修士論文
DNS 実装のデータベースフレームワークに関する研究

慶應義塾大学 政策・メディア研究科 修士 2 年
石原知洋

平成 15 年 2 月 19 日

修士論文要旨 2002 年度 (平成 14 年度)

DNS 実装のデータベースフレームワークに関する研究

論文要旨

DNS は、インターネットで最も利用されている広域分散協調データベースであり、主にインターネットプロトコル (IP)[4] のアドレスとホスト名の変換を行っている。

そして、インターネットの爆発的な普及により、接続ホスト数は増え、名前空間も広大化した。そのため、DNS のデータ設定ファイルを肥大化し、データの変更も頻繁に行われるようになった。また、悪意を持つユーザーも多く現れ、DNS もセキュリティに関して注意を払わなければならなくなった。このため、DNS を見直す必要が発生し、それを受けて DNS に対していくつか拡張が考えられた。しかし、それによって DNS のデータは肥大化／複雑化し、従来のようにテキスト形式の設定ファイルで新しく拡張された DNS を管理することが難しくなってきた。

そこで、DNS において外部データベースを使うための仕組みが考えられた。本研究ではその仕組みを拡張することで、肥大化したデータの管理を容易にし、管理コストの低減を実現した。

キーワード

1. DNS 2. データベース 3. 動的更新 4. セキュリティ 5. 拡張 SDB

慶応義塾大学 政策・メディア研究科

石原知洋

Abstruct of Master's Thesis Academic Year 2002

A Study of Database Framework for DNS Implementation.

Summary

The Domain Name System(DNS) is a widely distributed collaborated database, tranlating host-names to IP addresses in the Internet Protocol(IP).

As the Internet grow at explosive speed, the number of host that connect to the Internet have increased as well as the number of name spaces in the Internet. This growth have consequently cause configuation files in DNS to enlarge expansively, and increased the need to make changes to the databse more frequently. Meanwhile, increase of malicious users have forced DNS to consider security issues. Consequently, a need to reconsider DNS have arised and few extensions has been proposed as solutions to these problems.

However, the proposed extensions enlarged and complexed the DNS database even more, making it more complex to maintain the extended DNS with text base configuration files used historically.

To overcome this problem, a newframe work has been proposed to use an external database with the DNS. By making extensions to the propsed framework, we have realized in simplifing to maintain the enlarged database, and decreased managing cost.

Key Word
1. DNS 2. Database 3. Dynamic Update 4. Security 5. Advanced SDB

Keio University Graduate School of Media and Governance
Tomohiro Ishihara

目 次

図 目 次

表 目 次

第1章　序論

本章では、背景として DNS が現在抱える問題点を説明する。また、本研究の目的について述べ、さらに本論文の構成について述べる。

1.1　背景

Domain Name System(DNS)[2] は、インターネットにおいて重要な役割を果たしている。

DNS は、インターネットで最も利用されている広域分散協調データベースであり、主にインターネットプロトコル (IP)[4] のアドレスとホスト名の変換を行っている。これにより、ユーザーは人間が記憶しにくい IP アドレスのかわりに、一意な名前によってインターネット上の各ホストの識別をすることができる。

DNS は、15 年以上前に基礎設計されたものであるが、その後、インターネットの急激な発展と普及により、いくつかの拡張が考えられた。

その一つが動的更新である。動的更新が提案されるまでの DNS は、すべての設定を管理者が手動で行なうことが前提だった。しかし、動的更新を利用することによって、クライアントがデータの変更・追加を管理者の手を介さずに行なうことができる。

もう一つがセキュリティ機構である。従来は DNS のデータは無保証で、虚偽のデータを送ることができた。そこで、秘密共有鍵を使って署名を行なうといったセキュリティ機構の拡張により、DNS のデータの内容を保証した。

しかしこれらの拡張により、設定ファイルは肥大化および複雑化し、従来のようなテキストファイルでの管理が難しくなった。

そのため、BIND[1] において DNS のデータを外部データベースに格納し、管理するためのインターフェースである (SDB: Simple Database Interface) が開発された。しかし、これは限定的な機能しかもっておらず、データの動的な更新や、鍵などのリソースレコード以外の設定情報を管理することができない。

本研究では、この問題を解決するために SDB の拡張を提案する。

1.2　目的

本研究では、現在の DNS における静的情報の肥大化および複雑化に対応するため、SDB を拡張したシステムを提案し、その設計と実装および評価を行う。

1.3　本論文の構成

まず第 2 章において、研究対象のシステムである DNS の概要について述べ、現在の運用状況とその問題点を述べる。

第 3 章でその問題点を解決するための手法を提案し、第 4 章で提案するシステムの設計について説明する。第 5 章で実装の詳細について述べ、第 6 章でそのシステムの評価を述べる。第 7 章で本研究のまとめと今後の課題について述べる。

第2章　DNS の現状

DNS は、1987 年に基礎設計されたものであるが、その後、インターネットの急激な発展と普及により、いくつかの拡張が考えられた。本章ではそれらの拡張を含めた DNS の現状と、現在の DNS が抱える問題点について述べる。

2.1　DNS に対する新たな要求

DNS が設計された頃のインターネットは、まだ接続ホストも少なく、アドレスや名前空間がほとんど使われていなかった。そのため、DNS の静的情報は小さく、変更が行われることが少なかった。また、大部分のユーザーが研究者がであったため、セキュリティに関する考慮は全くされていなかった。

しかし商用プロバイダが登場し、インターネットが世界的に普及するとともにユーザー数が爆発的に増加し、接続ホスト数の増加した。そのため、DNS の静的情報を肥大化し、データの変更も頻繁に行われるようになった。また、悪意を持つユーザーも多く現れ、DNS もセキュリティに関して注意を払わなければならなくなった。

2.2　新しく追加された DNS の機能

前述のように、従来の DNS に対して見直しが必要となり、いくつかの拡張が施された。ここでは、その新しく DNS に追加された機能を述べる。

2.2.1　動的更新

動的更新は、権威サーバにリソースレコードの追加および削除を行う機構である。

従来の DNS では、リソースレコードを書き換えるためにサーバの設定ファイルを直接書き換えなければならなかった。しかし、動的更新を利用することにより、クライアント側からリソースレコードの追加および編集が行える。

この機構を利用すると、クライアントのアドレスが変化したとき、管理者が設定を手動で変更することなく、アドレスと名前の対応を新しいものに変えられる。これによって管理コストを減らすことができ、管理者の存在しない環境、たとえば一般家庭などでも DNS を運用することが可能となる。また、DHCP やプロバイダへのダイアルアップなど、アドレスが変化する環境においても一意な名前で通信先の特定ができる。

2.2.2 セキュリティ機構

もう一つ追加された機能としてセキュリティ機構がある。DNS は分散データベースという性質上、一元的なセキュリティの確保が困難である。一般的なインターネット上のプロトコルがサーバとクライアント各 1 つずつ、合計 2 つの対象に関して考慮すれば済むことに対して、DNS においては複数のサーバについて考慮しなければならない。そのため、DNS のセキュリティ機構には 2 点間のサーバの通信内容を保証する機構だけでなく、リソースレコードの正当性を保証する機構が存在する。前者の通信内容を保証する機構が TSIG [7] であり、後者のリソースレコードそのものを保証する機構が DNSSEC [6] である。

TSIG

TSIG は、秘密共有鍵を用いて、DNS サーバとクライアントとの間で行われるトランザクションの認証を行う技術である。サーバとクライアントで秘密鍵を共有することで、あらかじめ認証されたクライアント以外からのサーバの利用を制限したり、DNS メッセージの完全性を認証できる。また、サーバ間で TSIG を適用することにより、ゾーン転送の認証や、アクセス制限を行うことが可能となる。ただし、秘密共有鍵を用いているため、あらかじめ信頼できる方法によって鍵を共有する必要がある。また、利用するユーザーの数だけ共有鍵をもっていなければならない。(違うユーザーで同じ鍵を共有する方法もあるが、セキュリティ的に問題がある)

DNSSEC

DNSSEC は、公開鍵暗号方式を用いて、リソースレコードの正当性を保証する技術である。まず、あるゾーンに対して、秘密鍵と公開鍵の鍵対を作成する。そして、ゾーンに存在する各リソースレコードに秘密鍵を用いて署名を行い、その署名と公開鍵をリソースレコードとして公開する。名前解決を行なうクライアントは、署名と公開鍵を用いて、データの正当性の確認を行う。このようにして、データの正当性を保証する。

この際、配布されている公開鍵の正当性を保証するために、公開鍵自体も、そのデータの起源が保証されている必要がある。すなわち、公開鍵自体が、上位のゾーンの秘密鍵を使って署名されている必要がある。こうして、上位ゾーンが下位ゾーンの公開鍵を署名して、それをゾーンの委譲に沿って繋げていくことによって、DNS のツリー構造全体のデータを保証する。

2.2.3 新しいレコードの追加

DNS の分散データベースという利点を生かして、電子証明書や、サーバの物理的な位置情報などといったそれほど大きくないデータを管理することが考えられた。新しいレコードにはいくつかの種類が存在する。以下にその例を示す。

cert record RFC2538 [8] で規定されている電子証明書のレコード

loc record RFC1876 [9] で規定されている位置情報 (経度、緯度等) を示すレコード

2.3 現状の問題点

以上のように DNS に対する拡張が考えられたが、これより DNS はさらに複雑化、高度化し、いくつかの問題が発生している。

2.3.1 静的情報の肥大化

BIND は、テキスト形式による設定ファイルでデータを管理している。そのため、インターネット上のノード数の増加や、リソースレコードの多様化によって、設定ファイルが肥大化し、管理が煩雑となる。

肥大化した設定ファイルの変更を反映させる際には、設定ファイルを最初からすべて読み直し、字句および構文の解析を経て情報を取得する必要がある。このため、肥大化した設定ファイルの情報を書き換えて動作に反映させる場合、時間がかかる。

巨大なゾーンを管理しているいくつかのサイトでは情報をデータベースによって管理している。しかし、実際に使用する際はそれらの情報を一旦スクリプト等でテキストに落としてから使用するなどといったことを行っており、これらの動作にも時間がかかる。

2.3.2 動的更新の運用上の問題

動的更新を行う場合、現実的には共有鍵を利用する TSIG を使用することがセキュリティの上で必須である。現状では、共有鍵はゾーンごとに関連付けられ、ある共有鍵を持つユーザーはその鍵に関連付けられたゾーン全てを変更することが可能となる。

しかし ISP 等で実際に運用する場合には、特定の鍵で、つまり特定のユーザがゾーン全てを編集できる場合では、名前を変更する事によって誤ったホストへのアクセスをしてしまうという問題がある。図 2.1 に例を示す。

