修　士　論　文

時空間アドレス割り当て機構の設計と実装

指導教員　　**瀬崎 薫　助教授**

東京大学 大学院

情報理工学系研究科 電子情報学専攻

氏名 **56409 岡野 諭**

提出日 **平成 19 年 2 月 2 日**

目次

図目次

第 1 章

序論

デバイス製造技術の向上などにより，無線通信機能を搭載した様々な携帯用小型端末が偏在するようになってきている [1][2][3]（図. 1.1）．測位技術についても，GPS において PDA に接続可能なものや携帯電話内蔵型など小型・安価なものが広まってきたのに続いて，無線 LAN を用いた手法 [4][5][6]，歩行者用推測航法モジュール [7]，電子タグによるものなどが実用化されてきている．これら多様な手

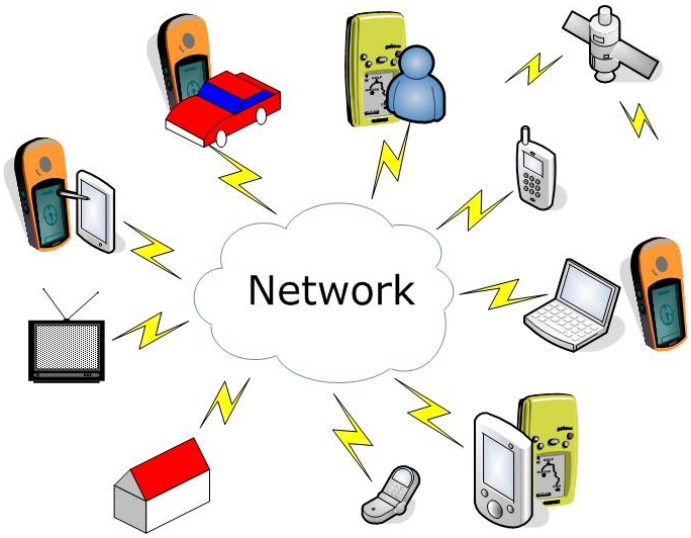

図 1.1　小型無線端末と測位デバイス

法により取得される位置情報自体は非常に価値のあるものであり，ある種のフレームワークを設計しこれらをシームレスに利用することができれば有用なサービスを実現することができる．サービスを提供する際，位置情報に基づいたデータ配信ができること，端末の詳細な行動履歴は隠蔽しつつ端末同士の位置情報を交換できることが重要であるが，そのために端末のアドレスとして位置情報が含まれていることが望ましい．

以上のような要請に基づき，端末の存在する時空間情報をもとに算出する時空間アドレス（Spatio-Temporal Address：STA）という手法が提案されている [20][21]．STA とは端末の存在する（位置，時刻）の組み合わせを基本情報として用いるアドレスである．

本研究ではさらに STA の具体的な実現方法として，端末のインタフェースに割り当てる IPv6 アドレ

スの中に STA を埋め込む方法が提案する．これは IPv6 のグローバル・ユニキャスト形式のアドレスのうち，一般には MAC アドレスが埋め込まれている後半 64 ビット，およびその直前でネットワーク管理者が決めることのできる 16 ビットの合計 80 ビットを使用し全地球上を緯度経度方向では約 1m 粒度で表現するというものである．またこの提案手法を実装し，位置情報交換・高精度化システムの一部として協調的に動作し，他の位置情報サービスの基盤となるソフトウェアとして，Linux 上で動作する STA Management Daemon の開発を行った．

本論文では時空間アドレス割り当て機構 STA Management Daemon の設計と実装を詳細に述べ，この機構を中心として実現されたシステムの評価実験を行った結果について報告，考察をおこなう．

本論文の構成は次のようになっている．第 2 章 位置情報サービスでは背景として，近い将来実現が期待されており，本研究の目標ともなる位置情報サービスについて説明する．第 3 章 既存手法では，関連研究を紹介する．モバイルアドホックネットワーク上で位置情報サービスを実現する上で重要となる位置情報ルーティング手法，モバイルアドホックネットワークでの端末へのアドレス割り当て手法の 2 つに分けて既存手法を紹介する．第 4 章 提案するシステムの設計では，まず，今回当研究室で提案するシステムの全体像を概説し，その後本研究において提案する時空間アドレス割り当て機構 STA Management Daemon の設計と，実現する機能や制限などについて述べる．第 5 章 提案するシステムの実装では，時空間アドレス割り当て機構の実装について詳細に述べる．第 6 章 実験では，以上で説明した実装を用いた評価実験を行い結果を述べ，考察を行う．第 7 章 結論は本論文のまとめである．あわせて今後の課題を述べる．

第 2 章

位置情報サービス

近年普及している小型端末は，あらゆる場所に移動可能であり，標準的に無線通信機能を搭載していて周囲の端末と無線で通信が可能である．測位デバイスもこれら小型端末に接続可能で安価なものが登場してきており，無線通信と位置情報を組み合わせたさまざまなサービスが考えられている．本章では研究背景として，近い将来実現が期待されている位置情報サービスについて，周辺の問題点を含めて概説する．研究段階の解決策についても一部紹介する．

2.1　防災支援

新潟県中越地震やインドネシア・スマトラ沖地震・津波など大規模な災害は国内外問わず発生している．地震の場合，P 波による初期微動を検知し，S 波の伝搬よりも早く警報を伝えることにより被害を最小限にとどめることができる．このようなシステムとしてユレダス，ナウキャスト [8][9] などがある．インターネットを通じて早期情報を配信する実証実験も行われている [10]．災害の警報配信では，当然であるが災害が発生している付近に情報を伝えなければ意味がなく，避難の誘導（図.2.1）なども想定されるため，情報を配信する対象の地理的範囲，道路の寸断など事故が起こっている位置に関する情報は重要である．範囲を限定して情報配信するサービス，イベントが起きている位置がわかるような仕組みが期待されている．通常の火災，地震直後のがれき中への生き埋め事故，雪崩事故などにおいて

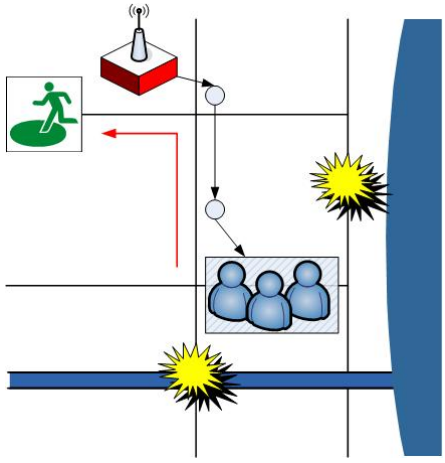

図 2.1　防災・避難支援

も，図.2.2 の曲線に示すように死亡率は時間とともに急激に上昇するため迅速に被災者の位置を特定して救助することが極めて重要である．そのために被災者の位置をおおよそ数 m 程度，建物内でいえば 1 部屋 1 部屋のレベルで特定できるようなシステムが期待されている．

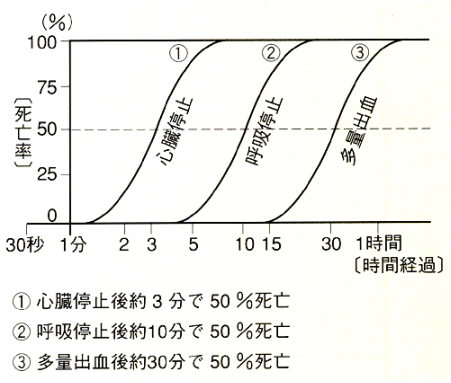

図 2.2　カーラーの救命曲線

2.2　子どもの見守り

小学校の校舎内や，通学路近くで児童が襲われる事件が多発している．集団下校をさせたり，保護者が迎えに行くなどの対応をとっている学校もあるが，主要な通学路上までは集団で下校しても自宅近くではどうしても 1 人になる，あるいは保護者も常に監視していることは難しいといった問題がある．近年は近所づきあいがなくなり，見守るべき対象の児童がわからないという新しい問題もでてきている．

これらの問題に対し，校区内の安全情報を GIS 上に集積したり，児童に GPS 端末や携帯電話を持たせて行動を把握したりするなどの実験がなされているが，個人情報保護の意識が高まりプライバシーとの両立の問題も注目されるようになってきている．普段から常時子どもを見守ることができ，さらにプライバシーとの両立にも配慮したサービスが望まれている．

2.3　自律移動支援

自動車にカーナビゲーションシステムが搭載されていることはもはや珍しくなくなったが [11]，歩行者にも同様のナビゲーションを提供しようという要求が高まってきている．ナビゲーションだけでなく，付近の観光情報，商店の情報などを組み合わせて配信して新たなサービスを構築できるようになることが期待されている．

視覚障碍者の移動支援に応用することも期待されている．従来の点字ブロックによる情報は単純な「停止」や「誘導」のみであったが，たとえば点字ブロックに無線タグを入れて白杖を読み取り機にすれば，単純な「停止」「誘導」だけでなく「ここは○号館」とか「何 m 先に交差点があります」など詳しい情報を伝えることができるようになる（図.2.3）．国土交通省などが研究・実験を進めているが，単に点字ブロックに無線 IC タグを内蔵するだけでは，白杖がブロックからはずれてしまうと読み取ることができなくなる．また白杖を日常的に利用しないユーザーには利用できないということもある．ある程度どのような状況でもシームレスにユーザーの位置取得，移動支援ができるようになることが期待さ

れている.

図 2.3　視覚障碍者の誘導

第 3 章

既存手法

本章では位置情報サービスの要素技術について，さまざまな既存手法を紹介し，問題点を述べる．また位置情報適応型サービスに実際に適用する場合の問題点についても検討を行う．

3.1　位置情報ルーティング手法

本節では，位置情報を用いた無線ネットワークルーティング手法について既存研究を紹介する．MANET 上での位置情報サービスに実際に適用する場合の問題点についても検討する．

3.1.1　Location-Aided Routing : LAR

LAR[17] は経路構築要求パケットをブロードキャストによって中継する，いわゆるフラッディングベースのプロトコルである．経路構築を行うとき，基本的には送信ノードからそのまわりのノードに次々とパケットがフラッディングされていく．ただしここで，(1) 送信ノードと宛先ノードを含む長方形を定義し，その中に含まれる近隣ノードにのみ転送を行う，(2) 各ノードと宛先ノードとの距離を計算し，距離が短くなる場合にのみ転送を行う，という 2 種類の方法を用いて制御パケットの削減を行っている．完全なフラッディングに比べると高効率であるが，省電力化の要求が厳しい MANET では実用化が難しいといえる．

3.1.2　Greedy Perimeter Stateless Routing : GPSR

Karp らにより [18] において提案されているのが Greedy Perimeter Stateless Routing（GPSR）である．GPSR では自ノードの位置は既知とし，さらに無線半径内でビーコンを一定時間ごとに送信することで近隣ノードの位置を取得する．その上で Greedy Forwarding と Perimeter Forwarding という 2 種類の転送方法で通信を行う．

3.1.2.1　Greedy Forwarding

Greedy Forwarding は，自ノードの無線半径内にあるノード（近隣ノード）のうち，宛先ノードに最も近いものを 1 つ選んで転送していく単純な方法である（図.3.1）．

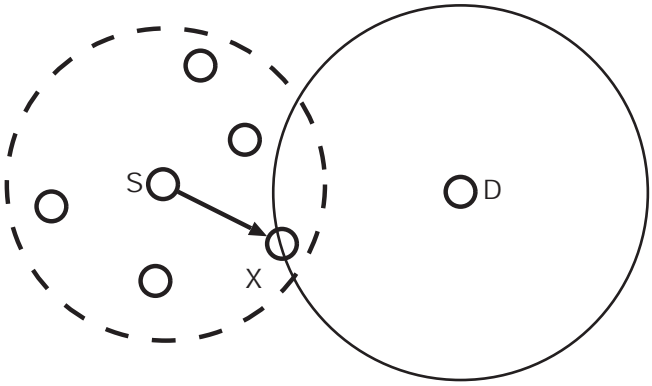

図 3.1　Greedy Forwarding

3.1.2.2　Perimeter Forwarding

無線半径内のどのノードよりも自ノードの方が宛先ノードに近いが，無線半径が宛先アドレスまでの距離よりは小さいため直接宛先に転送することができない場合に用いられるのが Perimeter Forwarding である．

