卒業制作 2003 年度 (平成 15 年度)

分散ハッシュテーブルを用いたONSの設計と実装

指導教員

徳田 英幸

村井 純

楠本 博之

中村 修

南 政樹

慶應義塾大学 環境情報学部

廣瀬 峻

t00811sh@sfc.keio.ac.jp

平成 16 年 1 月 15 日

概 要

RFID の普及に伴い、自動個体認識技術を応用した様々な研究が行われている。その中の一つとして、EPC Global、Auto-ID Lab が中心となって研究を進めている Auto-ID システムが挙げられる。Auto-ID システムでは、製品に EPC と呼ばれる ID を持った RFID タグを付け、この ID を無線を用いて読みとることで、個々の製品を識別し、製品の出荷情報や位置情報といったメタ情報をネットワーク上に展開し、管理する利用モデルを提唱している。

製品の情報をネットワーク上に展開し、管理するためには、情報を保持したサーバを特定する名前解決機構が必要である。現在の Auto-ID システムでは、製品に割り振られた EPC を検索キーとして用い、インターネットの名前解決機構である DNS を用いて目的のサーバを特定する方法が考えられている。

ONS に DNS を用いた場合について考え、運用面から名前空間の管理権限について検討を行った。システムの規模性という観点から耐故障性、負荷分散、局所性の三つの事項について検討を行った。また、非構造的な名前空間に対する名前解決について検討を行った。

本研究では、Auto-ID システムの名前解決機構である ONS について、システムとしての必要要件を整理し、DNS とは異なる名前解決機構を分散ハッシュという技術を用いて構築した。

本研究により、Auto-ID システムにおける名前解決機構の一実装を行い、名前解決を行うことができた。これにより、EPC に限定されることなく、様々な ID 空間に対応した名前解決機構を構築することができた。

キーワード

1, 名前解決　2, 分散ハッシュ　3, Object Name Service　4, Auto-ID

慶應義塾大学 環境情報学部
廣瀬 峻

abstract

With the popularization of RFIDs, various researches using the automatic individual recognition technology are being accelerated for its infinite possibility. One of these is the Auto-ID system lead by EPC Global and Auto-ID Laboratory. The Auto-ID system propose a use case model where products are maintained by attaching RFID tags which have an ID called EPC, and detecting the tags over radio transmission allowing identification of individual products. The meta information of each product, such as the shipping status and their location, are maintained over a network for use with various applications.

When maintaining product information over a network, it is necessary to have a name resolution feature which resolves the specific server holding the product's information. The proposed method for this in the current Auto-ID system uses the same resolution method used in the Internet called Domain Name Server (DNS). EPC allocated in each product is used as keys when searching for the destination server.

The EPC used in the Auto-ID retain its uniqueness by structuring the ID from the vendor ID, the product number, and the serial number. This structure involves a concern for privacy, since anyone capable of reading the ID can easily guess the product itself. Thus, a mechanism to encrypt the ID is needed and is currently being proposed to insure privacy. However, the proposed method uses plain text when handling the ID by decrypting the encrypted ID. In a case with package delivery service using Auto-ID, the sender may use the encrypted ID for the product before handing the package to the postman. If plain text is used when handling the product, any third party can obtain the original ID. Therefore, it is necessary to utilize the encrypted ID without any decryption. When applying this in the Auto-ID system, name resolution system which can handle ID that are not in a structured format, same as those encrypted ID, is necessary.

In this research, we design and implement a name resolution system capable of handling the mapping of unstructured ID with its server holding the meta information, as an extension to the proposed Auto-ID system. This name resolution independent ID format was realized by using Distributed Hash Table (DHT), mapping search keys with the hash space.

From this research, name resolution system capable of handling any sophisticated ID format is realized. By covering the meta information which were visible in the original system to provide privacy, and still retaining the uniqueness of the ID, the effectiveness of the system was shown.

Keywords

1, Name Resolve　2, Distributed Hash Table　3, Object Name Service　4, Auto-ID

Faculty of Environmental Information, Keio University
Shun Hirose

目 次

図 目 次

表 目 次

第1章　序論

本章では、本研究の背景および研究の目的について述べる。また本論文の構成を示す。

1.1　背景

Radio Frequency IDentification(RFID) の普及により個体自動認識技術が広く展開しつつある。

現在 RFID は、RFID タグや IC カードなどの形で、広く普及している段階にある。例えば、RFID を使った IC カードの代表例として、Sony が開発した Felica[1] が挙げられる。Felica は、JR 東日本の改札に採用されている Suica[2] に使われており、電波読み込みのためのアンテナを内蔵した非接触型 IC カードに情報を内蔵している。これにより、利用者は、財布や定期入れから切符や定期を出さずに、財布や定期入れをそのまま改札にかざすだけでスムーズに改札を通ることができる。また、バスカードを IC カードにすることにより、料金精算の合理化、不正防止、路線毎の情報収集などを目的として導入された「長崎スマートカード」[3] や、FeliCa の技術を用いた電子マネーサービス「Edy」[4] などが存在する。この他にも、RFID を用いた例として、図 1.1に示したように、Omron の近接タグを用いた入退場管理システムや、空港での航空手荷物管理システム [5] などが既存システムとして挙げられる。

図 1.1: 様々な RFID

RFID タグを製品につけることで、Supply Chain Management(SCM) におけるコストは、流通過程での盗難防止や在庫管理の効率化が可能となり、大幅な削減が期待できる。また、病院

では薬品や患者を RFID で管理することにより、薬品投与ミスや患者とり違いなどの医療事故を未然に防ぐことができる。我々一般消費者も、買物をはじめ様々な支払場面において、RFID で個人を特定することによりオンライン課金を利用して、現金支払い不要な生活が可能となる。このように、RFID を用いた新しいシステムに対する要求は大きい。

これらの要求にともない、製品に一意な ID を割り当て、製品情報をネットワーク上に展開し、管理する個体認識のインフラストラクチャ構築を主目的として、Auto-ID システムが EPC Global[6]、Auto-ID Laboratory(旧 Auto-ID Center)[7] で研究されている。

Auto-ID システムは、もともと現在多くの製品に付いているバーコードの次世代を担う製品識別技術の開発を主目的として始まった。製品の製造段階で、一意性が保証されている Electronic Product Code(EPC)[8] が割り振られた RFID タグを製品につけ (Source Tagging)、ネットワーク上に置かれた EPC Information Service(EPCIS)[9] と呼ばれる製品情報蓄積サーバに情報を蓄積する。製品の流通過程で、製品情報を EPCIS に蓄積していくことでトレーサビリティを実現する。製品情報を EPCIS に登録する時だけでなく、EPCIS に蓄積された情報を閲覧する時にも、ある製品の情報はどこに登録すべきなのか、またどこに蓄積されているのか、という情報が必要となる。すなわち、EPC を検索キーとして、EPCIS のネットワーク上での位置を特定する名前解決システムが必要となる。Auto-ID システムでは、これを Object Name Service(ONS)[10] として定義している。

現在の ONS は、インターネットの名前解決機構である Domain Name System(DNS)[11] を基にしている。DNS は、名前解決のために、検索キーが構造化されていることを前提に ID 空間を分割し、委譲することで検索ツリーを組んで検索を行っている。しかし、Auto-ID システムの名前解決を DNS で行った場合、検索の最初に問い合わせが行われる Root サーバへの規模性が達成されない懸念がある。

1.2 目的

本研究では、現在インターネットの名前解決機構である DNS を基にして構成されている、Auto-ID システムの名前解決機構である ONS について考えられる問題を整理し、既存の ONS とは異なる名前解決機構を実現することを目的とする。

1.3 本論文の構成

本論文は 7 章から構成される。

第 2章では、Auto-ID システムにおける名前解決機構について、現在の問題点を整理する。第 3章で、その問題を解決するためのアプローチについて述べる。第 4章では、そのモデルにもとづいて設計を行う。第 5章で本機構の具体的な実装について述べる。第 6章で、実装された本機構の評価を行なう。第 7章において、まとめと今後の課題を挙げ、本論文の結論とする。

第2章　現在の Auto-ID システムにおける名前解決の問題点

本章では、Auto-ID のシステム概要と前提を述べた後、既存システムの問題点を整理し、その問題を解決するための機能要件を考える。

2.1　Auto-ID のシステム概要

Auto-ID システムでは、EPC と呼ばれる ID が割り振られた RFID タグを、製品の製造時に貼り付ける。そして、その EPC タグを製品の流通経路上でリーダを用いて検知し、その EPC に対応した情報サービスサーバに製品データを蓄積していく。

製品データを蓄積したサーバに対して、様々なアプリケーションがアクセスし、製品情報を取得・利用することが想定されている。つまり、EPC を利用したアプリケーションの共通インフラとして、Auto-ID システムが存在している。

