修士論文 1995 年度 (平成 7 年度)

階層的手法を用いたネットワークの自動設定機構に関する研究

慶應義塾大学 大学院
政策・メディア研究科

冨永 明宏

修士論文要旨 1995 年度 (平成 7 年度)

階層的手法を用いたネットワークの自動設定機構に関する研究

本論文では階層化モデルに基づいたアドレスの自動設定機構を提案し、このモデルに基づいて実現したネットワークの自律的な構成およびアドレスの付け替え支援システムについて述べる。

本システムは、サーバ群が自律的に階層構造を決定するためのサーバ木構成部、木構造に基づきネットワークの設定を行なうネットワーク設定部、個々の計算機を設定するためのホスト設定部の 3 つの要素から構成される。サーバ木構成部には Distance-Vector 型の経路制御アルゴリズムに変更を加えたものを適用した。その結果として階層構造を素早くかつ簡潔に決定できるようになった。ネットワーク設定部は多様なネットワーク管理方針を実現できるようにポリシと独立にプロトコルを設計および実装した。さらに単一ネットワークの簡易エミュレータを作成し、このネットワークを組み合わせて構築した仮想 LAN 上でネットワーク設定部の基本動作を検証した。ホスト設定部は他の 2 つの部分からは独立に実装し、実際のネットワーク上でアドレス付け替えが行なわれることを確認した。これらの結果から、ネットワークのアドレス自動設定管理機構として階層化モデルに基づいた本システムが有効であることを実証した。

本システムを用いることにより、インターネットへ計算機を接続する際に必要なホストアドレスなどの共有資源の割当およびゲートウェイアドレス等の環境に関する情報取得・設定が可能となる。これに加えて今後到来する大規模な “Ubiquitous Computing” の環境において必要となる、自律的、動的なネットワークの構成機構およびネットワーク単位でのインターネット接続を実現した。さらに本システムはアドレスの付け替え作業を容易なものとすることにより、現在のインターネットで問題となっているネットワークアドレスの枯渇や経路情報の増加にともなうネットワーク運用の障害への対策に効果的であることが分かった。

キーワード

1　ネットワーク自動設定	2　アドレス管理	3　アドレス付け替え
4　自律構成	5　インターネット	

慶應義塾大学 大学院 政策・メディア研究科

89432121　冨永 明宏

Abstract of Master's Thesis Academic Year 1995

Research on Automatic Network Configuration Mechanism with Hierarchical Scheme

This paper proposes an mechanism based on the hierarchical model for the automatic address configuration. It also describes about the system which is based on this model, which enables autonomous network construction and supports the address renumbering.

This system consists of three parts: A server-tree construction part to determine hierarchy of servers by themselves, the network configuration part to configure network on the basis of the tree construction, and a host configuration mechanism to configure each hosts. The server-tree construction part uses a modified routing algorithm which is known as a Distance-Vector. As a result, it made possible the server hierarchy to be determined rapidly and briefly. The network configuration part is designed and implemented as a policy independent of its protocol to support various network management policies.

The network configuration mechanism works on the virtual LAN constructed from simple network emulators. The host configuration part is implemented independently of other parts, and is confirmed to be enable automatic addressing on the network. As a result, this system based on hierarchy model is proved to be valid as auto addressing configuration mechanism.

This system enables to assign shared resources like host addresses which is required when connecting computers to Internet and to configure and obtain information about environments such as a default gateway address. In addition, in forthcoming large "Ubiquitous Computing" environment, autonomous and dynamic network construction and Internet connectivity are realized.

Furthermore, this system made address renumbering easy, which confirms effectiveness to obstacles in network administration with the shortcoming of network addresses and the increase of routing information as problems in the today's Internet.

Key Words

1	Automatic Network Configuration	2	Address Management	3	Address Renumbering
4	Autonomous Construction	5	Internet		

Keio University Graduate School of Media and Governance

89432121 Akihiro Tominaga

目 次

第 1 章 序論 **1**
1.1 背景 1
1.2 目的 2
1.3 本論文の構成 3

第 2 章 インターネットの現状と問題点 **4**
2.1 携帯型計算機の普及 4
2.2 IP アドレスの枯渇 5
2.3 経路情報の急増によるネットワーク運用への支障 6
2.4 プロバイダベース経路制御の普及 9
2.5 家庭内 LAN の登場 10
2.6 アドレス付け替えに関する考察 11
2.7 Ubiquitous Computing(遍在する計算機環境) 12

第 3 章 ネットワーク設定管理手法の現状 **14**
3.1 Reverse Address Resolution Protocol (RARP) 14
3.2 Network Information Protocol (NIP) 15
3.3 Bootstrap Protocol (BOOTP) 15
3.4 Point-to-Point Protocol (PPP) 16
3.5 Dynamic Host Configuration Protocol (DHCP) 17
3.6 IPv6 Autoconfiguration 18

第 4 章 階層化モデルによるネットワーク自動設定機構の提案 **19**
4.1 設計目標 19
4.2 新しいネットワーク設定管理モデル 20
4.2.1 ネットワークおよび計算機に関する前提 20
4.2.2 ネットワークの二重運用モデル 22
4.2.3 階層構造モデルの導入 23

第 5 章 ネットワーク自動設定機構の設計 **25**
5.1 サーバ木構成部の設計 25
5.1.1 設計目標 25

5.1.2 設計 26
5.2 ネットワーク設定部の設計 30
5.2.1 設計目標 31
5.2.2 設計 32
5.3 ホスト設定部の設計 36
5.3.1 設計目標 38
5.3.2 設計 39
5.4 アドレス付け替えの概要 40
5.4.1 ホスト設定部における支援 42
5.4.2 ネットワーク設定部における支援 42
5.5 ネットワーク管理ポリシに関する考察 43

第 6 章 階層型ネットワーク自動設定機構の実装 **44**
6.1 サーバ実装の構造 44
6.2 サーバ木構成部の実装 46
6.3 ネットワーク設定部の実装 48
6.4 ホスト設定部の実装 51
6.4.1 サーバの実装 51
6.4.2 クライアントの実装 54
6.5 Ethernet エミュレータの実装 56

第 7 章 評価・考察 **59**
7.1 実験環境について 59
7.2 評価 60
7.2.1 サーバ木構成部の実行例 60
7.2.2 サーバ木構成部に関する考察 62
7.2.3 ネットワーク設定部の評価および考察 62
7.2.4 ホスト設定部の評価および考察 64

第 8 章 結論 **65**
8.1 まとめ 65
8.2 今後の課題 65
8.2.1 短期的課題 65
8.2.2 長期的課題 66

謝辞 **68**

図 目 次

2.1 IP アドレスの利用率 (サブネット長別) 6
2.2 IP アドレスの割り当て数 (全世界) 7
2.3 あるルータ上の経路情報数の推移 8
2.4 経路の集約が可能な場合 9
2.5 経路の集約が不可能な場合 9

3.1 ARP/RARP のパケットフォーマット 14

4.1 VLSM の必要な例 21
4.2 連続/不連続に設置されたサーバ群 22
4.3 ネットワークの二重運用例 23
4.4 ネットワーク階層化の一例 24

5.1 サーバ木構成メッセージのフォーマット (Ethernet の場合) 30
5.2 ネットワーク設定メッセージのフォーマット (Ethernet の場合) 34
5.3 アドレスバルクの状態遷移 37
5.4 アドレスプールの状態例 38
5.5 ホスト設定メッセージのフォーマット (Ethernet の場合) 41

6.1 Solicitation メッセージのフォーマット 49
6.2 エミュレータの概念図 57
6.3 エミュレータ登録メッセージのフォーマット 58

7.1 仮想実験ネットワーク 59
7.2 実験ネットワーク 60
7.3 仮想ネットワークに構築された階層構造 62

表 目 次

5.1 サーバ木構成メッセージの各フィールド 31
5.2 ネットワーク設定部で用いるメッセージの種類 33
5.3 ネットワーク設定メッセージの各フィールド 34
5.4 アドレスバルクの状態 35
5.5 ホスト設定メッセージの各フィールド 41

6.1 schedule 構造体 45
6.2 Solicitation メッセージの各フィールド (Ethernet の場合) 49
6.3 ホスト設定サーバ用 interface 構造体 52
6.4 netinfo 構造体 53
6.5 addrinfo 構造体 54
6.6 ホスト設定クライアント用 interface 構造体 55
6.7 エミュレーションメッセージのヘッダ 58
6.8 エミュレータ登録メッセージの各フィールド 58

7.1 ネットワーク設定部の評価 63
7.2 ホスト設定部の評価 64

第 1 章

序論

1.1 背景

インターネットでは、経路、遅延、ネットワークアドレスなどが動的に変化する。インターネット上で提供されるサービスや接続される計算機および各種デバイスなども数、種類ともに増加し続けている。それらの変化に対応できる柔軟性こそがインターネットの特徴である。

インターネット上の計算機が互いに通信する際に、相手を一意に指定するための識別子がIP アドレスである。また各種サービスの識別子としてもIP アドレスはポート番号と組で利用されている。IP アドレスの値は、接続されるネットワークに依存して決定される。常に変化するインターネットにおいても、IP アドレスは計算機がインターネットに接続された瞬間から切断されるまで変更されないのが一般的であった。同時に計算機も持ち運ぶのに不適当な大きさだったため、ネットワークへの接続から切断までの周期も比較的長かった。

計算機が一旦ネットワークに接続されるとネットワークへの接続ポイントや計算機を取り巻くネットワーク環境は希にしか変化せず、あるいは変化したとしても手動で十分対応できる程度の頻度でしか発生しないとの仮定にもとづき、識別子としてIP アドレスが用いられていた。しかし近年のインターネットを取り巻く状況の変化により、この前提に基づいて議論することが不可能になってきている。

そうした状況の変化として、第一に計算機およびネットワークカードの小型・高性能化があげられる。多くのユーザがサブノート型の計算機を持ち歩き、ネットワークへの接続と切断を頻繁に繰り返す。ネットワークへ接続された計算機は共有資源であるIP アドレスの割当と設定を必要とする。さらに各種サービスを提供するサーバホストのIP アドレスやルータのIP アドレス等のネットワーク環境に関する情報を取得し、設定する必要もある。これらの作業は煩雑であり、かつ手動で行なう場合は誤りやすく、利用者への負担となっていた。

第二に近い将来に実現される “Ubiquitous Computing”[1] の環境 (身の回りの物すべてに計算機が含まれる) においては、さらに大規模な動的変化がネットワーク上に発生する。Ubiquitous Computing の環境では、極めて多数の計算機が自律的にネット

ワークを構成し、ネットワーク単位で他のネットワークへ接続や切断を繰り返す。また従来のような単体の計算機での接続および切断の頻度もさらに増大する。このような状況において従来のような手動によるネットワーク設定管理手法は負担が大きすぎて実運用に適さない。

第三にネットワークアドレスの枯渇 [2] と、経路情報数の急増によるネットワーク運用への支障 [3] の問題があげられる。インターネットでは未割り当ての IP アドレスが極端に減少しており、近い将来に新しい計算機に対して IP アドレスを割り当てることができなくなる。同時に経路情報の総数が激増しており、メモリの少ないルータでは処理しきれないという状況が発生している。これらの問題に対する抜本的な対策として IPv6 が開発されている [4]。同時にインターネットの運用が IPv6 へ移行するまでの短期的な対応策も実施されている。具体的には CIDR[3] と呼ばれる手法を用いて適切な大きさにネットワークを分割し、アドレスの利用率を向上させてアドレス空間の枯渇を先伸ばしにしている。現在のネットワーク管理手法では、動的にアドレス空間の大きさを変更することは難しい。そのため、あらかじめ十分余裕を持った大きさのアドレス空間を 1 つのサブネットに割り当てるのが一般的である。しかし、実際にそのアドレス空間を使い切ることは希である。またネットワークケーブルの総延長距離や建物の構造等の物理的制約が存在するために、1 サブネットに接続される計算機の総数はアドレス空間の大きさに比例しない。したがって、大きなサブネットになるほどアドレスの利用効率が低下する傾向にある[1]。CIDR を用いると任意の大きさのサブネットが設定できるため、サブネットの大きさを小さくしてアドレスの利用効率を向上させることができる。

経路情報数の急増問題に対しては、複数のネットワークへの経路情報を一つに集約し、経路の総数を減少させる試みが行なわれている。これらの対応策の効果を高めるためにはネットワークアドレスの付け替えが必要となる。しかしネットワークアドレスの付け替えに必要な労力は非常に大きいため、現状ではこれらの手法が適用されるのは新規にネットワークを構築する場合がほとんどである。さらにアドレス付け替えの前後を通じて、インターネットへの接続を安定して確保するための体系的手法は依然として確立されていない。これらの理由によりネットワークアドレスの付け替えは敬遠される傾向にあり、CIDR を用いた短期的な対策も十分な効果を上げていない。

1.2 目的

これらインターネットを取り巻く状況の変化に対応した新しいネットワーク自動設定管理機構について提唱し、実現することが本研究の目的である。本研究の最終的な目的は「大規模ネットワークの自律構成および自動管理」であるが、特に本論文では以下の 3項目について論述する。

1. **大規模ネットワークの設定および管理を自動化する**

[1] 第 2.2節, 図 2.1参照

ネットワークアドレス等の資源割当およびルータアドレス等のネットワーク環境に関する情報の設定を自動化する。単一ネットワークセグメントだけでなく、複数のネットワークセグメントおよびルータからなる中〜大規模ネットワークの設定管理を実現する。

2. **ネットワークアドレスの付け替えを支援する**

   ネットワークアドレスの効率的な利用や経路情報の集約に必要なネットワークアドレスの付け替えを支援する機構を実現する。

3. **ネットワークアドレスの効率的利用を実現する**

   現状では利用率の低い IP アドレスを節約することによりアドレス空間を効率的に利用する。

すなわち本論文では特に IP アドレスという共有資源の自動割当、回収および設定方法について考察する。

1.3 本論文の構成

第 2 章でインターネットを取り巻く状況の変化について述べる。とりわけネットワークアドレスの付け替えが緊急の課題であることを指摘し、アドレスの付け替えによって発生するさまざまな問題について考察する。次に第 3 章で現在のネットワーク設定および管理の手法について述べる。特にネットワークアドレスの設定方法に着目する。またこれらの議論を通じて既存の手法がインターネットの現状に対しては不適切であることを明らかにする。第 4 章では問題解決のためのモデルとして階層的手法を提案する。さらに全体の設計目標を明らかにし、ネットワーク設定機構の設計を行なう。

本論文では階層化モデルに基づいてシステムを「サーバ木構成部」「ネットワーク設定部」「ホスト設定部」の 3 つに分割する。第 5 章では各々の設計目標について議論し、詳細設計について述べる。第 6 章では実装目標を示し、各部の実装について論じる。また、本システムの実験を行なうために作成した Ethernet ドライバをエミュレートするプログラムについても述べる。第 7 章では各部ごとに評価を行なう。最後に第 8 章で結論を述べ、今後の課題を示す。

第 2 章

インターネットの現状と問題点

本章ではインターネットを取り巻く状況の変化の変化を示す事例として「携帯型計算機の普及」「IP アドレスの枯渇」「経路情報の急増によるネットワーク運用への支障」「プロバイダベース経路制御の普及」「家庭内 LAN の登場」の 5 つについて述べる。その後、現在のインターネットでは IP アドレスの付け替えが不可欠であることを示し、付け替えにともなって発生する諸問題について考察する。最後に近い将来の計算機環境として提唱されている Ubiquitous Computing について述べる。

2.1 携帯型計算機の普及

数年前までは重量・容積などの点から計算機を携帯することは不可能であり、またネットワーク機能を利用することができるのは、据え置き型の計算機に限定されていた。したがって計算機の物理的な位置が変更される頻度は低く、同様にネットワーク上の位置も変更されることは少なかった。ネットワーク上の識別子である IP アドレスはネットワーク上の位置に依存して決定されるため、IP アドレスの変更頻度も低かった。

これに対し、近年は計算機の小型・高性能化の傾向が著しい。イーサネットカードなどのネットワーク機器も同様に小型化し、一般個人ユーザに普及した。加えてパーソナルコンピュータ用オペレーティングシステムでも、ネットワーク機能が標準的に提供されるようになっている。その結果として利用者はこれらの携帯型計算機を持ち歩き、手近なネットワークに接続してインターネットを利用したいと望むようになった。

計算機をネットワークに接続した場合には、IP アドレスなどの共有資源の割当を行ない、さらにゲートウェイの IP アドレスのようなネットワーク利用環境に関する情報を設定する必要がある。これらの作業はネットワークに関する基礎知識を必要とする上に、手動で行なう場合には誤りやすいものである。また誤って設定された計算機はインターネット上の他の計算機と通信することができないばかりか、場合によっては正常な計算機に対して悪影響を及ぼすこともある。計算機の接続および切断の頻度が高くなるにつれ、ネットワークに関連した計算機の各種設定を自動的に行なう機構の欠如が問題となってきていた。

2.2 IP アドレスの枯渇

インターネットでは識別子として 32 ビットの整数からなる IP アドレスを使用する。したがって理論上では $2^{32} = 4294967296$ 台の計算機が識別できることになる[1]。これは IP が考案された当初は十分な大きさであると考えられていた。

IP アドレスは運用上のさまざまな理由から、任意の大きさではなくクラス A〜C と呼ばれる 3 通りの大きさを単位として割り当てられていた。したがって、ある組織に 1 台しか計算機が存在しない場合でもクラス C アドレスが 1 つ割り当てられていた。1 つのクラス C アドレスには $2^8 - 2 = 254$ 台の計算機が接続できることを考えると、253 個の IP アドレスが無駄になる。同様にして、1 つのクラス A アドレスには $2^{24} - 2 = 16777214$ 台、1 つのクラス B アドレスには $2^{16} - 2 = 65534$ 台の計算機がそれぞれ接続できるが、実際に利用されているのはその中のごくわずかであることが多い。

図 2.1は慶應義塾大学湘南藤沢キャンパスのネットワークである CNS、同大学大学院政策・メディア研究科のネットワークである ERNS、WIDE インターネット藤沢 NOC(Network Operation Center) 内のネットワークの 3 つについて実際に IP アドレスの利用率を調べたものである。本集計は、ネットワークアドレスとサブネットマスク[2]の 2 つによって指定されるサブネットに対し、その中に含まれる全ての IP ホストアドレスへ個別に ICMP echo request[3]を送り、1 秒以内に応答を返すホストの数を集計したものである。このグラフから、サブネットが小さくなる (サブネットマスクのビット長が長くなる) のに比例して IP アドレスの利用率が高くなることが分かる。

このようなアドレスの利用効率の低さと近年のインターネットの普及により、IP アドレスが数年のうちに枯渇する見通しとなった [2]。図 2.2に IP アドレスの割り当て状況を示す。このグラフには現在までの IP アドレスの割り当て数の推移を示すプロットが描かれている。3 つの斜線は、今後の IP アドレスの割り当て数を予測したものであり、傾きが小さい順に「最も楽観的な予測」「最近の割り当て状況に基づく予測」「最も悲観的な予測」をそれぞれ意味する。グラフの上部を横切る水平線は、IP アドレスの総数を示し、斜線と交わる場所は IP アドレスが枯渇すると予想される時期である。

アドレスの枯渇に関する問題を回避する根本的な対策はアドレス空間を拡大する、すなわちアドレス長を増やすことである。次世代 IP として開発中の IPv6 ではアドレス長が 128 ビットになっている。しかし次世代 IP の開発および移行は一度に実施できるものではないため、当面は現在の IP アドレス空間を使い続けなければならない。そのため現在ではクラス A やクラス B 単位でのアドレス割り当ては原則として行なわず、1 つの組織に複数のクラス C アドレスを割り当てることで IP アドレスの節約を図っている。また既にクラス B のアドレスを取得済みの組織が、それを返却して代わりにクラス C アドレスを使用するという試みも行なわれている。本研究の目的の 1 つは、こ

[1] ただし実際には、あるネットワーク上の全ての計算機を示すネットワークブロードキャストアドレス、ネットワークを示すネットワークアドレス、マルチキャストに利用されるマルチキャストアドレスなどのために確保された領域が存在するため、計算機の識別に利用できるアドレスの数は、これよりも少ない。