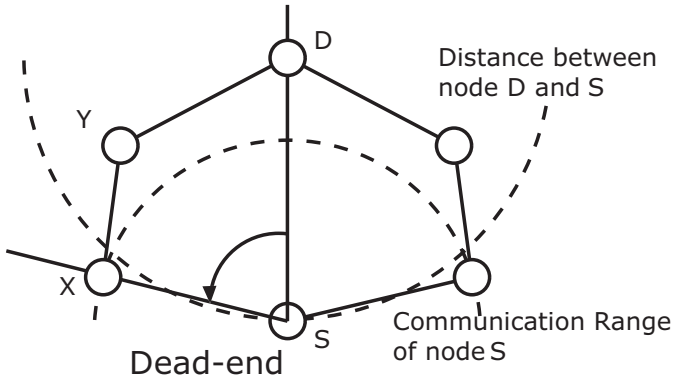

図 3.2　Perimeter Forwarding

Perimeter Forwarding では，図.3.2 のように自ノードと宛先ノードを結ぶ直線を中心に反時計方向の近隣ノードを探し，転送する．何度か Perimeter Forwarding をしたあと，より宛先に近いノードが見つかればその時点で再び Greedy Forwarding を開始する．

GPSR ではフラッディングを用いないため，LAR などと比べて経路構築の成功確率は若干落ちるものの，制御パケットは非常に少量で済み，電力の節約につながるため MANET に適しているといえる．

3.1.3　Routing Protocol with Ellipsoids : RPE

以上で概説した位置情報ルーティング手法は X，Y の二次元のみを考えたものであった．現実には高さの差によって，ルーティング性能が影響を受けることもあり実世界でのサービス展開のためには 3 次元を考慮することが必須条件となる．このような場合に対応するため Z 座標を考慮にいれて提案され

たのが Routing Protocol with Ellipsoids（RPE） [21][22] である．

RPE では，送信ノードと宛先ノードを焦点とする楕円体を利用して経路制御を行う．送信ノードと宛先ノードを焦点とする楕円体は無数に考えられるが，ある 1 つの近隣ノードを楕円体上の 1 点とすればそれぞれについて楕円体が一意に決定される．送信ノードは各近隣ノードについて楕円体を決定し，それらのうち，最も直線に近い楕円体上に存在するノードを次ホップノードとして選択する（図.3.3）．このような手法により，3 次元空間中においても制御パケット量を抑えつつ，効率よい経路制御が可能となる．

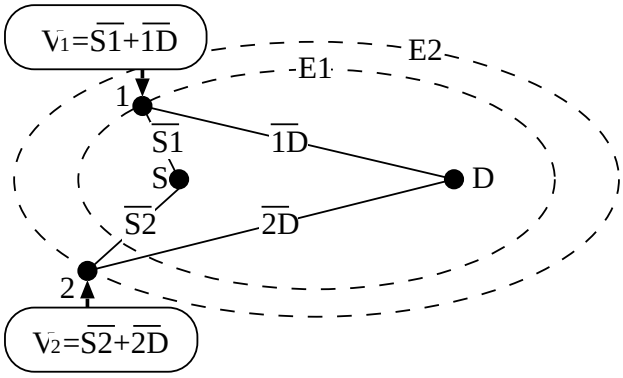

図 3.3　Routing Protocol with Ellipsoids

3.2　アドレッシング手法

本節では集中型制御，分散型制御双方におけるアドレッシング機構について述べる．

3.2.1　集中型

集中型制御におけるアドレッシング機構の例として，インターネットで広く用いられている DHCP を取り上げる．

3.2.1.1　Dynamic Host Configuration Protocol （DHCP）

現在，インターネットではアドレス割り当てのために動的ホスト構成プロトコル（Dynamic Host Configuration Protocol：DHCP）が広く使われている．DHCP はインターネットのホストにホスト固有の構成パラメータを伝えるメカニズムとホストのネットワークアドレスを割り当てるメカニズムとからなる．

ネットワークアドレス割り当ての概要は以下のようになっている．クライアントが適当な間隔でアドレスの使用を要求し，割り当てメカニズムは要求された時間内にはそのアドレスを再割り当てしないことを保証し，クライアントがアドレスを要求するたびに同じネットワークアドレスを返すよう試みる．

割り当てを行うサーバーは一貫性のチェックとしてたとえば ICMP エコーリクエストを使って，再利用するアドレスを割り当てる前にそのアドレスを調べるべきである．クライアントは，たとえば ARP

を使って，新たに受け取ったアドレスを調べるべきである．

このように，一貫性のチェックは厳密には既定がないため，アドレス衝突の検出が難しい MANET などにはそのまま実装することはできない．

3.2.2 分散型

分散型制御におけるアドレッシング機構の例として，主に MANET 上で使うことを想定した手法を 2 つ紹介する．

3.2.2.1 MANETconf

MANETconf は，ルーティングプロトコルなどとは関係なく一般的に MANET で使用可能な分散型ホスト構成プロトコルである．[13]

MANETconf の基本的な考え方は，新たに割り当てる候補のアドレスをネットワーク内にブロードキャストして提案し，すべてのノードから承認された場合にそのアドレスを割り当てるというものである．(図.3.4)

また Zeroconf では新しく参加したノードが IP アドレスを選んで提案していたが,

MANET では一般的には新しく参加するノードと通信できないため，新しく参加するノードに近いノードをイニシエータとして定め，イニシエータがアドレス割り当てを実行する．

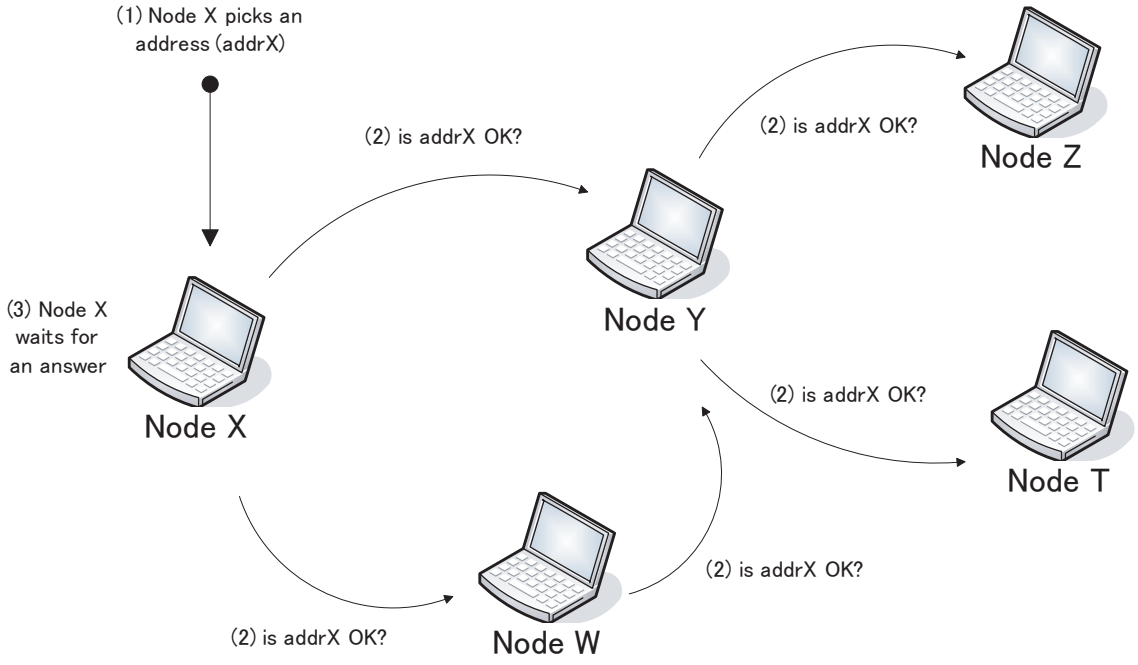

図 3.4 Conflict-detection allocation scheme

イニシエータはまだ割り当てられていないアドレスを選び，それをネットワーク内にブロードキャストする．このブロードキャストを受け取ったノードは，そのアドレスが割り当ての途中であることをマー

クして，イニシエータに肯定の返答を返す．すべてのノードから肯定の返事が返ってくれば割り当ては完了する．

ネットワークから離脱しようとするノードはメッセージをブロードキャストすることで他のノードに知らせ，アドレスを解放することができる．他のノードに知らせることなく突然停止してしまったりした場合は，次に新しいノードが参加してきてイニシエータがメッセージをブロードキャストしたときにそのノードから返答がないことから停止したと判断され，アドレスが解放される．

同時並行的なアドレス割り当てを認めているので，2 つのイニシエータが同時に同じアドレスを異なるノードに割り当てようとして衝突する可能性がある．衝突したときにはイニシエータの IP アドレスによって優先順位をつけその順位に従って割り当てを行う．若い IP アドレスをもっているイニシエータが優先されて割り当てを行うようになっている．

random waypoint mobility model[16] を用い，アドレス割り当てにかかる時間，やりとりされるメッセージの種類と数を調べるシミュレーションが行われた．ランダムで選ばれた始点から終点まで $5m/s$ の速度で進み，終点に着いたら $10s$ 止まって次の点まで進む．ネットワーク中の最大ノード数は 40，50，60，80 に設定されノードの密度は最大で 1 ノードあたり $0.02km^2$ と設定された．最大 50 ノードのケースでは元からアドレスが与えられているのが 30 ノード，新しく参加するのが 20 ノードとされ，最大 40 ノードの場合は元からアドレスのあるノードは 25 ノード，最大 60 ノードの場合は 35 ノード，最大 80 ノードの場合は 45 ノードとされた．新しくノードが参加する時間間隔は $0s$ から $75s$ まで一様に分布している．ノードの生存時間も一様に分布していて $0-1000s$，$0-2000s$，$0-15000s$ の 3 つの範囲が試された．シミュレーションは $3500s$ 実行された．

75% のノードが通知を出してから離脱し，25% のノードが突然離脱した場合にアドレス割り当てに要する時間を示した結果が図.3.5 である．大部分は $0.5s$ 以下で割り当てられているが，かなり長い時間がかかっている部分もある．長い時間がかかってしまっているのは，イニシエータがアドレス割り当てのためにブロードキャストをした後，返答を待っていたノードが離脱してしまったか，パケットが失われたかによってイニシエータからの呼びかけに答えないノードがあった場合である．

ノードの生存時間が $0-1000s$ で，75% のノードが通知を出してから離脱する場合，ユニキャスト，マルチキャスト，ブロードキャストのメッセージ数は図.3.6 のようになった．

割り当て 1 回あたりのブロードキャストメッセージは 5 回程度とかなり少なく，マルチキャストも少ないことがわかる．ただしこれはアドレス割り当て 1 回あたりであり，割り当て回数が増えれば同様にブロードキャスト回数も増えることになる．

3.2.2.2 Tayal らによる方法

MANETconf は新しく参加するノードに対してランダムでアドレスを選び，その後でアドレスの重複を検出するという方法であった．

これに対し Tayal らは，アドレスのプールを用い，新しく参加するノードにはプールの半分を分け与えていくことでアドレスの重複が起こさないようにするという手法を提案している (図.3.7)[14]．この手法を以下で詳しく説明する．

