ構造体、Neighbor データ構造体を利用する。このため、Message Handler はデータ構造にとらわれずに動作する OSPFD の中核として設計した。

4.5 LSA Handler イベント群

LSA Handler は構造上は Message Handler 内に含まれる (図 4.1)。

経路制御情報の伝播のために用いられる LSA は、メッセージに格納されて伝播される。このため、LSA Handler は Message Handler イベントを呼ぶ。また、受信した LSA から新しい経路表を計算するために、Routing Table Calculation イベントを呼ぶ。

LSA Handler は LSA の送信および受信を行なう。LSA の送信は以下の場合に行われる。

- トポロジの変更により、新しい LSA を作成し送信する。トポロジの変更は Neighbor イベントおよび Interface イベントから通知される。
- LSA の更新時期に、新しい LSA を作成し送信する。LSA Handler 内部のタイマによって LSA の更新時期が通知される。
- 他の経路制御プロトコルからの情報を受け、新しい LSA を作成し送信する。

作成されたLSAを送信するため、LSA HandlerはMessage Handlerイベントを呼ぶ。また、新しく得たLSAを利用して経路計算を行なうため、Routing Table Calculationイベントを呼ぶ。各LSAはFlooding scopeに適合したデータ構造体内に格納され、Routing Table Calculationイベント群に渡される。

4.6 Hello Protocolイベント群

Hello Protocolは構造上はInterfaceイベントおよびNeighborイベント内に含まれる(図4.1)。

Hello ProtocolはInterfaceイベント、Message Handlerイベント、およびHello Protocol内部のタイマから呼ばれ、Interfaceイベント、Neighborイベント、Message Handlerイベントを発生させる。

Hello ProtocolはNeighborデータ構造体の動的な作成を行なう。

4.7 Interfaceイベント群

Interfaceイベント群は以下の役割を担う。

- ルータの設定情報の取得、格納
- DR,BDRの変更から、他のイベントを発生させる。

OSPFDは、設定などから変更されたルータのインターフェースを検知し、この設定情報を取得し格納しなければならない。この設定情報はMessage Handler,Hello Protocol,Neighborイベント群など多くのイベントによって利用される。この設定情報の取得、格納はInterfaceイベント群が行なう。

またプロトコルの設計に習って、InterfaceイベントはDR,BDRなどのInterface状態を見張り、発生した状態の変更に対応した他のイベントを発生させる。

4.8 Neighborイベント群

Hello ProtocolによってNeighborデータ構造体が動的に作成された場合、それぞれのNeighborデータ構造体毎にNeighborイベント群が動作を始める。

それぞれのNeighborイベントは主にHello Protocolによって発生され、プロトコルの設計に習って広告の伝播と同期を個別のNeighborに対して行なう。

4.9 Routing Table Calculation イベント群

Routing Table Calculation イベント群は、LSA の変更を LSA Handler から通知され、経路の計算を行なう。作成した経路表は、ルータのオペレーティングシステム内の経路表に反映される。

Routing Table Calculation イベントは他のイベントを呼ばない。

第5章　OSPFDの実装

5.1　実装目標

本実装では、

1. OSPF for IPv6[4] の設計に沿った動作をする実装を作成する
2. 実装を運用することによって、OSPF for IPv6 の設計に重要な問題点が無いかを調査する事が可能な環境をつくる

事を目標とした。

5.2　実装

前述した目標を元に、IPv6 パッケージ KAME[15] をインストールしたオペレーティングシステム FreeBSD 2.2.7[1] 上で、C 言語を用いて実装した。4 章で述べたように、Zebra ソフトウェア内の経路制御デーモンとして動作する。実装は 9 モジュール、約 9300 行となった。

5.3　状態遷移

OSPF では、Interface データ構造体と Neighbor データ構造体の二つのデータ構造体に状態遷移を持つ。状態遷移の実現方法は、以下の二つが考えられる。

1. 状態を関数で実現する方法
2. データ構造体に状態をあらわす変数を持ち、それを操作する事で実現する方法

Zebra のタイマ機能およびタスク機能は、関数単位のスレッドで実現される。各タスクの実行に要する時間が長い場合、タイマ機能の精度に影響を及ぼす。

このことから、1 の実現方法では、長時間の状態がタイマや他の状態遷移に影響を及ぼす。本実装では 2 の方法を利用する。

Interface 構造体は state 変数を持つ。Interface 構造体の state 変数は設定および Hello プロトコルによって制御される。Interface 構造体の state を以下に示す。

```
/* interface state */
#define IFS_NONE                0
#define IFS_DOWN                1
#define IFS_LOOPBACK            2
#define IFS_WAITING             3
#define IFS_PTOP                4
#define IFS_DROTHER             5
#define IFS_BDR                 6
#define IFS_DR                  7
```