[2] ネットワークの大きさを示すビットマスクのこと。

[3] ICMP echo request メッセージを受信したホストは、ICMP echo reply メッセージを返す。

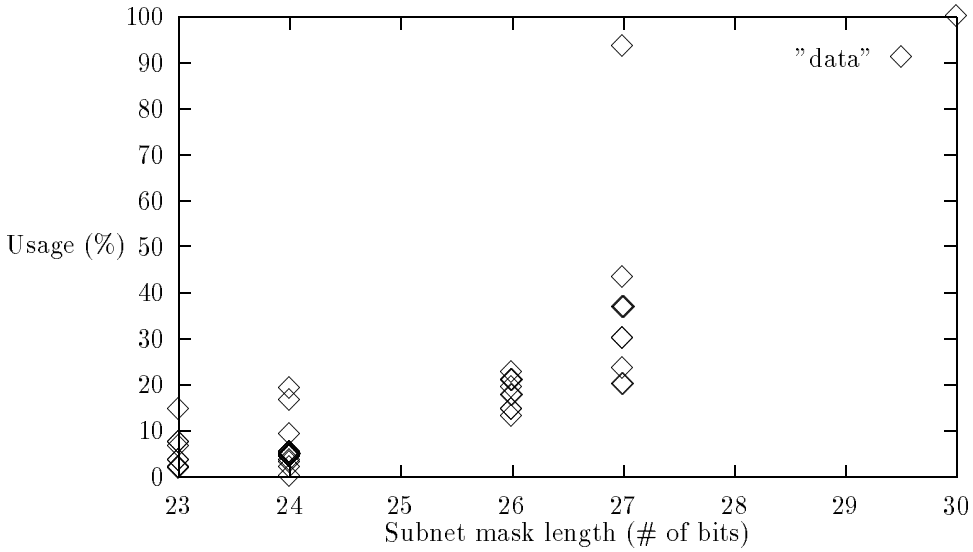

図 2.1: IP アドレスの利用率 (サブネット長別)

のようなアドレスの付け替えを支援することである。

2.3 経路情報の急増によるネットワーク運用への支障

ルータは自らが管理する経路表に基づいて IP パケットの転送を行なうが、経路情報を 1 つ保持するのに、最低限でも宛先 IP アドレスと転送先となるルータの IP アドレスを保持しなければならない。さらに通常はメトリック[4]やネットマスクなどの情報も保持される。したがって経路の増加に比例して、それを保持するのに必要なメモリの量も増大する[5]。ルータのメモリは有限であるため、保持できる経路表の大きさも有限である。

IP アドレスの枯渇問題に対する解決策として、クラス B アドレスを 1 つ割り当てる代わりに複数のクラス C アドレスを割り当てて、IP アドレスの利用率を向上させていることは既に述べた。しかし IP アドレスの利用率が向上する代わりに、クラス C アドレスの各々につき 1 つの経路情報がインターネットを構成している各ルータで保持されることになる。さらに近年のインターネットの普及が経路情報の増加に拍車をかけている。これらの結果、ルータの保持できる経路数の限界を超えた経路情報が各ルータに配送される事態が発生している。限界を超える経路情報を保持しようとしたルータは誤動作するだけでなく、インターネットに悪影響を及ぼすことさえある。

このように経路情報が急増することによって、ネットワークの運用に支障が発生す

[4]宛先までの距離を表す値のこと。

[5]例えば宛先 IP アドレス (4 オクテット), ネットマスク (4 オクテット), 転送先ルータの IP アドレス (4 オクテット), メトリック (1 オクテット) を保持すると仮定すると、1 つの経路につき 13 オクテットのメモリが必要となる。

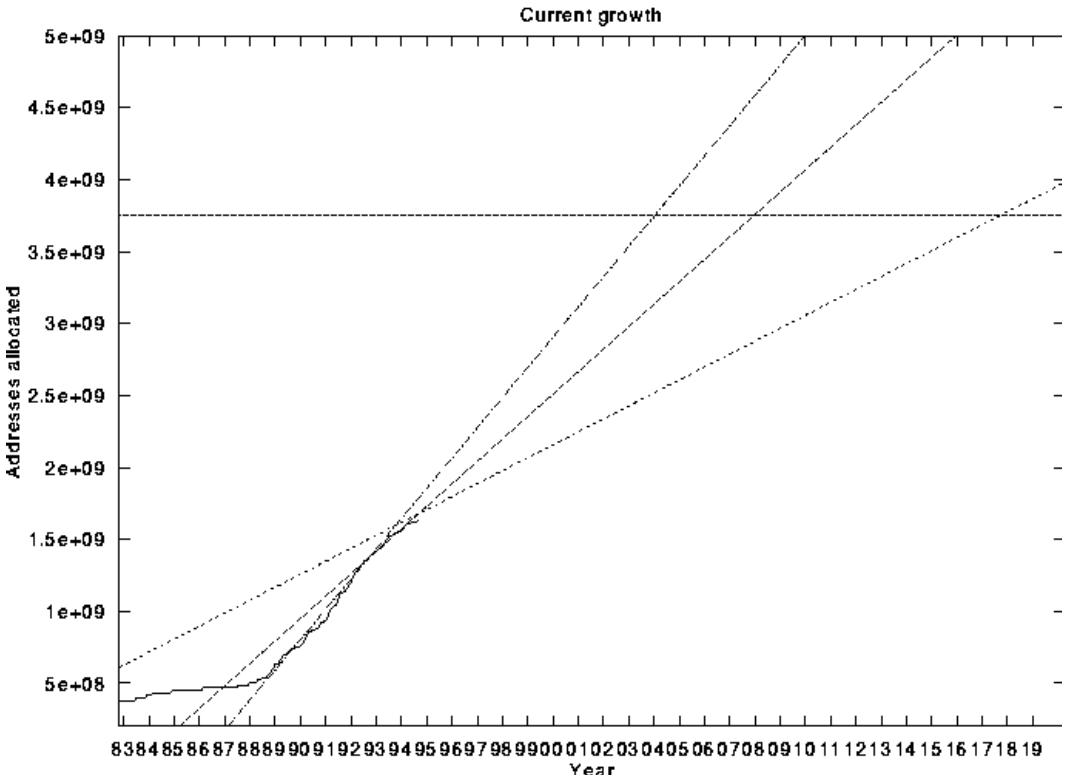

図 2.2: IP アドレスの割り当て数 (全世界)

るようになりつつある。近年の経路情報の増大をグラフにしたものが図 2.3である。こ

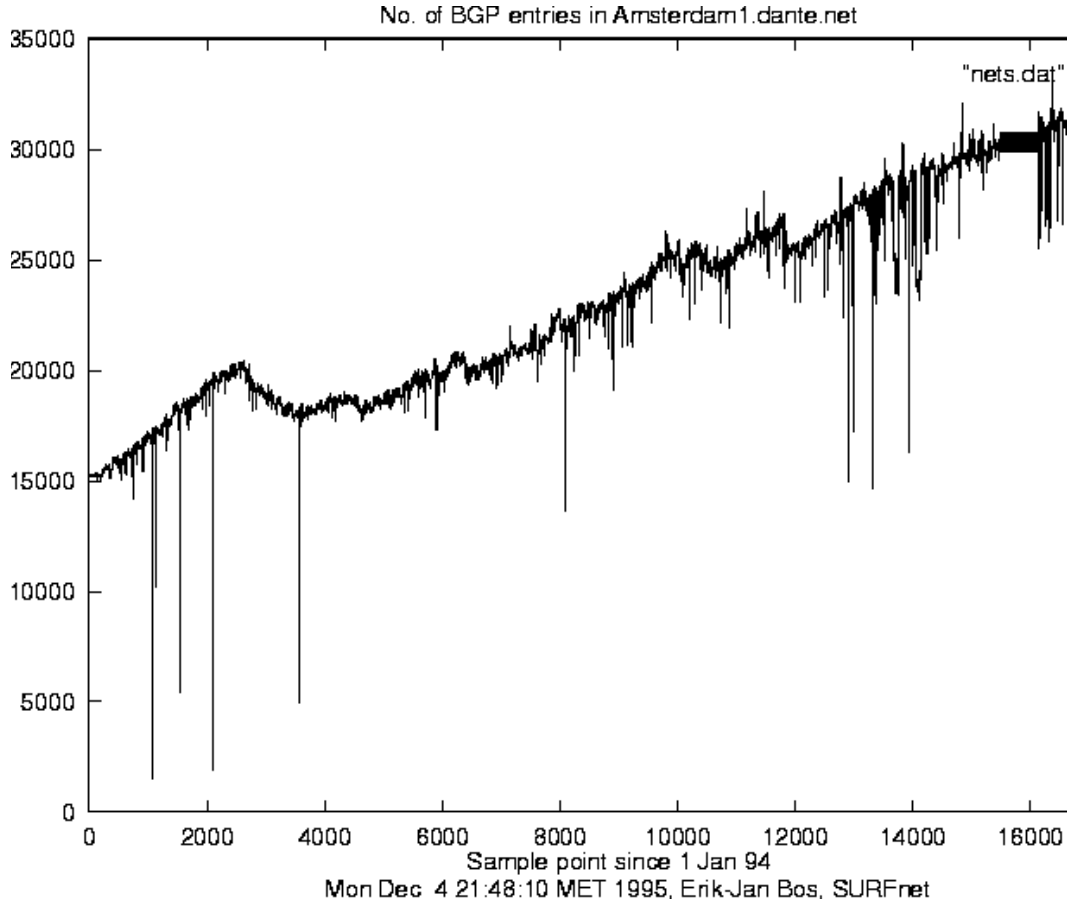

図 2.3: あるルータ上の経路情報数の推移

のネットワーク運用への支障を解決するために、インターネットでは経路の集約 (Aggregation) という手法を採用した [3]。これにより複数の経路情報を 1 つの経路情報に圧縮することができる。図 2.4に経路の集約が可能な場合を示す。従来は、図中のルータは A.B.C.*[6]および A.B.D.*の 2 つのネットワークに関する経路情報を一つずつアナウンスしていた。しかし図 2.4において、A.B.C.*および A.B.D.*の 2 つのネットワークはともに同一のルータを経由してインターネットに接続している。この 2 つのネットワーク以外に A.B.*.*というプレフィックスを持つネットワークが存在しない場合、図中のルータが A.B.*.*という経路情報をアナウンスしても問題ない。このような場合を「経路集約が可能である」という。

一方、経路の集約が不可能である場合を図 2.5に示す。この場合には同じ A.B.*.*というプレフィックスを持つ A.B.E.*というネットワークが別のルータに接続されてい

[6] この表記方法では IP アドレスを 1 バイトごとに区切って記述する。A.B.C.*ならば「1 バイト目が A, 2 バイト目が B, 3 バイト目が C, 4 バイト目は任意」という意味を示す。

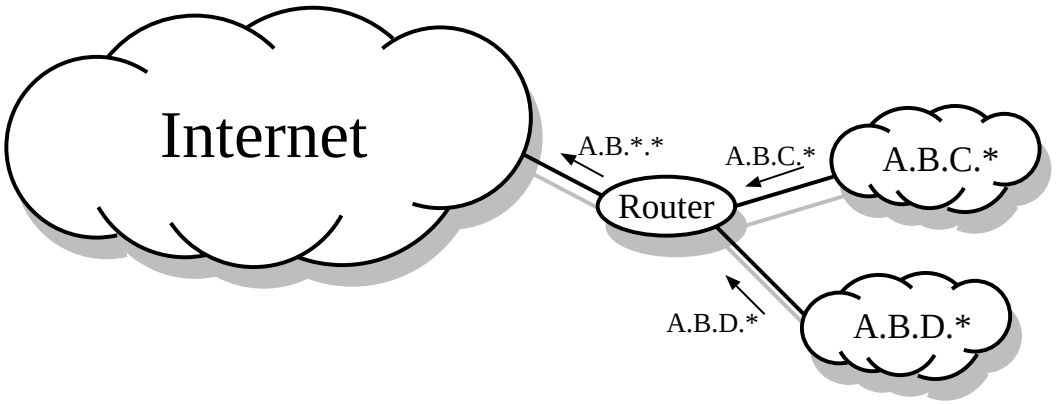

図 2.4: 経路の集約が可能な場合

る。したがって、どちらのルータも A.B.*.*という集約された経路をアナウンスすることはできない。

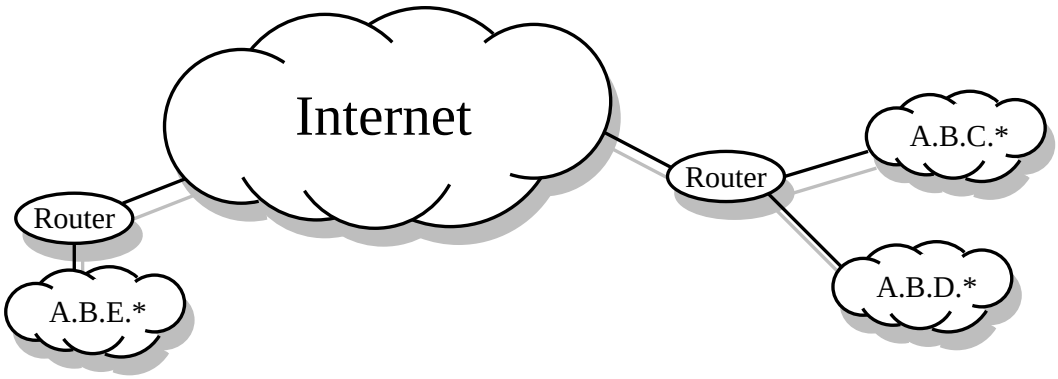

図 2.5: 経路の集約が不可能な場合

このように経路情報の集約を行なうためには、ネットワークトポロジ (ネットワークの形状) の上で隣接するネットワーク同士は、使用する IP アドレスも連続している必要がある。したがって、このような方法で既存のネットワークの経路情報を減少させるためには、本研究の目的の一つである IP アドレスの付け替えを行なう必要がある。

2.4 プロバイダベース経路制御の普及

インターネットへの接続性を提供するのがインターネットプロバイダである。プロバイダ同士では BGP[5] などの自律システム間経路制御プロトコルによって経路情報を交換するのが一般的である。前述の経路情報の集約も、プロバイダ毎に実施される

場合が多い。

各インターネットプロバイダには、料金やサービス内容などに差がある。ユーザは自らのニーズに適合したプロバイダを選択するが、何らかの事情によりプロバイダを変更しなければならない場合がある。ある組織が新たに選択したプロバイダと今まで利用していたプロバイダは、通常は不連続なネットワークアドレスを用いている。仮にその組織が以前のネットワークアドレスを利用し続けようとすると、経路の集約が不可能となる。そのためインターネットプロバイダは、組織がネットワークアドレスを用いるために別料金を請求するか、さもなければアドレス付け替えの実施を求めることになる。これもアドレス付け替え機構の実現が要望される理由の 1 つである。

2.5 家庭内 LAN の登場

TCP/IP は通信に利用する物理媒体を選ばない。例えば電話線やシリアルラインを通じて IP 接続するために、PPP[6] や SLIP[7] といったプロトコルが提供されている。

現在のパソコン用 OS や UNIX における PPP や SLIP の実装は、単体の計算機を接続することしか想定していない場合が多い。また接続時に取得することができる IP アドレスは 1 つのみである。しかし各種ネットワーク機器の低価格化によって、家庭内にネットワーク (家庭内 LAN) を構築することが特別珍しくなくなってきた。結果として、PPP や SLIP による IP 接続を用いて家庭内 LAN をインターネットへ接続したいという要求が生じている。この要求を実現するには、次の 3 つの方法がある。

- 家庭内 LAN に設定されているネットワークアドレスの経路情報を、受け口となるネットワークプロバイダが外部ネットワークに対して動的にアナウンスする。
- ネットワークプロバイダが保持するアドレスを、あらかじめ家庭内 LAN に必要な量だけ割り当てておく。
- ファイアウォールを用いて、家庭内 LAN を単一のホストに見せかける。

この 3 つの中でよく用いられるのが、3 番目のファイアウォールによって家庭内 LAN をインターネットから隠蔽する方法である [8]。ファイアウォールは本来セキュリティを実現するための技術だが、多くのファイアウォールは組織外から組織内のネットワークを隠蔽することができる。その場合、組織内の計算機は、プライベートアドレス [9] と呼ばれる IP アドレスを用いて互いに通信する。組織外の計算機と通信する場合には、ゲートウェイとなるファイアウォールマシンがプライベートアドレスをファイアウォールマシンの IP アドレスに変換する。このようにして組織内の計算機からは、通常とほとんど代わらずに外部のネットワークが利用できる。この時、外部の計算機からはファイアウォールマシンと通信を行なっているように見える。逆に外部の計算機から内部の計算機を指定して通信することができない。

1 番目の方法ではネットワークプロバイダが、不連続なネットワークの経路情報を外部ネットワークに対して配布しなければならない。現在の経路制御では IP アドレス

に基づく経路情報のフィルタリング[7]が行なわれていることも多いため、インターネットとの接続性が不完全なものになってしまう可能性がある。また 2 番目の方法を用いていても、さらに別のネットワークプロバイダに接続する場合には同様の問題が発生する。

本研究の目指すネットワークの自動設定を適用することで、家庭内 LAN の複数の計算機に対してプロバイダのアドレスを割り当てることができる。さらに第 4.2.2節に示す二重運用モデルを適用する事により、家庭内 LAN のインターネットへの接続が円滑に行なえるようになる。

2.6 アドレス付け替えに関する考察

IP アドレスの枯渇や経路情報の増加によるネットワーク運用への障害などの問題点を解決するために、インターネットではネットワークアドレスの付け替えを支援する機構が求められている。

第一に、IP アドレスの枯渇を回避するためにクラス B アドレスからクラス C アドレスへの移行が実施されているが、これにはアドレス付け替えが不可欠である。第二に、経路情報の集約を行なうにはネットワークトポロジに基づいたネットワークアドレスの割当が必要となる。ところが過去のネットワークアドレスの割当は必ずしもそのように行なわれて来なかったために、効率的な経路情報の集約を行なう際の障害となっている。これを解決するには、やはりアドレス付け替えが必要である。また例えばプロバイダを変更する際には同時にアドレスの付け替えを求められる可能性が高い。

このようにアドレス付け替えに対する要望は大きい。しかし現状ではアドレス付け替えに要する労力が大きいために、アドレス付け替えは活発に行なわれていない。したがってアドレス枯渇や経路情報の急増によるネットワーク運用への障害などの問題に対する解決策も望ましい効果をあげていない。

アドレスの付け替えによって発生する、その他の問題のいくつかを以下に示す。

- **論理通信の連続性**

  TCP/IP で通信を開始した後に IP アドレスが変更されると、それまでに行なわれていた通信を継続することができない。IP アドレスの変更が計算機の起動時にしか発生しないのであれば問題ないが、運用中に変更が発生する場合にはネットワークを利用しているアプリケーションを再起動しなければならない。それよりもアドレス付け替えの前後において連続的に通信が継続できることが望ましい。

- **名前-ネットワークアドレスの対応**

  各種アプリケーションの多くは通信の相手先を指定するのにホスト名を利用する。インターネット上のホストは通信を開始するにあたって、ホスト名から IP アドレスを得る必要がある。ホスト名から IP アドレスへの変換を行なう方法としては、決められたファイル (UNIX では/etc/hosts というファイル名) に記述さ

[7]受け入れる経路情報を制限すること。

れた変換表を参照する方法とNIS[10]やDNS[11, 12]のような名前-アドレス変換サーバを利用する方法の2通りが一般的である(現在のインターネットではDNSを用いるのが一般的である)。当然ながら計算機のIPアドレスが変更された場合には、対応するエントリを書き換えなければならない。書き換えが完了するまでの間は、名前-アドレス変換機構は誤った古いIPアドレスを返すため、ホスト名を指定して通信を開始することが不可能になる。

各計算機に存在する名前-アドレス変換ファイル(cf. /etc/hosts)を自動更新することはセキュリティの面で問題が多い。また名前-アドレス変換サーバは高速化のためにキャッシュを用いているが、分散して存在する多数のキャッシュと元データの一貫性を維持することは難しい。前述のDNSに動的更新の機構を組み込もうとする試みがあるが[8]、現状では普及していない。

- **経路制御の透過性**

  経路制御に対してネットワークアドレスの変更が影響を及ぼすか否かは、状況に依存する。例えばデフォルトルータのみを指定している場合[9]には、ネットワークアドレスの変更によりデフォルトルータのIPアドレスも変更されるため、外部ネットワークとの新規接続が不可能になる。経路制御プロセスも動作中にネットワークアドレスが変化することを考慮していない。したがって強制的にネットワークアドレスを変更しても、経路制御プロセスは元のアドレスが無効になったことを検知できず、古い無効な経路をアナウンスし続けると考えられる。これはネットワークの正常な運用にとって有害である。

  経路制御がネットワークアドレスの変更の前後で透過的に動作し続けるためには、アドレスの変更を確実に通知し、経路制御プロセスに新しい設定を行なう機構が必要である。さらにネットワーク上のあるホストは新しいアドレスに移行を終了し、別のホストは古いアドレスを使い続けているといった状況が発生しないように同期を取る必要がある。変更を行なっている最中には、IPによる通常の通信が行なえるとは限らないということについても考慮する必要がある。