図 2.1に、Auto-ID システムの描く一般的な利用モデルを示す。

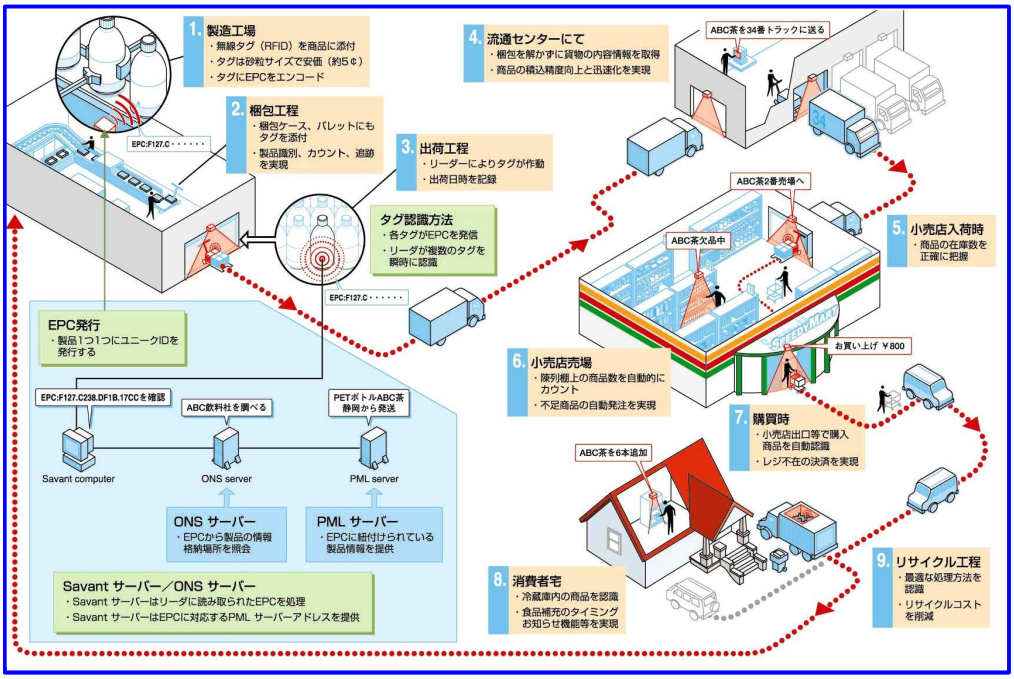

図 2.1: Auto-ID の利用シーン (「Auto-ID Center 資料」より)

次に、現在 Auto-ID Center で提案されているシステム概要を図 2.2に示す。Auto ID システムは製品の近傍に展開する **EPC**、EPC タグ、リーダ、およびインターネット上に展開する **Savant**[12]、**ONS**、**EPCIS** からなる。

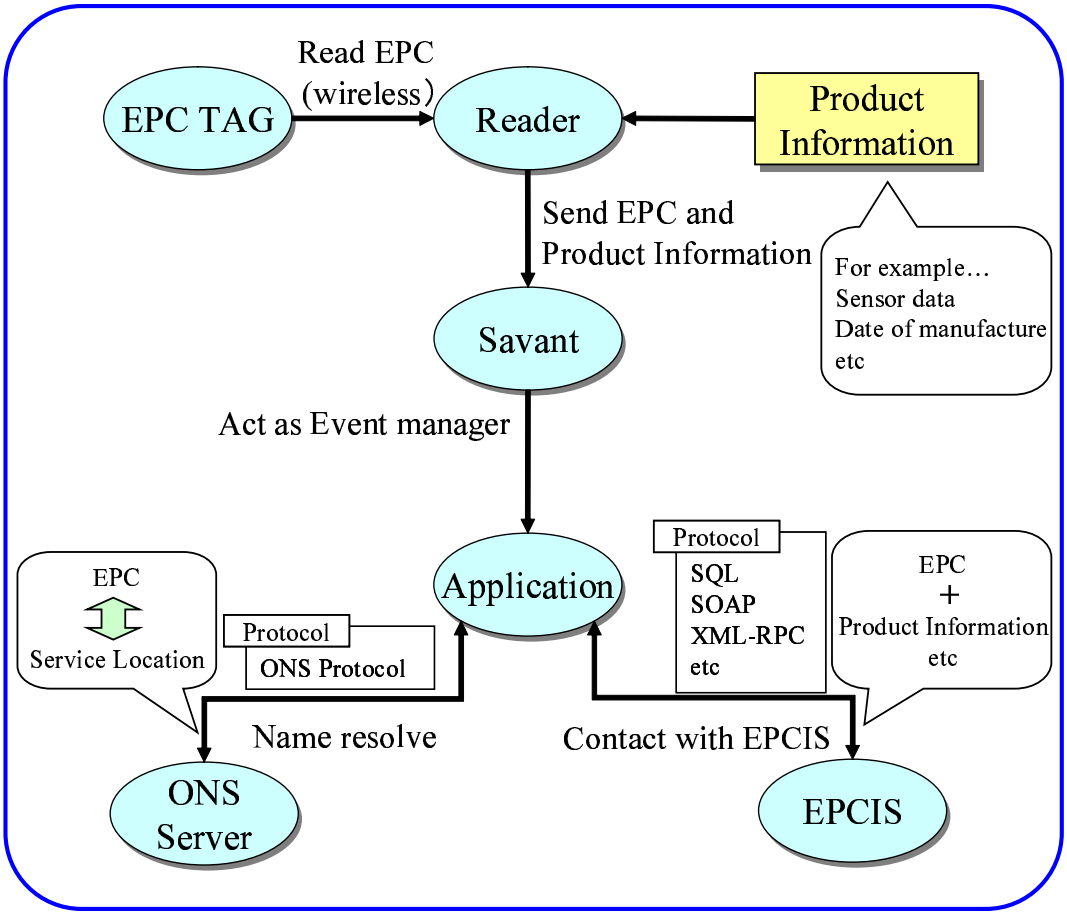

図 2.2: Auto-ID システムアーキテクチャ

以下に図中の用語について概要を示す。

- EPC
  EPC は、バーコード同様、製品の識別子として Auto-ID Center で提案された次世代の製品コード体系である。一意性が保障され、製品の製造段階で貼り付けされる (Source Tagging)。

  EPC の具体的なコード体系を図 2.3に示す。

  EPC は、8 ビットのバージョン ID、28 ビットの製造会社 ID、24 ビットの製品 ID、36 ビットの「シリアル ID、という構造化された 96 ビットの ID から構成される。この他にも、64 ビットや 256 ビットの EPC も提案されている。

- EPC タグ

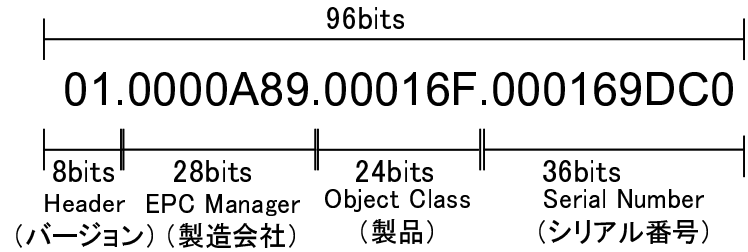

図 2.3: EPC コード体系図

EPC タグは、EPC が書き込まれたハードウェアである。タグには、電源を内部に持ち、タグ自身が電波を発生する比較的高価なアクティブタグと、電源を内部に持たず、リーダからの電波を受けることで電波を発生する比較的安価なパッシブタグの 2 種類がある。また、Auto-ID システムは、タグ自身が持つ情報は ID だけであることを前提としている。

- EPC リーダ
  EPC リーダは、リーダの検出範囲内に存在する EPC タグに格納された EPC を取得するデバイスである。EPC の読みとりには、無線を用いる。また、取得したタグと関連するセンサー群からデータを受ける機能も併せ持つ。

- Savant
  Savant は、リーダから受信したタグやセンサーの情報を処理するために設計されたミドルウェアである。Savant は、EPC リーダで検出された EPC をアプリケーションや、EPCIS、あるいは他のシステムにデータを送信する際に、検出されたタグデータのカウント、フィルタリング、集約を行うイベントマネージャとして用いられる。

- EPCIS
  EPCIS は、EPC に基づく情報を提供する情報サービスである。それらの属性情報は、Simple Object Access Protocol(SOAP)[13] や XML Remote Procedure Call(XML-RPC)[14]、SQL などのプロトコルを用いて格納される。

- ONS
  ONS は、EPC とその EPC に関連する EPCIS を対応付ける名前解決機構である。ONS のシステムは、EPC に基づくクエリを発行する ONS クライアントと、名前空間を分散して管理して、クエリに対する返答を行う ONS サーバ郡から構成される。ONS サーバ群は DNS サーバを利用して運用される仕様であり、個々の EPC に対応する EPCIS の位置を示す情報を格納する。ONS クライアントは、EPC を基にした DNS クエリを作成し、ONS サーバ郡に対して問い合わせを行う。ONS サーバは、該当する EPCIS の位置情報として、NAPTR リソースレコード (RR)[15] を ONS クライアントに対して返答する。