Neighbor 構造体は state 変数を持つ。Neighbor 構造体の state 変数は Hello プロトコルおよび Message Handler によって制御される。Neighbor 構造体の state を以下に示す。

```
/* Neighbor state */
#define NBS_NONE                0
#define NBS_DOWN                1
#define NBS_ATTEMPT             2
#define NBS_INIT                3
#define NBS_TWOWAY              4
#define NBS_EXSTART             5
#define NBS_EXCHANGE            6
#define NBS_LOADING             7
#define NBS_FULL                8
```

5.4 Message Handler

メッセージバッファは iovec 構造体の配列として持つ。iovec 構造体の配列をリストのように扱うため、以下の関数を使用する。これによって、カプセル化されたメッセージを処理する事を実現した。

```
int iov_clear (struct iovec *iov, size_t iovlen);
int iov_free (int mtype, struct iovec *iov, u_int begin, u_int end);
int iov_count (struct iovec *iov);
int iov_index (struct iovec *iov, void *base);
int iov_totallen (struct iovec *iov);
void *iov_prepend (int mtype, struct iovec *iov, size_t len);
void *iov_append (int mtype, struct iovec *iov, size_t len);
void *iov_realloc (int mtype, struct iovec *iov,
                   u_int index, size_t len);
void *iov_attach_last (struct iovec *iov, void *base, size_t len);
void *iov_attach_first (struct iovec *iov, void *base, size_t len);
```

Message Handler は階層化された関数で実現した。これを図 5.1 に示す。

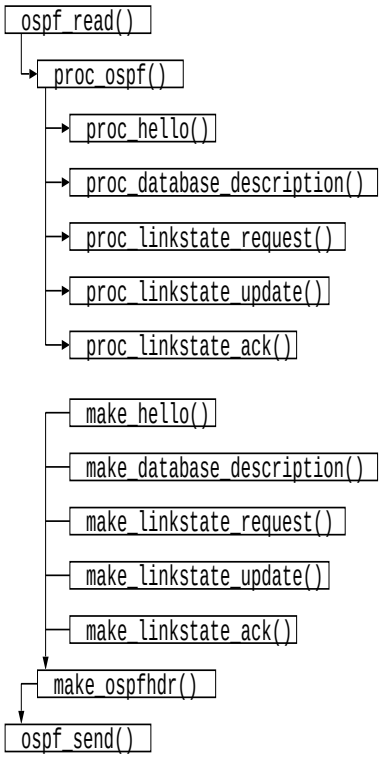

図 5.1: Message Handler

- ospf_read()

  メッセージの受信を行なう。OSPF メッセージヘッダを peek し、OSPF メッセージの長さを調べる。その長さに割当てた受信メッセージバッファにメッセージを読み込む。proc_ospf() 関数を呼んだ後、メッセージに適した proc 関数を呼ぶ。

- proc_ospf()

  どのメッセージを受信した場合も ospf_read() 内にて呼ばれる。OSPF メッセージヘッダを処理する。

- proc_hello()

  Hello メッセージを処理する。処理結果から Interface イベント、Neighbor イベントを呼ぶ。

- proc_database_description()

Database Description メッセージを処理する。Neighbor データ構造体を対象にして動作し、その Neighbor に対しての Message イベントを呼ぶ。

- proc_linkstate_request()

  Link State Request メッセージを処理する。Neighbor データ構造体を対象にして動作し、その Neighbor に対しての Message イベントを呼ぶ。

- proc_linkstate_update()

  Link State Update メッセージを処理する。Neighbor データ構造体を対象にして動作し、flooding の処理関数を通して適した Neighbor に対しての Message イベントを呼ぶ。

- proc_linkstate_ack()

  Link State Acknowledgement メッセージを処理する。Neighbor データ構造体を対象にして動作する。他のイベントは呼ばない。

- make_hello()

  Interface データ構造体から Hello メッセージを作成し、ospf_send() 関数を呼ぶ。

- make_database_description()

  Neighbor データ構造体から Database Description メッセージを作成し、ospf_send() 関数を呼ぶ。

- make_linkstate_request()

  Neighbor データ構造体から Link State Request メッセージを作成し、ospf_send() 関数を呼ぶ。

- make_linkstate_update()

