2004 年度 卒業論文

IPv6 アドレスによる RFID システム利用方式

提出日： 2005 年 2 月 2 日

指導：後藤滋樹教授

早稲田大学　理工学部情報学科
学籍番号： 1G01P030-1

河野　真也

目次

図一覧

表一覧

第 1 章

序論

1.1 研究の背景

128bit という膨大なアドレス空間をもつ IPv6 の普及により、身の回りすべての物に IP アドレスを付与し、通信を行えるようにするユビキタスネットワークの実現が近づきつつある。ユビキタスネットワークにおいては IPv6 アドレスは単なるネットワークアドレスにとどまらず、物を識別するための固有 ID としても期待されている。

しかし、固有 ID として IPv6 アドレスを割り当てた場合、 IPv6 アドレスには所属するネットワーク ID も含まれているため、アドレスを割り当てられた物が、そのアドレスが本来所属するべきであるネットワークから移動してしまうと、アクセス対象へのパケットが正しくルーティングされなくなり、アクセスが不可能になってしまうといった問題が起こる。

また、通信を行うためには、ネットワークインターフェイスやプロトコルスタックが実装されていることが前提となるが、それらの実装はコストが高いため、もともとのコストが低い無線タグのような物に対しては、総コストに占める実装コストの割合が高くなってしまうため、あまり現実的ではない。

一方で、 EPC (Electronic Product Code) と呼ばれる固有 ID が格納された小型の無線タグを付与することによって、様々な物を識別・管理する RFID (Radio Frequency IDentification) システムがある。近年のタグのコスト低下といった要因もあって、バーコードに代わる個体識別技術として注目を集めている。 RFID タグは、今後ますます小型化、低コスト化が進んでいく見込みで、バーコードに代わり様々な物に付与されるようになることが予想される。

1.2 研究の目的

本論文では、将来的に RFID タグが低コストで様々な物に付与される可能性が高いことに着目し、ユビキタスネットワークを実現する方式として、タグに IPv6 アドレスを割り当てることに

より、IPv6 ネットワークを経由してタグとの通信を可能にする方式を提案する。また、その際にタグに割り当てる IPv6 アドレスとして、タグが属するネットワークのプレフィックスとタグ固有の ID である EPC をもとに生成された IPv6 アドレスを用いる方式を提案する。

1.3 本論文の構成

本論文は以下の章により構成される。

第 1章 序論

本研究の概要について述べる。

第 2章 RFID

RFID システムについて解説する。

第 3章 既存の研究

固有 ID として IPv6 アドレスを固定的に割り当てる方式について解説する。

第 4章 新方式の提案

IPv6 ネットワークを経由で RFID タグにアクセスする方式を提案し、考察する。

第 5章 実験

実際にプログラムを作成し、動作検証を行う。

第 6章 結論

本論文の結論を述べるとともに、本論文において残された問題点を今後の課題として提起する。

第 2 章

RFID

2.1 概要

RFID システムとは、 RFID タグ (無線 IC タグ) と呼ばれる無線チップを用いて人や物を識別・管理する仕組みである。 RFID システムには、同じ自動認識技術であるバーコードと比較して、

- 非接触で読み書き可能
- 汚れやほこりなどの影響を受けにくい
- 障害物があってもデータの交信が可能
- 複数のタグに同時にアクセス可能
- データ容量が大きい

といったメリットがあり、バーコードに代わる次世代の自動認識技術として大きく期待されている。

RFID システムは家畜管理や図書館の蔵書管理、空港のコンテナ管理、クリーニング店の衣類管理などに利用されているほか、近年ではコスト低下という要因もあって、試験的運用ではあるが、空港の手荷物管理や小売店の在庫管理、イベント会場等での入出場管理などに用いられるようになってきている。

2.2 利用周波数帯による分類

RFID システムでデータの交信に用いることが可能な周波数帯は、世界各国の電波の割り当てにより制限されている。そのため RFID タグは利用する周波数帯により、 135 kHz 以下、 13.56 MHz 帯、 UHF (860 MHz ～ 960 MHz) 帯、 2.45 GHz 帯の 4 種類に分類できる。それぞれの

周波数帯におけるタグの特徴を表 2.1 に示す。各国の電波行政により規制される電力が異なるため、同じ周波数帯を利用しているタグでも、国によって通信可能な距離が異なる。

初期は 135 kHz 以下のタグが主流であったが、現在は 13.56 MHz 帯のタグに移行しつつある。また、近年では小型化が可能な 2.45 GHz 帯や、より通信距離の長い UHF 帯のタグにユーザの興味が移りつつある。

表 2.1: 各周波数帯における RFID タグの特性

周波数	通信距離	備考	主な利用例
135 kHz 以下	2 m	最も歴史が長い	家畜管理、クリーニング
13.56 MHz 帯	1.5 m	現在の主流	図書館での蔵書管理、物流管理
UHF 帯	3 〜 7 m	通信距離が最長	空港でのコンテナ管理
2.45 GHz 帯	1 〜 2 m	タグのサイズが最小	駐車場での自動車の入出庫管理

2.3 電源による分類

RFID タグは電源を内蔵しているか、リーダから電源供給を受けるかにより 2 種類に分類することができる。前者はアクティブタグ、後者はパッシブタグと呼ばれる。それぞれのメリット・デメリットを以下に述べる。

2.3.1 アクティブタグ

リーダからの電源供給を受ける必要がないため、リーダと距離が離れていても通信を行うことができる。また、記憶領域を大きくすることが可能である。

しかし、電源を内蔵するため、小型化が難しい、高価である、電池の寿命を考慮しなければならないといったデメリットがある。

2.3.2 パッシブタグ

電源を内蔵しないため、小型・薄型化が容易で、ラベル型、コイン型、カード型、スティック型など様々な形状があり、用途に応じて選択できる。また、電池の寿命を気にする必要がないため、半永久的に利用することが可能である。近年 RFID に期待が高まっているのは、このパッシブタグが非常に安価で生産できる見込みが出てきたためである。

しかし、動作するためにはリーダから電源供給を受ける必要があるため、アクティブタグに比べて通信距離が短くなってしまう。

パッシブタグの動作原理

パッシブタグの動作原理は、タグが利用する周波数帯によって異なる。

13.56 MHz 以下の周波数帯を利用しているタグは、電磁誘導方式を使用している。リーダは送信するデータを変調し、電磁波としてアンテナコイルから送信する。タグ自身は電源を持っていないため、リーダは交信中、常に電磁波を出力しつづけなければならない。タグ側はその電磁波を受信し、回路を起動するための電力に変換すると同時に、データ成分を抜き出す。送信時と受信時でデータの変調方式を変えることにより、リーダ同士やタグ同士で誤った通信が行われるのを防止している。

タグが受け取るパワーは、リーダとの距離が離れるにしたがって小さくなる。得られるパワーがある下限値を下回ると通信できなくなる。この下限値のパワーを発生させることができる磁界強度を最小動作磁界強度という。タグはこの最小動作磁界強度が小さいほど通信距離が長くなる。最小動作磁界強度はタグの消費電力が少ないほど小さくて済む。タグの消費電力は IC チップの規模によって異なってくる。機能を増やして回路が複雑になるほど、消費電力が大きくなるため通信距離が短くなってしまう。

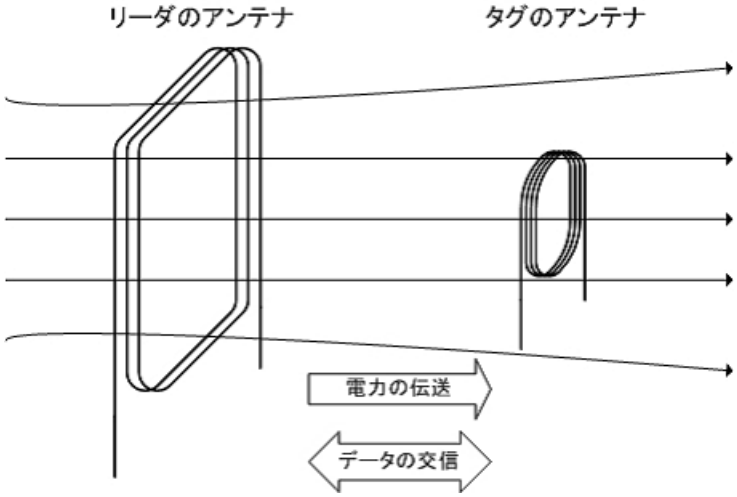

図 2.1: 電磁誘導方式

それに対して、 UHF 帯や 2.45 GHz 帯を利用しているタグは、電波エネルギーを電力パワーに変えるマイクロ波方式を使用している。タグは、リーダが発生する電波から、電流とデータを受け取る。多くの場合、マイクロ波方式のタグは UHF 帯から 2.45GHz 帯まで幅広く受信できるように作られている。その場合、余計な電波を拾う危険性が増えるが、それに対してはタグのアンテナの形状や長さを変え、受信する周波数を限定することによって対応する。つまり、同じ IC チップでもアンテナを変えることにより、異なる周波数に対応したタグを作ることができる。リーダからの信号を正しく受信するためには、タグのアンテナの設計が非常に重要である。アンテナ