初期状態　最初に 1 つだけのノードがある状態を考える．このノードは $ADDR\_REQ$ メッセージを

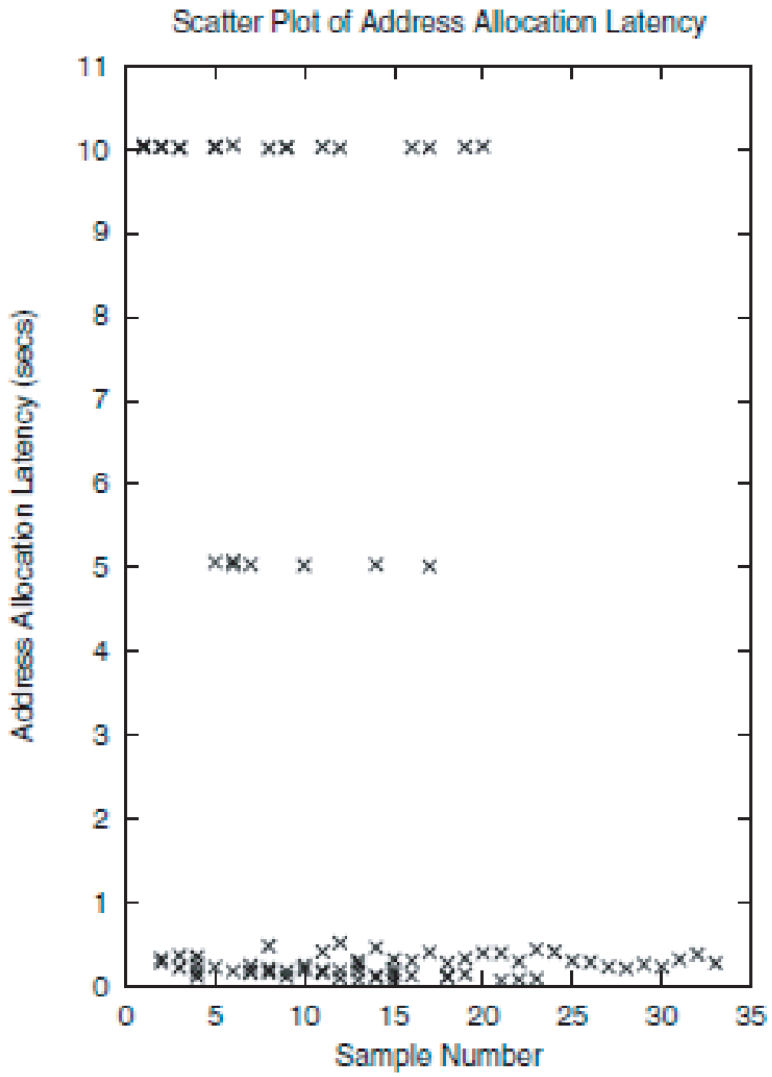

図 3.5　Address Allocation Latency Distribution in MANETconf

送信して $ADDR\_REQ\_TIMER$ 時間だけ待つ．まだ他にはノードがないので返答はないが，これを $ADDR\_REQ\_THOLD$ 回繰り返す．$ADDR\_REQ\_THOLD$ 回繰り返しても返答はないため，最初のノードであると判断され，デフォルトのアドレスを自分に割り当てて，デフォルトのプールを設定する．

ノードの参加　新たにノード i が加わることを考える．i に近いノード j が窓口となり，もし j がプールをもっている場合はその半分を i に与えて，i はプールの先頭アドレスを自分のアドレスとして割り当てる．もし j がプールをもっていない場合は j は $SEARCH\_ADDR$ メッセージをブロードキャストしてアドレスの探索を始める．プールをもっていないノードで $SEARCH\_ADDR$ メッセージを受け取った場合は，否定的な ACK を返し，さらに $SEARCH\_ADDR$ の転送を続ける．プールをもっているノードが $SEARCH\_ADDR$ を受け取った場合は，プールの半分の情報を返送し，その部分は割り当て中であることをマークする．そして j から $POOL\_ACCEPTED$ メッセージが届くのを待つ．j ははじめに届いたプールを受け取って $POOL\_ACCEPTED$ で応答し，受け取ったプールを i に割り当てる．

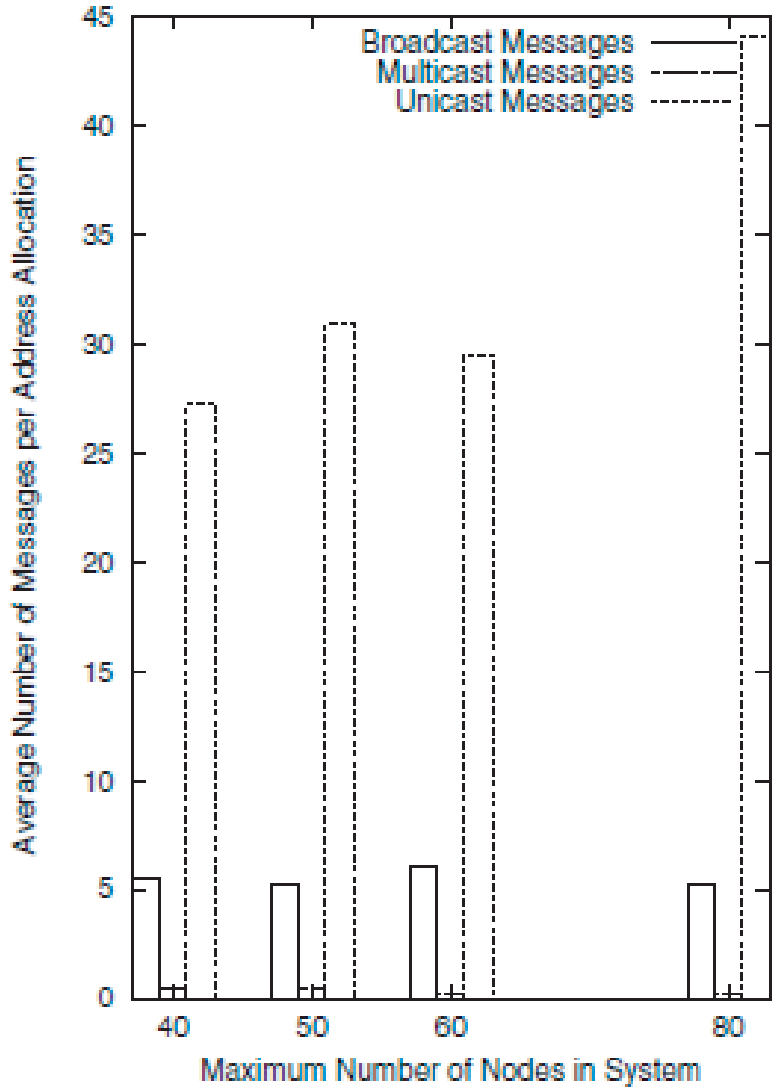

図 3.6　Number of Procotol Messages Exchanged in MANETconf

ノードの離脱　次にノードがネットワークから離脱する場合を考える．離脱するノードは $NETWORK\_LEAVE$ メッセージをブロードキャストする．$NETWORK\_LEAVE$ メッセージは離脱ノードのアドレスと，プールのサイズをもっている．離脱ノードの隣のアドレスをもっているノードがプールを回収し，プールは再利用される．

突然の離脱　ノードがクラッシュするなどして突然離脱した場合は次のようにしてアドレスを回収する．クラッシュしたノードはアドレスプールをもっていたが，ネットワーク中のそれ以外のノードはプールをもっていなかったとして考える．

ノードのクラッシュの後に新たなノード i が参加しようとして，すでに参加しているノード j に $ADDR\_REQ$ を送る．j はプールをもっていないのでアドレスの探索をするため $SEARCH\_ADDR$ をブロードキャストし，タイマー $SEARCH\_ADDR\_TIMER$ を開始する．$SEARCH\_ADDR\_TIMER$ がタイムアウトした後，j は受信した返答を確かめる．クラッシュのためすべてのノードからは返答が返ってきていないはずであるが，j はこの時点では

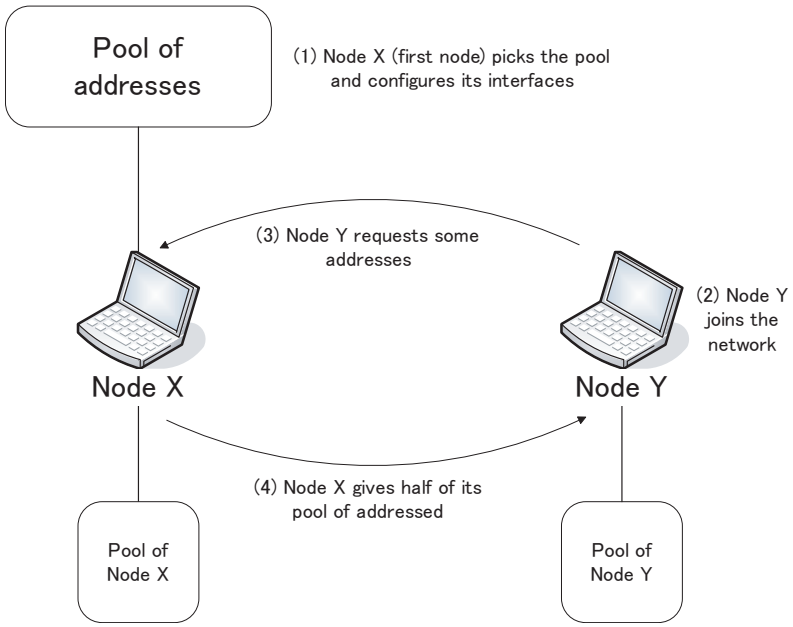

図 3.7　Conflict-free allocation scheme

メッセージが失われたり遅れたりしただけなのか，ノードがクラッシュしたのか区別することができない．そこで j は返答がなかったノードに対してのみ $SEARCH\_ADDR$ を再送する．$SEARCH\_ADDR\_THOLD$ 回再送を行っても返答がなかった場合，返答しないノードのアドレスは j によって解放され，そのうち 1 つが i に割り当てられ，$NODE\_CRASH$ メッセージがブロードキャストされる．$NODE\_CRASH$ メッセージはクラッシュしたノードのアドレスをもっている．クラッシュしたノードの隣のアドレスをもっているノードがクラッシュしたノードのアドレスを回収し，以後の割り当てで再利用される．

割り当て中の移動　新しく参加してアドレスを要求したノード i が，窓口となったノード j からアドレスの割り当てを受ける前に他のノード k の近辺に移動してしまった場合を考える．i は自身がノード k の近辺に来たことを検出し，k にメッセージを送る．このメッセージは j のアドレスを含んでいる．k はこのメッセージを受け取ると j と通信し，j から i にアドレスプールを転送できるようにする．

ネットワークの合併　ネットワークが合併したことを検出した場合は，全ノードが IP アドレス，プールのサイズ，デフォルトのプールのサイズ, ネットワーク識別子を含むメッセージをブロードキャストする．これで双方のネットワークのすべてのノードが，他方のネットワークのあらゆる情報を得ることになる．しかし，ネットワークの合併によってアドレスの衝突が起こる可能性がある．これは以下のようなアルゴリズムで解決する．

```
If（送信ノードのアドレス == 自分のアドレス）{
  If（自分の優先度 < 送信ノードの優先度）{
    自分のアドレス，アドレスプールを消去.
    新しくネットワークに参加するときと同じ方法でアドレスとプールを取得.
  }
}
```

```
If（自分のアドレスプールが他のノードのスターティングアドレスを含む）{
  自分のアドレスプールを
    自分のアドレスプールの末尾 + 1 == 他のノードのスターティングアドレス
  となるように縮小する.
}
```

割り当てにかかる時間と，ブロードキャストメッセージ数を計測するシミュレーションが行われている．CMU のモビリティパターンジェネレータを用いて作成されたパターンを使い，ノードの最大移動速度は $10m/s$，停止時間は $2.0s$，トポロジーは $670m \times 670m$ で測定時間は $1500s$ として実行された．