  Link State Update メッセージを作成し、ospf_send() 関数を呼ぶ。LSA の再送の場合、対象となる Neighbor のみに送信する。flooding の処理関数から呼ばれた場合は対象となる Interface にマルチキャストで送信する。終点となるマルチキャストアドレスは Interface データ構造体の状態によって異なる。

- make_linkstate_ack()

受け取った LSA の状態によって、Acknowledge するかどうかが flooding 処理関数内で評価され、する場合に呼ばれる。Neighbor データ構造体から Link State Request メッセージを作成し、ospf_send() 関数を呼ぶ。

- make_ospfhdr()

  OSPF メッセージヘッダをメッセージのはじめに作成する。どのメッセージにおいても ospf_send() 関数内にて呼ばれる。

- ospf_send()

  make_ospfhdr() 関数を呼んだ後、メッセージの送信を行なう。

5.5 Hello Protocol

Hello Protocol は、構造上主に Interface イベントおよび Neighbor イベントに含めた。Interface イベントおよび Neighbor イベントに含まれない Hello Protocol イベントは dr_election() のみである。

dr_election() は、Interface データ構造体を引数にとり、Interface データ構造体の状態を返す。

```
int dr_election (struct interface *);
```

内部では、Interface データ構造体とそれに含まれるすべての Neighbor データ構造体から、このネットワークにおいて DR および BDR を選出する。結果を Interface データ構造体内に格納し、ルータ自身がこのネットワークにおいて DR に選出されたかを、Interface データ構造体の状態として返す。

5.6 Interface イベント

Interface イベントは、プロトコルの設計から以下のものがある。

```
/* interface event */
int interface_up (struct thread *);
int interface_down (struct thread *);
int wait_timer (struct thread *);
int backup_seen (struct thread *);
int neighbor_change (struct thread *);
```

また、設計から、オペレーティングシステムのインターフェース情報を取得するイベントを実装した。

```
void get_interface_all ();
```

5.7 Neighbor イベント

```
/* Neighbor event */
int hello_received (struct thread *);
int twoway_received (struct thread *);
int negotiation_done (struct thread *);
int exchange_done (struct thread *);
int loading_done (struct thread *);
int adj_ok (struct thread *);
int seqnumber_mismatch (struct thread *);
int bad_lsreq (struct thread *);
int oneway_received (struct thread *);
int inactivity_timer (struct thread *);
```

また、Init 状態から ExStart 状態へ進むか、2Way 状態へ進むかを決定するため、以下の関数を実装した。

```
int need_adjacency (struct neighbor *);
```

これは、Neighbor データ構造体を引数にとり、ExStart 状態に進む場合は 1 を、2Way に進む場合は 2 を返す。

5.8 LSA Handler

5.8.1 LSA 内部表現

LSA は、それぞれ独立に扱われなくてはならない。この為、Link State Update メッセージで受け取られた LSA は、一度コピーされる。

LSA は、内部では以下に示す構造体と組にして扱われる。この構造体をインターナルヘッダと呼ぶ。

```
struct lsa_internal
{
  struct lsa_hdr   *lsh;
  u_int32_t         birth;     /* tv_sec when LS age 0 */
  u_int32_t         installed; /* tv_sec when installed */
  struct thread    *expire;
  struct thread    *refresh;   /* For self-originated LSA */
  struct neighbor  *from;
  struct area      *area;
  struct interface *iface;
};
```

LSA は、その操作をする場合にどの Neighbor から受けとったかを参照する。これは、LSA 受信時に lsa_internal 構造体内の from フィールドに格納される。

また、LSA の age を計算する為の変数も lsa_internal 構造体に格納されている。これは、5.8.3 節で説明する。

5.8.2 LSA データベース

LSA データベースは、lsa_internal 構造体へのポインタのリストで実現している。LSA データベースは、LSA type と LSA ID によってハッシュされている。

```
  list lsdb[AREALSTYPESIZE][HASHVAL];
          /* (struct lsa_internal *) as Data */
```

また、Neighbor データ構造体に持つ LSA のリストは、以下の様に実装した。

```
  list summarylist;  /* (struct lsa_internal *) as Data */
  list retranslist;  /* (struct lsa_internal *) as Data */
  list requestlist;  /* (struct lsa_internal *) as Data */
```

5.8.3 LSA Age

LSA の Age は、一秒毎に増やさなくてはならない。しかし、無数ある LSA の変数を一秒毎に変更すると、計算量の多さからデーモン内全てのタイマに影響が出る事が予想される。