の形状はループ状である電磁誘導方式とは異なり、ポール状である。

タグからリーダへの通信では、タグのアンテナの中で発生した電流が IC チップの中で反射する性質を利用する。 IC チップ内で反射した電流はアンテナに流れ込み、リーダに対する電磁波を発生させる。リーダはこの電磁波を読み取ってデータを受信する。

マイクロ波方式でも電磁誘導方式と同様に、タグに用いられている IC チップの消費電力により通信距離が変わる。 IC チップの消費電力が小さいほうが、リーダから得られるパワーが小さくても動作するので、通信距離は長くなる。

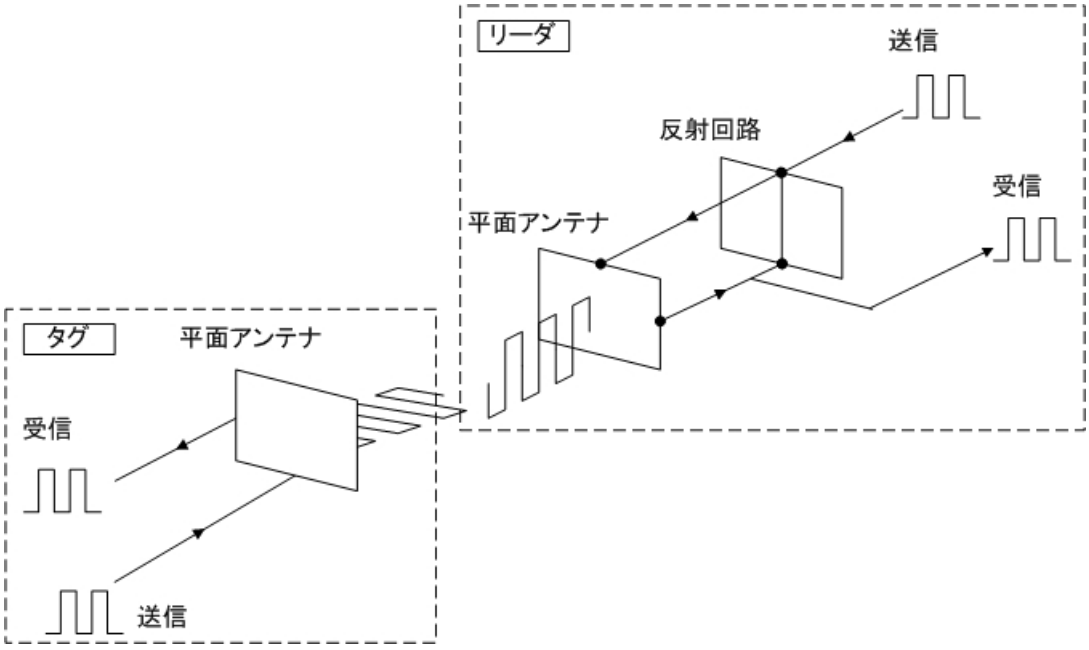

図 2.2: マイクロ波方式

2.4 EPC Network

EPC Network とは、 RFID 技術とネットワーク技術を組み合わせて作られた EPC (Electronic Product Code) を用いたシステムのサービスアーキテクチャである。 EPC が付加された物は様々な情報を EPC Network を通して蓄積・利用できるようになる。

2.4.1 EPCglobal

EPCglobal とは RFID システムの研究開発、実証実験、国際的なインフラの構築とそれらの標準化を目的とした国際的な非営利法人である。流通コードの国際的な標準化機関である国際 EAN (European Article Number) 協会と米国の流通コードセンターである Uniform Code Council (UCC) により、 2003 年 11 月に設立された。 EPCglobal の前身は 1999 年に米国ボストンの

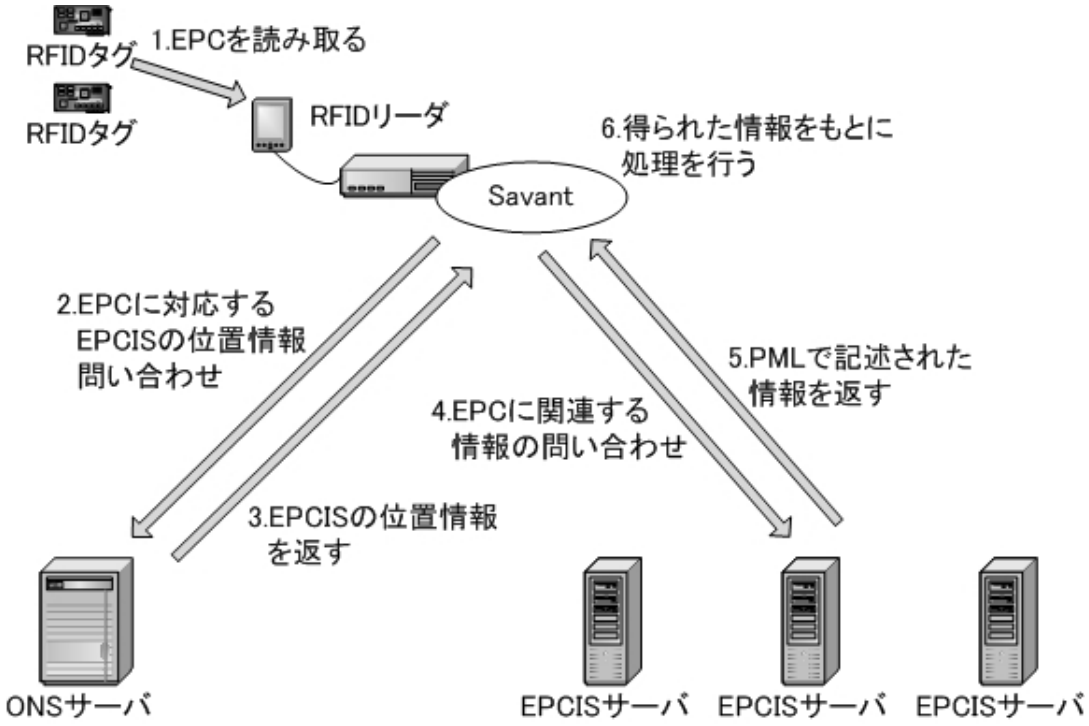

図 2.3: EPC Network の例

マサチューセッツ工科大学 (MIT) に設立された Auto-ID Center である。旧 Auto-ID Center の研究拠点は、現在では Auto-ID Labs. として研究開発および実証実験を行っている。日本では、EPCglobal の日本代表機関である財団法人流通システム開発センターが運用・普及を担い、慶応義塾大学 Auto-ID Labs. Japan が研究拠点として貢献している。

2.4.2 EPC (Electronic Product Code)

EPCglobal では、 RFID タグに EPC と呼ばれる固有の ID を記憶し、それを用いて物を識別・管理すること提案をしている。 EPC は ID の bit 長によって 64 bit、 96 bit、 256 bit に分けられ、さらに同じ bit 長でも数種類の規格が存在する。 EPC にはいずれの規格においても、先頭から順に以下の 4 種類のフィールドが含まれる。

- Version Number
  下部構造を規定するためのヘッダ。
- Domain Manager
  生産者や管理者を表わす番号。
- Object Class
  タイプを表わす番号。生産者や管理者によって割り当てられる。

- Serial Number
  個体識別番号。

それぞれの規格における各フィールド長を表 2.2 に示す。

表 2.2: EPC Version 別の各フィールド長

Version		Version Number	Domain Manager	Object Class	Serial Number
EPC-64	Type I	2 bit	21 bit	17 bit	24 bit
	Type II	2 bit	15 bit	13 bit	34 bit
	Type III	2 bit	26 bit	13 bit	23 bit
EPC-96	Type I	8 bit	28 bit	24 bit	36 bit
EPC-256	Type I	8 bit	32 bit	56 bit	192 bit
	Type II	8 bit	64 bit	56 bit	128 bit
	Type III	8 bit	128 bit	56 bit	64 bit

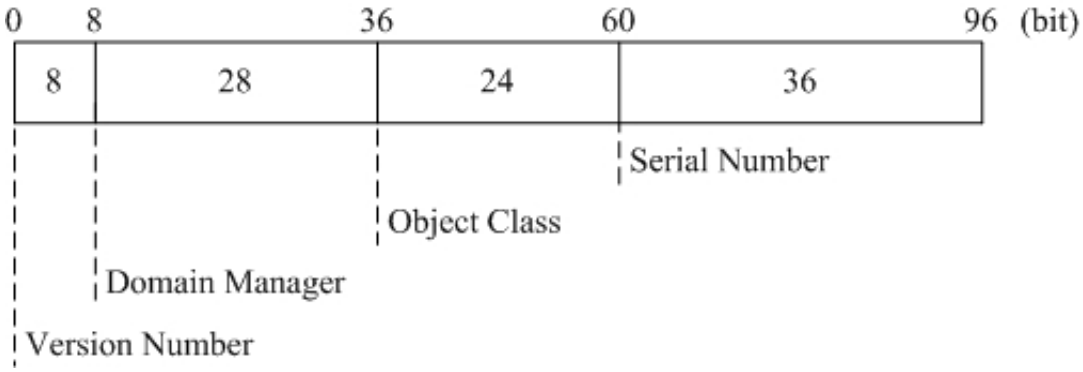

図 2.4: EPC-96 TYPE I

2.4.3 PML (Physical Markup Language)

EPC が表わしているのは固有の ID だけである。そのままでは個体の名前や製造年月日、賞味期限などといった情報を表わすことができない。そこで、 EPC が付与された物に関連する情報を表現するために Auto-ID Center によって提案された言語が PML である。 PML は XML (eXtensible Markup Language) に準拠した設計となっている。 PML を用いることで EPC に関連付けられた名前などの属性を記述することが可能となる。

