卒業論文　　2002 年度 (平成 14 年度)

ANARCH: MANET に適した名前解決機構

指導教員

慶應義塾大学環境情報学部

徳田　英幸

村井　純

楠本　博之

中村　修

南　政樹

慶應義塾大学総合政策学部

青木 基衣

卒業論文要旨　　2002 年度 (平成 14 年度)

ANARCH: MANET に適した名前解決機構

近年，既存のインフラストラクチャに依存せずに, マルチホップ通信機能を持つ無線端末同士が即興的に構築するモバイルアドホックネットワーク（MANET）の研究に注目が集まっている．MANET では，端末自身が他端末から送られたパケットを中継する機能を有し，こうした端末が複数集まってマルチホップ転送機能が実現される。そのため，既存の通信インフラストラクチャを必要とせず，無線通信を用いて様々な場所でネットワークを構築することができる．こうしたネットワークは, 情報機器同士のみならず人間同士の協調作業を行うためのアプリケーション基盤になると予想される．そのため，インターネットで利用されているように情報端末に識別子としての名前を付け，ユーザに対してより利便性の高いネットワーク環境を構築することは重要である．

現在インターネットでは DNS (Domain Name System) と呼ばれる名前解決機構が広く用いられており，IP アドレスに分かりやすく覚えやすい文字列型のホスト名が対応付けられている．そのためユーザは IP アドレスを意識することなくホスト名だけで通信を行うことが可能であり，WWW を始めとしたインターネットのアプリケーションで広く使用されている．しかし，MANET は既存の有線ネットワークとは異なり，ネットワークリンク・トポロジが不安定であり IP アドレスがノードに必ずしも固定されているとは限らないため，サーバのような中央管理システムを想定することが困難である．このような理由から，インターネットで用いられている名前解決機構をそのまま利用するだけでは不十分である．MANET で名前解決を行うためには，その特徴に応じた名前解決機構が必要になるが，現在はそのような機構についてまだ十分に研究されていない．本論文では, アプリケーション構築を支援するための基盤技術としてアドホックネットワークに適した名前解決機構 ANARCH (Autonomous Name Resolution Scheme) を提案する．

ANARCH は, ユーザの好みに基づいたネットワーク内で一意なホスト識別子となる名前割り当てを行い, その名前を用いた通信機構をアプリケーションに提供する．本論文では，アドホックネットワークを構築するためのルーティングプロトコルの一つである DSR (Dynamic Source Routing) を用いて 8 台のノードで構築したアドホックネットワーク上で ANARCH を実装し評価を行った．名前解決に要する時間の評価結果から，ANARCH は名前解決に起因するオーバヘッドを最小限にとどめながら，3 ホップ先のノードの名前解決の場合約 5 秒という現実的な時間内での名前解決を実現しつつ端末数の増加に対するスケーラビリティを持つことが示された．

慶應義塾大学　総合政策学部
青木基衣

Abstract of Bachelor's Thesis

ANARCH: A Name Resolution Scheme for Mobile Ad Hoc Networks

Due to recent advancement in technology of wireless communication, use of small sized computers in a mobile environment is becoming proliferated. Mobile ad hoc networks are temporarily constructed by mobile hosts in the absence of established infrastructures. Such networks can constitute a base for human cooperative works as well as for communications between information systems. However, there has not been sufficient work of name resolution systems suitable for ad hoc networks. In this paper, we propose a name resolution system called Autonomous Name Resolution Scheme (ANARCH) suitable for ad hoc networks as a base technology for application construction environment. ANARCH can provide a node with a unique user-oriented name in an ad hoc network, thereby adapting to dynamic changes of network identifier configuration. In addition, ANARCH relieves users and their applications from the necessity of identifying the network addresses (IP addresses) of destination nodes by introducing unique names as employed in the Domain Name System (DNS). We have designed and implemented ANARCH on FreeBSD, and evaluated the performance in an ad hoc network system built with PCs running Dynamic Source Routing (DSR) protocol. In our experimental results, ANARCH exhibited effective name resolution in a practical time while reducing the overhead of operations.

Motoi Aoki

Faculty of Policy Management
Keio University

目 次

図 目 次

表 目 次

第1章　序論

1.1　研究動機

近年，無線通信機能を持つ情報端末がルータの役割を担いパケットを中継することで，即時的にネットワークを構成するモバイルアドホックネットワーク (MANET) に関する研究が盛んに行われている．MANET では，中間端末のマルチホップ機能により，直接通信できない端末同士でもネットワークを組み，パケットを送受信することが可能となる．そのため，既存のインフラストラクチャを必要とせず，無線通信を用いて様々な場所でネットワークを構築することができる．この特徴から，災害地での利用，また会議などに持ち寄った端末同士で組むネットワーク，タクシー，バスなどで構築される交通ネットワークなどの利用方法が考えられる．MANET を構築するための経路制御プロトコルも多く提案されており，MONARCH プロジェクト [24] の DSR (Dynamic Source Routing) [4] や，AODV (Ad Hoc On-Demand Distance-Vector Routing) [10]，OLSR (Optimized Link State Routing) [9]，TBRPF (Topology Broadcast Based on Reverse-Path Forwarding) [12] などが挙げられる．

こうしたネットワークは, 情報機器同士のみならず人間同士の協調作業を行うためのアプリケーション基盤になると予想される．そのため，インターネットで利用されているように情報端末に識別子としての名前を付け，ユーザに対してより利便性の高いネットワーク環境を構築することは重要である．現在インターネットでは DNS (Domain Name System) [1] と呼ばれる名前解決機構が広く用いられており，IP アドレスに分かりやすく覚えやすい文字列型のホスト名が対応付けられている．そのためユーザは IP アドレスを意識することなくホスト名だけで通信を行うことが可能であり，WWW を始めとしたインターネットのアプリケーションで広く使用されている．

しかし，MANET は既存の有線ネットワークとは異なり，ネットワークリンク・トポロジが不安定であり，一つの IP アドレスが特定のノードに必ずしも固定されているとは限らないため，サーバのような中央管理システムを想定することが困難である．このような理由から，インターネットで用いられている名前解決機構をそのまま利用するだけでは不十分である．MANET で名前解決を行うためには，この特徴に応じた名前解決機構が必要になるが，現在はそのような機構についてまだ十分に研究されていない．そこで本論文では, アプリケーション構築を支援するための基盤技術として MANET に適した名前解決機構を提案する．

1.2 本研究の目的および意義

MANET が実用化される場面を考えると，ホームネットワークや，小さな会議室でのデータ共有を目的とするような小規模なネットワークから，交通ネットワークや災害地での利用などの大規模なネットワークまで広い用途が考えられる．また，ネットワークを構成する端末の移動が著しい MANET や，比較的静的に構築される MANET があることも想定される．ネットワークの規模の違いによって，シングルホップのみで構築可能か，またはマルチホップ通信機能が利用されるかも異なる．そのため MANET に適した名前解決機構も利用される MANET の特徴に応じて変わるであろう．

また，名前解決を行う目的も様々であると考えられる．利便性のためにホスト名を利用したい場合や，特定のサービスの検索を可能にすることを目的として，端末が提供しているサービスに関連付けられた名前，または GPS などで取得される位置情報に関連付けられた名前を利用したい場合などである．

ANARCH は単体でこれらの要求すべてに対応することを目的としたシステムではないが，どのような特性の MANET または名前を利用する際にも基盤として利用できる汎用的な機構として設計されている．ANARCH はユーザの好みに基づいた MANET 内で一意なホスト名を割り当てることに焦点を当てているが，他のモジュールと併用することで，サービスや位置情報などと関連付けられた名前を利用することも可能である．また名前解決を行う範囲を指定できるため，大規模な MANET においても必要な範囲で名前解決を提供することが可能であり，各端末は名前解決に依る負荷を調整することができる．加えて，ANARCH のパラメータを調整することで様々な特性の MANET に適応させることができる．

1.3 本論文の構成

本論文の構成として，まず第 2 章で研究背景として MANET および名前解決機構について述べ，関連研究を紹介する．第 3 章では MANET における名前解決の問題点を分析し，要件を考察する．そして幾つかの選択可能な手法とそのトレードオフについて述べる．さらに第 4 章，第 5 章でフラットな名前空間と「名前の自律割り当て」手法を用いた ANARCH の設計と実装について述べ，最後に第 6 章でシングルホップネットワークとマルチホップネットワークにおけるオーバヘッドと名前解決に要する時間を評価し第 7 章で結論を述べる．

第2章　研究背景と関連研究

本章ではまず研究背景として MANET と名前解決機構について説明し，次に類似した問題意識を扱った関連研究について考察する．

2.1　研究背景

本節では本研究の背景として，まず無線を通信媒体として移動端末間で一時的，局地的に構築されるモバイルアドホックネットワーク（MANET）について，そしてネットワークの利便性を高める重要なサービスの一つである名前解決機構について説明する．

2.1.1　MANET

モバイルアドホックネットワーク（MANET）とは，無線を通信媒体とした移動体端末（以下ノード）により，既存のネットワークインフラストラクチャに依存せずに即興で一時的，局地的に形成される自律分散型のネットワークである．MANET では，ノードは基地局などの既存のネットワークインフラストラクチャを介してではなく，直接通信を行う．その際，通信先のノードと直接リンクが接続されていない場合はその中間にあるノードを中継してデータのやり取りを行う．これをマルチホップ通信と呼ぶ．MANET は中央管理的なシステムを持たず，各ノードの自律的動作によって構築されるため，不安定なネットワークリンクやネットワークトポロジの動的な変化にも対応することができるという特徴がある．そのため, 当初は主として軍事利用や非常用の通信手段として研究が始められた．

図 2.1 に MANET のイメージを示す．さらに MANET の特徴を以下に列挙する．

- 自律分散ネットワーク

  サーバや無線基地局などの中央集中管理的なノードが存在せず，すべてのノードが同等の機能を持ち同じ動作を自律的に行うことにより構築される分散ネットワークである．

- 既存のネットワークインフラストラクチャが不要

  無線通信機能を持つ移動端末から動的に構成されるネットワークであるため，無線基地局などの既存ネットワークインフラストラクチャが不要である．この様子を図 2.2 に示す．

図 2.1: MANET イメージ図

- ネットワークトポロジ，通信経路の頻繁な変化

  ネットワークを構成するモバイルノードの移動に伴い，リンクが頻繁に接続・切断され，ネットワークトポロジ，通信経路が動的に変化する．

- マルチホップ通信による，無線通信範囲外の端末同士の通信が可能

  直接リンクが接続されていない，無線通信範囲外にいる端末同士であっても，お互いの通信範囲内に存在する別の中間端末を経由することで通信が可能となる．通信の際にデータが別の端末を経由することをホップと呼ぶ．マルチホップ通信機能の様子を図 2.3 に図示する．

- リンク品質が不安定

  ノードの移動や電波状態の変化などのため，ネットワーク内のノード間のリンク品質が常に一定でであるとは限らない．

- セキュリティの脅威

  MANET は，物理的なセキュリティ面において，有線ネットワークに比べ脅威にさらされる危険性がはるかに高い．MANET の研究において，セキュリティは残されている重要な課題である．

- 様々なルーティングプロトコルの存在

  MANET 上でも IP を基盤とした技術を利用するため，様々なルーティングプロトコルが開発されてきた．DSDV [2] などは，有線ネットワークで利用されているルーティングプロトコルを応用・拡張したものである．一方，AODV [10], DSR [4] などは，MANET を構築するために新たに開発されたプロトコルである．一般に，ルーティングプロトコルは状況に応じて最適なものが選択される．

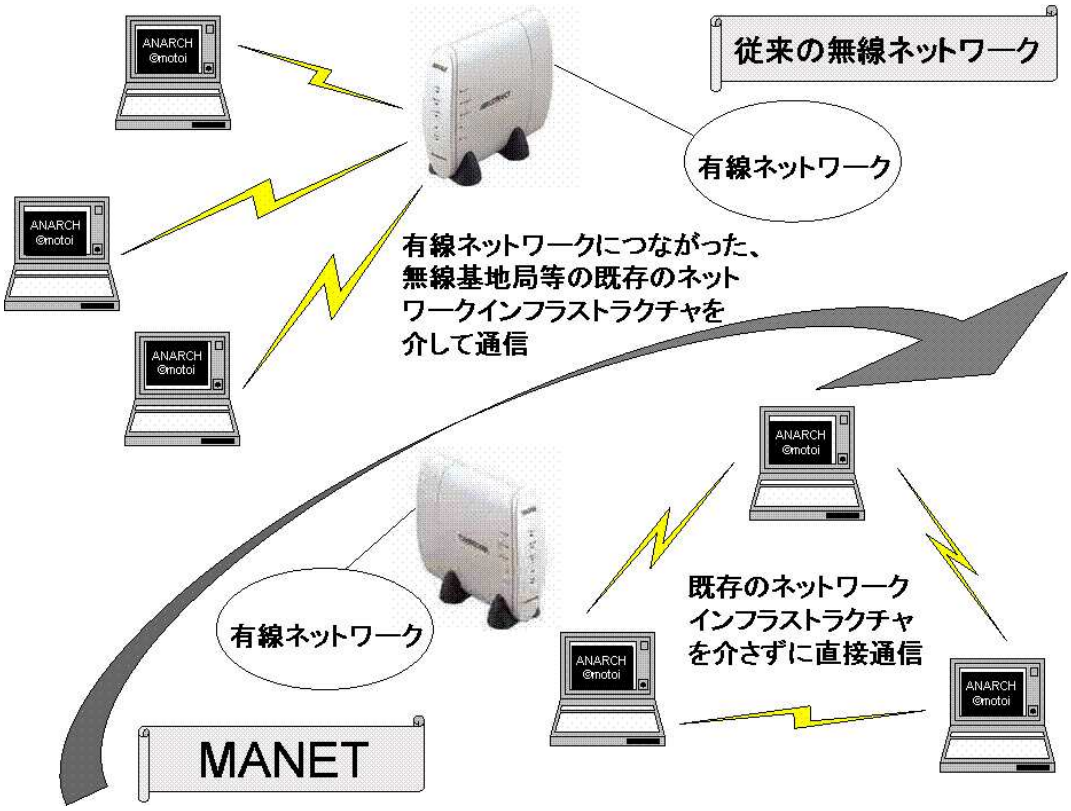

図 2.2: 既存のネットワークインフラストラクチャに非依存

MANET の研究は，主に IETF の MANET（Mobile Ad-hoc Network）ワーキンググループ [20] で進めてられてきた．MANET ワーキンググループでは，MANET におけるインターネット標準トラックの導入を目標として，ルーティングプロトコルの開発及び策定が行われており，MANET を実現するための様々なアーキテクチャモデルやプロトコルが提案されている．MANET のルーティングプロトコルは経路探索と経路維持を可能にするもので，代表的なものとして前述の DSR [4]，AODV [10]，OLSR [9]，TBRPF [12] などが挙げられる．

近年では，このような MANET の特徴を生かした適用範囲が広がりつつあり, さまざまな形態への応用が考えられている．MANET のホップ機能を利用した比較的大規模なネットワークを想定したものとしては，車車間ネットワーク，交通ネットワーク，センサネットワーク，災害地での利用などが挙げられる．一方，モバイル環境下でより小規模なネットワークを構築する利用法も注目されている．例えば，携帯電話やノートパソコン，PDA などの携帯情報機器間でネットワークを構築することにより, 会議中の議事録やメモの共有, Web などのサービスの提供，名刺交換, デジタルカメラからノートパソコンへの簡易なデータ転送を行うなどの利用である．また，博物館，駅，商店などの公共の場での情報提供といった利用も考えられている．

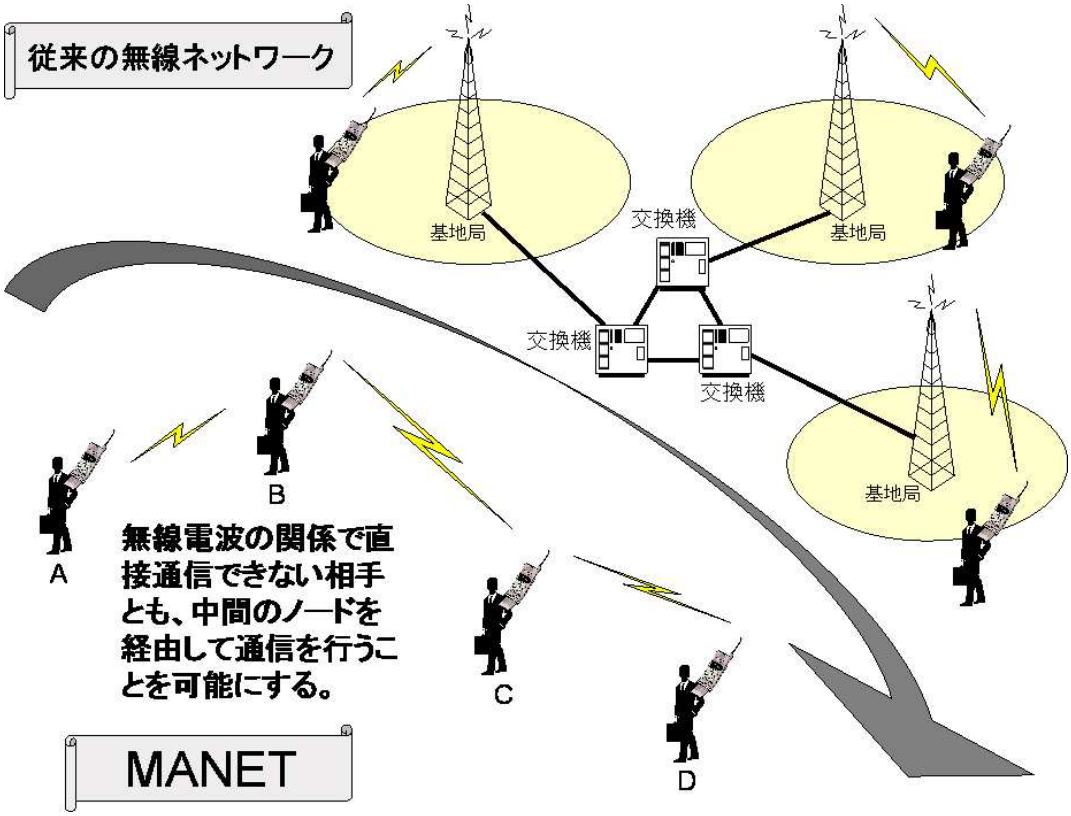

図 2.3: マルチホップ機能による通信が可能

2.1.2 名前解決機構

名前解決機構とは，人間にとって識別が容易な情報端末識別子を用意し，それをネットワークプロトコル用の識別子に対応付けするシステムのことである．一般的に，人間にとって識別しやすい文字列型データを IP で使用する IP アドレス情報に対応付ける．

例えば DNS (Domain Name System) [1] では，TCP/IP ネットワーク環境においてホスト名と呼ばれる階層化された文字列型の識別子を用意し IP アドレスに対応付けている．DNS はホスト名と IP アドレスの対応関係を記述したデータベースを管理し，クライアントからの要求に応じてホスト名から対応する IP アドレスを取得できるサービスを提供する．これにより，通常ユーザは覚えにくく分かりにくい IP アドレスではなく，ホスト名だけを意識して通信を行うことが可能になる．本論文でも以後 IP ネットワークを想定して，IP アドレスに名前を対応付けるための名前解決機構について述べる．

2.2 関連研究

本節では，本研究と類似した問題意識を扱った関連研究を紹介する．まず動的ネットワーク環境に適応する名前解決機構について幾つか述べ，次に MANET において一意な識別子を割り当てることを目的とした研究について述べる．