このため LSA の Age は、インストール時に Age 0 の時刻を計算して記録し、必要になる度に Age を計算する。Expire や Refresh のために、各 LSA に対してタイマを設定する。

インストール時に計算するための関数を以下に示す。

```
int
calc_lsa_age_internal (struct lsa_internal *lsi)
{
  struct timeval now;

  gettimeofday (&now, (struct timezone *)NULL);
  lsi->birth = now.tv_sec - ntohs (lsi->lsh->lsh_age);
  lsi->expire = thread_add_timer (master, expire_lsa_age, lsi,
                                  lsi->birth + MAXAGE - now.tv_sec);
  return 0;
}
```

Age 0 の時刻を計算し、lsa_internal 構造体の birth フィールドに格納する。LSA Expire のタイマを設定し、後にキャンセルする事ができるように、expire フィールドにタイマへのポインタを格納する。

LSA の現在の Age を計算する関数を以下に示す。

```
u_short
calc_lsa_age_external (struct lsa_internal *lsi)
{
  struct timeval now;

  gettimeofday (&now, (struct timezone *)NULL);
  return (now.tv_sec - lsi->birth);
}
```

5.8.4 LSA インストール

LSA のインストール時には、まず既に同じ LSA がインストールされているかが調べられる。Neighbor データ構造体内の LSA リストには、この LSA へのポインタが含まれている可能性がある。そのため、LSA を変更する場合には前の LSA に付随する lsa_internal 構造体を使用しなくてはならない。これは、lsa_internal 構造体へのポインタを変更しないためである (図 5.2)。

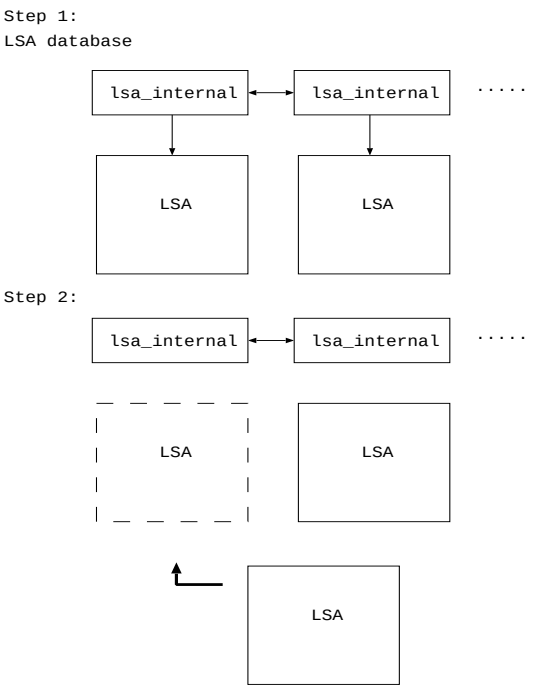

図 5.2: LSA の入れ換え

5.9 経路表計算

5.9.1 Routing Table Calculation イベント

経路表の計算を行なう Routing Table Calculation イベントは、二つのイベントとして実現した。

```
int spf_calculation (struct area *area);
int routing_table_calculation (struct area *area);
```

spf_calculation() 関数は、lsa_install() 関数の中でトポロジ計算が必要か調べられ、必要である場合に呼ばれる。この関数はトポロジを計算する。

routing_table_calculation() 関数は lsa_install() 関数から呼ばれる。この関数は OSPFD 内部の経路表を作成する。以前の経路表を記憶しており、作成された経路表との差分がオペレーティングシステムの経路表に反映される。

5.9.2 Shortest Path Tree

トポロジのグラフの頂点は、以下の構造体で記述した。

```
struct vertex                /* Transit Vertex */
{
  u_int32_t            vtx_id[2];
               /* [0:Router-ID][1:Interface-ID (which can be 0)] */
  struct lsa_internal *vtx_lsa;      /* Associated LSA */
  list                 vtx_nexthops; /* For ECMP */
  u_int32_t            vtx_distance; /* Distance from Root (Cost) */
  list                 vtx_path;
                   /* List of Path described by (struct vertex *) */
  struct vertex       *vtx_parent;   /* for vertex on candidate list */
  u_int8_t             vtx_depth;    /* for vertex on spf tree */
};
```

Vertex ID の変数が u_int32 の配列になっているのが、OSPFv2 からの変更点である。配列の 1 番目には Router ID を、2 番目には Interface ID を格納する。Interface ID が 0 であった場合、ルータを表わす頂点であることがわかる。