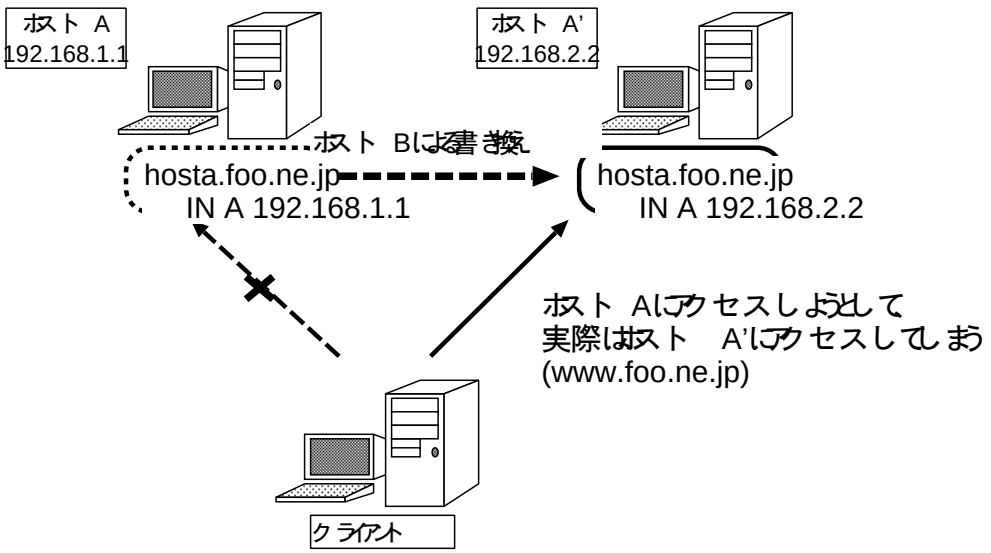

図 2.1: 名前の変更による誤ったホストへのアクセス

当初、hosta.foo.ne.jp の A リソースレコードとしてホスト A (アドレス: 192.168.1.1) が登録されていたとする。その後、ホスト A'(192.168.2.2) が hosta.foo.ne.jp のリソースレコードを自分のアドレスを指示するように書き変えてしまうと、ホスト A にアクセスしようとしているクライアントがホスト A' にアクセスしてしまうことになる。この問題は、最悪の場合不正アクセスなどに利用される可能性もある。

このため動的更新を行う際には、特定の鍵で特定のリソースレコードのみ編集可能であることが望ましい。これを可能にするためには、鍵をゾーン全体ではなく、特定のリソースレコードと関連づけなければならない。

2.3.3 ユーザ固有の情報の管理

上記のように ISP でユーザごとに共有鍵を使って動的更新を実施する場合、それぞれのユーザに関連付けられた鍵およびリソースレコードといった情報を、その他のユーザ情報が管理されるデータベースとは別にネームサーバの設定ファイルによって管理する事は情報の分散を生み、管理コストが高くなる。

2.4 問題点に対するアプローチ

以上の問題は、ネームサーバがテキスト形式による設定ファイルで動作の設定を行っていることと、ユーザ固有の情報を自身で管理していることから発生している。

巨大なゾーンを扱うような大きなデータや、それぞれの要素が関連づけらているデータを扱う場合、リレーショナルデータベースを使うのが適当である。肥大化したゾーンの管理コストを減らすことができ、相互に関連付けられたデータを整合性を崩すこと無しに編集を行える。また、データベースで使用するテーブルに対して適切なスキームを設定することにより、異常なデータの登録を防ぐことができ、再読み込みの際に字句・構文解析、およびそれに付随したエラー処理などをその度に行わずにすむという利点もある。

リレーショナルデータベース等の外部データベースを使うというアプローチはすでに BIND において実装されている。しかし、この機能は限定的なもので、データベースから情報の読み出ししか行えない。このため、この機能では動的更新といった新しい機能を使うことができない。また、扱える情報はリソースレコードのみであり、鍵などの情報を管理することはできない。

第3章　問題解決のための手法

本章では、外部データベースを使用する関連研究とその問題点をまとめ、本研究で提案するシステムの提案とその要求事項について述べる。

3.1　関連研究

以下に、本研究がターゲットとしている問題に関する関連研究を述べる。

3.1.1　Simplified DataBase interface(SDB)

前章で述べた問題を解決するために、BIND に対して機能追加が施された。Simplified DataBase interface(SDB) は、外部データベースを使うためのインターフェースである。SDB を使うことにより、ゾーンのリソースレコードを外部のデータベースを使って管理することが可能となる。標準で対応している外部データベースには、postgres[11], ldap[12] 等がある。その他のデータベースをには、SDB のインターフェースを備えたデーターベース用のドライバを実装することで対応する。

SDB の問題点

現状の SDB の機能は限られており、リソースレコードの読み出ししか行えない。そのため、SDB を使う場合は動的更新が利用できない。また、問い合わされたリソースレコードそれぞれについてデータベースサーバに問い合わせ、キャッシュを行わない。よって、データーベースサーバが違うホストで動作し、ネットワーク経由で問い合わせた場合、特にパフォーマンスが低下する。

3.1.2　MyDNS

MyDNS [10] はリレーショナルデータベースの MySQL[13] をデータベースエンジンとして、新たに開発された DNS 実装である。ゾーン情報のリソースレコードを MySQL のデータベース上に置くことができる。検索結果のキャッシュを保持しており、データベースに対する検索結果を記憶し、それ以降の検索時に使用する。

MyDNS の問題点

MyDNS はネームサーバ実装としては最低限の機能しかなく、再帰問い合わせや DNSSEC/TSIG などのセキュリティ機構、動的更新その他最近になって DNS に加えられた拡張などの機能が存在しない。

また、使用されている期間もユーザー数も少ないため、現段階では実装としての安定度に不安がある。

3.2 新しいデータベースフレームワークの提案

本研究では外部データベースを用いると共に、動的更新、セキュリティなどを考慮したデータベースフレームワークを定義することで、2.3 節で述べた問題の解決を図る。

3.3 要求事項

2.4 節で述べたアプローチを踏まえ、本システムの要求事項を以下に記述する。

3.3.1 システムの目標

本研究で狙いとしている解決すべき問題と、それに対する解決のための手法に必要な事項は以下の通りである。

- **ゾーン情報の管理の効率化**
  外部データベースを利用することで、様々な管理用ツールとの連携を容易にし、ゾーン情報の管理コストを下げる。また、ゾーン情報のテーブルに制約を設けることで、ヒューマンエラーを抑制する。
- **データ更新の高速化**
  ゾーン情報を字句解析、構文解析およびデータの正当性の評価が終った段階のデータとして保存することにより、ゾーン情報更新時の処理を高速化する。
- **ユーザーに関連したデータの一元化**
  ユーザーの鍵およびリソースレコードの変更権限などを、その他のユーザーデータと同列に管理できるようにすることで、総合的な管理コストの低減を図る。
- **動的更新時の詳細なアクセスコントロール**
  動的更新によるリソースレコードの編集を行う際に、それぞれの鍵およびリソースレコード毎にアクセスコントロールを行えるようにする。

3.3.2 システムの機能

このシステムが備えるべき機能として、以下のものが挙げられる。

1. リソースレコードの問い合わせ
   外部からの要求を受け、リソースレコードを外部データベースに対して問い合わせ、結果を報告する。要求が署名されていた場合は、3) および 4) の処理を先に行う。
2. リソースレコードの登録および更新
   外部からの要求を受け、リソースレコードの登録および更新を外部データベースに対して行う。要求が署名されていた場合は、3) および 4) の処理を先に行う。

3. TSIG によるトランザクションの署名を確認する
   署名された要求を、鍵を使って検証する。検証に使う鍵は 5) の処理を行って取得する。

4. TSIG 署名鍵のリソースレコードに対するアクセス権を問い合わせる

5. TSIG 署名鍵を問い合わせる
   内部からの要求を受け、署名鍵を外部データベースに対して問い合わせる。

3.3.3 システムの条件

システムは、上記の機能を実現させるとともに、以下の条件を満たさなければいけない。

1. 本システムでは、 扱うデータに関して、その正当性が何らかの形で保証されなければならない

2. 従来の方式である内部データベースを使う場合に比べて大幅なパフォーマンス低下があってはならない

3. 新しい外部データベースへの対応が簡単であること

3.4 SDB の拡張

本システムは BIND に実装されている SDB の拡張によって作成する。BIND は DNS 実装としてもっとも多く使われており、実績も多い。また、ソースコードも広く公開されている。ユーザーが多いため成果物を多くの人に使ってもらえるという利点もある。よって、本研究では BIND をプラットフォームとして利用できる SDB に対する拡張という方針を採った。

本システムは、BIND version 9 に対する拡張モジュールとして設計を行う。version 9 は現在の開発バージョンであり、様々な新しい試みが機能として追加されている。version 8 に関しては、すでに開発はほぼ終了しており、バグの修正しか行われないバージョンであるためターゲットから除外した。

第4章　拡張 SDB システムの設計

本章では、本論文で提案する拡張 SDB システムの設計について述べる。

4.1　拡張 SDB システムの概要

拡張 SDB システムは、従来の SDB にリソースレコードの登録および削除インターフェース、検索結果に対するキャッシュ、署名鍵の管理と鍵によるアクセスコントロールを加えたものである。

本システムは BIND version 9 に対する追加モジュールとして設計を行う。

4.2　基本設計

3 章で述べた要求に従い、以下のように設計した。

- **動作部と通信部の分離**
  さまざまな種類のデータベースに対応させるため、システムは BIND 側のインターフェースになり、本システム全体の動作を管理する部分と、外部データベースとの通信を担当する部分に分ける。これより、新しい外部データベースには、外部データベースへの通信部分のみ実装すれば対応できるようにもなる。

  動作部と通信部は定められたインターフェースによって通信し、新しい外部データベースへの対応を容易にするため、通信部は外部データベースの違いを吸収できる程度の最低限の機能しか要求しないようにする。

  通信部および外部データベースは、動作部の要求に対して、インターフェースで規定されているデータを答えることさえできれば、外部データベースのテーブル構造および通信部の構造は自由に決めてよい。

- **ゾーン情報ドライバに編集用命令を用意**
  リソースレコードの動的更新のため、従来の SDB が持つリソースレコードを取得するドライバに、新たにリソースレコードの追加と削除のインターフェースを加える。

- **ゾーン情報、鍵情報、権限付与情報の 3 つを外部データベースで管理**
  従来の SDB にある、リソースレコードを管理するためのデータに加えて、まず、TSIG で使用する鍵を管理するデータ、そしてリソースレコードごとのアクセスコントロールを行うために、鍵とアクセス権の関連を管理するデータの 3 つの外部データベースで管理ができるようにする。
  このため、従来のリソースレコード検索用のドライバに加えて、 TSIG で使用する鍵の検索用ドライバ、鍵によるリソースレコードのアクセス権限チェック用のドライバあわせて 2 つを追加する。