  なお、DNS クエリの作成手順を図 2.4に示す。

  現在の ONS の仕様では、

  1. ID を URI フォーマットに変換する
  2. シリアル番号の部分を削除して、ヘッダ、製造者、製品種類の順番を逆にする

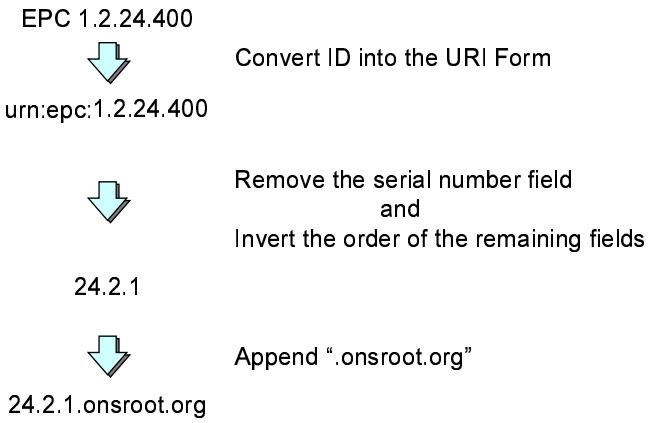

図 2.4: ONS クエリ生成プロセス

3. 文字列の最後に、”onsroot.org” を付ける

という手順を踏んでいる。

次に Auto-ID システムの動作手順を述べる。最初に、EPC リーダが製品に付けられた EPC タグから EPC を検知し、その EPC を Savant に送信する。この時、その製品に関連したセンサーデバイス情報も EPC と共にリーダから Savant に送信される。Savant は、イベントマネージャとして EPC イベントを処理し、アプリケーション、および EPCIS に EPC の情報を渡す。アプリケーションは、EPC をもとに ONS で名前解決を行い、EPCIS の位置情報を取得する。そして、EPCIS にアクセスし目的の EPC に対応した製品情報を取得する。

2.2 既存 ONS の問題点

本節では、既存 ONS についての問題点を整理する。

既存の Auto-ID システムにおける名前解決機構である ONS は、インターネットの名前解決機構である DNS を基にしている。

図 2.5を用いて動作概要について示す。詳細については 2.1節で述べたのでここでは割愛する。

図中の ONS Resolver は EPC をアプリケーションから受信すると、クエリを生成し ONS サーバ群に問い合わせを行う。ここで、問題となるのが Root サーバへのクエリ集中である。検索クエリが最初に問い合わせを行うのが Root サーバであり、世界中の製品に ID を割り振った場合に用いられる膨大な検索クエリ量は容易に想像ができる。つまり、Root サーバへの検索クエリに対する規模性が問題となる。

また、DNS を用いることで、データの登録は、名前空間の管理者でないと登録できない問題がある。これによって、ある製品の所有者が変わり、新しい所有者がその製品についた名前を管理したくてもできないことになる。

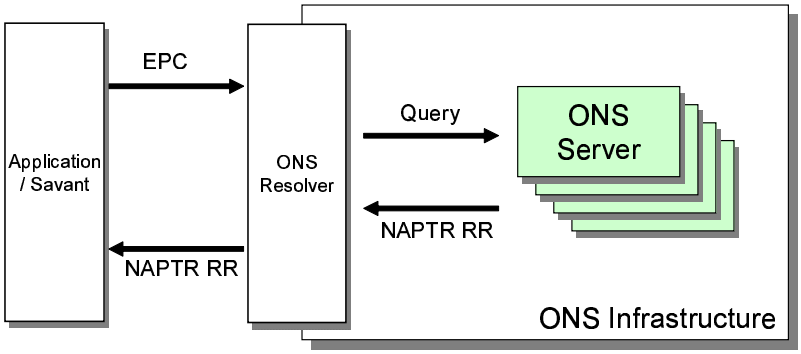

図 2.5: 既存の名前解決の流れ

2.3 新しい名前解決機構への要件

- 耐故障性の実現
  システムとして、Single Point of Failure を回避する必要がある。すなわち、サーバが故障しても、その事実をクライアントから隠蔽し、変わりないサービスを提供するためには、サーバの複製が存在し、その内部状態をなんらかの方法により共有し、交換可能な状態にあることが必要である。

- 負荷分散
  Auto-ID システムでは、現在 DNS が扱っているクエリ数よりもさらに多くのクエリが生成され、検索されることが予想される。これを一つのサーバで集中管理するのは非現実的であり、分散システムの形式をとることが必要である。

- 様々な名前空間への対応
  現在の ONS は、EPC に関して、2.1節の後半で述べた手順で検索クエリを生成している。しかし、今後 EPC は異なる ID 体系を持った名前空間が利用されることも十分考えられる。よって、それら多様な ID 体系を持った名前空間を解決できることが望まれる。

以上のことを考慮しながら新たな名前解決機構を設計する必要がある。

第3章　問題解決へのアプローチ

本章では、第2章で述べた問題点を踏まえ、要件を満たした名前解決機構を実現するためのアプローチを述べる。

3.1　分散ハッシュテーブル

分散ハッシュテーブル(Distributed Hash Table、以下DHT) とは、あるハッシュ空間を複数のノードで構成し、検索キーとデータをハッシュされたIDをもとに、各ノードに割り当てるシステムである。このDHTは、peer to peerシステムを基にした技術であり、規模性に優れ(scalable)、堅牢性(robust)を持った分散システムを構築する。

3.2　DHTを用いたONSに対する考察

本節では、DHTを用いてONSを考えた場合の利点・欠点について、DNSを基にした現在のONSと比較、検討を行う。図3.1に、その概要をまとめる。詳細については3.2.1節以降で述べる。

表 3.1: DHTを用いたONSの検討

			DNS	DHT
運用	管理権限	データ登録	△	○
		データ保持	○	△
その他	規模性	耐故障性	△	アルゴリズムに依存
		負荷分散	○	○
		局所性	○	アルゴリズムに依存
	その他	非構造的名前空間	×	○

3.2.1　管理権限

現在、DNSではデータ(リソースレコード、RR)を登録できるのは、その名前空間の管理者に限られている。SCMの場面を想定すると、製品の所有者が変わった場合に、だれがそのIDを管理するのか、という問題が考えられる。製品の所有者が、製造会社から個人消費者に移った場合に、その製品の名前管理は製造会社が引続き担当する場合もあれば、消費者が自分で管理したい場合も考えられる。

これに対し、DHT を用いると、データの登録は誰にでも行える。つまり、製品の所有者が変わってもデータの登録を行える。また、システム内のノードは全て自律的に動いているため、どのノードにデータの登録要求を行っても同じ結果となる。これにより、消費者が自分が購入した製品をローカルなデータベースで管理したい、といったニーズにも対応できる。ただし、誰にでもデータの登録ができるということは、悪意あるユーザが偽りのデータを登録することも可能となる。この問題は、Public Key Infrastructure(PKI)[16] による個人認証や、公開鍵暗号を用いたデータ改ざんの防止といった技術と連携させ、データ登録の信頼性を上げることで対処できる。

このように、データの登録制限という点で DHT は有利である。

他方で、データ保持の問題が考えられる。DNS では、ツリー構造を構成し、名前空間の保持者が明確である。DHT を用いるとデータは、ハッシュ値を基にシステム内のどこかに割り当てられることになる。つまり、たとえばコカコーラ社がある製品の名前空間を自社で管理したい場合でも、データはハッシュ値でコカコーラ社とは関係のないノードに割り当てられてしまう。(これについての考察が必要。。。)。

3.2.2 規模性

システムとしての規模性を比較するにあたり、耐故障性、負荷分散、局所性の三点について検討を行う。

耐故障性

DNS は、階層構造 (ツリー構造) によって、データの検索を行っている。よって、Root ネームサーバがしばしば攻撃対象となり、Single Point of Failure の性質を持っている。この問題を解決するために、各ネームサーバはセカンダリサーバを置く事で Single Point of Failure を回避している。また、DNS は Root に近いサーバの NS RR の TTL を長くもち、上位のサーバが落ちても下位のサーバは Cache を利用することでシステムとしての耐故障性を向上させている。

一方、DHT ではアルゴリズムにより耐故障性に優れたもの、そうでないものがある。DHT の耐故障性については第 3.1節にて詳しく述べる。

負荷分散

DNS では名前空間の管理者がツリー構造を設計することで、管理空間の委譲を行っている。管理空間を委譲していくことで、一人の管理者が管理する空間を小さくし、負荷を軽減している。

DHT では、ハッシュ関数による自律的な負荷分散を行っている。すなわち、ハッシュ空間をシステム内のノードが分割して自律的に管理することで負荷分散を実現している。

局所性

DNS の特徴として、検索クエリがある程度集中する場所があることが挙げられる。すなわち、局所性を持ったシステム構成になっている。

一方で、DHT で、クエリは localize されない。ハッシュID をもとに論理リンクを張り、転送ホップ数を削減し、データ検索を効率化している。

3.2.3 その他

対応可能な名前空間

DNS では名前空間を構造化することで、ID の一意性が保たれている。また、構造化することで経路の集約を行っている。

一方、DHT はハッシュ値でデータを持つので、名前空間の構造に左右されず名前解決を行える。これは DNS にはない特徴であり、DHT を用いた名前解決機構が既存 ONS より有利な点であると言える。

システム参加へのインセンティブ

DNS において、末端の方に位置するサーバは自分たちで名前空間の一部を管理するというインセンティブがありサービスを立ち上げる。しかし、DHT に限らず P2P システム一般に言える事として、システムに参加するためのインセンティブをどう示すかという問題がある。この問題は、今後一般ユーザが各々でサービスをあげるに十分値する利用モデルを考えていく必要がある。

3.3 DHT におけるアルゴリズム比較

分散ハッシュの検索アルゴリズムとして、Chord[17]、Tapestry[18]、CAN[19] などが挙げられる。これらのアルゴリズムでは、ノードの動的な追加や削除に対応し、データの検索を高速に行える。表 3.2に、各アルゴリズムの特徴比較を示す。

