1997 年度卒業論文

次世代ネットワーク環境をサポートする名前サービスの実現方法に関する研究

指導教官
徳田 英幸 ・村井 純
楠本 博之 ・中村 修

環境情報学部 79452949
村井研究室
土井裕介

概 要

名前サービスとは、利用者向けの識別子であるドメイン名と、コンピュータが通信の識別に利用するネットワークアドレスとを対応付けるサービスである。現在、もっとも一般的な広域ネットワークであるInternetで使われている名前サービスは、DNS(Domain Name System)と呼ばれる。しかし、DNSは設計が古く、現在のInternetの環境に対応できない面が存在する。

本研究では、特に近年発達した動的なネットワーク環境と次世代のInternetを実現するプロトコルIPv6に焦点をあてる。動的な名前サービスを中心とする、DNSに関連して開発が進められている技術を応用し、次世代のInternetに適応するための名前サーバを実装した。併せて、動的な名前サービスを効率的に利用する為の周辺システムを実装した。

Abstract

The name service matches domain name with network address. Both of domain name and network address identify network node. Users use domain name to identify nodes in most case, but nodes always use network address to identify themselves. DNS(Domain Name System) is the name service system for the Internet, the most popular distributed network. But, DNS was designed so many years ago and can't serve well for contemporary envrionment of the Internet.

The main focus of this research is on the dynamic network environment and IPv6, the protocol to make next generation of the Internet. A name server was implemented with ability of some technologies around the DNS, such as dynamic name service. And support system to make use of such dynamic name service was also implemented.

目次

図目次

表目次

第1章　はじめに

Internet は、現在最も普及している広域分散環境である。その Internet 上で重要かつ複雑な役割をしているといわれる仕組みが 3 つ存在する。一つは、信頼性の低いネットワークを用いつつ、できる限り信頼性の高い通信を行う TCP(Transmission Control Protocol) である。もう一つは、複雑に広がったネットワーク上のデータを正しく経路付け、正しい最終目的地へ届ける、経路制御技術およびそのためのプロトコル群である。そして最後の一つが、名前を管理する DNS(Domain Name System) である。

現在用いられている DNS の基礎は、1987 年に設計 [7][8] されたものである。だが、時代の移り変わりとともに、現在のネットワーク環境にそぐわない点が多く出てきた。具体的には、ネットワークが動的に変化するようになったことと、ネットワークの規模が飛躍的に増大し、従来の枠組では支え切れなくなったことである。

また、従来の DNS のデータベースは静的で、その管理はテキストファイルと手作業に基づいている。これでは、現在の環境には十分でない。次世代のネットワーク環境を考えた修正が必要となる。

そこで本稿では、以下の二つの新しい環境に DNS を適応させる。

- 動的なネットワーク環境
- IPv6(Internet Protocol version 6)

具体的には、動的な環境に適応するための動的なデータベースの実装と、そのデータベースを外部から更新するプロトコルの実装である。また、外部からの更新を利用して、ノードの設定と同時に名前データベースを更新させる試みを行う。これにより、次世代のインターネットプロトコル IPv6 の管理による負担を軽減することを目的とする [11]。

1.1　研究目的

研究の最終的な目標は、DNS を基礎としたより包括的なデータベースシステムを構築することである。このデータベースシステムには、ノードのみではなく、利用者の位置情報やその他の接続性に関する情報、さまざまな知識や資源に関する情報を含む。それらの情報を、DNS レコード検索という単純な手法で入手するシステムである。

今回の研究では、そのための道筋の一つとして、動的名前サービスを中心とした、DNS のサービスの実装を行なう。特に、従来の DNS では考慮されていなかった、動的環境へ適応できる名前サービス環境と、IPv6 ネットワークの運用で発生する問題に対処するための実装を行ない、評価する。

1.2　論文の構成

まず第 2 章で、研究対象のシステムの基礎となる、DNS について詳しく述べる。とくに DNS の動作原理と、reverse lookup の仕組みと問題、動的名前サービスについて詳しく述べる。

次に第 3 章で、今回開発した newbie-I の設計について述べ、第 4 章でサーバに相当するコアシステム、第 5 章でクライアントに相当する補助システムそれぞれの実装について詳細に述べる。

第6章では、評価実験について述べ、第7章で考察を行なう。最後に第8章で、今後の課題をまとめ、結論とする。

第2章　DNS – Domain Name System

本章では、Internet で名前データベースを分散管理している DNS の、必要性、概略、利点、欠点、新技術について述べる。

2.1　広域分散環境における名前サービスの役割

広域分散環境では、ノード相互の識別を行なわなければ通信できない。一般に、ノードにはそれぞれ一意となる識別子 (アドレス) を割り当てる。しかし、アドレスは利用者にとって覚えやすい形式ではない。そこで、利用者に分かりやすい名前を付ける必要がある。アドレスへの分かりやすい名前の割り当ては、広域分散環境を用いる上で必須の機能である。

2.2　DNS 概略

Internet 上の名前は、単一の名前空間を分割する木構造を持つ。その木構造のデータベースを保持するために、DNS では木のそれぞれの節を担当するサーバを関係付けている。

まず、DNS の重要な概念や、構造、動作の流れに関して述べる。

2.2.1　名前の構造

Internet 上の名前は、ピリオドで区切られた一連のラベルから構成される。例えば “bono.sfc.wide.ad.jp” という形で表される。そして、ラベルは右側が大きな集合を表し、左側が小さな集合を表す。上の例では、“bono.sfc.wide.ad.jp.” という名前は、“jp” という集合に含まれている。そして、“jp” に包含される “ad” に含まれ、さらに “ad” の中に存在する “wide” に含まれている。

それぞれの集合をドメインと呼ぶ。また、それぞれのドメインに付けられた名前をドメイン名と呼ぶ。ドメイン名は、そのドメインから最上位のドメインまで全てを連ねてはじめて一意になる。したがって、ドメイン名を利用する場合は、一部の例外を除いて右側の部分は省略できない。

例外は、リゾルバと呼ばれる利用者側のクライアントが右側の部分を補完する場合であり、名前サーバへの問い合わせは必ず全てのラベルを連ねたドメイン名で行なう。また、このような完全な形のドメイン名を特に FQDN(Fully Qualified Domain Name) と呼ぶ。

2.2.2　木構造のドメイン

ドメインは、図 2.1 のような木構造である。名前データベースもドメイン構造の射影なので、この木構造になる。それぞれのドメインは、自分の下位ドメインとなるドメインについての情報を持つ。また、全ての名前サーバは、木の根についての情報を持つ。この木の根に相当するドメインを、ルートドメインと呼び、ルートドメインについてのサービスを行なっている名前サーバを特にルートサーバと呼ぶ。ルートドメインから末端方向にたどると、DNS 上の全ての情報が入手可能になる。

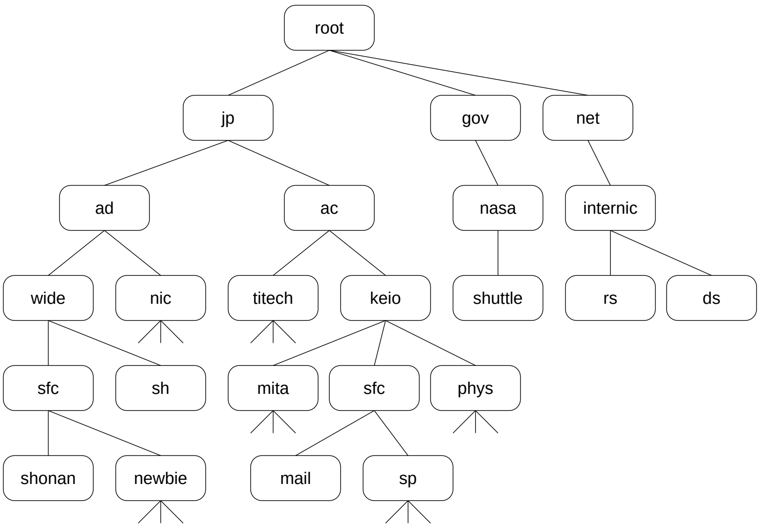

図 2.1: ドメインの構造 (一部のみ)

2.2.3 名前サーバ

それぞれのドメインには名前サーバが一つ以上存在する。一つの名前サーバは、複数のドメインを受け持つことができる。そのドメインを担当するネームサーバを、ドメインに権限がある (authorative) と言う。権限がある名前サーバは、データベースを管理し、必要があれば障害対策のために複製を作成し、問い合わせがあれば返答する。

名前サービスは、ネットワークにとって非常に重要なサービスである。したがって、単一のサーバに依存すると、そのサーバが利用できない場合、ネットワーク全体が影響を受ける。このような事態を防止するために、データベースの複製を持つ名前サーバをおくことができる。これらのサーバを二次サーバと呼ぶ。これに対して、データベースの原本となるゾーンマスターファイルと呼ばれるファイルを持つサーバを、一次サーバと呼ぶ。

また、名前サーバは、そのサーバ自身が権限を持たない情報を問い合わされた場合、同等の問い合わせをより適切なサーバに対して行い、得た答えを問い合わせ元に返答しても良い。この過程を再帰問い合わせと呼ぶ。この過程で得た情報は、定められた時間だけ記憶できる。また、記憶している情報に対する問い合わせは、その記憶に基づいて返答を行う。この記憶をキャッシュと呼ぶ。

名前サーバは、問い合わせられた情報に付随する情報を、返答に含めるべきである。これを追加データ処理 (Additional Data Processing) と呼ぶ。

名前サーバがドメイン木を構成するには、下位ドメインの情報を持つ必要がある。具体的には、下位ドメインの名前サーバのドメイン名を持つ。これにより、上位ドメインから下位ドメインへの橋渡しが可能になり、名前サーバによりドメイン木を構成できる。

そして、全ての名前サーバはヒントと呼ばれる、ルートサーバについての情報を持つ。ヒントによって、名前サーバは木の根を調べられる。木の根から末端に向かってたどることで、名前サーバは全ての名前空間を調べる。

2.2.4 リゾルバ

リゾルバとは、プログラムの内部でDNS問い合わせを行う部分を指す。一般にライブラリ関数として実装される。

リゾルバは、DNSの問い合わせを行い、返答を呼び出し元のユーザプロセスに戻す。この際、リゾルバは最も近いと思われる名前サーバを一つ以上知っていれば良い。必要な情報は、その名前サーバが再帰問い合わせによって収集する。

リゾルバも再帰問い合わせを行い、キャッシュを持てる。しかし、キャッシュは集約されている方が効率が高い。したがって、ネームサーバにキャッシュを集約するために、リゾルバは単純な問い合わせのみを行うことが多い。

2.2.5 データの形 – resource record

DNSは、登録されたデータをresource record(RR)と呼ばれる単位で管理する。RRは、ドメイン名、クラス、型、データから成り立つ。ドメイン名はそのデータが所属するドメインの名前である。クラスはデータが属するプロトコルだが、今回はIN(Internet)クラス以外取扱っていない。型は、データの形式を示し、DNSの仕様を定めたRFC[8]などで定義されている。RRのデータは、型によって異なった形式を持つ。

今回の実装で取扱うリソース型について以下に述べる。A / NS / CNAME / SOA / PTR / HINFO / MX / AAAA と呼ばれるレコードである。

A(Address)レコードは、IPv4アドレスを格納するレコードである。データは32bitで、IPアドレスを直接格納する。

NS(Name Server)レコードは、ドメインのネームサーバを示すレコードである。データはドメイン名で、可変長である。NSレコードの検索は、追加データ処理を引き起こす。名前サーバは、NSレコードのデータのドメイン名が示すドメインのAレコード(IPv4アドレス)及びAAAAレコード(IPv6アドレス)を、可能ならば応答に加えるべきである。

CNAME(Canonical Name)レコードは、そのドメインの正式名を示すレコードである。CNAMEレコードが所属するドメイン名は、CNAMEレコードのデータが指定したドメインの別名であると示す。CNAMEレコードが存在するドメインには、他のリソースレコードは存在できない。また、NSや、PTR、SOAなどのレコードデータとして、CNAMEレコードを指定できない。CNAMEレコードのデータは、ドメイン名で、可変長である。特定のドメインの特定のCNAME以外のレコードを検索して、同じドメインにCNAMEレコードが存在した場合、名前サーバはCNAMEの指し示すドメイン名を調べる。もしそこに検索すべき対象とレコード型とクラスが一致するレコードが存在すれば、そのレコードをもって検索要求に対する答えとする。

SOA(Start Of Authority)レコードは、ドメインに一つだけ存在するレコードである。データは二つのドメイン名と、五つの32bit長整数である。二つのドメイン名は、ドメインの管理責任者のメールアドレス(但し@マークはピリオドで置き換える)と、ゾーン全体に影響する値五つである。SOAの持つ五つの値はそれぞれ、serial/refresh/retry/expire/minimumと呼ばれる。serialはデータベースの通し番号で、二次サーバはこの値をもとにデータベースが更新されたかを調べる。refreshは、二次サーバが、マスターサーバのデータベースが更新されたかを調べる時間間隔を記述している。retryは、refreshでマスターサーバが利用不可能だった時、再試行を行なう時間間隔である。expireは、データベースが更新できなかった場合、データ無効にするまでの時間を記述している。minimumは、厳密にはゾーン全体でのRRの生存時間の最小値だが、実際はRR生存時間の省略値としている実装が多い。