鍵検索用ドライバは TSIG の鍵 ID より TSIG の鍵のデータ本体を取り出す。アクセス権限チェック用のドライバは指定された鍵で対象とするリソースレコードおよびゾーンに対してオペレーションが行えるかどうかを判別する。

- **それぞれのドライバを独立**
  ゾーンの情報と、鍵の情報それぞれを違うデータベース実装で管理する、例えばゾーンの情報はリレーショナルデータベースで、鍵の情報は radius でそれぞれ管理するといった場合に対応できるようにするため、ゾーン情報、鍵情報、権限付与情報それぞれを独立したドライバとする。インターフェースもそれぞれに独立のものを定義する。

- **キャッシュを保持**
  本システムでは外部データベースにクエリーを出して検索を行うので、内部的にデータを持っている場合に比べてどうしてもオーバーヘッドが生じてしまう。そこで、本システムではデータベースに対する問い合わせ結果をキャッシュし、次回以降の問い合わせに利用することで、オーバーヘッドを軽減する。データベースとキャッシュの整合性を維持するために、本システムを経由せずにデータベースに変更が加えられた際には、データベースのトリガ動作等で BIND に対して通知し、キャッシュを破棄する。

4.3 システム概観

図 4.1 にシステムの概観を示す。

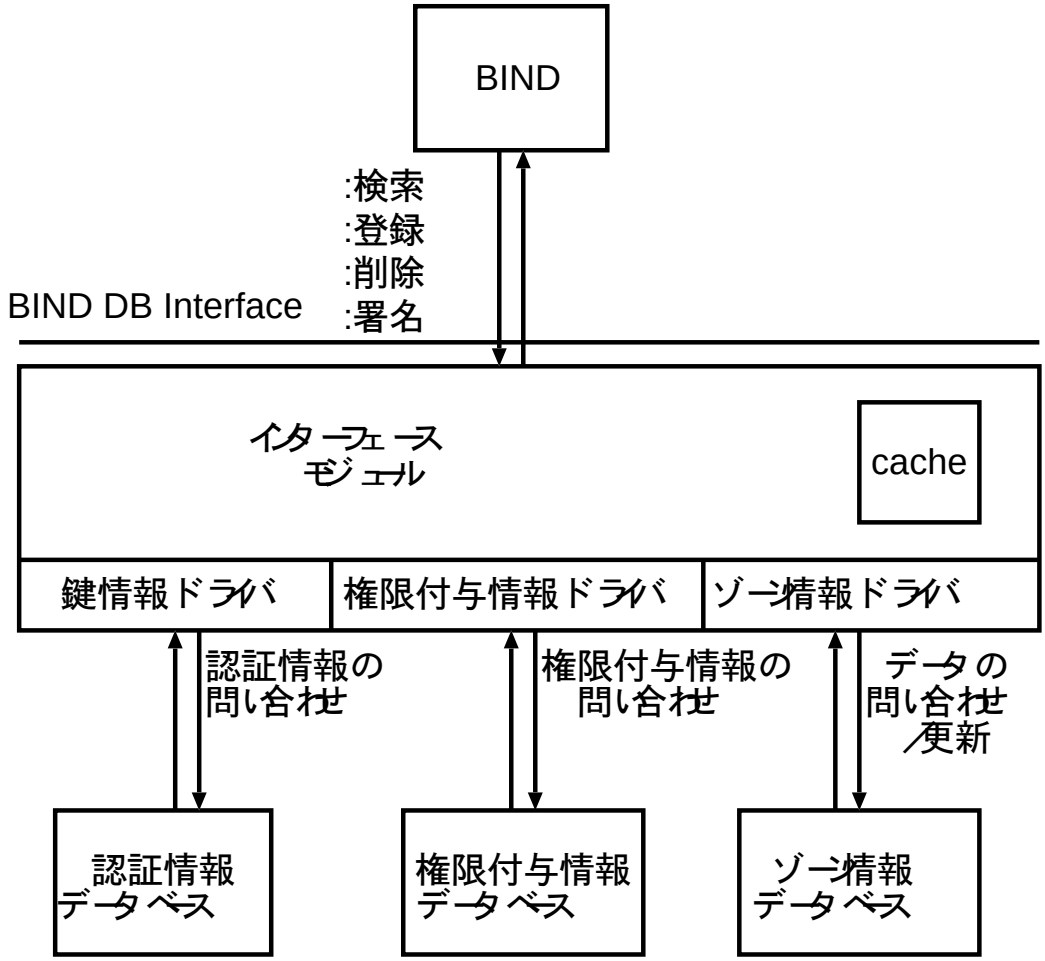

図 4.1: 拡張 SDB の概観

システム概観の各部分についての説明は以下の通り。

インターフェースモジュール

本システムの全体的な動作を担当するモジュール。BIND に対するインターフェース部分となり、BIND 本体から発行された検索や登録、削除といった命令を外部データベースへの窓口となるドライバを使いつつ実行する。各種ドライバに非依存な処理はすべてここで行う。

鍵情報データベース/鍵情報ドライバ

TSIG で署名の検証に使用する鍵を格納するデータベースと、そのデータベースにアクセスするためのドライバ。TSIG の鍵 ID より、鍵データと署名に使うアルゴリズムを取得する。

権限付与情報データベース/権限付与情報ドライバ

リソースレコードに対するアクセス権を管理するためのデータベースと、アクセス権の確認をするためのドライバ。さまざまなアクセス条件を構成できるように、ドライバ内でアクセス権の有無を検証し、その結果のみをインターフェースモジュールに通知する。

ゾーン情報データベース/ゾーン情報ドライバ

リソースレコードを格納するデータベースと、そのデータベースにアクセスするためのドライバ。リソースレコード名よりデータを取得する。

4.4 システムの全体的な動作

システムは以下のように動作する。

1. BIND はインターフェースモジュールに対して検索／更新などの命令を発行する。
2. 命令を受け取ったインターフェースモジュールは鍵情報ドライバに指定した鍵 ID に対応する鍵データを取得を命令する。
3. インターフェースモジュールは取得した鍵で署名を検証する。
4. 署名が正しく検証できたら、命令が正当な権限のもとに行われているかを権限付与情報ドライバに確認する。
5. 確認できた場合、ゾーン情報ドライバを通して命令を実行する。
6. 得られた実行結果を呼び出し元に送る。
7. 実行の際に手にいれたデータベースの情報はキャッシュに保存する。

例として、検索を行う場合の各モジュール間のシーケンス図を図 4.2 に示す。

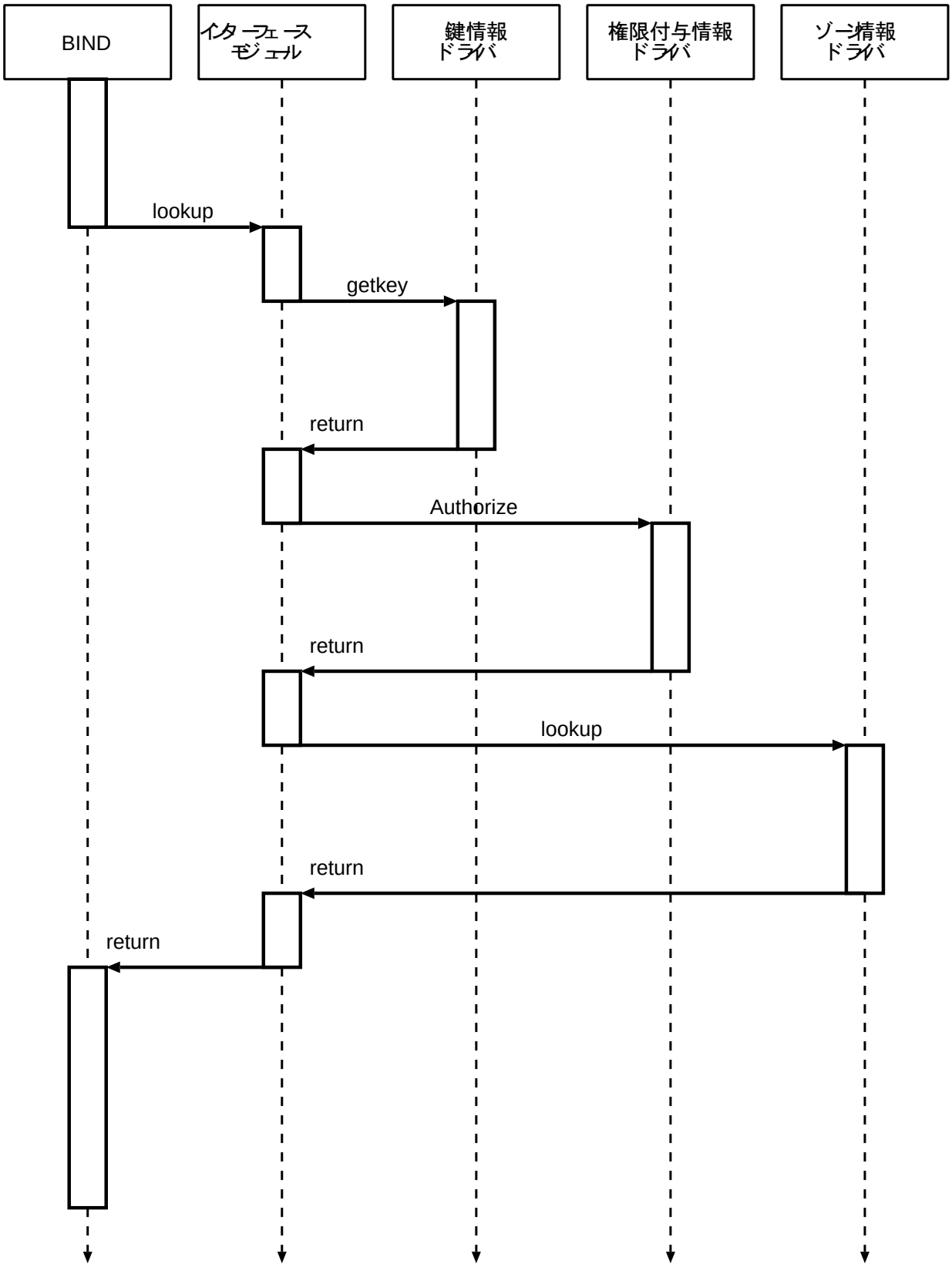

図 4.2: 動作のシーケンス図 (検索の場合)

4.5 データベースドライバの仕様

各データベースドライバは BIND の初期化時にドライバ固有の初期化ルーチンが呼び出され、その中でドライバのインターフェースを実装している関数の関数ポインタ群を呼び出し側に登録する。