2.4.4 EPCIS (EPC Infomation Service)

EPC に関連した情報を提供するためのフレームワーク。 EPC Network においては RFID タグに記憶されているのは EPC だけであることが前提となっている。そのため、 EPC に関連した情報を保持するサーバが必要である。 EPCIS サーバは後述の Savant からの要求に対し、 EPC に対応する PML を送信する。

2.4.5 ONS (Object Name Service)

EPC に関連した情報を得る際に EPCIS サーバの位置解決を行うサービス。現在提案されている ONS 1.0 は DNS をもとに設計されている。

2.4.6 Savant

RFID リーダが接続されたハードウェア上で動作するミドルウェア。リーダから検出された EPC をもとに、 EPCIS からその EPC に関連する情報を得て、それらに対する処理を行う。複数のリーダを制御することも可能である。

第 3 章

既存方式の解説および考察

個体に固有の ID を割り振り、その ID を用いてネットワーク経由でアクセス可能にする方法は、すでにいくつかの方式が検討されている。

固有 ID として IPv6 アドレスを固定的に割り振る場合について、 JPNIC (Japan Network Information Center) において考えられている方法の中から以下の 3 通りの実現方法を比較、考察する。

- /128 のアドレスをインターネットに広告する
- Mobile IPv6 を利用する
- VPN (Virtual Private Network) を利用する

また、比較のために、固有 ID として IPv6 アドレスを固定的に付与しない以下の 2 通りの実現方法についても比較、考察する。

- DDNS (Dynamic DNS) を利用する
- 独自の ID を利用する

3.1 /128 のアドレスをインターネットに広報する

端末に付与された /128 のアドレスを、そのままインターネット上に経路として広報する方法である。

/128 のアドレスがルーティング可能であるならば、もっとも単純かつ確実な方法である。しかし、 IPv6 は IPv4 におけるルーティングテーブルサイズの肥大化問題を踏まえたうえで、経路集約が可能になるようなアドレスの割り振りが行われている。具体的には、アドレスの割り振りのサイズを /32 単位とし、 BGP などの広域ルーティングシステム上に初期に割り振られた

/35 を除き、 /32 よりも長い経路広告が発生しないように配慮している。したがって、 /35 よりも長い経路広告がなされた場合、ルータによっては破棄されてしまう可能性がある。また、 /128 を広告することは膨大な IPv6 のアドレス空間を個々の端末単位で経路広告をすることになるため、ルータの性能上ほぼ不可能であるだろう。

現実的には、ネットワーク部は接続するネットワークから付与するか、 Mobile IPv6 などのルーティングを可能にする技術を利用することになるであろう。

利点

- 実装のコストが低い
  ルーティングが可能であるならば、 IPv6 による通信が可能であること以外の機能はいらない。

欠点

- ルーティングの実現が困難
  現在のルータの性能では、 /128 の IPv6 アドレスをルーティングすることは現実的にほぼ不可能。

3.2 Mobile IPv6 を利用する

Mobile IPv6 を利用して接続性を実現する方法である。

Mobile IP とは、端末が IP ネットワーク上を移動した場合においても、端末が移動する前の IP アドレスを用いた接続性を確保する技術である。 IP レイヤで実現されるため、アプリケーションレベルでは、端末が移動したことは隠蔽される。

アクセス対象となる MN (Mobile Node) には、 HoA (Home Address) のアドレス空間の中にある IPv6 アドレスの一つが固定的に割り振られている。 HA (Home Agent) は、ネットワーク上で HoA のアドレス空間に対してルーティングされている場所に設置される。アクセスする側は Mobile IPv6 を利用して対象となる物と通信を行い、 CN (Corresponding Node) として振舞う。

Mobile IPv6 を利用する場合の通信シーケンスを以下に示す。

1. 移動先に設置された MN は、移動先の IPv6 アドレス＝ CoA (Care of Address) を取得し、 HA に対して BU (Binding Update) メッセージを送信する。 BU メッセージは IPsec によって保護されなければならない。

2. MN からの BU メッセージを受信した HA は MN に対して BA (Binding Acknowledgement) を送信する。 BA メッセージは BU メッセージと同様に IPsec によって保護されなければならない。

3. CN は MN の移動先 IP アドレス (CoA) を知らないので、 HoA 宛にパケットを送信する。

4. HoA のアドレス空間上に設置された HA は、 HoA 宛のパケットを MN に代わって受信する。 HA は MN の移動先 IP アドレス (CoA) を知っているので、 CN が送信したパケットを CoA 宛のパケットでカプセル化し、 MN に送信する。

5. HA によってカプセル化された CN からのパケットを受け取った MN は、 CN に対して HA 経由で HoTI (Home Test Init) メッセージを送信する。 HA を経由させるために、 MN から HA の間は HoTI メッセージをカプセル化する。

6. 同時に MN は CN に対して直接 CoTI (Care-of Test Init) メッセージを送信する。

これより先の動作は CN が Mobile IPv6 に対応している場合とそうでない場合で異なる。 CN が Mobile IPv6 に対応していない場合、 CN は HoTI、 CoTI の受信に対し ICMP Parameter Problem メッセージを送信するか、もしくは何も送信しない。この場合、 MN は CN との通信を常に HA 経由で行う。 HA と MN の間は、 MN・ CN 間のパケットをカプセル化する。

CN が Mobile IPv6 に対応している場合の通信シーケンスを以下に示す。

7. HoTI と CoTI メッセージを受信した CN は HA 経由で HoT (Home Test) メッセージを送信する。

8. 同時に CN は MN 対して直接 CoT (Care-of Test) メッセージを送信する。

9. 正しい HoT メッセージと CoT メッセージを受信した MN は CN に対して BU メッセージを送信する。 HA に対して BU メッセージを送信する場合と異なり、 IPsec によるメッセージの保護は必要ない。

10. MN からの BU メッセージを受信した CN は MN に対して BA メッセージを送信する。 HA が BA メッセージを送信する場合と異なり、 IPsec によるメッセージの保護は必要ない。

11. HoTI・ CoTI、 HoT・ CoT のやり取りが終了すると、 MN と CN は HA を解さず直接通信を開始する。

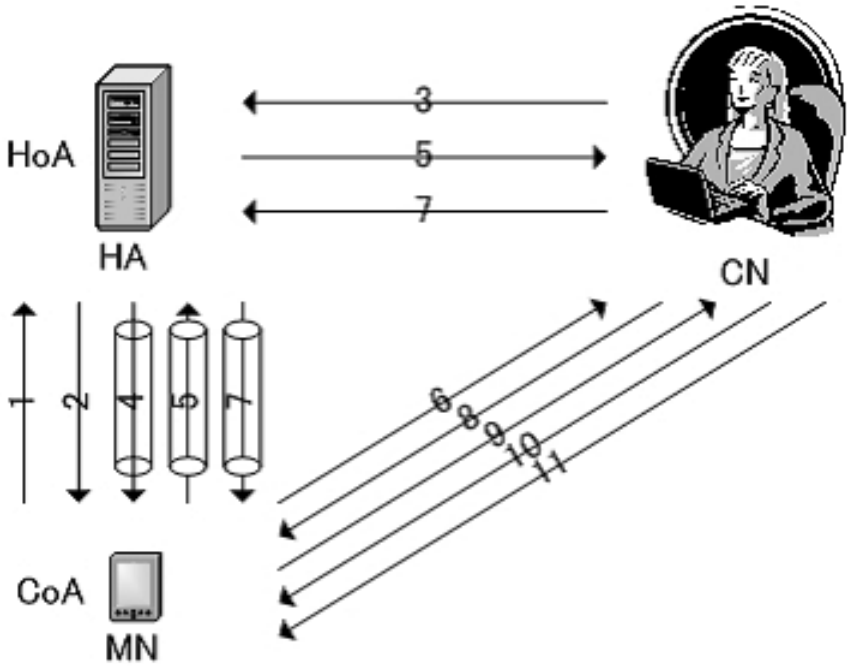

図 3.1: Mobile IPv6 利用時の通信シーケンス

利点

- 新たな独自プロトコルの規定が不要
  端末の移動は Mobile IPv6 により隠蔽される。
- セキュリティ機能が標準で提供される
  十分議論されているため、脆弱性のリスクが低い。