PTR(PoinTeR)レコードは、reverse lookupで用いられる。reverse lookupについては詳しくは後述す

るが、IP アドレスから対応するホスト名を得るための手法である。データは、IP アドレスに対応するドメイン名を格納する。

MX(Mail eXchanger) は、メール交換サーバを示す。データは 16bit の優先度と、メール交換サーバのドメイン名である。MX は主に sendmail などによるメール交換に用いられる。MX は追加データ処理を引き起こす。名前サーバは、示されたメール交換サーバの A レコード及び AAAA レコードを返答に含めるべきである。

HINFO(Host INFOrmation) は、そのドメイン名が割り当てられているホスト (ノード) の情報を書き込む。情報は可変長であり、CPU とよばれる、ハードウェアの情報と、OS と呼ばれる、ソフトウェアの情報から成る。HINFO は、主に管理者が管理の都合で記入する物であり、必須ではない。

AAAA(quad-Address) レコードは、IPv6 のアドレスを格納する。データ部は 128bit の整数であり、A レコードの 4 倍である [13]。

2.2.6 メッセージ形式

DNS の通信に用いられるメッセージの形式は複雑である。通信に用いるトランスポート層プロトコルは、TCP(Transmission Control Protocol) と UDP(User Datagram Protocol) の両方を利用する。UDP では、IP の通信で保証される最小限のデータ長が 512byte なので、メッセージ長も 512byte に制限している。データがそれ以上の長さになる場合は TCP を用いる。

DNS は、メッセージは以下の五つの部分から成る。

- Header 部
- Question 部
- Answer 部
- Authority 部
- Additional Record 部

Header 部は、そのほかの各部のレコードの数や、メッセージの性質を表す数値などがまとめられている。ヘッダ部の構造を図 2.2 に示す。

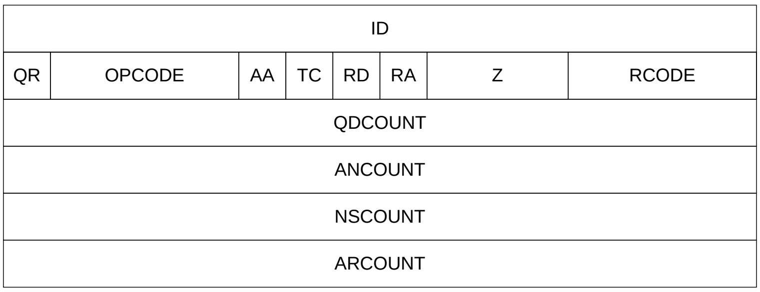

図 2.2: DNS メッセージ:ヘッダ部

ID は、要求メッセージと応答メッセージで同一の物を用い、どの応答がどの要求に対するメッセージなのかを識別する。

QR は、このメッセージが要求 (0) か応答 (1) かを示す。

OPCODE は、メッセージが何に対する要求なのかを示す 4bit のフィールドである。

AA は、応答が権限を持つ (1) か、それともキャッシュなどから学んだデータを応答した (0) のかを示す。

TC が 1 ならば、UDP のパケット長制限により、パケットが短縮させられたことを示す。

RD が 1 ならば、要求元が、名前サーバに対して再帰問い合わせを行なうかを要求することを示す。

RA が 1 ならば、応答するサーバが、再帰問い合わせを行なうかどうかを示す。RA が 0 の場合は、再帰問い合わせを行なわない。

Z は、将来の使用のために予約されている。0 でなければならない。

RCODE は、応答のエラーを示す。0 ならばエラー無しであり、0 でなければエラーの発生を示す。

以降 4 つの 16bit の数値は、それぞれ question 部、answer 部、authority 部、additional record 部がいくつづつ存在するかを示す。

question 部は、要求するデータのドメイン名、クラス、型を持つ。図 2.3 に question 部の構造を示す。

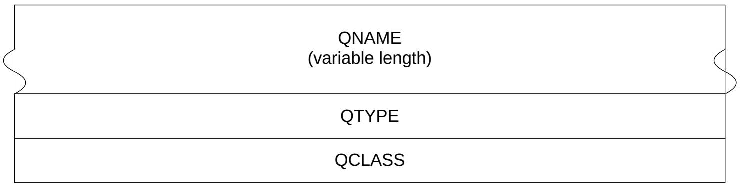

図 2.3: DNS メッセージ:question 部

answer 部、authority 部、additional record 部はそれぞれ、ヘッダに示された数だけの連続した RR である。

RR の形式を図 2.4 に示す。

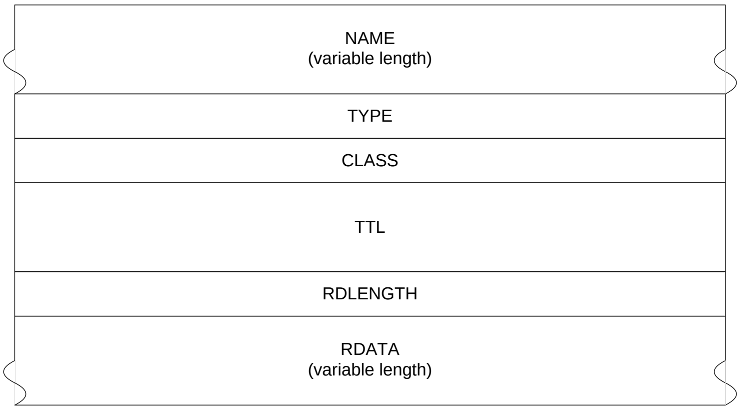

図 2.4: DNS メッセージ:resource record

2.3 DNS の利点

Internet は、現在は全世界に広がるネットワークとなった。そのネットワーク上の全てのノードに関しての情報を、適切に素早く取り扱えるシステムは、現在のところ DNS のみである。ここでは、DNS の利点について定性的に述べる。定量的な解析は現在のところあまり行われていない。

2.3.1 規模性

DNS は、Internet 上の全てのノードに関する情報を分散的に保持している。このことは、DNS の優れた規模性を表すものである。

DNS の規模性を向上させている重要な要素は、トランスポート層に可能な限り UDP を使っていることと、キャッシュの存在である。

DNS では、メッセージ長が 512byte を越えない限り、UDP を用いて通信を行なう。TCP はそのコネクション確立の際に、複数の IP データグラムを必要とし、またトランスポート層でコネクションの状態を保持する必要がある。UDP を用い、それらコストを免れる。

また、名前サーバはキャッシュ機能を実装しており、一度再帰問い合わせを行なったレコードに対しては、キャッシュが有効な間、再び問い合わせる必要が無い。キャッシュの有効時間は、そのレコードの変更頻度に基づいて管理者が決定する。

キャッシュの一つとして、ネガティブキャッシュとよばれる機能がある。ネガティブキャッシュは、誤った検索の乱発を防止している。すなわち、検索したドメイン名やレコードが確かに存在しないと分かった場合、名前サーバは「存在しない」というキャッシュエントリを作成する。同じレコードに対する二度目以降の問い合わせに対しては、そのキャッシュエントリを元に「存在しない」という返答するので、不要な問い合わせを避けることができる。

NSFNet のバックボーントラフィック上の DNS メッセージを解析した研究がある [3]。この研究によって、DNS トラフィックの無駄の大部分は、実装上の誤り (bug) に起因する物だと明らかになっている。そしてネガティブキャッシュは、その様な誤りから発生する大量の要求に対して、防火壁の役割を果たす。

ゾーン転送と二次サーバも、DNS の規模性を高めている要因である。名前サーバの負荷が高いゾーンは、二次サーバを増やして対応できる。二次サーバへのゾーン転送の負荷が高い場合は、差分転送 [14] を用いて負荷を軽減できる。また将来的には、IPv6 で標準となる anycast 機能を用いて、最も近いネームサーバへの通信を実現し、ネットワークへの負荷の軽減を図ることができる。これらの技術を用いて、DNS の規模性を保ち、改善することができる。

2.3.2 分散管理

DNS の利点の一つに、分散管理が可能という点がある。名前サーバのデータベースは、それぞれのドメインの管理者が管理する。したがって、その管理者によって、管理するドメインの情報を反映することは簡単である。

もしも、whois サービスの様に、特定の組織が集中管理していた場合、データベースの更新が怠られ、データベースが役に立たなくなるという結果になる。しかし、DNS の場合は、全世界のドメインの数だけサーバを設定でき、それぞれ独自に管理できる。したがって、Internet が現在の規模に発展した今も、名前情報は全世界的に新鮮に保て、ネットワークのどのドメインとの通信も円滑に行われている。

2.4 従来の DNS の問題点

Internet 上で、コンピュータ同士の相互識別を支えている DNS は、設計が 1987 年と古く、これまでの運用から欠点がいくつか明らかになった。ここでは、データベースの更新の問題と、管理上の問題に焦点を当てる。

2.4.1　データベースファイルの更新

従来の DNS データベースは、ASCII 形式のテキストファイルで書かれたゾーンマスターファイルに基づいている。そして、データベースの更新の際は、以下の二つの作業を行なう。

1. 管理者がエディタを用いてゾーンマスターファイルを更新
2. ネームサーバがゾーンマスターファイルを読み直す (reload)

設計段階ではデータベースの更新は頻繁に起きるものではなかったので、この形式で十分であった。しかし、動的にネットワーク環境の設定を行うシステムがいくつか作られネットワーク環境が動的になると、データベースが静的ではいくつかの不都合が生じる。

従来の DNS には、動的に設定されたネットワーク環境の情報を、DNS の名前データベースに反映させる方法が無い。したがって、動的に設定されたノードを、そのノードの名前で識別できない。また、ゾーンマスターファイルが ASCII テキスト形式なので、頻繁な更新による挿入や削除に適さない。

2.4.2　管理者の負担

従来の DNS では、データベースは全てゾーンマスターファイルに依存していた。そして、一般のユーザはゾーンマスターファイルに対する書き込み権限を持っていないので、データベース変更は全て管理者が行う。つまり、ネットワーク管理者に負担が集中する。

また、新しいノードの導入の場合も、管理者は新しいノードにアドレスを設定した後、そのアドレスをもう一度ゾーンマスターファイル中に書かなければならない。

外部から DNS のデータベースを変更する手段があれば、これらの問題を軽減できる。しかし、変更手段が無いため、仮にアドレスが自動設定されたとしても、そのアドレスを DNS のデータベースに手作業で書き写すことになる。また、手作業でノードを設定したとしても、その設定を DNS データベースに反映するために、二度同じ内容を入力しなければならない。

このような環境で IPv6 を導入した場合、管理者は、いままで IPv4 のアドレスを入力していた頻度と同じだけの頻度で、128bit もの長さの IPv6 アドレスを入力しなければならない。つまり、IPv6 の特長の一つの自動設定も、このような環境では十分な効果を発揮できない。

2.5　新しい技術

このように、DNS は優秀なシステムだが、問題点も存在する。いままで挙げてきた問題点、あるいはそれ以外の問題点に対する解決案となり得る新技術がいくつか提案されている。ここでは、それらの新しい技術に関して述べる。

2.5.1　クライアント側からの動的な更新

従来の DNS には、外部から名前サーバのデータベースを更新する手段が無かった。しかし、外部からの更新の需要は高く、IETF でも議論の対象になっている。その結果、Dynamic UPDATE と呼ばれるプロトコルが定義された [19]。Dynamic UPDATE を用いて、クライアント側からの名前サーバのデータベース更新が可能である。動的更新については 2.7 節で詳しく述べる。

2.5.2 更新通知

名前サーバには、障害対策などの理由で、データベースの複製を持つ機能がある。複製を持つサーバ、すなわち二次サーバは、適当な時間間隔で、複製の元のデータベースが更新されていないかを調べる。更新されていた場合のみ、そのデータベースを再度複製する。

実際に動的な環境に対応する名前サービスを行う場合、データベースが頻繁に変更される。しかし、データベースの変更は、そのままでは二次サーバに反映されない。二次サーバからの問い合わせがあってはじめて、変更を反映できるようになる。言い替えれば、二次サーバからの問い合わせがあって更新が反映されるまで、二次サーバの利用者は古いデータを用いなければならない。

このような状況を防ぐために、データベースの更新通知を行う仕組みが提案された。その仕組みはDNS NOTIFYと呼ばれている[15]。

これは、データベースの複製元となるサーバ(マスターサーバ)が、それぞれの複製を要求したサーバ(スレーブサーバ)に対して、更新通知メッセージを送信するという仕組みである。更新通知のために、マスターサーバは自分のスレーブサーバを記憶し、その記憶に従って更新通知を行う。更新通知を受け取ったスレーブサーバは、マスターサーバからより新しいデータベースを入手する。