BIND は登録されたドライバから鍵情報ドライバ、権限付与情報ドライバ、ゾーン情報ドライバそれぞれについて設定によって決められたドライバを選んで使用する。各ドライバはデータベースへの接続・切断用のインターフェースを持ち、ドライバ選択時に接続命令が呼び出される。図 4.3 に接続・切断インターフェースを示す。

```
/* データベースへの接続 */
typedef isc_result_t
(*dns_sdbcreatefunc_t)(const char *zone, int argc, char **argv,
                       void *driverdata, void **dbdata);

/* データベースからの切断 */
typedef void
(*dns_sdbdestroyfunc_t)(const char *zone, void *driverdata, void **dbdata);
```

図 4.3: ドライバ接続・切断関数

dns_sdbcreatefunc_t は、zone に対象とするゾーン名、argc, argv には接続設定が代入され呼び出される。この関数ではデータベースへの接続を確立する。dbdata は呼び出し側に保存されるポインタで、呼び出し側との情報の受け渡しや、ドライバの状態を記憶するために使用する。dbdata はここで設定した値が各データベース関数の呼び出し時に指定される。汎用ポインタである dbdata の指示する先のメモリ領域はドライバ内で確保し、dns_sdbdestroyfunc_t が呼び出されたときに責任を持って解放しなければならない。確保するメモリ領域の構造はドライバ側で自由に決めてよい。driverdata は将来のための予約となっている。

argc, argv で指定する接続設定には、設定ファイルに記述してある文字列がそのまま代入され、その書式は自由である。これは、データベース実装によって接続に必要なデータが大きく違い、統一的な書式を規定できないためである。

dns_sdbdestroyfunc_t は、ゾーン名 (zone) と dns_sdbcreatefunc_t で得られた driverdata と dbdata が代入されて呼び出される。この関数はデータベースへの接続を切断する。

dns_sdbcreatefunc_t および dns_sdbdestoryfunc_t はそれぞれ接続・切断が失敗したときに偽を返す。

以下に、各ドライバのインターフェースの仕様について説明する。ドライバはこれらのインターフェースの仕様に従って実装しなければならない。

4.5.1 鍵情報ドライバ

鍵情報ドライバは、指定された鍵 ID から、鍵データと署名アルゴリズムを検索するためのドライバである。鍵情報ドライバは以下のインターフェースを持っている。

```
/* 鍵情報ドライバ インターフェース構造体 */
typedef struct dns_sdbkey_methods {
    dns_sdbgetkeyfunc_t    getkey;          /* 鍵データの取得 */
    dns_sdbcreatefunc_t    create;          /* データベースへの接続 */
    dns_sdbdestroyfunc_t   destroy;         /* データベースからの切断 */
} dns_sdbmethods_t;
```

図 4.4: 鍵情報ドライバインターフェース

create と destroy は接続・切断用のドライバ共通関数である。その他の関数については以下の通り。

```
/* 署名鍵の取得 */
typedef isc_result_t
(*dns_sdbgetkeyfunc_t)(dns_tsigkey_t **tsigkey, dns_name_t *name,
                       dns_name_t *algorithm, void *dbdata);
```

図 4.5: 鍵情報取得関数

鍵データ取得用の関数で、name に鍵の ID、alogorithm に鍵の署名アルゴリズム (HMAC-MD5 など)、dns_sdbcreatefunc_t において設定した dbdata のポインタが指定されて呼び出される。この関数は、dns_tsigkey_t に該当する鍵データをコピーする。該当するデータが発見された場合は ISC_R_SUCCESS、発見されなかった場合、この関数は ISC_R_NOTFOUND、なんらかの異常によって検索が正しく終了しなかった場合は ISC_R_FAILURE を返す。

4.5.2 権限付与情報ドライバ

権限付与情報ドライバは、リソースレコードの操作に対するアクセス権限情報を取得するためのドライバである。

```
/*権限付与情報ドライバ インターフェース構造体 */
typedef struct dns_sdbauthorize_methods {
    dns_sdbauthorizefunc_t  authorize;      /* 権限付与情報の取得 */
    dns_sdbcreatefunc_t     create;         /* データベースへの接続 */
    dns_sdbdestroyfunc_t    destroy;        /* データベースからの切断 */
} dns_sdbmethods_t;
```

図 4.6: 権限付与情報ドライバインターフェース

create と destroy は接続・切断用のドライバ共通関数である。その他の関数については以下の通り。

```
/* 権限付与情報の取得 */
typedef isc_result_t
(*dns_sdbautorizefunc_t)(dns_tsigkey_t *tsigkey, dns_name_t *name,
                         dns_name_t *zone, dns_opcode_t opcode,
 void *dbdata);
```

図 4.7: 権限付与情報取得関数

権限付与情報取得用の関数で、tsigkey に署名された鍵、name にリソースレコードの名前、zone にゾーンの名前、opcode に操作の種類、dbdata に dns_sdbcreatefunc_t において設定した dbdata が代入されて呼び出される。指定された鍵の権限で、名前で指定されたリソースレコードに対して opcode で示される操作が可能ならば ISC_R_SUCCESS、不可能ならば ISC_R_NOTFOUND、なんらかの異常によって情報の取得が行えなかった場合は ISC_R_FAILURE を返す。

4.5.3 ゾーン情報ドライバ

ゾーン情報ドライバは、ゾーンのリソースレコードを操作するためのドライバである。

```
/* ゾーン情報ドライバ インターフェース構造体 */
typedef struct dns_sdbmethods {
    dns_sdblookupfunc_t     lookup;         /* リソースレコードの検索 */
    idns_sdbaddfunc_t       add;            /* リソースレコードの追加 */
    dns_sdbdeletefunc_t     delete;         /* リソースレコードの削除 */
    dns_sdbdeleteallfunc_t  deleteall;      /* リソースレコードの全削除 */
    dns_sdbauthorityfunc_t  authority;      /* 権威サーバを取得 */
    dns_sdballnodesfunc_t   allnodes;       /* 全リソースレコードの取得 */
    dns_sdbcreatefunc_t     create;         /* データベースへの接続 */
    dns_sdbdestroyfunc_t    destroy;        /* データベースからの切断 */
} dns_sdbmethods_t;
```

図 4.8: ゾーン情報ドライバインターフェース

create と destroy は接続・切断用のドライバ共通関数である。その他の関数については以下の通り。

```
/* リソースレコードの問い合わせ */
typedef isc_result_t
(*dns_sdblookupfunc_t)(const char *zone, const char *name, void *dbdata,
                       dns_sdblookup_t *lookup);
```

図 4.9: リソースレコードの問い合わせ関数

リソースレコードの問い合わせを行う関数で、zone にゾーン名、name にリソースレコード名、dbdata に dns_sdbcreatefunc_t で指定したポインタが代入され呼び出される。指定された名前のリソースレコードをデータベースに問い合わせ、結果を lookup で指示された構造体に代入する。該当するデータが発見された場合は ISC_R_SUCCESS、発見されなかった場合、この関数は ISC_R_NOTFOUND、なんらかの異常によって検索が正しく終了しなかった場合は ISC_R_FAILURE を返す。

```
/* 権威サーバの問い合わせ */
typedef isc_result_t
(*dns_sdbauthorityfunc_t)(const char *zone, void *dbdata,
                          dns_sdblookup_t *authority);
```

図 4.10: 権威サーバの問い合わせ関数

ゾーンの権威サーバを問い合わせる関数で、zone にゾーン名、dbdata に dns_sdbcreatefunc_t で指定したポインタが代入され呼び出される。指定されたゾーンの権威サーバを問い合わせ、結果を authority で指示された構造体に代入する。該当するデータが発見された場合は ISC_R_SUCCESS、発見されなかった場合、この関数は ISC_R_NOTFOUND、なんらかの異常によって検索が正しく終了しなかった場合は ISC_R_FAILURE を返す。

```
/* ゾーン内の全てのデータの問い合わせ */
typedef isc_result_t
(*dns_sdballnodesfunc_t)(const char *zone, void *dbdata,
                         dns_sdballnodes_t *allnodes);
```

図 4.11: 全ノードの問い合わせ関数

ゾーンの全てのリソースレコードを問い合わせる関数で、zone にゾーン名、dbdata に dns_sdbcreatefunc_t で指定したポインタが代入され呼び出される。指定されたゾーン全てのリソースレコードを問い合わせ、結果を allnodes で指示された構造体に代入する。該当するデータが発見された場合は ISC_R_SUCCESS、発見されなかった場合、この関数は ISC_R_NOTFOUND、なんらかの異常によって検索が正しく終了しなかった場合は ISC_R_FAILURE を返す。

```
/* リソースレコードの登録 */
typedef isc_result_t
(*dns_sdbaddfunc_t)(const char *zone, const char *name, const void *dbdata
                    dns_sdblookup_t *add);
```

図 4.12: リソースレコードの登録関数

リソースレコードを登録する関数で、zone にゾーン名、name にリソースレコード名、dbdata に dns_sdbcreatefunc_t で指定したポインタ、add に追加するリソースレコードの情報が代入され呼び出される。指定されたリソースレコードを指定されたゾーンに追加する。正しく追加ができた場合は ISC_R_SUCCESS、指定されたリソースレコードがすでに存在する場合は ISC_R_EXISTS、何らかの異常によって登録が正しく行えなかった場合は ISC_R_FAILURE を返す。

```
/* リソースレコードの削除 */
typedef isc_result_t
(*dns_sdbdeletefunc_t)(const char *zone, const char *name, void *dbdata,
                       dns_sdblookup_t *delete);
```

図 4.13: リソースレコードの削除関数

リソースレコードを削除する関数で、zone にゾーン名、name にリソースレコード名、dbdata に dns_sdbcreatefunc_t で指定したポインタが、delete に削除するリソースレコードの情報が代入され呼び出される。指定されたリソースレコードを指定されたゾーンから削除する。正しく削除ができた場合は ISC_R_SUCCESS、指定されたリソースレコードが存在しなかった場合は ISC_R_NOTFOUND、何らかの異常によって削除が行えなかった場合は ISC_R_FAILURE を返す。

同じ名前のリソースレコード全てを削除する関数で、zone にゾーン名、name にリソースレコード名、dbdata に dns_sdbcreatefunc_t で指定したポインタが代入されて呼び出される。指定された名前のリソースレコードをすべて指定されたゾーンから削除する。1 つ以上のリソースレコードを削除した場合は ISC_R_SUCCESS、指定された名前のリソースレコードが存在していなかった場合は ISC_R_NOTFOUND、何らかの異常によって削除が行えなかった場合は ISC_R_FAILURE を返す。

```
/* リソースレコードの全削除 */
typedef isc_result_t
(*dns_sdbdeletefunc_t)(const char *zone, const char *name, void *dbdata);
```

図 4.14: リソースレコードの全削除関数

4.6 キャッシュ機構

インターフェースモジュール内にはキャッシュを設け、データベースに問い合わせをした際にその問い合わせ内容をキャッシュに記憶する。また、新しくリソースレコードをデータベースを登録した際にもその結果をキャッシュする。