2.7 Ubiquitous Computing(遍在する計算機環境)

近い将来に実現すると考えられている計算機環境として、Ubiquitous Computing(遍在する計算機環境)がある。Ubiquitous Computingの環境では身の回りのほとんどあらゆるものに計算機が内蔵されると考えられている。もちろん現在でも多くの電化製品には何らかの形で集積回路が内蔵されている。しかしこれらの集積回路が互いに通信することは、ほぼ皆無であるのに対し、Ubiquitous Computingでは、それらの計算機群が互いに通信し連携する。

---

[8] Dynamic Updates in the Domain Name System (DNS), draft-ietf-dnsind-dynDNS-05.txt というDocumentに記述されている。これはInternet-Draftと呼ばれ、6ヵ月で消去されるため参考文献には含めない。

[9] この設定は、ルータが1つしか接続されていないネットワークセグメント内のホストによく用いられる。

そのような状況において莫大な数の計算機を手動で設定することは事実上不可能である。また現在の携帯型計算機が単体で持ち運びされるのに対し、Ubiquitous Computingの世界では計算機群として持ち運ばれるようになる。例えば自動車に複数の計算機が搭載されているとする。これらは自動車の中だけでLANを構成すると共に、自動車の外部のネットワークとも通信を行なう。このような場合、移動するのはあくまで個々の計算機だとみなすことも可能だが、ネットワークが移動すると見なす方がより自然である。また自動車内部のLANは人の乗降にともなって(その人が携帯している)計算機が接続あるいは切断され、動的に構成が変化する。したがって計算機単体のネットワーク環境を自動設定するだけでなく、ネットワークを自律的かつ動的に構成し、諸設定を行なう機構が必要となっている。

第 3 章

ネットワーク設定管理手法の現状

本章では関連研究とそこで用いられているネットワーク設定管理ツールについて概観し、これらのツールがインターネットを取り巻く状況の変化に適さなくなりつつあることを示す。

3.1 Reverse Address Resolution Protocol (RARP)

Reverse Address Resolution Protocol (RARP)[13] は、MAC アドレス[1]からネットワークアドレスへの変換を行なうためのプロトコルである。RARP は Address Resolution Protocol (ARP)[14] と同一のパケットフォーマットを用いる。ネットワークアドレスが IP アドレス、MAC アドレスが Ethernet アドレスの場合のフォーマットを図 3.1に示す。RARP 要求を送信するホストは、Target Hardware Address フィールドに自身の Ethernet アドレスを設定してブロードキャストする。要求を受け取った RARP サーバは、データベースから対応する IP アドレスを検索し、Target IP address フィールドに挿入して、要求したホストに RARP 応答を送信する。

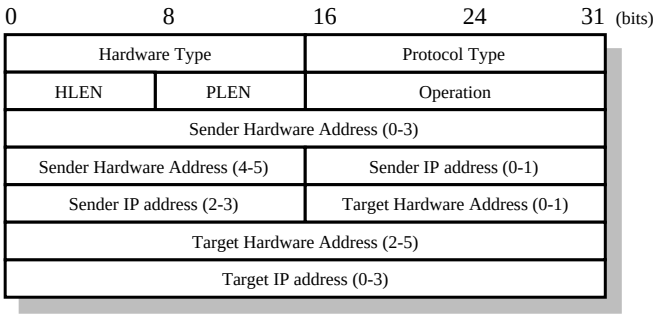

図 3.1: ARP/RARP のパケットフォーマット

[1]OSI 7 層モデルにおける物理層のアドレスの事。

RARP はディスクレスワークステーションや X 端末などの起動時に用いられるが、アドレスの取得以外の機能は持たない。RARP ではネットワークアドレスと MAC アドレスの対応をあらかじめ管理者が静的に登録しておく。このため個々の計算機を設定する代わりに、サーバで集中的に設定することが可能となる。RARP は計算機を自動的に設定したいという要求のごく一部を実現するものといえる。アドレスの動的な割り当てや、(ルータアドレスなどの) ネットワーク環境に関する情報の自動設定、アドレスの付け替え、回収などは実現されていない。

過去に RARP を拡張した Dynamic RARP (DRARP)[15] というものが提案され、ネットワークアドレスの動的な割当が試みられた。DRARP は動的なアドレス割当が可能であり、一時的に割り当てたアドレスは 1 時間経過すると回収されるが、その他の面では RARP と同じ欠点を持つ。

3.2 Network Information Protocol (NIP)

Network Information Protocol (NIP)[16] は、MIT の Athena プロジェクトで開発されたもので、アドレスの動的な割り当てを行なうことができる。NIP は Ethernet のフレームを直接読み書きすることで動作し、MAC アドレスに対して IP アドレスを割り当てる機構を持つ。

NIP サーバはアドレス割り当ての要求に対して「未使用と思われる」アドレスの集合を返す。割り当てを受けるホストは、その集合の中から定義されたアルゴリズムに基づいてアドレスを 1 つ選択する。さらに「本当にそのアドレスが未使用である」ことを確認するために、ARP 要求を送信する。もし応答があった場合には、そのアドレスは使用中であると判断してアドレスの選択からやり直す。

これを “Polling/Defense mechanism” と呼ぶが、この機構ではアドレスが絶対に重複しないことを保証できない。クライアントが選択したアドレスを使用している計算機の負荷が高かったために ARP 応答を行なうのが遅れた可能性もあるし、あるいは計算機が停止中だという可能性もある。さらにどの計算機がどのアドレスを使用しているかという情報が保持されていないため、起動するたびに以前と異なるアドレスが割り当てられる可能性がある。

NIP は計算機の自動設定と動的なアドレス割り当てという要求を部分的に解決している。取得できる情報には IP アドレスの他に、ネットマスクやブロードキャストアドレス、デフォルトルータなどがある。しかしアドレスの付け替え、回収などは実現されていない。また RARP 同様にネットワーク上の計算機を設定することのみを対象としていて、ルータを含むネットワーク全体を自動的に設定することは考慮していない。

3.3 Bootstrap Protocol (BOOTP)

Bootstrap Protocol (BOOTP)[17] は、IP アドレスとルータアドレスなどのネットワーク環境に関する情報を自動的に設定するためのプロトコルである。BOOTP も通

常はディスクレスワークステーションなどの起動時に用いられる。

BOOTP は RARP 同様に MAC アドレスと IP アドレスの対応を静的なデータベースに記述しておく。したがって割当は静的な物であり、一時的なアドレス割当や使用されなくなったアドレスの回収といった機構は含まれていない。BOOTP を利用することで IP アドレスの管理が集中して行なえるようになるが、ネットワーク管理者は依然として個々の IP アドレスを手動で割当・管理しなければならない。ただしネットワーク環境に関する情報はネットワークセグメント内で共通なため、BOOTP をもちいて自動的に設定することができる。

3.4 Point-to-Point Protocol (PPP)

PPP[6] 自体はポイント-ポイント間のデータリンクプロトコルである。PPP の構成要素の一つである Link Control Protocol (LCP) ではアドレスの割当を行なうための機構が定義されている。IP 用の LCP を Internet Protocol Control Protocol (IPCP)[18] と呼ぶ。

IPCP では、PPP のコネクションを設定する際にコネクションの両端の IP アドレスを設定する方法が規定されている。文献 [6, 18] および、それに基づいた実装を参照すると、以下のような手順でアドレスの設定が行なわれている。以下の説明では、ホスト X、ホスト Y の二つが存在すると仮定する。

1. ホスト X が Source Address(自らが使用するアドレス) を知らない場合
   (a) ホスト X は、ホスト Y にアドレスの割り当てを要求する。
   (b) ホスト Y は、Destination Address(= X 側のアドレス) が設定されている場合にはそれをホスト X に返す。
   (c) ホスト X は、それを自身のアドレスとして使用する。
2. ホスト X が Source Address(自らが使用するアドレス) を知っている場合
   (a) ホスト X は、自身の Source Address をホスト Y に通知する。
   (b) ホスト Y は、Destination Address(= X 側のアドレス) が設定されていないならば、通知されたアドレスを使用する。
   (c) 逆に Destination Address(= X 側のアドレス) が設定されていて、さらにそれがホスト X から通知されたアドレスと異なる場合には、その Destination Address を X に返す。
   (d) ホスト X は、返ってきたアドレスと自分のアドレスが違うために、接続を断念する。

このように PPP は、単純なアドレスの設定機能を持っている。しかし、実際にアドレスをどのように割り当てるかということに関する規定は存在しない。通常は受け側の

ネットワーク管理者が静的に設定しておく。ネットワーク環境に関する情報の取得設定機能は IPCP には含まれていない。ただし次に説明する DHCP[19, 20, 21] に、この機能を持たせようという試みがある。

ネットワークアドレスの回収はポイント-ポイント間の通信が切断された時点で行なわれ、次に接続した計算機に再割当される。したがってアドレスもリンクが継続している限り有効である。アドレスの付け替えについては考慮されていない。

3.5 Dynamic Host Configuration Protocol (DHCP)

DHCP[19, 20, 21] は BOOTP を拡張したプロトコルである。手動割当、動的割当、自動割当、情報通知の 4 つの割当機能を持つ。

- **動的割当**

  DHCP の最も特徴的な割り当て機能である。サーバはネットワーク管理者によってアドレスプールと呼ばれるアドレスの集合を割り当てられている。このアドレスプール中から IP アドレスを選択し、クライアントに期限つきで割り当てるのが動的割当である。この際に IP アドレスに加えて各種の設定情報を通知する。有効期限は最短 1 時間から最長約 100 年までの値をとる。この動的割り当てによりアドレスを一時的に割り当てることが可能となる。

- **自動割当**

  動的割り当て同様に、サーバがアドレスプールの中から IP アドレスの選択を行なう。ただし割り当ての有効期限は無期限である。この機構によりアドレスを自動的に割り当てることが可能となる。現在の DHCP では、使用されなくなった無期限のアドレスを回収する機構は定義されていない。

- **手動割当**

  前述の 2 つとは異なり、この割り当て機構ではあらかじめ管理者が割り当てておいたアドレスを通知するためだけに DHCP が利用される。いわゆる「静的割り当て」と呼ぶべきものであり、これは既存の BOOTP の機構とほぼ等価である。

- **情報通知**

  この機構ではアドレスの割当は行なわれない。ネットワーク環境に関する情報の通知と設定のためだけに DHCP が利用される。ネットワークアドレスは何らかの外部機構 (手動あるいは前述の IPCP など) によってあらかじめ設定されている場合に用いられる。

DHCP は IP アドレスの割当を自動的に行なうため、ネットワーク管理者や一般ユーザの作業時間が軽減される。アドレスの再利用も実現しているが、アドレスの回収の機構は実現していない。またルータを含めたネットワーク全体の設定についても考慮していない。

アドレス付け替えに関しては、DHCP を利用して実現しようという提案が Epilogue Technology 社の Lowell Gilbert によってなされており[2]、近い将来に実現されると思われる。

3.6 IPv6 Autoconfiguration

次世代 IP として開発された IPv6[4] のアドレスは 128 ビット (=16 オクテット) の長さを持つ。現在の IP アドレスは 32 ビット (=4 オクテット) であるため、IP アドレスを覚えることもたやすいし、手動で設定することも可能である。しかし IPv6 のアドレスを記憶したり、あるいは手動で誤りなく入力することは困難である。そのため IPv6 は初めからネットワークアドレスの自動設定機構を組み込むことを念頭に設計されている。以下では IPv6 のアドレス自動設定機構について述べる。

IPv6 のアドレス自動設定機構には Stateless モードと Stateful モードの 2 つのモードがある。Stateless モードは、さらに Autonomous モードと Semi-autonomous モードの二つに分類される[3]。

現在提案されている唯一の Stateful モードの自動設定機構は、DHCP を IPv6 用に拡張した DHCPv6[4]だけである。DHCPv6 でもクライアントの要求に対してサーバがアドレスを割り当てるという基本事項には変化がない。したがって、ここでは Stateless モードに限定して述べる。

2 つ Stateless モードのうち、Autonomous モードはローカルセグメント内でのみ通用するアドレスを設定するためのものである。Autonomous モードでは、例えば Ethernet アドレス 6 オクテットと、あらかじめ定められている 10 オクテットのプレフィックスを繋げることでアドレスが決定される。Semi-autonomous モードでも、ルータが Advertise するプレフィックスと、例えば Ethernet アドレス 6 オクテットを繋げることでアドレスが決定される。Semi-autonomous モードで設定されるアドレスの有効範囲はルータの Advertise するプレフィックスによって決定される。

Semi-autonomous モード用に Advertise するメッセージには、アドレス付け替えのためのフラグが設けられている。Deprecate フラグがセットされた Advertise メッセージを受信したクライアントは、その Advertise で指定されたプレフィックスを持つアドレスの使用を可能な限り行なわないようにする。例えば新しくコネクションを開く時には、可能な限り別のアドレスを用いるようにする。

このように IPv6 には最初からアドレスの自動設定機構とアドレスの付け替え機構が組み込まれている。ただしネットワーク単位の自動設定機構などは組み込まれていない。

---

[2]未発表の Document であるため、参考文献には示さない。ftp://ftp.epilogue.com/pub/lowell/から入手可能である。

[3]Stateless モードに関しては draft-ietf-addrconf-ipv6-auto-07.txt に記述されている。

[4]draft-ietf-dhc-dhcpv6-03.txt として入手可能である。

第 4 章

階層化モデルによるネットワーク自動設定機構の提案

本章では初めに「大規模ネットワークの設定および管理の自動化」「ネットワークアドレスの付け替え」「ネットワークアドレスの効率的利用の実現」の 3 つの目的を実現する新ネットワーク設定管理システムについて、設計目標の点から論じる。次に階層構造を持ったネットワーク設定管理モデルが、これらの目的に適していることを明らかにし、システム全体の概要設計について述べる。

4.1 設計目標

新しいネットワーク設定管理システムの設計目標として、以下の 5 項目を設定した。

1. **負荷分散を実現し、大規模性を確保する**

   キャンパスネットワークあるいは社内ネットワークのようなネットワークでの利用を目指す場合に大規模性が重要となる。また Ubiquitous Computing の世界では計算機の総数は飛躍的に増大する。したがってサーバ単体の性能を向上させるだけでなく、システム全体として負荷分散を実現する機構が必要である。

2. **ネットワーク管理者が直接操作するサーバを少数にする**

   負荷分散を行なうため複数のサーバを設けることで、管理者の負担が増大するのは望ましくない。管理者が設定を行なうサーバは 1 つないし、ごく少数に抑えるべきである。同時に、個々のネットワークや個々のホストに対する設定は管理者に極力依存しないようにする。

3. **意図的なアドレス付け替えを支援する**[1]

   インターネットにおけるさまざまな問題を解決するために、アドレスの付け替えが必要なことは既に述べた。それにも関わらず現在までにアドレスの付け替え

[1] アドレスの付け替えには管理者が意図的に行なうものと、何らかの理由で意図せずに発生してしまうものの 2 通りがある。

を支援する機構はほとんど存在せず、提案されているものは機能的に不十分である。ルータを含めたネットワーク全体のアドレス付け替えを考慮しなければならない。

4. **ネットワーク管理ポリシとメカニズムを分離する**

   ネットワークによって管理者のポリシはさまざまに異なる。「アドレスの利用率を向上させることが最優先」である場合もあれば、「経路情報が集約可能である」あるいは「意図しないアドレス付け替えを最小限に抑える」ことが最優先事項となる場合も存在する。多様なネットワーク管理ポリシに柔軟に対応するためには、本システムのメカニズムをポリシ独立なものにする必要がある。

5. **モジュール性を高めて汎用性を確保する**

   アドレス付け替えやネットワークの自動設定は次世代インターネットプロトコルである IPv6 でも重要となる。さらに末端の計算機を動的に設定するプロトコルとして、DHCP や IPv6 の Autoconfiguration などが普及しつつある。こうした既存プロトコルを利用することができるようにするためには、本システムをモジュール化して設計し汎用性を高めることが必要である。

4.2 新しいネットワーク設定管理モデル

ここでは本研究が対象とするネットワークについて前提条件を定義し、さらに前節に示した設計目標を実現するための新しいネットワーク設定管理モデルについて考察する。

4.2.1 ネットワークおよび計算機に関する前提

本システムが対象とするネットワークでは、単純な経路制御のみが用いられていることを想定する。単純な経路制御の例としては、default route のみ、RIP、単一自律システム内の OSPF[22] などがある。ネットワークアドレスに強く依存した設定、すなわちネットワークアドレスによるフィルタリングなどが行なわれているような状況については取り扱わない。

さらに本システムが対象とするネットワークは、単一の管理ポリシの元に運営されていると仮定する。すなわちネットワークの各部ごとにポリシが異なってはならない。現実のネットワークでは社内の部署ごとにネットワークを切り分け、別々のポリシで運用することを求められる場合があるが、そのようなケースは取り扱わない。

効率的なアドレス利用を実現するために、全てのルータは VLSM(Variable Length Subnet Mask)[3, 23] をサポートしているものとする。VLSM をサポートするルータでは、そのルータの持つ複数のネットワークインタフェースでネットマスクの長さが違う場合でも適切な経路制御を行なうことができる。VLSM が必要なネットワークの例を図 4.1に示す。

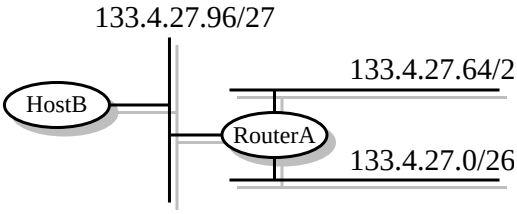

図 4.1: VLSM の必要な例

図のホストBがVLSM対応である場合、経路表にはネットマスクも保存されている。したがって例えば 133.4.27.34 宛のパケットを送信したい場合には、133.4.27.0/26[2]という経路に一致するのでルータ A に転送すれば良いことが分かる。逆にホスト B が VLSM 対応でない場合、その経路表にはネットマスクが保存されていない。そのため 133.4.27.34 にパケットを送信しようとした場合に、以下のように処理が行なわれる。

1. 133.4.27.34 は、自身と同じネットワークに属している
2. 自身のネットワークインタフェースは 27 ビットのネットマスクを持つ
3. したがって 133.4.27.34 というアドレスも 133.4.27.32/27 [3]というネットワークに属すると仮定する
4. 経路表には 133.4.27.32(ネットマスクなし) というエントリが存在しない
5. 経路が見つからないのでデフォルトルータに向けて送信する。
6. デフォルトルータがルータ A であれば問題ないが、ルータ A 以外のルータ C がデフォルトルータだとすると、そのルータ C にパケットが転送される。
7. デフォルトルータ C が VLSM をサポートしていない場合にはパケットが正しく届かない。

このように VLSM をサポートしていないホストやルータが存在すると、通信に支障が発生する場合がある。これを確実に回避するためには、最も大きなサブネット (最も短いネットマスクをもつネットワーク) に全てのネットワークのネットマスクを合わせなければならないため、無駄な IP アドレスの割当が増えるという望ましくない状況が発生する。

本システムはサーバ-クライアントモデルに基づくものとする。自明であるが、アドレス付け替えを実現するためにはネットワーク上の全てのホストでクライアントが動作しなければならない。現実には必ずしもそういった理想的な状態であるとは限らな

[2] 133.4.27.0〜133.4.27.63 までの範囲のアドレス
[3] 133.4.27.32〜133.4.27.63 の範囲のアドレス

いが、本システムでは問題を簡潔にするために「対象となる論理ネットワーク[4]上の全てのホストでクライアントが動作している」と仮定する。この仮定に加えて、サーバ群は連続して存在しなければならないものとする。すなわち本システムでは、図 4.2(a) のようにサーバが連続して存在する場合を取り扱う。図 4.2(b) のようにサーバが不連続に設置されていてはならない[5]。