一次サーバは全てマスターサーバである。しかし二次サーバは、スレーブサーバであると同時に他の二次サーバのマスターサーバにもなることができる。

DNS NOTIFYにより、二次サーバは常に最新のデータを持てるだろう。

2.5.3 差分転送

DNSのマスターサーバ、スレーブサーバの間では、データベースが更新される度に、そのデータベースをネットワーク経由で複製する。これをゾーン転送と呼ぶ。しかし、更新される度にデータベースの全てを転送する必要があり、効率が悪い。

その上、DNS NOTIFYが実用になっている場合、何らかの変更が一次サーバに加えられる度に、サーバの数だけのゾーン転送を行うのでは、ますます効率が悪くなる。

そこで、データベースの更新差分のみを転送する技術が考案された。これはに、差分ゾーン転送(Incremental Zone Transfer)と呼ばれる[14]。

それぞれの名前サーバは、データの更新をある程度記録しておき、ゾーン転送要求があった場合、転送元のデータベースと転送先のデータベースの差分に相当する部分のみを転送する。

この技術により、従来のゾーン転送に比べて、ネットワークへの負荷が低減する。

2.5.4 DNSへのセキュリティ拡張

DNSは従来、セキュリティの概念をあまり必要としなかった。しかし、Dynamic UPDATEを現実に適用するためには、セキュリティの概念を無視できない。DNSのセキュリティ拡張を、DNSSecと呼ぶ[16]。

DNSSecでは、DNSを用いて、公開鍵の配送をおこない、かつ、DNSサーバによる暗号署名を可能にする。暗号アルゴリズムはRSA公開鍵暗号系を用いている。DNSsecを用いて、DNSのメッセージの同一性が保証でき、認証を行なうことができる。

2.6 reverse lookup と IPv6

DNSの重要な役割は、「名前」と「アドレス」の対応付けである。DNSのデータベースはInternetでの名前に対応する構造になっている。従って、データベースを調べて、名前から登録してあるアドレスを

知る。

しかし、Internet 上のアドレスは、“203.178.140.101” や、“5f09:c400:cbb2:8c03::1” といった表記法であり、DNS で検索のためのインデックスに用いられるドメイン名の形とは異なる。したがって、アドレスはそのままではデータベースの検索インデックスとして使用できない。

ここでは、DNS でアドレスからノードの名前を検索する際に用いられる reverse lookup という手法について述べる。あわせて、IPv6 での reverse lookup で想定される問題について論じる。

2.6.1 reverse lookup とは

reverse lookup は、直接 DNS に登録できない形のアドレスを、DNS データベースに登録するための手法である。reverse lookup は、アドレスを特定の手法で加工して、アドレスに一対一で対応するドメイン名を作る。そのドメインに、アドレスに対応するノードの名前を登録しておく。

IPv4 アドレスに一対一で対応するドメイン名には、最後に “in-addr.arpa.” というラベルが付く。そして、その左に IPv4 アドレスを、左右を入れ換えて配置する。IPv4 アドレスでは、最も左側の数値が最も大きい単位を表し、右に行くにつれ小さい単位を表す。しかし、ドメイン名は逆である。従って、IPv4 アドレスをドメイン名に変換するためには、数値の左右を入れ換える必要がある。

具体例を挙げる。203.178.140.101 の IPv4 アドレスを持つノード名を知りたいとする。この場合、リゾルバは、“203.178.140.101” のアドレスを “101.140.178.203.in-addr.arpa.” というドメイン名に変換する。変換後の文字列はドメイン名であり、対応する名前サーバが存在する。その名前サーバは、ドメイン名に対応する情報として、“203.178.140.101” の IPv4 アドレスを持つノードの名前を持つ。仮に “bono.sfc.wide.ad.jp.” が登録されていたとする。リゾルバはそのドメイン名に関する情報を、203.178.140.101 に対応するノード名であると理解する。

2.6.2 IPv6 での reverse lookup

IPv6 では、IPv4 で行われていた形の reverse lookup をほぼそのままの形で引き継ぐことになっている。ただし、IPv6 のアドレスは 8 組の 4 桁の 16 進数をコロンで区切り、さらに途中の 0 の省略を許すという複雑なものであり、全くそのままでは引き継げない。

例えば、“5f09:c400:cbb2:8c03::1” は、途中の “::” が 0 の省略形である。全く省略をせずに記述した形式を、以下に示す。

“5f09:c400:cbb2:8c03:0000:0000:0000:0001”

また、この IPv6 アドレスに直接対応するドメイン名は、以下の通りである。

“1.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.3.0.c.8.2.b.b.c.0.0.4.c.9.0.f.5.ip6.int.”

すなわち、アドレスの全ての桁をピリオドで区切り、左右を入れ換え、最後に “ip6.int.” をつけたものである。

2.6.3 考えられる問題

これまで述べたように、IPv6 での reverse lookup は最大 34 段階の名前データベースの木を持つ。この reverse lookup の木に関して二つの問題がある。一つは管理上の問題であり、もうひとつは lookup のコストの問題である。

従来の IPv4 の木に比較しても IPv6 の reverse lookup の木は非常に大きく、その管理は繁雑になるだろう。今のところ、IPv6 のアドレス構造には、経路制御の集約化という問題を考慮した構造になろうとしている。このアドレス構造は現段階では IETF で議論中である [20]。このような構造の場合、例えば現在の案では、一つの管理主体 (サイト) が 16bit の SLA(Site Local Aggrigator) という値と、64bit の Interface ID を持つ。SLA はサイト内の経路制御に用い、Interface ID は終点ノードの識別に用いる。

あるサイトの管理者は、場合によってはこれら全てを管理しなければならない。特に Interface ID の部分は 64bit、ラベル 16 個に相当する。

また、reverse lookup は、最大 34 階層の名前データベースの検索である。前述の経路の集約化を考慮したアドレス構造では、アドレスを 5 つ程度の部分に分ける。DNS では、一つのデータベースの中に複数階層の名前構造が存在できる。従って、このような場合は reverse lookup も IP アドレスの 5 階層に加えて "ip6.int." の 2 階層で、合計 7 階層を調べれば十分である。しかし、管理主体によっては、この名前階層が増加するかもしれない。

2.7 動的名前サービス

本研究で中心的なシステムとなる動的名前サービスについて、その必要性と機能、Dynamic UPDATE で制定された仕様などについて述べる。

2.7.1 動的名前サービスの必要性

ある可搬ノードがネットワークに接続した場合、その接続の度の設定は面倒であり、また一般的な利用者には難しい。しかし、それらの設定を利用者に代わり実行し、利用者にネットワークの設定を意識させないシステムがいくつか存在する。

例えば、DHCP[17] による接続、また、LAN 接続されていないコンピュータの公衆網を用いた一時的なダイアルアップ接続と、その上での自動設定 [9] などが可能になった。これらの技術では、ネットワーク接続のための設定を自動的に行なう。したがって、環境に応じてネットワークの構成や、接続ノードが変化する。このような環境を仮に「動的なネットワーク環境」と呼ぶ。

しかし、ネットワークの情報を取り扱うべき DNS のデータベースは、ASCII テキストで書かれたデータベースに依存している。ASCII テキストは通常、頻繁な書き換えに向いておらず、動的なネットワーク環境を取り扱うには不十分である。

その上、従来の DNS には、データベースを外部から変更する手段が用意されていなかった。この問題に対する取り組みについては第 2.5.1 で述べている。つまり、Dynamic UPDATE を用いると、DNS のデータベースを外部から更新できる。

2.7.2 動的なネットワーク環境と DNS

動的なネットワーク環境は、現在の段階では DHCP や PPP 上の IPCP などにより実現されている。

ただし、DNS はこの設定の結果を反映しない。例えば、あるノードが DHCP によって "203.178.140.125" というアドレスを入手したとする。しかし、DNS のデータベースには、このアドレスに相当するドメイン名は、"dhcp-t31-6.sfc.wide.ad.jp." と書かれている。これは、ノードの名前ではなく、IP アドレスに対応する名前である。従来の DNS では、この名前をクライアント側から変更できない。従来は IP アドレスとノードの対応が静的であったため、これで問題はなかったが、現在は前述の技術により、IP アドレスとノードの対応が頻繁に変更され得る。しかしドメイン名はノードに対してではなく、IP アドレスに対して

割り当てられているため、このように問題が発生している。

この問題は、DHCP サーバが割り当てたノードの名前を元にして、DNS のデータベースを変更する機能を持つことで解決できる。例えば、DHCP 接続要求の際、要求に自分のノードの名前を付加する。すると DHCP サーバは、そのノード名とアドレスに該当するデータベースを書き換える。その結果、外部からそのノードを、人間が理解しやすい単一の固有名で呼べるようになる。つまり、ドメイン名とノードを直接関係付けることができる。

2.7.3 IPv6 と DNS

DNS にとって、IPv4 と IPv6 との差異は、要求されるデータ型の違いに過ぎない。データ型さえ正しく認識すれば、IPv6 向けのサービスは簡単に実現できる。

しかし、現実に IPv6 ノードを多数持つドメインの運用は、従来の IPv4 ノードのドメインの運用よりも難しい。IPv6 アドレスは、コロンで区切られた 8 組の 4 桁の 16 進数で表される。したがって IPv6 アドレスは、人間が扱うには覚えにくく、誤りが発見しづらい。

管理者は、管理するためにこの長く複雑なアドレスを取り扱う必要がある。特に DNS の設定では、通常のドメイン名からアドレスへの変換テーブルだけでなく、アドレスからノード名への変換テーブルも管理する。reverse lookup では、IPv6 のアドレスをあらわす DNS のドメイン名が存在する。そのドメイン名は 32 桁の 16 進数をピリオドで区切って、さらに逆順にした物である。この reverse lookup のドメイン名は、人間が取り扱うにはあまりにも長い。

人間工学の研究によると、人間が一時的に記憶できる量は、記憶単位にして 5 から 9 程度だと言う [21]。記憶単位とは、事物を記憶する時の単位であり、無意味な文字列であればそれぞれの文字、単語の列であればそれぞれの単語が記憶単位となる。このことから、人間工学的見地から見ても、IPv6 アドレスは人間が日常的に取り扱える限界を越えた長さを持っていると言える。

以上の意味で、IPv6 を用いた環境では自動設定は必須のものとなり、DNS もその影響を受ける。DNS は動的設定を用いてノードのアドレス割り当てを反映する。したがって、管理者は自動的な設定が行えない部分のみ手作業で管理すれば良い。

2.7.4 動的名前サービスへ要求される機能

動的名前サービスには以下の機能が要求される。

- 外部からデータベースへのレコードの追加
- 外部からのデータベースのレコードの削除
- SOA レコードのシリアル番号の自動的なインクリメント
- データベース変更の際の必要条件の指定

このうち、データベース変更の際の必要条件の指定とは、データベースが変更される際、期待したレコードが存在するかどうか、あるいは期待しないレコードが存在しないかどうかを調べることを意味する。例えば、chick というノードを新しく加える時に、chick というノードが既に存在すると、アドレスレコードの追加として取り扱われる。その結果、データベースには、あたかも chick というノードが複数のアドレスをもつかのように登録されてしまう。実体がアドレス毎にそれぞれ異なる場合は、この結果は好ましくない。

また必須ではないが、以下の機能が望まれている。

- ゾーン更新通知機能

- ゾーン差分転送機能
- 変更の認証機能

このうち、変更の認証機能とは、変更を行う者が変更のための権限を持っているかどうかを調べるものである。安全に情報の更新を行う仕組みは、すでにIETFで定義されている[19]。

2.7.5 Dynamic UPDATE

IETFのDNSIND-wg(Domain Name System IXFR, Notify, and Dynamic Update working group)で議論された、DNSの動的更新のプロトコルである。プロトコルは、基本的に従来のDNSメッセージの形をそのまま継承し、それぞれのフィールドの意味を変更する。

メッセージ

Dynamic UPDATEで用いられるメッセージについて述べる。

まず、通常のDNSでは、先頭からHeader部、Question部、Answer部、Authority部、Additional Record部となっている。Dynamic UPDATEでもメッセージ形式は同じだが、それぞれの部分の意味が異なっている。各部分の名称を以下に示す。

- Header部
- Zone部
- Prerequisite部
- Update部
- Additional Record部

従来のDNSメッセージと異なる部分は、Zone部、Prerequesite部、Update部である。また、Header部も若干異なっている。

また、特殊な意味を持つリソースクラス(NONE)が追加されている。これは、Prerequisite部やUpdate部で用いる。詳細は後程述べる。

Header部

Header部は、基本的には通常のDNSと同じ形だが、不要なフラグを未定義にしている。図2.5にDynamic UPDATEのHeaderを示す。

Dynamic UPDATEのOPCODEは常に5である。また、RCODEには更新に必要ないくつかの番号が追加されている。Dynamic UPDATEでは「必要条件(prerequisite)」を指定できる。例えば、特定のドメインが存在しないという必要条件を指定できる。そのドメインがもし存在した場合、エラーとなり、YXDOMAINというOPCODEを返す。また、指定したゾーンの外のレコードを更新しようとした場合もエラーとなり、ゾーン外であるという意味のNOTZONEというOPCODEを返す。