欠点

- Mobile IPv6 実装のコストが高い
  Mobile IPv6 を実装するには基盤のサイズ、パフォーマンスの両面でコストが高くなる。
- IPsec の処理負荷
  MN・ HA 間の BU メッセージ、 BA メッセージは IPsec の使用が必須。
- セキュリティ
  明示的に制限しない限り、 HoA を指定すれば誰でも MN にアクセス可能。
- HA のスケーラビリティ
  特に、 CN が Mobile IPv6 対応でない場合が課題。

3.3 VPN (Virtual Private Network) を利用する

VPN トンネリングを利用して接続性を実現する方法である。

アクセス対象となる端末には、アドレス集約点が持つアドレス空間の中にある IPv6 アドレスの一つ（ここでは Addr_equip とする）が固定的に付与される。また、アドレス集約点は、ネットワーク上でルーティングされている場所に設置されている。

VPN を利用する場合の通信シーケンスを以下に示す。

1. 移動先に設置された端末は、 RA (Router Advertisement) などの IPv6 標準的手法で、移動先で使用するアドレス（ここでは Addr_local とする）を取得する。アドレスを取得した端末は、 VPN トンネルを設定する何らかのプロトコルを用い、トンネル集約点に対して VPN トンネルの接続要求を送信する。
2. トンネル集約点はこの要求の是非を判断し、応答する。
3. 端末にアクセスする際は、固有アドレス Addr_equip 宛てにパケットを送信する。
4. トンネル集約点は、このパケットをカプセル化し、 Addr_local 宛てに送信する。

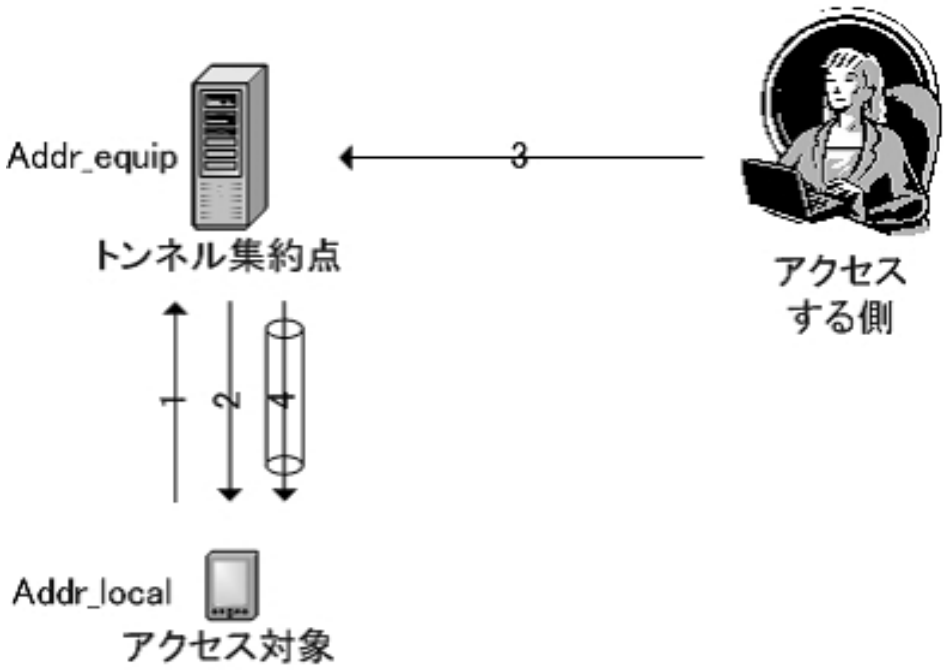

図 3.2: VPN 利用時の通信シーケンス

利点

- Mobile IPv6 と比較して、実装に必要な IC チップの規模が小さい
  低コストでの実装が可能である。
- 端末に付与した IPv6 アドレスに対し、 IPv4 ネットワークを経由して接続できる
  IPv6 ネットワーク・インフラの普及を待たずに、 IPv6 アドレスを固有 ID として使用できる。

欠点

- トンネルを動的に構成するための一般的なプロトコルがない
  通信のためのシーケンスを別途定義、実装する必要がある。
- セキュリティ
  明示的に制限しない限り、 Addr_equip を指定すれば誰でもアクセス可能。
- すべての通信がトンネル集約点を通過する

3.4 DDNS (Dynamic DNS) を利用する

DNS (Domain Name System) に対してリソースレコードを動的に登録可能な Dynamic DNS を用いる方法である。

端末に固有 ID を振ることなくアクセス可能とする方法。この方法では、 DDNS サーバを構築、運用することが前提となる。

DDNS を利用する場合の通信シーケンスを以下に示す。

1. 端末には、あらかじめ DDNS サーバの所在と、サーバに登録すべきネーム情報を事前に登録しておく。移動先に設置された端末は、 IPv6 の標準的手法で、移動先で使用するアドレスを取得し、取得した IPv6 アドレスとネーム情報を、事前に設定された DDNS に登録する。
2. 端末にアクセスする際は、まずネーム情報をもとに DDNS サーバから端末の IPv6 アドレスを得る。
3. アクセスする側は得られた IPv6 アドレスに対して直接通信を開始する。

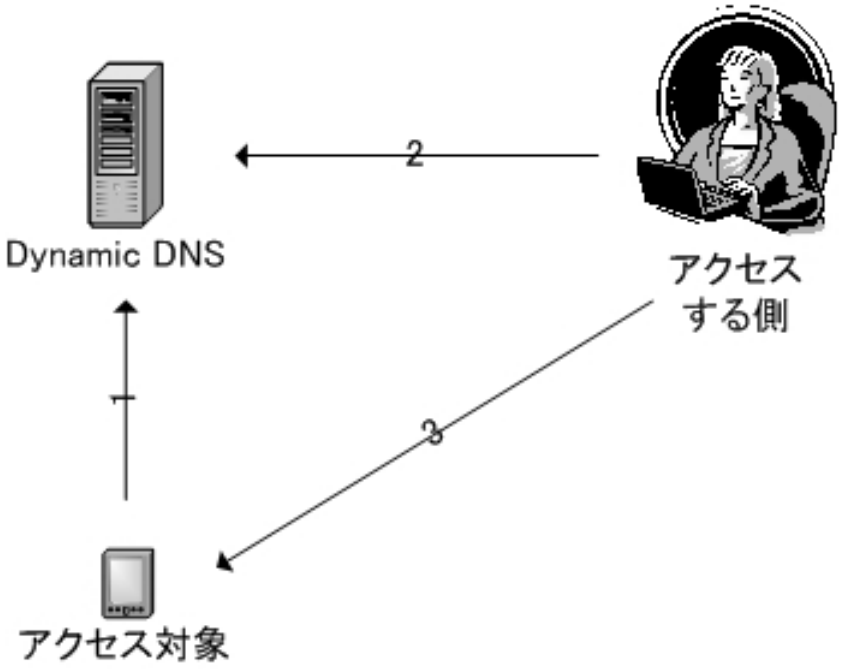

図 3.3: Dynamic DNS 利用時の通信シーケンス

利点

- 標準プロトコルの利用
  DDNS は IETF (Internet Engineering Task Force) において RFC (Request For Comments) として標準化され、インターネットサービスとしても広く用いられている技術であることから、安定動作や相互接続の容易さが期待できる。

欠点

- アドレスの代わりに名前管理が必要
  アドレスを ID として用いるという当初の目的を満たしていない。

- セキュリティ
  DNS はインターネットに公開されるサービスであることから、セキュリティやプライバシーが求められる場合には対応策の検討が必要。

3.5 独自の ID を利用する

独自の ID を付与し、その ID を利用して接続性を実現する方法である。

移動先に設置された端末は、まず IPv6 の標準的手法でアドレスを取得する。アドレスを取得した端末は、独自のプロトコルを用い、登録センタに取得したアドレスを登録する。端末にアクセスする際は、 ID をキーとして登録センタに端末のアドレスを問い合わせる。

端末は登録センタに登録されたアドレスが常に最新となるよう、定期的にアドレスを登録センタに登録する。また、アドレスの変更を検出した場合も、登録を行う。悪意を持つユーザにより、実際に取得したアドレスとは異なるアドレスが登録センタに登録されることを防ぐため、登録のシーケンスは何らかの方法により保護されなければならない。また、登録センタとアクセスを行う側が異なる場合、その間のセキュリティについても考慮する必要がある。