割り当てにかかる時間は図3.8 のようになった．ノード数が多くなると密度が上がり，割り当て中にタイムアウトするノードも少なくなるため割り当てにかかる時間も短くなっていることがわかる．しかし，ネットワーク中のノード数がプールのサイズに近くなると割り当てにかかる時間は増え始める．

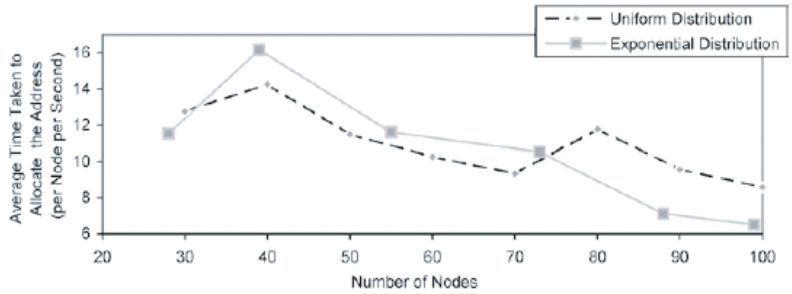

図 3.8　The average time taken to allocate the address for both type of interarrival times in Tayal

またブロードキャストメッセージの数は，$2.5 messages/node$ となり，これは MANETconf のほぼ 50% であった．

MANETconf では，ノードが参加するごとにブロードキャストメッセージを発行するので効率が悪いという問題があったが，このように Tayal らの方法ではブロードキャストメッセージ数をほぼ半分に減らすことができた．また Tayal らの方法の特徴として，IPv4 にも IPv6 にも適用でき特定のルーティングプロトコルには依存しない，ネットワークの分裂，合併に対応しているという点もあげられる．

3.3　時空間アドレス（Spatio-Temporal Address : STA）

前章で述べた位置情報サービスを実現するためにはアドレス中に位置情報を埋め込むことが有効である．そこで各端末の時空間情報を基本情報としてアドレスを算出する時空間アドレス（Spatio-Temporal Address：STA）という手法が提案されている．

STA もアドレッシングの一手法であるが，単に分散型ネットワークの各端末にアドレスを割り当てるという機能だけでなく，位置情報を考慮した特殊な手法であるため本節で取り上げ，以下概略を述べる．

3.3.1 STA

STA とは端末の存在する時刻・位置の情報を基本情報として用いるアドレスである（図.3.9）．時刻については各端末が持っている時計から，位置情報は GPS などの位置同定システムから取得可能である．すなわち STA は各端末が自律的に算出することができる．また理論的に同時刻・同位置に複数の物体が存在することはあり得ないので一意性が保証されている．

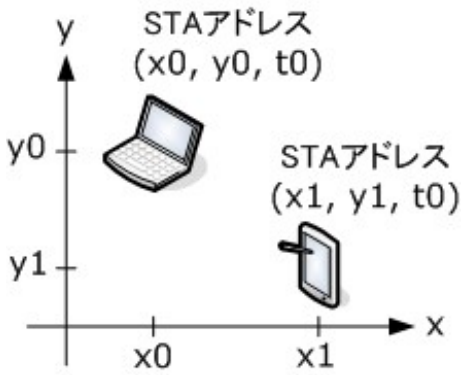

図 3.9　STA の概念

欠点としては，その一意性が時空間情報の粒度に依存することがあげられる．粒度は測定機器の性能にも依存するが，アドレスのデータサイズの影響も大きい．各端末の位置情報を緯度経度で表現する場合，仮に全地球上を 100m 粒度で表現する場合には緯度経度それぞれに関して 19 ビット程度が必要とされる．さらに高度に関しても考慮する場合，高度 1km まで物体が存在可能であるとすれば，高度も 1m 粒度で表現するためには 10 ビット程度が必要となる．さらに時刻情報として 1 日を 1 秒粒度で表現するためには 17 ビット程度が必要とされる．

しかしながら，たとえば位置情報の粒度が 100m であった場合には 150m 離れた端末では同一の位置情報をもつことはなく，時刻情報についても同様である．つまり粒度以上に離れていれば衝突はあり得ない．特に位置に関しては，通信半径より位置情報の粒度が詳細であるとすると，隣接端末間のみで衝突検知を行えばよく，常にネットワーク全体に対して衝突検知を実行する必要のある IP アドレスと比べて大幅にオーバーヘッドを削減することが可能となる．

さらにもう 1 つの欠点として，端末が移動すると共に，アドレスを変更しなければならないという点が挙げられる．この場合まず，自身と同じ STA を取得可能な範囲を確実に通信半径内に収め，アドレスの衝突検知を保証しなければならない．以下の式.3.1 で表される STA 利用領域（図 3.10）を定める．ここで (x_0, y_0) は STA$(x_0, y_0, *)$ が取得可能なグリッドの基点座標，CR は通信半径，LG は位置情報の粒度である．

$$
\begin{aligned}
&(x - x_0)^2 + (y - y_1)^2 \leq CR^2 \\
&(x \geq x_0 + \frac{LG}{2}, y \geq y_1 + \frac{LG}{2}) \\
&(x - x_0)^2 + (y - y_1 - LG)^2 \leq CR^2 \\
&(x \geq x_0 + \frac{LG}{2}, y \leq y_1 + \frac{LG}{2}) \\
&(x - x_0 - LG)^2 + (y - y_1)^2 \leq CR^2 \\
&(x \leq x_0 + \frac{LG}{2}, y \geq y_1 + \frac{LG}{2}) \\
&(x - x_0 - LG)^2 + (y - y_1 - LG)^2 \leq CR^2 \\
&(x \leq x_0 + \frac{LG}{2}, y \leq y_1 + \frac{LG}{2})
\end{aligned}
\tag{3.1}
$$

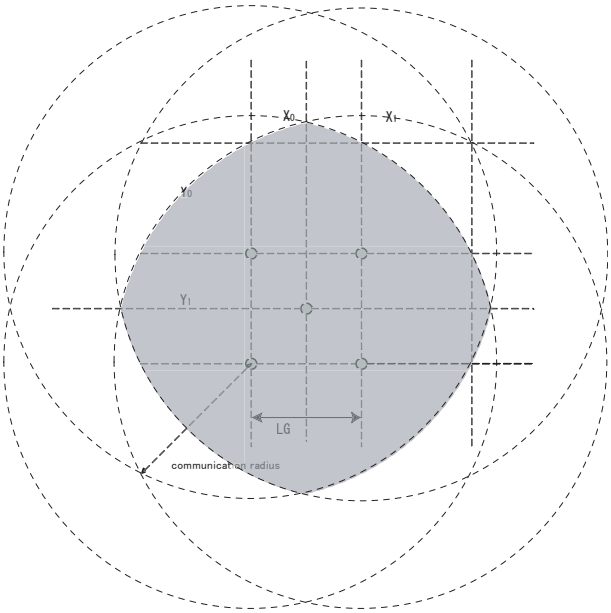

図 3.10　STA 有効範囲

ある STA を取得した端末はこの STA 利用領域からでた場合には自律的に STA を更新する．このため，実行中の通信が切断されてしまうおそれがある．

この問題に対しては，新旧のアドレス間でハンドオーバ処理をすることで解決可能である．STA の更新を行う前に通信相手との交渉を追加することにより，パケットロスを軽減する．ここで交渉とは STA 更新を行なう端末が通信相手に新規 STA を適用する前に STA の更新を通知することを意味する．新規の STA は通信相手が更新を了承した後に適用されるため，既に廃棄された STA に対してパケットが誤送信されることを回避することができる．フローチャートを図 3.11 に示す．

ただしこの交渉を追加した場合遅延が発生することは回避できない．データパケットを制御することにより遅延を最小限に留める手法の検討が必要となる．

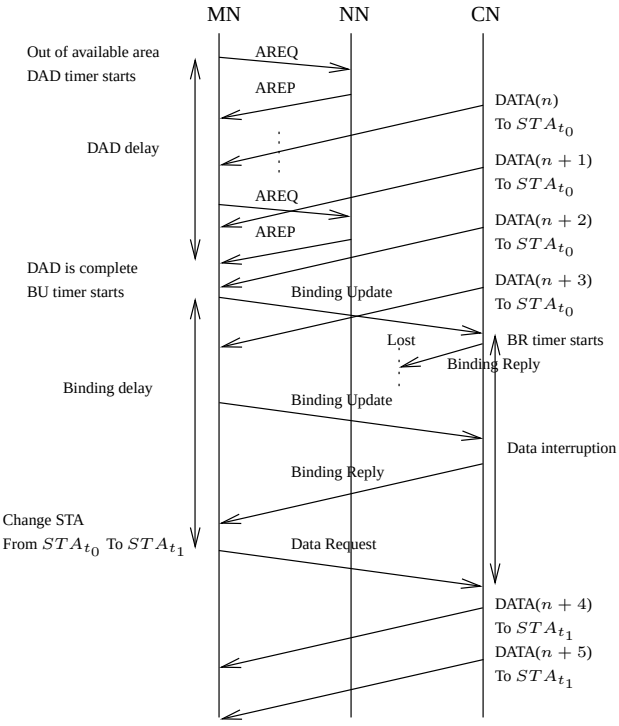

図 3.11　Time sequence of Mobile Node (MN), Neighbor Node (NN) and Correspondent Node (CN)

3.4　まとめ

位置情報を用いたルーティング手法はいくつかの提案がなされており現在主に計算機シミュレーションによる評価が進められている．

以上で述べた他にも分散型ネットワーク上での動作を考慮したアドレッシング機構は多数存在する．しかし，アドレス衝突を後から検出する方法，アドレス衝突を起こさないようにする方法の 2 つに大別することができ，あとはネットワークの分割/合併を考慮しているかなど細かな差異にとどまっている．[15] にあげたように，ここ数年 IETF でも種々のプロトコルの比較調査を行い Internet Draft として発行されるなど，標準化に向けての作業が進んでいるようである．単に分散型ネットワーク上の各ノードに自動的にアドレスを割り当てるという課題に対しては，解決策が出そろってきたといって良さそうである．

第 4 章

提案するシステムの設計

本章ではまず STA の具体的な実現方法を提案し，その後それを実装するシステムの設計について述べる．STA の割り当ては STA Management Daemon というプログラムで行うが，今回，当研究室において提案，実装を行った位置情報交換・高精度化システムの全体像についてもここで概説する．

4.1 STA の実現方法

4.1.1 STA を実現する層

STA を実現する際，(1) アプリケーション上の仮想的なアドレスとして実現する方法と，(2) 純粋にネットワークアドレスとして実現する方法とが考えられる．(1) の方法をとった場合，純粋なネットワークアドレスは従来通りの IP アドレスを利用することになると考えられ，ルーティングプロトコルは位置情報と関係ないものを使うことになる．そのため位置情報ルーティングをしようとするときには位置からアドレスへの解決が必要になる．我々が目標としている防災支援などのサービスでは臨時に構築されたネットワークを使用することが想定され，全端末の位置を既知とするのは非現実的である．一方，(2) の方法は位置とアドレスの対応を考える必要がなくなり，ルーティングの際に位置情報を利用することが自然な形で可能になる．従って，本研究では (2) の方法を採用することとした．

4.1.2 STA に必要なビット数

RFC 3513[23] によれば，IPv6 アドレスの後半 64 ビットの Interface ID は必ずしも MAC をもとにしたものでなくてもよいことになっている．グローバル ユニキャスト形式のアドレスの場合，さらにその直前の Subnet ID 16 ビットもネットワーク管理者が自由に決めてよいとされている．すなわち合計で 80 ビットは新しい方法で自由に定めることができる．

ここで STA に必要なビット数について考える．まず緯度経度方向であるが，1 章・2 章で述べたサービスを実現するための要求事項より少なくとも数 m の粒度が必要であるという点，また [20] のシミュレーション結果より位置情報の粒度を精細にすることでアドレス重複を効率よく抑えられるという点から，$1m$ 粒度とする．地球半径を $6378km$ とすると，$\log_2(2 \times 6378 \times \pi \times 10^3) = 25.25\cdots$ より緯度経度それぞれ 26 ビットとなる．次に時刻については 1 日を 10 秒粒度で表現することとした場合には $\log_2(24 \times 60 \times 60/10) = 13.07\cdots$ より 14 ビット必要となる．自由に定めることができるのは