2.2.1 動的ネットワーク環境に適した名前解決機構

DNS

既存の名前解決機構として広く用いられているシステムに DNS (Domain Name System) [1] がある．これは現在のインターネットで広く利用されている．DNS は，名前を管理する空間を階層的なツリー構造で構成した柔軟性に富むシステムであるが，構成が静的なネットワークを想定して設計されている．そのため，階層化された中央管理サーバが IP アドレスと名前の対応付け管理テーブルを静的に保持しており，設定はゾーンデータを記述したファイルを修正して反映させるという静的な定義によって構築される．しかし，MANET では中央サーバに静的な設定ファイルを置くという手法は不十分である．

DDNS

DNS を動的なネットワーク環境に対応させたものとして，DDNS (Dynamic Domain Name System) [15] が挙げられる．これは， DNS サーバ内のテーブルを自動的に更新するシステムで，1998 年初頭ごろから実装が進み，現在実用段階に入っている．DDNS は，元々 DHCP (Dynamic Host Configuration Protocol) [3] やダイアルアップ接続サービスを利用している場合などの，ネットワークに接続するたびに IP アドレスが変わってしまう環境で，端末に固定の名前を割り当てることを目的としていた．これは，端末が IP アドレスを取得するたびにそれを DDNS サーバのデータベースに自動的に登録することで，従来の DNS が提供しているのと同じ機能を提供するシステムである．

DDNS が想定している，IP アドレスが毎回一定でないという特徴はアドホックネットワークの特徴と類似している．そこで，この DDNS サーバをネットワークのすべてのノードに実装することで MANET における名前解決を行うことが考えられる．しかし，この方法を実現するには幾つかの課題が残る．まず，ネットワーク内の DDNS に登録するため DDNS サーバを検索する手法が必要になる．また，ネットワークリンク・トポロジが不安定なネットワーク内のサーバ同士でデータの一貫性を保つ必要もある．さらに，ネットワーク内に設けるべきサーバの数も考慮しなければならない．もしすべてのノードがサーバにならないならば，サーバにするノードを動的に選出する方法が必要になる．またもしすべてのノードがサーバになるなら，情報の同期をとることが問題となる．さらにその場合，各ノードが自分で名前情報を管理することと同義になり，他のサーバに問い合わせる必要性は低下する．すなわち，アドホックネッ

トワーク内で DDNS を用いても，MANET における名前解決の根本問題を解決できるわけではない．

NetBIOS (Network Basic Input Output System)

NetBIOS (Network Basic Input Output System) [22] とは，IBM の LAN Manager で採用された，ネットワークサービスを呼び出すための API インターフェイスである．これは Microsoft Windows [25] ネットワークにおける標準的なトランスポート層プロトコルの 1 つであり，Windows LAN 内で名前解決を行う．NetBIOS は 16byte の，LAN 内でのみ有効な NetBIOS 名を提供する．これには 1 つの IP アドレスに対応する一意な名前と，複数の IP アドレスに対応するグループ名が含まれる．名前の重複に対しては，最初にその名前を使い始めた方が優先される「早い者勝ち」の原則が適用される．

NetBIOS では基本的にブロードキャストを利用して通信相手ノードを検索するが，この方法ではネットワークの規模が大きくなるにつれオーバヘッドが増大するため，Lmhosts [19] と呼ばれる静的ファイルと WINS (Windows Internet Name Service) [26] と呼ばれる中央サーバを併用することも可能である．Lmhosts ファイルは DNS における hosts [17] ファイルに当たるもので，IP アドレスと名前との関係を静的に記述しておくことができる．また WINS と呼ばれる名前解決サーバをネットワーク内に設置することによりブロードキャストが軽減されるが，同時に中央管理システムに依存した名前解決が行われることになる．

NetBIOS を基礎に構築されるシステムとして，NetBEUI (NetBIOS Extended User Interface) [21]，NBT (NetBIOS over TCP/IP) [23] が挙げられる．NetBEUI は，IBM，Microsoft，3COM が LAN Manager を開発する際に NetBIOS におけるサービス内容の拡張や標準的なパケット構造を決めたものであり，LAN Manager を始め LAN Server や Windows for Workgroup，Windows 95/98，Windows NT などで広く用いられている．NetBEUI は自己調整型のプロトコルであり，各ノードに固有のホスト名 (NetBIOS 名) を付けておくと，自動的にお互いを識別し通信することができる．これは小規模ネットワークを想定したもので，数台から 200 台程度までの小〜中規模の LAN 環境において，良好なパフォーマンスを発揮する．欠点はブロードキャストの多用によるネットワークの飽和が起こりやすいことである．逆に NBT は大規模ネットワークを想定したもので，ルーティング可能なプロトコルである．

Service Discovery in On-Demand Ad Hoc Networks

IETF の MANET（Mobile Ad-hoc Network）ワーキンググループ [20] によりドラフトが提出された Service Discovery in On-Demand Ad Hoc Networks [13] は，サービス・ディスカバリをサポートするよう MANET での通常のルート検索プロトコルを拡張したものである．これは，既存のルーティングメカニズムを崩すことなく同時にサービス・ディスカバリを行うことを可能にする．このシステムにより提供される

サービス情報はネットワーク内で一意である必要はない．このサービス情報は service selector と呼ばれ，ポート番号，サービス URL，その他の情報が含まれる．提案されている手法では，route discovery と呼ばれる IP アドレスへのパスを検索する機能に加え，service discovery と呼ばれる service selector と IP アドレスの対応を検索する機能が追加されている．

Name Resolution in on-demand MANETS and external IP Networks

IETF の MANET（Mobile Ad-hoc Network）ワーキンググループ [20] によりドラフトが提出された Name Resolution in on-demand MANETS and external IP Networks [11] も，通常の経路要求に便乗して名前解決を行う．さらに，インターネット・ゲートウェイ (IGW) と呼ばれるノードを通して，外部のグローバルインターネットとも接続可能であり，そのノードは DNS プロキシとして DNS と通信を行う．

2.2.2 MANET における一意な識別子割り当て

zeroconf

MANET 内で衝突しない識別子を割り当てるという点では IETF の zeroconf ワーキンググループ [27] が zeroconf ネットワークを提案している．これは既存設定の存在しないネットワークでリンク層の IPv4 アドレスを割り当てるものである．新たにネットワークに加わるノードは，まだ使用されていない IP アドレスを選択して利用するため，IP アドレスの衝突を避けることが可能になる．zeroconf ネットワークも中央管理サーバが存在しない環境を想定している完全な分散システムであるが，人間に分かりやすい識別子を提供するという点では不十分である．しかし，MANET の IP アドレスの割り当てには十分利用することができる．

第3章　MANET における名前解決機構

本章ではまず名前解決機構を定義し，次に MANET で名前解決をする際の問題点と既存のシステムで対応する場合の限界から考察される要件を示す。そして設計上の考察を述べる。

3.1　定義

本論文で述べる「MANET における名前解決機構」とは，MANET を構成するノードにネットワーク内で一意なホスト名を付けるシステム，と定義される．個々のノードが自律的にユーザの好みに基づいた名前を選択し利用することに加え，ネットワーク内でのそれら名前の一意性を管理することにより，MANET 内で一意な名前を提供しネットワークの利便性を高めることを目的としている．

MANET を利用する場合，まず通信しようとする相手のノードを特定する必要があるが，ネットワークを構築するたびにユーザが通信相手の IP アドレスを確認して互いを認識する方法は現実的ではない．なぜなら，MANET ではあるノードが常に同じ IP アドレスを使用できるとは限らないため，一つの IP アドレスで必ずしも同じノードを特定できないからである．MANET における IP アドレスは，IP 通信を行うために一時的に各ノードに一意に割り当てられるため，ノードに IP アドレスを固定することはできない．さらにネットワークの規模が大きくなるとすべての IP アドレスを確認して相手を特定することは非現実的である．そこでユーザの利便性を向上させるために，通信相手を特定するための容易に認識できる識別子をノードに割り当てる必要が生じる．これを実現するのが MANET における名前解決機構である．

3.2　要件

MANET は，無線を通信媒体として動的に構成されるネットワークであり，有線ネットワークと比較すると表 3.1 に示すように異なる特徴を持つ．そのため，名前解決の際に次のような問題が生じる．

1. ネットワークリンク，ネットワークトポロジの動的な変化，頻繁に起きるノードの出入りなどの理由により，名前解決のための中央管理サーバを想定できない．

表 3.1: 既存の有線ネットワークと無線 MANET の特徴の違い

条件	既存の有線ネットワーク	無線 MANET
ネットワークトポロジ	静的	動的
ネットワークリンク	ほぼ常に安定	ノードの移動に伴い変化
ネットワーク規模	グローバルインターネット（全世界）	ローカルネットワーク（IP サブネットワーク）
IP アドレス	端末に固定	ネットワークを組む度に変化
MAC アドレス	ネットワークインタフェースに固定	ネットワークインタフェースに固定
中央管理機構	有り	無し
名前の重複	無し	有りうる

2. IP アドレスがノードに固定されていないため，IP アドレスと名前の対応情報を静的に保持することができない．そのため名前を何らかの方法で動的に割り当てる必要がある．

3. 名前を動的に割り当てた場合，ネットワーク内で名前の衝突が起きる可能性が生じる．

まず，MANET ではネットワークリンク，ネットワークトポロジの不安定さのため中央管理システムを設置することが困難である．さらに，ノードの移動に伴いネットワークリンクの状態が常に変化しているため，サーバとなったノードと常にリンクが保たれていることは保証されない．また構成ノードが頻繁に変化するため，サーバとなったノードが常にネットワーク内に存在するとは限らない．これらの理由から完全に中央管理機構に依存する設計では不十分である．

第二の問題として IP アドレスと名前の対応情報を静的に保持できないことが挙げられる．これは同じノードが常に同一の IP アドレスを使用できるとは限らないためである．同じ理由で，名前を静的に割り当てることにも問題があるので，名前解決機構が分散的，動的な方法で名前を割り当てる必要がある．

しかし，これが三番目の問題につながる．すなわち，ネットワーク内で動的に名前を割り当てると名前の衝突を引き起こす可能性がある．ある時点では一意な名前であっても，新しいノードが追加された時や，他のアドホックネットワークと合併した時に名前の衝突が生じる可能性がある．そのため，名前解決機構は名前の衝突に動的に対応できなければならない．

3.3 設計上の考察

MANET に適した名前解決機構を実現するためには，先に述べた要件を満たす必要がある．本節では，MANET に特有な問題に対応可能な名前解決機構として ANARCH（Autonomous Name Resolution Scheme：自律的な名前割り当てによる名前解決機構）を設計する際の，利用する名前空間，名前の割り当て手法，名前の管理手法に関してそれぞれ選択可能な手法とそのトレードオフについて述べる．

3.3.1 名前空間

名前解決機構が提供する名前のデータ構造には以下の二種類が考えられる．

- フラットなデータ構造
- 階層化されたデータ構造

まず，単純な文字列データを名前として用いる方法がある．この場合，名前はフラットな構造になる．これは具体的には ‘dali’ などのシンプルな文字列のことである．そしてもう一つは構造化されたデータを名前として用いる方法である．この場合名前は階層化される．具体的には ‘dali.sfc.keio.ac.jp’ というインターネットで用いられているような名前が挙げられる．

ネットワークの規模が大きい場合は階層化されたデータを名前として用いる方が現実的である．名前が階層化されていれば，名前を管理するデータベースも階層化することが可能になり大規模なネットワークでも管理が容易になる．また MANET で階層化された名前を用いることには，以下の長所が考えられる．

- 名前を論理的に分類することによってネットワーク自体を階層的に扱える
- 物理的な情報を名前に含めることによってネットワークの利便性を高める

MANET は IP プライベートネットワークとして構築されるため，そのままではフラットな構造のネットワークである．しかし，階層化された名前を用いることによって複雑なグルーピングをしなくてもノードを階層的に扱うことが可能になる．例えば，論理的に与えられた位置情報や所属情報などを名前に含めることが考えられる．また，位置情報や提供しているサービス情報などの物理的な情報を名前に含めることによってネットワークの利用性を高めることが可能になる．例えば端末が GPS 情報を取得できる場合，その位置情報を利用することが考えられる．これにより，名前は単に相手を特定するだけでなく付加的な情報も含んだ名前システムの構築を支援するものになるが，同時に名前解決機構の正確さや迅速さを犠牲にする可能性がある．さらに，意味のある名前を利用する場合，名前解決機構はその名前情報を利用するシステムにとっての単一故障点となる．

階層的な名前の他の弱点としては，名前解決のためにシステムにかかる負荷が高くなることや，完全な分散型システムとして実装するのが困難であることが挙げられる．また階層化された名前は，ノードの移動が頻繁に起こり，ネットワークトポロジが激しく変化するネットワークには不適切である．これは，位置情報の変化やノードの移動のたびに名前を変えるとネットワークへの負荷もかかる上，名前の一貫性を保つことが困難になるためである．ANARCH では接続端末が 100 台程度のローカルネットワークを想定しているため，階層化された名前ではなくフラットな名前を採用した．比較的小規模なネットワークではフラットな名前構造を用いた方がシステムへの負荷が少なくなるためである．

3.3.2 名前割り当て手法

名前割り当て手法とは，各ノードにどこでどのように名前を割り当てるかということであり，以下の二つの方法が考えられる．

- 中央管理システムからの割り当て
- 個々のノードによる自律割り当て

適切な名前の割り当て手法は，ネットワークの規模によっても異なる．比較的小規模なネットワークであれば，個々の端末が自律的に好きな名前を自ノードに割り当てることができるだろう．しかし大規模なネットワークになると，名前が衝突しないように特定のシステムが名前を管理してそれを割り当てたり，一定のルールに従って名前を決める必要が生じる．

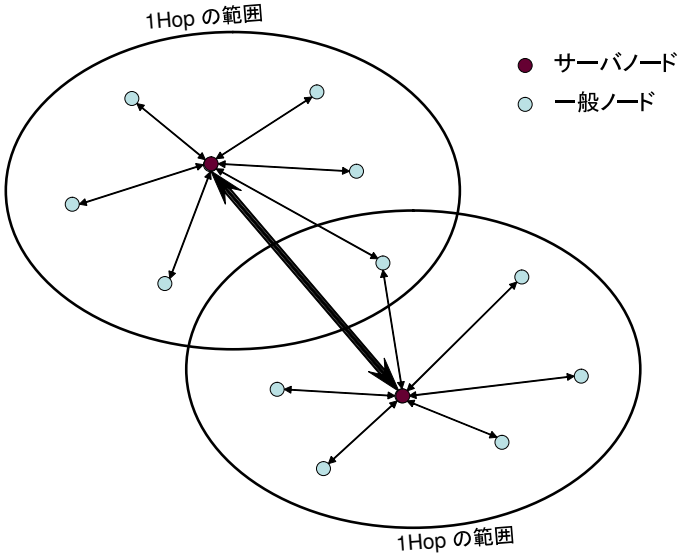

図 3.1: 中央管理システムからの割り当て

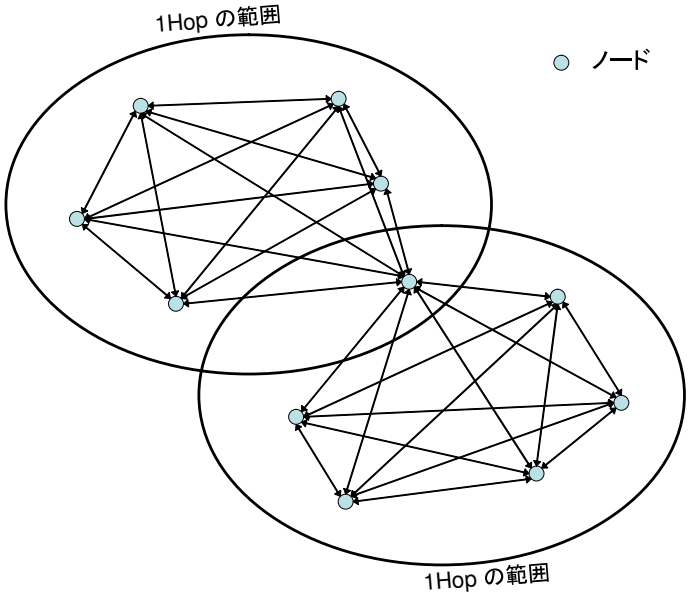

図 3.2: 自律的な割り当て

ネットワーク内の中央管理サーバが名前情報を集中的に維持管理し，割り当てを行う手法の最大のメリットは，名前の衝突が生じないことである．また，サーバがネットワーク内のノード情報を一括管理するため，ネットワーク内のノードはサーバの存在だけを知っていればよいことになる．この方法では，従来の DNS も一部使用可能である．MANET では，サーバとしたノードと常にリンクが保たれているとは限らないため，サーバを幾つか分散させて持つ方法も考えられる．図 3.1 は，MANET 内に名前を割り当てるサーバを幾つか配置して，各ノードは 1 ホップで通信可能なサーバにコンタクトするというモデルを示している．この場合，サーバ同士は互いに既知で名前が衝突しないよう管理されていることを前提とする．表 3.2 に示す疑似コードに，中央管理システムが名前を管理するモデルを略述する．

しかしこの手法ではサーバへのリンクが切れた場合の新たなサーバの選出が問題となる．さらにサーバ間の一貫性を保つこと，サーバへのリンクが切れた場合の処理を考慮すると，システム構築のコストに見合うだけの効果を得ることは難しい．中央サーバへ依存した機構は，構成ノードが頻繁に入れ替わるような動的な MANET にはあまり適していない．しかし，特定のノード間で明示的に構築されるような非常に静的な MANET には適しているだろう。

個々のノードが自律的に自分に名前を割り当てる手法は完全な分散型システムとして実現できる．これは，ネットワーク内の各ノードが独自に任意の名前を割り当て，それをネットワークに通知することで実現される．ANARCH では，名前割り当て手法としてこの方法を採用した．自律割り当てモデルを簡易に示すと図 3.2 のようになる．この図では，自分に割り当てた名前を 1 ホップ以内で届く範囲のノードに通知する方法を取っている．しかし自律割り当てを行う場合，名前解決の負荷を最小限に抑えるよう名前情報交換の手法や衝突解決アルゴリズムを設計する必要がある．これが不適

切であるとネットワークに過大な負荷をかけてしまう．また名前が階層化されている場合に，自律割り当て手法を取るとネットワークにかかる負荷が高くなるため，この手法はフラットな構造を持った単純な名前を用いている場合に適している．表 3.3 に示す疑似コードに，自律名前割り当てモデルを略述する．この疑似コードは自ノードへの名前割り当て，ネットワークへの名前の広告，名前衝突時の解決を含んでいる．

自律割り当て手法では中央管理システムが全く存在しないため，ネットワーク内の各ノードは，自分自身の名前情報を他に通知すると共に，他のノードの名前情報を収集してそのデータを維持管理する必要が生じる．また，重要な問題としてネットワーク内での名前の衝突が挙げられる．ANARCH では，各ノードにネットワーク内の名前情報を監視する機能を持たせることによりこの問題を解消した．その点については第 4 章で詳述する．