利点

- 実装のコストが低い
  必要最小限の機能のみを実装すればよく、低コストが期待できる。

欠点

- 新たな独自プロトコルの規定が必要
  プロトコルを標準化できなかった場合には、登録センタをプロトコル別に設置する必要がある。

- セキュリティについて独自の検討が必要
  十分議論がなされなければ、脆弱性の要因となる可能性がある。

第 4 章

提案方式

4.1 概要

システムの概要を図 4.1 に示す。また、本システムを利用した場合の通信シーケンスを以下に示す。

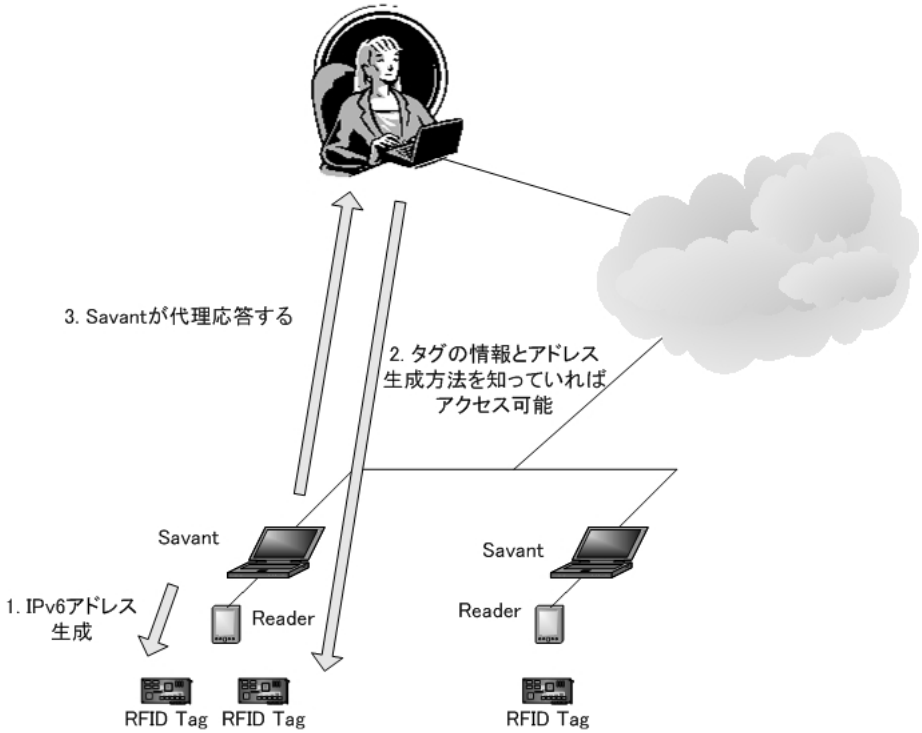

図 4.1: RFID を利用したシステム概要

1. RFID タグがリーダの検知範囲内に入ると、リーダが接続された Savant はタグに記憶されている情報をもとに何らかの方法で IPv6 アドレスを取得し、そのアドレスをタグに割り当てる。 IPv6 アドレスの取得方法は後述する。

2. アクセスする側は、タグに記憶されている情報と IPv6 アドレスの取得方法を知っていれば、その IPv6 アドレスに対して直接アクセスすることが可能である。

3. タグにはネットワークインターフェイスが実装されていないので、 Savant はタグへのアクセスに対してタグの代わりに応答し、タグに関する情報を返す。

4.2 IPv6 アドレス取得方法

RFID タグに IPv6 アドレスを割り当てるためには、タグの何らかの情報をもとに IPv6 アドレスを取得しなければならない。タグにあらかじめ IPv6 アドレスが記憶されているとすると、 3 章で述べたいずれかの方式を用いることにより、アクセス可能である。しかし、本研究では将来一般に普及する RFID タグを流用することにより安価にシステムを構築することを目的としている。 RFID タグに記憶される情報としては、現在のところ EPC と呼ばれる固有 ID のみになる見込みであるため、タグに IPv6 アドレスが記憶されることは期待できない。

したがって、 EPC をもとに IPv6 アドレスを取得しなければならない。しかし、 EPC は 96bit 長であるため、そのままでは IPv6 アドレスとして利用することができない。 3.1 節で述べた「/128 のアドレスを経路広告する方式」と同様、 /96 のアドレスをインターネットに経路広告することは現実的に不可能である。また、 RFID タグには EPC 以外の情報は記憶されていないから、 3.2 節や 3.3 節のように Mobile IPv6 や VPN を利用した方式も不可能である。

ルーティングを可能にするためには、 RFID タグが属するネットワークのプレフィックス (/64) に 64 bit 長に縮めた EPC をつける方法が考えられる。しかし、単純に EPC の一部分を抜き出して用いるという方法では、同一の製品などのもともと EPC が類似している物が複数存在した場合にアドレスが重複してしまう可能性が高い。

そこで本論文では、 IPv6 アドレスの上位 64 bit は RFID タグが所属するネットワークのプレフィックス、下位 64 bit は EPC をハッシュ関数で 64 bit 長に縮めたものを用いる方式を提案する。

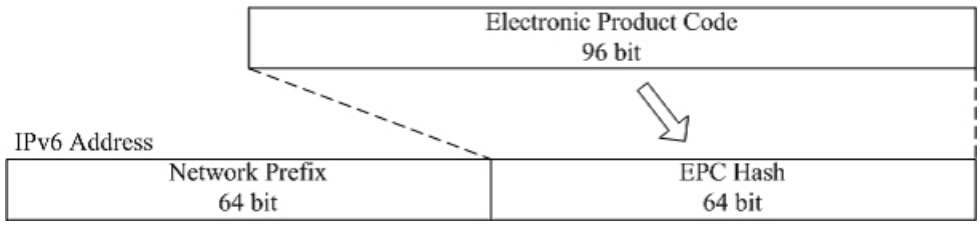

図 4.2: ハッシュ関数を用いた IPv6 アドレス取得方式

ハッシュ関数

ハッシュ関数とは、可変長のデータを固定長のデータに変換する関数のことで、以下の特徴を持つ。

- 入力データの長さが異なっても出力されるダイジェスト長は一定
- 入力データが少しでも異なればダイジェストは大きく異なる
- ダイジェストから入力データの算出は困難
- 同じダイジェストとなる異なった入力データを見つけるのは困難

提案方式では、96 bit の空間を持つ EPC を 64 bit 空間に写像して用いるため、取得したアドレスが重複してしまう可能性がある。ハッシュ関数が生成する出力データがランダムであると仮定すると、N 個の入力データがある場合に出力データが重複する確立 $P(N)$ は、1 から出力データが重複しない確立 $P'(N)$ を引くことにより求められる。

$$\begin{aligned} P(N) &= 1 - P'(N) \\ &= 1 - \left(\frac{2^{64}}{2^{64}} \times \frac{2^{64}-1}{2^{64}} \times \cdots \times \frac{2^{64}-(N-2)}{2^{64}} \times \frac{2^{64}-(N-1)}{2^{64}} \right) \\ &= 1 - \left(\frac{1}{2^{64\times N}} \frac{2^{64}!}{(2^{64}-N)!} \right) \end{aligned}$$

ここで、$P'(N)$ の値は N が十分に小さければ、ほぼ 1 である。したがって、現実的な利用の範囲内では出力データが重複する可能性は限りなく 0 に近いと考えられる。

4.3 考察

本方式の利点、問題点を以下に挙げる。

4.3.1 利点

低コストで実装可能

通常、IP による通信を行うためには、ネットワークインターフェースとプロトコルスタックが実装されていることが条件となる。また、3 章で示した方式のように IPv6 アドレスを固有 ID として用いる場合、Mobile IPv6 や VPN といった機能の実装も必要となってくる。

それに対して、提案方式では IP 通信に必要な機能は Savant のみに実装すればよく、アクセス対象となる物には、RFID タグのみを付与すればよい。そのため、低コストでの実装が可能である。

IPv6 アドレスが容易に推測できないことによる不正アクセスの回避

IPv6 アドレスを物固有の ID として利用する方式では、同種の製品間では ID が近いものになることから、 IPv6 アドレスを簡単に推測できてしまう。

一方、提案方式ではネットワークプレフィックスと、ハッシュ化した EPC を組み合わせて用いるため、ネットワークプレフィックスと EPC の値を知らない限り、生成された IPv6 アドレスを推測することは難しい。そのため、外部の人間からの ID 推測による不正アクセスを低減することが可能である。

また同様に IPv6 アドレスからもとの EPC を求めることは非常に困難であるため、利用アドレスからアクセス対象となっているタグを特定されることもない。

全ての EPC 規格にも対応可能

2.4.2 節で示したように、 EPC には現在 64 bit、 96 bit、 256 bit の 3 種類の長さの規格が存在する。

提案方式では、 EPC をハッシュ関数により 64 bit に変換してアドレス取得に利用する。ハッシュ関数は入力データの長さが異なっても出力されるダイジェスト長は一定という特性を持つため、入力される EPC がいずれの規格であっても、 Savant は動作をまったく変更することなく対応可能である。同様に、将来新たな規格が提案されても Savant を変更する必要はない。

4.3.2 問題点

製造者がアクセス不可能

3 章で解説した方法は、全て製造者側が自社の製品に対してアクセスすることに重点を置いて考案された方法である。そのため、固有 ID を知っている製造者は容易にアクセス可能である。

しかし、提案方式では固有 ID である EPC だけでなく、タグが属するネットワークのプレフィックスが分からなければアクセス不可能であるため、たとえ製造者であろうとタグに自由にアクセスすることはできない。製造者からのアクセスを可能とするためには、 EPC とネットワークプレフィックスを製造者側に登録するなどの作業が必要となる。

第 5 章

実験

5.1 実験の目的

4 章で提案した方法で通信が可能であることを、実際にプログラムを作成して検証する。実験に用いたプログラムはすべて Java を用いて記述している。ここで用いたプログラムのソースコードを付録 A に載せた。