80 ビットであったから残りを高さ方向に割り振るとして 14 ビットである．これはたとえば +9000m 〜-11000m までを 2m 粒度で表現することができる．$\log_2(20000/2) = 13.28\cdots$ だからである．

以上のことをまとめると，IPv6 アドレス後半の 80 ビットを用い，緯度経度それぞれに 26 ビットを割り当てて 1m 粒度で表現，高さに 14 ビット割り当てて 2m 粒度で表現，時刻にも 14 ビット割り当てて 10 秒粒度で表現することが可能である．全地球を表現することが可能であると確認できたので，上記の通りのビット数で，以下の図.4.1 のように IPv6 アドレス中に STA を埋め込んで表現することとした．

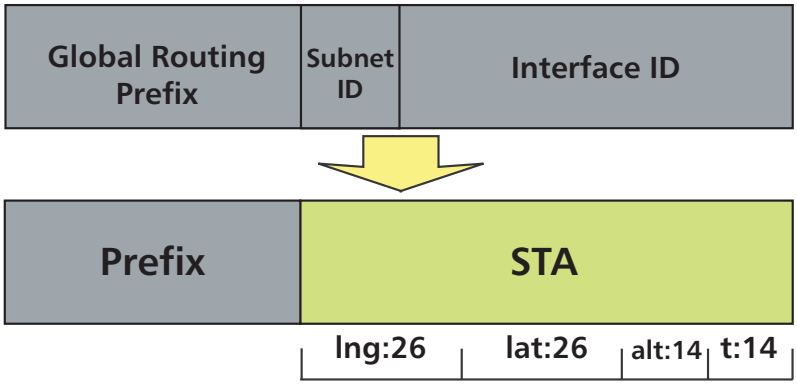

図 4.1　IPv6 への STA の埋め込み

4.1.3　STA のプレフィックス

上図.4.1 で示した IPv6 アドレスの前半 48 ビットは地域レジストラなどが取得し，一般的にはプロバイダなどから自動的にユーザーに設定されるものである．STA 埋め込み IPv6 アドレスにおいては，他の一般の IPv6 アドレスと STA 埋め込み IPv6 アドレスを区別するため，このような通常の自動設定は用いず，STA 埋め込み IPv6 アドレス独自のプレフィックスを使うことが必要である．将来的には正式な規格として成立させることが望ましいが，本研究の範囲では通常使用されていないプレフィックス（2007:dead:beef）を設定することとした．

4.1.4　STA の割り当て単位

一般的に IP アドレスは，端末ごとではなく，端末のインターフェースごとに割り当てられるようになっている．1 つの端末に複数のインターフェースが存在する場合，複数の IP アドレスが割り当て可能である．しかし STA の場合，端末ごとに 1 つのアドレスを割り当てるべきであり，複数のインターフェースが存在する場合でも単一の STA を用いるべきである．なぜならば STA は端末の存在する位置と時刻から算出されるアドレスであり，同一端末上の複数のインターフェースの位置のずれは，上で定義した STA の位置の粒度（1m）に比べて無視できるほど小さいからである．

今回行った実装では，各端末の無線 LAN インターフェースが 1 つだけであったため，このインターフェースに対して STA を割り当てることで，上記の要求を満たすこととした．複数の無線 LAN インターフェースが存在する場合への対応は今後の課題とする．

4.2 位置情報交換・高精度化システムの全体像

本研究は当研究室で進めている位置情報交換・高精度化関連のプロジェクトの一環として行われている．そのため他研究テーマの成果と協調して動作することが必要不可欠である．当研究室の研究プロジェクトにおいて開発したシステムの全体像を以下の図 4.2 に示す．これは 1 台の端末上に搭載するシステムを示している．

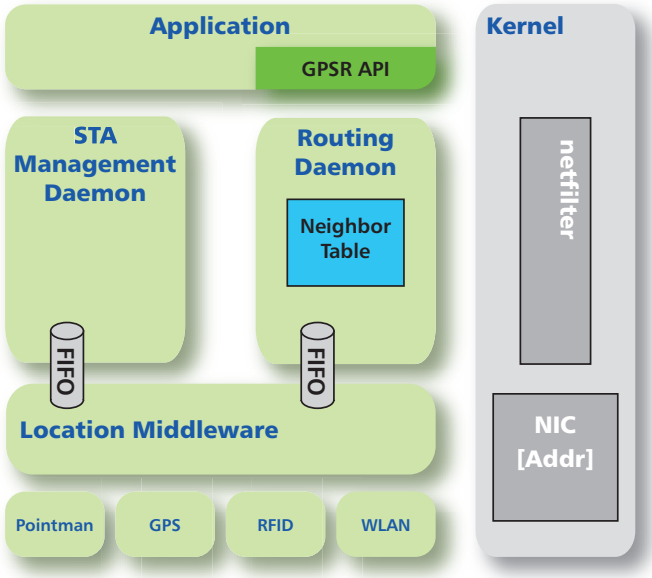

図 4.2 システムの全体像

まず測位デバイスの代表例として，Pointman DRM，GPS，RFID，無線 LAN を図中左下に示した．本システムではこのように，さまざまな測位デバイスからの入力を並列的に受け付け可能なよう設計した．さまざまな規格の位置データに対応し，並列的な受け付けを可能とするため，またこれらデバイスからの位置情報をさらに高精度化するため，Location Middleware と名付けたミドルウェアを測位デバイス直上に配置した．

測位デバイスはデータの入力方法も RS-232C 経由，PC カードを使いドライバソフトウェア経由で入力するものなど多様であり，データの規格も GPS の NMEA，RTCM，RINEX などをはじめとして多様である．Location Middleware はこれらの差異を吸収し，緯度経度をもとにした統一的・標準的なデータに変換して他ソフトウェアに受け渡す．また Routing Daemon の機能を利用して他端末と相互の位置情報の交換を行い，位置情報の高精度化を行う．

4.3 STA Management Daemon

4.3.1

4.3.2

第 5 章

提案するシステムの実装

本章では，データ構造，関数群など提案システムの詳細について説明する．ただし以下で詳述するのは本研究の主たるターゲットである時空間アドレス割り当て機構 STA Management Daemon に限ることとする．今回は位置情報に基づいたルーティング機構としては Greedy Perimeter Stateless Routing : GPSR[18] を選択し，University of Southern California の Embedded Networks Laboratory による Linux への実装 [19] を改造して用いることとした．

5.1 データ構造

5.1.1 構造体 _arep_flag_reserved

AREP パケットの 16 から 31 ビット目，1 ビットの AREP_FLAG と 15 ビットの予約領域からなる．

構造体メンバ

unsigned _arep_flag_reserved::arep_flag

AREP フラグ．重複かどうかを示す．

unsigned _arep_flag_reserved::reserved

予約領域．現在は未使用．

5.1.2 構造体 _PositionOut

ミドルウェアからの出力を格納する．

構造体メンバ

long unsigned int _PositionOut::index

出力データの index．

int _PositionOut::nodeid[16]

ノード ID.

time_t _PositionOut::time

データを得た時刻.

double _PositionOut::lat

緯度.

double _PositionOut::lon

経度.

double _PositionOut::alt

高度.

matrix_t _PositionOut::error

測位の誤差を表現するための分散共分散行列.

double _PositionOut::radio_range

無線半径.

5.1.3 構造体 _pthreadarg_addr_buf

recv_from_udp_child 関数の引数を格納する構造体．送信元アドレスとデータが格納されているバッファのアドレス.

構造体メンバ

struct sockaddr_storage _pthreadarg_addr_buf::fromaddr

送信元アドレス

char* _pthreadarg_addr_buf::buf

受信バッファのアドレス

int _pthreadarg_addr_buf::usedlen

受信バッファの長さ

5.1.4 構造体 _temporary_address_status

入力された位置情報に基づき生成したがまだ DAD の結果が出ず，割り当ては完了していない仮のアドレスを格納しておく構造体.

構造体メンバ

struct sockaddr_in6 _temporary_address_status::address

仮のアドレス.

time_t _temporary_address_status::generated_time

生成された時刻.

pthread_mutex_t _temporary_address_status::mutex

この構造体をロックするために使用する mutex.

address_status _temporary_address_status::flag

重複しているかどうか．0 で重複なし，1 であり.

5.1.5 構造体 _time_event_t

タイマーイベントの情報を格納する構造体.

構造体メンバ

int _time_event_t::timer_id

タイマー ID

int _time_event_t::duration

持続期間.

void(* _time_event_t::func)(void)

起動する関数へのポインタ

int _time_event_t::status

タイマーの状態．0 でオフ，1 でオン．

pthread_mutex_t _time_event_t::timer_mutex

この構造体をロックするために使用する mutex.

pthread_cond_t _time_event_t::timer_cond

持続期間が切れるか手動でオフされたときにイベントを検知するための状態変数.

5.2 ファイルとその関数群

プログラムを構成する主要なファイルと，それぞれの中で定義されている関数について述べる．

5.2.1 sta_timer.c

pthread を使ってタイマーを実現するための処理群.

依存関係は図 5.1 のようになっている．

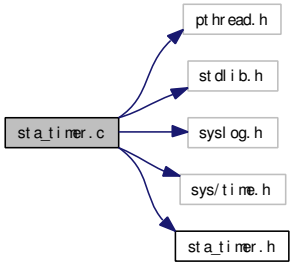

図 5.1 sta_timer.c のインクルード依存関係図

関数

- static void * thread_timer_on (void *arg)
  タイマー

- void timer_on (int timer_id, void(*func)(void), int duration)
  タイマーを起動する

- void timer_off (int timer_id)
  タイマーを停止する

5.2.1.1 static void * thread_timer_on (void * *arg*) [static]

pthread_cond_timedwait を使って ts だけ待つタイマー．切れると t_event->func を起動する．

引数:

arg time_event_t 型の構造体

戻り値:

NULL を返す

5.2.1.2 void timer_off (int *timer_id*)

経過時間に関係なく指定したタイマーを停止する．

引数:

timer_id 停止するタイマーの ID

5.2.1.3 void timer_on (int *timer_id*, void(*)(void) *func*, int *duration*)

指定したタイマーを起動する．持続期間，タイマーが切れたときに起動する関数も指定する．

引数:

timer_id 起動するタイマーの ID
func 切れたときに起動する関数
duration 持続期間

関数の呼び出しグラフ:

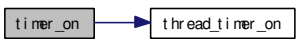

図 5.2 sta_timer.c 関数の呼び出しグラフ

5.2.2 sta_timer.h

pthread を使ってタイマーを実現する処理群，ヘッダファイル．

どのファイルから直接，間接的にインクルードされているかを以下のグラフで示す．

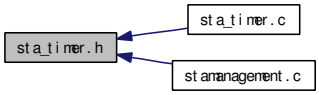

図 5.3　sta_timer.h のインクルード依存関係図

データ構造

- struct _time_event_t

マクロ定義

- #define MAX_NUM_TIMER 4

型定義

- typedef _time_event_t time_event_t

変数

- time_event_t sta_timers [MAX_NUM_TIMER]