表 3.2: 検索アルゴリズムの比較

	Chord	Tapestry	CAN
データ構造	Skiplist	Plaxton 構造	d 次元構造
データ空間	1 次元	1 次元	d 次元
検索コスト	$O(log_2 N)$	$O(log_b N)$	$O(d * N^{1/d})$
各ノードが持つ経路数	$O(log_2 N)$	$bLog_b N$	$O(d)$
データ insert によるメッセージ数	$O(log^2 N)$	$O(log_b^2 N)$	$O(d * N^{1/d})$
負荷分散	○	○	○
アルゴリズムの簡易性	○	○	△
局所性	×	○	×
耐故障性	○	×	△

注. N : ID space, b : IDs of base b, d : 定数

次に、各アルゴリズムの特徴を以下にまとめる。

- Chord
  データをハッシュした値を基に、論理ネットワーク上にあるノードに、データを割り当てる。各ノードはN ビットの ID 空間で、N 個のルーティングテーブルを保持することで検索効率を高めている。

- Tapestry
  データをハッシュした値を基に、ハッシュ値の部分一致をにより検索を行う。

  Tapestry は Plaxton 構造 (Plaxton、Rajaman、Richa によって考案された分散データ構造) に基づいており、局所性を追求しているのが特徴である。

- CAN
  データから d 個のハッシュ値を求め、d 次元の座標空間に対応させる。この d 次元の ID 空間は複数のノードによって分散管理される。

  各ノードは自身の管理領域 (ゾーンと呼ぶ) に隣接する ID 領域を管理するノードへの情報を持ち、クエリルーティングを行う。

CAN アルゴリズムは、データに対して求めるハッシュ値の数によって検索コストなどが変化し、アルゴリズム的にも他の二つと比較して複雑である。また、Tapestry には、局所性があるので検索には有利である。しかし、システムに参加する際にルートノードが必要となり耐故障性の面で不安を抱える。一方、Chord は局所性を持たない。これにより、データを保持したノードが fail することでしばらくデータへの到達性が失われる。しかし、各ノードが自律的に振る舞うことでシステムとして耐故障性を実現している。また、Chord アルゴリズムでは、一つのノードは不特定多数のノードと通信するのではなく、最大 $logN$ 個のノードと通信を行えばよい。すなわち、各ノードが持つ経路数が少なくて済むという利点を持つ。

ゆえに、本研究では、検索アルゴリズムに Chord を用いる。

第4章　DHTを用いたONSの設計

本章では、DHT を用いた ONS のシステム設計を行う。

4.1　設計概要

本研究では、Auto-ID システムにおいて、UID と EPCIS を結びつけるための名前解決機構を構築する。なお、本章では、UID を EPCIS 以外の ID と定義する。

Auto-ID システムにおいて ONS が機能するためには、EPC をもとに検索クエリを生成し送信するインターフェースと、検索するための機能が必要である。すなわち、既存のシステムにおいて、**ONS リゾルバ**と各種検索システムがこれらにあたる。本研究では、この ONS リゾルバ、および第 3章で述べた DHT を用いた検索システムを名前解決機構の中に構築する。

次節では、ONS リゾルバ、および検索システムの設計を行う。

4.2　システムの設計

本節では本研究で構築する名前解決機構のシステム構成について述べる。

4.2.1　本研究におけるシステムモデル

本研究では、特に DNS と DHT、二つのネーミングシステムを利用することを想定する。また、本稿では、DHT を用いた名前解決機構を DHT ONS と呼ぶことにする。図 4.1に、本研究における名前解決機構のシステムモデルを示す。

このシステムモデルでは ONS リゾルバが API から ID を受信する。ONS リゾルバは、受け取った ID をもとに DNS、または DHT ONS のどちらかに検索クエリを送信する。クエリを受信したネーミングシステムは、DNS では NAPTR RR、DHT ONS では IP アドレスをそれぞれ検索結果としてアプリケーションに返信する。

なお、本節で述べる**データ**を、検索キーとなる ID とサービスロケーションを表す IP アドレスの組合せだと定義する。また、本システムでは、3.1節で述べたように、システムに参加するノード、およびデータは、US Secure Hash Algorithm 1(SHA1)[20] を用いて 160 ビットのハッシュID を持つことを前提とする。SHA-1 を用いた理由としては、EPC が 96 ビットの ID 空間なので、これを重複することなくハッシュID 空間にマッピングするために十分大きな空間を持つ必要があることが挙げられる。

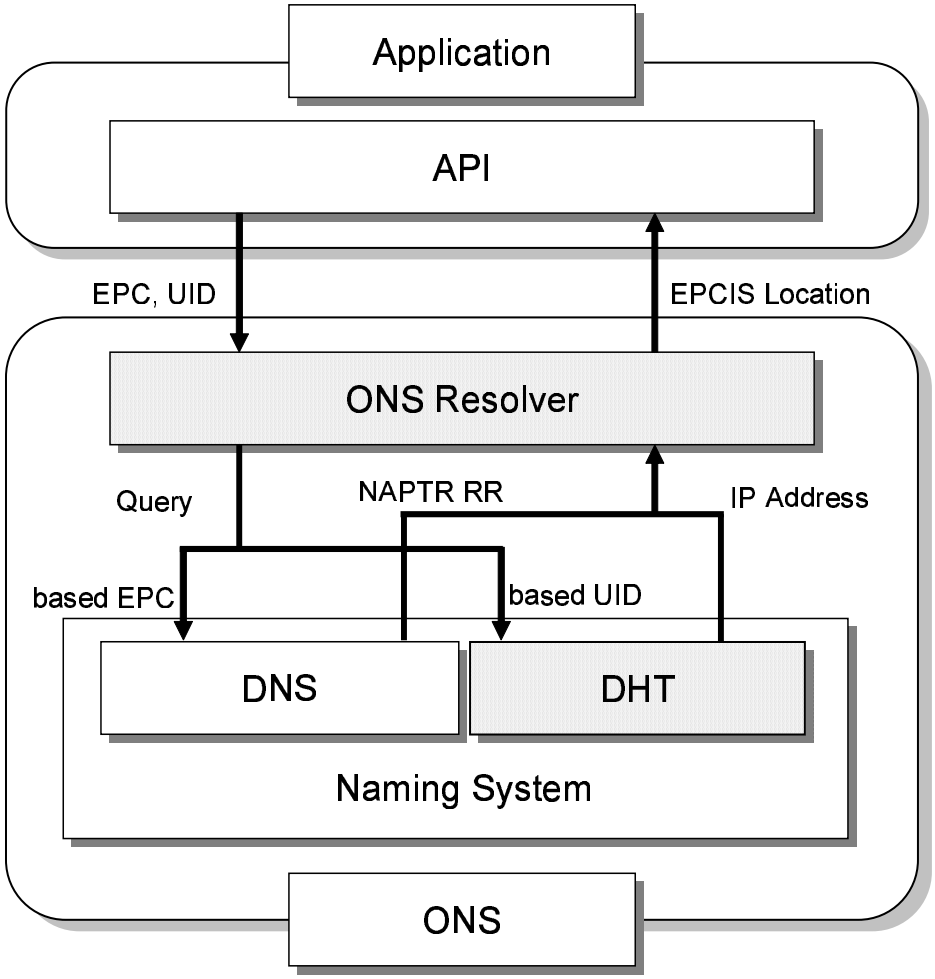

図 4.1: 提案するサービス解決の流れ

4.2.2 ONS リゾルバ

ONS リゾルバは、何らかのネーミングシステムに対して、検索キーをもとに検索クエリを生成し、ネーミングシステムに対してその検索クエリを送信し、取得したサービス (製品のメタ情報) の場所、すなわち”Service Location” を値として返すインターフェースである。現在、SAG(Software Action Group)[21] で議論されている ONS リゾルバの概要を示したのが図 4.2である。

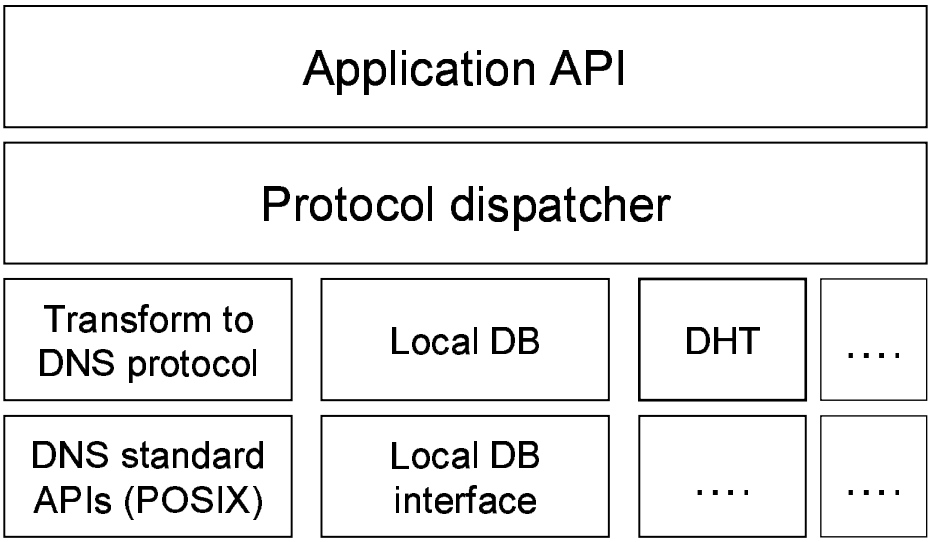

図 4.2: ONS リゾルバ

図 4.2に示したリゾルバは、EPC から EPCIS を検索するために検索クエリを生成し、DNS や、ローカルデータベース、DHT を用いた名前解決システムなど多様なネーミングシステムに対してクエリを送信することで EPCIS のロケーションを取得する。このリゾルバモデルは、特定のネーミングシステムに依存していないので、検索のニーズ、オプションに応じて問い合わせるネーミングシステムを変え、柔軟な検索を行えることが大きな特徴である。

