平成 26 年度　特別研究報告書

AODV ルーティングプロトコルの拡張による安定した通信経路構築の検討

龍谷大学　理工学部　情報メディア学科
T110458　内藤　奨

指導教員　三好　力　教授

内容概要

近年、無線通信技術の発達と端末の小型化によって、多くの人が携帯端末を持つようになった。無線通信可能端末が広く普及したため、どのような状況でも無線端末を利用したアドホックネットワークの構築が可能になっている。

アドホックネットワークとは、無線基地局などを介さずに端末間のみで構築するネットワークである。このことから、無線基地局の故障や大規模災害時のインフラが整っていない場合において有効な通信手段となる。

良質なアドホックナットワークの構築のためには均質なネットワーク、端末の電力消費、端末の移動によるネッワークの変更に伴うルーティングプロトコルなどの問題をクリアしなければならない。既存のルーティングプロトコルでは、通信端末の無線通信可能範囲内に存在する端末を選択し通信経路を構築するものが基本となっている。しかし、高速で移動している端末を選択してしまいすぐに通信が切れてしまう可能性があった。

本研究では、各端末間の移動速度を考慮することで安定した通信経路の構築ができるルーティングプロトコルの研究に取り組む。

目次

第1章　はじめに

1.1　研究背景

無線通信技術を利用した様々なデバイスの登場により、モバイルアクセスの時代の到達が感じられる。通信形態としてインターネットが主流となり、インターネットモバイルアクセスは、日々の生活に欠かせないものになっている。

無線通信端末の代表例は携帯電話であると考えられる。携帯電話は 2013 年度末で 1 億 4400 万件、日本の総人口に対する普及率は 112.5%にまで至っている。現在の携帯電話は、電話はもちろんのことメールや Web 検索などのパソコンで可能であったことがほぼ携帯電話の機能として搭載されている。ネットワークとしてみた携帯電話は、基地局と携帯電話間の1対1のネットワークであるが、Bluetooth 機能やデザリング機能を搭載した機種の登場により、新たなネットワーク利用の可能性が出てきている。

一般化している無線ネットワークは、無線基地局などのアクセスポイントを必要とするネットワークである。これに対し、無線基地局などのインフラを必要としないネットワークがアドホックネットワークである。アドホックネットワークは、人が持つ端末を経由し通信を確立するネットワークである。上記で示したように、携帯電話の普及率が 100%を超えほぼ全ての人が携帯電話を持つようになったことで、携帯電話を端末としたアドホックネットワークが形成しやすくなった。災害時など基地局に障害が発生した場合のネットワークとして活躍が期待できる。しかし、基地局を持たないネットワークということで解決しなければならない問題が多く残されている。

1.2　携帯電話の通信

携帯電話の通信では、携帯電話同士が直接通信するのではなく、携帯電話からの電波を最寄りの無線基地局に送信し、交換機や基地局など様々な通信設備を経由し宛先の最寄りの無線基地局から宛先の携帯電話に電波を届けている。互いの最寄り無線基地局から間に介在している交換機や基地局は有線または無線、衛星回線で繋がれている。

上記より、携帯電話での通信は各基地局が重要であることがわかる。基地局が機能しなくなってしまうと携帯電話での通信が不可能になる。通信不可になる状況を以下に示す。

- 過度のトラフィック集中による輻輳によって基地局の容量オーバー
- 無線基地局や交換機を繋ぐ有線ケーブルの切断
- 長時間の停電によって、無線基地局が稼働できなくなる場合

先に述べた有線ケーブルの切断や長時間の停電は、地震や台風などの自然災害時に起こる可能性が高い。その理由としては、携帯電話の通信回線の命である無線基地局のトラブルが発生する可能性が、地震や台風などの自然災害の多い日本の風土では低くないといことである。その根拠として、マグニチュード 6.0 以上の地震の発生件数の 2 割が日本で発生していることが挙げられる。携帯電話はそのような災害時にこそ通信を行い安否確認などができることが望ましい。