5.2.2.1　マクロ定義

[

MAX_NUM_TIMER]#define MAX_NUM_TIMER 4 タイマーの数

5.2.2.2　型定義

[

time_event_t]typedef struct _time_event_t time_event_t タイマーイベントの構造体

5.2.2.3　変数

[

sta_timers]time_event_t sta_timers[MAX_NUM_TIMER] タイマーの配列

5.2.3 stamanagement.c

STA Management Daemon STA のセット，更新などを行うデーモンプログラム．

stamanagement.c のインクルード依存関係を図 5.4 に示す．

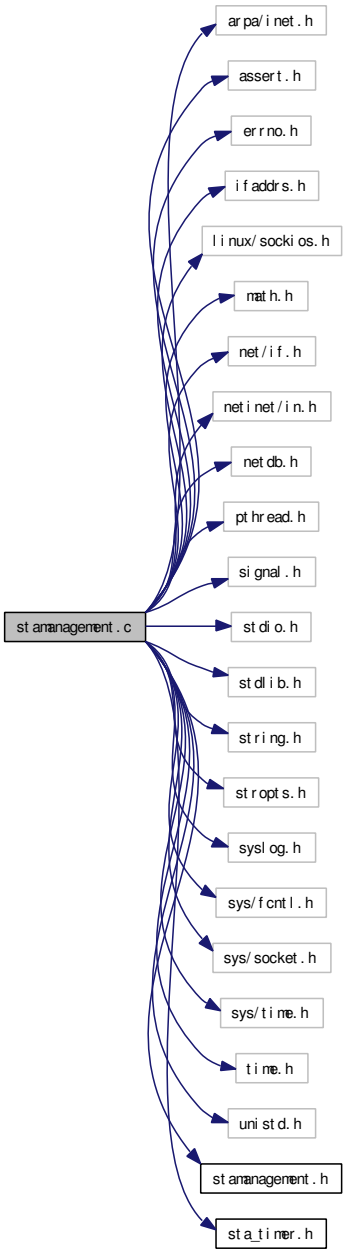

図 5.4 stamanagement.c のインクルード依存関係図

データ構造

- struct in6_ifreq
  in6_ifreq

関数

- static int in6_addr_equal (const struct in6_addr ∗a, const struct in6_addr ∗b)
  2 つの *v6* アドレスが等しいか調べる

- void ∗ recv_from_fifo (void ∗arg)
  FIFO からの受信

- static int encode_to_sta (PositionOut po, struct in6_addr ∗newsta)
  ミドルウェア出力から *STA* に変換する

- static int decode_from_sta (struct in6_addr ∗sta, PositionOut ∗po)
  STA から *PositionOut* に変換する.

- static int add_sta (struct sockaddr_in6 ∗newsta)
  STA を *add* する.

- static int delete_sta (struct sockaddr_in6 ∗oldsta)
  古い *STA* を *del* する.

- static int allocation_request_start (struct sockaddr_in6 newsta)
  AREQ をブロードキャストして *WT* 待つ

- static void allocation_request_timeout ()
  AREQ ブロードキャスト後のタイムアウト処理

- static int is_inside_valid_range (const PositionOut ∗const real, const PositionOut ∗const decoded)
  STA 有効範囲内にあるかどうか判定する.

- static double lat2y (double lat)
  緯度を *m* 単位に変換する

- static double lon2x (double lon, double lat)
  緯度 *lat* での経線 *lon* 度分の長さを *m* 単位で求める.

- static int get_socket_for_afinet6 ()
  AF_INET6 用の *socket* を作成して *fd* を返す.

- static int init_udp_socket (pthread_t recv_from_udp_thread_id)
  DAD のための *UDP* ソケットを初期化

- void init_temporary_address_status ()

仮アドレスの構造体を初期化

- void * recv_from_udp (void *arg)
  DAD のための *UDP* 受信処理

- void * recv_from_udp_child (void *arg)
  DAD のための *UDP* 受信，子スレッド

- static int packet_type_is_areq (char *buf)
  AREQ かどうか判定する

- static int packet_type_is_arep (char *buf)
  AREP かどうか判定する

- static int is_duplicate (char *buf)
  重複ありかなしか判定する

- static void sigaction_handler (int sig, siginfo_t *si, void *context)
  シグナルハンドラ

- static void init_parameters ()
  グローバル変数の初期化

- static void usage ()
  使用法説明

- int main (int argc, char **argv)
  メイン関数

5.2.3.1 static int add_sta (struct sockaddr_in6 * *newsta*) [static]

ネットワークインターフェースに新しい STA を割り当てる.

引数:

newsta 新しい STA

戻り値:

0 成功
-1 失敗

関数の呼び出しグラフ:

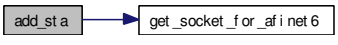

図 5.5 stamanagement.c add_sta 関数の呼び出しグラフ

5.2.3.2 static int allocation_request_start (struct sockaddr_in6 *newsta*) [static]

AREQ(Allocation REQest) パケットを無線半径内にブロードキャストして WT 秒待つ.

戻り値:

0 重複なし
1 重複しているとの返答あり

関数の呼び出しグラフ:

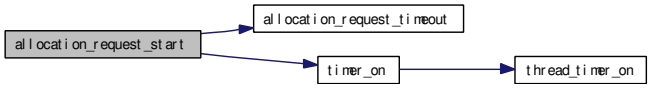

図 5.6 stamanagement.c allocation_request_start 関数の呼び出しグラフ

5.2.3.3 static void allocation_request_timeout (void) [static]

AREQ をブロードキャストしたあとタイムアウトしたときの処理. 重複の返答がなければアドレスを確定, または重複していたら何もせず単に抜ける.

戻り値:

0 常に 0 を返す

5.2.3.4 static int decode_from_sta (struct in6_addr * *sta*, PositionOut * *po*) [static]

STA から PositionOut に逆変換する.

引数:

← *sta* 変換元 STA
→ *po* 変換先 PositionOut

戻り値:

0 成功
-1 失敗

5.2.3.5 static int delete_sta (struct sockaddr_in6 * *oldsta*) [static]

ネットワークインターフェースの古い STA を削除する.

引数:

oldsta 削除する古い STA

戻り値:

0 成功
-1 失敗

関数の呼び出しグラフ:

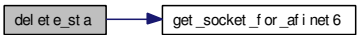

図 5.7 stamanagement.c allocation_request_start 関数の呼び出しグラフ

5.2.3.6 static int encode_to_sta (PositionOut *po*, struct in6_addr * *newsta*) [static]

FIFO 経由で受け取ったミドルウェア出力から STA に変換する.

引数:

← *po* ミドルウェアからの出力
→ *newsta* 変換した STA をいれて返す

戻り値:

0 成功
-1 失敗

5.2.3.7 static int get_socket_for_afinet6 () [static]

AF_INET6 用の socket を作成してそのファイルディスクリプタを返す. net-tools 内の lib/af.c, lib/sockets.c などを参考にした.

戻り値:

作成したファイルディスクリプタ

5.2.3.8 static int in6_addr_equal (const struct in6_addr * *a*, const struct in6_addr * *b*) [static]

2 つの v6 アドレスが等しいか memcmp を使って調べる. winpcap を参考にした.

引数:

a 比べるアドレスその 1
b 比べるアドレスその 2

戻り値:

1 2 つが等しい
0 異なる

5.2.3.9 static void init_parameters (void) [static]

各種パラメータを保存しているグローバル変数を初期化する．

5.2.3.10 void init_temporary_address_status (void)

仮アドレスの構造体を初期化する．

5.2.3.11 static int init_udp_socket (pthread_t *recv_from_udp_thread_id*) [static]

DAD の結果受信用の UDP ソケットを初期化する．

戻り値:

-1 失敗
0 成功

関数の呼び出しグラフ:

図 5.8 stamanagement.c init_udp_socket 関数の呼び出しグラフ

5.2.3.12 static int is_duplicate (char * *buf*) [inline, static]

AREP パケットのフラグを見て重複ありかなしか判定する．ただし，どちらとも判定できない場合は abort する．

引数:

buf パケットへのポインタ

戻り値:

0 重複なし
1 重複あり

5.2.3.13 static int is_inside_valid_range (const PositionOut *const *real*, const PositionOut *const *decoded*) [static]

自分が STA 有効範囲内にいるかどうか判定する.

引数:

real 判定に使う最新の現在位置．Location Middleware より FIFO 経由で受け取ったものである.
decoded 現在使用している STA から逆算したグリッドの基点の位置.

戻り値:

1 範囲内
0 範囲外

関数の呼び出しグラフ:

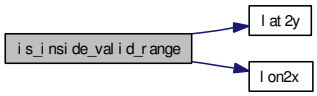

図 5.9 stamanagement.c is_inside_valid_range 関数の呼び出しグラフ

5.2.3.14 static double lat2y (double *lat*) [inline, static]

緯度 1 度=約 111km であることを利用して，緯度の差分を y 方向の m 単位の長さに変換する.

引数:

lat 緯度方向の差分

戻り値:

m 単位の長さ

5.2.3.15 static double lon2x (double *lon*, double *lat*) [inline, static]

緯度 lat における経線 lon 度分の長さを m 単位で求める.

引数:

lon 変換元経線方向の差分
lat 計算する地点の緯度

戻り値:

m 単位の長さ

5.2.3.16 int main (int *argc*, char ∗∗ *argv*)

メイン関数．コマンドライン引数に応じてパラメータを設定し，自分をデーモン化，データ受信スレッドを起動する．

引数:

argc コマンドライン引数の数
argv コマンドライン引数の配列

戻り値:

0 0 を返す

関数の呼び出しグラフ:

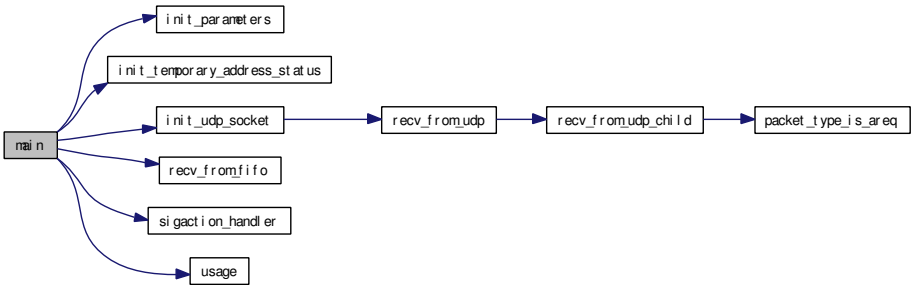

図 5.10 stamanagement.c main 関数の呼び出しグラフ

5.2.3.17 static int packet_type_is_arep (char ∗ *buf*) [inline, static]

パケットが AREP かどうか判定する．

引数:

buf パケットへのポインタ

戻り値:

0 AREP 以外
1 AREP

5.2.3.18 static int packet_type_is_areq (char ∗ *buf*) [inline, static]

パケットが AREQ かどうか判定する．

引数:

buf パケットへのポインタ

戻り値:

0 AREQ 以外
1 AREQ

5.2.3.19 void* recv_from_fifo (void * *arg*)

ミドルウェアから FIFO 経由でデータを受信する．別スレッドで実行される．

引数:

arg 実質使われていない

戻り値:

0 0 を返す

5.2.3.20 void* recv_from_udp (void * *arg*)

DAD の結果を待ち受ける処理．スレッドで実行される．受信だけしてすぐにさらなる子スレッドに処理を任せる．自分がスタータの場合は DAD の返答を受け取り，リゾルバの場合は AREQ を受け取る．

関数の呼び出しグラフ:

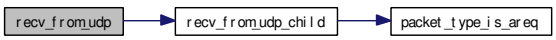

図 5.11 stamanagement.c recv_from_udp 関数の呼び出しグラフ

5.2.3.21 void* recv_from_udp_child (void * *arg*)

DAD の実際の処理を受け持つ子スレッド．自分がスタータの場合は DAD の返答を受け取り，リゾルバの場合は AREQ を受け取る．

関数の呼び出しグラフ:

図 5.12 stamanagement.c recv_from_udp_child 関数の呼び出しグラフ

5.2.3.22 static void sigaction_handler (int *sig*, siginfo_t * *si*, void * *context*) [static]

SIGINT などをキャッチするハンドラ．

引数:

sig Signal Number

si シグナル情報構造体
context ユーザーコンテキスト構造体

5.2.3.23 static void usage (void) [static]