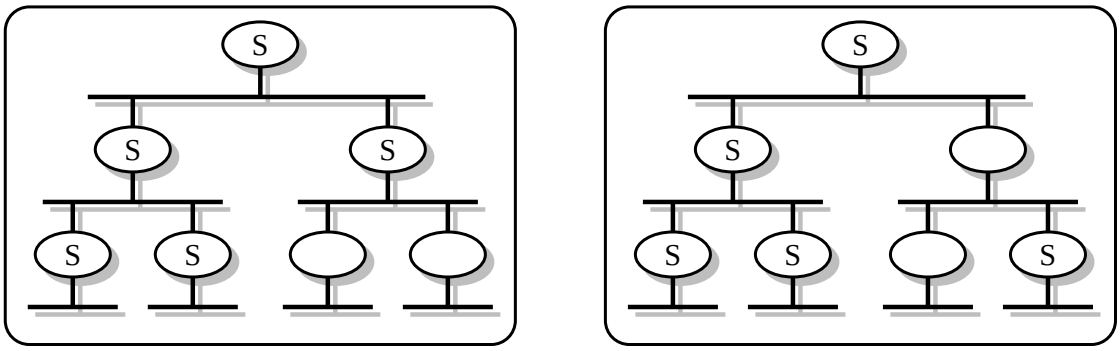

図 4.2: 連続/不連続に設置されたサーバ群

4.2.2 ネットワークの二重運用モデル

ここで、前節において「対象となる**論理ネットワーク**上の全てのホストでクライアントが動作する」と明記した理由を述べる。

現在のルータの多くは 1 つのネットワークインタフェースに複数のネットワークアドレスを設定することができる。したがって 1 つの物理ネットワークを複数の論理ネットワークとして取り扱うことができる。これを適用して、1 つの論理ネットワークは従来通りのネットワークとして、もう 1 つの論理ネットワークを本システムを使って運用するネットワークとして取り扱うことができる。そのようなネットワークの二重運用の例を図 4.3に示す。図中の NetA として表したネットワークが本システムを利用して運用するネットワークであり、NetB として表したネットワークは従来のネットワークである。NetB のネットワークと NetA のネットワークが重なっている部位は、そのセグメントが本システムに移行中であることを示す。NetB のネットワークと NetA のネットワークは論理的には別のネットワークであるため、本システムに未対応のホストは NetB のネットワークに接続しておけば NetA のネットワークには影響を及ぼさない。このようにすれば本システムへの移行を緩やかに行なうことができる。

ルータだけでなくネットワーク上の全ての計算機は一つのネットワークインタフェースに複数のネットワークアドレスを設定することができるものとする。ネットワーク

[4]「論理ネットワーク」については次の第 4.2.2節で述べる。

[5]図中の S はサーバを意味する。何も記述されてないホストは単なるルータであり、本システムのサーバは動作していない。

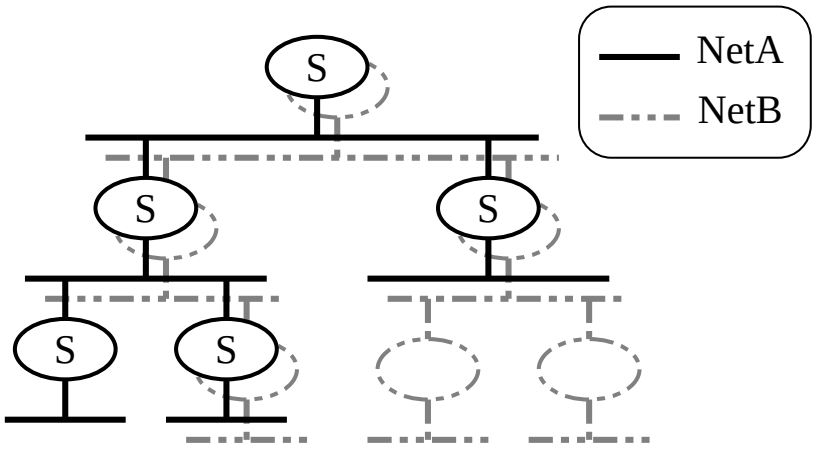

図 4.3: ネットワークの二重運用例

アドレスの付け替えを行なう場合に、ネットワークインタフェースにつき 1 つしかアドレスが設定できないと、ある瞬間を境に完全にアドレスが付け替わってしまう。そうすると論理通信路が切断されてしまうため、ホストは複数ネットワークアドレスを次のように使用して緩やかなアドレス付け替えを実現する。

1. 古いアドレスはそのままで、新しいアドレスを設定する
2. 古いアドレスを宛先に持つパケットも受信する
3. 新しく通信を開始する場合は、新しいアドレスを始点として使用する
4. 可能な限り長い時間 (古いアドレスを使った通信が限りなく少なくなるまで) が経過した後に、古いアドレスは削除する

このように徐々に新しいアドレスに切替えていくことで、ユーザの視点からはアドレス付け替えの前後にわたって通信が連続的に行なわれたように見せることが可能である。

4.2.3 階層構造モデルの導入

新しいネットワーク設定管理モデルでは、管理対象となるネットワーク[6]上の全てのルータ上でサーバが動作する。通常の IP 経路制御はネットワークアドレスの変化に対応していないが、全てのルータ上でサーバが動作するために、独自の中継機構を利用することができる。したがってネットワークアドレスの付け替えにも影響を受けないし、マシン立ち上げ時のようにネットワークアドレスを取得する前であっても動作すると考えられる。

[6] サーバは連続して設置されていなければならないため、管理対象となるネットワークも連続的に存在する。

新モデルでは、サーバ群が自律的にネットワークトポロジに基づいた階層構造 (Spanning Tree) を構築する。階層化の例を図 4.4に示す。仮に図 4.4(a) のようなネットワークが存在する場合に、各ルータ上に存在するサーバ群は図 4.4(b) のような階層構造を構成する。

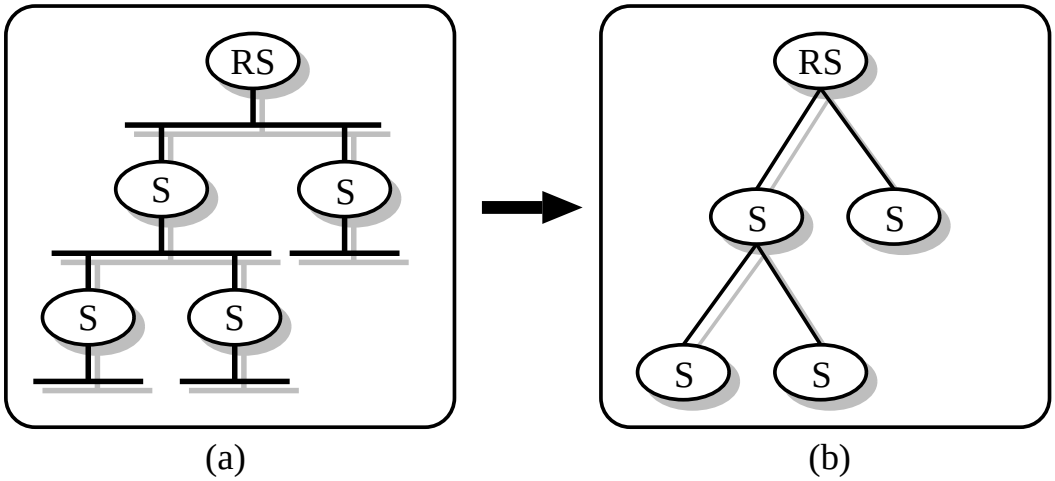

図 4.4: ネットワーク階層化の一例

階層構造を導入する理由は、第一に大規模性が確保しやすいためである。階層化は大規模性を確保する手段として DNS などでも用いられている。サーバはクライアントからの要求を受け取った時に、それを自分自身で処理できない場合にのみ、上位サーバ (階層的に上位に位置するサーバ) ないし下位サーバ (階層的に下位に位置するサーバ) へ要求を転送する。したがって階層化は負荷分散の効果も持っている。

さらに階層化することにより、ネットワーク管理者が管理するサーバを階層構造のトップに位置するサーバ (木構造の根に位置するサーバのことで、以後ルートサーバと呼ぶ) だけに限定することが容易になる。もちろん階層構造でなくても、サーバ群の集中管理を実現することは可能である。しかし、その場合にはネットワークのトポロジを集中管理用サーバが把握する必要があり、システムを複雑なものにする。

第 5 章

ネットワーク自動設定機構の設計

本章では階層化ネットワーク設定管理システムの詳細設計を行なう。本システムは「サーバ木構成部」「ネットワーク設定部」「ホスト設定部」の 3 つに分割して設計する。各部の設計にあたってアドレス付け替えの支援について考慮したが、これについては各部の設計について述べた後に論じる。本章の最後で、さまざまなネットワーク管理ポリシについて考察する。

5.1 サーバ木構成部の設計

サーバの階層構造は管理者が静的に設定することも可能だが、柔軟性が低いうえに管理者が誤った設定を行なう可能性がある。本システムの他の部分は階層構造を基準に動作するよう設計されているため、誤った設定はシステム全体に影響を及ぼす可能性が高い。また耐故障性を考慮する場合に静的な設定では素早く対応できない。したがってサーバ群が自律的に階層構造を決定するためサーバ木構成部が必要となる。ただしルートサーバを自動的に決定する機構を組み込むことは考えず、管理者が明示的に指定するものとする。本論文では 1 つの論理サブネットを複数のサーバが管理することは想定しないため、複数のサーバが接続されている場合に調停を行なって 1 つだけサーバを選定する機能を含めなければならない。本節では最初にサーバ木構成部の設計目標を明らかにし、次に詳細な設計を行なう。ただしアドレス付け替えの方法に関しては、第5.4章で詳しく論じるものとする。

5.1.1 設計目標

以下の 6項目を設計目標とする。

1. **IP ルーティングや IP アドレスに依存しない**

   サーバ群が階層構造を自律的に構成するために、通常の IP アドレスや IP ルーティングを利用する方法がある。例えば次のような手法を用いることができる。

   - ルートサーバの IP アドレスを何らかの手段で通知する

- その IP アドレスに対応する経路制御表を検索する
- 宛先となるルータを上位サーバ[1](の候補) とみなす

しかしこの方法には「本システムにおいては IP アドレスは不変ではない」および「システムの起動時には IP アドレスも経路制御表も存在しない」という 2 つの問題点がある。したがって IP アドレスや IP ルーティングに依存せず、独立に階層構造を決定できなければならない。

2. **階層構造を素早く収束させることが可能である**

   サーバ群が起動してから階層構造が決定されるまでの時間は短い方が望ましい。したがってネットワークの構成を可能な限り素早く把握できる必要がある。

3. **階層構造の変更は少なくする**

   収束性が高くなると、ネットワーク上の些細な変動にも素早く反応してしまう。しかし階層構造が変化するとアドレスの付け替えなどを引き起こす可能性がある。(意図しない) アドレスの付け替えはコストが大きいので可能な限り回避しなければならない。ネットワーク上に発生する変化を素早く察知することは必要だが、不用意に階層構造を変更することは避けなければならない。

4. **少ない計算量で動作する**

   階層構造を決定するために必要な計算量は少ない方が望ましい。

5. **任意のサーバの起動が遅延しても矛盾しない**

   階層構造を決定し、本システムが管理を開始した後に、新しくサーバが追加される場合もある。そのような場合に、既に収束している階層構造には可能な限り影響を与えないようにする。

6. **できるかぎり簡潔な機構にする**

   可能な限り簡潔に設計し容易に実装できるようにする。

5.1.2 設計

はじめに設計目標に対する解決方針の概略を示す。

1. **IP ルーティングや IP アドレスに依存しない**

   独自の経路制御プロトコルを設計し、経路情報に基づいて階層構造を決定する。今回は Distance-Vector 型の経路制御方式を採用する。

2. **階層構造を素早く収束させることが可能である**

   Distance-Vector 型のアルゴリズムに Triggered Update の手法を取り入れることで階層構造の素早い収束を可能にする。

[1]階層構造の上位にあたるサーバのこと。木構造の根に近い側のサーバを意味する。

3. **階層構造の変更は少なくする**

   収束途中の経路情報が階層構造に影響を及ぼさないように経路情報の更新を一時経路表で行ない、経路情報が安定してから本経路表に複製する。また階層構造の決定後に新たにサーバが起動しても階層構造が変動しないようにDesignatedメッセージというものを交換する。

4. **少ない計算量で動作する**

   他の経路制御方式として、収束の早いLink-State型があるが、計算量の少なさを重視してDistance-Vector型を採用する。

5. **任意のサーバの起動が遅延しても矛盾しない**

   階層構造の決定にサーバIDの大小比較を取り入れることで、起動時に多少の遅延が発生しても影響が出ないようにする。

6. **できるかぎり簡潔な機構にする**

   アルゴリズムや実装の簡単なDistance-Vector型のアルゴリズムを採用する。

本節の残りでは、解決方針および設計の詳細について述べる。

サーバ群の階層構造を作るためには、ルートサーバまでのホップ数を利用する方法と、ネットワークの接続状況に関する地図を作成して、その地図から階層構造を決定する方法が考えられる。ただし両方式ともに通常のIP経路制御によるホップ数やネットワーク地図は利用できないため、本システム独自に決定する必要がある。前者はRIP[24]などのDistance-Vector型(あるいはBellman-Ford型)経路制御に相当し、後者はOSPF[22]などのLink-State型経路制御に相当する。

一般にはLink-State型経路制御方式の方が収束が早いといわれているが、Distance-Vector型アルゴリズムも変更を加えることで収束を十分に早くすることができる。また実装はDistance-Vector型アルゴリズムの方が容易である。さらにLink-State型はネットワーク上の接続がわずかに変化しただけで大量の演算が必要となってしまう。これらの理由から今回はDistance-Vector型のアルゴリズムを採用する。

Distance-Vector型経路制御の基本的な動作を以下に示す。

1. すべてのルータの経路表には自分自身がコスト0で設定されている。

2. 自分自身以外のすべての宛先はコスト“無限”となっている。

3. ルータは一定時間ごとに自分自身の経路表を近接ルータに通知する。

4. 近接ルータから経路情報を受け取ったら、その近接ルータまでのコストと経路情報に含まれるコストを合計する。合計がもし今までよりも小さなコストになった場合には、経路表を(宛先, 近接ルータ, 合計コスト)のように更新する。

これをそのままサーバ木構成部に適用することができる。その場合は、ルートサーバを宛先にもつ経路情報だけを管理すれば良いのでトラフィックは少なく抑えることができる。しかしルートサーバを第 0 階層とする木構造が n 階層あり、経路情報のアナウンスを m 秒に 1 回行なうと仮定すると、最下層まで情報が到達するのに mn 秒程度の時間を要する。このままでは収束速度が不十分であるため、Triggered Update に類似した手法を取り入れる。Triggered Update とは、経路が変化した場合に直ちに近接ルータへ更新情報を送る方式である。これにより階層が深くなってもルートサーバから最下層まで伝達するのに必要な時間は実用に耐える程度に改善できる。

サーバ木構成のアルゴリズムを以下に示す。IP アドレスや通常の IP ルーティングには依存しないため、以下「経路表」や「経路情報」は本システム独自のものを意味し、通常の IP ルーティングのものとは無関係とする。

1. ルートサーバは初期状態で;

   宛先:ルートサーバ ID, 次ホップ:ルートサーバ ID, ホップ数:0

   という経路表を保持している。それ以外のサーバの経路表は初期状態では空である。

2. ルートサーバは全てのネットワークインタフェースから定期的に次のような経路情報を流す:

   宛先:ルートサーバ ID, Sender:ルートサーバ ID, ホップ数:0, Sequence

   ただし際限なく経路情報が伝播していかないように、ホップ数が上限値を超えた場合には、近接サーバに経路情報を流さない。また管理者があらかじめ指定したネットワークインタフェースからは経路情報を流さない。なお本システムではホップ数の仮の上限値を 16 とした。

   Sequence は、経路情報のループを防ぐためにルートサーバによって管理される連続番号である。ルートサーバは経路情報を送信する度に Sequece 番号を 1 ずつ増加させる。

3. 階層構造の変動を少なくするために、ルート以外のサーバは一時経路表と本経路表の 2 つの経路表を管理する。また収束を早くするための Triggered Update の手法に基づき、各サーバは以下のように動作する。

   (a) 現在の一時経路表に記録されているものよりも大きい Sequece 番号を持つ経路情報を受け取った場合には、一時経路表に次のような経路を記録し;

   宛先:ルートサーバ ID, 次ホップ:Sender ID, ホップ数:受信ホップ数 m+1, Sequence

   さらに全てのネットワークインタフェースから次のような経路情報を流す:

宛先:ルートサーバ ID，Sender:自分の ID，ホップ数:m+1，Sequence

(b) 現在の一時経路表と等しい Sequence の経路情報を受け取った場合、一時経路表のホップ数と比較する。「より小さいホップ数」の場合には一時経路表を更新し、再度全てのネットワークインタフェースから経路情報を流す。「同じホップ数, 異なる次ホップ」の場合は、ID を比較して小さい方を経路表に登録する。ただし近接ルータには通知しない。これは階層構造に影響を与えないようにするために、サーバの起動に多少の遅延が発生しても常に特定のサーバが経路表に登録されるようにするためである。「より大きなホップ数」の場合は、何ら処理を行なわない。

ID は同じ木構造に属するサーバ群の中でユニークであれば、どのようなものでも構わない。例えば Ethernet アドレスを利用できる。大小関係の比較方法はここでは定義しない。

(c) 経路の更新は常に一時経路表で行ない、一定時間が経過した後に (一時経路表が収束したとみなして) 本経路表に複製する。このように Triggered Update を導入することで収束を早くするとともに、収束途中の経路が階層構造の変動を引き起こさないようにする。

4. 最終的に全てのサーバが、自身の上位サーバとなるのに適したサーバを決定できる。

5. その上位サーバの候補に対して UpperRequest メッセージを送信し、上位サーバになるように要求する。UpperRequest を受け取ったサーバは UpperAck を返し下位サーバとして登録する。この段階で初めて下位サーバの存在を検知できる。

経路表から求められる上位サーバと現在の上位サーバが異なる状態が長期間継続した場合 - 例えば上位サーバの故障や、よりホップ数の小さな上位サーバが発見された場合 - には、上位サーバを変更する必要がある。しかし上位サーバ変更は影響が大きいので、頻繁に行なうべきではない。どのような基準、どの時点でサーバ変更を行なうかを決定することは今後の課題の 1 つである。

前述のようにして決定された木構造を元に、あるサーバは上位サーバ側のネットワークインタフェース (および明示的に指定したネットワークインタフェース) を除く、全てのネットワークインタフェースを管理する。ただし、複数のサーバが同一セグメントを管理するという状態が発生しないように、次のような処理を行なう。あるネットワークセグメントを管理するサーバのことを Designated サーバと呼ぶ。

1. 1〜3 秒までの乱数をふり、タイマーをセットする。

2. タイムアウトするまで Designate メッセージの受信待ちをする。Designate メッセージには送信するサーバからルートまでのホップ数を含める。「自身のホップ数より小さい」あるいは「同ホップ数かつ、より小さいサーバ ID」を含む Designate メッセージを受け取った場合には、そのセグメントの管理は他のサーバに頼るも

のとし、これ以上の処理は行なわない。同ホップ数の場合に小さいサーバ ID を持つものを選択するのは、サーバの起動時刻に多少の遅延があっても、上位サーバの選択に影響を及ぼさないようにするためである。

3. タイムアウトまでに Designate メッセージを受け取らなかった場合や、「自身のホップ数よりも大きい」あるいは「同ホップ数かつ、より大きいサーバ ID」を含む Designate メッセージを受け取った場合には、Designate メッセージを送信し、送信回数を 1 つ加算する。さらに N 秒後にタイマーをセットする。送信回数が M 回未満の場合には 2へ戻る[2]。

4. タイムアウトまでに「自身のホップ数より小さい」あるいは「同ホップ数かつ、より小さいサーバ ID」を含む Designate メッセージを受け取った場合には、そのセグメントの管理は他のサーバに頼るものとする。受信できなかった場合には、そのセグメントの管理を行なうものとする。

5. あるネットワークを管理することが決定した後に Designate メッセージを受け取った場合には、ホップ数を 0 にして送信する。これは、あとからより小さな ID を持つサーバが起動しても、階層構造に変更が起きないようにするためである。

最後にサーバ木構成部で用いるメッセージフォーマット (図 5.1) と各フィールドの説明 (表 5.1) を示す。

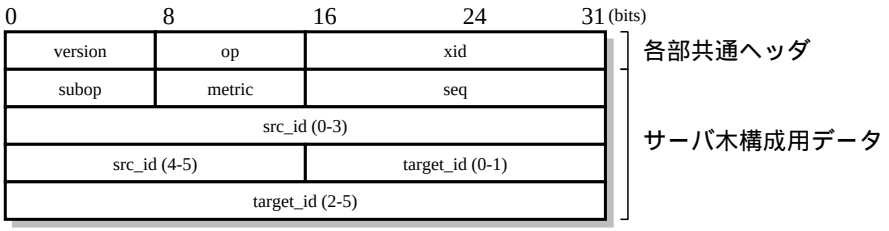

図 5.1: サーバ木構成メッセージのフォーマット (Ethernet の場合)

5.2 ネットワーク設定部の設計

サーバ木構成部によって決定した階層構造に基づき、ネットワークアドレスの集合の授受を行なってネットワークを設定するのがネットワーク設定部である。本節でも設計目標を論じてから設計について述べる。以後、連続したアドレスの集合をアドレスバルクと呼び、あるサーバが管理するアドレスバルクの集合をアドレスプールと呼ぶ。