ZOCOUNTは、Zone部のレコード数である。Dynamic UPDATEでは、一つの要求では一つのゾーンの内部しか更新できない。従って、ZOCOUNTは常に1である。PRCOUNT, UPCOUNT, ARCOUNTはそれぞれPrerequisite, Update, Additional Recordの各部のレコード数である。

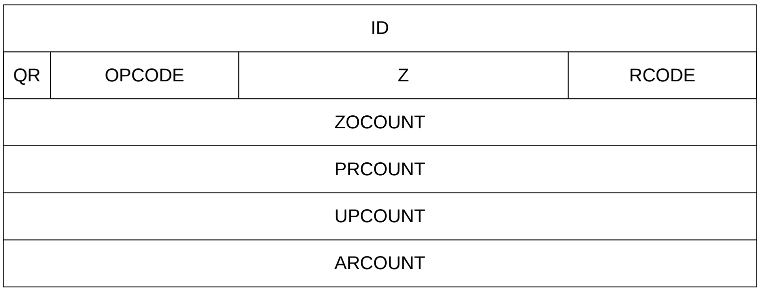

図 2.5: Dynamic UPDATE Header

Zone 部

Zone 部は、このメッセージで更新するゾーンを指定する。形式は従来のメッセージの Question 部 (図 2.3) と同等だが、リソース型は SOA でなければならない。また、リソースのクラスは、更新するゾーンのクラスと同じ物を選択する。

Prerequisite 部

Prerequisite 部では、このメッセージの更新が成立するための必要条件を指定する。形式は通常のメッセージのレコード (図 2.4) と同じである。Dynamic UPDATE では、ドメインの更新は先にサーバに到着した要求が受け入れられ、競合する可能性がある。そこで、競合した場合を検知し、更新を拒否できるように必要条件を指定する。

Prerequisite 部では、必要条件として以下の種類が指定できる。

1. 指定したドメインに指定したリソース型の集合 (RRset) が存在する
2. 指定したドメインに指定したリソース型で、指定した内容のレコードが存在する
3. 指定したドメインの指定したリソース型が存在しない
4. 指定したドメインが存在する
5. 指定したドメインが存在しない

通常のメッセージのレコード部をそのまま用いたのでは、これだけの多岐に渡る意味を表現できない。したがって、リソース型 (RTYPE)、クラス (RCLASS)、リソースデータ (RDATA) の組合せでこれらの意味を表現する。これらの値は本来の利用法と異なるので、“metavalue” と呼ばれる。

metavalue の一覧を表 2.1 にまとめる。

表 2.1: Prerequsite 部の metavalue

意味	RTYPE	RCLASS	RDATA
RRset が存在する	指定するリソース型	ANY	empty
特定の RR が存在する	指定するリソース型	ゾーンと同じ	指定する内容
RRset が存在しない	指定するリソース型	NONE	empty
ドメインが存在する	ANY	ANY	empty
ドメインが存在しない	ANY	NONE	empty

以上のように新しく定義されたリソースクラス “NONE” は、存在しないという必要条件を指定する際に用いる。

Update 部

Update 部の形式も、通常のメッセージのレコード (図 2.4) と同じである。Update 部は、実際に更新するレコードを指定する。

Update 部も、Prerequisite 部と同じように、いくつかの異なった意味を持つ。Update 部の意味を以下に示す。

1. 指定したドメインの全てのレコードを削除
2. 指定したドメインの指定したリソース型のレコードを全て削除
3. 指定したドメインの特定のリソースを削除
4. リソースを追加

これらの意味付けも Prerequisite 部と同様に、metavalue で行なう。その対応表を表 2.2 に示す。

表 2.2: Update 部の metavalue

意味	RTYPE	RCLASS	RDATA
全て削除	ANY	ANY	empty
特定の RRset を削除	指定するリソース型	ANY	empty
特定の RR を削除	指定するリソース型	NONE	指定する内容
特定の RR を追加	指定するリソース型	ゾーンと同じ	指定する内容

Additional Record 部

Additional Record 部には、更新の際の認証情報など、その他のレコードを必要に応じて加える。

2.7.6 動作の流れ

Dynamic UPDATE では、更新の要求は権限を持つネームサーバのうちいずれかに送ればよい。実際の更新を行なうサーバは、当然原本となるデータベース (Zone Master File) を持つ一次サーバである。しかし、Firewall 等の理由で一次サーバに直接到達できない場合は、二次サーバに更新要求を送る。

一次サーバの役割

データベースの原本は一次サーバが持っており、全ての更新要求は最終的にその原本に対するものである。つまり、一次サーバは全ての更新要求を処理する必要がある。

更新要求の処理は以下の手順で行なわれる。

1. 要求のゾーンに対して権限を持っているかを調査
2. 全ての必要条件が成立しているかを調査

3. ゾーン更新の許可を与えるかどうかを判断
4. 更新要求を処理

エラーが発生した場合や、必要条件がなりたたなかった場合は相当するRCODEをメッセージに書き込み、メッセージを返送する。

また、一次サーバはデータベースが変更されたならば、SOAレコードのSerialの値を増やさなければいけない。この増加は、データベースが変更になってはじめてSerialが参照された時に行なわれる。

二次サーバの役割

二次サーバは、更新要求を上位のサーバに転送する。上位のサーバとは、ゾーン転送を行なう際にデータベースの転送元となるサーバである。通常は上位サーバは一次サーバになることが多いが、防火壁を越える場合など、二次サーバを上位サーバとして持つ場合もある。その場合、上位の二次サーバはさらに上位サーバを持ち、更新要求を上へ上へと転送する。エラーが起きなければ、最終的には一次サーバに要求が到達する。

また、要求を受けた二次サーバは、上位のサーバから更新要求に対する返答が届き次第要求元に返答する。

2.7.7 考えられる問題点

名前データベースの外部からの更新により、DNSの柔軟性は飛躍的に向上する。しかし、外部からの更新を許せば、同時に外部の悪意ある利用者からの攻撃を許す。また悪意が無くても、頻繁な更新に対応すると、事故や実装上の誤りなどによりデータベースが破壊される可能性が高まる。

Dynamic UPDATEを利用する際は、セキュリティ機構を必ず利用するべきである。[16][19] しかし、セキュリティ機構はサーバのみの実装では実現できない。また、セキュリティに用いる暗号の演算がサーバにかける負荷の大きさは、運用するまでは未知数である。

また、データベースが安全なテキストファイルに依存していた従来のシステムに比べて、管理者にはより深いセキュリティの知識が要求される。システムは複雑になり、管理のための知識も同様に増大するだろう。適切な管理には、管理用インターフェイスの改善や、ツール類の充実、ノウハウのドキュメント化が必要になるだろう。

第3章　newbie

本章では、newbie(New Environment With Basic Internet Entities) ついて述べる。newbie の最終目標を定義する。その上で、今回開発したシステム (newbie-I) の設計について述べる。

3.1　newbie の最終目標

newbie の最終的な目標は、DNS を応用した、より包括的なデータベースシステムの構築である。そのデータベースシステムは、ノードのみではなく、利用者の位置情報やその他の接続性に関する情報、さまざまな知識や資源に関する情報を含む。

今回の研究は目標への第一段階である。そのためのシステムを newbie-I とする。newbie-I では、DNS に焦点を当てて、DNS と Internet を取り巻く名前サービスの問題点を探り、より新しい環境へと適応させるための試みを行う。

3.2　本稿で対処する問題とその解決

ここまで述べてきたように、二つの新しい環境が Internet にもたらされつつある。一つは「動的なネットワーク環境」であり、もう一つは「次世代インターネットプロトコル (IPv6)」である。

今回の newbie-I では、それらの新しい環境によって、DNS に発生する問題に対処する。

3.2.1　動的なネットワーク環境

newbie-I では、従来の DNS の機能の実装に加えて、動的な名前サービスを実装する。その動的な名前サービスのプロトコルには、Dynamic UPDATE を用いる [18]。そして newbie-I システムは、Dynamic UPDATE を用いて必要に応じて DNS データベースを更新する。

また、newbie-I の実装は従来の DNS の実装を基礎としない。設計の段階から柔軟性と動的な更新への対応を考慮に入れている。したがって、その内部データベースは従来の ascii 形式ではなく、更新に適するが可読性の無い方式となっている。

このようにして、従来の DNS を元に、動的なネットワーク環境に対応可能なシステムを作成する。

3.2.2　IPv6 – long address

newbie-I では、IPv6 環境下でも可能な限り管理者の人為的ミスを増やさないシステム作りを試みる。

ミスを避けて円滑なネットワーク運用を行うには、アドレス入力を回避できる限り回避する必要がある。したがって、ノードのアドレスを何らかの手段で設定する時、同時に DNS のデータベースへの登録を行う。

これにより、ホストの自動設定との連動、あるいは動的なネットワーク環境との連動を行い、管理者の人為的ミスの減少が期待できる。

3.2.3 IPv6(2) – reverse lookup

IPv6 の reverse lookup は、128bit のアドレス空間を 4bit 毎に区切り、それぞれに一桁の 16 進数をラベルとして割り当てる [13]。そして、この 32 個のラベルを元に reverse lookup の木を作成する。しかし、これでは reverse lookup の木の階層が最大 34 階層にわたり、規模性が減少すると考えられる。

また、アドレスをネットワークの経路制御に利用する部分と、全世界でのノードを識別する部分で分けて、ノードの移動透過性を改善する試みが考えられている。本稿執筆段階では、IETF で議論が行なわれており、結論は明らかになっていない。しかし、仮に IPv6 がこのような方式を採用するとすれば、従来方式の reverse lookup は不可能になる。

以上のような問題を解消するには、reverse lookup の働きそのものを見直すという案がある。RFC で実験的に定義された、ICMP Domain Name Message がその案の基礎となる [10]。

具体的な仕様は議論中であり未定だが、仕組みは単純である。IP アドレスから名前を知りたい場合、その IP アドレスに対して ICMP 名前要求メッセージを送る。このメッセージを受け取ったノードは、メッセージに自分の名前を書き込み、送信元に返答するのである。

3.3 newbie-I

newbie-I は二つの構成要素から成る。一つは、動的更新など、新しい環境へ適応する機能を備えた DNS サーバであり、これを newbie-I コアシステムと呼ぶ。もう一つは、ネットワーク上の物体 (entity) それぞれの情報を newbie-I コアシステムへと送り、更新を行う周辺ルーチンであり、これを newbie-I サポートシステムと呼ぶ。コアシステムとサポートシステムが協調動作を行い、動的に変化するネットワーク環境の情報を、管理者の手を煩わせずに他のノードに伝えられる。

3.3.1 動作モデル

newbie-I で想定している動作モデルは以下の通りである。

1. DHCP 等が利用可能な環境での動的設定ノードへの名前による接続性のサポート
2. 自動更新を用いた、ノードの名前データベースへの登録の自動化
3. ICMP メッセージを利用した、reverse lookup の簡略化

項目 1 は、自動更新の応用の一つである。DHCP 等が利用可能で、動的な設定を行なうノードに対して、それぞれのノードを識別する名前による接続性を確保するものである。この問題には、第 2.7.2 節で触れている。

この問題を解決するために、二つのモデルを考える。一つは、CISCO Systems の Yakov Rekhter 氏によって考案されたモデルであり、IETF Dynamic Configuration Working Group において議論されている。DHCP サーバが IP アドレスからノード名の対応を記述するレコードを変更し、DHCP クライアントがノード名から IP アドレスの対応を記述するレコードを変更するというものである。

またもう一つは、Y.Rekhter モデルに比較して、より簡単なモデルである。DHCP 等により動的な設定のネットワーク接続を行なうノードが、独力で自分のノード名に対応する CNAME レコードを更新し、現在位置を示すドメイン名を指し示す方式である。

項目 2 は、自動更新を用いた設定の省力化である。この問題には、第 2.7.3 節で触れている。

自動更新を用いた設定の省力化には、現在の設定を DNS に反映する仕組みが必要となる。この仕組みは、基本的には項目 1 の仕組みを一般化して作成できる。

項目 3 は、前述の二つとは形態が異なる。これは、ICMP を利用して reverse lookup を簡略化する。

従来の reverse lookup を用いずに ICMP を利用する理由については、第 3.2.3 節で述べている。また他にも、項目 1 での、CNAME 更新方式で、ノード識別子を得るための逆引きとして利用する場合が考えられる。

実現方法は、リゾルバに組み込む方法や、名前サーバが代理で ICMP メッセージ要求をやりとりする方法などが考えられる。

3.3.2 newbie-I コアシステム側の設計

newbie-I コアシステムは、実際には DNS サーバである。しかし、この DNS サーバは、Internet で広く用いられているプロトコルを用いてサポートシステムと協調動作を行い、ネットワーク環境の情報を入手できる。