また、Shortest Path Tree は以下の構造体で記述した。

```
struct spftree
{
  struct vertex *root;
  list searchlist[HASHVAL][HASHVAL]; /* (struct vertex *) */
  list depthlist[MAXDEPTH];          /* (struct vertex *) */
};
```

root 変数には、常に計算するルータ自身を表す vertex 構造体へのポインタが格納される。

対象となっている Vertex が Shortest Path Tree に既にインストールされているかを調べるには、searchlist リストを利用する。配列の 1 次元目は Router ID でハッシュされ、2 次元目は Interface ID でハッシュされている。

経路表の計算には depthlist リストが利用される。経路は 1 ホップ目から順に、routing_table_calculation() 関数内で計算される。

Vertex を Shortest Path Tree にインストールする関数を以下に示す。

```
int
spf_install (struct vertex *v, struct area *area)
{
  struct vertex *parent;
  struct nexthop_info *nh;
  listnode n;

  if (v->vtx_parent == (struct vertex *)NULL)
    {
#ifdef DEBUG_SPFCALC
      log ("SPFCALC: Installing Root...\n");
      print_vertex (v);
#endif
      area->spftree.root = v;
    }
  else
    {
      list_add_node (v->vtx_parent->vtx_path, v);
    }

  list_add_node (area->spftree.searchlist
                 [hash (v->vtx_id[0])][hash (v->vtx_id[1])], v);
  list_add_node (area->spftree.depthlist [v->vtx_depth], v);

  return;
}
```

第6章　評価

この章では、5 章で述べた実装を実際に動作させ、評価を行なう。

6.1　評価環境

図 6.1 の評価環境において評価を行なった。評価環境は、ループを形成する複雑なトポロジとした。

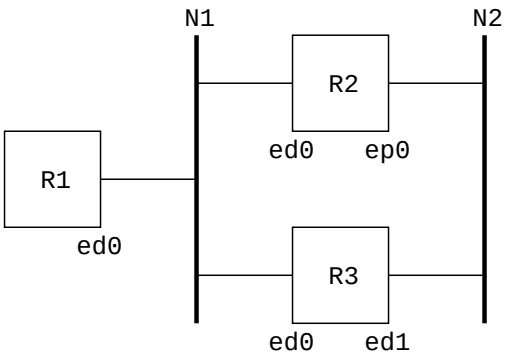

図 6.1: 評価環境

図 6.1 において、ルータは R、ネットワークは N、インターフェースは IF で表わす。各ルータ R1、R2、R3 はネットワーク N1 へのインターフェースを持つ。ルータ R2、R3 はそれぞれネットワーク N2 へのインターフェースを持つ。このネットワークトポロジでは、ルータ R1 からネットワーク N2 への経路は、

1. ルータ R2 経由
2. ルータ R3 経由

の 2 通り存在する。

評価環境において割り当てた IPv6 グローバルアドレスを表 6.1 に示す。

また、IPv6 においてネクストホップアドレスに利用される、リンクローカルアドレスを表 6.2 に示す。

表 6.1: 割り当てた IPv6 グローバルアドレス

		アドレス
R1	ed0	3ffe:501:100c:1::1
R2	ed0	3ffe:501:100c:1::2
	ep0	3ffe:501:100c:2::2
R3	ed0	3ffe:501:100c:1::3
	ed1	3ffe:501:100c:2::3
N1		3ffe:501:100c:1::/64
N2		3ffe:501:100c:2::/64

表 6.2: 割り当てた IPv6 リンクローカルアドレス

		アドレス
R1	ed0	fe80::200:f4ff:fe36:e2ad
R2	ed0	fe80::204:acff:fe90:1c9
	ep0	fe80::2a0:24ff:feaa:936a
R3	ed0	fe80::280:adff:fe16:8638
	ed1	fe80::200:f4ff:fe5d:10d9

Router ID のフィールドの長さは OSPFv2 から変更されていないので、IPv4 アドレスで表す。Router ID は以下の様に設定した。

R1　192.168.0.1
R2　192.168.0.2
R3　192.168.0.3

6.2　実装評価

前述の評価環境のそれぞれのルータにおいて、本実装 OSPFD を起動し、動作確認を行なった。

R2 の設定を以下に示す。

```
router ospf
 ospf version 3
 ospf router id 192.168.0.2
  interface ep0 area 0.0.0.0 up
  interface ep0 cost 1
  interface ep0 hellointerval 10
  interface ep0 routerdeadinterval 40
  interface ep0 rxmtinterval 5
  interface ep0 routerpriority 1
  interface ep0 inftransdelay 1
!
  interface ed0 area 0.0.0.0 up
  interface ed0 cost 1
  interface ed0 hellointerval 10
  interface ed0 routerdeadinterval 40
  interface ed0 rxmtinterval 5
  interface ed0 routerpriority 1
  interface ed0 inftransdelay 1
!
```