5.2 実験の環境

実験に使用した機器を表 5.1 に、ネットワーク構成を図 5.1 に示す。 Savant PC 上では RFIDSavant、 Reader PC 上では RFIDReader、 Client PC 上では Client をそれぞれ実行する。 Reader と Savant を別のマシンにしているのは、実験に用いた PhidgetRFID Kit が Windows XP 上でしか動作しないためである。

5.3 プログラムの動作

5.3.1 RFIDReader, RFIDSavant

実験に用いた PhidgetRFID Kit の性能上、一度に読み取れるタグは一つのみなので、タグを読み取るとリーダの検知範囲内に入ってきたものとし、次に読み取られたときに検知範囲から出て行くものとする。また、 PhidgetRFID Kit のタグに記憶されている情報は EPC ではなく 40 bit の ID であるが、動作に変わりはないのでそのまま用いることとする。

タグが読み取られた際の動作を以下に示す。

1. Reader はタグが検出されると、タグから EPC を読み取り Savant へ送信する。
2. EPC を受信した Savant はハッシュ関数を用いて 64 bit 長のハッシュ値を求める。

表 5.1: 実験に利用した機器の仕様

Savant PC	CPU	Xeon 3.2 GHz
	Memory	1 GB
	OS	RedHat Linux 9 (Linux 2.4.21)
	Java 実行環境	J2SE SDK v5.0
Reader PC	CPU	Pentium III Mobile 800 MHz
	Memory	256 MB
	OS	Windows XP Professional SP 2
	Java 実行環境	J2SE SDK v1.4.0
Client PC	CPU	Pentium4 2 GHz
	Memory	1 GB
	OS	RedHat Linux 9 (Linux 2.4.28)
	Java 実行環境	J2SE SDK v5.0
RFID リーダ, タグ	製品名	PhidgetRFID Kit
	動作方式	パッシブ
	利用周波数	125 kHz

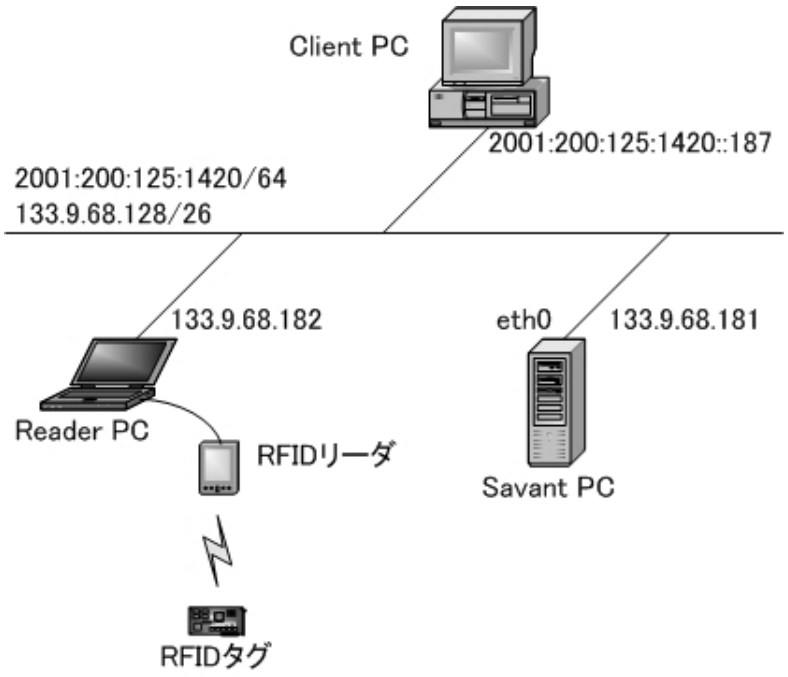

図 5.1: ネットワーク構成

3. Savant はあらかじめ登録されたネットワークプレフィックス (/64) と、 EPC のハッシュ値を用いて IPv6 アドレスを取得する。

4. 生成された IPv6 アドレスが Savant のネットワークインターフェースに登録されていなければ登録し、すでに登録されている場合は削除する。

5.3.2 Client, RFIDSavant

Client からタグへアクセスする際の動作を以下に示す。

1. Client はユーザから入力されたネットワークプレフィックスと EPC から、 Savant と同じ方法で IPv6 アドレスを取得する。

2. 取得した IPv6 アドレスに対して通信を開始する。

3. Client はコマンドを送信する。

4. コマンドを受信した Savant はコマンドに応じた処理を行い、結果を Client に返す。

なお、 Savant にはアクセスされたタグの EPC を返すコマンド (getEPC) のみ実装した。

5.4 実験の内容

5.4.1 実験 1

Reader にタグを読み取らせ、 Savant であらかじめ設定したネットワークプレフィックスと Reader が読み取った EPC から IPv6 アドレスが取得されることを確認する。

5.4.2 実験 2

Client で同様にネットワークプレフィックスと EPC から IPv6 アドレスを生成し、生成したアドレスに対して通信が行えることを確認する。

5.5 実験の結果

5.5.1 実験 1

まず、 Reader の動作を見る。なお、それぞれのプログラムは動作の詳細が確認できるようにデバッグモードで起動している。

```
> java RFIDReader
<< Debug Mode >>
Destination Address : 133.9.68.181
Destination Port    : 5100
Polling Interval    : 500 msec

<< Read a Tag >>
Time : 1105688393770
EPC  : 01024c6bfd
Send EPC to Savant.
```

この例で Reader は ”01024c6bfd” という ID を読み込み、 133.9.68.181 のマシン (Savant) の 5100 番ポートへ送信している。

次に Savant の動作を見る。

```
> java RFIDSavant
<< Debug Mode >>
Listen Port (Reader) : 5100
Listen Port (Client) : 5101
Interface            : eth0
Network Prefix       : 2001020001251420

<< Connected from Reader >>
From             : 133.9.68.182
To               : 133.9.68.181
Receive EPC      : 01024c6bfd
Generate Address : 2001:200:125:1420:30cb:6c55:ecaa:e49
Exec : ifconfig eth0 inet6 add 2001:200:125:1420:30cb:6c55:ecaa:e49/64
Connection closed.
```

Savant は Reader から EPC を受信すると、それをもとに 64 bit 長のハッシュ値を生成する。この例では、受信した ”01024c6bfd” をもとに、ハッシュ値 ”30cb6c55ecaa0e49” を生成している。そして、あらかじめ登録されたネットワークプレフィックス ”2001020001251420” と結合して IPv6 アドレス ”2001:200:125:1420:30cb:6c55:ecaa:e49” を取得し、 ifconfig コマンドを利用してネット

ワークインターフェースに登録している。

Savant 上で ifconfig コマンドを実行すると、取得した IPv6 アドレスが登録されていることが確認できる。

```
> ifconfig eth0
eth0      Link encap:Ethernet  HWaddr 00:11:43:33:24:9F
          inet addr:133.9.68.181  Bcast:133.9.68.191  Mask:255.255.255.192
          inet6 addr: fe80::211:43ff:fe33:249f/64 Scope:Link
          inet6 addr: 2001:200:125:1420:30cb:6c55:ecaa:e49/64 Scope:Global
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:211665113 errors:0 dropped:0 overruns:0 frame:1
          TX packets:149170090 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:100
          RX bytes:3344772320 (3189.8 Mb)  TX bytes:3250960277 (3100.3 Mb)
          Interrupt:28
```

次に、もう一度同じタグを Reader に読み込ませた際、つまりタグがリーダの検知範囲から出て行った際の Savant の出力を示す。

```
<< Connected from Reader >>
From             : 133.9.68.182
To               : 133.9.68.181
Receive EPC      : 01024c6bfd
Generate Address : 2001:200:125:1420:30cb:6c55:ecaa:e49
Exec : ifconfig eth0 inet6 del 2001:200:125:1420:30cb:6c55:ecaa:e49/64
Connection closed.
```

Savant は 1 度目に EPC を受信した際と同様の手順で IPv6 アドレスを取得し、 ifconfig コマンドを用いてネットワークインターフェースからアドレスを削除している。

5.5.2 実験 2

実験を行う前にまず Reader に実験 1 で用いたものと同じタグを読み込ませて、タグがリーダの検知範囲内にある状況を作り出しておく。

Client を起動すると、まずタグが存在するネットワークのプレフィックスと、タグの EPC の入力が要求される。

```
> java Client
Enter Prefix : 2001020001251420
Enter EPC    : 01024c6bfd
Connecting to 2001:200:125:1420:30cb:6c55:ecaa:e49 ...
> getEPC
01024c6bfd
> exit
Connection Closed.
```

Client は入力されたネットワークプレフィックス ”2001020001251420”と EPC”01024c6bfd”をもとに、 Savant と同様の方法で IPv6 アドレスを取得し、そのアドレスに対して通信を開始する。ここでは例として ”getEPC”コマンドを送信している。それに対して Savant はアクセスされたタグの EPC である ”01024c6bfd”を返している。

Savant 側の出力を以下に示す。

```
<< Connected from Client >>
From             : 2001:200:125:1420:0:0:0:187
To               : 2001:200:125:1420:30cb:6c55:ecaa:e49
Receive Command  : getEPC
Return to Client : 01024c6bfd
Receive Command  : exit
Connection closed.
```