使用方法の説明，コマンドライン引数の説明を表示して終了する．

5.2.4 stamanagement.h

STA Management Daemon STA のセット，更新などを行うデーモンプログラムのヘッダファイル．

どのファイルから直接，間接的にインクルードされているかを以下のグラフで示す．

図 5.13 stamanagement.h のインクルード依存関係図

データ構造

- struct _PositionOut
- struct _pthreadarg_addr_buf
- struct _temporary_address_status
- struct _arep_flag_reserved

マクロ定義

- #define FIFOPATH "/tmp/sta.fifo"
- #define WLAN_INTERFACE "ath0"
- #define WAITING_TIME 10
- #define UDP_PORT_NUMBER 5003
- #define UDP_RECV_BUF_SIZE 512
- #define IN6ADDR_MC_LINKLOCAL_INIT { { { 0xff,0x02,0,0,0,0,0,0,0,0,0,0,0,0,0,0x1 } } }
- #define AREQ_PACKET_SIZE 160
- #define AREP_PACKET_SIZE 160
- #define UNUSED(x) ((void)(x))
- #define IN6_IS_ADDR_STA(a)

型定義

- typedef enum _address_status address_status
- typedef enum _packet_type packet_type
- typedef double matrix_t [4]
- typedef _PositionOut PositionOut
- typedef _pthreadarg_addr_buf pthreadarg_addr_buf

- typedef _temporary_address_status temporary_address_status
- typedef _arep_flag_reserved arep_flag_reserved

列挙型

- enum _address_status { DAD, DUPLICATE, NOT_DUPLICATE }
- enum _packet_type { AREQ, AREP }

変数

- int daemonize = 1
- char fifo_path [256]
- char wlan_interface [5]
- int udp_port = 0
- int waiting_time = 0
- int sockfd
- temporary_address_status temp_address
- static struct in6_addr in6addr_linklocalmulticast = IN6ADDR_MC_LINKLOCAL_INIT
- static volatile sig_atomic_t srv_shutdown = 0

5.2.4.1 マクロ定義

#define AREP_PACKET_SIZE 160

AREP のパケットサイズ.

#define AREQ_PACKET_SIZE 160

AREQ のパケットサイズ.

#define FIFOPATH "/tmp/sta.fifo"

Location Middleware とのデータやりとりに使用する FIFO のデフォルトのパス.

#define IN6_IS_ADDR_STA(a)

値:

```
(((__const uint16_t *) (a))[0] == htons(0x2001)                  \
     && ((__const uint16_t *) (a))[1] == htons(0x200)                    \
     && ((__const uint16_t *) (a))[2] == 0)
```

プレフィックスをみて STA かどうか判定する． netinet/in.h にアドレスの種類判別用マクロがあるのでそれを参考にした.

#define IN6ADDR_MC_LINKLOCAL_INIT { { { 0xff,0x02,0,0,0,0,0,0,0,0,0,0,0,0,0,0x1 } } }

DAD のために使用する，IPv6 リンクローカルマルチキャストの宛先アドレス.

#define UDP_PORT_NUMBER 5003

DAD のために使用する，UDP のポート番号．GPSR の DEFAULT_DAEMON_PORT と DEFAULT_-OAM_PORT の次とした.

#define UDP_RECV_BUF_SIZE 512

UDP の受信バッファサイズ.

#define UNUSED(x) ((void)(x))

未使用の引数がある場合のコンパイルエラーを抑制するためのマクロ.

#define WAITING_TIME 10

DAD の待ち時間，秒単位.

#define WLAN_INTERFACE "ath0"

STA を割り当てる無線 LAN インターフェースのデフォルト値．今回は Atheros 社製チップ，madwifi の組み合わせを使用するので ath0 とする.

5.2.4.2 型定義

typedef enum _address_status address_status

DAD している途中，重複あり，重複なしの 3 状態を表現する.

typedef struct _arep_flag_reserved arep_flag_reserved

AREP パケットの 16 から 31 ビット目，1 ビットの AREP_FLAG と 15 ビットの予約領域からなる.

typedef double matrix_t[4]

自位置の存在範囲誤差を示すための分散共分散行列.

typedef enum _packet_type packet_type

AREQ，AREP などパケットのタイプを表現する.

typedef struct _PositionOut PositionOut

ミドルウェアからの出力情報を格納するための構造体.

typedef struct _pthreadarg_addr_buf pthreadarg_addr_buf

recv_from_udp_child 関数の引数のための型．メンバとして送信元アドレスとデータが格納されているバッファのアドレスとをもつ.

typedef struct _temporary_address_status temporary_address_status

入力された位置情報に基づき生成したが まだ DAD の結果が出ず，割り当ては 完了していない仮のアドレス.

5.2.4.3 列挙型

enum _address_status

アドレスの状態を表現するための列挙型．DAD している途中，重複あり，重複なしの 3 状態を表現する.

enum _packet_type

パケットのタイプを表現するための列挙型．AREQ，AREP などをあらわす.

5.2.4.4 変数

int daemonize = 1

デーモン化するかどうかを表すフラグ.

char fifo_path[256]

Location Middleware とのデータのやりとりに使用する FIFO のパスを保持する.

struct in6_addr in6addr_linklocalmulticast = IN6ADDR_MC_LINKLOCAL_INIT `[static]`

DAD のために使用する，IPv6 リンクローカルマルチキャストの宛先アドレス.

int sockfd

UDP 受信ソケットのディスクリプタ

volatile sig_atomic_t srv_shutdown = 0 [static]

SIGINT などの受信を検知したことを表すフラグ．シグナルハンドラ中でアトミックにセットされるようにするため volatile sig_atomic_t 型とする．

temporary_address_status temp_address

割り当て未完了状態の仮アドレス

int udp_port = 0

UDP ポート番号．DAD のために使用する．

int waiting_time = 0

DAD の待ち時間，単位は秒．

char wlan_interface[5]

STA を割り当てる無線 LAN インターフェース．

第 6 章

実験

6.1　実験システム概要

今回の実験では端末としては一般的なノート PC （Lenovo 社製 Thinkpad X60 [25]）を使用した．OS は Linux kernel 2.6，Community Enterprise Operating System（CentOS）4.4 [26] を使用した．無線 LAN は，802.11a/b/g に対応した内蔵の Atheros 社製チップを使用したものである．Linux で動作させるため Atheros チップ用ドライバである MadWifi [27] を使用した．GPSR 予備実験以外では，ログデータ収集のために syslog 関数を用いた．従来の syslogd はパフォーマンス上の問題がありログが取りこぼされることがあるため，syslog-ng [28] を用いることとした．

6.2　18 ノードでの歩行実験

2007 年 1 月 13 日（土），東京大学柏の葉キャンパス，総合研究棟 1 階部分（図.6.1）において，18 ノードを用い被験者に歩行してもらい、端末へのアドレス割り当て，位置情報の交換・高精度化を評価するための実験を行った．当日の天気は晴れであった．

6.2.1　実験方法

（1）擬似的な街路を設定し，その街路の範囲で自由に歩行してもらう場合，（2）広場をすべて使用して制約なく歩行してもらう場合，それぞれについて 7 分間ずつ行った．フィールド端にスタート地点を設定し，被験者は 1 人ずつフィールドに入るようにした．前の被験者がフィールド中央に達したタイミングで次の被験者が入ることとした．

6.2.2　結果

（2）の実験で，故障した 3 ノードを除き，残りの各 15 ノードが取得したアドレスにより表される座標と時刻を図.6.2 に示した．(1)，(2）ともにアドレスの重複は発生しなかった．またアドレス取得はほぼ 10 秒の周期で行われていることが観察された．

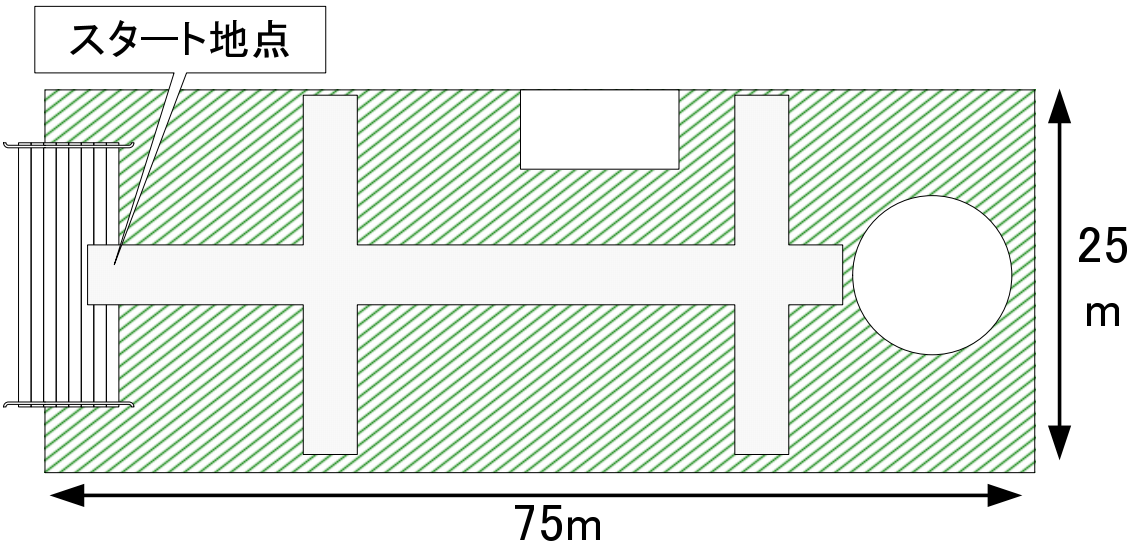

図 6.1　実験フィールド

6.2.3　考察

まずアドレスの重複が発生しなかったことについて，これは主に緯度経度方向の粒度を小さく (1m) したこと，ノード数がたかだか 18 程度であったことにより同一のタイミングでアドレス取得が起こらなかったと考えられる．アドレス取得の周期が約 10 秒周期だったことについて，これは重複検知のための待ち時間そのものである．すなわち，アドレスの取得を開始し，重複検知の待ちに入るが重複はおきていないため待ち時間の 10 秒だけ待ち，アドレスが確定される．その後 Location Middleware からの入力があるとすぐにその新たな位置入力に基づいてアドレスの取得を開始するのである．確定後すぐに再度取得を開始するのは，10 秒間歩くことにより有効範囲を脱出しているからと考えられる．ただし，STA 有効範囲は無線半径を用いて定義されるが，動的に半径を得ることは不可能であり固定的なパラメータを用いていることで計算が不正確になっている可能性がある．

6.3　2 ノードでの実験

2007 年 1 月 25 日（木），沖縄県国頭郡伊江村における電子情報通信学会アドホックネットワークワークショップ会場周辺にて，2 ノードおよび擬似的な電子タグを用いた実験を行った．2 人の被験者に別ルートをたどり，最終的に出会うよう歩いてもらった．端末へのアドレス割り当て，位置情報の交換・高精度化の評価を行った．当日の天気は晴れであった．

6.3.1　実験方法

会場玄関の位置情報リファレンス用擬似電子タグを用いて初期位置を補正した後，会場近くの海岸付近を約 10 分間自由に歩行した．海岸のある地点で出会い，互いの位置情報を交換して補正し，再び会

場に戻った．

6.3.2 結果

6.4 GPSR 予備実験

GPSR の Linux への実装では，位置情報はそれぞれの端末で起動時に手動で設定することになっている．今回 STA Management Daemon，Location Middleware と協調して動作させるため，位置情報は自端末の STA を参照することで取得するよう，また IPv6 での通信に対応するよう，当研究室において改造を行った．さまざまな問題により正確に性能を評価するには至らなかったが，改造後の GPSR Daemon が正常に動作していることを確認するための予備実験として以下のように GPSR を用いた通信を行った．