R1、R3 についても同様に設定を行なった。cost と router priority はすべてのルータにおいて 1 に設定した。

結果として、各ルータにおける Neighbor の出力と、内部経路表の出力を図示する。

R1 から N2 への経路は二通りあるが、本研究では Equal Cost Multi-Path[11] が実装されていないために、図 6.5 では経路が 1 つの様に見える。

```
ospfd¿ show neighbor
RouterID      InterfaceID   State   DR            BDR           I/F[State]
192.168.0.2   0.0.0.11      FULL    192.168.0.2   192.168.0.3   ed0[DROTHER]
192.168.0.3   0.0.0.1       FULL    192.168.0.2   192.168.0.3   ed0[DROTHER]
```

図 6.2: ルータ R1 における Neighbor の出力

```
ospfd¿ show neighbor
RouterID      InterfaceID   State   DR            BDR           I/F[State]
192.168.0.3   0.0.0.2       FULL    192.168.0.2   192.168.0.3   ep0[DR]
192.168.0.3   0.0.0.1       FULL    192.168.0.2   192.168.0.3   ed0[DR]
192.168.0.1   0.0.0.8       FULL    192.168.0.2   192.168.0.3   ed0[DR]
```

図 6.3: ルータ R2 における Neighbor の出力

```
ospfd¿ show neighbor
RouterID      InterfaceID   State   DR            BDR           I/F[State]
192.168.0.2   0.0.0.11      FULL    192.168.0.2   192.168.0.3   ed0[BDR]
192.168.0.1   0.0.0.8       FULL    192.168.0.2   192.168.0.3   ed0[BDR]
192.168.0.2   0.0.0.10      FULL    192.168.0.2   192.168.0.3   ed1[BDR]
```

図 6.4: ルータ R3 における Neighbor の出力

```
ospfd¿ show table
Routing Table
DESTINATION     /   LEN   NEXTHOP                    IF    COST
————-
3ffe:501:100c:1::   /    64   ::                        ed0      1
3ffe:501:100c:2::   /    64   fe80:1::280:adff:fe16:8638   ed0      2
```

図 6.5: ルータ R1 における経路表の出力

```
ospfd¿ show table
Routing Table
DESTINATION    /   LEN   NEXTHOP    IF   COST
———-
3ffe:501:100c:2::  /    64   ::          ep0      1
3ffe:501:100c:1::  /    64   ::          ed0      1
```

図 6.6: ルータ R2 における経路表の出力

```
ospfd¿ show table
Routing Table
DESTINATION    /   LEN   NEXTHOP    IF   COST
———-
3ffe:501:100c:1::  /    64   ::          ed0      1
3ffe:501:100c:2::  /    64   ::          ed1      1
```

図 6.7: ルータ R3 における経路表の出力

R1、R2、R3 の経路表では、すべてのネットワークへの経路が存在していることがわかる。

次に、ネットワーク管理者の意向から経路を変更することができるかを確認するため、関係するインターフェースの cost を変更し、R1 からN2 への経路を調べる。R3 のインターフェースed1 の cost を 10 に設定した場合のR1 の経路表を図 6.8 に、R2 のインターフェースep0 の cost を 10 に設定した場合のR1 の経路表を図 6.9 に示す。

```
ospfd¿ show table
Routing Table
DESTINATION     /   LEN   NEXTHOP                    IF    COST
————-
3ffe:501:100c:1::    /     64   ::                        ed0         1
3ffe:501:100c:2::    /     64   fe80:b::204:acff:fe90:1c9 ed0         2
```

図 6.8: R3 のed1 の cost が 10 の場合のR1 の経路表

```
ospfd¿ show table
Routing Table
DESTINATION     /   LEN   NEXTHOP                    IF    COST
————-
3ffe:501:100c:1::    /     64   ::                        ed0         1
3ffe:501:100c:2::    /     64   fe80:1::280:adff:fe16:8638 ed0         2
```