以上の結果より、タグの EPC から取得した IPv6 アドレスに対して通信を行い、タグの情報を取得できることが確認できた。

第 6 章

結論

6.1 まとめ

本研究では、RFID タグに IPv6 アドレスを割り当てることにより、IPv6 ネットワークを介して RFID システムを利用する方式を考案した。また、提案方式において IPv6 アドレスを生成する方法として、タグ固有の ID である EPC のハッシュ値と、タグが属するネットワークのプレフィックスを組み合わせて IPv6 アドレスとする方式を提案した。

さらに、プログラムを作成し動作検証を行うことにより、考案した方式を利用して実際に RFID システムを利用できることを示した。

6.2 今後の課題

本研究で残された課題を以下にあげる。

Savant が生成する IPv6 アドレスが重複した場合の処理

4 章の提案方式では 96 bit 長の EPC を 64 bit 長に縮めて利用している。そのため、もともとの EPC が異なっていても、ハッシュ関数によって生成される値が重複してしまう可能性がある。

本論文では重複する可能性が限りなく低いことを計算によって示したが、少しでも可能性がある以上、システムを実用化する際には生成された IPv6 アドレスが重複した場合の処理を検討する必要がある。

ハッシュ関数の改良

5 章の実験で用いたプログラム中では、IPv6 アドレスを生成する際のハッシュ関数として、MD5 アルゴリズムで 128 bit のハッシュ値を求め、その上位 64 bit と下位 64 bit の排他的論理

和を 64 bit のハッシュ値とする方法をとっている。

しかし、この方法では 4.2 節で示したハッシュ関数の要件を満たしていることが保証されない。上記の方法の数学的証明を行うか、もしくは新たな 64 bit ハッシュアルゴリズムを設計するべきである。

Savant の改良

実験で用いた Savant プログラムは、 IPv6 アドレスを生成し、ユーザと通信を行う機能しか持たない。実際に利用するためには、 ONS や EPCIS を利用して、タグの EPC に関連する情報を得る機能も実装する必要がある。

謝辞

本学士論文の作成にあたり日頃より御指導を頂いた早稲田大学理工学部の後藤滋樹教授に深く感謝致します。また、多大なるご協力を頂いたフランステレコム株式会社の美尾治生氏、井口誠氏に感謝いたします。最後に、研究を進める上で貴重なアドバイスを頂いた後藤研究室の福田浩章氏、竹谷賢二氏、荒井大輔氏、笹川真氏、関宏規氏、実験環境を提供して頂いた石井勇弥氏、日頃より助言を頂いた後藤研究室の諸氏に感謝いたします。

参考文献

[1] Tag Data Standard Work Group, Steve Rehling, "EPC Tag Data Specification Version 1.1", April 2004.

[2] David L. Brock, "Integrating the Electronic Product Code (EPC) and the Global Trade Item Number (GTIN)", November 2001.

[3] Daniel Engels, "The Use of the Electronic Product Code", February 2003.

[4] 鈴木由佳, "IPv6 アドレス活用に関する家電 メーカー需要調査 ", JPNIC, July 2003.

[5] IPv6 アドレスポリシー企画策定専門家チーム, "IPv6 の新しいアドレス利用形態に関する報告書 ", JPNIC, March 2004.

[6] S. Deering, R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", RFC 2460, December 1998.

[7] D. Johnson, C. Perkins, J. Arkko, "Mobility Support in IPv6", RFC 3775, June 2004.

[8] 日本自動認識システム協会, "これでわかった RFID", オーム社, September 2003.

[9] NTT データ・ユビキタス研究会, 荒川弘熙 編, "IC タグって何だ?", カットシステム, November 2003.

[10] RFID テクノロジ編集部, "無線 IC タグのすべて ", 日経 BP 社, April 2004.

[11] 根日屋英之, 植竹古都美, "ユビキタス無線工学と微細 RFID", 東京電機大学出版局, July 2004.

[12] 美崎薫, "ユビキタスがわかる本 ", オーム社, April 2004.

[13] EPCglobal
`http://www.epcglobalinc.org/`

[14] RFID Journal
http://www.rfidjournal.com/

[15] Auto-ID Labs. JAPAN
http://www.auto-id.jp/

[16] RFID テクノロジ
http://itpro.nikkeibp.co.jp/rfid/

[17] Phidgets Inc.
http://www.phidgets.com/

付録 A

作成したプログラム

A.1 RFIDReader.java

```
import java.io.*;
import java.net.*;
import java.util.*;
import Phidgets.*;

public class RFIDReader extends _IPhidgetRFIDEventsAdapter {
    private static final String CONF_FILE = "RFIDReader.conf";
    private static final String DELIMITER = ";,\t ";

    private final boolean debug;    // Debug Mode
    private final String dstAddress;    // Savant Address
    private final int dstPort;  // Savant Listen Port
    private final int interval; // Polling Interval

    private Map map;

    public static void main(String[] args) {
        new RFIDReader();
        return;
    }

    private RFIDReader(){
        boolean db = false;
        String add = "";
        int port = 0;
        int itv =0;
        try {
```

```
        FileReader fr = new FileReader(CONF_FILE);
        BufferedReader br = new BufferedReader(fr);

        db = new StringTokenizer(br.readLine(), DELIMITER).nextToken().equals("true");
        add = (new StringTokenizer(br.readLine(), DELIMITER)).nextToken();
        port = Integer.parseInt((new StringTokenizer(br.readLine(), DELIMITER)).nextToken());
        itv = Integer.parseInt((new StringTokenizer(br.readLine(), DELIMITER)).nextToken());

        br.close();
        fr.close();
    } catch (NumberFormatException e) {
        e.printStackTrace();
        System.exit(1);
    } catch (FileNotFoundException e) {
        e.printStackTrace();
        System.exit(1);
    } catch (IOException e) {
        e.printStackTrace();
        System.exit(1);
    }

    debug = db;
    dstAddress = add;
    dstPort = port;
    interval = itv;

    if(debug){
        System.out.println("<< Debug Mode >>");
        System.out.println("Destination Address : " + dstAddress);
        System.out.println("Destination Port    : " + dstPort);
        System.out.println("Read Interval       : " + interval + " msec");
    }

    map = new HashMap();

    PhidgetRFID phid = new PhidgetRFID();
    phid.add_IPhidgetRFIDEventsListener(this);
    if (phid.Open(false) == false){
        System.out.println("Could not find a PhidgetRFID");
        return;
    }
```

```
        if(debug) System.out.println("Start Reading ...");
        phid.start();
    }

    public void OnTag(_IPhidgetRFIDEvents_OnTagEvent e){
        String epc = e.get_TagNumber();
        Date d = new Date();
        long currentTime = d.getTime();

        long lastTime;
        if(map.containsKey(epc)){
            lastTime = ((Long)map.get(epc)).longValue();
        }else{
            lastTime = 0;
        }

        if(currentTime > lastTime + interval){
            if(debug) System.out.println("Read a Tag.");
            if(debug) System.out.println("EPC : " + epc);

            TCPSender tcps = new TCPSender(dstAddress, dstPort, epc);
            if(debug) System.out.println("Session Start.");
            if(debug) System.out.println("");
            tcps.start();
        }

        map.put(epc, new Long(d.getTime()));
    }
}

class TCPSender extends Thread {
    private String address; // Send Address
    private int port;   // Send Port
    private String data;    // Send Data

    public TCPSender(String dstAddress, int dstPort, String sendData){
        address = dstAddress;
        port = dstPort;
        data = sendData;
    }
```

```
    public void run(){
        try {
            Socket sock = new Socket(address, port);
            DataOutputStream dos = new DataOutputStream(sock.getOutputStream());
            dos.writeUTF(data);
            dos.close();
            sock.close();
        } catch (UnknownHostException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
        return;
    }
}
```