インターフェースモジュールが問い合わせ要求を受け取った際にはまずモジュール内のキャッシュを参照し、結果が見つからなかった場合にデータベースに対して問い合わせを発行する。

データベースに対して外部より変更、つまりドライバを経由せず直接データベースに対して変更があった際にはデータベースより BIND に通知を行い、すべてのキャッシュを破棄する。

第5章　実装

本章では、本研究で提案したシステムのインターフェースモジュール、データベース、ドライバの実装の詳細について述べる。

5.1　インターフェースモジュールの実装

インターフェースモジュール上で実装すべき機能は以下のとおりである。

- BIND へのインターフェース
- 各データベースドライバの初期化
- 各データベースドライバの選択
- ドライバを経由したデータベースへの問い合わせ
- 検索結果のキャッシング
- TSIG 署名の検証

これらの実装の詳細について以下で述べる。

5.1.1　BIND へのインターフェース

BIND が内部的に使っているデータベースへのインターフェースを実装する。内部インターフェースの詳細については付録に記載する。

5.1.2　各データベースドライバの初期化

インターフェースモジュールは、BIND の初期化時にデータベースドライバの初期化を行う。インターフェースモジュールは、組込まれているドライバ全ての初期化ルーチンを実行し、それぞれのドライバからドライバのインターフェース構造体へのポインタを取得する。

5.1.3　各データベースドライバの選択

設定ファイルの内容に従って、鍵情報ドライバ、権限付与情報ドライバゾーン情報ドライバそれぞれについて使用するドライバを決定する。

使用するドライバを決めた後、それを通じて接続ルーチンを呼び出し、データベース操作実行のための準備を整える。

5.1.4 ドライバを経由したデータベースへの問い合わせ

定められたインターフェースを経由して、ドライバに命令を発行する。

5.1.5 検索結果のキャッシング

ドライバより得られた検索結果のキャッシングを行う。本実装では、単純なハッシュ関数を使ったキャッシュテーブルに検索結果を格納している。キャッシュしたデータは、古いものから破棄するよう実装しているが、キャッシュ領域が十分広いため明示的に破棄をしない限り、ほとんど破棄される事はない (リソースレコード数約 25 万の英語 jp ドメインでも、容量にして約 7 MByte 程度で、現在のコンピュータに置けるメモリ容量から考えると十分に小さいと言える) 。

5.1.6 TSIG 署名の検証

要求の DNS クエリーが TSIG によって署名されていた場合、鍵 ID をもとに鍵情報ドライバより鍵データとアルゴリズムを取得し、DNS クエリーの署名を確認する。

5.2 データベースおよびドライバの実装

本システムでは外部データベースに DNS のデータを問い合わせるためのインターフェースのみを規定しており、データベースのテーブルの構造やドライバの実装などに関する指定はない。

そのため、リレーショナルデータベースを利用して各データベースの関係を構築することも、別々のデータベースを独立に使用することも可能である。

ここでは、データベースのテーブルの構造を実装例として挙げ、さらにリレーショナルデータベースを利用する場合と、独立したデータベースを利用する場合の 2 つに関して、その実装を述べ、相違点を論ずる。

5.2.1 各データベースのテーブル構造およびドライバの実装

外部データベースのテーブルの構造は、以下のようになっている。

鍵情報データベース/ドライバ

鍵情報データベースは、鍵 ID より鍵のデータと署名アルゴリズムを取得する。そのため、この情報に関して単独のテーブルを作成する場合、表 5.1 のようなテーブル構造となる。

表 5.1: 鍵情報データベースのテーブル構造

内容	データ型
鍵 ID	文字列
アルゴリズム	文字列
鍵データ	バイト列

ドライバは、このテーブルに対して鍵 ID をキーとして署名アルゴリズムと鍵データを取得するように実装する。

権限付与情報データベース/ドライバ

権限付与情報データベースは、鍵 ID, リソースレコード名、レコードタイプ、行う操作の 4 つを指定して、その操作が行えるかどうかを判断する。そのため、テーブルは 表 5.2 に示す構造にし、レコードが存在する場合にその操作が行えると定義する。

表 5.2: 権限付与情報データベースのテーブル構造

内容	データ型
リソースレコード名	文字列
操作	整数
レコードタイプ	文字列
鍵 ID	文字列

ドライバは、鍵 ID, リソースレコード名、レコードタイプ、行う操作の 4 つをキーとして、そのレコードが存在するかどうかを調べる。
存在した場合は真を返し、存在しなかった場合は偽を返すように実装する。

ゾーン情報データベース/ドライバ

ゾーン情報データベースは、リソースレコード名、レコードタイプ を指定して、リソースレコードのデータと TTL を取得する。また、type が any だった場合は、指定された名前のリソースレコード全てを取得する。
そのため、このテーブルは表 5.3 のような構造になる。

表 5.3: ゾーン情報データベースのテーブル構造

内容	データ型
リソースレコード名	文字列
キャッシュ生存期間 (TTL)	整数
レコードタイプ	文字列
データ	文字列

ドライバは、リソースレコード名、レコードタイプをキーとして TTL およびデータを取得する場合と、リソースレコード名のみをキーとして、その名前全てのレコードの TTL とデータを取得する場合の 2 つを実装する。

5.2.2 リレーショナルデータベースの利用

リレーショナルデータベースを使えば、テーブルごとの関係性を利用してテーブル間の整合性を自動的に保つ事が可能となるので、管理コストを下げる事が可能となる。

図 5.1 に 5.2.1 節で説明したテーブルのリレーショナルデータベースで構成する場合における相互関係を示す。

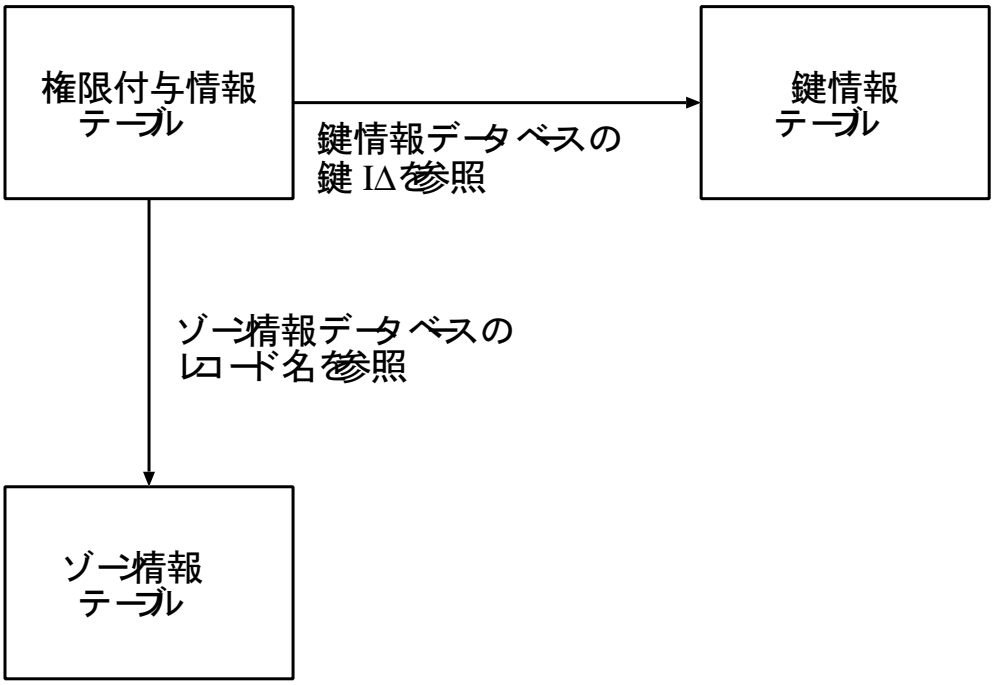

図 5.1: テーブル間の相互関係

テーブル間の関係およびテーブルの制約を挙げると、まず鍵情報テーブルにおいては、

- 鍵 ID は一意でなければならない。

よって、SQL での宣言は図 5.3 のようになる。

```
CREATE TABLE authenticate (
    KEY_ID  VARCHAR(255),
    ALGORITHM   text,
    data    BLOB,
    PRIMARY KEY (KEY_ID);
);
```

図 5.2: 鍵情報テーブルの宣言

また、鍵情報ドライバのが使うクエリーは以下となる。

```
SELECT ALGORITHM,KEY FROM authenticate where KEY_ID = <鍵 ID>;
```

図 5.3: 鍵情報ドライバで使用するクエリー

次に、権限付与情報テーブルにおいては、

- 権限付与情報テーブルにおいて、同じ内容のカラムが存在してはならない。
- 権限付与情報テーブルで使用する鍵 ID は鍵情報テーブルに存在していなければならない。
- 鍵 ID を鍵情報テーブルより削除した場合、その鍵 ID を利用した権限付与情報はテーブルから削除する。

よって、SQL での宣言は図 5.4 のようになる。

```
CREATE TABLE authorization (
    NAME    VARCHAR(255),
    OPCODE  NUMBER,
    TYPE    text,
    KEY_ID  text,
    FOREIGN KEY (KEY_ID) REFERENCES authenticate(KEY_ID) ON DELETE CASCADE,
    PRIMARY KEY (NAME, OPCODE, TYPE, KEY_ID)
);
```

図 5.4: 権限付与情報テーブルの宣言

また、権限付与情報ドライバのが使うクエリーは図 5.5 となる。

```
SELECT OPCODE FROM authorization
      where NAME = <リソースレコード名>, TYPE = <レコードタイプ>
            KEY_ID = <鍵 ID>;
```

図 5.5: 権限付与情報ドライバで使用するクエリー

最後に、ゾーン情報テーブルの制約は、

- ゾーン情報テーブルにおいて、同じ名前とレコードタイプとデータの組を持ったリソースレコードは存在してはならない。
- 権限付与情報を削除した場合、削除した権限によって追加されたリソースレコードはゾーン情報テーブルより削除する。

であり、これより SQL でのテーブル宣言は図 5.6 で示すものとなる。

```
CREATE TABLE zonedata (
    NAME    text,
    TTL     NUMBER,
    RDTYPE  text,
    RDATA   text,
    PRIMARY KEY (NAME, RDTYPE, RDATA)
);
```

図 5.6: ゾーン情報データベースの宣言

また、ゾーン情報ドライバのが使うクエリーは図 5.7 となる。

```
SELECT TTL,RDTYPE,DATA FROM zonedata
      where NAME = <リソースレコード名> [, TYPE = <レコードタイプ> ];
```

図 5.7: 権限付与情報ドライバで使用するクエリー

5.2.3 独立したデータベースの使用

独立したデータベースを使用する場合、リレーショナルデータベースを使用する場合と違い、テーブル間の整合性をデータベースに頼ることができない。そのため、以下のような方法をとる必要がある。