[2] M および N の値は未定義である。

表 5.1: サーバ木構成メッセージの各フィールド

名称	用途
version	プロトコルのバージョンを指定する。現在のバージョンは 1。
op	サーバ木構成メッセージであることを示す 0 を入れる。
xid	トランザクション ID。サーバ間でメッセージを一意に識別する。
subop	サーバ木構成メッセージをさらに分類する。(0 = 経路情報交換用, 1 = 上位サーバ Request 用, 2 = 上位サーバ Ack 用, 3 = Designate 用)
metric	ルートサーバからのホップ数を示す。subop が 1, 2 の場合は 0(無効)。
seq	ルートサーバにより挿入される順序番号。subop が 0 以外の時は 0(無効)。
src_id	メッセージを送信したサーバの ID。図 5.1では、Ethernet アドレス (6 オクテット) を例として用いている。
target_id	subop が 0 の場合はルートサーバの ID、subop が 1, 2 の場合には上位サーバの ID、subop が 3 の場合は 0(無効) が入る。

5.2.1 設計目標

設計目標は、以下の 5項目である。ネットワーク設定部は本システムの中核部分であるために、本システム全体の目標と重複する項目が多い。

1. **ネットワーク管理ポリシとプロトコルを分離する**

   さまざまなネットワーク管理ポリシに対応するために、ネットワーク設定に必要なプロトコルについてのみ設計する。プロトコルをどのように用いてポリシを実現するかということについては本章では取り扱わない。

2. **ネットワーク管理者が操作するのはルートサーバのみとする**

3. **アドレス付け替えを実現する**

   アドレス付け替えには意図的なものと意図しないものの 2 通りがある。本システム全体としては意図するアドレス付け替えは支援し、意図しないアドレス付け替えは極力排除するべきである。しかし状況に応じては意図しないアドレス付け替えも実施しなければならない。したがって、意図したものかそうでないかに関わらずネットワークアドレスの付け替えの機能を持つ必要がある。

4. **メッセージの不必要な送信を防ぐ**

   あるサーバから割り当てを拒否された直後のアドレスバルク (アドレスの集合のこと) を再度割当要求したり、返却を拒否された直後のアドレスバルクを再度返却要求したりすることのないようにする。

5. **アドレスバルクの強制的な回収をサポートする**

   アドレス付け替えの最後の段階では古いアドレスの回収が必要となる。またアドレス付け替えとは独立に、ネットワーク管理の一環としてアドレスの回収が必要な場合もある。そのような状況での強制回収もサポートしなければならない。

5.2.2 設計

はじめに設計目標に対する解決方針の概略を示す。

1. **ネットワーク管理ポリシとプロトコルを分離する**

   アドレスプールの状態、アドレスプールへの操作、アドレスプールの状態遷移、使用するメッセージなどについてのみ設計する。

2. **ネットワーク管理者が操作するのはルートサーバのみとする**

   アドレスプールの状態遷移では、アドレスプールの状態と受信メッセージに基づいて出力 (Action) を定義する。その際にルートサーバに対しての要求は、適切な下位サーバまで伝播するように出力 (Action) を定義する。

3. **アドレス付け替えを実現する**

   アドレスプールの状態に「返却中」であることを示す状態を定義する。アドレスの返却によってアドレスの不足が発生し、下位サーバは新しいアドレスの割当を要求する。返却中のアドレスは割当の対象とならないために、アドレスの付け替えが実現できる。

4. **メッセージの不必要な送信を防ぐ**

   一旦 Nack を受け取った領域に対しては、しばらく Request を送ることがないように状態遷移を設計する。また要求を直ちに処理できない場合には Pending メッセージで応答する。

5. **アドレスバルクの強制的な回収をサポートする**

   返却要求メッセージに強制フラグを設ける。さらに上位サーバは一定時間経過すれば返却メッセージが届いていなくてもアドレスバルクを回収したものとして扱う。

本節の残りでは、多様なネットワーク管理ポリシに対応するためにポリシ独立なプロトコルを設計する。そのため、ここでは使用するメッセージとアドレスプールの状態に関してのみ定義する。

ネットワーク設定部で用いるメッセージ

ネットワーク設定部では「上位サーバから下位サーバへのアドレスバルク割当」と「下位サーバから上位サーバへのアドレスバルク返却」の 2 つの操作を基本として構成する。メッセージはアドレスの割当/返却を要求する側が再送の責任を持つ。ネットワーク設定部で用いるメッセージの一覧を表 5.2 に示す。またメッセージフォーマットを図 5.2に、各フィールドの説明を表 5.3に示す。

表 5.2: ネットワーク設定部で用いるメッセージの種類

名称	用途
AllocRequest	下位サーバから上位サーバへの割当要求
ReleaseRequest	上位サーバから下位サーバへの返却要求
Alloc	上位サーバから下位サーバへの割当
Release	下位サーバから上位サーバへの返却
AllocNack	下位サーバからの割当要求に対する拒否
ReleaseNack	上位サーバからの返却要求に対する拒否
AllocPending	AllocRequest を上位サーバが受信し、処理中であるという通知。受信した下位サーバは、一定時間後に AllocRequest を再送する。
ReleasePending	ReleaseRequest を下位サーバが受信し、処理中であるという通知。受信した上位サーバは、一定時間後に ReleaseRequest を再送する。

すべてのネットワーク管理ポリシは、これらのメッセージをどのように用いるかによって決定される。Request, Ack, Nack という一般的な操作に加えて、Pending という概念を導入したのは、以下の理由による:

> ネットワーク管理者が操作するのはルートサーバのみであるために、例えばあるアドレス領域の返却を求めた場合には、上位サーバから下位サーバへ要求が伝播していくようにしなければならない。階層が深くなると各サーバでの処理が積み重なって、要求の伝播にも時間がかかる。そのため「Request を受信したが下位サーバの処理が終わるまで待機」させる必要がある。

各サーバの基本動作を補足説明すると次のようになる。

- アドレスが不足すると上位サーバに割当を要求する。あるいは下位サーバに返却を要求する。
- 割当要求を受け取った場合、割当可能なら割り当てる。不可能なら拒否する。
- 割当要求を処理するのに時間がかかる (例えばさらに上位サーバに割当を要求しなければならない) 場合は、Pending メッセージを返す。

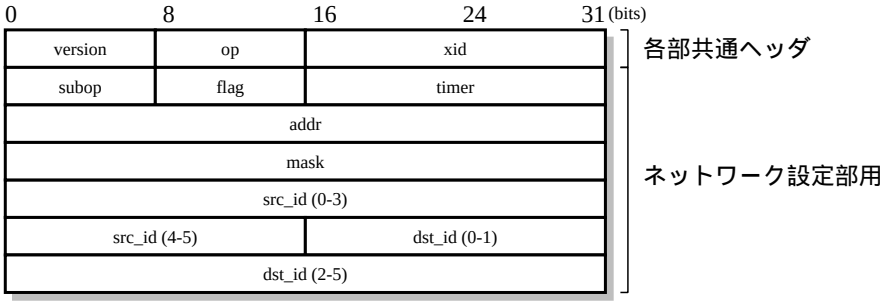

図 5.2: ネットワーク設定メッセージのフォーマット (Ethernet の場合)

表 5.3: ネットワーク設定メッセージの各フィールド

名称	用途
version	プロトコルのバージョンを指定する。現在のバージョンは 1。
op	ネットワーク設定メッセージであることを示す 1 を入れる。
xid	トランザクション ID。サーバ間でメッセージを一意に識別する。
subop	ネットワーク設定メッセージをさらに分類する。(0:AllocRequest, 1:ReleaseRequest, 2:Alloc, 3:Release, 4:AllocNack, 5:ReleaseNack, 6:AllocPending, 7:ReleasePending)
flag	最下位ビットは強制フラグとして用いる。残りは将来の拡張用。
timer	サーバは Pending を受け取ってから timer 秒後にメッセージを再送する。
addr	次の mask と組で、対象アドレスバルクを指定する
mask	addr フィールド参照
src_id	メッセージを送信したサーバの ID。図 5.2では、Ethernet アドレス (6 オクテット) を例として用いている。
dst_id	メッセージの宛先となるサーバの ID。図 5.2では、Ethernet アドレス (6 オクテット) を例として用いている。

- 返却要求を受け取った場合、返却可能なら返却する。不可能なら拒否する。
- 返却要求を処理するのに時間がかかる (例えばさらに下位サーバに返却を要求しなければならない) 場合は、Pending メッセージを返す。

アドレスが不足していると判定する基準や、割当要求と返却要求のどちらを行なうかを決定する基準、それらの要求を拒絶する基準などはポリシに深く関わるため、本章では定義しない。しかし例えば下位サーバに割り当ててあるアドレスバルクが返却された場合にどういった動作を行なうかについては、ポリシとはほとんど無関係である。そこで本節の残りでは、アドレスバルクの状態と状態遷移を定義し、これをネットワーク設定部の設計とする。

アドレスバルクの状態遷移

アドレスバルクの状態として 11 個の状態を定義した。表 5.4に概要と共に示す。

表 5.4: アドレスバルクの状態

状態名	概要
Null	空のアドレスバルクを意味する初期状態。ルートサーバ以外のサーバは初期状態では Null 状態のアドレスバルクだけを保持している。
Free	どのサーバにも割り当てられていないアドレスバルク。ルートサーバは初期状態で Free 状態のアドレスバルクを保持している。
Allocated	下位サーバに割り当てられているアドレスバルク。あるいは自身で管理している (クライアントに割り当てている) アドレスバルク。
Requesting	上位サーバへ割当を要求中のアドレスバルク。割り当てられると Free 状態に遷移する。
Request Failure	割当要求を拒否された (AllocNack を受け取った) アドレスバルク。この状態のアドレスバルクは再度上位サーバに対して割当要求してはならない。
Retrieving	下位サーバへ返却を要求中のアドレスバルク。返却されると Free 状態に遷移する。
Retrieval Failure	返却要求を拒否された (ReleaseNack を受け取った) アドレスバルク。この状態のアドレスバルクは再度下位サーバに対して返却要求してはならない。
Reserving	下位サーバからの割当要求に応じて上位サーバへ割当を要求中のアドレスバルク。
Reserved	上位サーバから割り当てられ、下位サーバからの割当要求を待っているアドレスバルク。一定時間内に要求がなければ Free 状態に遷移する。
Deprecating	上位サーバからの返却要求に応じて下位サーバへ返却を要求中のアドレスバルク。
Deprecated	下位サーバから返却され、上位サーバからの返却要求を待っているアドレスバルク。一定時間内に要求がなければ Free 状態に遷移する。

Request Failure および Retrieval Failure の 2 つの状態はメッセージの不必要な再送を防ぐために設けた。これらは「要求に失敗した」という情報を一定時間保持するために用いられる。例えば Request Failure 状態のアドレスバルクに対して下位サーバが AllocRequest を送信して来た場合には、AllocNack を返す。また Deprecating および

Deprecated 状態は、アドレスの付け替え時の古いアドレスを示すためのものである。これらの状態のアドレスバルクに対して AllocRequest が届いた場合には AllocNack を返す。

この 11 の状態をさらに詳しく、状態遷移図として示したのが図 5.3である。

例えば、あるアドレスバルクは上位サーバでは Allocated 状態であるが、同じアドレスバルクが下位サーバにとっては Free 状態であったりする。つまりアドレス空間の同じ領域であってもサーバごとに状態が異なる点に注意しなければならない (図 5.4)。

最後にネットワーク設定部の動作例を以下に示す。

1. ルートサーバ RS, RS の下位にサーバ B, サーバ B の下位にサーバ C があるとする。

2. RS には下流側ネットワークインタフェースが 1 つ、サーバ B, C には上流側と下流側にネットワークインタフェースがそれぞれ 1 つずつあるとする。

3. 起動直後には RS は 133.27.0.0/16[3] という Free アドレスバルクをアドレスプールに保持している。

4. サーバ B およびサーバ C のアドレスプールは空である。

5. サーバ C は下流側ネットワークで大きさ 16 のアドレスバルクが必要と推定し、サーバ B に対し AllocRequest を送る。同時に大きさ 16 で Requesting 状態のアドレスバルクを (空の) アドレスプールに追加する。

6. サーバ B は Null 状態のアドレスバルクに AllocRequest が届いたので、サーバ C に対して AllocPending を返す一方で、上位サーバ RS へ AllocRequest を送る。同時に大きさ 16 で Reserving 状態のアドレスバルクを (空の) アドレスプールに追加する。

7. サーバ B は下流側ネットワークで大きさ 32 のアドレスバルクが必要と推定し、ルートサーバに対し AllocRequest を送る。さらに大きさ 32, Requesting 状態のアドレスバルクをアドレスプールに追加する。

8. ルートサーバ RS は大きさ 16, 32 のアドレスバルクを割り当てる Alloc メッセージをサーバ B へ送る。割り当てられた領域はルートサーバ上では Free 状態から Allocated 状態へ遷移する。ルートサーバの下流側ネットワークにもアドレスバルクが割り当てられ、Allocated 状態となる。

5.3 ホスト設定部の設計

本システムはモジュール化を目標に全体設計を行なったため、ホスト設定部は他の部分からは独立に実現できる。しかしアドレス付け替えなどを支援するためには、複

[3] 133.27.0.0〜133.27.255.255 の範囲のアドレスを表す

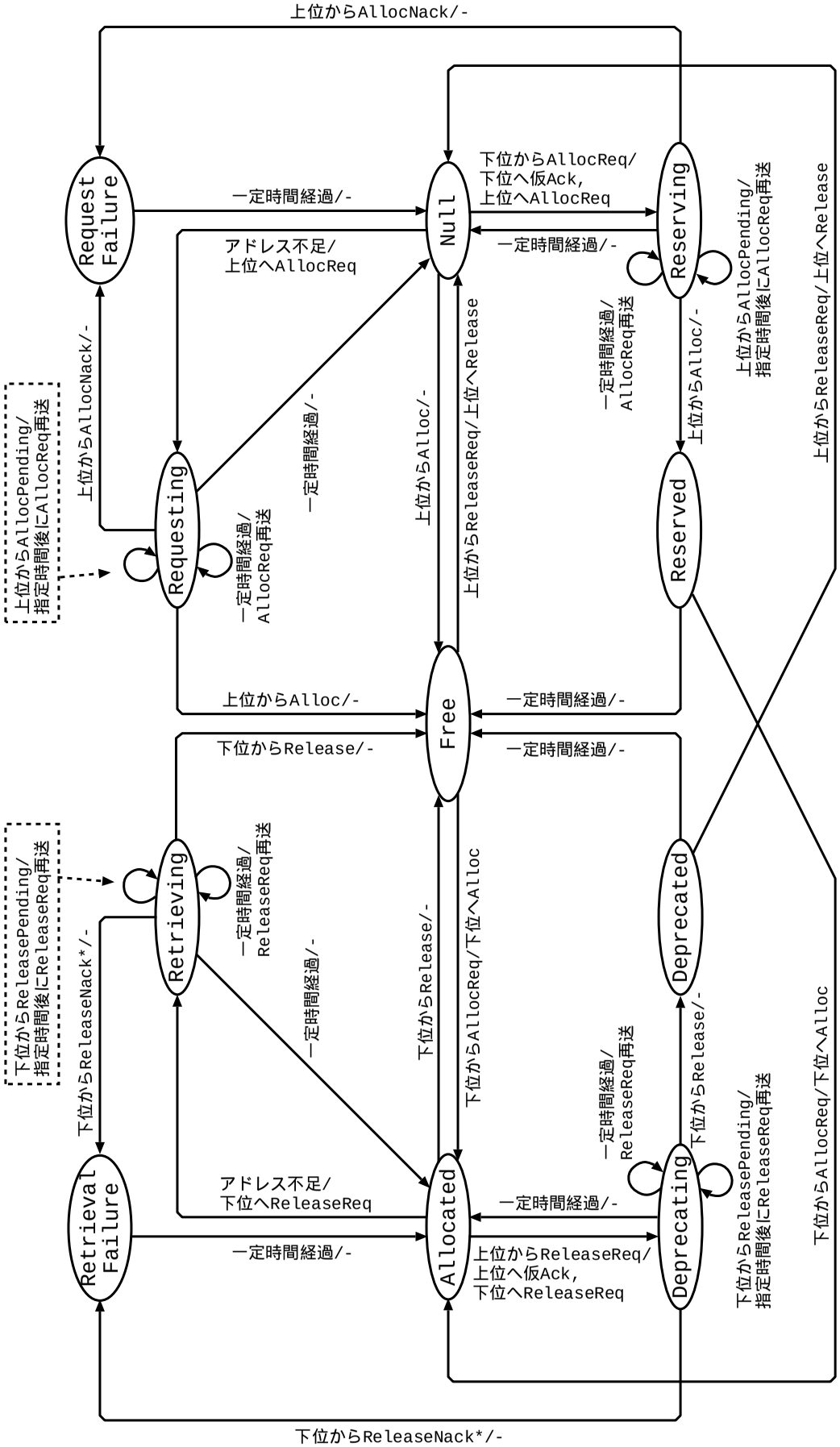

図 5.3: アドレスバルクの状態遷移

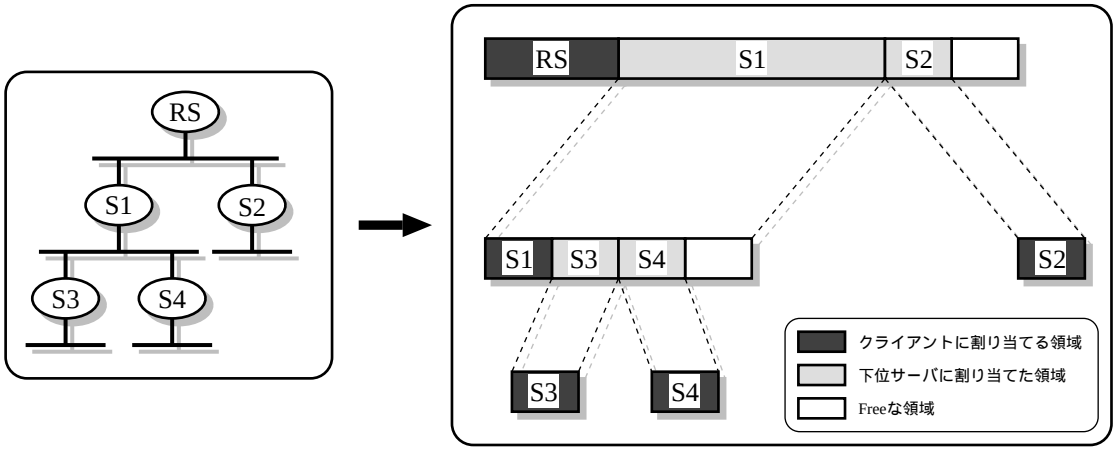

図 5.4: アドレスプールの状態例

数アドレスの設定が可能で、かつそれら複数アドレスの新旧を区別できなければならない、などのいくつかの要求がある。本章では、最初にホスト設定部の設計目標を明らかにし、次に詳細な設計を行なう。ただしアドレス付け替えの方法に関しては、第5.4章で詳しく論じるものとする。またホスト設定部の処理の流れについても、第 5.4章においてアドレス付け替えの方法とともに述べる。

5.3.1 設計目標

ホスト設定部の設計目標を以下に示す。

1. **複数アドレスの設定が可能**

   アドレス付け替えを実現するためには複数のネットワークアドレスを一つのネットワークインタフェースに設定できる必要がある。さらにネットワークアドレスごとに有効期限などの状態が管理できなければならない。

2. **アドレスの強制回収が可能**

   アドレス付け替えを実現するためには、ある時点をもって古いアドレスを強制的に回収する機構を持っていることが望ましい。クライアントが何らかの理由 (割当の有効期限切れなど) により自発的に再割当要求を送ってくるまで待つ方法もあるが、付け替えに要する時間などの面から見て望ましくない。

3. **ネットワークを移動したことを検知できる**

   携帯型計算機が、あるネットワークから別のネットワークへ移動した場合に、移動したことを自分自身で検知することは難しい。これはハードウェア的に検知できないためという場合もあるし、オペレーティングシステムでは検知できるが一

般プロセスからは検知できないためという場合もある。したがってホスト設定部に、そのような検出機構を組み込む。

4. **プロトコルを簡潔にする**

   本論文ではプロトコルを複雑にして高機能化するよりも、機能を限定してプロトコルを簡潔にすることを目標とする。例えば、DHCP には複数のサーバからの割当を比較し選択する機能があるのに対し、本システムには (各論理サブネットを管理するサーバは 1 つしか存在しないと仮定しているため) クライアントが複数サーバからの複数の応答を選択する機能を組み込まないものとする。

5.3.2 設計

前述の設計目標に対する解決方針は以下の通りである。

1. **複数アドレスの設定が可能**

   サーバは定期的にネットワークに関する情報を含む Advertise メッセージを送信する。クライアントは新しい Advertise メッセージを受信するとアドレスの割当要求を開始する。古いアドレスも有効期限が来るまでは保持できる。