よって携帯電話の機能に、周囲に存在する携帯電話を無線端末として通信する無線ネットワークを実装するとどのような状況でも望む相手と通信することができるようになる。

1.3　問題点

1.1 節でも述べたように、アドホックネットワークには技術的な課題が残っている。その課題の 1 つにルーティングプロトコルの問題がある。現在主流となっているルーティングプロトコルは、メッセージを通信可能な全ての端末にブロードキャストし、そのメッセージを宛先端末に届けられるまで繰り返し通信経路を生成する。送信元端末から宛先端末までの経路を構築する際、無線通信可能範囲内の端末を端末の速度や位置情報などの情報なしに選択し経路を構築している。しかし、この手法では端末が高速で移動している場合にすぐに通信が切断されてしまう経路を構築してしまう可能性がある。

1.4　研究の目的

先に示したように、携帯電話の普及から携帯電話のような移動端末を用いたネットワークの構成が重要になると考えられる。

人が簡単に携帯できる移動端末でのネットワークの構築は日本という中で非常に重要であると考え、Reactive 型のネットワーク構築の性能向上を目的に研究を進めていく。

本研究では、経路構築の際に移動速度が低い端末を選択し経路構築を行うルーティングプロトコルを提案する。これにより、通信の切断が少ない安定したネットワークを構築することを目的とする。提案するルーティングプロトコルは、1 度構築した経路が切断されるまでの時間、送信パケットと受信パケットの損失率の 2 点から評価し、既存手法と作成したルーティングプロトコルの比較を行う。

第2章　既存手法

2.1　アドホックネットワーク

端末の無線通信可能範囲内にある端末と相互に通信し、端末同士を結んでいくことで任意の端末と通信するためのネットワークを構築できるものである。このことから、基地局が使用できない状況でも通信をするための手段となる。