1. データベース編集時に整合性を取るように他のデータベースも編集する
2. 定期的にデータベースの参照関係を調べ、無効なデータを消去する (ガベージコレクション)
3. データベースのトリガ機能を使い、更新時に他のデータベースが自動的に更新されるようにする。

5.2.4 両方式の比較

リレーショナルデータベースを用いたほうが、整合性を容易に取ることができ、管理コストを押さえることができる。そのため、可能であるならば全てのテーブルを 1 つのリレーショナルデータベースで管理することが望ましい。

また、両方式は組み合わせて使うことも可能である。例えば、鍵情報は radius 等の認証サーバを使い、権限付与情報とゾーン情報はリレーショナルデータベースを使うなどである。

第6章　評価

本章では、評価用実装を使って本研究で提案したシステムの評価を述べる。

6.1　動作の検証

本システムで実現した、外部データベースを使用した動的更新、およびレコードごとの TSIG によるアクセスコントロール、それぞれの動作の検証を行う。

6.1.1　外部データベースを使用した動的更新の運用

BIND 添付の動的更新クライアントである nsupdate を使用して、動的更新機能の検証を行った。

検証は、nsupdate を利用してレコードを追加したのちに、コマンドラインのデータベースクライアントを利用して、データベースに直接 SQL のクエリーを投げて結果を確認するいう手順で行った。図 6.1 にその様子を示す。

```
## 動的更新によって、test.gt4.org という名前のリソースレコードを追加
led:~$ nsupdate
> update add test.gt4.org 300 in a 192.168.1.1
>
> ^C

## データベースクライアントで接続する
led:~$ mysql -u root -p test
Enter password:

   < 中略 >

## テーブルに追加したはずのデータを検索する
mysql> select * from gt4 where NAME = "test.gt4.org";
+---------------+------+--------+-----------------------+
| NAME          | TTL  | RDTYPE | RDATA                 |
+---------------+------+--------+-----------------------+
| test.gt4.org  |  300 | A      | 192.168.1.1           |
+---------------+------+--------+-----------------------+
1 rows in set (0.00 sec)
## 確かに存在している

mysql>
```

図 6.1: 動的更新機能の検証結果

正しく動的更新が行われていることが検証できた。

6.1.2 レコードごとの TSIG によるアクセスコントロール

同じく、nsupdate コマンドを利用して TSIG によるアクセスコントロールの検証を行った。

まず、鍵情報データベースに鍵 "key1" と鍵 "key2" を登録した。図 6.2 に登録後のテーブルの一覧を示す。

```
## 鍵情報テーブルに key1 と key2 が存在している
mysql> select * from authenticate;
+--------+-----------+------------------------------+
| KEY_ID | ALGORITHM | data                         |
+--------+-----------+------------------------------+
| key1   | HMAC-MD5  | JHF74W+xSfVHRQ98cYOFpA==     |
| key2   | HMAC-MD5  | DDRA6x/5slkrib4iMNPYNFHfb78P |
 +--------+-----------+------------------------------+
 2 rows in set (0.02 sec)
```

図 6.2: 鍵情報テーブルの内容

次に、権限付与情報データベースに test.gt4.org の A レコードを 鍵 ”key1” で登録 (OPCODE = 5) が可能なようにデータを追加した。図 6.3 に登録後のテーブルの一覧を示す。

```
## 権限付与情報テーブルにアクセスコントロール情報が存在する
mysql> select * from authorization;
+---------------+--------+------+--------+
| NAME          | OPCODE | TYPE | KEY_ID |
+---------------+--------+------+--------+
| test.gt4.org. |      5 | A    | key1   |
+---------------+--------+------+--------+
1 row in set (0.00 sec)
```

図 6.3: 権限付与情報テーブルの内容

そして、それぞれの鍵で test.gt4.org の登録を実行した。図 6.4 はその実行結果である。

```
## key1 で署名して動的更新を実行
led:~$ nsupdate -d -k Kkey1.+157+00000.key
> update add test.gt4.org 300 in a 192.168.1.1

 < 中略 >

Found zone name: gt4.org
The master is: ns1.v6.sfc.wide.ad.jp

## 正しく実行できた
Reply from update query:
;; ->>HEADER<<- opcode: UPDATE, status: NOERROR, id:   2855
;; flags: qr ra ; ZONE: 0, PREREQ: 0, UPDATE: 0, ADDITIONAL: 1
;; TSIG PSEUDOSECTION:
key1.              0       ANY     TSIG    hmac-md5.sig-alg.reg.int.
1042501383 300 16 Q2B8IOHGXT7izKDpSXhWMQ== 2855 NOERROR 0
                                              ^^^^^^^
-----

## key2 で署名して動的更新を実行
led:~$ nsupdate -d -k Kkey2.+157+00363.key
> update add test.gt4.org 300 in a 192.168.1.1

 < 中略 >

Found zone name: gt4.org
The master is: ns1.v6.sfc.wide.ad.jp

## 鍵が不正とのメッセージ
; TSIG error with server: tsig indicates error
  ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

## 実行されずエラーとなる
Reply from update query:
;; ->>HEADER<<- opcode: UPDATE, status: NOTAUTH, id:  26099
;; flags: qr ra ; ZONE: 0, PREREQ: 0, UPDATE: 0, ADDITIONAL: 1
;; TSIG PSEUDOSECTION:
key2.                   0       ANY     TSIG    hmac-md5.sig-alg.reg.int.
1042501611 300 0  26099 BADKEY 0
                        ^^^^^^
```

図 6.4: レコードごとのアクセスコントロールの検証結果

以上より、TSIG によるアクセスコントロールが正しく動作していることがわかる。

6.2 クエリーのスループット

通常の BIND version 9、SDB + MySQL、拡張 SDB + MySQL, MyDNS の 4 つの実装を用意し、条件を変化させ測定を行った。

まず、100Base-TX の Ethernet switch にサーバとクライアントのみを接続した環境で行った。測定環境は図 6.5 の通り。サーバは DNS サーバ兼データベースであり、SDB を使ったデータベースの問い合わせはローカルのデータベースに対して行う。

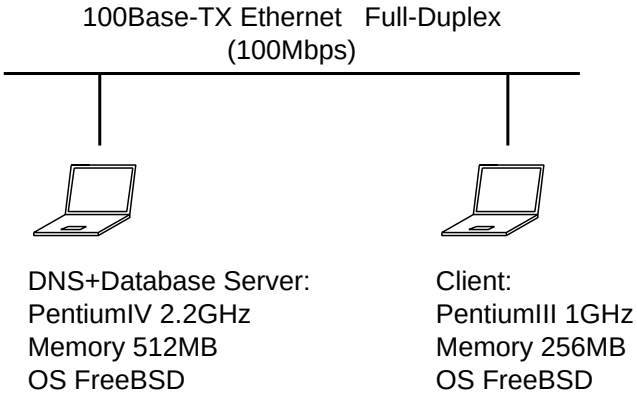

図 6.5: 実験環境：DNS サーバ、データベースサーバ同一

サーバ、クライアント共に OS は FreeBSD を使用し、サーバは CPU が PentiumIV 2.2GHz, Memrory 512MB である。クライアントは CPU PentiumIII 1GHz, Memory 256MB である。

ネームサーバは 1 万個のリソースレコードを保持し、そのリソースレコードの中からランダムに 10 万回問い合わせを出し、要した時間を計測した。時間は、クライアント側でクエリーの送信を開始した時間と最後に送ったクエリーを受信した時間を UNIX の gettimeofday() 関数を使用して測定した。

結果は図 6.6 の通りである。

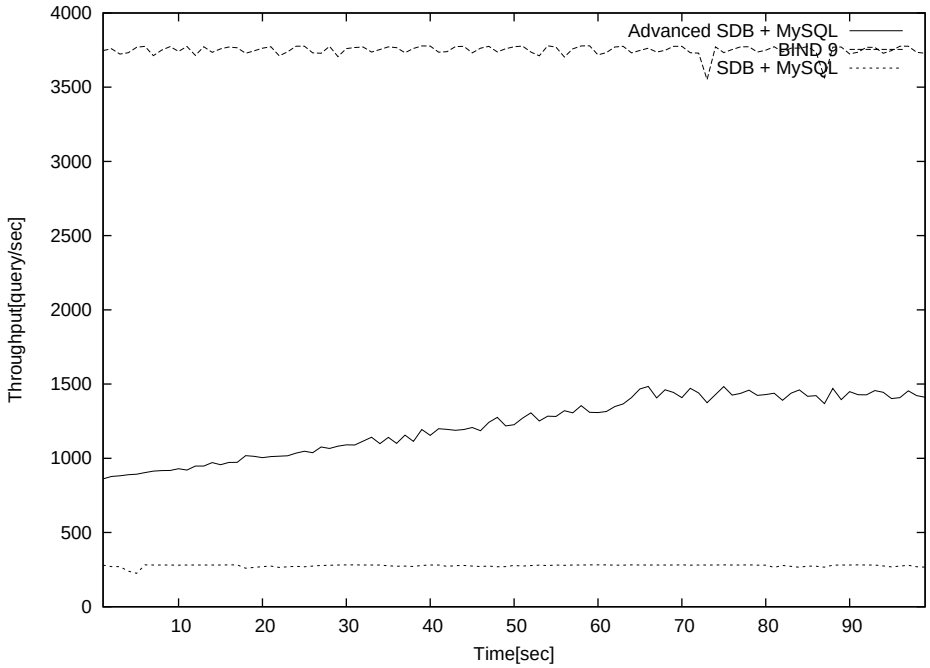

図 6.6: 実験結果：DNS サーバ、データベースサーバ同一

BIND 9 と SDB はデータベースに対するキャッシュ機構が無いのでスループットは一定である。

拡張 SDB はキャッシュがあるため、動作開始よりある程度スループットが上昇し、ある程度時間が経過すると一定になる。これは、検索を行うにつれて、ほとんどのデータがキャッシュに格納されるからである。一定になった部分を見ると、スループットは BIND 9 の 4 割程度になる。また、キャッシュの無い SDB に比べると、最高部分で 5 倍近いスループットがある。

次に、同じ Ethernet switch に DNS サーバとデータベースサーバを別々に 1 台ずつと、クライアントを接続した環境で測定を行った。SDB を使ったデータベースへの問い合わせはネットワーク経由で行う。測定環境は図 6.7 の通り。