3.3.3 名前管理手法

名前管理手法とは，どこで IP アドレスと名前の対応情報を管理するかということであり以下の二つの方法が考えられる．

- 中央サーバで管理
- 各ノードで管理

管理には，名前と IP アドレスの対応情報の保存，更新作業が含まれる．この保存先となるノードに，名前情報を要求する側は問い合わせることになる．これも，小規模なネットワークであれば各ノードがネットワーク上の他のノードの名前情報を管理することができる．しかしネットワークの規模が大きくなるとこの方法で常に最新の情報を管理することは非現実的になる．そのため，ネットワーク内に名前情報のデータベースを持つことが考えられる． DNS では，ホスト名と IP アドレスの対応情報を分散データベースとして管理しており，そのデータベース自体も構造的に管理されている．

一般的に，名前の管理方法は名前の割り当て手法がどのようなものかに大きく依存する．ANARCH では名前割り当て手法として完全分散型の自律割り当て手法を採用したため，名前の管理も個々のノード内で行われる．

中央割り当てモデル

```
<CLIENT>
query_name()  : 名前の利用をサーバに要求する
assign_name() : 自ノードに名前を付ける
IP address    : 現在の自ノードの IP アドレス

if(start_ANARCH_first_time == true){
   name = query_name(IP address);
   assign_name(name);
}

<SERVER>
query_receive() : 名前の利用を求める要求を受信する
generate_name() : ネットワーク内で一意な新たな名前を生成する
give_name()     : 要求してきたノードに新たな名前を割り当てる
registor_name() : 新たに利用される名前を登録する
alive_confirm() : クライアントの生存を確認する
T               : 生存確認を行う間隔
i=0             : タイム・カウンタ
IP address      : クライアントの IP アドレス

while(1){

   i++;

   if(query_receive() == true){
      name = generate_name();
      give_name(name, IP address);
      registor_name(name, IP address);
   }

   if(i > T){
      alive_confirmation(IP address);
      i = 0;
   }
}
```

表 3.2: 中央割り当てモデル疑似コード

自律割り当てモデル

```
<ALL NODES>
name_oneself() : ユーザの好みに基づいて自ノードに名前を付ける
send_message() : HELLO / CONFLICT メッセージの送信
HELLO          : 自ノードの名前を広告するメッセージ
CONFLICT       : ネットワーク内の名前の衝突を警告するメッセージ
msg_receive()  : メッセージを受信をする
uniqueness()   : 新たに受信した名前の一意性を確認する
registor_name(): 新たに割り当てられた名前を登録する
alive_confirm(): 登録されたノードの生存確認
T              : 生存確認を行う間隔
i=0            : タイム・カウンタ
IP address     : 他のノードの IP アドレス

while(1){

   i++;

   if(start_ANARCH_first_time == true){
      myname = name_oneself();
      send_message(HELLO);
   }

   if(msg_receive() == true){
      case(HELLO):
         if(uniqueness() == true)
            registor_name(name, IP address);
         if(uniqueness() == false)
            send_message(CONFLICT);

      case(CONFLICT):
         myname = name_oneself();
         send_message(HELLO);
   }

   if(i > T){
      alive_confirm(IP address);
      i = 0;
   }
}
```

表 3.3: 自律割り当てモデル疑似コード

第4章　ANARCH の設計

本章では，ANARCH の設計方針と概要，ANARCH が想定する MANET 環境を述べ，その後 ANARCH の二つのモジュールについて詳述する．

4.1　設計方針

本節では，システム利用時の利便性と名前解決の基盤技術としての汎用性，動的適応性，マルチホップへの適応性，名前空間に関してそれぞれ ANARCH の設計方針を述べる．具体的内容については後の節で詳述する．

4.1.1　システム利用時の利便性と汎用性

MANET を用いる場合，他のネットワークを利用する場合と同じように通信先を指定する必要がある．その際，ユーザの利便性を向上させるために IP アドレスに関連付けられた名前の利用を可能にすることには意義がある．

インターネットでは名前とは全世界で一意なホスト名であり，それによって指定する相手は特定のノードである．通常，特定のサービス，例えば www などを利用するために特定のノードを指定するが，そのノードと提供しているサービスの関連付けは名前解決機構の関与するところではない．ユーザは何か別の方法であるサービスを提供しているノードを検索し，そのノードに対して通信を確立するために名前を利用する．

しかし MANET が利用され得る環境を考慮すると，インターネットにおけるホスト名と同じ名前を提供することのみがネットワークの利便性を必ずしも向上させるわけではない．例えば MANET を用いたセンサネットワークでは，ユーザの興味は特定のノードではなく，特定の場所に存在しているなどの条件を満たしたノードである場合がある．また，ユーザの興味があるサービスを提供しているノードなら相手を問わない場合もある．さらに，その目的によっては名前がネットワーク内で一意である必要がない場合もある．つまり MANET を用いる目的によって，ネットワークの利便性を最も向上させる名前の種類も異なるのである．

ANARCH は，ユーザの好みに基づいたホスト名によるノードの特定を可能にすること，またその名前をネットワーク内で共有することによってネットワークの利便性を向上させることを目的とし設計を行った．同時に名前解決の基盤技術となるよう汎用性を持たせるために，幾つかのパラメータのチューニングや他のモジュールとの併用によって，様々な特徴を持つ MANET や他の異なる名前のセマンティックスを持つ

場合においても利用できるよう設計した.

4.1.2 動的適応性

ANARCH は MANET の様々な動的特徴に対応する．MANET では既存の有線ネットワークと異なり以下の要素がノードの移動に伴って頻繁に変化する.

- ネットワークトポロジ・通信経路
- 通信リンク
- ノードの IP アドレス
- ネットワーク規模

このため中央管理機構に依存することが困難であり，名前解決も動的，分散的に行う必要がある．ANARCH では，以下の手法により上に列挙した変化へそれぞれ動的に対応する.

- 定期的な名前情報のブロードキャスト，既知のノードとのユニキャストでの通信の維持

  初期状態では，定期的なブロードキャストを用いて周囲のノードを検出し，その後は既知のノードとユニキャストで通信を継続する．ブロードキャストによる検出も，間隔を徐々に広げて続けられる.

- タイムスタンプによる周囲のノード情報の管理

  周囲のノード情報を収集する際，タイムスタンプを付けてその情報を管理する．その後，直接通信できる範囲から外れてもユニキャストを用いて情報交換を続けようと試みるが，一定の時間が経過した後も通信が不可能である場合には，ネットワークから退出したと想定してそのノード情報を削除する.

- ノードの MAC アドレスの利用

  IP アドレスが一定でないため，ノードに固有な一意の識別子として MAC アドレスを利用する.

- 1 ホップ内でのみの情報交換，自ノードが名前解決を行う範囲の明示的な指定

  名前情報の交換は直接通信できるノード同士でのみ行われる．MANET 全体，あるいは部分へのフラッディングは一切行わない．直接通信できないノードの情報が必要な場合には，直接通信できるノードにそのノードが保持している名前情報テーブルの提供を要求する．また各ノードは必要に応じて名前解決を行う範囲を決定する．直接通信できる範囲のみで名前解決を行うノードもあれば，2 ホップまで，3 ホップまでと範囲を広げて名前解決を行うノードも存在することになる.

4.1.3　マルチホップ通信への適応性

ANARCH は，MANET のマルチホップ通信機能に対応する．マルチホップ通信機能とは第 2 章で述べたように，無線通信範囲外にいる端末同士がお互いの通信範囲内に存在する別の中間端末を経由することで実現される通信である．ANARCH ではネットワークをシングルホップ範囲とマルチホップ範囲の部分に分け，シングルホップで通信可能な範囲をコア・リージョン，マルチホップ通信機能を用いて通信を行う範囲を X リージョン（未解決エリア）と呼ぶ．

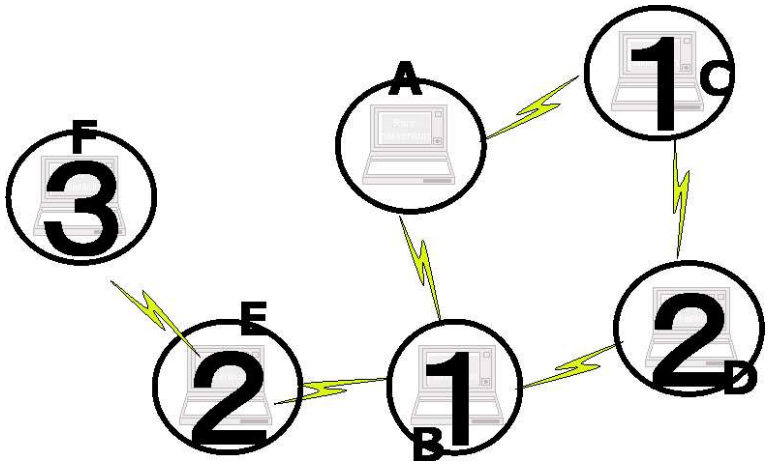

図 4.1: MANET のマルチホップ通信

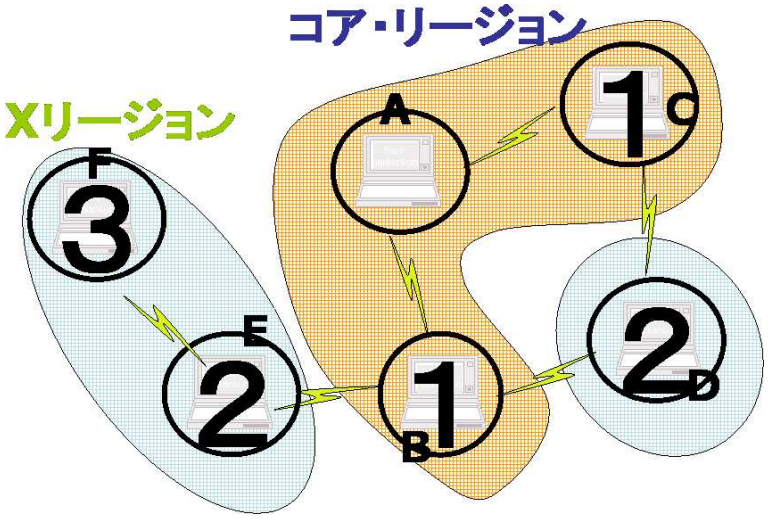

図 4.2: ANARCH のネットワーク分割

図 4.1，図 4.2 にこの概念を示す．図 4.1 には，6 台のノードがマルチホップ通信機能を用いて MANET を構成している様子が図示されている．ノード A から見た場合，

1 ホップで直接通信できるノードはノード B・ノード C であり，ノード D・ノード E はそれぞれノード C・ノード B を中継して 2 ホップで通信が可能である．またノード E とはノード B・ノード E を中継して 3 ホップで通信を行う．このネットワークにおいて，図 4.2 に示すように 1 ホップで直接通信できる範囲をコア・リージョン，マルチホップ通信機能を用いて通信できる範囲を X リージョンと呼ぶ．また，X リージョン内で n ホップで通信できる範囲を Xn リージョンと呼ぶ．このトポロジでは，ノード B・ノード C がコア・リージョンに，またノード D・ノード E が X_2 リージョンに，ノード F が X_3 リージョンに存在する．表 4.1 にこれら ANARCH のネットワーク分割の定義をまとめる．

表 4.1: ANARCH のネットワーク分割

定義	範囲
コア・リージョン	1 ホップで直接通信可能な範囲のみ
X リージョン	マルチホップで通信可能な範囲すべて
Xn リージョン	マルチホップを利用して、n ホップで通信可能な範囲 注）X_1 リージョン = コア・リージョン

基本的動作はすべてコア・リージョン内で行われる．そして，ユーザの要求に基づいて必要な場合のみ X リージョン内での動作に移る．ANARCH は，大規模な MANET にも対応するため，明示的に自ノードが扱うホップ範囲を定める．最も身近なノードの中で必要な相手を探すことを前提としており，その範囲内で必要な相手またはサービスを検出できなかった場合にのみ範囲を広げて名前解決を行う．これにより，規模の大きなマルチホップネットワークにおいてもノードの負荷と利便性の調整を可能にする．

4.1.4 名前空間

ANARCH では，名前解決によるオーバヘッドを最小限にするためフラットな名前空間を利用する．階層化された名前を用いるには MANET 内で共有される何らかの規則，または中央管理システムが必要になるが，それを動的な MANET で実現するにはコストがかかるためである．

4.2 ANARCH の概要

本節では ANARCH の概要として想定環境と，シングルホップネットワーク・マルチホップネットワークにおける ANARCH の基本動作，システムの全体構成を述べる．

4.2.1 想定環境

ANARCH は動的に構築される MANET の中でも，接続端末数が 100 台程度の比較的小規模なネットワークを想定している．具体例としては，会議などで持ち寄った端末同士で組むネットワークやバスネットワークなどが考えられる．これらのローカルネットワークでは，ノードとユーザが密接に関連しているため名前解決を行うことが有効であると考えられる．ANARCH はマルチホップ通信も想定している．

実際に名前解決機構を利用する場合，名前は特定のサービスやユーザと関係付けられていることが望ましい．ANARCH は MANET 内で一意な名前を提供することに焦点を当てているが，サービス名，ユーザ名などの付加的な情報の利用を支援する他のモジュールと共に用い，利便性を向上させることができる．また IP アドレスは，zeroconf [27] などの何らかの手法で既に一意に割り当てられていることを想定している．

4.2.2 基本動作

ANARCH は既存の DNS が有線インターネットにおいて実現しているような名前解決サービスを，MANET において提供する．ANARCH は次の機能を持つ．

1. 自ノードへの名前割り当て機能
2. 他ノードの名前情報管理機能
3. ネットワーク内の名前衝突解決機能
4. 既存のアプリケーションによる名前利用の提供機能

自ノードへの名前割り当て機能により，ユーザは自分の好みに基づいて自ノードに名前を付け，他ノードの名前情報管理機能により，他のノードから通知される名前情報を収集して管理する．その際，ネットワーク内の名前衝突解決機能によって名前が MANET 内で一意になるように調整される．

ANARCH では基本的に名前情報の交換はコア・リージョン（シングルホップで通信可能な範囲）内でのみ行われる．これは，その時利用する MANET の規模に応じて名前解決のために要する負荷の調整を可能にするためである．例えば会議室での一時的な利用など，シングルホップで通信が完了できるような MANET を利用する場合には ANARCH の基本動作のみで名前解決が実現される．また交通ネットワークなどのさらに規模の大きな MANET や，端末が移動することによって直接通信することが不可能になる可能性がある状況で MANET を利用する場合には，コア・リージョン内で行われる ANARCH の基本動作に加え，X リージョン（マルチホップで通信可能なエリア）に存在するノードの名前解決を要求する．その際，必要に応じて名前解決を行う範囲をホップ数で指定することが可能であり，これにより不必要な名前解決に因る負荷を避けることができる．なお，ANARCH によって，MANET 内で重複しない限りユーザの好みに基づいた一意な名前が提供されるが，この名前の一意性はシングルホップ内でのみ保証される．

4.2.3 全体構成

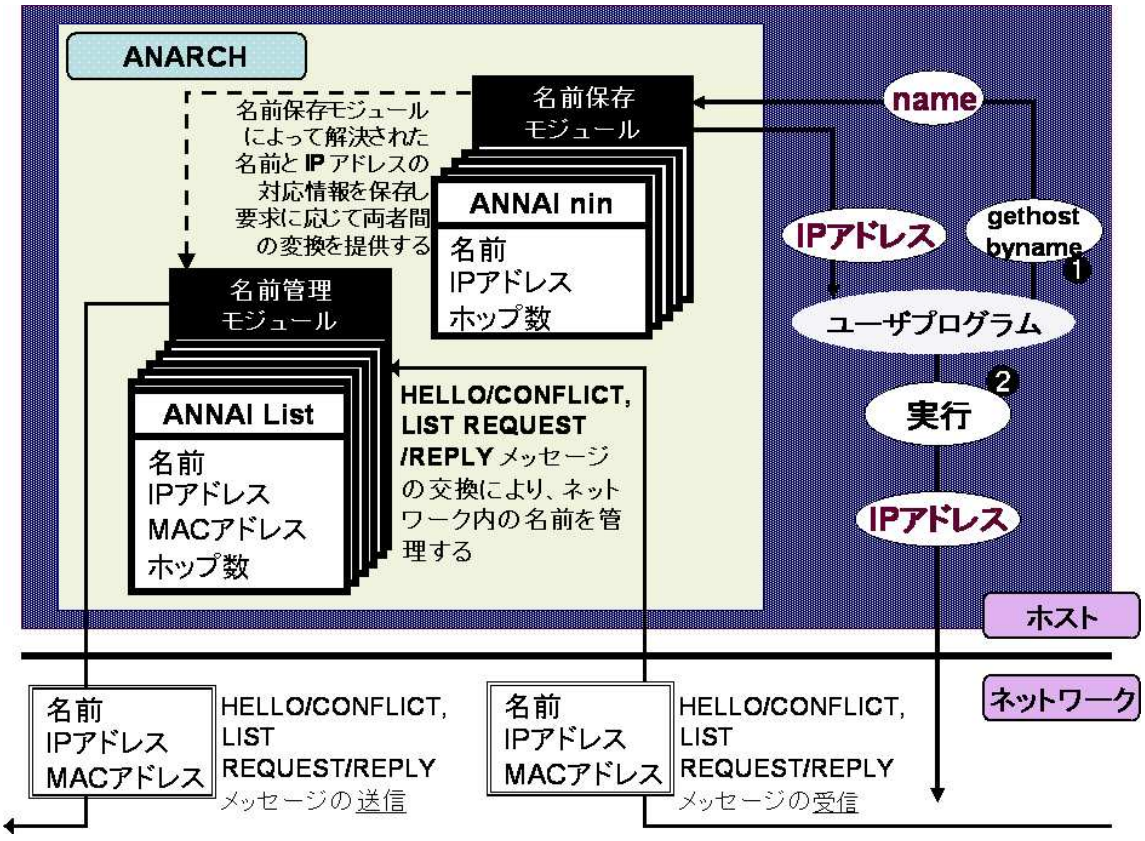

図 4.3: ANARCH のシステム概観

ANARCH では，システムに汎用性を持たせるため，これらの機能を二つのモジュールによって実現した．一つは，自ノードへの名前割り当て機能，他ノードの名前情報管理機能，ネットワーク内の名前衝突解決機能を提供する名前管理モジュールで，もう一つは既存のアプリケーションによる名前利用の提供機能を実現する名前保存モジュールである．この二つのモジュールの関係を図 4.3 に示す．

この図は，自ノードへの名前割り当て機能，他ノードの名前情報管理機能，ネットワーク内の名前の衝突解決機能を実現するため，各ノードの名前管理モジュール同士が通信を行って，ネットワーク内の名前を管理していることを示している．その際，HELLO, CONFLICT, LIST REQUEST, LIST REPLY メッセージと呼ばれる四つのコントロールメッセージを用いることが示されている．これらのメッセージについては後ほど詳述する．また一ノード内では，名前保存モジュールが名前管理モジュールによって解決されたネットワーク内の名前と IP アドレスの対応情報の複製を保存しており，既存のユーザプログラムに対して名前解決サービスを提供することを示している．次節ではこれらの各モジュールについて詳述する．

4.3 モジュールの設計

4.3.1 名前管理モジュール

名前管理モジュールは，前述した機能 1 から 3 を以下の四種類のメッセージを交換することで実現する．これらのコントロールメッセージはすべてコア・リージョンに対してのみ送信される．