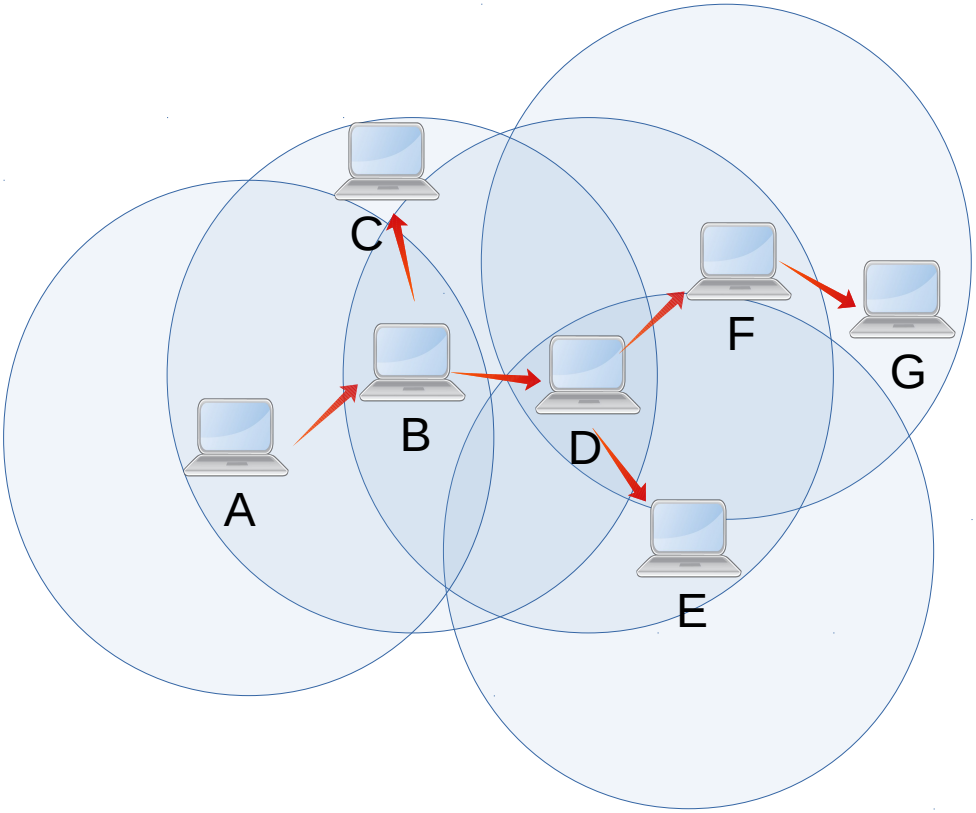

図1：無線通信の例

図1に示すのは無線通信の例である。アドホックネットワークは、図1のように各端末が無線通信可能範囲内にある他の端末と双方向の通信を行う。左端の端末 A が右端の端末 G と通信したい場合は、周りの端末に通信経路構築を要求し、端末 A から端末 G までの双方向の通信経路が構築される。

アドホックネットワークでは通信相手を探すための通信経路探索手続きである。そこで、ルーティングプロトコルが重要になってくる。また、各端末は移動速度に差があるものの常に移動していると考え、構築した経路が切断、切断後新しい経路を再構築できることが必要である。アドホックネットワークの利点を以下に示す。

- 耐障害性に強い
- 運用/管理が容易
- 端末同士が直接通信することで、電波を効率良く使える

アドホックネットワークは端末間で独自にネットワークを作成し通信を行うので、作成したネットワークの端末には一意のアドレスが付与される。上記などより、アドホックネットワークには以下のような技術的な課題が存在する。

- ネットワーク構築のためのルーティングプロトコル
- 通信切断時のネットワークの再構築
- ネットワークごとに一意のアドレスを用いる

2.2 ルーティングプロトコル

ワイヤレスアドホックネットワークにおいて、通信する際に送信元端末の無線通信可能領域に宛先端末が存在することは稀である。このことから通信元端末は、宛先端末のアドレス情報を含むパケットを送信し、宛先端末の場所を特定してから送りたいデータなどの送信を開始することになる。ルーティングプロトコルとはこの送信元端末と宛先端末を繋ぐ経路の構築を司る。

アドホックネットワークにおけるルーティングプロトコルは、どの端末を経由してマルチホップし、最も効率よくルートを構築できるか、というネットワークに欠かせない重要な技術である。

ルーティングプロトコルは、大きく”Reactive(On-demand)型”と”Proactive(Table-drive)型”の2つに分けられる。それぞれ以下のような特徴を持つ。

2.2.1 Reactive(On-demand)型

Reactive 型プロトコルは、端末から通信要求が発生した時に初めて経路を構築していく。通信経路は通信要求を行った端末の周囲に存在する端末を確認し、宛先端末までの経路表を作成して確立させる。Reactive 型プロトコルには DSR,AODV,ABR などの代表的なプロトコルが存在する。これら Reactive 型プロトコルのプロトコルには大きく 2 つの利点が存在する。

1. 常に最新の経路表を作成するので、端末が存在しないという状況に陥らない
2. 通信要求時以外は端末間の通信を行わないため端末の電力消費を抑えることができる

1.は、移動する携帯端末がネットワーク内に存在しており、時間と共に端末が通信不可の場所まで移動したとしても対応できる利点がある。2.は、端末の稼働時間が長くなるため、バッテリー切れによる端末の消失を減少させネットワーク全体が長時間使えるようになる利点がある。

一方で、通信要求が発生してから経路表の作成を開始するため端末間の本格的な通信を開始するまでに遅延が発生してしまう欠点も存在している。

2.2.2 Proactive(Table-drive)型

Proactive 型プロトコルは、通信を行う前に予め経路表を作成し、通信要求時は作成した経路表に従った経路を構築し通信を行う。1 度作成した経路表は、端末の移動やバッテリー切れによる端末の消失に対応するため定期的に更新される。 Proactive 型プロトコルには OLSR,DSDV,FSR などの代表的なプロトコルが存在する。

Proactive 型プロトコルの利点は、端末からの通信要求が発生した際ただちに通信を開始することができる点である。また欠点は、定期的に経路表を更新する際パケットを送受信し隣接端末の存在を確認するため、ネットワークへの通信不可が発生すること、ネットワーク端末のバッテリー消費が Reactive 型と比較し大きい点である。