設計上は、コアシステムは動的更新の機能を持ち、また IPv6 環境を視野に含んだ設計を行う。設計段階で動的更新を視野に含めているので、内部データベースも頻繁な更新に向いている形式を選択した。

具体的には、コアシステムには以下の機能が要求される。

- 標準的な DNS サーバとしての機能
- データベースの外部からの更新/動的な更新機能
- IPv6 アドレスデータ形式のサポート
- ICMP 名前要求メッセージの代理送信機能

また、コアシステムには以下のような機能が存在することが望ましい。

- 外部からのデータベース更新の際の認証機能
- 差分転送機能
- マスターデータ更新通知機能

実際の実装では、必ずしも標準 (RFC 等) に準拠しているとは限らない。実装の詳細は第 4 章で述べる。

3.3.3 newbie-I サポートシステム側の設計

newbie-I コアシステムは単独ではただの名前サーバである。次世代環境に適応するには、newbie-I コアシステムに連係して作動する周辺システムが必要である。そのシステムを newbie-I サポートシステムと呼ぶ。

newbie-I サポートシステムは、サーバ・クライアントモデルで言うクライアントに相当する。newbie-I サポートシステムをネットワーク上のノードに組み込み、そのノードをとりまく環境を newbie-I サーバに報告できる。

サポートシステムは基本的に第 3.3.1 節の動作モデルに沿って設計され、実装する。いずれの動作モデルも、システムの中心には newbie-I コアシステムが存在する。そして、サポートシステムの選択によって、コアシステムに与える意味が変わる。

まず、動作モデル項目 1「DHCP 等が利用可能な環境での動的設定ノードへの名前による接続性のサポート」では、CNAME レコードを用いて動的な設定を行なうノード (以下「動的ノード」) の、現在位置を示すドメイン名を指し示す方式を実装する。

まず、動的ノードが何らかの手段を用いて、IP アドレスを割り当てられ、ネットワークに対する接続性を得る。そこで、動的ノードは割り当てられた IP アドレスの PTR レコードを調べる。この PTR レコー

ドに書いてあるドメイン名を用いて、Internet 全体から動的ノードを特定できるはずである。このドメイン名を「一時ドメイン名」と呼ぶ。

この時点で動的ノードは、一時ドメイン名を知っている。また、担当名前サーバに接続できる。

まず動的ノードは、通常の検索手法を用いて担当名前サーバの IP アドレスを知る。次に、その担当名前サーバに対して、ノード名に存在する古いレコードを全て削除して、新しく CNAME レコードを一つ追加するよう要求する。

Internet 上の他のノードは、動的ノードのノード名を用いて動的ノードを特定しようとする。ノード名は、担当名前サーバが権限を持ち、担当名前サーバによって CNAME が返される。その CNAME は、動的ノードの一時ドメイン名を指す。その結果、動的ノードに対して通信を行なうノードは、動的ノードに割り当てられたドメイン名を知る。

動作モデル項目 2「自動更新を用いたノードの名前データベースへの登録の自動化」では、いくつかの実現方法が考えられる。今回は、設定と同時の更新ではなく、設定完了後に、現在設定されている情報を読み取って更新する方法を採用する。本来なら認証機能を考慮すべきだが、今回は対象としない。

動作モデル項目 3 については、今回は考察のみにとどめる。実装は将来の課題である。

3.3.4 相互運用性

newbie-I のほとんどの通信は、既存の Internet 技術を応用した物であり、プロトコル的には従来の技術と相互通信が可能である。しかし、DHCP のホスト名要求や、ICMP 名前要求メッセージなど、Internet のプロトコル標準に存在しないプロトコルも考慮に入れている。

全ての通信は newbie システムで閉じているわけではないので、外部との相互運用性は重要である。今回の実装では、bind-4.9.4、bind-8.1 等を参照しつつ、実用に十分足りる相互運用性を確保する。また、ICMP 名前要求メッセージなど、従来の技術体系には存在しないプロトコルを用いる場合は、従来の技術による代替手段を同時に利用できるようにして、相互運用性を保ちつつ新技術の導入を図ることにする。

第4章　newbie-I コアシステムの実装

この章では、newbie-I コアシステムの実装について述べる。newbie-I コアシステムとは、newbie におけるサーバの役割を果たすシステムである。

4.1　概略

newbie-I コアシステムの実体は、DNS サーバである。このサーバは、データベース操作モジュール、イベントモジュール、ネットワーク入出力モジュール、メッセージ変換モジュール、検索モジュール、動的更新モジュール、及び状態機械で構成される。

基本的な入出力は全てネットワーク入出力モジュールを通され、イベントモジュールが状態機械への入力となる。その際、状態機械モジュールは必要に応じてメッセージ変換モジュールを呼びだしてネットワークから入力されたパケットを解析し、必要に応じて検索モジュールあるいは動的更新モジュールを呼び出す。そして、必要があれば結果をメッセージ変換モジュールでネットワークパケットの形に変換し、出力する。

4.2　状態遷移

状態遷移自体はそれほど複雑ではない。これは、元来 DNS が、サーバやネットワークにできるだけ負荷をかけないように設計されているためである。

状態遷移の記述を容易にするため、それぞれのトランザクションの状態遷移は個別に考える。それぞれのトランザクションの状態遷移を「コンテクスト」と呼ぶ。「コンテクスト」の状態遷移は、イベントによって与えられる。イベントはネットワークの入出力や、タイマ、OS のシグナルによって発生する。

図 4.1 はコンテクストの状態遷移を示す。実際にはこの状態遷移が newbie-I コアシステム内で多重化して発生する。

4.3　モジュールの関係

各モジュールの関係は図 4.2 のとおりである。

4.4　モジュール詳解

各モジュールの実装を以下に述べる。

4.4.1　データベース操作モジュール

newbie-I サーバでは、データベースの頻繁な変更を想定している。そのため、データベースの形式は、変更に適するが可読性のない形式を採用した。具体的には、Berkeley DB 1.85(以下 DB) を利用している

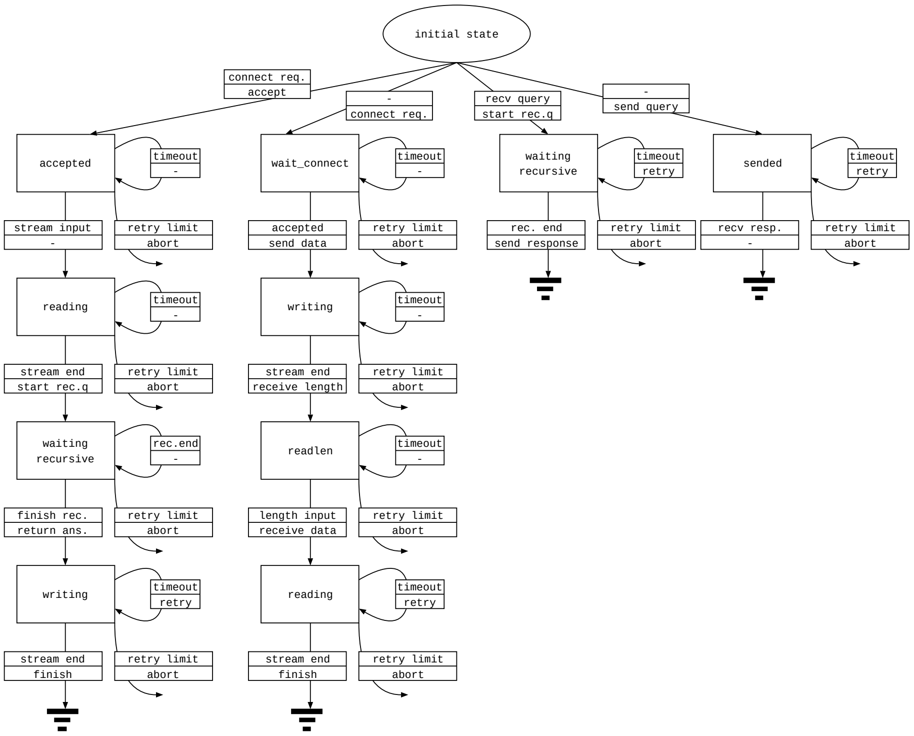

図 4.1: コンテクストの状態遷移

[6]。

DB を利用することで、ASCII 形式のデータベースファイルの利用に比べて以下のような利点がある。

- 変更の際のファイルアクセスが少ない
- データベースファイルの構文解析の必要が無く、起動が早い
- データベース利用の際、データがメモリにあるかハードディスク上にあるかを意識する必要が無い

しかしながら、ASCII テキスト形式に比べて、以下のような欠点もある。

- 可読性がなく、直接編集も実質的に不可能
- 障害が発生した場合の回復が困難

データベースは、対象となるドメイン名と、リソースのクラス、型を用いたハッシュデータベースである。この方式により、ドメインのサイズに依らないデータベース検索が可能になる。しかし、ハッシュ関数などは依然標準の物を利用しており、最適ではない。最適なハッシュ関数については今後の課題とする。

データベースのレコード部は、複雑な形態になっている。DB で取り扱うために、データ全体が連続したメモリ空間に格納される必要がある上、データそのものを可変長とするために、このような形になった。図 4.3 にレコード部の構造を示す。

まず、レコード部の先頭にはデータ数を表す 16bit のカウント (data_cnt) が存在する。そのあとに 16bit ごとに、それぞれのレコードのレコード部先頭からのオフセットが記入されている。そして、それぞれのレコードは、メモリ中のレコード先頭の番地に、対応するオフセットの値を足した番地からはじまる。そして、それぞれのレコードは、有効時間を示した 32bit のフィールド (TTL) と、データの長さをしめした 16bit のフィールド (rd_len) と、データのプライオリティを示した 16bit のフィールド (priority) を持つ。

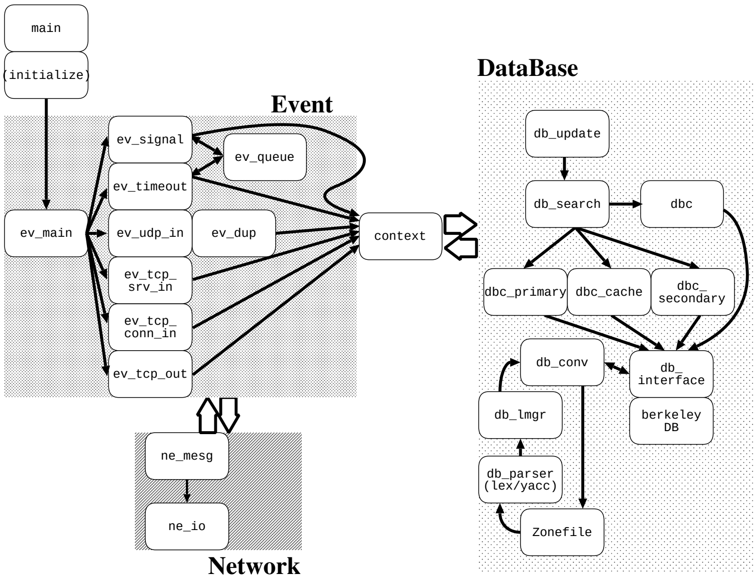

図 4.2: モジュール関係図

データベース操作モジュールは、データベースの open/close、排他制御、レコード追加、レコード削除、レコード検索、ASCII 形式でのダンプの機能を持つ。

4.4.2 データベース管理モジュール

データベースは基本的にゾーン単位で管理する。ただしキャッシュとヒントにはゾーンと言う概念が当てはまらないので、一元的に管理する。

データベースの管理は、データベースの性質に応じていくつかに分かれる。プライマリゾーンの場合はデータベースファイルのバックアップと更新履歴の管理が主な役割である。セカンダリゾーンの場合は、データベースが最新かどうかのチェックとゾーン転送要求が中心となる。また、キャッシュの場合は、TTL の切れたデータの破棄を行う。

これらの動作と、全てのデータベースに共通する動作をおこなうモジュールを併せて、データベース管理モジュールと呼ぶ。データベース管理モジュールは、それぞれのデータベースの形式毎にサブモジュールを持つ。

データベース管理モジュールの共通部では、データベースの open/close、データベースへの追加、データベースからの削除、データが存在するかの確認を行う。また、データベース操作は、直接行わずにデータベース操作モジュールを介して行なう。

データベース管理モジュールは、データの追加や削除の際に特殊なレコードを操作している。ハッシュを用いたデータベース操作ルーチンでは、レコードの種類が違うと、データベースの全く異なった場所にレコードが格納される。しかし、特定のドメインの全レコードが必要となる場合がある。そのような場合に備えて、データベースに登録されているレコードの種類のリストを、データベースの操作と同時に管理している。そして、データが存在するかの確認を行う場合などに、登録レコードの種類のリストを照会し、実際の確認を行う。