A.2 RFIDSavant.java

```
import java.io.*;
import java.net.*;
import java.util.*;

public class RFIDSavant{
    private static final String CONF_FILE = "RFIDSavant.conf";  // Config File
    private static final String DELIMITER = ";,\t ";

    private final boolean debug;            // Debug Mode
    private final int listenPortReader; // Listen Port (Reader)
    private final int listenPort;   // Listen Port (Client)
    private final String prefix;            // Network Prefix

    private Ifconfig ic;
    private Map map;

    public static void main(String[] args){
        new RFIDSavant();
        return;
    }

    private RFIDSavant(){
```

```
int lpr = 0;
int lp = 0;
boolean db = false;
String pre = "";
String intf = "";

try {
    FileReader fr = new FileReader(CONF_FILE);
    BufferedReader br = new BufferedReader(fr);

    db = new StringTokenizer(br.readLine(), DELIMITER).nextToken().equals("true");
    lpr = Integer.parseInt((new StringTokenizer(br.readLine(), DELIMITER)).nextToken());
    lp = Integer.parseInt((new StringTokenizer(br.readLine(), DELIMITER)).nextToken());
    pre = new StringTokenizer(br.readLine(), DELIMITER).nextToken();
    intf = new StringTokenizer(br.readLine(), DELIMITER).nextToken();

    br.close();
    fr.close();
} catch (NumberFormatException e) {
    e.printStackTrace();
    System.exit(1);
} catch (FileNotFoundException e) {
    e.printStackTrace();
    System.exit(1);
} catch (IOException e) {
    e.printStackTrace();
    System.exit(1);
}

debug = db;
listenPortReader = lpr;
listenPort = lp;
ic = new Ifconfig(intf, debug);
prefix = pre;

if(debug){
    System.out.println("<< Debug Mode >>");
    System.out.println("Listen Port (Reader) : " + listenPortReader);
    System.out.println("Listen Port (Client) : " + listenPort);
    System.out.println("Interface            : " + intf);
    System.out.println("Network Prefix       : " + prefix);
```

```
            System.out.println();
        }
        map = new HashMap();

        EPCReceiver epcr;
        Receiver r;
        try {
            epcr = new EPCReceiver(listenPortReader);
            epcr.start();
            r = new Receiver(listenPort);
            r.start();
        } catch (IOException e1) {
            e1.printStackTrace();
            System.exit(1);
        }
    }

    private class EPCReceiver extends Thread{
        private int listenPort;
        ServerSocket ssock;

        public EPCReceiver(int p) throws IOException{
            listenPort  = p;
            ssock = new ServerSocket(listenPort);
        }

        public void run(){
            try {
                while(true){
                    Socket sock = ssock.accept();
                    SessionFromReader s = new SessionFromReader(sock);
                    s.start();
                }
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }

    private class SessionFromReader extends Thread{
        private Socket sock;
```

```
    public SessionFromReader(Socket s){
        sock = s;
    }

    public void run(){
        try {
            if(debug) System.out.println("<< Connected from Reader >>");
            if(debug) System.out.println("From             : "
                                        + sock.getInetAddress().getHostAddress());
            if(debug) System.out.println("To               : "
                                        + sock.getLocalAddress().getHostAddress());

            BufferedReader br = new BufferedReader
                                (new InputStreamReader(sock.getInputStream()));
            String epc = br.readLine();
            br.close();
            sock.close();

            if(debug) System.out.println("Receive EPC      : " + epc);
            InetAddress address = IPv6AddressGenerator.generate(prefix, epc);
            if(debug) System.out.println("Generate Address : " + address.getHostAddress());
            if(map.containsKey(address)){
                ic.deleteIPv6(address);
                map.remove(address);
            }else{
                ic.addIPv6(address);
                map.put(address, epc);
            }
            if(debug) System.out.println("Connection closed.");
            if(debug) System.out.println();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}

private class Receiver extends Thread{
    private int listenPort;
    ServerSocket ssock;
```

```
    public Receiver(int p) throws IOException{
        listenPort  = p;
        ssock = new ServerSocket();
        InetAddress ia = InetAddress.getByName("::");
        InetSocketAddress isa = new InetSocketAddress(ia, listenPort);
        ssock.bind(isa);
    }

    public void run(){
        try {
            while(true){
                Socket sock = ssock.accept();
                SessionFromUser s = new SessionFromUser(sock);
                s.start();
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}

private class SessionFromUser extends Thread{
    private static final String COMMAND_GET_EPC = "getEPC";
    private static final String COMMAND_EXIT        = "exit";

    private Socket sock;

    public SessionFromUser(Socket s){
        sock = s;
    }

    public void run(){
        try {
            String command, ret;

            if(debug) System.out.println("<< Connected from Client >>");
            if(debug) System.out.println("From             : "
                                        + sock.getInetAddress().getHostAddress());
            if(debug) System.out.println("To               : "
                                        + sock.getLocalAddress().getHostAddress());
```

```
                BufferedReader br = new BufferedReader
                                    (new InputStreamReader(sock.getInputStream()));
                BufferedWriter bw = new BufferedWriter
                                    (new OutputStreamWriter(sock.getOutputStream()));
                while(true){
                    command = br.readLine();
                    if(debug) System.out.println("Receive Command  : " + command);
                    if(command.equals(COMMAND_EXIT)) break;
                    else if(command.equals(COMMAND_GET_EPC)) ret = getEPC();
                    else ret = command + " : Command Not Found.";
                    if(debug) System.out.println("Return to Client : " + ret);
                    bw.write(ret);
                    bw.newLine();
                    bw.flush();
                }
                br.close();
                bw.close();
                sock.close();
                if(debug) System.out.println("Connection closed.");
                if(debug) System.out.println();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }

        private String getEPC(){
            InetAddress ia = sock.getLocalAddress();
            String ret = (String) map.get(ia);
            return ret;
        }
    }
}
```

A.3 IPv6AddressGenerator.java

```
import java.net.*;
import java.security.*;

public class IPv6AddressGenerator {

    public static InetAddress generate(String prefix, String epc){
```

```
    byte[] bPrefix = stringToByte(prefix);
    if(bPrefix.length != 8) return null;
    byte[] bEPC = stringToByte(epc);
    byte[] hEPC = hash(bEPC);
    byte[] bAddress = new byte[16];
    for(int i = 0; i < 8; i++){
        bAddress[i] = bPrefix[i];
        bAddress[i + 8] = hEPC[i];
    }
    InetAddress address = null;
    try {
        address = InetAddress.getByAddress(bAddress);
    } catch (UnknownHostException e) {}
    return address;
}

public static byte[] hash(byte[] source){
    MessageDigest md;
    byte[] digest = new byte[8];
    try {
        md = MessageDigest.getInstance("MD5");
        md.update(source);
        byte[] tmp = md.digest();
        for(int i = 0; i < 8; i++){
            digest[i] = (byte) (tmp[i] ^ tmp[i + 8]);
        }
    } catch (NoSuchAlgorithmException e) {
        e.printStackTrace();
    }
    return digest;
}

private static byte[] stringToByte(String str){
    byte[] byteset = {0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
                      0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f};
    byte[] ret = new byte[str.length() / 2];
    char[] strArray = str.toCharArray();
    for(int i = 0; i < strArray.length; i += 2){
        byte left = byteset[charToInt(strArray[i])];
        byte right = byteset[charToInt(strArray[i + 1])];
        ret[i / 2] = (byte) ((left << 4) | right);
```

```
        }
        return ret;
    }

    private static int charToInt(char c){
        char[] charset = {'0', '1', '2', '3', '4', '5', '6', '7',
                          '8', '9', 'a', 'b', 'c', 'd', 'e', 'f'};
        char ret;
        for(ret = 0; ret < 15; ret++){
            if(c == charset[ret]) break;
        }
        return ret;
    }
}
```

A.4 Ifconfig.java

```
import java.io.*;
import java.net.*;

public class Ifconfig {
    private String intf;
    private boolean debug;

    private static final String IFCONFIG    = "ifconfig";
    private static final String ADD     = "add";
    private static final String DELETE  = "del";
    private static final String INET6 = "inet6";
    private static final String PREFIX  = "/64";

    public Ifconfig(String str){
        intf = str;
        debug = false;
    }

    public Ifconfig(String str, boolean b){
        intf = str;
        debug = b;
    }

    public void addIPv6(InetAddress address) throws IOException{
```

```
        String[] command = {IFCONFIG, intf, INET6, ADD, address.getHostAddress() + PREFIX};
        Process process = Runtime.getRuntime().exec(command);
        if(debug) printVerbose(command, process);
    }

    public void deleteIPv6(InetAddress address) throws IOException{
        String[] command = {IFCONFIG, intf, INET6, DELETE, address.getHostAddress() + PREFIX};
        Process process = Runtime.getRuntime().exec(command);
        if(debug) printVerbose(command, process);
    }

    private void printVerbose(String[] command, Process process){
        System.out.print("Exec : ");
        for(int i = 0; i < command.length; i++){
            System.out.print(command[i]);
            System.out.print(" ");
        }
        System.out.println();
        try {
            Reader in = new InputStreamReader(process.getInputStream());
            int c = -1;
            while ((c = in.read()) != -1) { System.out.print((char) c); }
            in.close();
            Reader err = new InputStreamReader(process.getErrorStream());
            c = -1;
            while ((c = err.read()) != -1) { System.out.print((char) c); }
            err.close();
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }
}
```

A.5 Client.java

```
import java.io.*;
import java.net.*;

public class Client {

    public static void main(String[] args) {
```

```
        BufferedReader stdin = new BufferedReader(new InputStreamReader(System.in));

        try {
            System.out.print("Enter Prefix : ");
            String prefix = stdin.readLine();
            System.out.print("Enter EPC    : ");
            String epc = stdin.readLine();
            InetAddress ia = IPv6AddressGenerator.generate(prefix, epc);
            System.out.println("Connecting to " + ia.getHostAddress() + " ...");
            Socket sock = new Socket(ia, 5101);
            BufferedReader br = new BufferedReader
                                (new InputStreamReader(sock.getInputStream()));
            BufferedWriter bw = new BufferedWriter
                                (new OutputStreamWriter(sock.getOutputStream()));

            while(true){
                System.out.print("> ");
                String command = stdin.readLine();
                bw.write(command);
                bw.newLine();
                bw.flush();
                String ret = br.readLine();
                if(ret == null) break;
                System.out.println(ret);
            }

            System.out.println("Connection Closed.");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```