結果は図 6.8 の通りである。

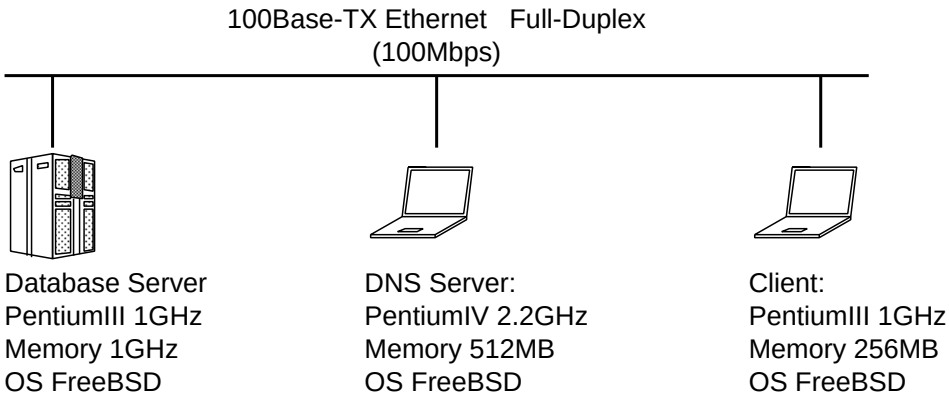

図 6.7: 実験環境：DNS サーバ、データベースサーバ別

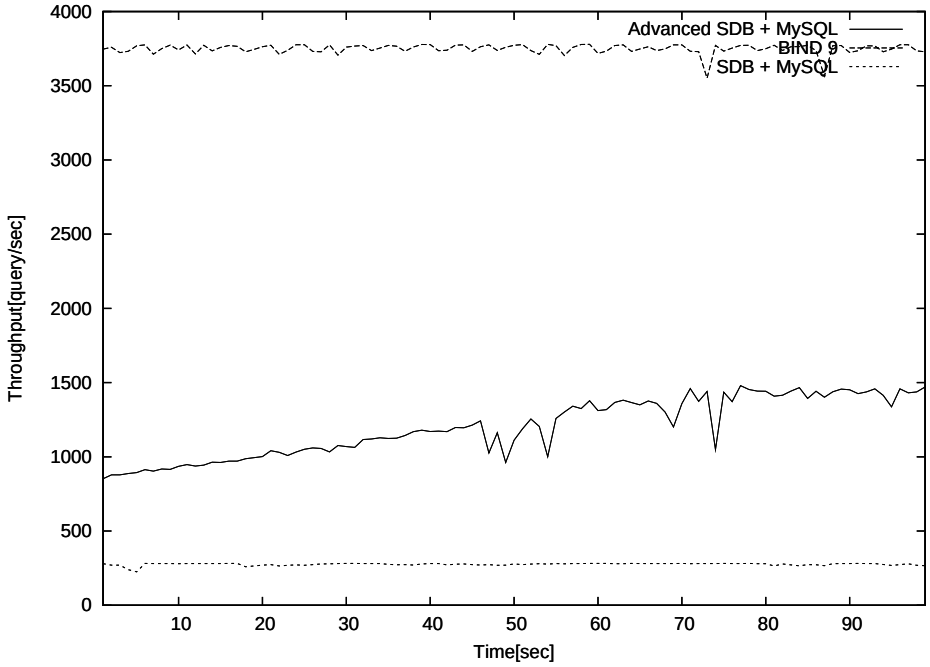

図 6.8: 実験結果：DNS サーバ、データベースサーバ別

拡張 SDB のスループットは前出の DNS サーバとデータベースサーバが同じ場合に比べて多少低い程度で、ほとんど違いはない。通常の SDB に関しても同じことが言える。

6.3 結果の考察

拡張 SDB のスループットは従来の SDB に比べて 3〜5 倍程度で、BIND の 4 割ぐらいであった。 これは、キャッシュの効果に因るものが大きいと思われる。

今回の評価実装ではリレーショナルデータベースのチューニングをしていない。よって、いくらか高速化する余地が残されていると思われる。またキャッシュに関しても、評価実装は単純なハッシュ関数を使ってキャッシュのサーチを行っている。このため、 BIND の内部データベースで使われている red black tree 等の構造を取り入れればキャッシュのサーチ性能も向上すると考えられる。これら高速化の問題については今後の課題とする。

第7章　まとめ

7.1　結論

本論文では、現在の DNS が抱えるデータベースの肥大化による問題と、動的更新の運用上の問題を説明し、それを解決するために、外部データベースを使用する仕組みの拡張を提案し、設計および実装を行った。

設定ファイルの肥大化に対して、外部データベースを使うことができ、管理コストの低下を実現した。

また、動的更新の運用上の問題に対して、権限付与情報を外部データベースで管理することにより、レコード単位でのアクセスコントロールが可能となった。

そして、リレーショナルデータベースを外部データベースとして使うことにより、ユーザ固有の情報など、さまざまなゾーン情報以外のデータも一元的に管理ができるようになった。

また、キャッシュを導入することで、パフォーマンス向上も行った。

7.2　今後の課題と展望

6 章でも述べたが、現時点ではデータベースのテーブルおよびキャッシュのデータ構造は非常に単純な方式を採用している。これらのデータ構造を見直すことによって、さらなるパフォーマンスの向上を目指す。

また、DNS はインターネットの根幹をなすシステムであるため、DNS 実装は安定して動作することと、スケーラビリティがあることが必須となる。よって、開発を続けることにより高い信頼性と規模性を確保する。

また、本研究は BIND version9 に対して実装を行った。この実装をさらに洗練させ、本研究のシステムを正式に BIND に組み込むように努め、研究成果をより多くの人が享受できるようにする。

謝辞

本研究を行うにあたり、非常に多くの方からの助言、助力をいただきました。慶應義塾大学の、徳田英幸博士、村井純博士、楠本博之博士、中村修博士、南政樹氏には日頃より暖かいご指導を賜りました。ここに心から感謝します。また、副査を快諾してくださった東京大学情報基盤センターの加藤朗氏に感謝します。

そして、本研究を進めるにあたってさまざまな助言をしてくださった ISC の Paul Vixie, Peter Loster, Michel Graff, Mark Andrews, 東芝研究開発センターの神明達哉氏、東京大学情報基盤センターの関谷勇司氏に感謝します。

最後に日頃から様々な激励と助言をいただいた、慶應義塾大学徳田・村井・楠本・中村研究会の諸氏に感謝します。ありがとうございました。

関連図書

[1] http://www.isc.org/products/BIND/

[2] P.V Mockapetris, "Domain names - concepts and

[3] P.V. Mockpertris, "Domain names - implementation and specification.", RFC1035, November 1987

[4] J. Postel, "Internet Protocol", RFC791, September 1981. facilities.", RFC1034, November 1987.

[5] P. Vixie, Editor, T. Thomson, Y. Rekhter, and J. Bound, "Dynamic Updates in the Domain Name System (DNS UPDATE)", RFC 2136, April 1997.

[6] D. Eastlake, "Domain Name System Security Extensions", RFC 2535, March 1999.

[7] P. Vixie, O.Gudmundsson, D. Eastlake, B. Willington, "Secret Key Transaction Authentication for DNS (TSIG)", RFC2845, May 2000.

[8] D. Eastlake, O. Gudmundsson, "Storing Certificates in the Domain Name System (DNS)", RFC2538, March 1999

[9] C. Davis, P. Vixie, T. Goodwin, I. Dickinson. "A Means for Expressing Location Information in the Domain Name System" RFC1876, January 1996

[10] http://mydns.bboy.net/

[11] http://www.postgres.org/

[12] M. Wahl, T. Howes, S. Kille. "Lightweight Directory Access Protocol(v3)", RFC2251, December 1997

[13] http://www.mysql.com/

付 録A　DNS のメッセージ形式

DNS のメッセージは UDP と TCP の両方を選択的に使用する。通常のデータ問い合わせおよび応答は、送信/受信のコストが低い UDP を使用し、ゾーン転送など信頼性を必要とする通信には TCP を使用する。また IP の最低受信バッファサイズが 576 byte である関係から、DNS で使用する UDP のパケットサイズの上限は 512 byte に定められている。そのため、普段は UDP を使用する通信であっても、そのサイズが 512 byte を越える場合は TCP を使用する。

以下に DNS メッセージのパケットフォーマットを示す。このフォーマットは問い合わせやその回答、およびゾーン転送などで使用されるすべての DNS のメッセージにおいて共通である。

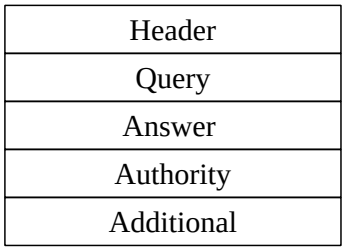

図 A.1: DNS のパケットフォーマット

図 A.1 のように、DNS のメッセージは 5 つの部分にわかれている。Header はかならず存在し、そのメッセージに含まれているセクションの数やメッセージの種別などを示す。Query はネームサーバに対する問い合わせを示す。Answer は Query に対する回答を示し、Authority はそのリソースレコードに関する情報を持っている権威サーバを示す。

Additional は Answer に関連したデータを格納する。**??** 節で説明したように、リソースレコードにはデータとして A/AAAA リソースレコードを持つものがある。これらのリソースレコードを利用して通信を行う場合は、A/AAAA リソースレコードからさらにアドレスに変換する必要があるが、この変換のためにもう一度問い合わせをネームサーバに出すのは無駄である。そのため、A/AAAA リソースレコードをデータとして持つリソースレコードが問い合わされた場合、その A/AAAA リソースレコードを Additional に格納して送信することにより、通信の無駄を省く。

A.1　Header の構造

以下に、Header の構造を示す。

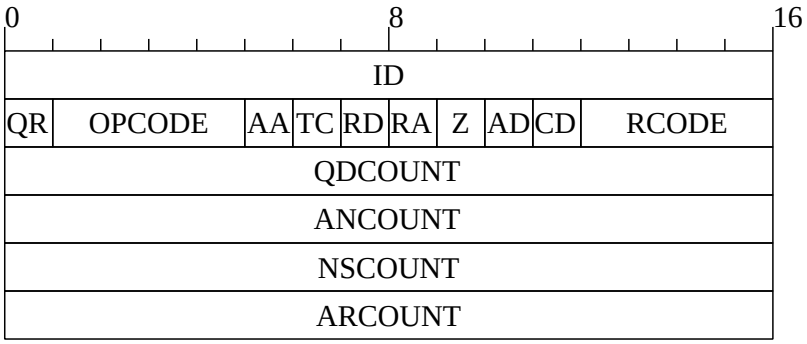

図 A.2: header の構造

表 A.1: OPCODE の値

OpCode	名前	説明
0	Query	リソースレコードの問い合わせ
1	IQuery	逆問い合わせ
2	STATUS	サーバ状態の取得
3	available for assignment	未使用で割り当て可能
4	Notify	ゾーン変更の通知
5	Update	動的更新
6-15	available for assignment	未使用で割り当て可能

ID は問い合わせを出す側が割り当てる識別子で、これにより問い合わせと応答の対応をとる。
QR はメッセージが問い合わせなのか応答なのかを識別する。0 の場合問い合わせであり、1 の場合応答である。
OPCODE はメッセージの問い合わせの型を示す。値は表 A.1 で示す通り。