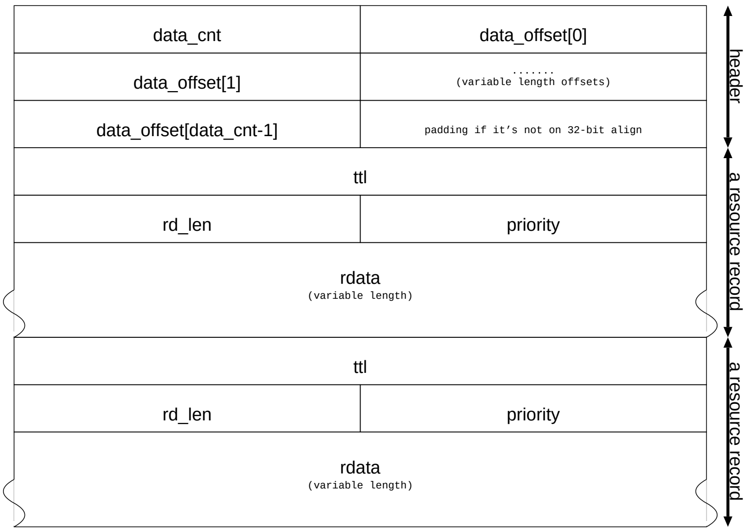

図 4.3: データベースルーチンのレコード部

4.4.3 イベントモジュール

イベントモジュールは、大きく3つに分けられる。一つはタイムアウトイベントモジュールであり、時間に基づくイベントを発行する。一つはシグナルイベントモジュールであり、OSから与えられるシグナルを元にイベントを発行する。もうひとつは入出力イベントモジュールであり、ネットワークからの入力と、ネットワークへの出力を元にイベントを発行する。

タイムアウトイベントモジュールは、時系列に配置されたイベントのリストを持つ。そして、先頭のイベントのタイムアウト時刻を設定し、その時刻にOSによるアラームを設定する。

アラームによる割り込みが行われると、時系列に配置されたイベントのリストを先頭から読みだし、処理すべきイベントに対してはその内部で記述されている処理関数を呼び出す。呼び出された処理関数では、該当するイベントに対してタイムアウトがあったのだと解釈し、タイムアウトに対処する処理を実行する。

シグナルイベントモジュールは、OSからシグナルが与えられると同時に、リストにそのシグナルに対応するイベントを接続していく。そして、メインとなる処理ルーチンのもっとも早い段階で、与えられたシグナルに対応する動作をまとめて行う。

入出力イベントモジュールはさらに細かく分かれ、TCPサービスソケット入力イベント、TCP接続済ソケット入力イベント、TCP接続済ソケット出力イベント、UDPソケット入力イベントで構成される。

TCPの入出力では、それぞれ異なるソケットを持つので、ソケットを基軸にした入出力の識別を行う。具体的には、ソケットと、そのソケットに対応するコンテクストの処理ルーチンの対応を記述したリストを内部に持ち、特定のソケットに入力があった場合は、そのソケットに該当する処理ルーチンを呼び出す。ただし、サービスソケット入力イベントは、常にTCPでサービスを行っているソケットへの要求となるので、該当するコンテクストを新たに作成する。

また、UDPの入力では一度データを読み込み、そのヘッダ部のIDやOPCODEなどを読み込む。その上で、該当するコンテクストが登録されていれば、そのコンテクストの処理ルーチンに送る。もし該当するコンテクストが無く、かつデータが何らかの要求であれば、その要求に答えるためのコンテクストを新たに作成する。

イベント処理の流れを、PADと呼ばれる形式で図4.4に示す。優先順位は、シグナルイベントがもっとも高く、次にタイムアウト、最後に入出力となる。

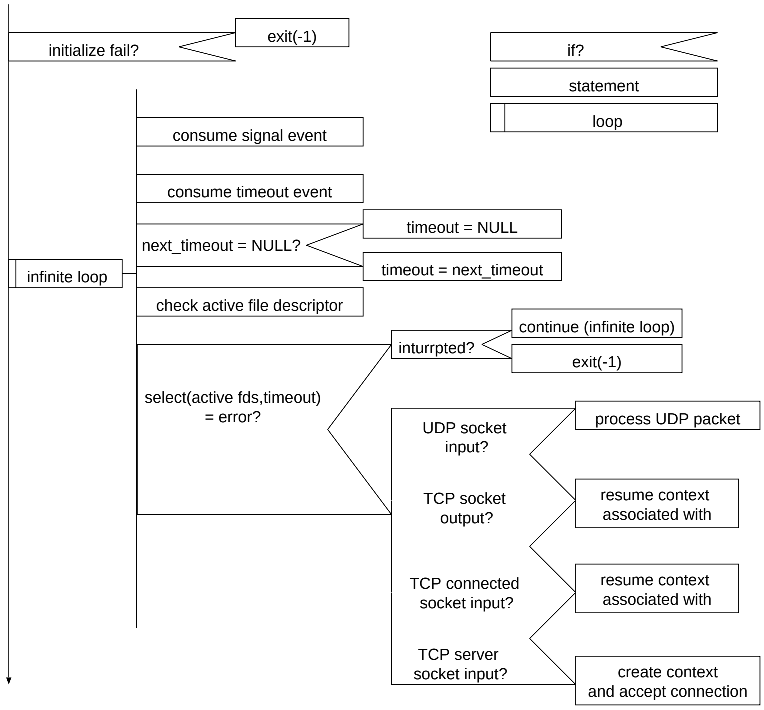

図 4.4: イベント処理

シグナルイベントは、システムや管理者からの要求であり、素早く処理する必要がある。特にプログラム終了を行なう interrupt シグナル等では、データベースファイルの同期を取る必要があり、重要である。従って、シグナルイベントの優先順位を最も高くしている。

タイムアウトと入出力イベントでは、タイムアウトイベントの方が優先順位が高い。もしもタイムアウトよりも入出力イベントの方が優先順位が高かったとすると、過剰な入力があった場合に処理が滞る。従って、タイムアウトイベントの処理を優先している。

4.4.4 ネットワーク入出力モジュール

ネットワーク入出力モジュールは、基礎的なネットワークの入出力を行う。また、UDP の入出力の際、要求に対する返答は、要求の入力のあったインターフェースから行う必要がある。そのために、インターフェースを選択する機能を持つ。

また、基礎的なネットワークコードをこのモジュールに集めたので、将来ネットワーク入出力部を IPv6 化する場合は、このモジュールを変更すれば良い。

4.4.5 メッセージ変換モジュール

DNS の通信で用いられるメッセージは特殊であり、可変長のドメイン名をさらに圧縮したものとなっている。そのようなネットワークメッセージを解釈し、内部データベースで取り扱いやすい形式に変換する。

また、逆に内部データベースのデータを元に、ネットワークメッセージを作成する。同時に、ドメイン名の圧縮を行う。

圧縮されたドメイン名の比較や、SOA serial の比較など、メッセージに関連する細かいルーチンが存在

する。

4.4.6　検索モジュール

データベース検索モジュールは、データベースの設定ファイルを読み込み、データベース操作モジュールを用いて、それぞれのデータベースを開く。その際に、管理するデータベースのリストを作成する。

データベース検索の要求が与えられると、データベース検索ルーチンは以下のように検索を行う。

1. 検索の対象になるゾーンの選択
2. 要求を満たすレコードの検索
3. 該当するレコードが存在した場合は、ADP 検索を行う
4. CNAME 要求でないときに CNAME が返って来た場合、検索対象を CNAME で置き換え、再検索
5. 検索対象に最も近い上位の名前サーバ (NS) レコードを検索
6. 検索対象を含むゾーンの SOA レコードを検索
7. 検索対象ドメインが存在する可能性が残されているかのチェック
8. 権限を持つレコードが答えに存在しなければ、キャッシュデータベースをチェック
9. NS レコードが存在しない場合、ヒントファイルから入手する

実際には、下位の検索を行う際に、この検索ルーチン自身を再帰的に呼び出している。また、ルーチンに与えるフラグによって、動作が若干変わり、複雑さを増している。このルーチンの最適な実装は、今後の課題とする。

4.4.7　動的更新モジュール

動的更新モジュールは、動的更新要求がサーバに与えられた場合にコンテクストにより呼び出される。

動的更新モジュールは、以下の手順で動作を行う。

まず、与えられた更新要求を解析し、該当するゾーンへの権限を持っているかどうかを調べる。権限を持っていない場合は、要求の発信者にエラーを返す。

該当するゾーンの二次サーバを行っている場合、その要求をゾーン転送を行う元のサーバに転送する必要がある。この機能は、まだ実装されていない。

該当するゾーンの一次サーバを行っている場合は、以下の手順で更新を行う。

1. prerequisite section の処理
2. permission check
3. update section の前処理
4. update section の処理

prerequisite section の処理とは、要求の必要条件となるレコードの存在の確認である。要求が指定されたゾーンの外部にある場合、存在を要求されるレコードが存在しない場合、及び存在してはいけないレコードが存在した場合、要求の形式が誤っている場合、それぞれ該当するエラーを RCODE(result code) にセットし、返信する。

permission check では、要求が正当なものかを調べる。DNS でのセキュリティ機能が実装されていないので、今回はなにも行っていない。

update section の前処理では、update section の形式が正しいかを調べる。もし指定されたゾーンの外部のレコードを更新しようとしていたり、update section の形式が誤っていれば、該当するエラーを RCODE

にセットし返信する。

update section の処理は、要求にしたがってレコードの追加及び削除を行なっている。その際、SOA の削除や、CNAME とそれ以外のレコードの重複など、DNS 的に誤りとなる要求は無視する。

また、update section の処理は、RFC ではパケットの先頭から順番に処理する必要がある。しかし、現段階では、実装の都合上パケットを一度すべて解析し、それぞれのレコードを RRset にまとめた上で処理を行なっている。つまり、このセマンティクスは、RFC で指定された仕様とは大幅に違う。

4.4.8 状態機械モジュール

冒頭で「コンテクスト」と表現した、状態機械に相当するモジュールである。コンテクストのデータ構造体の定義を以下に示す。

```
/* コンテクストのための構造体 */
typedef struct context {
  struct context* parent; /*<- 親コンテクストへのポインタ */
  struct context* child; /*<- 子コンテクストへのポインタ */
  int type; /*<- 通信形式 */
  int current_state; /*<- 現在の状態 */
  union {
    void* p;
    Cos_Query*           cq;
    Cos_TCP_Request*  t_req;
    Cos_TCP_Response* t_res;
    Cos_UDP_Request*  u_req;
    Cos_UDP_Response* u_res;
  } param; /*<- 各通信形式に対応する要素 */
  union {
    Mesg_Hdr* hdr; /*<- メッセージヘッダ */
    u_char* p; /*<- メッセージバッファへのポインタ */
  }mesg;
  u_char* wp; /* <- メッセージの末尾 */
  int buf_len; /*<- メッセージバッファの長さ */
  int mesg_len; /*<- メッセージの長さ */
  u_char* qname; /*<- query の名前 */
  int qname_len; /*<- qname の長さ */
  u_int16_t q_class; /*<- query のクラス */
  u_int16_t q_type; /*<- query の型 */
  u_int32_t q_id; /*<- 通信の ID */
  G_List* an_list; /*<- 答のリスト */
  G_List* ns_list; /*<- 名前サーバのリスト */
  G_List* ar_list; /*<- 追加情報のリスト */
  u_int32_t lock_id; /*<- 排他制御 ID */
  struct ev_to_data* tout; /*<- タイムアウトイベントへのポインタ */
  int (*process)(struct context* ctx); /* 次の状態に進む関数 */
  int (*retry)(struct context* ctx); /* リトライ関数 */
```

```
} Context;
```

コンテクストは、親子関係を持つ。この親子関係は、問い合わせが与えられた場合に、その問い合わせを別のサーバに転送する場合に用いられる。別のサーバに問い合わせ、回答を得るコンテクストを子コンテクストとして作成して、再帰問い合わせを実装している。

コンテクストは、それぞれのコンテクストの形式に応じた変数を持つ。これらは主に、プロトコルによって形式の違う、現在の通信相手の情報を格納している。

また、コンテクストは、入出力に用いるバッファのポインタを持っている。ポインタはヘッダ構造体との共用体になっており、簡単に参照が可能である。また、バッファに関する各種情報も同時に持つ。

コンテクストは、データベース検索や再帰問い合わせで入手した情報を、answer list, nameserver list, additional record list の三つにわけて保持する。この三つは、そのまま返信メッセージの該当する部分に対応する。

また、コンテクストは、それぞれ次の状態に遷移する関数へのポインタと、タイムアウトが発生した場合の関数をもっている。イベントルーチンは、これらのポインタを参照し、イベントの発生をコンテクストに通知する。

4.5 実装と標準との相違

未実装の項目とプログラム上の誤りを除いて最も大きな違いは、Dynamic UPDATE の際のセマンティクスの違いである。標準 [18] では、Update 部は先頭から順に処理するよう規定されている。この規定通りならば、最初に特定のドメインの全てのレコードを削除したあと、そのドメインに望むレコードを加える、といった一連の操作を、一つの要求で行なえる。