本研究で構築する名前解決機構において、検索の柔軟性は重要である。なぜなら DNS で UID が解決できないように、今後本機構でも解決できない利用モデルが出現する可能性は否定できないからである。その時に、既存の名前解決機構を拡張する、あるいは新しい機構の作成するなど、システムの拡張を行うにあたり、Protocol dispatcher のようなモジュールを組み込んでおくことは拡張性という点で将来的に有利である。
リゾルバで、各ネームシステム毎のクエリを生成するためには、API が ID を送信する時に、ネームサービスを指定するフラグを付加することや、リゾルバに送信するデータフォーマットを統一すること、リゾルバ側にコンフィグファイルを置く方法などが具体的にあげられる。

本研究で想定するモデルでは、96 ビットの EPC と、EPC とは異なるコード体系をもつ UID という二種類の ID が混在した環境を想定している。そこで、本研究では、API から送信される ID のヘッダ部分と ID の長さに着目する。ID のヘッダとは、図 2.3の先頭 8 ビットで区切ら

れている、すなわち構造化されている場合は DNS を基にした既存のネーミングシステムに検索をかけ、そうでなければ、新しい名前解決機構に検索をかける。ID の長さでネーミングシステムを振り分ける場合は、ID の長さが 96 ビットであれば DNS を基にした既存の名前解決機構に検索をかけ、それ以外の場合は、新しい名前解決機構に検索クエリを送る。

4.3 DHT による名前解決

本節では、DHT を用いた名前解決機構の細部設計について述べる。

4.3.1 分散ハッシュテーブルの構築

本システムでは、システムに参加するノードは自身の IP アドレスを SHA-1 アルゴリズムを用いてハッシュすることでノード ID を取得する。そして、このノード ID をもとにシステムに参加し、分散ハッシュテーブルを構築する。

4.3.2 システムに参加するノードの信頼性

ハッシュテーブルの構築の際に問題となるのが、悪意あるノードの参加である。悪意のあるノードが参加すると、データを保持しつつ故意に接続性をなくしたり、嘘のデータを投げたりすることが可能となる。

よって、ノードがシステムに参加する時点で、ノードを認証する枠組が必要となる。

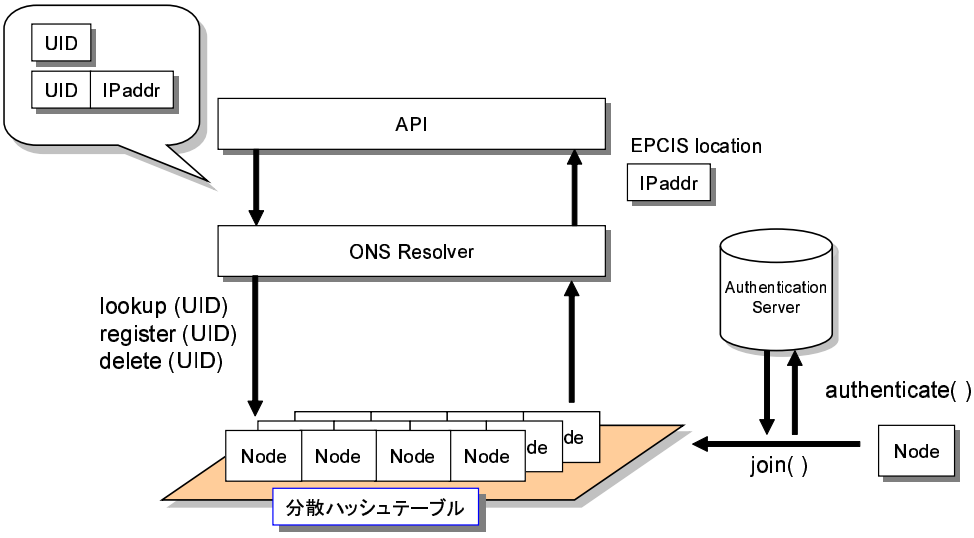

図 4.3: 名前解決機構でのモジュール図

図 4.3に本研究で実装する名前解決機構のアーキテクチャ図を示す。システムに新たに参加するノードは最初に認証サーバで認証を行う。これにより悪意のあるノードを排除する。ノードの認証には、公開鍵、秘密鍵を用いて行うこととした。

4.3.3 データ登録

データの登録には ID を用いる。ID をノードの参加時と同じように SHA-1 アルゴリズムでハッシュすることで HID を求め、システム内のいずれかのノードにデータを割り当てる。データを割り当てるノードに自身のデータがある場合は、データの更新を行う。

4.3.4 データ検索

本システムで行うデータ検索は、”ID をもとに、サービスロケーションを求める”、というものである。データの検索には、ID が検索キーとなる。本研究において、データの登録、および検索は 96 ビットの ID を全てハッシュするので、完全一致でしか検索ができないという欠点をもつ。

4.3.5 データの保持

データの保持の仕方には、リスト構造、ローカルデータベース、などいくつか考えられる。

Auto-ID システムでは、データの処理イベント、すなわち、登録、削除、検索が頻繁に起こると予想される。よって、データの登録、削除、検索をはやく、かつ効率的に行えるデータ構造が望ましい。

図 4.4に連結リストへのデータ挿入の様子を示した。連結リストでは、データに変化が起こっても、ポインタの指す先を変えればいいだけなので、動的に変化するデータ群を保持するのに有利である。また、データベースのようにプラットホームの環境に大きく依存しないので大規模なシステムになった時に、より多くのノードが参加できる可能性がある点で有利である。

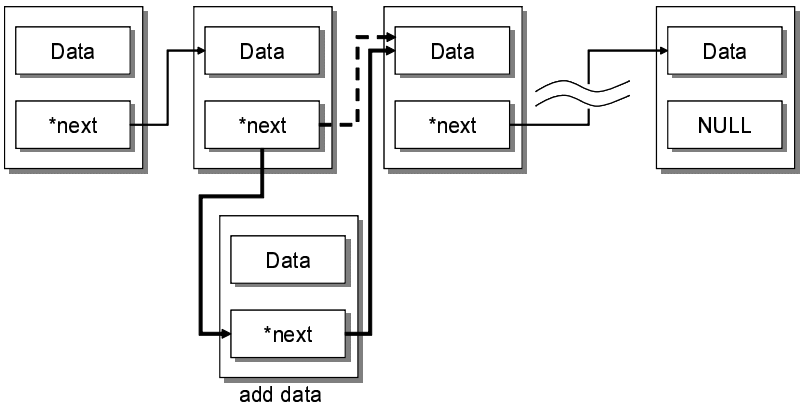

図 4.4: データ登録による連結リストの変化

4.3.6 データへの到達性

本システムでは、各ノードがデータをおおよそ均一に保持している。しかし、データを保持したノードが何らかの理由により到達性を失う可能性は十分に考えられ、それに伴ってデータ

への到達性が失われる。Chord のアルゴリズムでは、環上で時計周りに論理的に近いいくつかのノードがバックアップとしてデータを保持することでこれに対応する。

本研究では、今回この機能はサポートしないが、ID がリーダに読みとられ、Savant がイベントをマネージする段階でポーリングを行い、適宜データを送り直す、という手法でデータ到達性を回復する。

第5章　DHTを用いたONSの実装

本章では、本研究で行った実装について述べる。

5.1　実装環境

DHT を用いた ONS(以下、DHT ONS) の実装環境を、表 5.1に示す。

表 5.1: DHT ONS の実装環境

OS	FreeBSD 5.1-RELEASE
CPU	Pentium4　2.5GHz
メモリ	1024Kbyte
言語	C 言語

5.2　実装

本節では、本研究における実装について述べる。

5.2.1　システム内でのノード ID、および検索キー

本研究における実装では SHA-1 アルゴリズムを用いて 160 ビットの ID 空間上にノードなら IP アドレスをハッシュし、データなら検索キーとして用いる ID をハッシュし、計算して求められたハッシュ値をもとに配置した。なお、C 言語では 160 ビットの数値を扱える型がないため、本実装では unsigned char 型の配列を用いてハッシュされた ID(以下、HID) を扱う。

```
typedef unsigned char  uint8_t;    //uint8_t 型を unsigned char として定義

uint8_t Message_Digest[20];        //ハッシュされた ID を格納するための配列
```

図 5.1: SHA1 によるハッシュID

図 5.1に本実装における HID 変数の定義を示す。HID については、unsigned char 型を uint8_t

として定義し、HID を格納するための Message_Digest[20] という配列を宣言し、$8bit \times 20 = 160bit$ の ID を 4 ビットごとに区切り 40 文字の文字列として扱う。