2. **アドレスの強制回収が可能**

   Advertise メッセージに、強制回収のためのフラグを設ける。このフラグが設定された Advertise メッセージを受信したクライアントは、アドレスの返却処理を開始する。

3. **ネットワークを移動したことを検知できる**

   Advertise メッセージが定期的に送信されるため、これが一定時間以上受信できなかったり、新しい Advertise メッセージを受信した場合にはネットワークを移動したことが検知できる。

4. **プロトコルを簡潔にする**

   プロトコルを簡潔にするため、アドレスには有効期限などの差を設けない。また各セグメントを管理するサーバは 1 つしか存在しないと仮定することでプロトコルの簡略化を目指した。

以下では、解決方針および設計の詳細について述べる。ただし目標 1, 2 については次の第 5.4節で、アドレス付け替えに関する議論とともに述べる。

DHCP はアドレスの割当に 4 回メッセージを交換する必要があった。その概要を以下に示す。

- Discover メッセージ ‥ サーバを発見し、割当可能なアドレスを問い合わせる
- Offer メッセージ ‥ Discover に対し、各サーバが割当可能なアドレスを提示する

- Request メッセージ ‥ 複数サーバからの Offer を選択し、割当を要求する

- Ack メッセージ ‥ Request に対して、アドレスの割当を行なう

4 つのメッセージ交換が必要な理由として「1. 1 つのネットワークセグメントに複数のサーバが存在する場合があるため」および「2. サーバの割当可能なアドレスには有効期限の長さなどの差が存在するため」の 2 つがある。しかし本システムでは複数サーバが一つのネットワークセグメントに存在することは考えないため、1 番目の理由については考慮する必要がない。さらに本システムでは「割当可能なアドレスには質的な差を設けない」という方針を設定するため、2 番目の理由についても考慮する必要がない。

この方針はプロトコルを簡潔に設計するという目標に基づくものである。DHCP の実装および運用の経験から判断して、Offer される複数アドレスの間の差というのは、有効期限の長さのみという場合が多い。さらにネットワーク管理者がネットワークセグメントに複数のサーバを設ける場合でも、それらの設定には差をつけないことが普通であると思われる。したがって「割当可能なアドレスには質的な差を設けない」という方針を設定しても実用上の問題はない。

これらの理由から、本システムでのアドレスの割当は Request, Reply という 2 つのメッセージ交換によって実現することにした。Reply には割当を行なうための Ack と拒否するための Nack がある。またクライアントへ通知される情報は、IP アドレス, ネットマスク, デフォルトルータの 3 つのみとした。

さらに「ネットワークの移動を検知する」という目標をオペレーティングシステムに依存せずに実現するために、サーバから定期的にネットワークに関する情報を Advertise する。この Advertise メッセージが受信できなくなったり、あるいは別のネットワークに関する Advertise メッセージを受信した場合には移動したと判定する。素早く移動を検知するためには Advertise の間隔を短くする必要がある。実際の Advertise の間隔は、ネットワークの管理者がポリシに基づいて指定できるようにする。

ホスト設定メッセージのフォーマットを図 5.5 に、各フィールドの説明を表 5.5 に示す。ホスト設定部は独立なため、サーバ木構成部やネットワーク設定部と共通のヘッダを用いる必然性はないが、今回は共通なものを用いることとした。これらのメッセージを用いたホスト設定部の処理の流れについては、次節で述べる。

5.4 アドレス付け替えの概要

アドレス付け替えを支援するためにサーバ木構成部は IP アドレスに依存しない独自の機構として設計した。アドレス付け替えを実現するためには、さらにホスト設定部とネットワーク設定部にいくつかの機能が必要である。

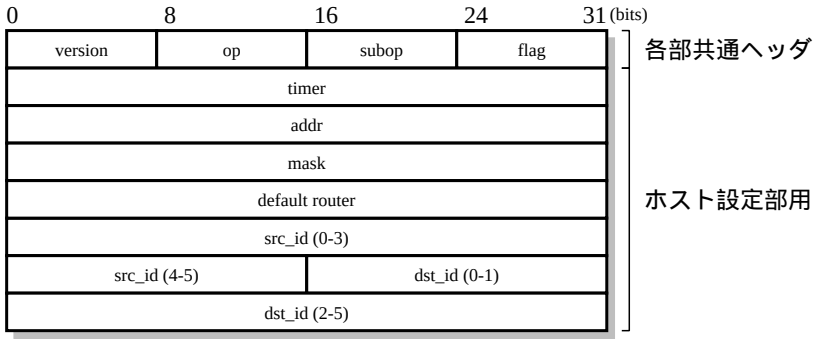

図 5.5: ホスト設定メッセージのフォーマット (Ethernet の場合)

表 5.5: ホスト設定メッセージの各フィールド

名称	用途
version	プロトコルのバージョンを指定する。現在のバージョンは 1。
op	ホスト設定メッセージであることを示す 2 を入れる。
subop	ホスト設定メッセージをさらに分類する。 (0 = Request, 1 = Ack, 2 = Nack, 3 = Advertise)
flag	Advertise の場合のみ、最下位ビットだけを Deprecating フラグ[4] として用いる。
timer	Ack および Advertise で、アドレスの有効期限を秒数で示す。
addr	Request の場合は希望する IP アドレスを、Ack の場合は割り当てる IP アドレスを示す。Advertise の場合については次の mask の用途を参照。
mask	Ack の場合は割り当てる IP アドレスのネットマスクを示す。Advertise の場合には、addr フィールドと組み合わせてネットワークアドレスを示す。
default router	Ack の場合のみ用いられ、デフォルトルータのネットマスクを示す。
src_id	メッセージ送信側の ID。図 5.5では、Ethernet アドレス (6 オクテット) を例として用いている。サーバは Request メッセージの src_id を Reply メッセージの dst_id にコピーする。
dst_id	メッセージの宛先の ID。

5.4.1 ホスト設定部における支援

アドレスの付け替えを実現するために、ホスト設定部は「サーバが任意の時点でクライアントに対してアドレスを返却させる」機能を持っている必要がある。以下に本システムでの実現方法、通常のホスト設定処理の流れと共に述べる。

1. サーバは定期的に自分の管理するネットワークアドレス (address と netmask の組) を Advertise する。またアドレス付け替えの最中には新旧それぞれのネットワークアドレスについて Advertise する。ただし付け替え中の古いネットワークアドレスについては、それを示す Deprecating フラグおよび有効期限を Advertise メッセージに含める。

2. クライアントは、random 秒後に Request を送り、Reply を受け取る。

3. クライアントは自分の属するネットワークアドレスの Advertise を一定時間以上受信できないか、あるいは Advertise メッセージに Deprecating フラグがセットされている場合には、異なるネットワークアドレスに関する Advertise を送信されるのを待つ。

4. 自分の属するネットワークアドレスの有効期限が来たら、そのアドレスの利用は中止する。

5. 異なるネットワークアドレスの Advertise を受け取ったら 2に戻る。

5.4.2 ネットワーク設定部における支援

ネットワーク設定部では「新しいアドレスの割当」「古いアドレスの回収」「経路制御の変更」の 3 つを実施する必要がある。アドレス付け替え手順の概要を以下に示す。

1. 期限付き (X 秒後とする) で ReleaseRequest を下位サーバに送る。

2. 受け取った下位サーバは、その領域が Free であれば直ちに上位サーバに返却する。

3. その領域を下位サーバに割り当ててある場合には、下位サーバに RelaseRequest を送る。同時に上位サーバには ReleasePending メッセージを送る。ReleasePending メッセージのタイマーフィールドは Y 秒[5]にセットする。

4. その領域を自分で使用している (クライアントに割り当てている) 場合には、上位サーバにタイマーフィールドを X 秒にセットした ReleasePending メッセージを送る。同時に上位サーバへ新たなアドレスの割当を要求し、前述のネットワークアドレス Advertise を開始する。

5. 期限が来るまでに、上位サーバへアドレスを返却する。

[5] Y は X よりも小さい。

6. サーバは自らが使用 (クライアントへ割り当て) を開始する時に経路情報をアナウンスし始める。逆に上位サーバへアドレスを返却した時点で、経路情報のアナウンスを止める。

5.5 ネットワーク管理ポリシに関する考察

ネットワーク管理の方針はネットワークごとに大きく異なる。以下では、ネットワーク管理ポリシの例を示す。

はじめに考えられるのが「アドレスの利用効率を重視するポリシ」である。このポリシはアドレスの効率的な利用を第一目標とする。アドレスの効率的利用を目指すためには、サーバに割り当てるアドレスプールとアドレスプールの間の空き領域を可能な限り少なくしなければならない。逆に、あるネットワークに接続された計算機の数が増加した場合に、アドレス空間の拡大をしようとしても既にその領域が他のサーバに割り当てられている可能性が高い。したがって、希望する大きさのアドレス空間を得るためには、アドレスの付け替えを行なう必要が生じる。現在のネットワークアーキテクチャではアドレス付け替えによる影響は大きいため、現実的なポリシではない。

次に考えられるのが「アドレス付け替えの頻度を可能な限り小さくするポリシ」である。このポリシでは、意図しないアドレス付け替えはできる限り発生しないようにする。アドレス空間の拡大を行なう場合でもアドレス付け替えが発生しないようにするには、サーバに割り当てるアドレスプールとアドレスプールの間に十分な空き領域を確保する必要がある。これは前述の「アドレスの利用効率を重視する」というポリシとは矛盾する。このポリシは非常に現実的であり、一般的な手動によるネットワーク管理ではこのポリシが重視されている。

3番目のポリシは「経路情報の集約が可能となるように割り当てる」というものである。このポリシでは、ネットワークトポロジに対応する形でアドレスプールを割り当てる必要がある。このポリシを優先すると、新しくネットワークが接続された場合に他のネットワークのアドレスを付け替える必要が発生する。これも現実的なポリシとは言えない。また経路情報の集約は、外部ネットワークと組織内ネットワークの境界に存在するルータのみ行なう場合が多い。本システムの適用範囲は主に組織内ネットワークなので、このポリシを重視する必要性は低い。

この他にも多様なポリシが存在すると思われるが、それらのポリシは必ずしも共存できない。また階層構造のある部分と別な部分で別々のポリシを採用すると矛盾が生じてしまう可能性が高い。

今後は、どのポリシがどのような性質を持つのか、あるいはどのポリシが階層的なネットワーク設定管理機構と親和性が高いのかを検討する必要がある。

第 6 章

階層型ネットワーク自動設定機構の実装

第 4 章および第 5 章において述べた設計に基づき、BSD/OS 2.0.1 上で C 言語を用いてネットワーク自動設定機構を実装した。本章でもシステムを 3 つの部分に分けて順に論じる。さらに本システムの動作を検証するために作成した簡易ネットワークエミュレータについても述べる。

6.1 サーバ実装の構造

ここではサーバ実装全体の概要について述べる。

サーバは他のサーバから送られてくるメッセージの受信待ちを行ないつつ、多数のメッセージの送信や再送を行なう必要がある。そのような場合に UNIX では *select()* システムコールを用いて入出力の多重化を行なう方法が一般的である。しかし、本システムのサーバは複数のアドレスバルクに対する複数の操作を並行して行なう必要がある。例えば複数の下位サーバに対して同時に返却を要求するような状況で使用するためには、*select()* システムコールは機能が不十分である。

このように複雑な要求するを実現するためには、次のような方法が考えられる:

1. マルチプロセスのシステムとして実装し、プロセス間通信を用いて同期をとる

2. マルチスレッドのプロセスとして実装する

3. サーバプロセス自身でスケジューリングを行なう

1 の方法を用いる場合には、アドレスプールに対する操作の排他制御を実現する必要がある。また多数のプロセスが起動するために、サーバを実行しているマシンに不必要な負荷がかかる可能性がある。2 の方法を実現するためには、オペレーティングシステムがマルチスレッドをサポートしていなければならない。しかしマルチスレッドを使用すると、プログラムの移植が困難になる上に BSD/OS 自体もマルチスレッドをサポートしていない。これらの理由から、今回は 3 番の方法を採用することとした。

スケジューラはライブラリの *signal()* 関数[1]および *alarm()* 関数[2]を用いた非同期処理として実装したため、サーバは *select()* システムコールを用いてメッセージの受信待ちをすることができる。スケジュールは表 6.1のような構造体を用いて記述する。

表 6.1: schedule 構造体

メンバ名	型	用途
epoch	time_t	スケジュールが実行される時刻
flag	unsigned char	スケジュールの属性を示す
id	int	スケジュールの削除などに用いられる識別番号
func	int *()	指定時刻になった時に呼び出す関数へのポインタ
arg	void *	func() に渡される引数
dont_free_arg	unsigned char	スケジュール実行後に *free(arg)* [3]を実行させない
prev	struct schedule *	スケジュールへのポインタ (double linked list)
next	struct schedule *	スケジュールへのポインタ (double linked list)

スケジューラは厳密な時間管理を要求されるわけではないので、(実行予定時刻順に並んでいる) スケジュールリストの先頭から実行予定時刻の過ぎたスケジュールを取り出して実行する。実行が終わった場合に、dont_free_arg が 0 ならば *free(arg)* を実行してメモリを解放する。その後に次のスケジュール実行予定時刻にタイマをセットする。また新しくスケジュールが追加されたり削除されたりした場合には、次のスケジュール実行予定時刻を再計算する。

タイマによって非同期に呼び出されるスケジューラとは別に、*main()* ではメッセージ駆動型のループを用いている。*select()* で受信待ちを行い、受信したメッセージの op コードによってサーバ木構成部 (*treeconf* 関数)、ネットワーク設定部 (*netmng* 関数)、ホスト設定部 (*hostconf* 関数) を呼び出すようになっている。

サーバの移植性を高めるために、本システムは OS に組み込まずにアプリケーションプログラムとして実装した。メッセージの交換には任意の方法が利用できるが、IP や Ethernet に独自のプロトコルを組み込む場合にはプロトコル番号などの登録が必要となる。そのため今回は比較的自由に利用することのできる UDP を採用した[4]。また本システムはネットワークが設定される前に動作しなければならないため、通常の IP ユニキャストを用いることができない。そのためブロードキャストないしマルチキャストを利用する必要がある。今回の実装ではブロードキャストを用いることとした。移植の容易さを維持しつつブロードキャストを行なうために、本実装では Berkeley Packet Filter (BPF)[25] を利用した。

本システムはネットワーク管理を行なうために、どのネットワークインタフェース

[1] シグナルが届いた場合の処理方法や呼び出される関数を指定するための関数。
[2] 指定秒数後に自分自身に ALARM シグナルを送るための関数。
[4] TCP は IP アドレスが設定されていないと用いることができないため不適当である。

からメッセージを受信したかを認識する必要がある。またメッセージを送信する場合にも、どのネットワークインタフェースから送信するかを指定できなければならない。通常のソケットインタフェースでは、これらの要求を満たせないが、BPF を利用すればインタフェースドライバから直接読み書きできる。これも BPF を採用した理由の 1 つである。

BPF は BSD 系の UNIX には標準で実装されているため移植性は比較的高い。他のオペレーティングシステムでも何らかの形で BPF 同様の機能を提供しているので、それらの機能を用いれば移植が可能である。

6.2 サーバ木構成部の実装

サーバ木構成部に必要な機能は「経路情報の管理」「上位サーバの決定と登録」「Designated サーバの決定」の 3 つに大別できる。初めにサーバ木構成部の処理の流れについて述べる。サーバ木構成部の処理は次のような順序で行なわれる。

- ルートサーバは 10 秒ごとに経路情報をアナウンスする。
- ルート以外のサーバは、経路情報メッセージを受信したら処理を行ない、自分自身の経路情報を送信する。
- 初めて経路情報を受け取ってから、10 秒後に上位サーバの決定と登録を開始する。これは経路情報を受け取ってから 5 秒以内に一時経路表が安定し、本経路表に複製されるという仮定に基づく。
- 上位サーバは UpperRequest を受け取ると、送信者を下位サーバとして登録し、UpperAck を返す。UpperAck を受信した下位サーバは上位サーバを登録し、Designated サーバを選定するためにメッセージの送信を開始する。
- Designated メッセージの再送間隔 (5 秒) と再送回数 (5 回) から、少なくとも 30 秒程度経過すると Designated サーバが決定できる。
- Designated サーバが決定され、ネットワーク設定部の処理を開始する。

サーバ木構成部の主な関数を示す。

- **treeconf()**

  サーバ木構成メッセージを Routing メッセージ, UpperRequest, UpperAck, Designate メッセージに分類し、それぞれ *proc_routing(), proc_upper_request(), proc_upper_ack(), proc_designate()* 関数を呼び出す。ネットワークインタフェースリスト、受信ネットワークインタフェース[5]、サーバ木構成メッセージの 3 つを引数にとる。

[5] メッセージを受信したネットワークインタフェースのこと。

- **proc_routing()**

  受信したRoutingメッセージを処理し、必要ならば自らも経路情報を送信する。この関数は *treeconf()* と同じ3つの引数をとる。初めて呼び出された時には10秒後に *upper_request()* を呼び出すようにスケジュールする。また新しいsequence番号のRoutingメッセージを受け取った場合には、5秒後に一時経路表を本経路表に複製するようにスケジューリングする。

- **proc_upper_request()**

  受信したUpperRequestメッセージに対し、UpperAckを返すとともに下位サーバを登録する。引数は受信ネットワークインタフェースとサーバ木構成メッセージの2つである。

- **proc_upper_ack()**

  UpperAckを受信し、上位サーバを登録する。さらにUpperRequestの再送のスケジュールを *del_retrans_sched()* 関数を用いて削除する。またDesignatedサーバの選定を開始するために *start_designate()* 関数を呼び出す。引数はネットワークインタフェースリスト、受信ネットワークインタフェース、サーバ木構成メッセージの3つである。

- **proc_designate()**

  受信したDesignatedメッセージをチェックし、自身がDesignatedサーバにならないことが決定した場合には、Designatedメッセージの送信スケジュールを削除する。引数は受信ネットワークインタフェースとサーバ木構成メッセージの2つである。

- **advertise_route()**

  10秒に一度経路情報をアナウンスする。ルートサーバ以外のサーバでは呼び出されない。引数はネットワークインタフェースリストのみである。

- **upper_request()**

  UpperRequestを送信し、上位サーバへ登録を行なう。ルートサーバでは呼び出されない。引数はとらない。

- **start_designate()**

  Designatedであるネットワークインタフェースごとに、*send_designate()* を呼び出す。引数はネットワークインタフェースリストのみである。

- **send_designate()**

  Designateメッセージを送信する。引数はネットワークインタフェース構造体へのポインタのみである。

- **determine_designate()**

  最終的に Designated なネットワークインタフェースを決定し、ネットワーク設定部に処理を移す。引数はネットワークインタフェースリストのみである。

最後に経路表を保持する routing_tbl 構造体および上位・下位サーバを保持する tree 構造体、branch 構造体を以下に示す。

- **routing_tbl 構造体**

メンバ名	型	用途
root	struct netconf_id	宛先 (ルートサーバ) の ID を保持する構造体
next_hop	struct netconf_id	転送先 (上位サーバの候補) のサーバ ID
ifp	struct interface *	経路情報を受信したネットワークインタフェースへのポインタ
seq	unsigned short	経路情報に付随する順序番号
metric	unsigned char	ルートサーバからのホップ数

- **tree 構造体**

メンバ名	型	用途
upper	struct netconf_id	上位サーバの ID を保持する構造体
upper_if	struct interface *	上位サーバ側のネットワークインタフェースへのポインタ
branch	struct branch	下位サーバのリストを保持する構造体

- **branch 構造体**

メンバ名	型	用途
lower	struct netconf_id	下位サーバの ID を保持する構造体
lower_if	struct interface *	下位サーバ側のネットワークインタフェースへのポインタ
next	struct branch *	次の branch 構造体へのポインタ (linked list)

6.3 ネットワーク設定部の実装

ネットワーク設定部は図 5.3に示した状態遷移図に基づいて実装した。本実装の処理概要は以下のようになる。

1. Designated サーバに決定するとネットワーク設定部の処理を開始する。

2. ネットワーク設定部は、ネットワークインタフェース毎に必要なアドレスバルクの大きさを推定し、上位サーバに AllocRequest を送信する。AllocRequest の再送スケジュールも設定する。

3. 以後、上位サーバや下位サーバ、ホスト設定部とアドレスバルクを授受する毎に、アドレスバルクが不足していないか検証する。不足していれば上位サーバに AllocRequest を送信する。

4. 今回の実装では、自動的には下位サーバに返却を要求しない。返却要求 (ReleaseRequest) はネットワーク管理者がルートサーバに指示を送った場合にのみ送信される。