しかし、newbie-I コアシステムでは、全てのリソースレコードを RRset 単位に分解してから処理を行なう。従ってメッセージ上での RR の順番は保存されない。そのため、例えば前述のようにして特定のドメインを書き換える場合、レコード追加の後、最後に全レコード削除の要求が処理される、などという場合がある。

この問題を解決するためには、Update 部のみ特別扱いをして、メッセージの先頭からの順番を保証する必要がある。この問題の解決は、今後の課題とする。

第5章　newbie-I サポートシステムの実装

この章では、newbie-I サポートシステムの実装について述べる。newbie-I サポートシステムは、ネットワーク環境の変化を自動あるいは半自動で newbie-I コアシステム、すなわち DNS サーバへフィードバックする仕組みである。newbie-I コアシステムは単独ではただの名前サーバに過ぎない。このサポートシステムとの連係によって初めてシステムとしての意義を持つ。

5.1　動的設定ノードへの名前による接続性のサポート

動的設定ノードへの名前による接続性のサポートの実現は、cnameupd というコマンドの実装によって行なった。図 5.1 に動作概略を示す。

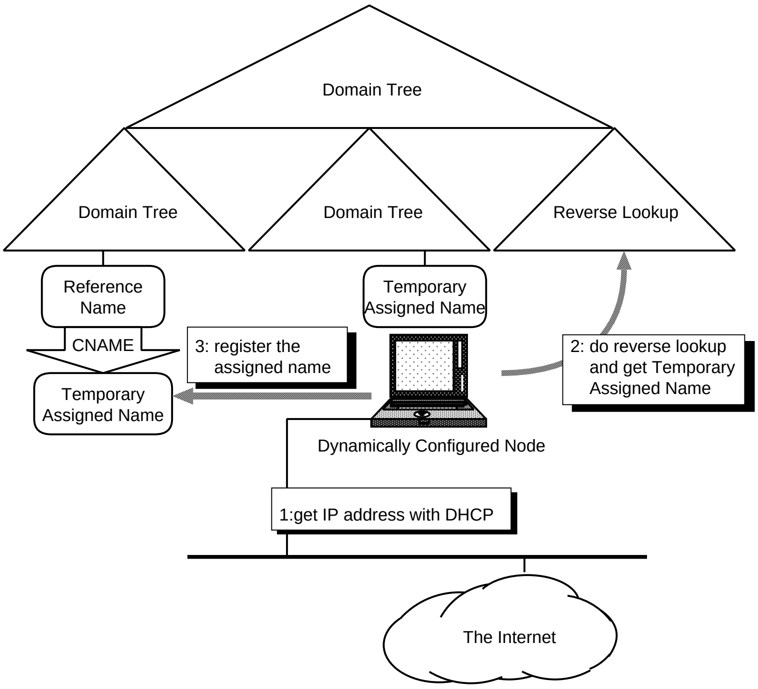

図 5.1: cnameupd 概略

cnameupd を用いる場合、ノードはまず動的な設定を用いて IP アドレスを取得する。図 5.1 中では項目 1 に対応する。ここで cnameupd を起動する。cnameupd は、割り当てられた IP アドレスを手がかりに一時的に割り当てられたドメイン名 (Temporary Assigned Name) を得る。この動作は、図 5.1 中では項目 2 に対応する。またここで、設定された参照名 (Reference Name) からノード名を知り、ノード名から担当名前サーバを調べる。そして、一時ドメイン名を、担当名前サーバ上のノード名に相当するドメインに、CNAME レコードとして登録する。ここが、図中項目 3 に対応する。その結果、DNS の名前データベース

の木 (Domain Tree) の参照名に、現在の一時ドメインを指し示す CNAME レコードが作成できる。

cnameupd は、ネットワークインターフェイスを引数として取る。現在の段階では通常のリゾルバを用いている部分がある。しかし、動的更新を行なうルーチン中にもリゾルバに相当する機能があり、将来的にはこれらを統合する予定である。

cnameupd は、動的更新の結果、失敗した場合は-1 を返す。また、成功した場合は 0 を返し、DNS サーバのエラーが発生した場合はそのエラーコードを返す。

このコマンドを実行して成功すると、ノード名に対応するドメインに、現在の接続を示す CNAME レコードを作成する。他のノードはこの CNAME レコードを頼りに接続を試みる。

動的更新を行なうルーチンの詳細は、第 5.4 節で述べる。

5.2 名前データベースへの登録の自動化

第 2.7.3 節で述べたように、IPv6 の普及による管理者の負担の増加が懸念される。すこしでも管理者の負担を減らすために、第 3.2.2 節にあるように、ノードの設定を DNS のデータベースに自動的に反映する仕組みを考える。

ノードの設定で、最低限必要な設定は A/AAAA レコードと PTR レコードである。DNS の主な役割は IP アドレスと PTR レコードの対応付けであり、そのためにはこれらのレコードが不可欠である。NS レコードや MX レコードなどは、末端のノードの設定にはあまり関与しない。

実現方法には次の 2 段階がある。一つ目は、ifconfig/dhcpc 等、インターフェイスに対してアドレスを実際に割り当てるコマンド内部に、DNS 動的更新クライアントを実装する方法である。意味的には、別のコマンドで動的更新を実現し、インターフェイス設定コマンドと連係させて動かすことと同義である。二つ目は、kernel 内部に動的更新クライアントを実装して、設定方法に依存しない動的更新を実現する方法である。

今回は、独立したコマンドを用いてノードの設定を名前データベースに反映させる方法を採用する。そのためのコマンド dnsregister は、インターフェイス名を取り、A/AAAA レコード、PTR レコードを登録する。引数指定で、マルチホームのノードでもインターフェイス毎に別のドメイン名を用いることができる。

基本的には第 5.1 節で述べた cnameupd の応用拡張であり、実装上の問題は少ない。しかし、運用上の経験が少なく、サイト全体をこの方式で管理できるかどうかは、今のところ、今後の研究課題とする。また、実際の運用に必要不可欠な、セキュリティの機能を持っていない。

5.3 reverse lookup の簡素化

第 2.6 節で述べたように、IPv6 での reverse lookup はいくつかの問題を持つ。その問題を改善する一つの案として、reverse lookup を ICMP メッセージで行なう方法があると述べた。

今回は、慶應義塾大学南政樹氏の協力でこの ICMP メッセージのしくみを実装した。newbie サーバに組み込み、実験を行なった。ただし、今回の実装では、実際に IPv6 による通信は行なわず、IPv4 を用いた実験のみを行なう。

5.4 libnewres.a

動的更新を行なうルーチンは libnewres というライブラリの形で提供される。これは newbie コアシステムの開発に伴い生まれたライブラリである。このライブラリによって、アプリケーションは簡単な動的更新要求を発行できるようになる。しかし、API を簡素化するために複雑な動的更新要求は作成できない。

動的更新は、以下の関数によって発行できる。

```
int res_update_record(int update_mode, int prereq_mode, char* domainascii,
                      u_int16_t rtype, u_int16_t rclass,
                      u_int32_t ttl, char* rdascii);
```

update_mode 及び prereq_mode は、更新と、更新の際の必要条件を、いくつかの雛型から選択する。Dynamic UPDATE の仕様では、より柔軟な必要条件を指定できるが、ここでは簡便のため、必要条件を絞った。また、仕様では一度の要求で、削除や追加などの一連の動作を記述できるが、今回は一度の要求で一つの削除か追加のみ記述する。

update_mode には以下の 5 種類の記号定数がある。

RES_UPDATE_ADD レコードの追加

RES_UPDATE_DEL_ALL 指定したドメインのレコードの全削除

RES_UPDATE_DEL_RRSET 指定したドメインの、指定した型のレコードの集合 (RRset) の全削除

RES_UPDATE_DEL_RR 指定したドメインの、特定のレコードの削除

prereq_mode には以下の 6 種類の記号定数がある。

RES_PREREQ_NO 必要条件を指定しない

RES_PREREQ_DOM_EXT 必要条件:指定したドメインが存在する

RES_PREREQ_DOM_NEXT 必要条件:指定したドメインが存在しない

RES_PREREQ_RRSET_EXTI 必要条件:指定した RRset が存在する

RES_PREREQ_RRSET_EXTD 必要条件:指定した型でかつ指定した値の RR が存在する

RES_PREREQ_RRSET_NEXT 必要条件:指定した型のレコードが存在しない

必要条件を指定した場合、名前サーバ上のデータベースが必要条件を満たす場合のみ更新が行なわれる。これによって、新規作成の意図でレコードを追加したものの、すでにレコードが存在していた、などという事故を防げる。

domainascii は、ASCII 文字列で示される、追加や削除の対象となるドメイン名である。この API では、簡便のために、必要条件の対象となるドメイン名は追加や削除の対象となるドメインと共通であるという仮定を行なっている。このドメイン名を示す文字列は、DNS の標準 [8] で定義されるドメイン名の記述法に沿う必要がある。

rtype, rclass, ttl は、それぞれ追加するドメイン名のリソース型、リソースクラス、Time To Live を表す整数である。リソース型などについては include ファイル、ns_const.h に記述されている。

rdascii は、ASCII 形式で書かれたリソースデータで、DNS 標準の記述法で指定する。例外は SOA レコードであり、括弧は使わず、すべてを一行に書き込む。例は、“203.178.140.101” (A record), “5 shonan.sfc.wide.ad.jp.” (MX record), の様になる。

第6章　評価

本章では、newbie-I システムの評価を行う。評価は、性能と機能の両面から行う。

評価の対象となる newbie-I システムはいずれも、version 0.05 である。

6.1　サーバ性能評価

性能評価の比較対象として、ISC で開発された bind-8.1 を選択した。bind-8.1 は、DNS の標準的な実装である。

6.1.1　query 反応速度

ローカルホストのネームサーバに対し、特定のパケットを 100 回繰り返し送信し、反応速度の平均を測定した。実験の環境は、Intel Pentium 120MHz / Memory 40MB の FreeBSD 2.2.1 マシンである。cron 等のコマンドの動作は、実験データを乱す恐れがあるので、すべて停止させた。また、送信したパケットは、単一のドメイン名のアドレスを調べるものである。ドメイン名は以下の 4 パターンを選択した。また、実験の対象となるデータベースには、sfc.wide.ad.jp. ゾーンのデータベースを用いた。

- データベースの先頭に存在するレコード (HEAD)
- データベースの中央付近に存在するレコード (MIDDLE)
- データベースの末尾に存在するレコード (TAIL)
- リソースレコードを複数個もつレコード (MULTI)
- ネームサーバレコード (NS)

表 6.1 に結果を示す。

表 6.1: query 反応速度表 (単位 ミリ秒)

システム名	HEAD	MIDDLE	TAIL	MULTI	NS
newbie-0.05	8.6	8.8	8.4	8.6	8.7
bind-8.1	8.2	7.9	8.7	8.5	7.5

参考までに、同様の手法で UDP echo (port 番号 7) の反応速度を測ったら、7.5 ミリ秒程度であった。

以下は、計測部のソースプログラムを抜きだしたものである。

```
for(i = 0; i < count; i++){
  /* anyway send packet */
  /* timeout */
```

```
  packet_hdr->id = i;

  tv_tmp = interval;
  fd_in = fd_org;

  if(sendto(sock, packet, packet_length, 0,
            (struct sockaddr*)&to, to.sin_len) < 0){
    perror("sendto");
    exit(-1);
  }
  gettimeofday(&tv_now, NULL);
  pstat[i].send_tv = tv_now;

  while(select(sock+1, &fd_in, NULL, NULL, &tv_tmp) > 0){
    if(recvfrom(sock, recvbuf, 512, 0, NULL, NULL) < 0){
      perror("recvfrom");
      exit(-1);
    }
    gettimeofday(&tv_now, NULL);
    id = hdr->id;

    pstat[id].recv_tv = tv_now;

    /* set next timeout = (default interval - (now_time - last_sent)) */
    tv_tmp.tv_sec = (interval.tv_sec -
                     (tv_now.tv_sec - pstat[i].send_tv.tv_sec));
    tv_tmp.tv_usec = (interval.tv_usec -
                      (tv_now.tv_usec - pstat[i].send_tv.tv_usec));
    if(tv_tmp.tv_sec < 0){
      tv_tmp.tv_sec = 0; tv_tmp.tv_usec = 0;
    }else{
      if(tv_tmp.tv_usec < 0){
        (tv_tmp.tv_sec)--;
        tv_tmp.tv_usec+=10000;
      }
    }
    /* the select(2) does not care tv_usec.... round it */
    tv_tmp.tv_usec = 0;
    tv_tmp.tv_sec++;

    fd_in = fd_org;
  }/*while(select() > 0)*/
}/* for(...) */
```

6.1.2 動的更新の効率

動的更新を行ない、反応速度を計測した。実験環境は、第6.1.1節と同様である。特定ドメインのレコードの追加と、同レコードの削除を交互に50回行ない、時間を計測した。