6.4.1 実験方法

University of Southern California による既存の GPSR と，今回改造した GPSR とを用い 1 ホップの環境において通信を行い，遅延，パケット配送率を測定した．本実験においては送信側・受信側ともに位置を固定とし，送信側からみて受信側のアドレスは既知であるとした．今回受信側からみて送信側のアドレスは未知であるため，パケットを返送することができず，Round Trip Time を測定することができない．そこで送信側端末・受信側端末の間で時刻同期を行い，同一パケットの受信時刻と送信時刻を比較して片道での遅延を測定することとした．時刻同期は NTP により行った．パケットサイズは 64 バイトとした．

6.4.2 結果

改造前，改造後どちらも配送率は 0.3 前後，遅延については 100 ms オーダの値となった．

6.4.3 考察

上述した結果には大きな誤差が含まれていると考えられ，正確な評価とはいえない．

誤差の理由として考えられる要因は様々ある．受信側から送信側にパケットを返送することができず，往復での測定ができないという制約から双方の端末で NTP による時刻同期を行い遅延を測定したが，この同期の誤差が遅延に影響している可能性がある．NTP による同期の誤差は一般に数 ms〜数十 ms オーダである．また，今回の実験では本予備実験に限り，syslog の仕組みを用いない通常のディスク上のファイルに対しログを出力したが，ログ出力の速度が通信に比べて遅く，正確にログを記録できていない可能性がある．

6.5 まとめ

柏の葉キャンパスにおけるこの無線半径内に約 20 ノードの密度（0.01 ノード$/m^2$）以下では緯度経度方向 1m 粒度，時刻方向 10 秒粒度で重複なく割り当てが可能である．例として東京都区部の人口密度は 0.013 人$/m^2$ であることから，小型センサを密に配置する場合や，商業地区に人・ノードが密集している場合などを除けば上記の粒度で重複なく割り当てられるといえる．

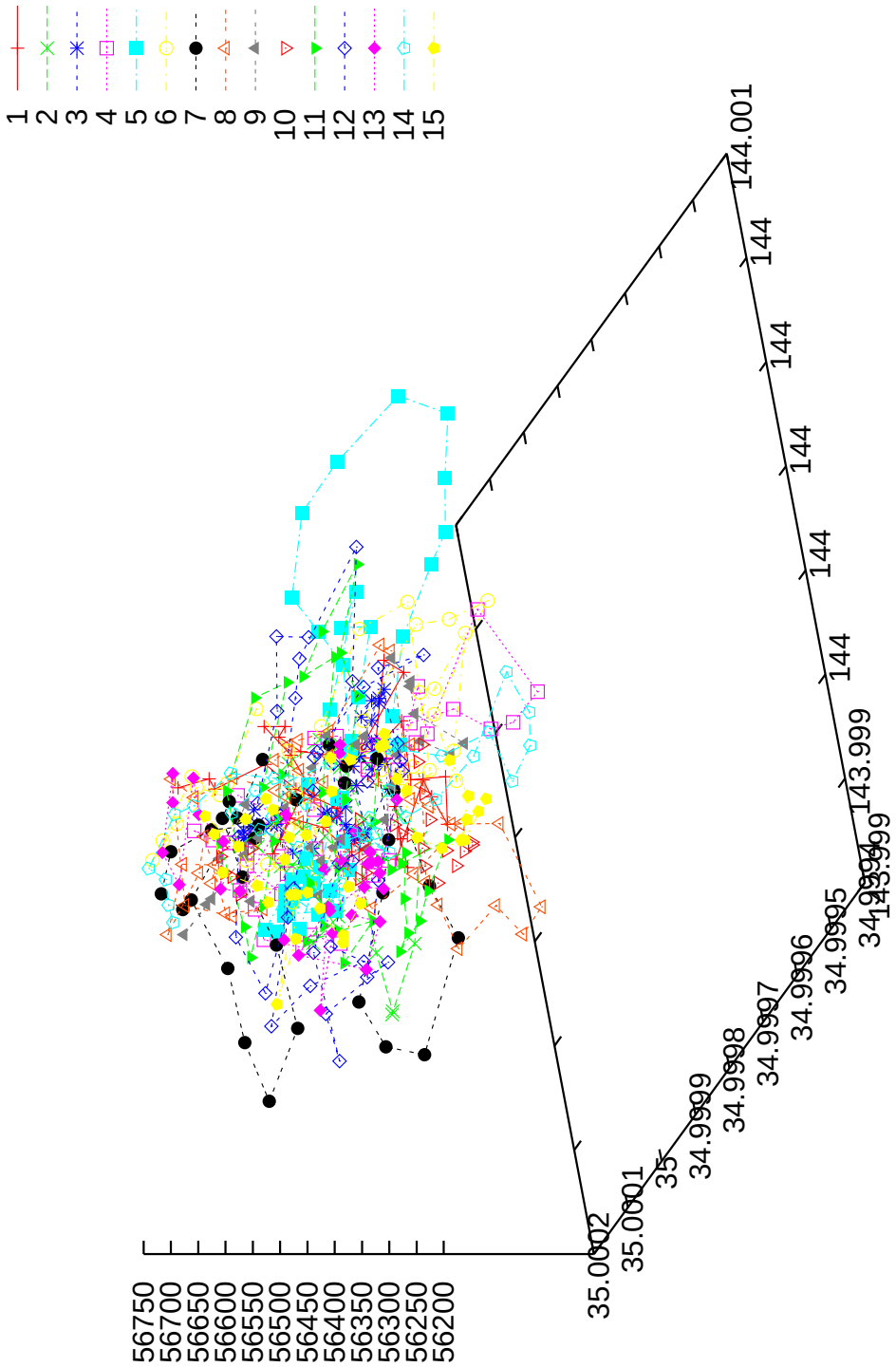

図 6.2　実験（2）で取得されたアドレスにより表される座標と時刻

第 7 章

結論

本研究を通じて得た結論と今後の課題について述べる.

7.1 結論

7.2 今後の課題

参考文献

[1] 電気通信事業者協会（TCA）,http://www.tca.or.jp/

[2] 社会実情データ図録,http://www2.ttcn.ne.jp/~honkawa/

[3] 総務省情報通信統計データベース,http://www.johotsusintokei.soumu.go.jp/

[4] HITACHI : ワイヤレスインフォベンチャーカンパニー,
http://www.hitachi.co.jp/wirelessinfo/airlocation/

[5] Sony Computer Science Laboratories, Inc., “PlaceEngine”, http://www.placeengine.com/

[6] Seigo ITO, Nobuo KAWAGUCHI, “Bayesian based Location Estimation System using Wireless LAN”, PerCom pp. 273-278, Mar. 2005

[7] Silicon Sencing Systems Japan, “Honeywell ポイントマン DRM”,
http://www.spp.co.jp/sssj/pointman3.pdf

[8] System and Data Research Co., Ltd., “Compact UrEDAS(コンパクトユレダス)”,
http://www.sdr.co.jp/uredas.html

[9] 明星電気株式会社, “ナウキャスト地震情報防災システム”,
http://www.meisei.co.jp/products/nowcast05.html

[10] 電子情報技術産業協会, “IT 自動防災システム”, http://home.jeita.or.jp/spp/index.html

[11] 国土交通省道路局 ITS ホームページ, “カーナビ・VICS の出荷台数”,
http://www.its.go.jp/ITS/j-html/topindex/work/topindex_navivics.html

[12] R. Droms, “Dynamic Host Configuration Protocol,” RFC 2131, March 1997

[13] S. Nesargi, R. Prakash, “MANETConf: configuration of hosts in a mobile ad hoc network,” Proceedings of the IEEE INFOCOM, 1059-1068, 2002

[14] Abhishek Prakash Tayal, L. M. Patnaik “An address assignment for the automatic configuration of mobile ad hoc networks,” Personal Ubiquitous Computing, 2004

[15] C. Bernardos, M. Calderon, “Survey of IP address autoconfiguration mechanisms for MANETs,” IETF Internet Draft, draft-bernardos-manet-autoconf-survey-00.txt , July 11 2005

[16] Josh Broch, David A. Maltz, David B. Johnson, Yih-Chun Hu, and Jorjeta Jetcheva, “A Performance Comparison of Multi-Hop Wireless Ad Hoc Routing Protocols”, MobiCom, 1998

[17] Young-Bae Ko and Nitin H. Vaidya, “Location-Aided Routing (LAR) in Mobile Ad Hoc Nerworks”, Proceedinf of Mobicom, 1998

[18] Brad Karp, H. T. Kung, “GPSR: Greedy Perimeter Stateless Routing in Ad HocWireless Networks”, Proceeding of Mobicom, 2000

[19] Embedded Networks Laboratory, “gpsr-linux”, http://enl.usc.edu/software.html

[20] 山崎浩輔, 瀬崎薫, “時空間での唯一性を利用したアドレッシングに関する一検討”，信学技報 NS6 月，2004

[21] Kosuke Yamazaki and Kaoru Sezaki, “Geographical Management of Mobile Ad Hoc Networks”, Ph.D. Thesis, University of Tokyo, 2005

[22] 山崎浩輔，瀬崎薫，“位置情報適応型サービスに向けた地理的経路制御手法の提案”，信学誌, 2002

[23] R. Hinden, S. Deering, “Internet Protocol Version 6 (IPv6) Addressing Architecture”, RFC 3513, 2003

[24] C-K. Toh, “アドホックモバイルワイヤレスネットワーク”, 共立出版, 2003

[25] Lenovo, “ThinkPad X60/X60s 製品情報”, http://www-06.ibm.com/jp/pc/notebooks/thinkpad/x-series/x60_features.shtml

[26] The CentOS Project, “The Community ENTerprise Operating System”, http://www.centos.org/

[27] MadWifi, “Multiband Atheros Driver for Wireless Fidelity”, http://madwifi.org/

[28] BalaBit IT Security, “Products BalaBit IT KFT”, http://www.balabit.com/products/syslog_ng/

発表文献

岡野 諭，瀬崎 薫．“時空間アドレスの応用に関する研究”．2006 年電子情報通信学会ソサイエティ大会 (2006-9).

岡野 諭，田中 隆浩，瀬崎 薫．“時空間アドレスを使った位置情報高精度化システムの展示”．2007 年電子情報通信学会アドホックネットワークワークショップ (2007-1).

田中 隆浩，岡野 諭，瀬崎 薫．“アドホックネットワークと電子タグを利用したシームレスな位置情報の取得”．2007 年電子情報通信学会アドホックネットワークワークショップ (2007-1).

岡野 諭，瀬崎 薫．“時空間アドレス割り当て機構の設計と実装”．2007 年電子情報通信学会総合大会，発表予定 (2007-3)

謝辞

修士 2 年間の研究を進めていく過程で直接ご指導いただき，さまざまな知識，研究に対する姿勢などをご教示下さり，また今年度は研究プロジェクトの立ちあげ，運営に尽力され，有益なご指導，ご批評，愛情に満ちた励ましの言葉をいただいた瀬崎 薫 助教授に感謝致します．

リサーチフェローの木實 新一氏

日常より研究環境の整備に尽力してくださった 小松 邦紀 助手，松本 夏穂 秘書，臼井 千佳子 秘書，小野 裕子 秘書に感謝致します．

東京電機大学の戸辺 義人教授には，お忙しい中，プログラミングを進める上で深夜にわたって大変熱のこもった直接のご指導をいただきました．戸辺研究室修士 1 年の石塚 宏紀君は，場所が離れているにもかかわらず，急な申し出にも迅速に対応して協力をしていただきました．お二方に深くお礼申し上げます．

修士 2 年の田中 隆浩君は本研究のすべての面において，また 2 年間の研究生活，その他諸々に渡って

本論文の作成にあたり，日頃から励ましていただいた博士 3 年の関根 理敏さん，魏 新法さん，博士 2 年の寺田 真介さん，角田 忠信さん，

最後に，科学技術振興調整費「電子タグを利用した測位と安全・安心の確保」に関わっておられる皆様に深く感謝致します．

2007 年 2 月 2 日