- HELLO メッセージ

  ノードがユーザの好みに基づいて自身に割り当てた自分の名前を通知する．システムが定期的に送信する．コア・リージョンに対してブロードキャストされる．

- CONFLICT メッセージ

  名前の衝突を検知したときにそれを通知する．システムが自動的に送信する．特定のノードに対してのみユニキャストされる．

- LIST REQUEST メッセージ

  X リージョン内のノードの名前情報を要求する．何ホップのノードまでの情報が必要かを要求に含める．ユーザが要求した場合のみ送信される．コア・リージョンに対してブロードキャストされる．

- LIST REPLY メッセージ

  LIST REQUEST メッセージに対する返信として，自ノードが保持する，要求されたホップ数内のノード情報をすべて返す．システムが自動的に送信する．要求を出したノードに対してのみユニキャストされる．

表 4.2: コントロールメッセージ

	HELLO	**CONFLICT**	**LIST REQUEST**	**LIST REPLY**
目的	名前の通知	名前衝突の通知	名前情報の要求	名前情報の提供
通知手法	ブロードキャスト	ユニキャスト	ブロードキャスト	ユニキャスト
名前を解決する範囲	コア・リージョン	コア・リージョン	X・リージョン	X・リージョン
送信トリガ	システムが自動的に送信	システムが自動的に送信	ユーザの要求に基づいて送信	システムが自動的に送信

これらのメッセージが送信される範囲，名前を解決する範囲と送信されるトリガを表 4.2 にまとめる．これらのメッセージは，次の 5 つのフィールドを持つ．

- message-type
- name
- IP address
- MAC address
- Option

4 つのメッセージそれぞれの，これらフィールドに含まれる情報を図 4.4，図 4.5 に示す．図 4.4 は HELLO・CONFLICT メッセージ，図 4.5 は LIST REQUEST・LIST REPLY メッセージの内容を示している．続いて，これらのメッセージ交換により実現される前述の 4 つの機能について述べる．本小節で詳述している名前管理モジュールによっては機能は 1 から 3 が，次次小節で詳述する名前保存モジュールによっては機能 4 が提供される．

HELLOメッセージ

message-type	name	IP address	MAC address	Option
[HELLO]	自身に割り当てた名前	自身の現在の IP アドレス	自身の MAC アドレス	Null

CONFLICTメッセージ

message-type	name	IP address	MAC address	Option
[CONFLICT]	重複していると検出された名前	重複していると検出されたノードの IP アドレス	重複していると検出されたノードの MAC アドレス	Null

図 4.4: HELLO / CONFLICT メッセージ

自ノードへの名前割り当て機能

ANARCH では名前の割り当て手法として「自律割り当て方式」を採用した．ネットワーク内の全てのノードは，独自に任意な名前を付けることができる．自ノードへの名前割り当てを行う際， ANARCH はフラットな構造のデータを名前として用いる．これは，接続端末数のそれほど多くない小規模なネットワークにおいて最も割り当て・

LIST REQUEST メッセージ

message-type	name	IP address	MAC address	Option
[LIST_ REQUEST]	自身に割り 当てた名前	自身の現在 の IP アドレス	自身の MAC アドレス	Null

LIST REPLYメッセージ

message-type	name	IP address	MAC address	Option
[LIST_REPLY]	要求してきた ノードの名前	要求してきた ノードの IP アドレス	要求してきた ノードの MAC アドレス	要求された ホップ範囲

図 4.5: LIST REQUEST / LIST REPLY メッセージ

管理コストのかからない名前である．この名前はユーザの好みで指定されるもので，システムが自動的に生成することはない．また名前自体が位置情報などの特定の情報を含むこともない．なお，この名前はネットワーク内で衝突が起きた場合にのみシステムから変更を求められ，ユーザに対して新しい名前を付けるよう要求する．

他ノードの名前情報管理機能

まず，各ノードはユーザから指定された名前を HELLO メッセージを用いてネットワークに通知し，同時に他のノードから通知される HELLO メッセージの収集を行う．各ノードがこの動作を繰り返すことで，ノードの移動によりトポロジが激しく変動している場合でもネットワーク内の他のノードの名前情報を徐々に集めていくことができる．ANARCH では，収集したネットワーク内の名前情報を ANNAI List (ANARCH Name Information List) と呼ばれるリストに登録する．記述する内容は，図 4.3 にも一部略述されているように以下の 5 つである．

- 名前（ホスト名）
- IP アドレス
- MAC アドレス
- ホップ数
- タイムスタンプ

この動作手順を図 4.6 に示す．なお，図 4.6 と図 4.13 の状態遷移図で用いられている各状態の定義は図 4.7 に示されている通りである．矢印の横には状態が遷移する条件

が書かれており Mtype は メッセージタイプを表す．また網掛けされた文字は引数として与えられるメッセージの種類を表しており，C は CONFLICT メッセージを，H は HELLO メッセージを示している．

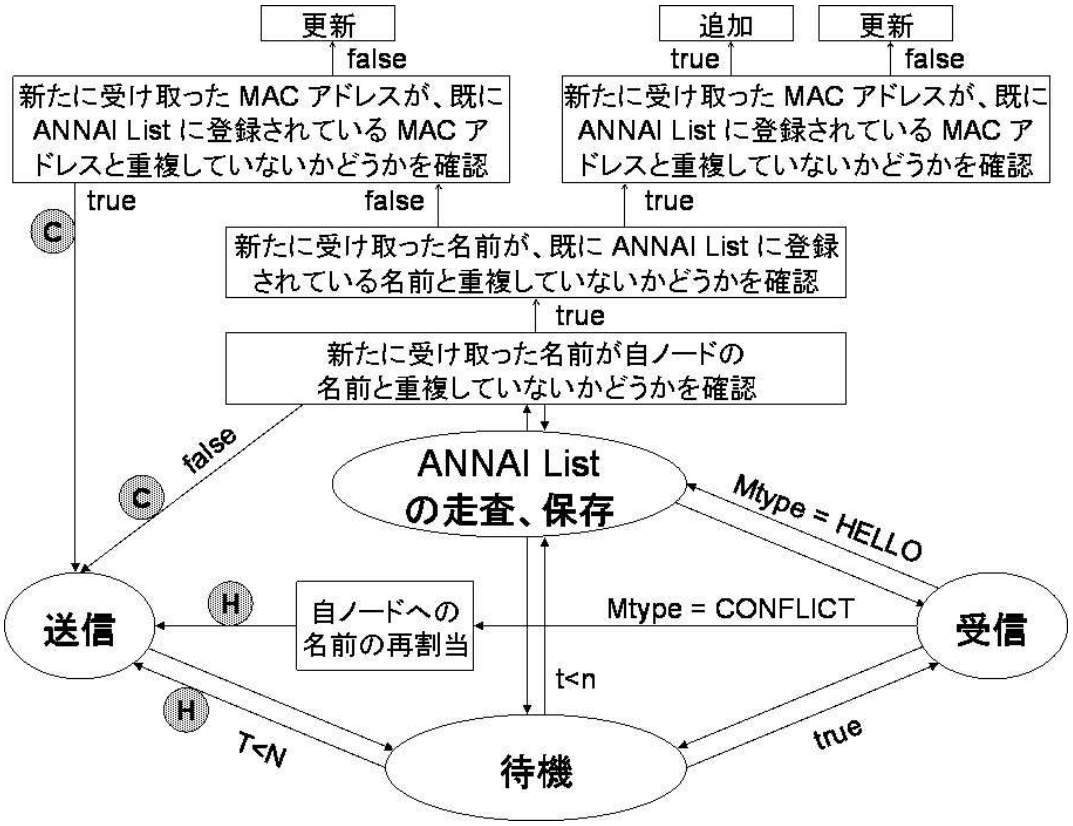

図 4.6: 名前管理モジュール

名前解決は，基本的にはコア・リージョン内でのみ行われ，要求があった場合のみ X リージョン内へ広げられる．しかしいずれの場合も，HELLO メッセージはコア・リージョンに対してのみ送信され，他のノードへの転送は行われない．名前通知を行う際のネットワークへの負荷を軽減するため，HELLO メッセージの送信は以下の手順で行われる．まず初期状態の各ノードはブロードキャストを用いてコア・リージョンに存在するすべてのノードに HELLO メッセージを送信する．そして一定の期間が経過したらブロードキャストを行う時間間隔を大きくして，代わりに自ノードの ANNAI リストに登録されたノードにユニキャストで HELLO メッセージを送信する．これにより，ブロードキャストによるネットワークへの負荷を最小限に押さえ，同時に一度 ANNAI List に登録されたノードはコア・リージョンから外れても通信を維持することが可能になる．これは，既知のノードが無線で直接通信できる範囲外に移動しても，IP アドレスが分かっていればマルチホップ通信機能を利用して通信を維持することができるためである．その際，ANARCH は既知の IP アドレスに対して通信を確立しようと試みるだけで，ルーティングプロトコルが目的の IP アドレスへの経路を検出す

待機
パケットの受信を待つ．受信すると true を返す．また n, N, t, T の四つのカウンタの管理を行う．

(n) リスト中のエントリのタイムスタンプを走査するための時間カウンタ

(N) HELLO メッセージを送信するための時間カウンタ

(t) リスト中のエントリが古くなっていないか走査するための間隔

(T) HELLO メッセージを送信する間隔

(p) 管理モジュールにポーリングするための時間カウンタ

(pt) 管理モジュールにポーリングする間隔

送信
与えられたメッセージタイプのパケットを送信する．

受信
パケットのメッセージタイプによって処理を分ける．

リストの走査（図 4.6、5.3）
自ノードが保持するリストのエントリを走査する．

- 名前の重複確認
  重複が検知された場合は false を返す．検知されなかった場合は MAC アドレスの重複確認に進む．
- MAC アドレスの重複確認
  重複が検知された場合は false を返す．
- エントリの追加
- エントリの更新

リストの走査（図 4.13）
自ノードが保持する ANNAI nin エントリのタイムスタンプを走査する。

自ノードへの名前の割当
ユーザの好みの名前を自ノードに割り当てる．

終了処理
カーネルの処理に戻りユーザ空間にパケットを渡す．

図 4.7: モジュールの状態遷移図 4.6、4.13、5.3 の各要素

る．ANARCH はルーティングプロトコルとは独立であるので，MANET を構築するために用いるルーティングプロトコルには依存しない．

またこのモジュールでは，リスト中の各エントリにタイムスタンプを付加してそのエントリを管理している．長期間 HELLO メッセージを受信しなくなったノードはネットワークから退出したと判断される．ANARCH では，名前と現在用いている IP アドレスに加え，MAC アドレスをネットワークに通知する．そして MAC アドレスを指標にノードを識別している．これは，MANET の特性上，IP アドレスも名前もノードへの一意性が保証されていないため，同一ノードを識別するのにノードに固定された別の識別子を必要とするからである．MAC アドレスは衝突がないようハードウェア的に割り当てられたものである．MANET の場合無線を用いているので，無線 LAN インタフェースの MAC アドレスがこれに当たる．しかし MAC アドレスを用いることも，文献 [14] で指摘されているように完全な方法ではない．なぜなら，MANET のノードは必ずしも 48bit の IEEE から割り当てられた一意な MAC アドレスを持ったネットワークインタフェースカードを使用しているとは限らないためである．またインタフェースカードの MAC アドレスをプログラムから書き換えることが可能であることも指摘されている．

ユーザが X リージョンの名前解決を要求する場合は次の手順で名前解決がなされる．まずユーザは何ホップ内のノードに関して名前解決を行うかを指定する．そして，コア・リージョン内のノードに LIST RIQUEST メッセージを用いて ANNAI List 要求を出す．そして要求を受信したコア・リージョン内のノードは，自ノードが保持する ANNAI List の中から要求に合ったホップ範囲に存在するノードについての情報を LIST REPLY メッセージとして返信する．この際，LIST REQUEST メッセージはコア・リージョン内にのみブロードキャストで送信され，LIST REPLY メッセージは要求を出したノードにユニキャストで返信される。ANARCH がユニキャストでメッセージを送信する場合は，利用するルーティングプロトコルの経路探索機能によって経路を発見するが，ANARCH としてはコントロールメッセージの転送，フラッディングは一切行わない．

図 4.8，4.9，4.10 に，ノードが LIST REQUEST メッセージを用いて X リージョンで名前解決を行う様子を示す．A から I までの黒丸はノードを表し，直接通信できるノード同士は実線で結ばれている．図 4.8 は，初期状態のノードがコア・リージョン内のノードの名前情報を収集した様子を示している．図 4.9 は各ノードが LIST REQUEST メッセージを用いて可能な限り多くの名前情報を収集した状態である．図 4.10 は各ノードが再び LSIT REQUEST メッセージを用いて名前情報を収集した様子である．このように，コア・リージョン内でのメッセージ交換のみでフラッディングを行わなくても徐々に広範囲の名前情報を収集することが可能になる．ANARCH での X リージョンにおける名前収集手法では，コア・リージョンに存在するノードが管理している情報量やノード同士を検出するタイミングなどの要素により検出に要する時間や検出できるノード数が異なる．また，ノードの頻繁な移動によって検出できるノード数が増加する．

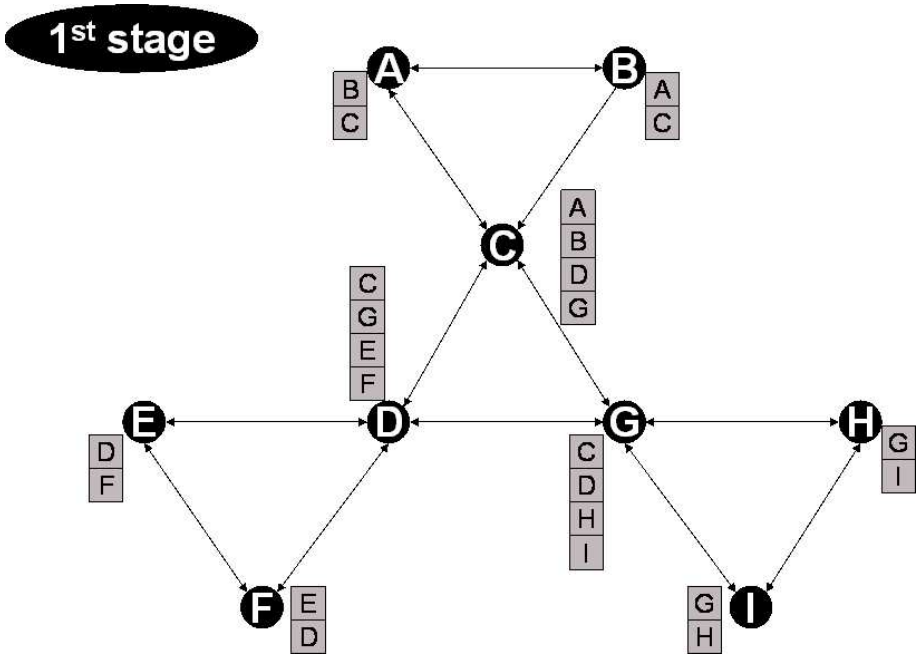

図 4.8: 初期状態：コア・リージョンの名前情報収集

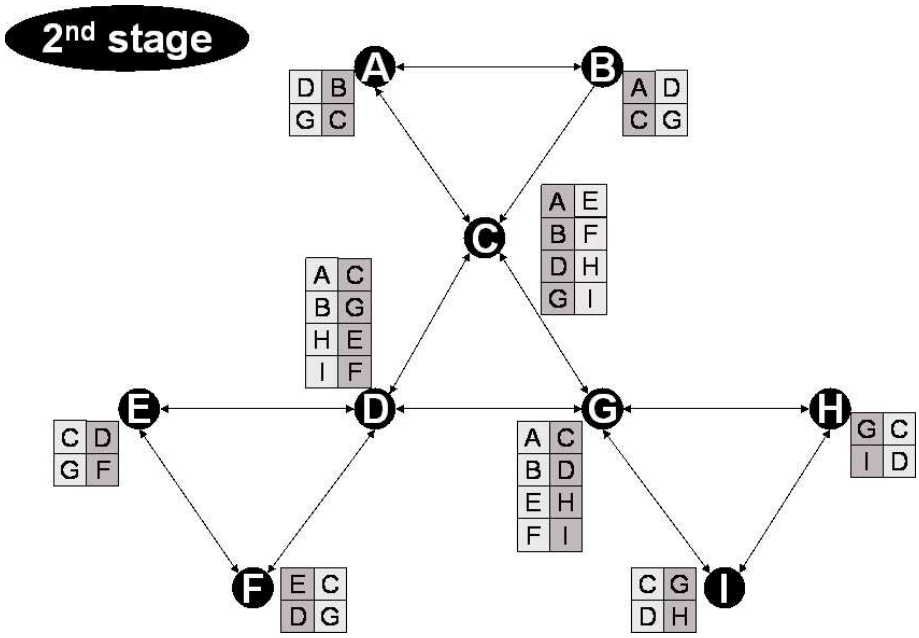

図 4.9: LIST REQUEST メッセージを用いた名前情報収集：1 回目

ネットワーク内の名前衝突解決機能

ネットワーク内で名前の衝突が起きた場合，それを検知した他のノードから CONFLICT メッセージが送信される．衝突した名前を受け取ったノードは，そのエントリ

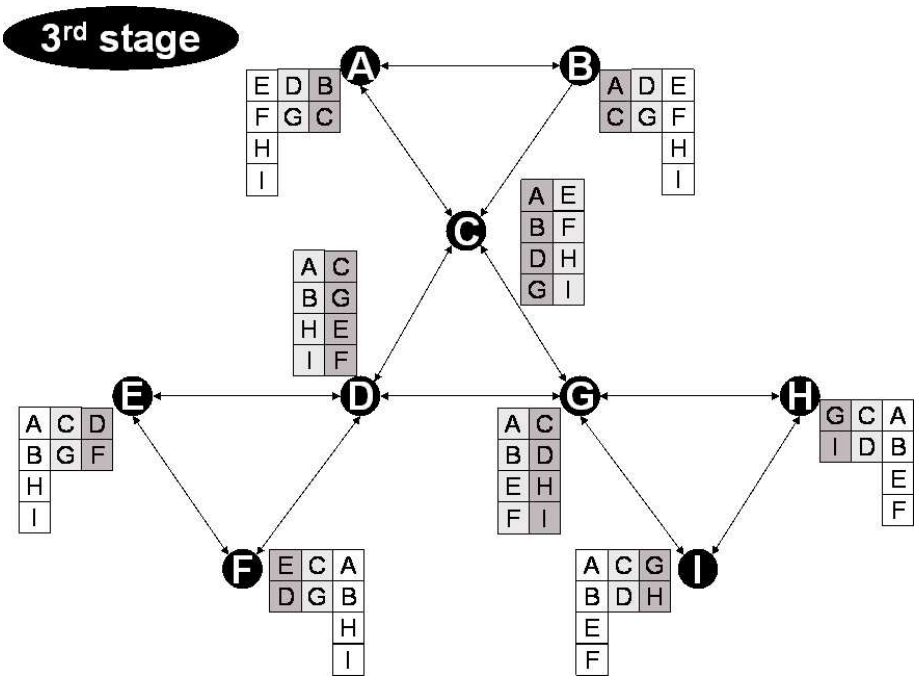

図 4.10: LIST REQUEST メッセージを用いた名前情報収集：2 回目

のリストへの登録を避けることにより自ノード内での名前の一貫性を保つ．ANARCH プロトコルでは，CONFLICT メッセージを受信した場合必ず名前を変更しなければならない．図 4.6 にも示されている通り，CONFLICT メッセージは次の二つの場合に送信される．

- 自ノードと重複した名前を通知する HELLO メッセージを受信した場合
- 自ノードが保持する ANNAI List 中のエントリと重複した名前を通知する HELLO メッセージを受信した場合