表6.2に結果を示す。

表6.2: update 反応速度表 (単位:秒)

システム名	時間
newbie-0.05	0.50
bind-8.1	3.13

6.1.3 サーバ起動速度

サーバが起動するまでの速度を計測した。この計測では特にデータベースの読み込みと解析にかかる時間を比較している。実際には、サーバのmain()関数の先頭を処理した時間と、要求に応答できるようになった時間の差分である。それぞれのサーバは、同様のデータベースに対するサービスを行なう。

表6.3に結果を示す。

表6.3: 起動時間 (単位:秒)

システム名	起動時間
newbie-0.05	0.14
bind-8.1	0.39

6.1.4 全体的なメモリ消費量

同等のゾーンに対するサービスを行なう設定で、起動時にどれだけのメモリを消費するかを調べた。

表6.4に結果を示す。

表6.4: メモリ消費量 (単位:KB)

システム名	メモリ消費量 (RSS)
newbie-0.05	640
bind-8.1	948

6.2 サーバ機能評価

機能評価は、さまざまな機能それぞれについての実装状況から判断した。機能評価をまとめたものを表 6.5 に示した。

表 6.5: サーバ機能評価

項目名	newbie-0.05	bind-4.9.6	bind-8.1
一次サーバ	○	○	○
二次サーバ	未実装	○	○
キャッシュ	未実装	○	○
再帰問い合わせ	○	○	○
動的な更新	○	×	○
差分転送	×	×	×
更新通知 (NOTIFY)	×	×	○
データベース可読性	×	○	○

6.3 newbie-I サポートシステムの評価

newbie-I サポートシステムは、動的な名前サービスの応用を行なう。そのための評価は、対比するシステムが少ないため、難しい。ここでは、DHCP と DNS の協調動作について、cnameupd と Yakov Rekhter/IETF 方式の比較を行なう。また、それぞれのサポートシステムが、目標をどの程度達成できたかを評価する。

6.3.1 動的設定ノードへの名前による接続性のサポート

第 3.3.1 節では、CNAME レコードを用いた接続性のサポートを行なうプログラム、cnameupd について述べた。また同時に、DHCP との連動を行なう、Y.Rekhter/IETF 方式についても述べた。表 6.6 で、その両方式を比較しまとめた。

表 6.6: 名前による接続性のサポート

項目名	cnameupd	Y.Rekhter/IETF
他のノードからの接続性	○	○
lookup/rev.lookup の対応	○	○
rev.lookup で分かる名前	一時的・可変	不変
プロトコル依存性	非依存	DHCP に依存
セキュリティ	未実装	未実装

第7章　考察及び今後の課題

今回の研究では、動的名前サーバとしての newbie コアシステム、次世代環境支援システムとしての newbie サブシステムの実装を行なった。

ここでは、今回の実装および評価の考察を行い、また newbie-I の今後の課題についても述べる。

7.1　考察 – 動的名前サービスの効果

7.1.1　運用

今回の newbie-I では、動的名前サービスを実現した。また、動的名前サービスは単体では動作しないので、サポートシステムを作成し、小規模ながら運用した。この運用から、いくつかの問題点が明らかになった。

まず、newbie コアシステムは、二次サーバを実装していない。当然、今後の課題として将来的には実装する計画である。二次サーバを置くことができない状況での運用は、非常に不安定であった。利用者は、一般に名前を用いてネットワーク上のノードを識別する。また、WWW が普及した現在、情報の識別も URL 形式であり、DNS に依存している。つまり、ネットワークの利用の殆んどが DNS に依存しているので、DNS の障害に対する耐久性は、非常に重要である。

また、動的な設定でネットワークに接続するノードの、名前によるサポートは大きな意味を持つ。従来はネットワークに接続している端末でも、相手の名前が分からないがために利用できないことが多かった。しかし、cnameupd などの実装で、動的な接続を行なうノードを一意の名前で識別できるようになった。この実装により、利便性が格段に向上している。

すなわち、いままではこのようなノードはあくまで端末としての位置づけであり、接続や情報を要求する一方であった。しかし、このように一意の名前で識別できると、移動先で DHCP で接続していても、あるいは公衆網を経由した PPP 接続でも、現実的なサービスを提供することができる。これにより、家庭での一般的なユーザが、サービスプロバイダのサーバなどに依存せずにサービスを提供することができるようになる。

このようなサービスは非常に限定的な物だが、末端利用者ができることを広げると言う意味で、意義を持つだろう。

7.1.2　性能

今回の実装は、まだまだ最適化されていない。しかし、最適化されていない実装でも、表 6.1 に示すように、通常の問い合わせ応答時間には、bind-8.1 と同程度の速度を得る事ができた。

これから最適化を行ない、速度的により改善できる。具体的には、メモリ上でのコピーの回避、検索ルーチンの見直しなどを行なうことになる。また、メモリ割り当ての方法を改善し、ページをまたぐようなメモリ利用を回避できれば、性能はさらに向上する。しかし、bind-8.1 に比べて実装していない機能が幾つか存在するので、単純な比較はできない。

動的更新の処理では、性能面で明確な差が出た。表 6.2 は、約 6 倍の性能差を示している。bind-8.1 では、データベースに従来型の ASCII テキストファイルを用いており、書き換えの為にはファイルアクセスが多数発生する。しかし、newbie では、DB1.85 を用いており、書き換えは比較的簡単に処理できる。動的更新の処理には、この差が顕著に現れた。

起動時間は、表 6.3 に示すように、newbie が bind-8.1 に比べて 2 倍以上早い。主原因は、newbie ではデータベースファイルの構文解析が必要ないという点であると考察できる。しかし、現時点の newbie は bind-8.1 に比べて、表 6.5 のように機能的に劣る。同等の機能を実装していない以上、単純な比較はできない。

機能的な差が現れる点にはもう一つ、メモリ消費量がある。表 6.4 に示すように、newbie に比べて bind-8.1 は起動時で約 1.5 倍のメモリを消費する。メモリ消費量は、起動時間と同様に実装されている機能の差が現れていると考察できる。機能の差だけではなく、データベースの形式の差が現れた結果だと考える事もできるが、今回の研究では十分な実験が行なえなかった。

7.2 今後の課題

今後の課題は多数存在する。以下に列挙する。

- 未実装項目の実装
- Update のセマンティクス問題の解決
- コアシステムの最適化
- ネットワーク通信の IPv6 化
- セキュリティの確保
- サポートシステムの実装と比較
- 大規模運用
- DNS の規模性の定量的な解析
- ドメイン名の再定義

newbie の目標は DNS を用いて包括的な情報環境を作成することであり、そのためには、現在のドメイン名の定義では不十分である。DNS はその設計段階では、ネットワーク上で名前の付けられる全ての存在 (entity) の情報を取り扱う為のシステムとして設計された [4]。しかし、現在のドメイン名は、あくまでも最終的にノードを指し示すものとして取り扱われている。

しかし、名前の付けられるすべての存在を取り扱えるのならば、様々な可能性を考える事ができ、有用である。特に、動的な名前サービスと、適切なサポートシステムを組み合わせ、応用した場合の可能性はまだ未知数である。この可能性を探ることを、今後の研究としたい。

第8章　結論

多数のコンピュータがお互いに通信しあうためには、全てのコンピュータが一意に識別できる必要がある。そして、社会がInternetを広く利用して行く為には、簡単に利用できる事が重要である。

Internetを簡単に利用するための手法は多数存在する。そのうち、もっともわかりやすい方向性を持っている手法は、ネットワーク環境の自動設定だろう。そして、公衆網を利用した接続を含めて、自動設定技術はこれからのInternetの広がりに不可欠な技術になりつつある。

しかし、現在実現している技術のみでは、Internetの広がりを支えるには不十分である。例えばDNSでのDynamic UPDATEは実現しつつあるが、その技術を活かす環境はまだまだ開発途上である。

今や、Intenet技術的な研究をする上で、Internetの社会での役割を無視する事はできない。そして、その役割を果たす為に、自動設定や、ネットワーク利用の簡便さを向上させる技術、そして何よりも、そのような技術を利用するための環境の開発が重要である。

newbie-Iの開発により、まだまだ利用できる環境が限られている動的な名前サービスを、いくつかの利用法に対して開放した。そしてこの開発は、動的名前サービスを活かしたネットワーク環境の実現の第一歩となるだろう。

また、その過程で作成したnewbie-Iの各システムは、それぞれ独立運用が可能である。特にコアシステムの名前サーバとしての性能は、bind-8.1に比較しても遜色がない。その上、動的名前サービスの効率はbind-8.1に比べて非常にすぐれている。コアシステムはbind-8.1に対して機能的には劣るが、これから改善できるだろう。

第9章　謝辞

まず、本研究を行なうにあたり、日頃からの暖かい御指導を頂きました徳田英幸教授、村井純教授に心より感謝致します。

また、全般的な研究活動においてさまざまな貴重なアイディア、助言、叱責などを頂き、またさまざまな議論をして頂きましたWIDEプロジェクトの皆様、その中でも特に深い御指導を頂きました、奈良先端科学技術大学院大学の山本和彦氏をはじめとするIPv6ワーキンググループの皆様に深く感謝致します。

そして、実際の執筆作業で、あらゆる面での暖かい援助と御指導を頂きました、政策メディア研究科の南政樹氏をはじめとする徳田・村井・楠本・中村研究室の皆様に深く感謝致します。

最後に、精神的な面も含めたさまざまな面で助けになって頂いた学友達、並びに、研究活動を含め、大学での活動を全面的に支えて頂いた両親に心より感謝致します。

関連図書

[1] 尾上 淳, 慶應義塾大学大学院理工学研究科, 広域解放型分散環境における名前サーバの設計と実装, 修士論文, Feb. 1990

[2] 大川 恵子, 慶應義塾大学大学院理工学研究科, インターネットネームサーバに関する一考察, 修士論文, Feb 1985

[3] Peter B. Danzig, Katia Obraczka, Anant Kumar, University of Southern California, *An Analysis of WIde-Area Name Server Traffic*, ACM Computer Communication Review 22(4), Oct. 1992

[4] , Douglas B. Terry, Mark Painter, David W. Riggle, Songnian Zhou, University of California Berkeley, *The Berkeley Internet Name Domain Server*

[5] , Tim Vann, 6611 Hardware Development, William McFall, Siemens Corporation, Mark McNeill, Duke Power, *An Introduction to BIND Source Code*, IBM Unclassified Documents

[6] Margo Seltzer, University of California Berkeley, Ozan Yigit, York University, *A New Hashing Package for UNIX*, USENIX, Winter 1991

[7] P. Mockapetris, ISI, *DOMAIN NAMES - CONCEPTS AND FACILITIES*, RFC1034, Nov. 1987

[8] P. Mockapetris, ISI, *DOMAIN NAMES - IMPLEMENTATION AND SPECIFICATION*, RFC1035, Nov. 1987

[9] G. McGregor, Merit, *The PPP Internet Protocol Control Protocol (IPCP)*, RFC1332, May 1992

[10] W. Simpson, Daydreamer, *ICMP Domain Name Messages*, RFC1788, Apr. 1995

[11] S. Deering, Xerox PARC, R. Hinden, Ipsilon Networks, *Internet Protocol Version 6 (IPv6) Specification*, RFC1883, Dec. 1995

[12] R. Hinden, Ipsilon Networks, S. Deering, Xerox PARC, *IP Version 6 Addressing Architechure*, RFC1884, Dec. 1995

[13] S. Thomson, Bellcore, C. Huitema, INRIA, *DNS Extensions to support IP version 6*, RFC1886, Dec. 1995

[14] M. Ohta, Tokyo Institute of Technology, *Incremental Zone Transfer in DNS*, RFC1995, Aug. 1996

[15] P. Vixie, ISC, *A Mechanism for Prompt Notification of Zone Changes (DNS NOTIFY)*, RFC1996, Aug. 1996

[16] D. Eastlake 3rd, CyberCash, C. Kaufman, Iris, *Domain Name System Security Extention*, RFC2065, Jan. 1997

[17] R.Droms, Bucknell University, *Dynamic Host Configuration Protocol*, RFC2131, Mar. 1997

[18] P. Vixie, ISC, S. Thomson, Bellcore, Y. Rekhter, Cisco, J. Bound, DEC, *Dynamic Updates in the Domain Name System (DNS UPDATE)*, RFC2136, Apr. 1997

[19] D. Eastlake 3rd, CyberCash, *Secure Domain Name System Dynamic Update*, RFC2137, Apr. 1997

[20] R.Hinden, Ipsilon Networks, M. O'Dell, UUNET, S. Deering, Cisco, *An IPv6 Aggregatable Global Unicast Address Format*, draft-ietf-ipngwg-unicast-aggr-01.txt, June 12, 1997

[21] 佐伯 胖編, 理解とは何か, 認知科学選書, 東京大学出版会, 1985

[22] SFC CNS ガイド編集委員会, *SFC CNS* ガイド, Apr. 1997