2番目に示した「必要なアドレスバルクの推測」は、起動直後だけ特別に処理する必要がある。すなわち通常、ホスト設定部はクライアントへのアドレス割当を管理しているため、必要となるアドレスの大きさも把握している。しかし起動直後には、クライアントへのアドレス割当は一切行なわれていないために、別な方法を用いなければならない。起動直後のアドレスバルクの推定には、次のような方法を用いることができる。

1. ネットワーク設定部は、Designated ネットワークインタフェースから、SolicitationRequest メッセージをブロードキャストする。
2. クライアントは、1〜5 秒間待機してから SolicitationReply を返す。
3. サーバはクライアントからの SolicitationReply の数をカウントする。これを 3 回繰り返し、最大値を採用する。

図 6.1に Solicitation メッセージのフォーマットを、表 6.2に各フィールドの用途を示す。

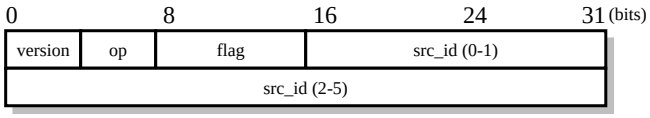

図 6.1: Solicitation メッセージのフォーマット

表 6.2: Solicitation メッセージの各フィールド (Ethernet の場合)

フィールド名	用途
version	プロトコルのバージョンを示す。現在はバージョン 1 である。
op	SolicitationRequest の場合は 0, SolicitationReply の場合は 1 を入れる。
src_id	送信者の ID を入れる。クライアントは Request メッセージの src_id に対して Reply を返す。

今回の実装は、サーバ木構成部およびネットワーク設定部はネットワークエミュレータを用いて検証した。これについては本章の最後で述べる。またホスト設定部はエミュレータを用いずに実装したため、Solicitation メッセージによる手法は用いなかった。代わりにアドレスバルクの大きさを乱数を生成して返却する関数を作成した。

ネットワーク設定部の中核をなすアドレスバルクを管理するため、アドレスバルクの 11 の状態に対応して次のような関数の配列を定義した:

```
int (*addr_fsm[])(struct addrpool *, struct event) = {
  null_addr,
```

```
  free_addr,
  allocated,
  requesting,
  request_failure,
  retrieving,
  retrieval_failure,
  reserving,
  reserved,
  deprecating,
  deprecated
};
```

サーバはネットワーク設定用メッセージを受信すると次のような処理を行なう。

- AllocRequest などのネットワーク設定用メッセージを受信すると、その中に含まれる先頭 IP アドレスおよびネットマスクを取り出す。この 2 つの値でアドレスバルクを完全に指定することができる。

- サーバのアドレスプールデータベースに、メッセージで指定されたアドレスバルクと完全に一致する (すなわち先頭 IP アドレスおよびネットマスクが等しい) アドレスバルクがあれば、そのアドレスバルクの状態に対応する関数を呼び出す。

- 指定されたアドレスバルクが、サーバにおいては複数の異なるアドレスバルクに分割されている場合には、各々に対して関数を呼び出す。(eg. メッセージでは 133.4.27.0/26 というアドレスバルクを指定したが、サーバでは Allocated 状態の 133.4.27.0/27 と Free 状態の 133.4.27.32/27 の 2 つのアドレスバルクに分割されているような場合。)

- メッセージで指定されたアドレスバルクを包含するアドレスバルクが存在する場合には、そのアドレスバルクを 2 の冪乗境界にそろえた大きさに分割してから関数を呼び出す。(eg. メッセージでは 133.4.27.0/27 を指定したが、サーバでは Free 状態の 133.4.27.0/26 を保持しているような場合。)

- 逆にメッセージで指定されたアドレスバルクの部分集合だけしか保持していない場合には、欠けている部分のアドレスバルクを補完してから関数を呼び出す。(eg. メッセージでは 133.4.27.0/26 を指定したが、サーバでは 133.4.27.0/27 しか保持していないような場合。)

ネットワーク設定部の設計では「ポリシとプロトコルを分離する」ことを目標とした。そのためネットワーク設定部を実装するのに必要なアルゴリズムのいくつかは指定されていない。本実装では次のようなアルゴリズムを採用した。

- ある 1 つのネットワークセグメントに割り当てるアドレスの総数の 50%を使い切った時点で、アドレスが不足したと判定する。

- アドレスが不足した場合に、下位サーバへ返却を要求することはしない。常に上位サーバへ割当を要求し、アドレス不足を解消する。
- 下位サーバへ返却を要求するのは、ネットワーク管理者がルートサーバにそのような操作を行なった場合のみとする。各サーバが自律的に返却を要求することはしない。
- アドレスが過剰であっても、上位サーバから返却を要求されない限りは返却を行なわない。

6.4 ホスト設定部の実装

本論文では、ホスト設定部をネットワーク設定部と統合せずに単独のプロセスとして実装した。理由として次の 3 つがあげられる:

- **負荷を分離する**

  何らかの理由 (例えば停電からの復旧直後に起こる計算機の一斉起動) により負荷が急激に増大した場合に、ホスト設定部の負荷がネットワーク設定部に影響を与えること (あるいはその逆) は望ましくない。単独のプロセスとして実装することで、負荷を分離し、残る一方の処理能力を維持することができる。

- **メンテナンスを容易にする**

  ネットワーク設定部とホスト設定部の処理を分離することで、両者のシステムメンテナンス (デバグなど) を容易にすることができる。

- **モジュール性の高さを実証する**

  設計目標に掲げたモジュール性の高さを実証し、既存のホスト設定機構や新しいホスト設定機構と組み合わせても動作することを確認する必要がある。今回は、その第一段階としてホスト設定部を独立に実装した。今後は、ネットワーク設定部とホスト設定部の間のインタフェースを規定し、両者を組み合わせて運用してみなければならない。

以下本章では、ホスト設定サーバとクライアントの実装について順に述べる。今回どちらも BPF を用いて実装した。また実装を簡潔にするために、物理メディアは Ethernet に限定した。

6.4.1 サーバの実装

サーバは「Advertise メッセージの定期送信」と「クライアントへのアドレス割当」および「ネットワーク設定サーバとの通信[6]」の 3 つの機能を提供する。ただし今回は

[6] ネットワーク設定部からホスト設定部を制御するために利用する。

ネットワーク設定サーバとの通信機能については実装を行なわなかった。これについては今後の課題とし、今回は簡単なユーザインタフェースでサーバを制御できるようにした。

ホスト設定サーバはクライアントからのアドレス割当要求を処理しながら、Advertiseメッセージの送信を定期的に行なう必要がある。ホスト設定サーバは複数のネットワークインタフェースを処理するため、少なくともネットワークインタフェースの数だけタイマが必要となる。アドレス付け替えの最中などは 1 つのネットワークインタフェースに複数のアドレスが割り当てられているために、必要なタイマの数はさらに多くなる。しかし UNIX では 1 つのプロセスが一度に管理できるタイマ (アラーム) の数は 1 つに限定されているため、ホスト設定サーバも、ネットワーク設定サーバと同じように独自のスケジューラを用いて実装した。スケジューラの実装および schedule 構造体はネットワーク設定サーバのものと同一なため、ここでは実装の詳細を省略する。

サーバがネットワークインタフェースに関する情報を保持するために使用する interface 構造体を表 6.3に示す。

表 6.3: ホスト設定サーバ用 interface 構造体

メンバ名	型	用途
name	char	ネットワークインタフェースの名前 (eg. “le0”)
mac_addr	char	Ethernet アドレスを保存する
fd	int	BPF のファイル記述子
netinfo	struct netinfo *	ネットワークに関する情報を保持する構造体を指す
next	struct interface *	次のネットワークインタフェースを指す (linked list)

netinfo 構造体は、ネットワーク設定サーバから割り当てられたアドレスバルクを保持するためのものである。すなわちアドレスバルクの先頭 IP アドレスとネットマスク、それにルータの IP アドレスを保持する。さらにアドレス付け替えの際には、古いアドレスが Deprecating 中であることを示すフラグやアドレスバルクの最終有効期限 (これを過ぎるまでに上位サーバにアドレスバルクを返却する) などを保持する (表 6.4)。

アドレスバルクの中から、割り当てるアドレスを決定する基準は管理ポリシに関わるため、ホスト設定部の設計で論じなかった。本実装では以下のアルゴリズムに基づいて割当を行なう。

- 希望するアドレスが指定されている場合、そのアドレスが
  - そのクライアントに割り当てられていて、かつ有効期限内ならば Ack を返す。
  - 他のクライアントに割り当てられておらず、かつ Deprecating の印がついていなければ、そのアドレスを Ack で割り当てる。
  - 他のクライアントに割り当てられているか、あるいは Deprecating の印がついている場合は Nack を返す。

表 6.4: netinfo 構造体

メンバ名	型	用途
addr	struct in_addr	アドレスバルクの先頭アドレスを保持する
mask	struct in_addr	アドレスバルクの大きさを示すネットマスクを保持する
dflt_route	struct in_addr	デフォルトルータの IP アドレス
deprecating	char	上位サーバへ返却を開始したアドレスバルクであることを示す
expire	time_t	アドレスの割当期限を保持する。Deprecating である場合には、返却期限の時刻を保持する。
next	struct netinfo *	同じネットワークインタフェースに属する次のアドレスバルクを指す (linked list)

- 希望するアドレスが指定されていない場合は、他のクライアントに割り当てられておらず、かつ Deprecating でないアドレスを選択し割り当てる。

最後にサーバを構成する主な関数を示す。

- **proc_request()**

  受信したアドレス割り当て Request をチェックする。希望アドレスが指定されていれば、*check_addr()* 関数を用いて割り当て可能かどうかチェックする。割り当て可能であれば *send_ack()* 関数で Ack を送信する。不可能な場合には *send_nack()* 関数で Nack を返す。希望アドレスが指定されていなければ、*choose_addr()* 関数を用いて割り当てるアドレスを決定する。この関数は、受信ネットワークインタフェースと受信メッセージを引数にとる。戻り値は 0(成功) か −1(失敗) のどちらかである。

- **check_addr()**

  引数として割り当て希望アドレスへのポインタと受信ネットワークインタフェースへのポインタ、netinfo 構造体へのポインタの 3 つを取る。そのネットワークインタフェースに関係するアドレスバルクをチェックし、割り当て可能なら引数で受け取った netinfo 構造体へ必要な情報をセットし 0 を返す。割当が不可能なら −1 を返す。

- **choose_addr()**

  引数として受信ネットワークインタフェースへのポインタと netinfo 構造体へのポインタを取る。そのネットワークインタフェースに関係するアドレスバルクをチェックし、割り当て可能なアドレスを決定する。割り当て可能なアドレスがあ

れば引数で受け取った netinfo 構造体へ情報をセットし 0 を返す。割当可能なアドレスがなければ –1 を返す。

- **send_ack()**

  ネットワークインタフェースへのポインタと割り当てるアドレスを示す netinfo 構造体へのポインタを引数に取り、Ack メッセージを送信する。戻り値はない。

- **send_nack()**

  ネットワークインタフェースへのポインタを引数に取り、Nack メッセージを送信する。戻り値はない。

- **advertise()**

  ネットワークインタフェースリストへのポインタを引数にとる。定期的 (5 秒ごと) に Advertise メッセージを送信する。戻り値はない。

6.4.2 クライアントの実装

クライアントも schedule 構造体を用いてスケジュール管理を行なう。またクライアントはネットワークインタフェースに割り当てられるアドレス 1 つに対し addrinfo 構造体を 1 つ保持する (表 6.5)。

表 6.5: addrinfo 構造体

メンバ名	型	用途
addr	struct in_addr	ネットワークアドレスを保持する
mask	struct in_addr	ネットマスクを保持する
dflt_route	struct in_addr	デフォルトルートを保持する
deprecating	char	付け替え中のアドレスであることを示す
counter	int	Advertise メッセージの受信回数をカウントする
expire	time_t	割当の有効期限、またはアドレスの返却期限
ifp	struct interface *	ネットワークインタフェースへのポインタ
next	struct addrinfo *	割り当てられている次のアドレスを指す (linked list)

addrinfo 構造体に含まれる ifp は interface 構造体へのポインタである。interface 構造体を表 6.6に示す。

クライアントは次のように処理を行なう。

1. 初期状態では addrinfo のリストは空である。

2. 新しい Advertise メッセージを受信したら、addrinfo 構造体のためにメモリ領域を確保し、addrinfo のリストに加える。さらに counter を 5 に初期化する。

表 6.6: ホスト設定クライアント用 interface 構造体

メンバ名	型	用途
name	char	ネットワークインタフェースの名前 (ex. “le0”)
mac_addr	char	Ethernet アドレスを保存する
fd	int	BPF のファイル記述子
addrinfo	struct addrinfo *	addrinfo 構造体のリストの先頭を指す。

3. random 秒後に Request の送信スケジュールを設定する。

4. 設定した時刻になったら Request を送り、再送をスケジュールする。

5. Ack を受け取ったら再送スケジュールを削除し、代わりに割当期限の延長要求をスケジュールする。

6. 延長の要求に失敗した場合には 2に戻る。

7. 定期的に counter を減らしていくようにスケジュールを設定する。

8. 適切な Advertise メッセージを受信したら counter を 5 に再初期化する。

9. もし一定時間以上、Advertise メッセージを受け取れない場合には、counter が 0 になる。この場合、2に戻る。

10. Deprecating フラグの設定された Advertise を受信したら、返却時刻を取り出して addrinfo 構造体の expire に設定する。

11. 返却時刻になったら、addrinfo を解放する。ネットワークインタフェースの設定も同じように行なう。

最後にクライアントが用いる主な関数について簡潔に述べる。

- **proc_advertise()**

  Advertise メッセージを処理する。受信したネットワークインタフェースおよびメッセージを引数にする。

- **send_request()**

  割当要求を送信する。引数としてネットワークインタフェースと、サーバの ID の 2 つを渡す。もし counter が 0 ならば送信を中止する。延長要求の場合に Deprecating がセットされていた場合も送信を中止する。

- **config_if()**

  ネットワークインタフェースへのポインタと、Ack メッセージへのポインタを引数として渡すと、ネットワークインタフェースやデフォルトルートなどについて適切な設定が行なわれる。戻り値は 0(成功) か−1(失敗) のどちらかである。

- **counter_monitor()**

  定期的に counter を減算する関数。引数は addrinfo リストへのポインタのみで、戻り値はない。

6.5 Ethernet エミュレータの実装

本システムは複数のネットワークセグメントを自動的に設定管理するためのものである。したがって実装の動作検証を行なうためには多数のネットワークと計算機などが必要となる。しかし現実には、そのような規模の実験環境を用意することは困難である。また仮にそのような環境が用意できたとしても、さまざまなネットワークトポロジを構築して実験、検証するには多くの時間が必要となる。

したがって仮想的にネットワークを構築し、実験できる環境を構築することが望ましい。そのような環境を実現するための方法としては、本システムを全てネットワークシミュレータ上に実現するという方法がある。もう一つはじめからから実装する方法である。エミュレータ方式は、メッセージの送受信部分以外は将来的に現実のネットワーク上にシステムを構築する場合に、そのまま利用できること、さらにシミュレータはそれ自体を実装するのに必要な労力が大きいことなどから、本論文ではエミュレータ方式を採用した。

今回実装したエミュレータは、UDP を用いた簡易なものである。エミュレータは実際にエミュレーションを行なうエミュレーションサーバと、エミュレーション機能を利用するための関数群から構成される。エミュレータを利用するためには「エミュレーション用の初期化ルーチンを呼ぶ」および「BPF に対して *read()*, *write()* などの操作を行なう部分をエミュレーション用の関数に置き換える」という 2 つの変更が必要となる (図 6.2)。

エミュレータは以下のようにして動作する。

- ネットワーク設定サーバがネットワークインタフェースを初期化する。この時、ネットワーク設定サーバからは BPF を利用しているように見える。

- 各ネットワーク設定サーバはエミュレーションサーバへ登録を行なう。登録に必要な情報は、エミュレータ用の設定ファイルから読み込むようになっている。

- 登録メッセージには、ネットワーク設定サーバ (正確にはネットワーク設定サーバのネットワークインタフェース) の MAC アドレスと接続すべきネットワークの識別番号が含まれる。エミュレータは識別番号ごとに設けられたリストに MAC アドレスを追加登録する。

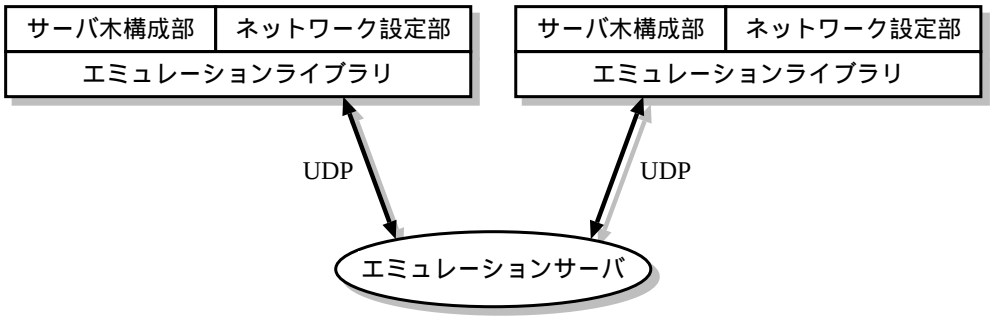

図 6.2: エミュレータの概念図

- 各ネットワーク設定サーバがネットワークに任意のデータを送信するとエミュレーションサーバに届く。この時、データの先頭部分には登録する時に用いたものと等しいネットワークの識別番号が付加されている。当然ながら、ここでもネットワーク設定サーバからは BPF を利用しているように見える。

- BPF を用いた場合と同様に、メッセージには宛先 MAC アドレスが含まれる。メッセージを受信したエミュレーションサーバは、取り出したネットワーク識別番号に対応するリストを検索する。エミュレーションサーバは検索の結果、宛先 MAC アドレスと一致したネットワーク設定サーバにのみメッセージを転送する。また宛先 MAC アドレスがブロードキャストの場合には全てのネットワーク設定サーバにメッセージを転送する。

このような実装で十分なエミュレーションが可能だが、現実のネットワークとは次のような差がある。

- パケットロストがほとんど発生しない。パケットロストを実現することは容易だが今回は影響は少ないと判断し、そのままとした。

- ブロードキャストはリストに登録された順に配送される。これに対し現実のネットワークでは、ほとんど同時にブロードキャストが到着する。ただし現実の環境でも、パケットが同時に到着していてもオペレーティングシステムから各プロセスにメッセージが渡されるまでの時間は異なる。したがってこれについても今回は無視した。

次に各ネットワーク設定サーバからエミュレーションサーバに送られるメッセージのヘッダの構造を表 6.7に示す。ヘッダは単純に送信するネットワークの識別番号と、バイト境界をそろえるための 1 オクテットのダミーフィールドからなる。ヘッダの後には、通常の BPF を読み書きした場合のように Ethernet フレームのヘッダ、IP ヘッダ、UDP ヘッダなどが続いているため、メッセージの宛先などは、エミュレーションメッセージのヘッダに含める必要がない。

表 6.7: エミュレーションメッセージのヘッダ

フィールド名	サイズ (オクテット)	用途
hub	1	ネットワークの識別番号
zero	1	境界を揃えるためフィールドで、0 を入れる

最後に登録メッセージのフォーマットを図 6.3に示す。また各フィールドに関する説明を表 6.8に示す。

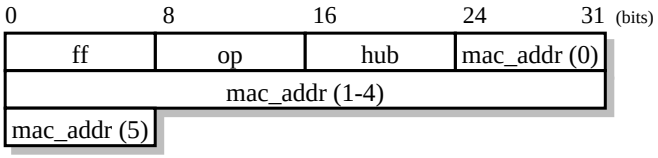

図 6.3: エミュレータ登録メッセージのフォーマット

表 6.8: エミュレータ登録メッセージの各フィールド

フィールド名	用途
ff	登録メッセージを他のメッセージと区別するためのフィールドで、0xff をいれる
op	登録要求 (Request) と登録確認 (Ack) の識別
hub	ネットワーク識別番号
mac_addr	ネットワーク設定サーバの仮想ネットワークインタフェースの MAC アドレス

第 7 章

評価・考察

本章では実装の評価および考察を行なう。はじめに実験環境について説明し、その後にサーバ木構成部、ネットワーク設定部、ホスト設定部の順に評価を行なう。

7.1 実験環境について

はじめにサーバ木構成部およびネットワーク設定部の動作確認に用いた仮想ネットワーク環境について述べ、次にホスト設定部のアドレス付け替え実験に用いた実験環境について述べる。

サーバ木構成部およびネットワーク設定部の 2 つについては、Ethernet エミュレータによる仮想ネットワーク上で実験を行なった。全てのプロセスは、1 台の計算機 (BSD/OS 2.0.1) 上で実行した。仮想ネットワークの構成を図 7.1に示す。