しかしネットワーク内で名前が衝突した場合，どちらの名前を優先させるかが問題となる．先にその名前を使っていたノードを優先させることが妥当であると考えられるが，MANET においてそれを正確に特定するのは難しい．そこで ANARCH では CONFLICT メッセージを受け取る量を調整することでこの問題を解決した．これは，ネットワーク内での認知度を指標にしてどちらが先に名前を用いていたかを決め，先に使用していたノードがその名前を引き続き使用できる確率を高める手法である．具体的には，各ノードが自分の名前と衝突した場合に加えて，自分の保持する ANNAI List 内の名前と衝突した場合に CONFLICT メッセージを送る．これにより新しく検知されたノードはそのネットワークでの認知度に応じた量の CONFLICT メッセージの量を受け取ることになる．すなわち，もしあるノードと名前が衝突した場合，そのネットワーク内に自分を既に認識しているノードの数が，相手を既に認識しているノー

ドの数より少なければ，自分の方が CONFLICT メッセージを相手より多く受け取ることになり，自分の方が名前を変える確率が高くなる．

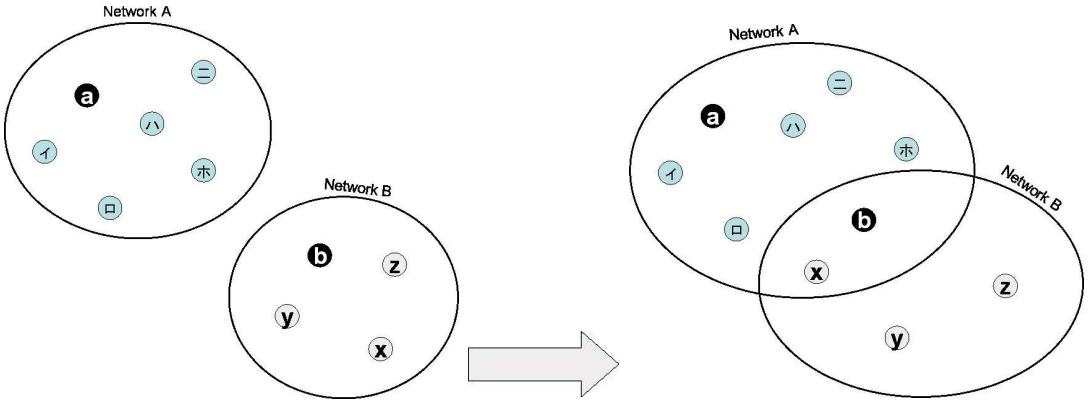

図 4.11: 複数のネットワークが合併した場合の名前衝突の解決：合併前

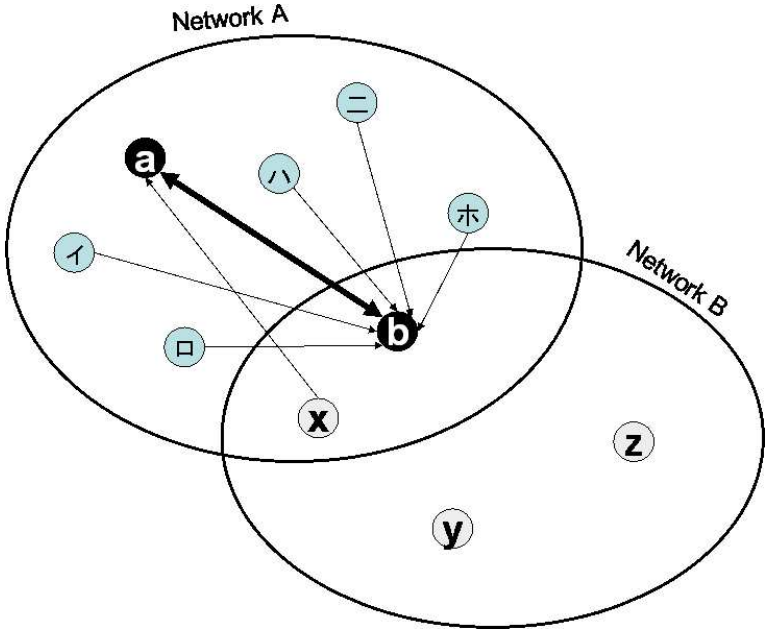

図 4.12: 複数のネットワークが合併した場合の名前衝突の解決：合併後

図 4.11，図 4.12 に，二つの MANET が合併した場合の名前衝突の解決を示す．図 4.11 は，二つの MANET が存在している状態である．ネットワーク A は 6 台，ネットワーク B は 4 台のノードで構成されている．また，ネットワーク A・B に存在するノード a・b が同じ名前を利用している．そして，図 4.12 はノードの移動に伴ってネットワーク・トポロジが変化し二つの MANET が合併した様子を示している．

この際ノード a・b の名前が衝突していることが検知され，図 4.12 に示すように CONFLICT メッセージが流れる．まず，ノード a・b は互いに CONFLICT メッセー

ジを送り合う．また，もともとネットワーク A に存在していたノード（イ）からノード（ロ）は既にノード a を認識しているため，ノード b に対して CONFLICT メッセージを送信する．同様に，もともとネットワーク B に存在していたノード x は既にノード b を認識しているため，ノード a に対して CONFLICT メッセージを送信する．すると，ノード a が受信する CONFLICT メッセージ数は 2（1+1），ノード b が受信する CONFLICT メッセージ数は 6（1+5）となる．それゆえ，名前が衝突したノード間では，もともと小さなネットワークに存在していたノード b の方が CONFLICT メッセージを多く受信するため名前を変更する確率が高くなる．

しかしこの方法でも確率を調整できるだけで完全に正確に判断できるわけではない．無線の性質上，先に CONFLICT メッセージを受け取った方が名前を変えることになるからである．また，もし名前が解決される前に両方が CONFLICT メッセージを受信すれば，両方が名前を変えることもありうる．

4.3.2 名前保存モジュール

本小節では，名前保存モジュールによって実現される既存のアプリケーションによる名前利用提供機能について述べる．

既存のアプリケーションによる名前利用の提供機能

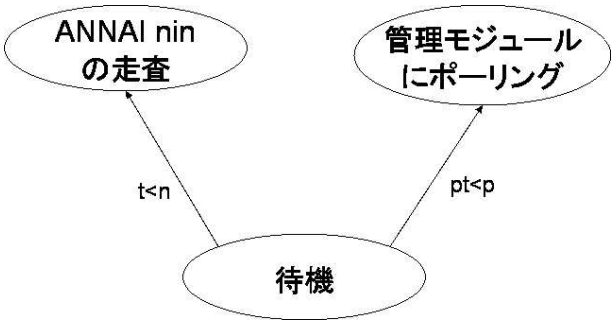

図 4.13: 名前保存モジュール

名前保存モジュールは，ユーザ空間の他のプログラムに対して DNS が提供しているような名前での通信を可能にする．図 4.3 に示したように，名前保存モジュールは名前管理モジュールに定期的にポーリングして解決が完了した名前の一覧を取得する．名前保存モジュールが保持する ANNAI nin (ANARCH Name Information guide) には，以下の項目が含まれる．

- 名前（ホスト名）
- IP アドレス

- ホップ数
- タイムスタンプ

このモジュール自体はネットワークに何かを通知したり，リストの管理を行うことはない．名前保存モジュールは図 4.13 に示されているように名前管理モジュールに定期的にポーリングを行い，名前管理モジュールが保存している ANNAI List を複製する．また，ANNAI nin 中のエントリのタイムスタンプを走査してデータを管理する．名前管理モジュールは，新しいメッセージを受信するたびにそのノードの情報を更新するため，他のノードの IP アドレスや名前の変化に対応する．

名前保存モジュールに求められる最も重要なことは，名前管理モジュールがネットワーク内で衝突した名前を解決している間もユーザプログラムに対して迅速な名前解決サービスを提供することである．この名前保存モジュールは主にユーザプログラムのためのものであり，ANARCH での名前解決には直接関わらないため，このモジュールが無くても ANARCH は動作する．

4.3.3 チューニング

ANARCH は MANET で名前解決を行うための基盤技術となるよう，汎用性がある設計になっている．ANARCH は次の要素に応じて最適化できる．

- ネットワークの特徴
- 利用したい名前の種類

まず，ネットワークの特徴によって最適な動作が異なるので，ANARCH のパラメータの調整によってチューニングを行う．ネットワークは，以下の要素によって特徴付けられる．

- 構成ノードの移動の頻繁性
- 構成ノードがネットワークにとどまる平均時間
- 名前解決のために払うことのできるコスト

なお，チューニングが可能な ANARCH のパラメータは以下の通りである．

- HELLO メッセージを送信する間隔

  これは HELLO メッセージをブロードキャストする間隔で，構成ノードの移動が頻繁に起きる動的な MANET では短くし，会議室などで比較的静的に構築される MANET では長くすることが適切である．

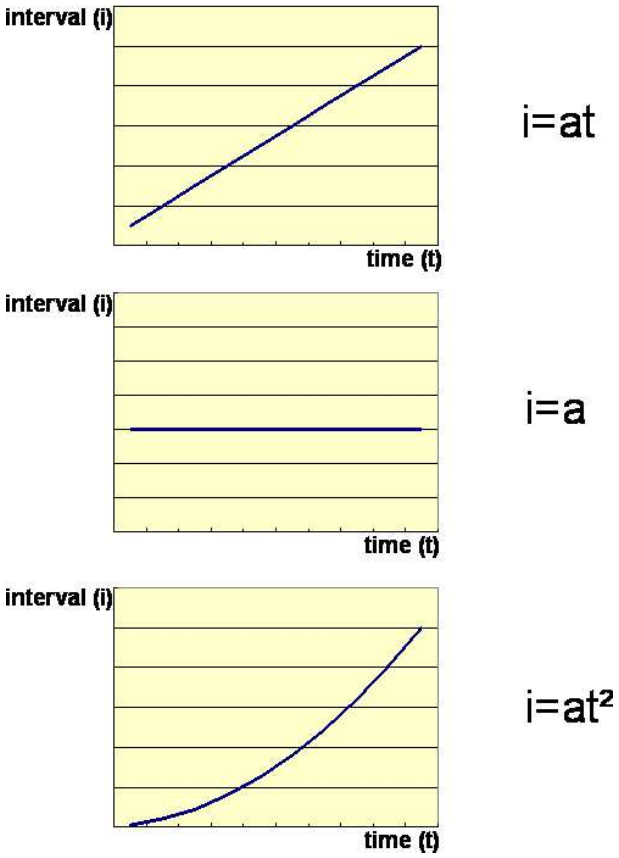

図 4.14: ANARCH のチューニング例

- HELLO メッセージを送信する間隔の広げ方

  ANARCH では，時間の経過に伴って HELLO メッセージをブロードキャストする間隔を変化させていく．変化のさせ方の例を図 4.14 に示す．構成ノードの移動が頻繁に起きる動的な MANET なら一定間隔で送信し続け，そうでない場合には徐々に間隔を広げていく．個々のノードが名前解決のために払うことのできるコスト量に応じて，オーバヘッドとの兼ね合いも考慮できる．

- エントリが有効である期間

  これは ANNAI List に登録したエントリが有効である期間である．ANARCH は周辺ノードを検出した際タイムスタンプを付けてそのエントリを管理する．そして HELLO メッセージを受信する度にタイムスタンプを更新する．動的な MANET の場合，とりわけ個々のノードがネットワークにとどまる時間が短いネットワークにおいてはこの間隔は短くてよい．

また，ANARCH は名前の拡張に応じてチューニングすることが可能である．ANARCH が提供する名前はフラットな構造のホスト名であるが，位置情報やサービス情報に関連付けられた名前など，拡張された名前のセマンティックスを用いる場合にはそ

れを実現する他のモジュールと共に用いることができる．また，名前拡張時の利用を意図した ANARCH コントロールメッセージの予約領域を利用することも可能である．

4.4　本章のまとめ

本章では，MANET に適した名前解決機構として ANARCH を設計した．ANARCH はユーザの好みに基づいた，IP アドレスに対応するホスト名を提供することでシステム利用における利便性を向上させる．ANARCH は MANET 特有の，ネットワーク・トポロジ，IP アドレス，名前などの動的な変動に対応する．

本システムは MANET 内でユーザの好みに基づいた一意なホスト名を割り当てることに焦点を当てているが，他のモジュールと併用することで，サービスや位置情報などと関連付けられた名前を利用することも可能である．また名前解決を行う範囲を指定できるため，大規模な MANET においても必要な範囲で名前解決を提供することが可能であり，各端末は名前解決による負荷を調整することができる．加えて，ANARCH のパラメータを調整することで様々な特性の MANET に適応させることができる．

MANET が用いられる用途は様々で，一律にネットワークの特徴を定義することは難しい．例えば会議室などで比較的静的に構築される MANET もあれば交通ネットワークなど非常に動的な性質のネットワークもある．また，ホスト名を提供することが有益な場合もあれば，サービス名など他の要素を名前として提供する方が有意義である場合もあることが予想される．ANARCH は単体でこれらの要求すべてに対応することを目的としたシステムではないが，どのような特性の MANET・名前を利用する際にも基盤として利用できる汎用的な機構として設計されている．

第5章　ANARCH の実装

本章では，まず実装環境と MANET の構築に用いた DSR の概要を述べ，次に ANARCH の実装について述べる．

5.1　実装環境

本研究では，ANARCH を FreeBSD3.3R [16] で動作する DSR [4] 上に C 言語で実装し評価した．また MANET を構築するルーティングプロトコルとして DSR を利用し実際に無線 MANET を構築した．無線通信媒体には 802.11b [18] 規格の MELCO BUFFALO 無線 LAN カード AIRCONECT WLI-PCM-L11 を使用した．

5.1.1　DSR の概要

DSR は CMU の MONARCH プロジェクト [24] において開発された MANET におけるルーティングプロトコルの一つであり，FreeBSD3.3R 上での実装が公開されている．DSR は送信パケットのヘッダに送信元から送信先までの経路を含め，経路発見を行ってデータの配信を可能にするソースルーティングを実現するプロトコルである．

5.2　実装の概要

本節では実装の概要について述べる．まずシステムアーキテクチャの全体像を示し，名前空間について述べる．また，付加的な機能である ANARCH アドレス帳機能について述べる．

5.2.1　システムアーキテクチャ

本研究では，ANARCH を名前管理モジュール・名前保存モジュールと呼ばれる二つのモジュールによって実現した．名前管理モジュールをユーザ空間に，名前保存モジュールをカーネル空間に実装した．図 5.1 に，両モジュールの関係を図示する．名前管理モジュールは，通常のユーザアプリケーションとして動作し，他のノードの名前管理モジュールと通信を行って名前解決を行う．一方名前保存モジュールはカーネル空間で動作しており，ANARCH パケットを受信した場合にアプリケーション層に上

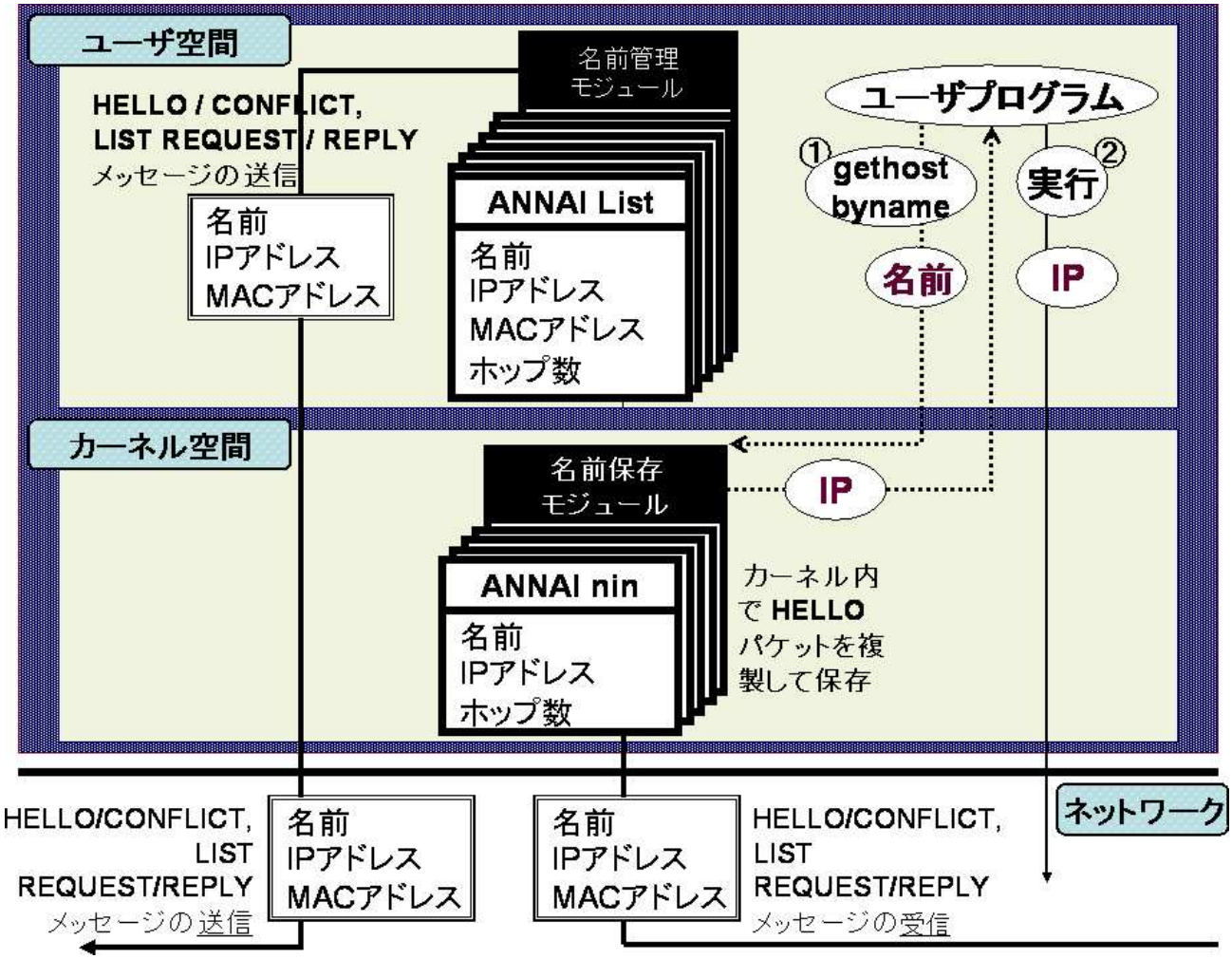

図 5.1: ANARCH システムアーキテクチャ

げる前に中身の複製を保存する．これは，既存のユーザプログラムから ANARCH で提供する名前の利用を可能にするためである．

5.2.2 名前空間

ANARCH ではフラットな名前を利用した．ユーザは 20 文字までの範囲で好みの名前を利用することができる．使用できる文字に特に制限はない．

5.2.3 ANARCH アドレス帳機能

ANARCH アドレス帳機能によって，MAC アドレスを指標として既知のノードを各ノード内に静的に登録しておくことができる．ANARCH アドレス帳は表 5.1 のように

記述される．ANARCH アドレス帳に登録された人物情報はユーザアプリケーションから利用できるホスト名ではないが，相手が既知のノードである場合，現在どのようなホスト名を利用していても目的のノードを識別することが容易になる．これは，名前の衝突により通常と異なるホスト名を用いている場合にも有益である．