図 6.9: R2 のep0 の cost が 10 の場合のR1 の経路表

このように、cost の設定から経路が変更される。ここから、ネットワーク管理者の意向を経路に反映する機構が実現されている。

6.3 プロトコル評価

本節では、経路制御トラフィックの量を RIPng と比べる事によって、経路制御プロトコルの評価を行なう。

経路制御トラフィックは、R2 のed0 を対象に、traceroute コマンドの出力から時間とトラフィックを解析し、図 6.10 に表した。

この図から、RIPng のトラフィックは常にトラフィックの量が多いのに対して、OSPFD のトラフィックは一時期を除いてトラフィックが少ない。

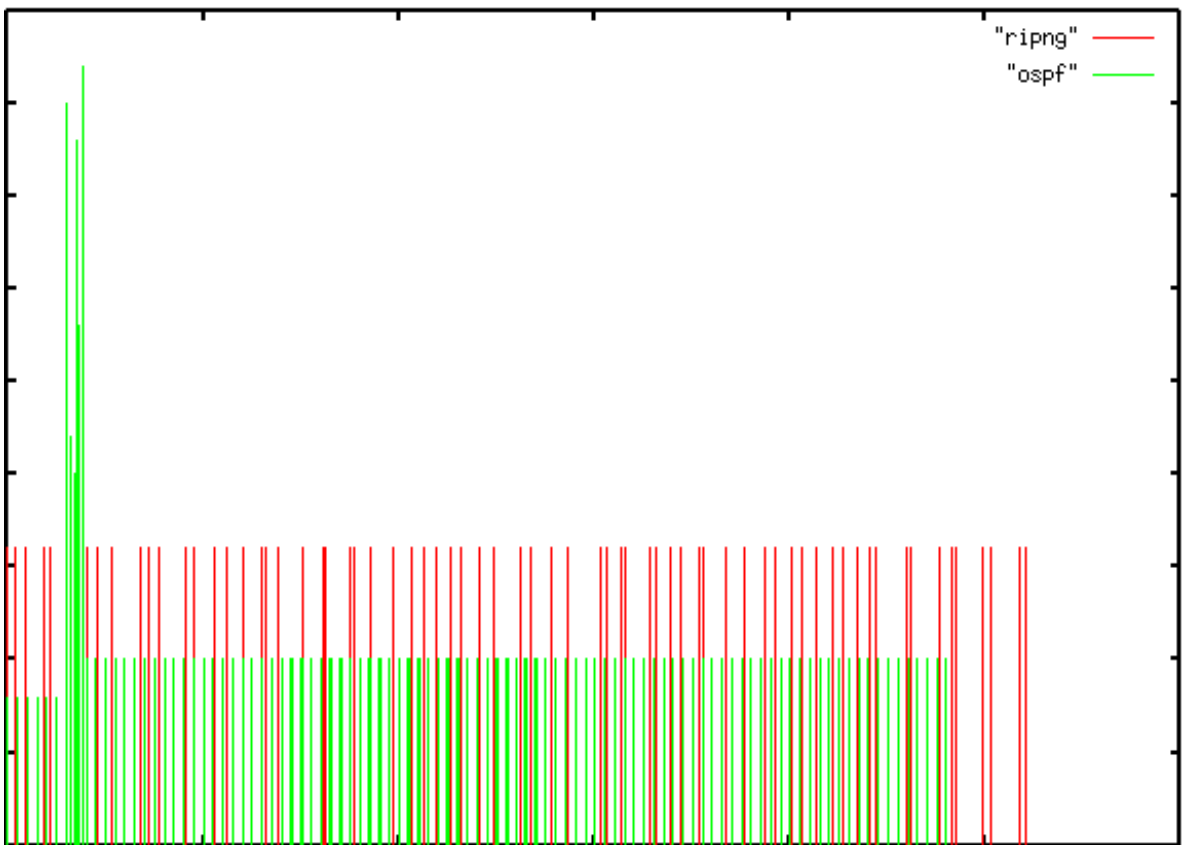

図 6.10: M1 における RIPng と OSPFD のトラフィック量

第7章　結論

7.1　本研究の概要

本研究は、以下の手順で行なった。

1. 現状調査

   インターネットの基盤技術における現状の調査を行なった。具体的には情報転送を行なうインターネットプロトコル、および経路制御環境の現状を調査した。これらを 1 章、2 章にまとめた。ここでは、IPv6、経路制御機構の概要および重要性を述べた。

2. 問題点の発見

   現状の調査を行なう中で発見された問題点を 2 にまとめた。ここでは、経路制御機構の特性として重要な事項を述べた。そして、そのそれぞれについて問題点を述べた。経路制御機構の特性として重要な事項は以下の通りである。