5.2.2 ルーティングテーブルにおけるデータメンバ

ルーティングテーブルのメンバ構成を表 5.2に示す。ルーティングテーブルのメンバはシーケンス番号である index、検索する ID(検索キー)、検索キーに対する問い合わせ先ノードのノード ID、問い合わせ先を表す IP アドレスからなる。

表 5.2: ルーティングテーブルのデータメンバ

index 番号	検索キー	ノード ID	宛先 IP アドレス

本実装では、図 5.2に示した構造体を定義して用いた。index 番号に関しては、ルーティングテーブルを配列にすることで、配列の添え字を index として用いた。

```
struct routeTable{
   uint8_t hid[20];        //ハッシュされた検索キー
   uint8_t nodeid[20];     //ノード ID
   char *ipaddr;           //IP アドレス
};
```

図 5.2: ルーティングテーブルエントリ

5.2.3 ノード情報

システムに参加するノードは、自身のノード情報を図 5.3に示す構造体として保持する。具体的には、ノード ID、自身の IP アドレス、システム内で自身の前後に存在するノードのノード ID と IP アドレスを保持する。

```
struct mynode_info{
  uint8_t hid[20];
  char *ipaddr;
  struct routeTable successor;
  struct routeTable predecessor;
};
```

図 5.3: ノード情報

5.2.4 ルーティングメッセージ

システムに参加している各ノードは、ルーティング情報の更新を行うため常に他のノードメッセージを交換している。本節では、ノード間通信で用いられるルーティングメッセージフォーマットについて述べる。

本実装では、データ検索の問い合わせ、ルーティングテーブル更新のための問い合わせなど様々なノード間通信に対して一つのメッセージフォーマットを定義した。その統一フォーマットが図 5.4に示す構造体である。

```
struct message{
  char message[5];                  //ノード間でやりとりするメッセージ
  uint8_t hid[20];                  //ハッシュされた検索キー
  char *ipaddr;                     //IP アドレス
  struct routeTable predecessor;    //predecessor のデータを保持
  struct routeTable successor;      //successor のデータを保持
};
```

図 5.4: ルーティングメッセージ

様々な問い合わせを統一したフォーマットで扱うことで、メッセージを受けるインターフェースが実装しやすくなる。しかし、問い合わせの種類によっては余計なパケットを投げてしまうという欠点もある。

message 構造体のメンバである message 配列には、以下の表 5.3にあるリクエストを定義した。このメッセージを受信した各ノードは、この message フィールドをもとに successor の変更やリクエストなどの処理を行う。

表 5.3: message の種類

message の種類	message の値
successor の要求	0x01
successor の応答	0x02
predecessor の要求	0x03
predecessor の応答	0x04
successor の更新要求	0x06
predecessor の更新要求	0x07
データの登録	0x08
検索キーの検索要求	0x09

5.2.5　イベント処理

本プログラムの処理は以下の流れを経る。

1. ノードのイニシャライズ
2. 自身の successor を探し、システムに参加
3. message 処理

図 5.5に、本実装プログラムでの処理の流れを示す。

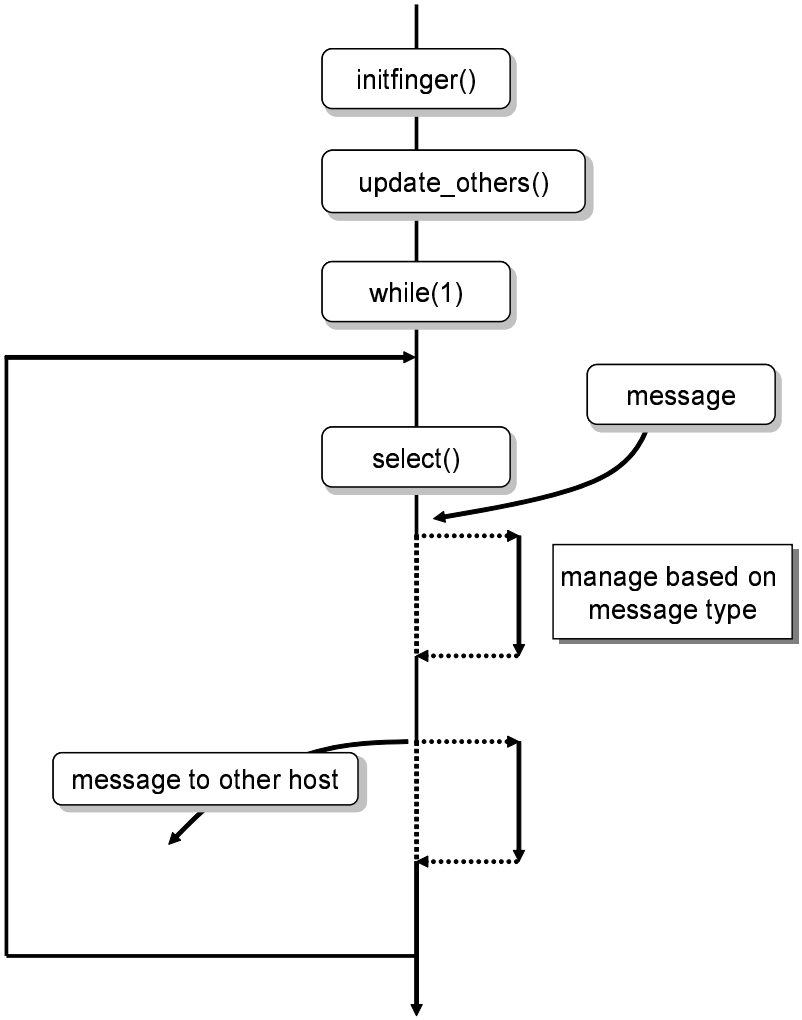

図 5.5: イベント処理のフローチャート

5.2.6　データ保持

本実装では、連結リスト構造を用いて実装を行った。具体的には、データの登録モジュール、データの削除モジュール、データの検索モジュールである。登録モジュール、検索モジュール

を利用することで、同じデータの二重登録を防止する。また、削除モジュールは、対象とするノードが削除すべき、または他のノードに移動しているべきデータがないかを調べ、該当するデータがあればそのデータを削除する。

```
struct list {
  struct list *next;
  char message[5];
  struct data_key data;
};

struct data_key{
  char *ipaddr;
  uint8_t hid[20];
};
```

図 5.6: 連結リストで用いるデータ構造体

図 5.7、5.8、5.9に、データマネージメントを行うプログラムを示す。

```
if((mp = lookup_entry(data)) != NULL) {
  return(-1);
}
if((new = (struct list *)malloc(sizeof(struct list))) == NULL) {
  perror("malloc()");
  return(-1);
}

new->data = data->data;

if(head == NULL) {
  head = new;
}
else {
  for(mp = head; mp->next != NULL; mp=mp->next) {
  }
  mp->next = new;
}
new->next = NULL;
```

図 5.7: データ登録モジュールの一部

```
if(mp == head) {
  head = mp->next;
}
else {
  for(p = head; p->next != NULL; p = p->next) {
    if(p->next == mp) {
      p->next = mp->next;
      break;
    }
  }
}
```

図 5.8: データ削除モジュールの一部

```
for(mp = head; mp != NULL; mp = mp->next) {
  …
  for(i = 0; i < 20; i++){
    if(data->data.hid[i] == mp->data.hid[i]){
      if(i == 19){
        return(mp);
      }
      continue;
    }else{
      break;
    }
  …
```

図 5.9: データ検索モジュールの一部

第6章　評価

本章では、本研究で実現された名前解決機構について評価し、その有意性を検証する。

6.1　定性的評価

本研究では DHT を用いた名前解決機構の設計と実装を行った。

評価の目的としては、様々ID と情報サーバの位置情報との対応データを登録し、検索する部分の動作を確認することとした。

評価項目としては、次の二点について行った。

- データがシステム内のノードに分散して登録できるか
- 様々な ID をもとにサーバの位置情報を検索できるか

評価環境としては、データの登録・検索クライアントを一台、データを保持するサーバを二台設置して実験を行った。

6.2　動作検証

本節では、実装を基に実際の動作検証を図 6.1のように行った。

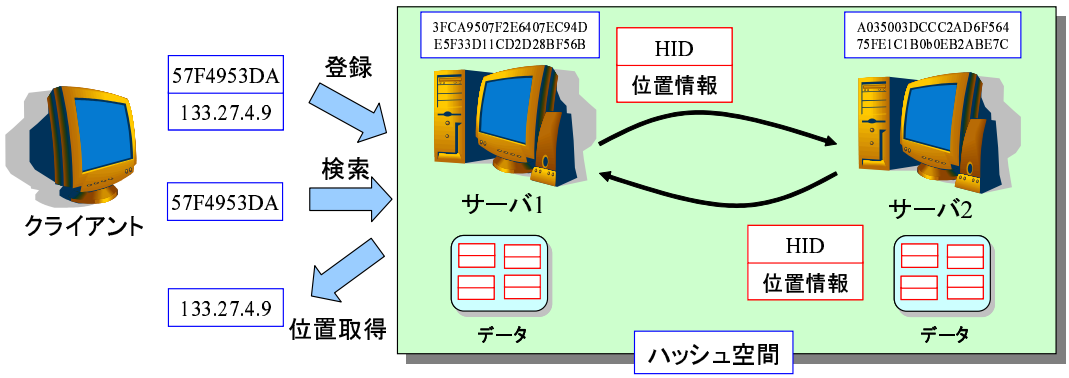

図 6.1: 動作実験

実験では、データを保持するサーバ二台とデータの登録や検索を行うクライアント一台で行った。

また、サーバ１を最初に起動し、サーバ２を次に起動したため、図 6.3の最初で引数にサーバ１のアドレスをとっている。データの登録、検索にともなう出力の一部を図 6.2、および図 6.3に示す。また、データの登録や検索時に参照するルーティングテーブルの一部を図 6.6に示す。