表 5.1: ANARCH アドレス帳

人物情報	MAC アドレス
徳田英幸教授（慶應義塾大学）	00:02:2d:00:3f:0e
間博人氏（慶應義塾研究科）	00:02:2d:00:63:3d
斎藤匡人氏（慶応義塾研究科）	00:02:2d:0e:e7:c4
Charlie (CSL)	00:02:2d:00:65:10

ANARCH アドレス帳機能を利用した様子を図 5.2 に示す．通常はひとつのノードにつき，現在ネットワークに存在しているかどうかを示す状態 (Status) と現在使用しているホスト名 (Name)，IP アドレス (User information) が表示されるが，ANARCH アドレス帳に登録してあるエントリに関しては IP アドレスの代わりに登録情報 (User information) が表示される．つまり，未登録のノードに関しては IP アドレスが表示されることになる．

図 5.2: ANARCH アドレス帳機能の利用

5.3　モジュールの実装

本研究では名前管理モジュールをユーザ空間に，名前保存モジュールをカーネル空間に実装した．本節では，各モジュールの実装について詳述する．

5.3.1　名前管理モジュール

名前管理モジュールは，HELLO・CONFLICT・LIST REQUEST・LIST REPLY の 4 つのメッセージを利用して名前解決を行う．すべてのメッセージは UDP 通信を用いて送受信される．また，HELLO メッセージ・LIST REQUEST メッセージはコア・リージョンにのみブロードキャストで送信され，CONFLICT メッセージ・LIST REPLY メッセージは目的のノードに対してのみユニキャストで送信される．またすべてのメッセージの送信の際には，ランダムな長さの休止を送信の前に入れることにより，ノード間のデータ送信時期の同期を避けるよう工夫した．

名前管理モジュールは ANARCH メッセージを受信するとまず HELLO メッセージで広告されている名前が自ノードの名前と重複していないかどうか調べ，もし重複していれば CONFLICT メッセージをブロードキャストを用いて送信する．次に自ノードが管理する ANNAI List に既に登録された名前と重複がないかを調べる．もし重複がなく，広告されている MAC アドレスも異なれば新たなエントリとして ANNAI List に登録する．もし名前が重複していなくても MAC アドレスが同じ場合はそのノードが名前を変更したとして ANNAI List のエントリを更新する．また，もし既に自ノードが保持する ANNAI List 中のエントリと広告された名前が重複している場合はユニキャストを利用して CONFLICT メッセージを送信する．CONFLICT メッセージを送信した場合はいずれも受信した HELLO メッセージは破棄し，ANNAI List に関しては何もしない．

5.3.2　名前保存モジュール

名前保存モジュールは，gethostbyname C ライブラリから利用できるようカーネル空間で動作する．カーネルが ANARCH メッセージを受信すると，名前保存モジュールはユーザ空間にパケットを上げる前に IP 層で複製を保存する．その際，既に保持している ANNAI nin に登録されている MAC アドレスと同一の MAC アドレスを持つエントリについては，ANNAI nin のタイムスタンプを更新する．同一の MAC アドレスであっても IP アドレスまたは名前が以前と異なっていた場合は，新たな情報に更新する．また，もし異なる MAC アドレスを持つにも関わらず同じ名前を通知しているメッセージがあった場合はそれを無視する．名前保存モジュールは，保持する ANNAI nin を検査して名前の重複があった場合，最初に登録した方のリストを保持し新たなエントリに関しては何もしない．そうすることで自ノード内の名前の一貫性を保つことができる．もしネットワークで名前衝突が解決された後に，そのエントリの

名前が変更された場合は再び ANNAI nin を更新することになる．ANNAI nin の各エントリにはタイムスタンプが付いており，一定時間を経過した場合はそのエントリは無効になる．同一ノードを判別する識別子は MAC アドレスである．この処理の流れを図 5.3 に示す．

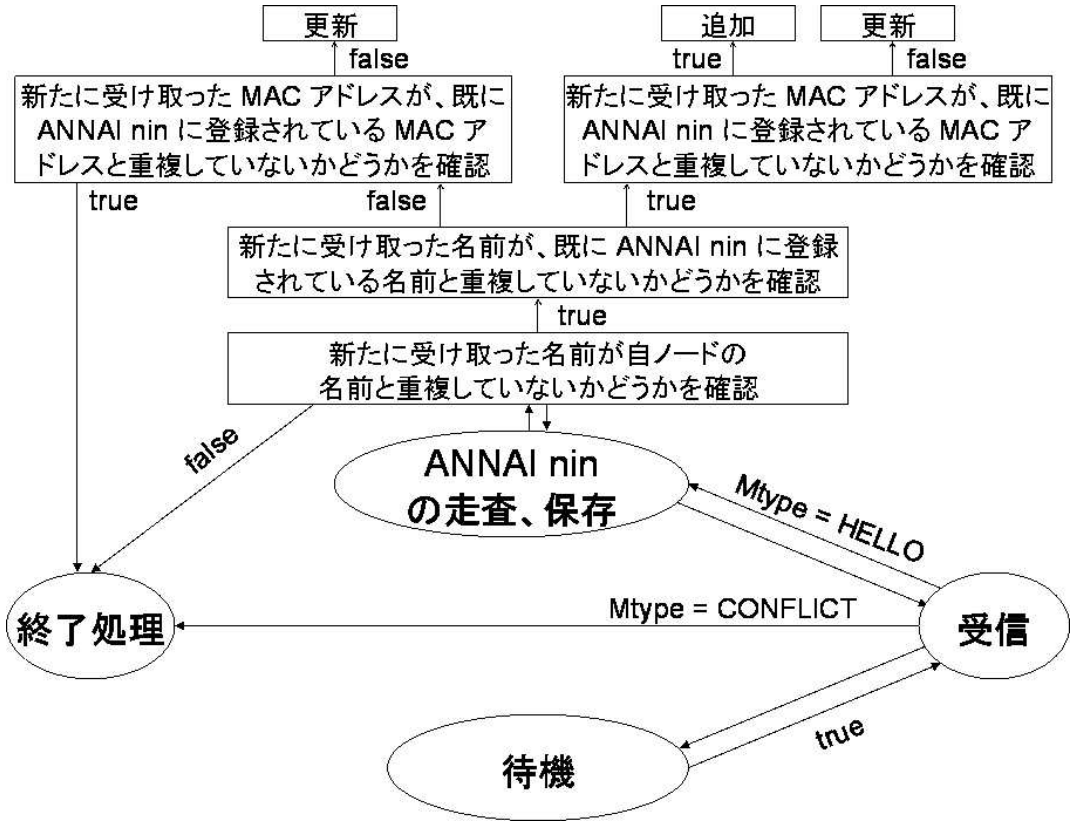

図 5.3: 名前保存モジュールの実装

カーネル空間に実装された名前保存モジュールは，I/O 読み込み機能を通じて，ユーザ空間の他のプログラムに対して DNS のような名前での通信を可能にする．gethostbyname C ライブラリ関数に加えられた ANARCH コマンドが I/O コントロール機能を通じてこのリストから該当する IP アドレスを取得しユーザプログラムに渡す．もし該当する名前がない場合 gethostbyname は続く標準コマンドを実行する．このモジュール自体は途中で名前情報の複製を作成するだけで，名前解決を行うためのコントロールメッセージを通知したり，ANNAI List の管理を行うことはない．

5.4 本章のまとめ

本章では ANARCH の実装全般について述べた．ANARCH の設計に従い二つのモジュールを実装し，汎用性を高めることにも成功した．名前管理モジュールは ANARCH システムの核心部分であり，HELLO，CONFLICT，LIST REQUEST，LIST REPLY

の 4 つのメッセージの交換により名前解決を行う．初期状態では，コア・リージョン内でのみ名前解決を行い，その後はユーザの要求に応じてホップ数をメトリックに必要な範囲まで名前解決を行う．このモジュールはユーザ空間のプロセスとして実装を行った．

また名前保存モジュールは付加的な役割を果たすものである．これは，名前管理モジュールから完成された名前のリストを複製し保存するもので，既存のユーザアプリケーションに対し ANARCH が解決した名前を提供する．ユーザアプリケーションは名前保存モジュールに問い合わせを行うことで，衝突した名前が解決されている間も一貫性のある名前を利用することが可能になる．このモジュールはカーネル空間に実装した．

第6章　ANARCH の評価

本章では，評価環境とシングルホップネットワーク・マルチホップネットワークにおける ANARCH の評価について述べる．

6.1　評価概要

本節では，FreeBSD3.3R [16] に実装した ANARCH を，DSR [4] を用いて構築した MANET 上で動作させた評価について述べる．まず評価環境を述べ，次にシングルホップ・マルチホップネットワークにおける ANARCH の評価について述べる．

6.1.1　評価環境

本研究では MANET を構築するルーティングプロトコルとして DSR を利用し，シングルホップで通信が完了する MANET と，マルチホップを利用して通信を行う MANET を構築した．シングルホップネットワークは，すべてのノード同士が 1 ホップで通信できるように設定した 8 台のノードで構築した．マルチホップネットワークは，4 台のノードを用意し最大 3 ホップのマルチホップを行うようネットワークを構築した．ノードの平均 CPU 周波数は 291.46 MHz であり，無線通信媒体には 802.11b [18] 規格の MELCO BUFFALO 無線 LAN カード AIRCONECT WLI-PCM-L11 を使用した．

6.2　シングルホップ

本節では，DSR を用いて実際に構築したシングルホップで通信が完了する MANET 上における ANARCH の性能評価の結果を述べる．測定環境としては ANARCH を組み込んだ 8 台の無線通信ノードを用意し，図 6.1 に示されているようなすべてのノードが 1 ホップ内で通信できるネットワークを構築した．

評価項目は，以下の二つである．

- ANARCH コントロールメッセージによるオーバヘッド
- 名前解決に要する時間

なお，この評価はネットワーク内のノード数と HELLO メッセージの送信間隔の二つのパラメータを変化させて行った．

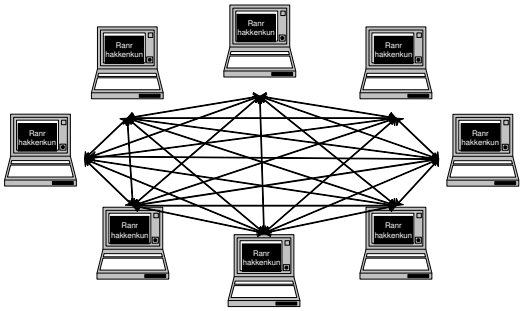

図 6.1: シングルホップ測定環境

6.2.1 コントロールメッセージによるオーバヘッド

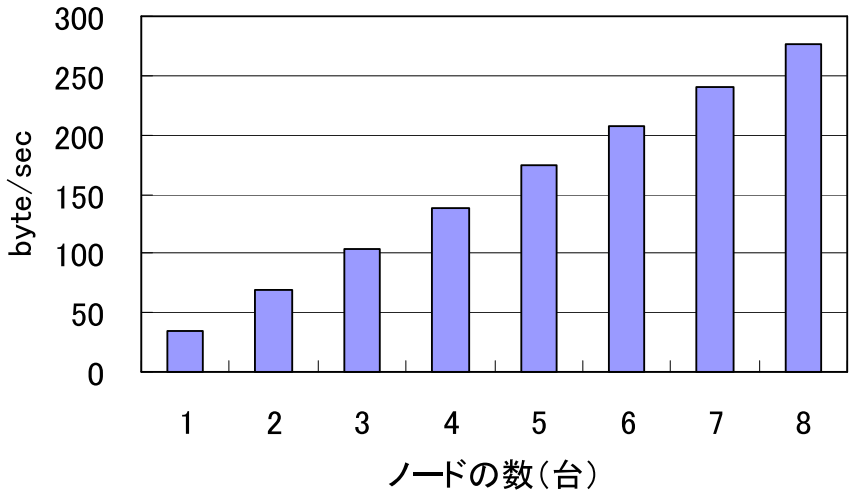

図 6.2: ノード数の変化による ANARCH のオーバヘッド

図 6.2 は名前解決のためのコントロールメッセージによるオーバヘッドの測定結果を示している．この測定は，名前の衝突は起こさず，HELLO メッセージを送信する間隔を 3 秒に固定し，ノード数を変化させて行った．台数に比例してオーバヘッドがかかり，最大 8 台の時は 277Byte/sec となった．3 台以上になると，1 台増えるごとに約 20 から 30%オーバヘッドが増加する．

図 6.3 は，同じ測定を HELLO メッセージを送る間隔を変化させて行った結果である．ノード数は 4 台に固定した．1 から 3 秒までは，1 秒間隔を広げるごとにオーバヘッドも 30%以上減少した．3 から 5 秒の間ではさほど変化が見られないため，このネットワークでは 3 秒程度が適当な値であると思われる．

このパラメータを調整することで，様々な特徴を持つ MANET に適応させることが

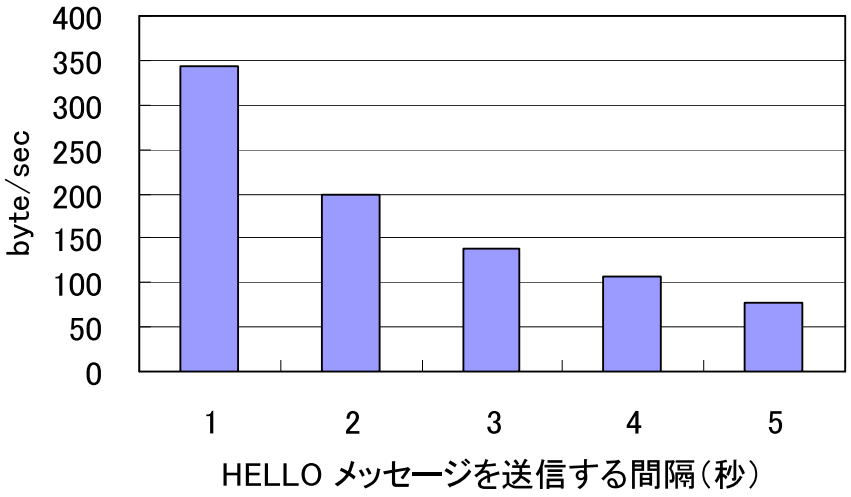

図 6.3: HELLO メッセージ送信間隔の変化による ANARCH のオーバヘッド

できる．例えば，構成ノードの変化や移動が頻繁に起こる MANET では間隔を短くすればよい．また比較的静的に構成される MANET や，ネットワークの規模またはノードの省エネルギーなどの理由で名前解決によるオーバヘッドを軽減したい場合には間隔を長くすればよい．利用する MANET に最適な値は，許容できるオーバヘッドと続く評価で示す名前解決に要する時間を考慮して選択される．

6.2.2 名前解決に要する時間

次に名前解決のために要する時間を評価した．これはネットワーク内のすべてのノードの名前を衝突させて，解決にどのくらいかかるか測定したものである．本来は，衝突が検出された時点でユーザに新しい名前を要求するべきであるが，ユーザの操作に要する時間を含めずにシステムの性能を測定するため，予め複数の名前を候補として ANARCH に与え，名前の変更時にそれらの名前を順次用いた．各ノードが最初に用いようとした名前はすべて衝突しているが，2 回目に用いようとする名前には重複がない．またネットワーク内のすべてのノードが一斉に ANARCH による名前解決を開始するようにした．図 6.4 と図 6.5 は 10 回の測定結果の平均値を示している．

図 6.4 は HELLO メッセージを送る間隔は 3 秒に固定し，ノード数を変化させて測定を行った結果を示している．すべてのノードの名前が重複している場合でも，3 台から 8 台の間では名前解決に要する時間の差は 1 秒以内に収まっており，最大 8 台の場合でも平均 2.5 秒以内に解決が終了する．

また，図 6.5 はノード数を 4 台に固定し HELLO メッセージを送信する間隔を変化させて名前解決に要する時間を測定した結果である．短い間隔で HELLO メッセージを送る方が名前解決が早く終了することが分かる．これは，HELLO メッセージを受け取る回数が増えるとそれに比例して CONFLICT メッセージを出す回数も増えるため

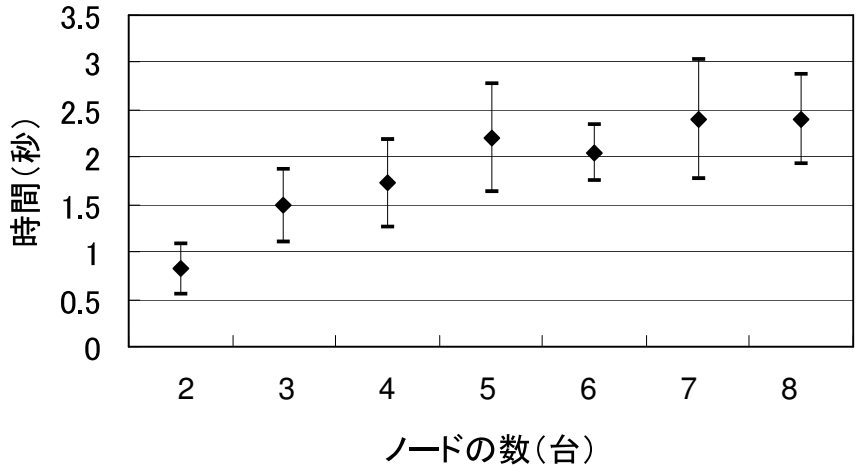

図 6.4: ノード数を変化させた場合の名前解決に要する時間

である．しかし，1 秒から 4 秒の間では名前解決にかかる時間の差は 1 秒以内に収まっており，さほど変わらない．そのため，図 6.2 とのトレードオフを考慮すると HELLO メッセージを送信する間隔はこの実験ネットワークでは 3 秒程度が適当である．3 秒より間隔を狭めてもそのオーバヘッドに対して名前解決に要する時間が改善されないためである．

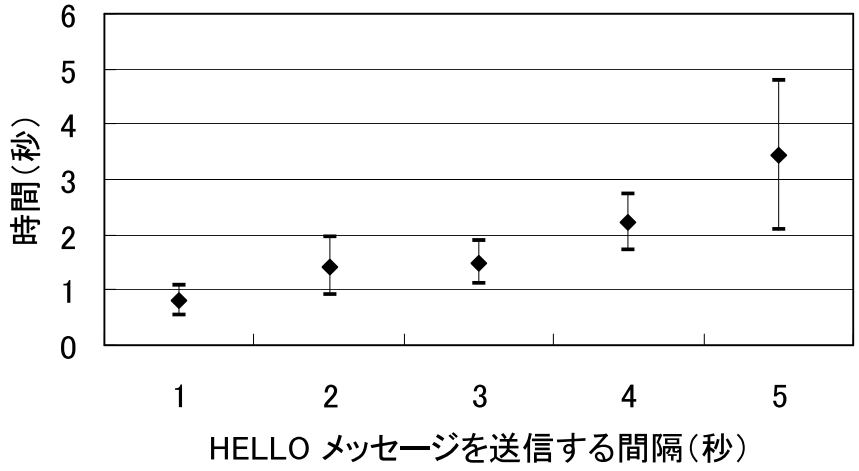

図 6.5: HELLO メッセージ送信間隔を変化させた場合の名前解決に要する時間

これらの測定結果は ANARCH がノード数 8 台のネットワークでそれほど大きなオーバヘッドをかけることなく，現実的な時間の範囲内で名前解決を実現していることを示している．また，ANARCH のスケーラビリティを 8 台まで確認した．さらに，HELLO メッセージを送信する間隔を調整することでネットワークにかかる負荷を調