AA は応答したサーバがそのゾーンの権威サーバであるかどうかを示す。
TC はパケット最大長よりメッセージが長くなったため、短縮されたことを示す。
RD はセットすることにより、サーバに問い合わせを再帰的に行うよう指示できる。
RA はサーバが再帰的な問い合わせが可能かどうかを示す。
Z は将来の拡張のために予約されている。
AD, CD は DNSSEC(2.2.2 節) で新たに定義されたビットである。AD は Answer と Authority に入っている全部のデータが DNSsec を使ってサーバにより確認されたことを示すビットで、CD はリゾルバが DNSsec による確認を行わないことを許容する場合に立てるビットである。
RCODE は応答コードで、表 A.2 で示す値に設定される。

表 A.2: RCODE の値

RCODE	名前	説明
0	NoError	エラー無し
1	FormEr	フォーマットエラー
2	ServFail	サーバに障害が発生
3	NXDomain	ドメインは存在しない
4	NotImp	実装されていない
5	Refused	クエリーは拒否された
6	YXDomain	動的更新で使用
7	YXRRSet	同上
8	NXRRset	同上
9	NotAuth	サーバーはゾーンの権威サーバではない
10	NotZone	名前はゾーンに含まれていない
11-15		未使用で割り当て可能
16	BADVERS	OPT のバージョンが不正
17	BADSIG	TSIG の署名に失敗
18	BADKEY	認められない鍵
19	BADMODE	不正な TKEY のモード
20	BADNAME	鍵の名前が重複
21	BADALG	サポートされていないアルゴリズム
22-3840		未使用で割り当て可能
3841-4095		プライベート用
4096-65535		未使用で割り当て可能

RCODE のフィールドは 4bit なので、通常は 0-15 までしか表現できない。16 以上の値は EDNS を利用することで表現する。

QNCOUNT, ANCOUNT, NSCOUNT, ARCOUNT はそれぞれ、Query, Answer, Authority, Additional の部分にいくつのデータが入っているかを示す整数である。

A.2 Query の構造

次に、Query の構造を示す。

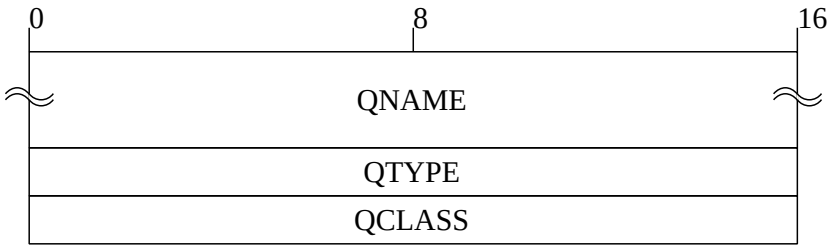

図 A.3: Query の構造

QNAME は問い合わせるリソースレコードのドメイン名を示し、QCLASS はクラス、QTYPE はタイプを示す。

A.3 Answer, Authority, Additional の構造

Answer, Authority, Additional のフィールドはすべて同じ構造をしている。フォーマットは以下の通り。

NAME はリソースレコードのドメイン名を示す。
TYPE はタイプを、CLASS はクラスを示す。
TTL はキャッシュの有効時間を示す。
RDLENGTH は RDATA のフィールドの長さを示す。
RDATA はリソースレコードのデータを示す。この情報のフォーマットは TYPE および CLASS によって変化する。

A.4 ドメイン名の表現

Query の QNAME 部分および Answer, Authority, Additional の NAME 部分はドメイン名を表わし、一連のラベルの集合として表現される。各ラベルは 1 バイトの長さフィールドと、それに続くその長さ分の文字列で表わされる。各ドメイン名はルートが最後にくるため、長さ 0 を示すバイトで終わる。

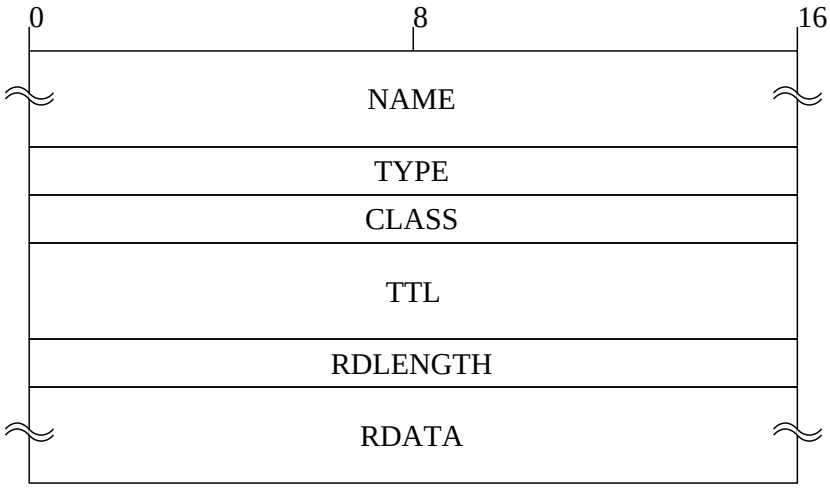

図 A.4: Answer, Authority, Additional の構造

長さを示すバイトの上位２ビットは０であるように定められている。そのため、１つのラベルの長さは 63 バイトに制限される。

またドメイン名は、すでに DNS メッセージの中で使われたラベルへのポインタを示すことで、繰り返し部分を省略することができる。ポインタは、以下で示すような２バイトからなる。

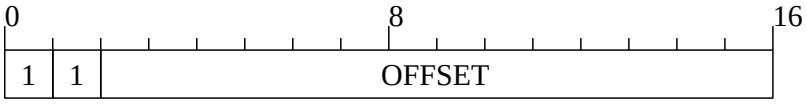

図 A.5: label の構造

先頭の２ビットは１である。ラベルであるならば先頭の２ビットは０のはずなので、ラベルとの違いを識別することができる。OFFSET フィールドは、DNS メッセージからのオフセットを示す。

付 録B　BIND の内部データベースインターフェース

```
typedef struct dns_dbmethods {

    /* データベースの割り当て */
    void        (*attach)(dns_db_t *source, dns_db_t **targetp);

    /* データベースの切り離し */
    void        (*detach)(dns_db_t **dbp);

    /* データベースの読み込みを開始 */
    isc_result_t    (*beginload)(dns_db_t *db, dns_addrdatasetfunc_t *addp,
                                 dns_dbload_t **dbloadp);

    /* データベースの読み込みを終了 */
    isc_result_t    (*endload)(dns_db_t *db, dns_dbload_t **dbloadp);

    /* データベースのテキストファイルへの書き出し */
    isc_result_t    (*dump)(dns_db_t *db, dns_dbversion_t *version,
                            const char *filename);

    /* 現在の version を割り当て */
    void        (*currentversion)(dns_db_t *db,
                                  dns_dbversion_t **versionp);

    /* 新しい version を開く */
    isc_result_t    (*newversion)(dns_db_t *db,
                                  dns_dbversion_t **versionp);

    /* version の割り当て */
    void        (*attachversion)(dns_db_t *db, dns_dbversion_t *source,
                                 dns_dbversion_t **targetp);

    /* version を閉じる */
    void        (*closeversion)(dns_db_t *db,
                                dns_dbversion_t **versionp,
```

```
                        isc_boolean_t commit);

/* ノードを検索する */
isc_result_t    (*findnode)(dns_db_t *db, dns_name_t *name,
                        isc_boolean_t create,
                        dns_dbnode_t **nodep);

/* 条件にもっともマッチする要素を検索 */
isc_result_t    (*find)(dns_db_t *db, dns_name_t *name,
                        dns_dbversion_t *version,
                        dns_rdatatype_t type, unsigned int options,
                        isc_stdtime_t now,
                        dns_dbnode_t **nodep, dns_name_t *foundname,
                        dns_rdataset_t *rdataset,
                        dns_rdataset_t *sigrdataset);

/* もっとも深い zonecut を検索 */
isc_result_t    (*findzonecut)(dns_db_t *db, dns_name_t *name,
                               unsigned int options, isc_stdtime_t now,
                               dns_dbnode_t **nodep,
                               dns_name_t *foundname,
                               dns_rdataset_t *rdataset,
                               dns_rdataset_t *sigrdataset);
/* ノードの割り当て */
void        (*attachnode)(dns_db_t *db,
                          dns_dbnode_t *source,
                          dns_dbnode_t **targetp);
/* ノードの切り離し */
void        (*detachnode)(dns_db_t *db,
                          dns_dbnode_t **targetp);

/* ノードに期限切れのマークをつける */
isc_result_t    (*expirenode)(dns_db_t *db, dns_dbnode_t *node,
                              isc_stdtime_t now);

/* ノードを表示 */
void        (*printnode)(dns_db_t *db, dns_dbnode_t *node,
                         FILE *out);

/* 列挙子を作成 */
isc_result_t    (*createiterator)(dns_db_t *db,
                                  isc_boolean_t relative_names,
                                  dns_dbiterator_t **iteratorp);
```

```
/* データセットを検索 */
isc_result_t    (*findrdataset)(dns_db_t *db, dns_dbnode_t *node,
                                dns_dbversion_t *version,
                                dns_rdatatype_t type,
                                dns_rdatatype_t covers,
                                isc_stdtime_t now,
                                dns_rdataset_t *rdataset,
                                dns_rdataset_t *sigrdataset);

/* 全てのデータセットを検索 */
isc_result_t    (*allrdatasets)(dns_db_t *db, dns_dbnode_t *node,
                                dns_dbversion_t *version,
                                isc_stdtime_t now,
                                dns_rdatasetiter_t **iteratorp);

/* データセットの追加 */
isc_result_t    (*addrdataset)(dns_db_t *db, dns_dbnode_t *node,
                               dns_dbversion_t *version,
                               isc_stdtime_t now,
                               dns_rdataset_t *rdataset,
                               unsigned int options,
                               dns_rdataset_t *addedrdataset);

/* データセットよりデータを削除 */
isc_result_t    (*subtractrdataset)(dns_db_t *db, dns_dbnode_t *node,
                                    dns_dbversion_t *version,
                                    dns_rdataset_t *rdataset,
                                    unsigned int options,
                                    dns_rdataset_t *newrdataset);
/* データセットの削除 */
isc_result_t    (*deleterdataset)(dns_db_t *db, dns_dbnode_t *node,
                                  dns_dbversion_t *version,
                                  dns_rdatatype_t type,
                                  dns_rdatatype_t covers);

/* データベースが安全か判断 */
isc_boolean_t   (*issecure)(dns_db_t *db);

/* データベース内のノード数を数える */
unsigned int    (*nodecount)(dns_db_t *db);

/* データベースが persistent か判断 */
```

```
    isc_boolean_t   (*ispersistent)(dns_db_t *db);

    /* キャッシュクリーニングの on / off */
    void        (*overmem)(dns_db_t *db, isc_boolean_t overmem);
} dns_dbmethods_t;
```