図 6.1におけるデータ登録の様子を図 6.4に、検索の様子を図 6.5に示す。

```
[Server1]> ./dht

returned HID is 3F7CA9507F266407EC94DE5F33D11CD2D28BF56B
IPaddress for HID : '203.178.141.41'

   ......

your_successor is 133.27.25.11
connected to 133.27.25.11
regist forwarded.

   ......

your_successor is 133.27.25.11
connected to 133.27.25.11
search forwarded.

   ......
```

図 6.2: サーバ 1 側でのメッセージ

6.3 評価結果

動作検証を行った結果、次の二点について動作確認がとれた。

- 一つのノードに対してデータ登録を行い、それらのデータが両方のノードに分散して登録できること
- 様々な ID を検索キーとした検索が行えること

以上より、本機構を用いて DNS とは異なる名前解決機構を構築する可能性を確認できた。

```
[Server2]> ./dht 203.178.141.41 30303

returned HID is A035003DCCC2AD6F1F56475FE1C1B0B0EB2ABE7C
IPaddress for HID : '133.27.25.11'

connected to 203.178.141.41

   ......

Data HID : 98291D0738C84A207B06A4536BDF074FFB7DB407
lookup_entry: No entries in the list.
register done.

   ......

Search HID : 98291D0738C84A207B06A4536BDF074FFB7DB407
Exist Data HID : 25C50D569FCFA4A1EA6F0384113E753EB8CD296C

Search HID : 98291D0738C84A207B06A4536BDF074FFB7DB407
Exist Data HID : 98291D0738C84A207B06A4536BDF074FFB7DB407

EPCIS IPaddr has be found!!

   ......
```

図 6.3: サーバ 2 側でのメッセージ

```
[Client]> ./regist 203.178.141.41 30303

Data 1: 1, '57F4953DA'
        98291D0738C84A207B06A4536BDF074FFB7DB407
```

図 6.4: データの登録画面

```
[Client]>  ./search 203.178.141.41 57F4953DA

Data 1: 1, '57F4953DA'
        98291D0738C84A207B06A4536BDF074FFB7DB407

EPCIS IPaddr has be found!!
IPaddress for requested HID is 133.27.4.9

[Client]>  ./search 133.27.25.11 57F4953DA

Data 1: 1, '57F4953DA'
        98291D0738C84A207B06A4536BDF074FFB7DB407

EPCIS IPaddr has be found!!
IPaddress for requested HID is 133.27.4.9
```

図 6.5: データの検索画面

```
Index 0 : 3F7CA9507F266407EC94DE5F33D11CD2D28BF56C,
          nodehid : A035003DCCC2AD6F1F56475FE1C1B0B0EB2ABE7C,
          <133.27.25.11>
Index 1 : 3F7CA9507F266407EC94DE5F33D11CD2D28BF56D,
          nodehid : A035003DCCC2AD6F1F56475FE1C1B0B0EB2ABE7C,
          <133.27.25.11>
Index 2 : 3F7CA9507F266407EC94DE5F33D11CD2D28BF56F,
          nodehid : A035003DCCC2AD6F1F56475FE1C1B0B0EB2ABE7C,
          <133.27.25.11>

   ..........

Index 157 : 5F7CA9507F266407EC94DE5F33D11CD2D28BF56B,
            nodehid : A035003DCCC2AD6F1F56475FE1C1B0B0EB2ABE7C,
            <133.27.25.11>
Index 158 : 7F7CA9507F266407EC94DE5F33D11CD2D28BF56B,
            nodehid : A035003DCCC2AD6F1F56475FE1C1B0B0EB2ABE7C,
            <133.27.25.11>
Index 159 : BF7CA9507F266407EC94DE5F33D11CD2D28BF56B,
            nodehid : 3F7CA9507F266407EC94DE5F33D11CD2D28BF56B,
            <203.178.141.41>
```

図 6.6: Server 1 側のルーティングテーブル

第7章　結論

本章では、本研究のまとめと今後の課題について述べ、本論の結論とする。

7.1　まとめ

本研究では、Auto-ID システムの名前解決機構である ONS について、必要要件を整理し、DNS とは異なった名前解決機構として、DHT を用いた名前解決機構の設計、実装を行った。本研究で行った実装をもとに、動作検証を行った結果、データを分散して登録し、システム内の任意のノードからの検索を実現できた。これにより、DHT の技術を用いて既存の ONS とは異なる名前解決機構の実現可能性を確認した。

7.2　今後の課題

今後の課題として、以下のような問題に取り組む必要がある。

- システムのデーモン化

  本研究で実装した名前解決機構はデーモン化されていない。しかし、名前解決システムはもともとバックグラウンドで動くサービスなので、デーモン化する必要がある。

- システムに参加するノードの信頼性

  DHT は元々Peer to Peer を基に派生した技術である。よって、ノードがシステムに参加する際に、なんらかの形でノードの認証を行うことにより、悪意のあるノードを排除する必要がある。これにより、システムとしての信頼性を向上させる。

謝辞

本論文の執筆にあたりまして、ご指導いただきました慶應義塾大学環境情報学部教授 村井純博士、並びに同学部教授 徳田英幸博士、同学部助教授 中村修博士、同学部助教授 楠本博之博士、同学部専任講師 南政樹氏に感謝いたします。

研究の要所要所において、大切な助言をくださった慶應義塾大学政策・メディア研究科博士過程 川喜田佑介氏、同大学政策・メディア研究科専任講師 羽田久一博士、IIJ 技術研究所 宇夫陽次朗博士、株式会社東芝 研究開発センター 土井裕介氏に感謝いたします。

また、研究室に入って以来、研究だけに限らず様々なところで面倒をみていただいた慶應義塾大学政策・メディア研究科修士課程 三屋光史朗氏、日野哲志氏、渡里雅史氏、小柴晋氏に感謝いたします。

私の研究に関して多大な時間を割いていただいた Spears WG の皆様、および Auto-ID Lab の皆様に感謝します。

慶應義塾大学 徳田・村井・楠本・中村・南合同研究室の皆様、特に nacm 研究グループと、SING 研究グループの皆様に感謝いたします。

参考文献

[1] Felica. http://www.sony.co.jp/Products/felica/.

[2] Suica. http://www.jreast.co.jp/suica/.

[3] 長崎スマートカード. http://www.shimatetsu.co.jp/bus/d-index.html.

[4] Edy. http://www.edy.jp/.

[5] e-エアポート. http://www.e-airport.jp/.

[6] EPC Global, 2003. http://www.epcglobalinc.org.

[7] Auto-ID Lab, 2003. http://www.autoidlabs.org.

[8] David L.Brock. "the electronic product code(epc) a naming scheme for physical objects", Jan 1999.

[9] Christian Floerkemeier, Dipan Anarkat, Ted Osinski, and Mark Harrison. "pml core specification 1.0", Sep 2003.

[10] Michael Mealling. "auto-id object name service (ons) specification version1.0", Sep 2003.

[11] P. Mockapetris. Domain names - implementation and specification, Nov 1987.

[12] Sean Clark, Ken Traub, Dipan Anarkat, and Ted Osinski. "auto-id savant specification", Sep 2003.

[13] Simple object access protocol. http://www.w3.org/TR/SOAP.

[14] Xml-rpc. http://www.xmlrpc.com/.

[15] M. Mealling and R. Daniel. *The Naming Authority Pointer (NAPTR) DNS Resource Record*, September 2000. RFC 2915.

[16] R. Housley, W. Polk, W. Ford, and D. Solo. *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*, April 2002. RFC 3280.

[17] Ion Stoica, Robert Morris, David Liben-Nowell, David R. Karger, M. Frans Kaashoek, Frank Dabek, and Hari Balakrishnan. "a scalable peer-to-peer lookup protocol for internet applications".

[18] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph. "tapestry: An infrastructure for fault-tolerant wide-area location and routing", Apr 2001.

[19] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. "a scalable content-addressable network", 2001.

[20] D. Eastlake 3rd and P. Jones. *US Secure Hash Algorithm 1 (SHA1)*, September 2001. RFC 3174.

[21] Software Action Group. http://develop.autoidcenter.org.

[22] P. Faltstrom. *E.164 number and DNS*, September 2000. RFC 2916.