整できることが分かった．また，その間隔の短縮化が，生じる負荷と比較して必ずしも名前解決時間の短縮に結びつかないことも考察できる．ネットワークの規模に応じてこの値を調整し ANARCH を有効に動作させることができるが，比較的小規模なネットワークでは 3 秒程度の間隔が適当であると思われる．

6.3 マルチホップ

本節では，DSR を用いて実際に構築したマルチホップで通信を行う MANET 上における ANARCH の性能評価の結果を述べる．測定環境としては ANARCH を組み込んだ 4 台の無線通信ノードを用意し，図 6.6 に示されているようなトポロジのネットワークを構築した．

このネットワークにおいて，矢印で結ばれているノード同士は直接通信することが可能だが，そうでないノード同士は直接通信できるノードを経由して通信する必要がある．このトポロジには最大でノード A・D 間の 3 ホップの経路が存在する．測定はすべてノード A で行った．ノード A から見ると，直接通信できるノード B はコア・リージョンに，2 ホップで通信できるノード C は X_2 リージョンに，3 ホップで通信できるノード D は X_3 リージョンに存在することになる．

評価項目は，以下の三つである．

- ANARCH コントロールメッセージによるオーバヘッド
- 名前解決に要する時間
- 重複した名前の解決結果

なお，この評価はネットワーク内のノード数と HELLO メッセージ，LIST REQUEST メッセージの送信間隔の二つのパラメータを変化させて行った．

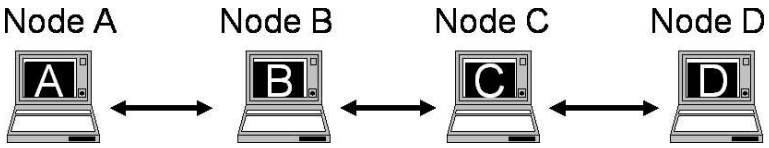

図 6.6: マルチホップ測定環境

6.3.1 コントロールメッセージによるオーバヘッド

まず，マルチホップネットワークにおける名前解決のためのコントロールメッセージによるオーバヘッドの測定結果を以下に示す．この測定は，ノード数 4 台のネットワークで HELLO メッセージ，LIST REQUEST メッセージを送信する間隔を変化さ

せ，流れるデータ量を 5 台目のノードにおいて計測したものである．すべてのノードは同時に ANARCH を開始した．

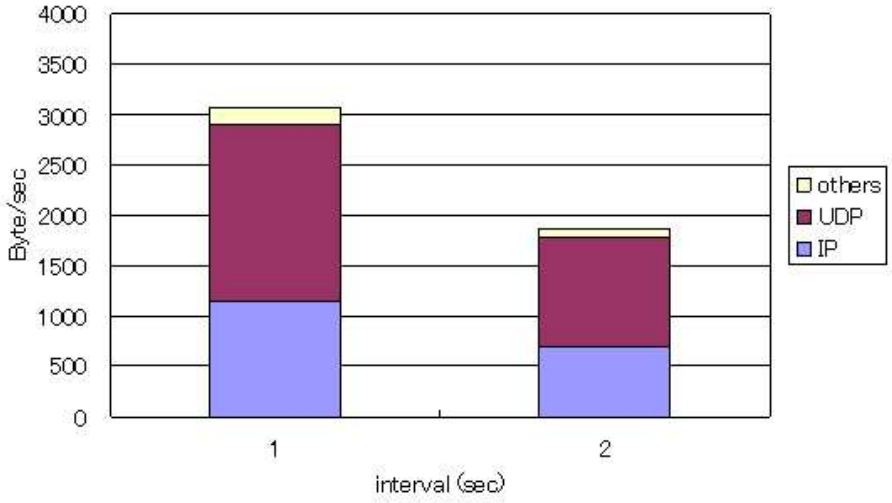

図 6.7: プロトコル別にみたネットワーク全体にかかるオーバヘッド：
名前の衝突がない場合

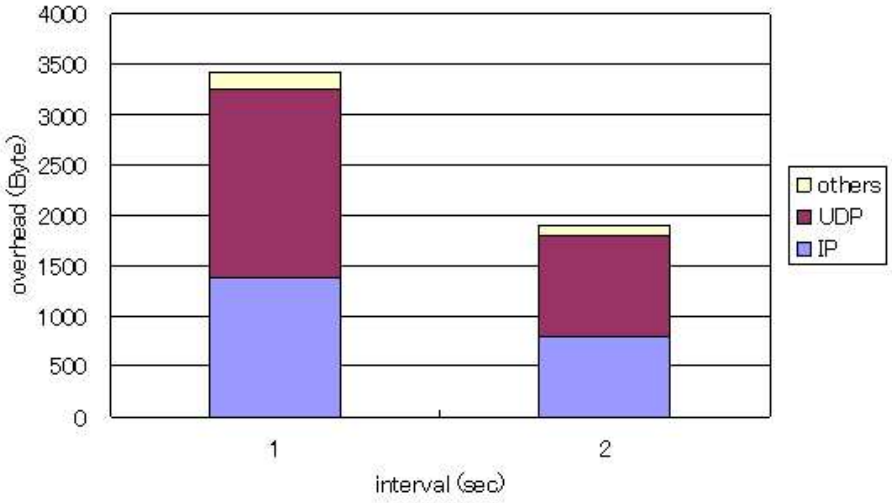

図 6.8: プロトコル別にみたネットワーク全体にかかるオーバヘッド：
名前の衝突がある場合

まず，プロトコル別にみた ANARCH のオーバヘッドを示す．図 6.7 は名前の衝突がない場合の結果を，図 6.8 は名前の衝突がある場合の結果をそれぞれ示している．これ

らの数値は 4 台で構築したマルチホップネットワーク全体に流れたパケット量の合計であり，評価用に別に用意した 5 台目の測定用ノードにおいて計測した．others に区分されているものには，802.11 Control Acknowledgment パケットや 802.11 Management Probe Req パケット（ether ヘッダ）が含まれる．名前の衝突を起こした評価では，始めに用いようとする名前を 4 台すべて衝突させ，第二候補としてどのノードとも衝突しない名前の使用を試みるようにした．その際ユーザの操作に要する時間を含めずにシステムの性能を測定するため，予め複数の名前を候補として ANARCH に与え，名前の変更時にそれらの名前を順次用いた．これは，前節で述べたシングルホップにおける名前衝突時の評価と同じ条件である．

この結果で注目できる点は，名前の衝突を解決した場合でも衝突がなかった場合と比べてほとんどオーバヘッドが変わらないことである．ANARCH は効果的に名前の衝突を解決することができる．

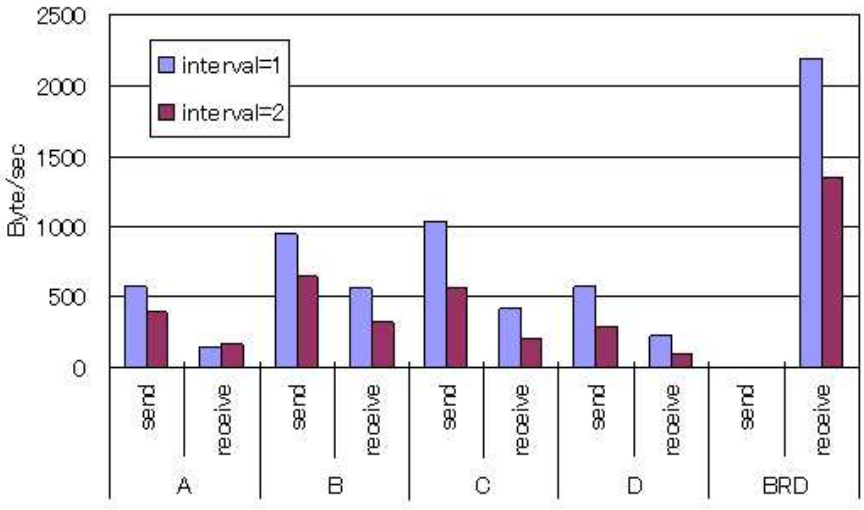

図 6.9: ノード別にみたオーバヘッド：名前の衝突がない場合

次に同じ評価をノード別に分析した結果を示す．図 6.9 は名前の衝突がない場合の結果を，図 6.10 は名前の衝突がある場合の結果をそれぞれ示している．ノード A からノード D に分類されているパケットはユニキャストで送受信されたもので，ブロードキャストに分類されているものはネットワーク全体に送信されたブロードキャストパケットの合計である．この場合も，名前の衝突があった場合でもその解決はそれほどのオーバヘッドにはならないことが分かる．

ユニキャストで送受信されているメッセージは CONFLICT・LIST REPLY メッセージであり，ブロードキャストで送受信されているメッセージは HELLO・LIST REQUEST メッセージである．マルチホップネットワークでは，オーバヘッドの値はシングルホップネットワークにおけるようにコントロールメッセージの送信間隔に全く比例しているわけではないと考えられる．またこの図には，ノード B・ノード C の方がノード A・

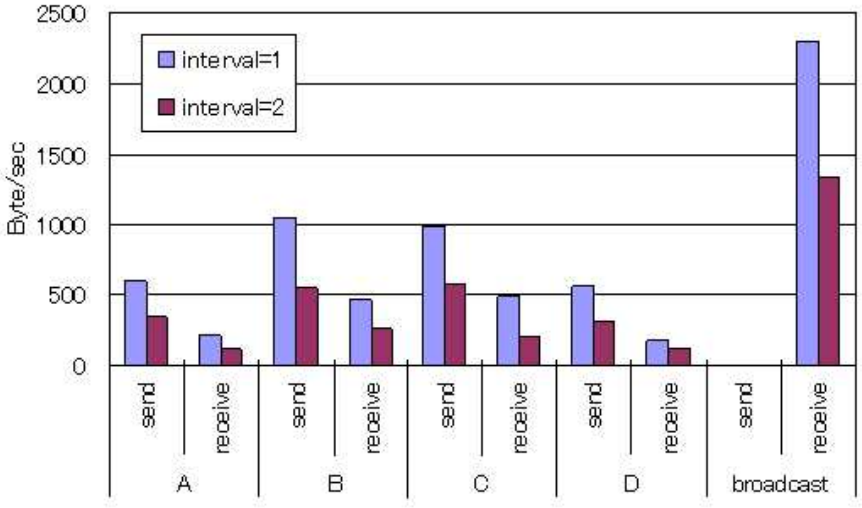

図 6.10: ノード別にみたオーバヘッド：名前の衝突がある場合

ノード D よりも多くの送受信を行ったことが示されている．これは，ノード A・ノード D はコア・リージョン（1 ホップで通信可能な範囲）に 1 台のノードのみが存在していたのに対し，ノード B・ノード D はコア・リージョンに 2 台のノードが存在していたためである．つまり，ネットワークの末端にいくほどオーバヘッドが少なくなるといえる．ANARCH ではコア・リージョンにおいてのみメッセージの交換が行われるため，コア・リージョンにどれだけのノードが存在しているかがオーバヘッドを左右することになる．

6.3.2 名前解決に要する時間

次に名前解決のために要する時間を評価した．ANARCH を利用する場面では，名前解決を開始するのに以下の 3 種類のパターンが考えられる．以下の評価では，この 3 パターンについてそれぞれ測定を行った結果を示す．

1. ネットワーク内のすべてのノードが同時に名前解決を開始する：名前の衝突がなかった場合
2. ネットワーク内のすべてのノードが同時に名前解決を開始する：名前の衝突があった場合
3. 既に存在するネットワークに新たに加わり名前解決を行う場合

パターン 1，2 については，ネットワーク内のすべてのノードは一斉に ANARCH による名前解決を開始し，X リージョン内の名前解決を要求する．パターン 2 では，本

来は衝突が検出された時点でユーザに新しい名前を要求するべきであるが，ユーザの操作に要する時間を含めずにシステムの性能を測定するため，予め複数の名前を候補として ANARCH に与え，名前の変更時にそれらの名前を順次用いた．各ノードが最初に用いようとした名前はすべて衝突しているが，2 回目に用いようとする名前には重複がない．これはシングルホップにおける名前衝突時の評価と同じ条件である．パターン 3 については，ノード B・C・D で構築された MANET にノード A を加えた．既に存在していたネットワークでは，名前解決は終了しているものとする．測定はノード A で行った．これは，ノード A が X_3 リージョンに存在するノード D（ノード A から 3 ホップで到達可能なノード）の名前を解決するのに要する時間の計測で，ノード A が ANARCH を開始した時点から，ノード A がノード D の名前を解決した時点までの測定である．

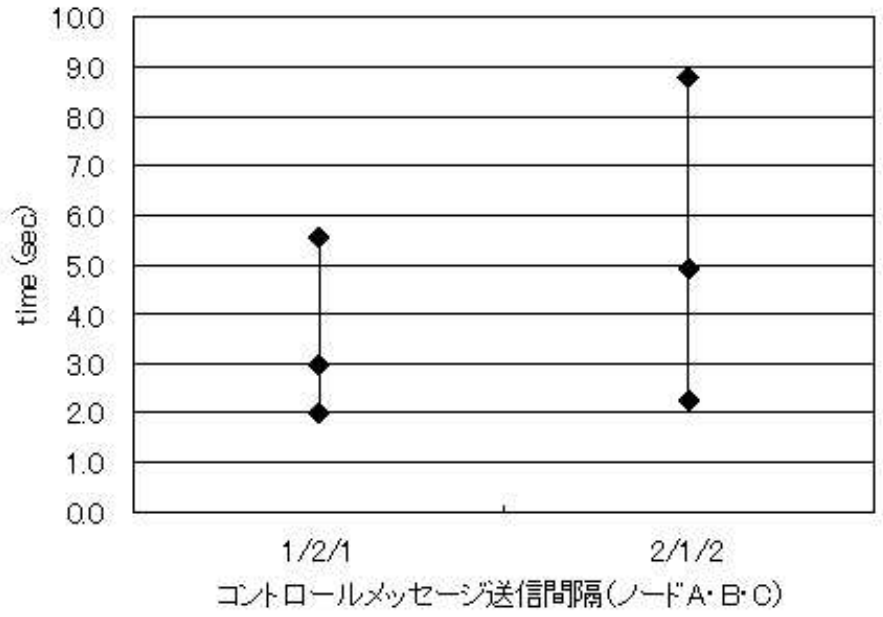

図 6.11: X_2 リージョンの名前解決に要する時間

まずノード A・B・C で構築した 2 ホップで到達可能なネットワークにおける測定の結果を図 6.11 に示す．これは HELLO・LIST REQUEST メッセージの送信間隔が，ノード A・B・C においてそれぞれ 1 秒・2 秒・1 秒の場合と，2 秒・1 秒・2 秒であった場合の名前解決に要した時間の最大値，平均値，最小値を示している．この評価では，1 秒・2 秒・1 秒とコントロールメッセージの送信間隔を全体で短くした方が平均 2 秒早く解決された．

次に，ノード A・B・C・D で構築した 3 ホップで到達可能なネットワークにおける評価結果を示す．図 6.12 は HELLO・LIST REQUEST メッセージの送信間隔が，ノード A・B・C・D においてそれぞれ 1 秒・2 秒・1 秒・2 秒の場合を示しており，図 6.13 は

1 秒・2 秒・2 秒・1 秒，図 6.14 は 2 秒・1 秒・1 秒・2 秒の場合を示している．値は 20 回測定を行った結果の平均値である．また x 軸の start・join・conflict はそれぞれ，パターン 1 （ネットワーク内のすべてのノードが同時に名前解決を開始する：名前の衝突がなかった場合），パターン 3（既に存在するネットワークに新たに加わり名前解決を行う場合）パターン 2（ネットワーク内のすべてのノードが同時に名前解決を開始する：名前の衝突があった場合）を示している．この評価では，X_2 リージョン，X_3 リージョンに存在するノードの名前解決に要する時間をそれぞれ測定して比較した．2hop，3hop という評価軸はそれぞれ，X_2 リージョンに存在するノード C（ノード A からみて 2 ホップで到達できるノード）の名前を解決するのに要する時間と，X_3 リージョンに存在するノード D（ノード A からみて 3 ホップで到達できるノード）の名前を解決するのに要する時間を示している．

これらの結果が示しているように，マルチホップネットワークではシングルホップネットワークに比べて名前解決に要する時間は長くなる．これは，ノード A がノード D の名前を解決するまでに以下の段階を踏むためである．

- ノード A がノード B の名前を解決
- ノード B がノード C の名前を解決
- ノード C がノード D の名前を解決
- ノード A がノード X リージョンの名前解決を要求
- ノード B がノード X リージョンの名前解決を要求
- ノード C がノード D の名前情報をノード B に提供
- ノード B がノード C・D の名前情報をノード A に提供
- ノード A がノード D の名前を解決

ANARCH ではすべてのメッセージはコア・リージョン内でのみ交換され，コントロールメッセージの転送によるフラッディングは一切行わない．しかし，上記の手順で X リージョン内の名前の解決が可能になる．

3 ホップネットワークで行った評価は，どれもネットワーク全体でみると HELLO メッセージ，LIST REQUEST メッセージを送信する量は同じであるが，他のノードがホップするノードとなるノード B・C がコントロールメッセージの送信間隔を狭めると名前解決が早く終了することを示している．また，既に名前解決が終了しているネットワークに新たに参加する場合には短い時間で名前解決が行われる．現実には，評価で行ったようにすべてのノードがまったく同時に ANARCH による名前解決を始めることよりは，多少の時間差で名前解決が終了したネットワークに新たなノードが加わっていく場合の方がが多いと予想される．そのため，パターン 3 の場合の評価結果である平均 2 秒から 4 秒が現実に最も近く，妥当な値であると考えられる．