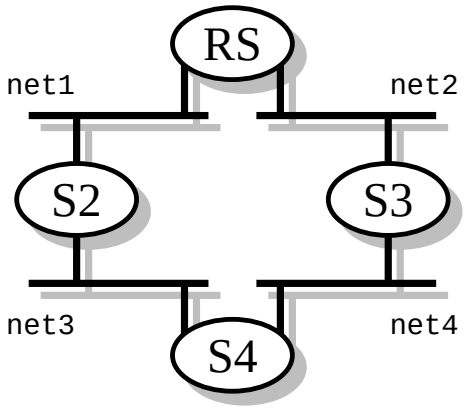

図 7.1: 仮想実験ネットワーク

次にホスト設定部のアドレス付け替え実験には通常のネットワークを用いた。実験ネットワークを図 7.2に示す。

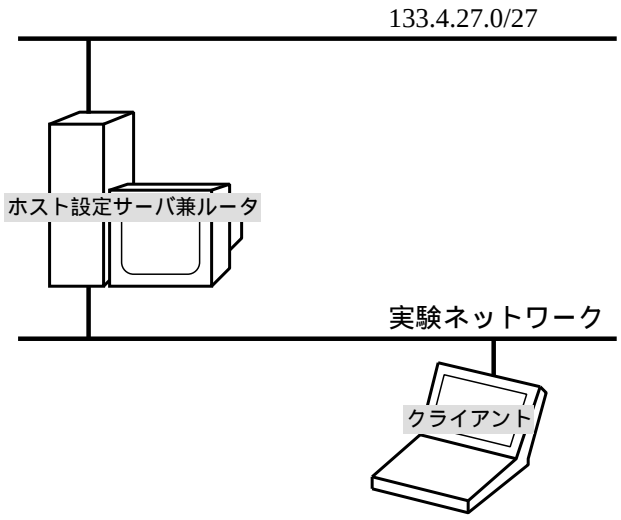

図 7.2: 実験ネットワーク

7.2 評価

7.2.1 サーバ木構成部の実行例

図 7.1の仮想実験ネットワークでは故意にループ状のネットワークを作成し、このようなネットワークでも階層構造が正しく構成されることを確認した。実行結果を以下に示す[1]。

```
 1  (tomy@nasu)% rncs conf.1 & ncs conf.2 & ncs conf.3 & ncs conf.4&
 2  rs> ne0 registration succeed
 3  rs> we0 registration succeed
 4  s2> ne0 registration succeed
 5  s2> we0 registration succeed
 6  s3> ne0 registration succeed
 7  s3> we0 registration succeed
 8  s4> ne0 registration succeed
 9  s4> we0 registration succeed
10
11  rs> start network configuration for net1
12  rs> start network configuration for net2
13
14  rs> advertise route to net1
15  rs> advertise route to net2
16  s2> receive RoutingMsg from net3
17  s3> receive RoutingMsg from net4
18  s4> receive RoutingMsg from net4
19  s4> receive RoutingMsg from net3
20  s2> receive RoutingMsg from net3
```

[1]見やすくするために順序を入れ替えるなどの編集をしてある。

```
21  s4> receive RoutingMsg from net4
22  s3> receive RoutingMsg from net4
23  s4> receive RoutingMsg from net3
24
25
26  s2> send UpperRequest from net1
27  rs> register lower s2
28  s2> receive UpperAck for net1
29
30  s3> send UpperRequest from net2
31  rs> register lower s3
32  s3> receive UpperAck for net2
33
34  s4> send UpperRequest from net4
35  s3> register lower s4
36  s4> receive UpperAck for net4
37
38  s2> start transmission of Designate message from we0(net3)
39  s2> determine that we0(net3) is Designated Interface
40  s2> start network configuration for net3
41
42  s3> start transmission of Designate message from we0(net4)
43  s3> determine that we0(net4) is Designated Interface
44  s3> start network configuration for net4
45
46  s4> start transmission of Designate message from ne0(net3)
47  s4> ne0(net3) is non-designated interface
```

実行結果についての説明を以下に示す。

- 1 行目で 4 つのサーバプロセスを起動している。ルートサーバだけは rncs という名前で、それ以外のサーバは ncs という名前で起動する。実行結果には含めなかったが、この前にエミュレータのプログラムを起動しておく必要がある。conf.* という引数はエミュレータ用ファイル名を指定する。これらのファイルの中にはネットワークインタフェースの仮想 MAC アドレスなどが記述されている。
- 2〜9 行目はネットワークエミュレータへのメッセージ登録を示す。
- 11,12 行目ではルートサーバがネットワーク設定部の処理を開始している。
- 14〜23 行目では経路情報の交換が行なわれている[2]。
- 26〜36 行目で上位サーバへの登録を行なっている。
- 38〜47 行目では Designated サーバの決定とネットワーク設定部の処理を開始している。

この結果を解析すると、図 7.3に示す階層構造が作られたことが分かる。ただし S4 は Designated なインタフェースを 1 つも持たないので、サーバとしては機能しない。

[2] これ以後も定期的に送信しているが、編集して削除した。

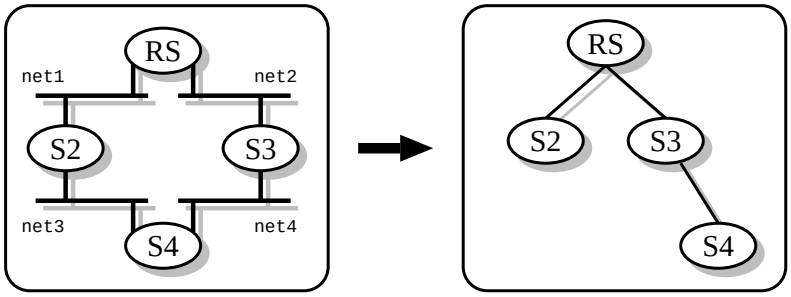

図 7.3: 仮想ネットワークに構築された階層構造

7.2.2 サーバ木構成部に関する考察

第 5.1節に示した設計目標の各々について考察する。

エミュレータによる仮想ネットワーク上の実験では「IP ルーティングや IP アドレスに依存しない」で動作することが確認できた。ネットワークのトポロジにループがある場合でも、設計通りに動作することが確認できた。今後は、より複雑なトポロジでも動作することや、あるいは現実のネットワークで動作することを確認する必要がある。

「素早く収束させることが可能である」という目標も達成することができた。例えば実験用の仮想ネットワークでは経路情報は 2 秒前後で収束する。収束までの所要時間についてはネットワークのトラフィックや計算機の性能によって異なると思われる。今回の実装では、初めて経路情報を受け取ってから 5 秒後には経路情報が安定すると仮定し、一時経路表から本経路表への複製を実行するようにした。しかしネットワークトポロジなどの条件によってはこの仮定が成立しない可能性があるため、今後さらに検証する必要がある。「最後に経路情報を受け取ってから 5 秒後に複製する」というようにアルゴリズムを変更してテストしてみる必要がある。

「階層構造の変動は少なく抑える」および「任意のサーバの起動が遅延しても矛盾しない」という目標に関して検証するために、故意に特定のサーバの起動を遅延させる実験を行なってみた。その結果、すでに確定したサーバの木構造には影響を及ぼさないことが確認できた。またサーバ木構成部の実装は約 600 行であるため「できるかぎり簡潔な機構にする」という目標も満たしたといえる。

最後の「少ない計算量で動作する」という目標については定量的に評価することができなかった。これについては今後の課題である。

7.2.3 ネットワーク設定部の評価および考察

サーバ木構成部と同じ仮想ネットワークで実験を行ない、基本動作を検証した。その結果、各サーバがアドレスバルクを上位サーバに要求し、ルートサーバから各々に対して割当が行なわれることが確認できた。またルートサーバを操作することにより、

下位サーバからアドレスを回収できることを確認した。

本実装では各サーバは自分自身で必要なアドレスバルクの大きさを決定すると直ちに割当要求を送信する。実験の結果、この方法ではトラフィック及び割当効率の点で無駄が生じることが分かった。すなわち起動時には自分自身よりも下流にあたる全てのサーバからの割当要求が届いた後に、上位サーバへ割当要求を送る方が「総メッセージ数」「アドレスの利用効率」などの点から考えて望ましい。しかし現在の実装方法では、個別に要求が送られるためにトラフィックに無駄が生じている。さらにアドレスバルクは要求のあった順に連続して割り当てるため、上位–下位関係にあるサーバの間でもアドレスプールが不連続になってしまうという問題が生じた。

「ネットワーク管理ポリシとプロトコルの分離」という目標は達成したが、今後は現実の多様な管理ポリシをうまく本システムで実現できることを確認する必要がある。ポリシとプロトコルの完全な分離は難しいため、ある種のポリシを実現するのに必要な機能がプロトコルに含まれていないという可能性は高い。さまざまなポリシに対応できるようにプロトコルの機能を拡張することが今後の課題である。

「アドレス付け替えの実現」についてもネットワーク設定部での支援機構は動作することを確認した。ただし今回実験したのは、ルートサーバから管理者が指示を与える「意図的な」アドレス付け替えについてのみであり、サーバがアルゴリズムに基づいて行なう「意図的でない」アドレス付け替えに関しては検証していない。

下位サーバが上位サーバにアドレス割当の要求を行なった場合に、Pending メッセージが返されることも確認した。これにより「メッセージの不必要な送信を防ぐ」ことができた。しかしながら AllocRequest を集約[3]する機能は実現できなかった。

「アドレスバルクの強制的な回収」については未検証である。

以上のようにネットワーク設定部の基本動作を確認することができた。しかしネットワーク設定ポリシを実現するためのアルゴリズムの設計が不適切であると、サーバ群全体も機能しなくなる可能性が高い。例えば下位サーバが上位サーバからの返却要求を受けてアドレスを返却した直後に、再び同じ領域の割当を求める可能性がある。今後は、アドレス分散管理アルゴリズムについて検証する必要がある。

最後に、以上の結果を表 7.1にまとめる。

表 7.1: ネットワーク設定部の評価

ネットワーク管理ポリシとプロトコルを分離する	○
ネットワーク管理者はルートサーバのみ操作する	○
アドレス付け替えを実現する	△ (意図的な付け替えのみ)
メッセージの不必要な送信を防ぐ	△ (一部のみ)
アドレスバルクの強制的な回収をサポートする	× (未検証)

[3] 下流からの AllocRequest も一括して申請する。

7.2.4 ホスト設定部の評価および考察

ホスト設定部については実際のネットワークを用いてサーバ、クライアントの動作確認を行なった。

「複数アドレスの設定」という目標は達成した。同時に「単一ホストのアドレスの付け替えを緩やかに実現」できることを確認した。すなわち古いアドレスでの通信を維持しつつ、新しいアドレスを用いた通信が可能なことを確認した。Advertise メッセージを用いたホスト設定は、その簡潔さにも関わらずうまく動作することが確認できた。「アドレスの強制回収」も実現することができた。強制回収に応じない誤ったクライアントも、ルータ側で経路情報のアナウンスを中止されると外部とは通信できなくなるため、他への影響を小さくすることができる。ただし同じネットワーク内の通信は、不当に継続してしまう。

「ネットワークを移動したことを検知する」ことも確認できた。ただし実際に移動してから、それを検知するまでに 30 秒前後必要とする。通常の LAN では移動の頻度がそれほど高くないので問題ないが、無線 LAN で用いる場合には問題がある。

DHCP などの既存プロトコルと比較して「プロトコルを簡潔にする」という目標は達成できている。最後にホスト設定部の設計目標の達成度を表 7.2 に示す。

表 7.2: ホスト設定部の評価

複数アドレスの設定	○
アドレスの強制回収	○
ネットワークを移動したことを検知	△ (検知に必要な時間に問題あり)
プロトコルを簡潔にする	○
アドレス付け替えの実現	○

第 8 章

結論

8.1 まとめ

本論文ではネットワークの自律構成およびアドレス付け替えの支援機構を実現するための手法として階層化モデルを提案した。この階層化モデルに基づいて実装したネットワーク自動設定機構は、計算機単体の設定のみならずネットワーク単位での設定を可能にした。このようなネットワークの自律構成、ネットワーク単位での自動設定およびアドレス付け替えを実現する機構は前例がなく、その点で本研究の意義は大きい。

本システムはサーバ木構成部、ネットワーク設定部、ホスト設定部の 3 つから構成されている。サーバ木構成部には Triggered Update の手法を取り入れた Distance-Vector 型経路制御アルゴリズムを適用し、階層構造の素早い構築を実現した。ネットワーク設定部はさまざまなネットワーク管理を実現できるようにポリシ独立に設計・実装し、エミュレータにより構成される仮想実験ネットワーク上で実験を行なった。実験の結果、階層構造に基づくアドレス管理を実現できることを示した。ホスト設定部はサーバがネットワーク情報を定期的に配布する手法を提案し、この手法がアドレスの付け替えや回収およびネットワークの移動を検知するのに有効であることを実証した。

8.2 今後の課題

8.2.1 短期的課題

今回の実装では検証できなかった事項で、比較的容易に解決あるいは実現できるものを以下に示す。

第一に実際のネットワークにおけるサーバ木構成部およびネットワーク設定部の動作確認が必要である。本論文ではエミュレータによる仮想ネットワークでこれらの動作を確認したが、何らかの問題を見落としていないか検証しなければならない。

第二にホスト設定サーバとネットワーク設定サーバのインタフェースを設計し実装する必要がある。ホスト設定部のモジュール性を高め、既存のホスト設定機構を利用できるようにするという設計目標を実現し、評価する。

第三にこれら全てを統合した完全な1つのシステムを、実際のネットワークにおいて試験運用しなければならない。ただしネットワーク管理アルゴリズムについては比較的単純なものを用いる。

その他にも「サーバ木構成に必要なトラフィックおよび演算量の評価」「意図的でないアドレス付け替えの発生頻度の測定」や「アドレスバルクの強制的回収に要する時間の評価」などを行なう必要がある。

8.2.2 長期的課題

第一にさまざまなネットワーク管理アルゴリズムを実装し、それらの性能を比較評価する必要がある。例えばアドレスの利用効率だけを重視するアルゴリズム、経路が集約可能になることを重視するアルゴリズム、アドレス付け替えをなるべく行なわないアルゴリズムなどを実装し、長所と短所を比較する。

また大規模ネットワークにおける運用実験を行ない、トラフィック解析やアドレス付け替えの所要時間などの性能評価を行なう必要がある。

最後に本論文と直接の関係はないが、本研究の最終目的である「大規模ネットワークの自律構成および自動管理」を達成するために解決しなければならない問題を示す。

- **各種サーバの自動設定**

  インターネット上には多様なサービスが存在するが、それらを提供するサーバの設定方法は各々で異なる。そのようなサーバの設定を自動化することが必要である。

- **通信のアドレス付け替え透過性の実現**

  ネットワークアドレスの付け替えが起きると、それまでに確立した論理的通信路は切断されてしまう。同様の問題が計算機の移動にともなって発生するが、それらに対する解決策がいくつか提案され、実現されている [26]。本論文では緩やかなアドレス付け替えを行ない、論理通信路の切断を実用に耐える範囲内に抑えることを目指した。しかしアドレス付け替えに要する時間が短いほど、自由なネットワーク管理ポリシ (アドレスの利用効率を最大に高める、経路の集約を目標とする) が実現しやすい。そのためにはアドレスの付け替えの前後を通じて透過的に通信が行なえることが望ましい。

- **ルータの自動設定**

  本論文ではルータの完全な自動設定については取り扱わなかった。すなわちネットワークアドレスに基づいて経路情報のフィルタリングを行なうような複雑な状況は考慮していない。Default route のみの設定や、RIP[24], 単一の AS[1] 内における OSPF[22] などの簡単なものについて取り扱うに留めた。今後はルータの自動設定についても実現する必要がある。

[1] Autonomus System(自律システム) の略。この場合、単一の運用方針に基づいた経路制御が行なわれているネットワークドメインを意味する。

- **“名前-ネットワークアドレス” マッピングの自動更新**

  現在のインターネットにおいては、通信を開始する場合に DNS(Domain Name System)[11, 12] を用いてホスト名から IP アドレスを取得するのが一般的である。ネットワークアドレスの設定や付け替えを行なった場合には DNS に登録されている名前とネットワークアドレスの対応を更新する必要がある。このような動的更新の実現方法と、それにともなって生じる「キャッシュの一貫性」に関する問題についても取り扱わなければならない[2]。

[2] この問題に関しては IETF の DNSIND WG で解決が試みられている。詳細は “Dynamic Updates in the Domain Name System (DNS)” というドキュメントに記されている。これは Internet-Draft と呼ばれる 6 カ月で消去されるドキュメントなので参考文献には含めない。

謝辞

本研究を進めるにあたり、御指導頂きました慶應義塾大学環境情報学部の徳田英幸博士と村井純博士に、また楠本博之博士と中村修博士の両氏に感謝します。論文に対してコメントしてくださった同大学同学部の武藤佳恭博士にも感謝します。多くの御助言・御指導・激励を頂きました (株) ソニーコンピュータサイエンス研究所の寺岡文男博士に感謝します。最後に本研究を進めていく上で相談にのって頂いた MUC Working Group をはじめとする WIDE Project の諸氏に感謝して謝辞と致します。

参考文献

[1] Mark, W.: *The Computer for the Twenty-First Century*, Scientific American, pp. 94–104, September 1991.

[2] Gerich, E.: *Guidelines for Management of IP Address Space*, RFC 1466, May 1993.

[3] Fuller, V., Li, T., Yu, J., and Varadhan, K.: *Classless Inter-Domain Routing(CIDR): an Address Assignment and Aggregation Strategy*, RFC 1519, September 1993.

[4] Deering, S., and Hinden, R.: *Internet Protocol, Version 6 (IPv6) Specification*, RFC 1883, December 1995.

[5] Rekhter, Y., and Li, T.: *A Border Gateway Protocol 4 (BGP-4)*, RFC 1771, March 1995.

[6] Simpson, W.: *The Point-to-Point Protocol (PPP)*, RFC 1661, July 1994.

[7] Romkey, J.: *Nonstandard for transmission of IP datagrams over serial lines: SLIP*, RFC 1055, June 1988.

[8] Koblas, D., and Koblas, M.R.: *SOCKS*, Proceedings of USENIX UNIX Security Symposium III, pp. 77–83, December 1995.

[9] Rekhter, Y., Moskowitz, R., Karrenberg, D., and de Groot, G.: *Address Allocation for Private Internets*, RFC 1597, March 1994.

[10] *SunOS Reference Manual — yp - NIS protocol*, Sun Microsystems, Inc.

[11] Mockapetris, P.: *Domain Names - Concepts and Facilities*, RFC 1034, November 1987.

[12] Mockapetris, P.: *Domain Names - Implementation and Specification*, RFC 1035, November 1987.

[13] Finlayson, R., Mann, T., Mogul, J., and Theimer, M.: *A Reverse Address Resolution Protocol*, RFC 903, June 1984.

[14] Plummer, D.: *Ethernet Address Resolution Protocol: Or converting network protocol addresses to 48.bit Ethernet address for transmission on Ethernet hardware*, RFC 826, October 1982.

[15] Brownell, D.: *Dynamic RARP Extensions for Automatic Network Address Acquisition*, Sun Microsystems, Inc., September 1989.

[16] Schiller, J. and Rosenstein, M.: *A Protocol for the Dynamic Assignment of IP Addresses for use on an Ethernet*, MIT Athena Project, 1989.

[17] Croft, W. and Gilmore, J.: *Bootstrap Protocol*, RFC 951, September 1985.

[18] McGregor, G.: *The PPP Internet Protocol Control Protocol (IPCP)*, RFC 1332, May 1992.

[19] Droms, R.: *Dynamic Host Configuration Protocol*, RFC 1541, October 1993.

[20] Tominaga, A., Osamu, N., Teraoka, F., and Murai, J.: *Problems and Solutions of DHCP*, Proceedings of INET'95, Vol. 1, pp. 481–490, June 1995.

[21] 冨永 明宏, 寺岡 文男, 村井 純: 「動的ホスト設定プロトコル (DHCP) の実装と評価」, 情報処理学会 マルチメディア通信と分散処理研究会 論文集 Vol.93, No.2

[22] Moy, J.: *OSPF Version 2*, RFC 1247, March 1994.

[23] Pummill, T., and Manning, B.: *Variable Length Subnet Table For IPv4*, RFC 1878, December 1995.

[24] Hedrick, C.: *Routing Information Protocol*, RFC 1058, December 1995.

[25] McCanne, S. and Jacobson, V.: *The BSD Packet Filter: A New Architecture for User-level Packet Capture*, 1993 Winter USENIX conference proceedings.

[26] Teraoka, F., Uehara, K., Sunahara, H., and Murai, J.: *VIP: A Protocol Providing Host Mobility*, CACM, Vol. 37, No. 8, pp. 67–75, August 1994.