付 録A　The Chord Protocol

A.1　アルゴリズム

ここでは Chord のアルゴリズムについて説明する。

Chord のアルゴリズムは、n ビットのフラットな ID 空間を保持する。この空間内で、仮想的な環を組み、その中に様々なデータやノードを、ハッシュした値をもとに割り当てていく。Chord では、データの検索や、ネットワークへの join/leave に特定のサーバを必要とせず、各ノードが完全に分散して処理を行う。

図 A.1に n を 3 として Chord アルゴリズムの概観図を示す。

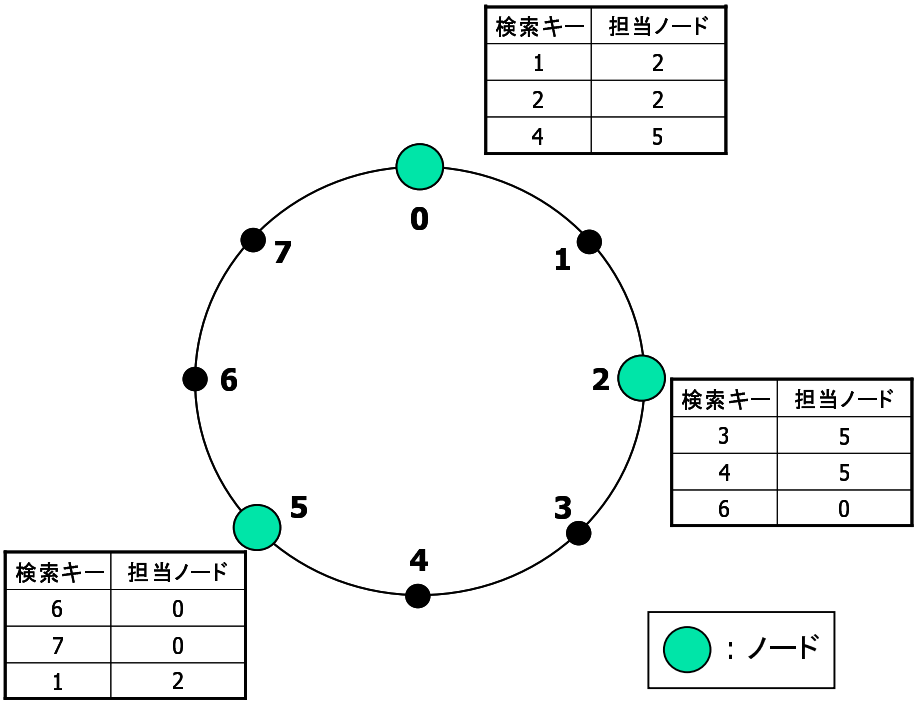

図 A.1: Chord アルゴリズム

Chord で組まれる仮想的な環に参加するノードは、**ルーティングテーブル**を保持する。これは、ある検索キーに対する検索で、どのノードに対して検索をしにいくか、ということを知るためのデータテーブルである。具体的には、ある n ビットの ID 空間の場合、n 個のデータエントリを保持し、自身の ID を s とすると、それぞれのエントリは $s + 2^{n-1} mod 2^n (1 \leq k \leq n)$ に対する担当ノードへのルーティング情報を保持する。

表 A.1にノード ID が 37104 のノードが保持するルーティングテーブルの例を示す。

表 A.1: ルーティングテーブル例

index	検索キー	ノード ID	ノードの IP アドレス
1	37105	37110	203.178.141.41
2	37106	37110	203.178.141.41
3	37108	37110	203.178.141.41
4	37112	37120	133.27.24.48
⋮	⋮	⋮	⋮

データの割り当てに関して、あるノード A とノード B(A <B) の間にある検索キーは、時計回りに直近に存在するノード、すなわちノード B に対して割り当てられる。このとき、ノード B をその検索キーの**担当ノード**と呼ぶ。また、各ノードは自身から見て、直前にあるノード (”**predecessor**”と呼ぶ。) と直後に位置するノード (”**successor**”と呼ぶ。) を常に正しくメンテナンスし、把握することでシステムとしての整合性を保っている。

Chord の環に参加に、新しいノードが参加する場合、そのノードは既に環に参加している既存のノードが指定され、最初にそのノードに自身の successor を問い合わせ、該当するノードに対して、predecessor を自身に変更するよう変更要求を送信する。

自身が環に参加したことによるシステムの不整合を修正するために、新たに参加したノードは、$s - 2^{n-1} mod 2^n (1 \leq k \leq n)$ の検索キーの前のノード ID を持つノードに対して、n 番目の検索キーに対する担当ノードを自身に設定しなおすようリクエストを送信する。

ある検索キーの担当ノードを検索するには、自身のルーティングテーブルを用いて検索を行う。ルーティングテーブルの持つエントリの中にその検索キーが存在すればその担当ノードを返す。そうでない場合には、検索キーの一つ手前のノードに検索をフォワードする。これにより、効率のいい検索を実現している。図 A.2に具体的な検索イメージを一般的な検索と対比させて示す。

一般的な順次検索が図 A.2の左側である。それに対し、図 A.2の右側が Chord における検索である。これを見ると、N42 のノードが持っているデータを検索するのに順次検索では論理的に 5 ホップ、Chord の検索では論理的に 1 ホップで検索していることが分かる。

また、ノード数 N の Chord ネットワークの性能を具体的に示すと以下のようになる。

- N 番目のノードが join/leave する際に、$O(1/N)$ の検索キーとデータの移動が生じる。
- ネットワークの維持には、他の $O(logN)$ ノードの情報を持つ必要がある。
- $O(logN)$ のメッセージ数で参照が可能である。
- join/leave する際のメッセージ数は $O(log^2 N)$ である。

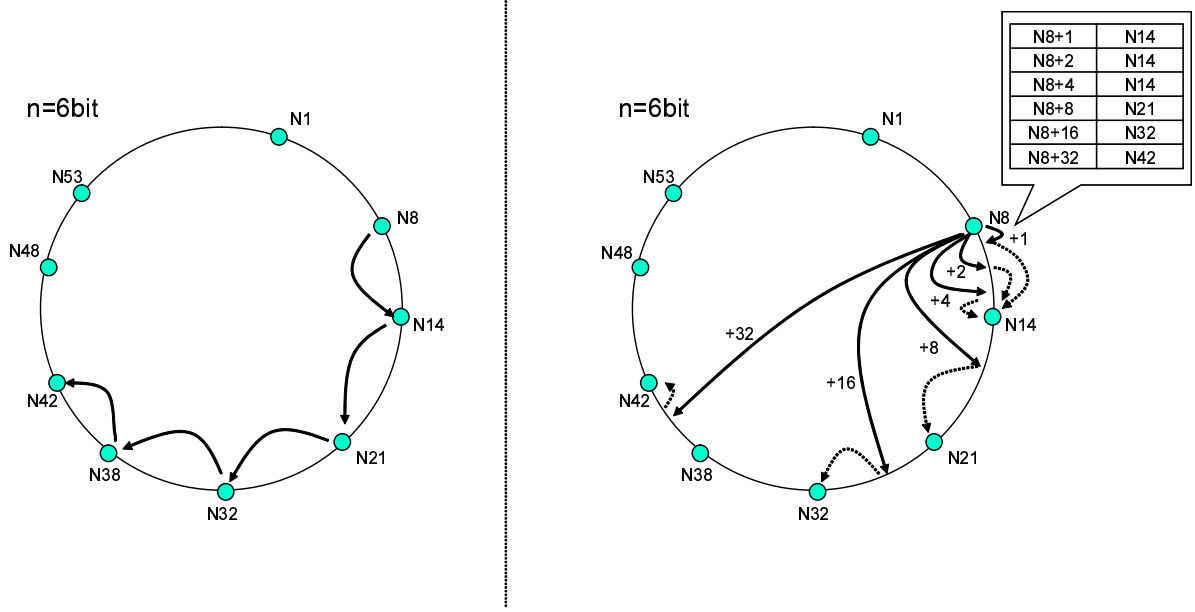

図 A.2: 単純検索と Chord 検索との比較

このように、Chord のアルゴリズムでは n = 160 ビットの ID 空間を保持しても、各ノードは 160 個のルーティングテーブルエントリを管理すればよく、スケーラブルなシステムを構築するのに向いている。

付 録B　Naming Authority Pointer

B.1　NAPTR RR

Auto-ID システムで、アプリケーションは、ONS を利用して EPC に対応する EPCIS の位置情報を得る。ONS は名前空間の管理に DNS を用いている。この ONS は、ある EPC に対応する EPCIS を Naming Authority Pointer(NAPTR) リソースレコード (RR)[15] として表現することで実現している。NAPTR RR は、アプリケーションのルールによって書き換えることができる RR である。この NAPTR が用いられている分野として、電話番号を用いてインターネット上の様々な通信サービスへの統一的なアクセスを可能とする ENUM[22] がある。NAPTR RR の各フィールドは以下の六つがあり、その利用例を表 B.1に示す。

ORDER
　レコードを処理する順番

PREFERENCE
　サービスの優先順位

FLAGS
　フラグ

SERVICES
　サービス

REGEXP
　URI に変換するための正規表現

REPLACEMENT
　置き換え用ドメイン名

表 B.1: NAPTR リソースレコード

データフィールド	データフィールドの中身	RR 例
ORDER	レコードを処理する順番	0
PREFERENCE	サービスの優先順位	0
FLAGS	フラグ	’u’
SERVICES	サービス	!∧$!http://company.com/ghi-bin/pmlservice!
REGEXP	URI に変換するための正規表現	‘.‘
REPLACEMENT	置き換え用ドメイン名	_http._tcp.company.com