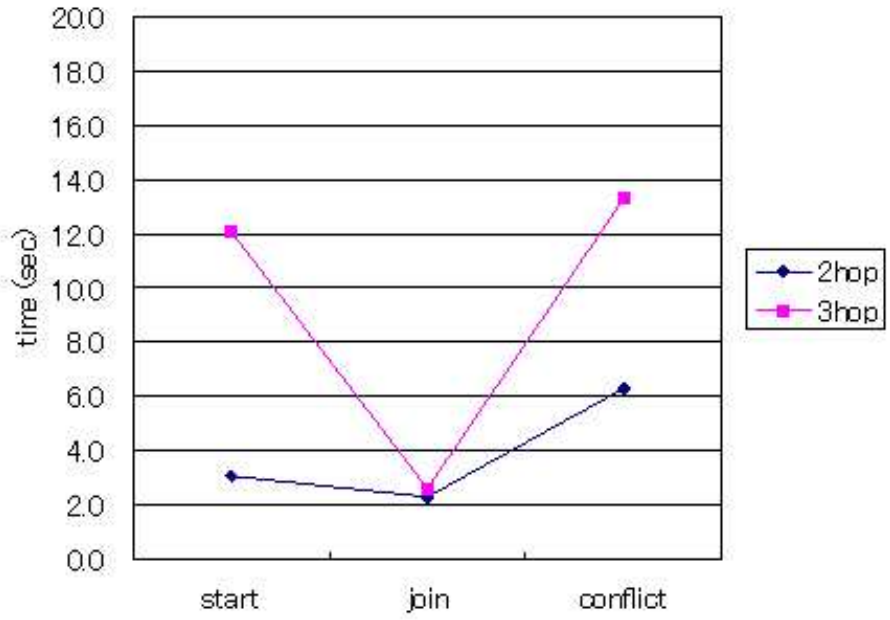

図 6.12: コントロールメッセージの送信間隔：1 秒・2 秒・1 秒・2 秒の場合

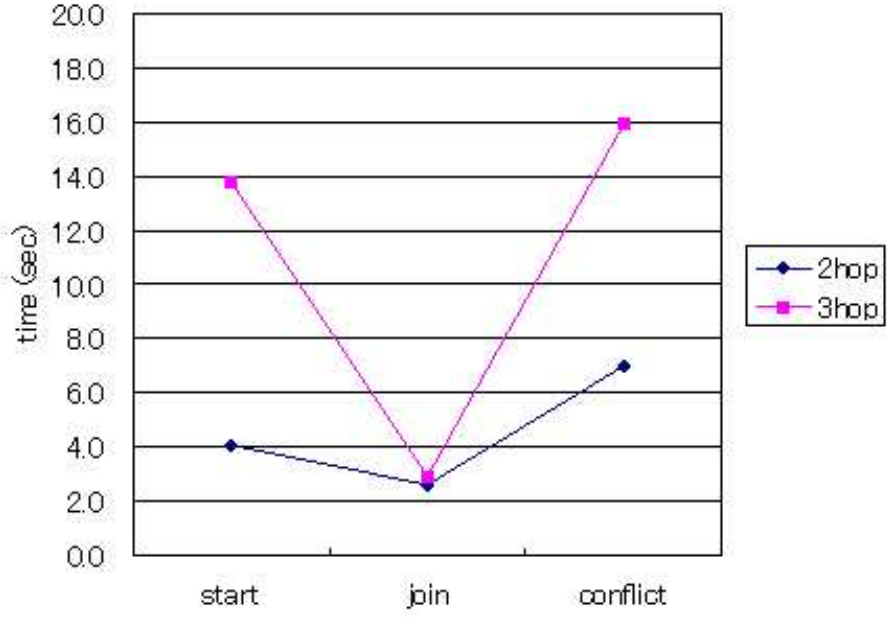

図 6.13: コントロールメッセージの送信間隔：1 秒・2 秒・2 秒・1 秒の場合

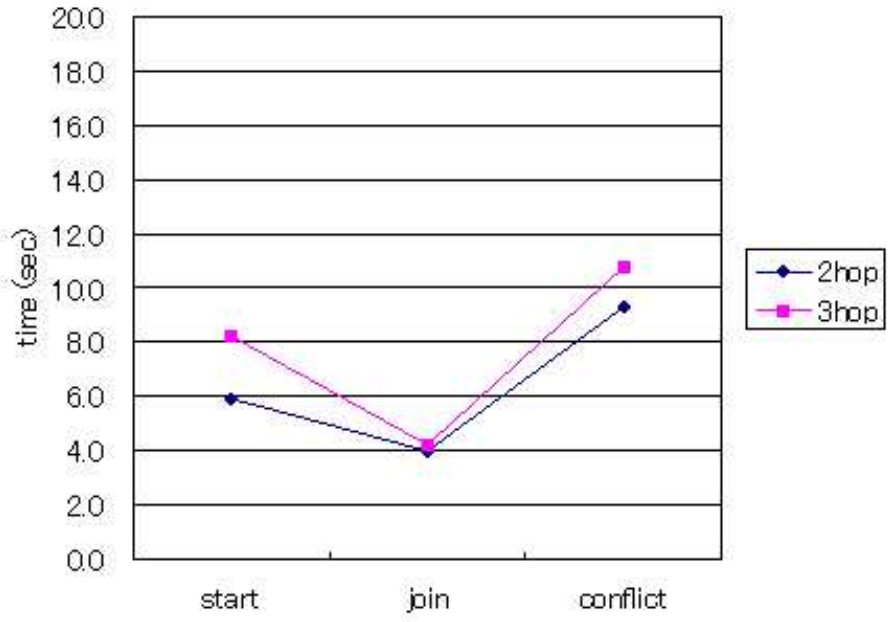

図 6.14: コントロールメッセージの送信間隔：2 秒・1 秒・1 秒・2 秒の場合

ここで，マルチホップネットワークにおいて衝突した名前を ANARCH で解決した場合，結果として各ノードはどの名前を用いることができるようになるのか興味深い点である．次小節でこの点を述べる．

6.3.3 重複した名前の解決

前小節では，ANARCH が名前解決に要する時間についての評価結果を述べたが，特に名前の衝突がある場合の評価を行う過程で ANARCH が実際にどのように衝突した名前を解決したのか興味深い点であったので，本小節で概説する．

前述した名前の衝突がある場合の評価では，最初 ANARCH が開始した時にはすべてのノードが同じ名前を使用しようと試み，次いで第二候補として皆が異なった名前の使用を試みるようにした．実際に用いられた各ノードの名前は表 6.1 の通りである．そして，実際に名前解決が行われた結果各ノードが用いた名前を図 6.15 から図 6.26 に示す．これはある連続した 15 回の実験の結果である．どのノードも第二候補の名前を利用することになった場合が多かったが，衝突していた名前を使い続けることのできたノードも存在していた．衝突していた名前を使い続けることのできた回数は表 6.2 の通りであった．

ノード A は図 6.16，図 6.19，図 6.20，図 6.26（実験 2・5・6・12）において，ノード B は図 6.23，図 6.24（実験 9・10）において，ノード C は図 6.22（実験 8）において，ノード D は図 6.15，図 6.17，図 6.18，図 6.25（実験 1・3・4・11）においてそれぞ

表 6.1: 名前を衝突させた実験で利用した名前

ノード	第一候補	第二候補
A	motoi	aa
B	motoi	bb
C	motoi	cc
D	motoi	dd

表 6.2: 衝突した名前の使用を継続することが可能であった回数

ノード	回数
A	4
B	2
C	1
D	4

れ衝突した名前の獲得に成功している．ノード A・ノード D が名前の獲得に成功する割合が高いが，これは CONFLICT メッセージを受信する割合が他のノードより低いためである．なぜなら CONFLICT メッセージを受信する数は，ノード A はノード B からの一つ，ノード B はノード A・ノード C からの二つ，ノード C はノード B・ノード D からの二つ，ノード D はノード C からの一つだからである．図 6.21（実験 7）ではタイミング悪く全てのノードが CONFLICT メッセージを受信したため，すべてのノードが名前を変える結果となっている．

6.4 本章のまとめ

本章では ANARCH を実際に構築した MANET 上で評価した結果を述べた．シングルホップネットワークとマルチホップネットワークを構築し，それぞれで ANARCH のコントロールメッセージによるオーバヘッドと，名前解決に要する時間を評価した．また実際にマルチホップネットワークにおいて名前が衝突した場合に ANARCH が名前解決を行った結果を示した．ANARCH はネットワーク内のすべてのノードの名前が衝突していた場合でも現実的な時間の範囲内で名前解決を行った．また，ANARCH は名前情報を直接通信できるコア・リージョン内のノードにしか送信しないが，いかなるフラッディングも行わずに X リージョン内の名前解決を正しく行った．

相手ノード＼自ノード	A	B	C	D
A 00:02:2d:00:63:3d		aa	aa	aa
B 00:60:1d:f2:c5:97	bb		bb	bb
C 00:02:2d:00:65:10	cc	cc		cc
D 00:60:1d:f2:c5:f9	motoi	motoi	motoi	

図 6.15: 実験 1

相手ノード＼自ノード	A	B	C	D
A 00:02:2d:00:63:3d		motoi	motoi	motoi
B 00:60:1d:f2:c5:97	bb		bb	bb
C 00:02:2d:00:65:10	cc	cc		cc
D 00:60:1d:f2:c5:f9	dd	dd	dd	

図 6.16: 実験 2

相手ノード＼自ノード	A	B	C	D
A 00:02:2d:00:63:3d		aa	aa	aa
B 00:60:1d:f2:c5:97	bb		bb	bb
C 00:02:2d:00:65:10	cc	cc		cc
D 00:60:1d:f2:c5:f9	motoi	motoi	motoi	

図 6.17: 実験 3

相手ノード＼自ノード	A	B	C	D
A 00:02:2d:00:63:3d		aa	aa	aa
B 00:60:1d:f2:c5:97	bb		bb	bb
C 00:02:2d:00:65:10	cc	cc		cc
D 00:60:1d:f2:c5:f9	motoi	motoi	motoi	

図 6.18: 実験 4

相手ノード＼自ノード	A	B	C	D
A 00:02:2d:00:63:3d		motoi	motoi	motoi
B 00:60:1d:f2:c5:97	bb		bb	bb
C 00:02:2d:00:65:10	cc	cc		cc
D 00:60:1d:f2:c5:f9	dd	dd	dd	

図 6.19: 実験 5

相手ノード＼自ノード	A	B	C	D
A 00:02:2d:00:63:3d		motoi	motoi	motoi
B 00:60:1d:f2:c5:97	bb		bb	bb
C 00:02:2d:00:65:10	cc	cc		cc
D 00:60:1d:f2:c5:f9	dd	dd	dd	

図 6.20: 実験 6

相手ノード＼自ノード	A	B	C	D
A 00:02:2d:00:63:3d		aa	aa	aa
B 00:60:1d:f2:c5:97	bb		bb	bb
C 00:02:2d:00:65:10	cc	cc		cc
D 00:60:1d:f2:c5:f9	dd	dd	dd	

図 6.21: 実験 7

相手ノード＼自ノード	A	B	C	D
A 00:02:2d:00:63:3d		aa	aa	aa
B 00:60:1d:f2:c5:97	bb		bb	bb
C 00:02:2d:00:65:10	motoi	motoi		motoi
D 00:60:1d:f2:c5:f9	dd	dd	dd	

図 6.22: 実験 8

相手ノード＼自ノード	A	B	C	D
A 00:02:2d:00:63:3d		aa	aa	aa
B 00:60:1d:f2:c5:97	motoi		motoi	motoi
C 00:02:2d:00:65:10	cc	cc		cc
D 00:60:1d:f2:c5:f9	dd	dd	dd	

図 6.23: 実験 9

相手ノード＼自ノード	A	B	C	D
A 00:02:2d:00:63:3d		aa	aa	aa
B 00:60:1d:f2:c5:97	motoi		motoi	motoi
C 00:02:2d:00:65:10	cc	cc		cc
D 00:60:1d:f2:c5:f9	dd	dd	dd	

図 6.24: 実験 10

相手ノード＼自ノード	A	B	C	D
A 00:02:2d:00:63:3d		aa	aa	aa
B 00:60:1d:f2:c5:97	bb		bb	bb
C 00:02:2d:00:65:10	cc	cc		cc
D 00:60:1d:f2:c5:f9	motoi	motoi	motoi	

図 6.25: 実験 11

相手ノード＼自ノード	A	B	C	D
A 00:02:2d:00:63:3d		motoi	motoi	motoi
B 00:60:1d:f2:c5:97	bb		bb	bb
C 00:02:2d:00:65:10	cc	cc		cc
D 00:60:1d:f2:c5:f9	dd	dd	dd	

図 6.26: 実験 12

第7章　結論

7.1　今後の課題

ANARCH の今後の課題を以下に列挙する．

- スケーラビリティの評価

  スケーラビリティについてさらに評価を行う．本論文では，実機により実際に構築した MANET での評価を行ったが，実機でできる評価には限りがある．シミュレータなどを利用して，200 台，300 台のノードでの実験を行う．

- 名前セマンティックスの拡張

  ANARCH ではフラットな名前空間を用いているが，名前のセマンティックスを拡張できるようメッセージには予約領域が残されている．その部分を利用して，サービス名や位置情報など，MANET が実用化されたときに要求される種類の情報に関連付けられた名前を利用することを可能にする．

- ノードを判別する手法

  MANET では IP アドレス・名前がノードに一定でないため，ノードを判別する何らかの識別子が必要になる．ANARCH ではノードに一意である識別子として無線インタフェースの MAC アドレスを利用したが，この一意性が必ずしも保証されるとは限らないことが指摘されている．MAC アドレスに代わる汎用性のある手法を探ることが必要である．

- セキュリティ

  現在の ANARCH は性善説に基づき，ネットワーク内のノードをすべて一律に同じ信頼度で扱うことを前提としている．しかし MANET が利用される場面を考えると，未知のノードが多く存在することが容易に予想される．そのため，セキュリティを考慮したシステムに発展させることが非常に重要な課題である．

7.2　まとめ

本論文では MANET における名前解決の問題点を指摘し，その要件を分析した．これをふまえて，MANET 環境に適した名前解決機構として ANARCH を提案し，実装・評価を行った．ANARCH は完全分散型の名前解決機構であり，MANET 特有の，

動的なトポロジ変化や通信リンク品質の不安定さ，IP アドレスの変化にも柔軟に対応する．また ANARCH では，各ノードが独自に任意の名前を付けることが可能であり，ネットワーク内でその名前が衝突した場合には動的に解決する．また，既存のユーザプログラムが ANARCH により割り当てられた名前を利用することも可能である．

評価実験の結果から，8 台のノードで構築されたネットワークにおいて，ANARCH が適切に名前解決を行えることを示した．具体的には，名前解決の際の ANARCH によるパケットオーバヘッドは平均 277Byte/sec であり，3 秒間隔の HELLO メッセージを用いた場合にはネットワーク内における名前衝突の解決に平均 2.5 秒の待ち時間を要した．これらの結果から，8 台で構築されたシングルホップの MANET においては 3 秒間隔の HELLO メッセージの送信が適当であることを示した．また，マルチホップネットワークにおいてもメッセージのフラッディングを行わずに適切な名前解決を行う．3 ホップのマルチホップネットワークにおいて，名前の衝突があった場合でも約 5 秒程度で名前解決を行えることを示した．本論文の評価環境において，ANARCH はオーバヘッドを小さく抑える努力をしつつ，現実的な時間内での迅速な名前解決を実現した．

謝辞

本研究を進めるにあたり，御指導を頂きました，慶應義塾大学環境情報学部教授徳田英幸博士に深く感謝致します.

また，貴重な御助言を頂きました東京電機大学の戸辺 義人博士に感謝の意を表します.

間 博人氏，斉藤 匡人氏をはじめとする慶應義塾大学徳田研究室 ECN 研究グループの諸氏，また徳田・村井・楠本・中村研究室の方々には，研究会の活動を通じて多くの御意見，御助言を頂きましたこと拝謝します.

最後に，研究の日々を共に過ごした，高橋 ひとみ氏，峰松 美佳氏，滝沢 允氏，その他多くの友人に深謝し，謝辞と致します.

2003 年 1 月 31 日

青木 基衣

参照論文

- 青木基衣, 斉藤匡人, 間博人, 徳田英幸
  “アドホックネットワークに適した名前解決機構: RANR (ランダム割り当て名前解決機構), ” 情報処理学会 モバイルコンピューティングとワイヤレス通信研究会 (MBL),
  Mar. 2002.

- Motoi Aoki, Masato Saito, Hiroto Aida, and Hideyuki Tokuda
  “ANARCH: A Name Resolution Scheme for Mobile Ad Hoc Networks , ”
  The International Conference on Advanced Information Networking and Applications (AINA '03),
  Mar. 2003.

参考文献

[1] Albitz. P. and Liu. C, DNS and BIND, O'Reilly & Associates, Inc., Sebastopol, CA, (1994).

[2] C. E. Perkins and P. Bhagwat, "Highly Dynamic Destination Sequenced Distance Vector Routing (DSDV) for Mobile Computers", In Proc. of ACM SIGCOMM, pages 234-244, London, England, Aug. 1994.

[3] Droms. R, "Dynamic Host Configuration Protocol.", RFC 1541, (October 1993).

[4] Johnson. D, Maltz. D, Hu. Y, and Jetcheva. J., "The Dynamic Source Routing Protocol for Mobile Ad Hoc Networks", IETF Internet-Draft [Work in Progress], (2001).

[5] Laura Marie Feeney, Bengt Ahlgren, Assar Westerlund, "Spontaneous Networking: An Application-Oriented Approach to Ad Hoc Networking", IEEE Communications Magazine, (June 2001).

[6] Laura Marie Feeney, Bengt Ahlgren, Assar Westerlund, Adam Dunkels, "Spontaneous Networking for Secure Collaborative Applications in an Infrastructureless Environment", Collateral material for a Spontaneous Networking demo at the Pervasive2002 Conference in Zurich, (August 2002).

[7] Lilley. J, "Scalability in an Intentional Naming System. Master of Engineering Thesis", Massachusetts Institute of Technology, (June 2000).

[8] Mesut Gunes, Jorg Reibel, "An IP Address Configuration Algorithn for Zeroconf. Mobile Multi-hop Ad-Hoc Networks", International Workshop on Broadband Wireless Ad-Hoc Networks and Services, Sophia Antipolis, France, (September 2002).

[9] P. Jacquet, P. Muhlenthaler, and A. Qayyum. Optimized Link State Routing Protocol. Internet Draft, draft-ietf-manet-olsr-*.txt. Work in progress.

[10] Perkins. C., Royer. E. and Das. S., "Ad Hoc On Demand Distance Vector (AODV) Routing", IETF Internet-Draft [Work in Progress], (January 2002).

[11] Paal. E., Geir. E., "Name Resolution in on-demand MANETS and external IP Networks", http://www.ietf.org/internet-drafts/draft-engelstad-manet-name-resolution-00.txt [Work in progress], (Feburary 2003).

[12] R. G. Ogier, F. L. Templin, B. Bellur, and M. G. Lewis, "Topology Broadcast Based on Reverse-Path Forwarding (TBRPF)", in draft-ietf-manettbrpf-05.txt, 1 March 2002.

[13] Rajeev. K, Charles E. Perkins., "Service Discovery in On-Demand Ad Hoc Networks", Manet Working Group INTERNET DRAFT (draft-koodli-manet-servicediscovery-00.txt). (2 October 2002).

[14] Sanket. N, Ravi. P., "MANETconf: Configuration of Hosts in a Mobile Ad Hoc Network", IEEE Infocom 2002. (2002).

[15] Vixie. P (ed), Thompson. S, Rekhter. Y, Bound. J., "Dynamic Updates in the Domain Name System (DNS UPDATE)", RFC 2136, (1997).

[16] FreeBSD Project, http://www.freebsd.org.

[17] DNS hosts file, http://www.dns.net/dnsrd/

[18] IEEE Local and Metropolitan Area Network Standards Committee. Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications, IEEE Std 802.11-1997.

[19] Lmhosts file, http://www.microsoft.com/windows2000/techinfo/reskit/en-us/default.asp?url=/windows2000/techinfo/reskit/en-us/cnet/cnfi_brs_gwew.asp

[20] MOBILE AD-HOC NETWORKS WORKING GROUP (MANET), http://www.ietf.org/html.charters/manet-charter.html, (1998-11-29)

[21] NetBEUI (NetBIOS Extended User Interface), http://www.microsoft.com/technet/treeview/default.asp?url=/TechNet/prodtechnol/windows2000serv/reskit/intnetwk/part5/intch16.asp?frame=true#a

[22] NetBIOS (Network Basic Input Output System), http://msdn.microsoft.com/library/default.asp?url=/library/en-us/netbios/netbios_7q1x.asp

[23] NBT (NetBIOS over TCP/IP), http://msdn.microsoft.com/library/default.asp?url=/library/en-us/netbios/netbios_4awe.asp

[24] The MONARCH Project at Carnegie Mellon University, http://www.monarch.cs.cmu.edu/.

[25] Microsoft Windows, http://microsoft.com/windows/default.mspx

[26] Windows Internet Name Service,
http://www.microsoft.com/windows2000/techinfo/howitworks/communications
/nameadrmgmt/wins.asp

[27] Zero Configuration Networking,
http://www.ietf.org/html.charters/zeroconf-charter.html.