   - プロトコルの汎用性
   - 経路の収束速度
   - 経路制御トラフィックの量
   - ネットワーク管理者の意向を正確に反映できる機構の有無

3. 解決案の特定

   本研究における問題点の解決案として、リンクステート型経路制御プロトコルである OSPF を取りあげた。提案された設計を吟味し、3 章にまとめた。

4. 解決案の実現

   解決案の実現として、本研究では IPv6 をサポートした OSPF を設計、実装した。設計は 4 章に、実装は 5 章にまとめた。

5. 解決案の評価

   本研究において実装した OSPFD を、試験トポロジにおいて動作させ、評価した。これは 6 章にまとめた。評価によって、本研究の解決案が、本研究でとりあげた問題点を解決している事が確かめられた。

7.2 まとめ

インターネットの拡大を抑制しないために、新しいインターネットプロトコルである IPv6 への移行が必要である。しかしながら、現在の IPv6 ネットワークには、経路制御に起因するさまざまな問題点が存在する。

経路制御トラフィックの多さ、トポロジによって経路制御が不可能である事、経路の収束が遅いなどの問題点は、劣悪な通信環境を生む事が予想される。また、ネットワーク管理者の意向を経路制御に正確に反映する事は、ネットワーク資源を有効活用するために重要であるが、IPv6 ネットワークにおいてこの機構が欠如している。

本研究では、これらの問題点の重要性を踏まえ、その解決案として IPv6 をサポートした OSPF 経路制御プロトコルである OSPFv3 を実装し、評価した。

IPv6 ネットワークにおいて OSPFv3 を利用する事によって、本研究で取り上げた問題点が解決する事が確認された。

7.3 今後の課題

本実装の今後の課題として、スケーラビリティの試験が挙げられる。本研究の評価試験においては、巨大なネットワークを経路制御するという事は成されていない。巨大なネットワークにおいて定常運用する事によって、スケーラビリティの試験を行なう必要がある。

また、経路制御環境の今後の課題として、以下の事項が挙げられる。

- 経路制御機構における自動設定
- マルチキャスト通信をサポートした経路制御
- 片方向通信路など、新しい通信媒体のサポート
- 実時間通信をサポートした経路制御

謝辞

本研究を進めるにあたり、御理解と御指導頂きました慶應義塾大学環境情報学部教授の徳田英幸博士と同学部教授の村井純博士に感謝します。また、日頃から絶えず多くの助言を頂きました環境情報学部助教授の楠本博之博士、同学部講師の中村修博士、同学部助手の植原啓介氏に感謝します。

また、常に的確なアドバイスと励ましの御言葉を頂きましたWIDEプロジェクトv6ワーキンググループ、ならびに徳田・村井・中村・楠本研究会の皆様に感謝します。

最後に、本稿を書き上げるにあたり、根気良くアドバイスを頂きました慶應義塾大学政策・メディア研究科の重近範行氏、土本康生氏、関谷勇司氏、土井裕介氏の諸氏に感謝の念を表し、謝辞と致します。

ありがとうございました。

参考文献

[1] Freebsd home page. WWW. http://www.freebsd.org/.

[2] Zebra home page. WWW. http://www.zebra.org/.

[3] R.W. Callon. Use of osi is-is for routing in tcp/ip and dual environments. Request For Comments 1195, IETF, December 1990.

[4] R. Coltun, D. Ferguson, , and J. Moy. Ospf for ipv6. Internet draft, IETF, November 1997. Working in progress.

[5] C. L. Dedrick. Routing information protocol. Request For Comments 1058, IETF, June 1988.

[6] K. Egevangand P. Francis. The ip network address translator(nat). Request For Commnets 1631, IETF, May 1994.

[7] V. Fuller, T. Li, J. Yu, , and K.Varadhan. Classless inter-domain routing (cidr):an address assignment and aggregation strategy. Request For Comments 1519, IETF, September 1993.

[8] Christian Huitema. *IPv6 The New Internet Protocol Second Edition*. PRENTICE HALL, 1997.

[9] Charles L. and Hedrick Rutgers. Cisco - igrp/eigrp. WWW. http://www.cisco.com/warp/public/103/.

[10] G. Malkin and R. Minnear. Ripng. Request For Comments 2080, IETF, January 1997.

[11] J. Moy. Ospf version 2. Request For Comments 2328, IETF, April 1998.

[12] IETF ngtrans Working Group. 6bone home page. WWW. http://www.6bone.net/.

[13] D. Oran. Osi is-is intra-domain routing protocol. Request For Comments 1142, IETF, February 1990.

[14] J. Postel. Internet protocol. Request For Comments 791, IETF, September 1981.

[15] Kame Project. Kame home page. WWW. http://www.kame.org/.