2.3 AODV

AODV は各端末が自身の通信可能な端末の情報を得ておくことで、通信経路構築が可能である。また、Reactive 型という端末の電池消費を抑えたプロトコルなので端末に対する負担が低い点から携帯電話などの携帯端末に実装することで生活が便利になると考えられる。以上の理由から本研究では AODV のプロトコルに着目する。AODV と他の Reactive 型のプロトコルの特徴を以下に記す。

- DSR
  パケット転送は、ルーティングテーブルに従って行なわず、ソースルーティングという、パケットの発信元があらかじめ全体の経路を指定する方式を用いている。通信を行なうノード間のホップ数は、せいぜい数ホップの範囲を想定しており、数十ホップという長い経路は考えていない。
- ABR
  リンク切断が長時間起こらない，安定した経路を選択する．各ノードは一定間隔ごとに隣接端末へビーコンを送信する．より多くのビーコンを受信した端末からなるリンクは持続性が高いと期待できるため，安定した経路が生成できる。
- AODV
  各端末はシーケンス番号を管理し経路構築に積極的に利用している。各端末は非常に短い間有効な経路表を持ち、データパケットはそれを用いて転送される。各経路表のエントリーには precursor リストがあり、リンクに障害があったときに利用される。

AODV(Ad hoc On-demand DistanceVector)ルーティングプロトコルは、Reactive 型のルーティングプロトコルであるため、端末からの通信要求が発生した際に経路構築を行う。
また、構築する経路は常に1つであり構築した経路の通信が切断された場合に新たな経路を構築するため、経路構築動作をはじめから行う。

送信元端末と宛先端末間で RREQ（Read Request）と RREP（Read Replay）の 2 つのメッセージを使い通信経路の構築を行う。RREQ は送信元端末からブロードキャストにし、RREQ を受け取った中間端末は RREQ を他の端末にブロードキャストしていくことで宛先端末に到達させる。RREQ が到達すると宛先端末は、RREP を RREQ を送信してきた経路を通して送信端末に送信する。RREP が送信元端末に到達した時点で経路を確立する。複数の経路が存在する場合は、送信元端末が RREQ を送信してから RREP を受信するまでの時間が最も短い（ホップ数が少ない）ものが選択される。例を以下に示す。

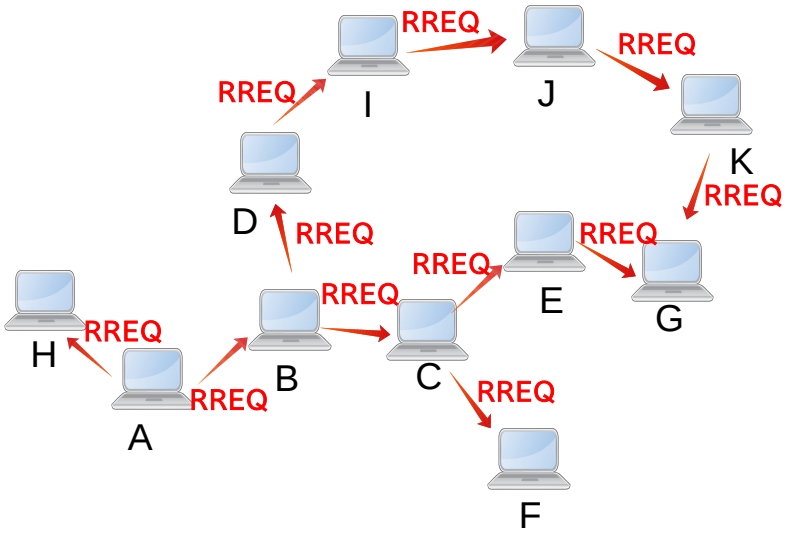

図 2：通信要求した際の動作図

図 2 から、A を通信元端末、G を宛先端末と設定すると A は B,H に RREQ をブロードキャストし、B,H は隣接端末に RREQ をブロードキャストする（H は省略）。その後、宛先端末に RREQ が到達するまで各端末は同様の動作を行う。図 4 では A→B→C→E→G と A→B→D→I→J→K→G の2つの経路が G まで到達する。

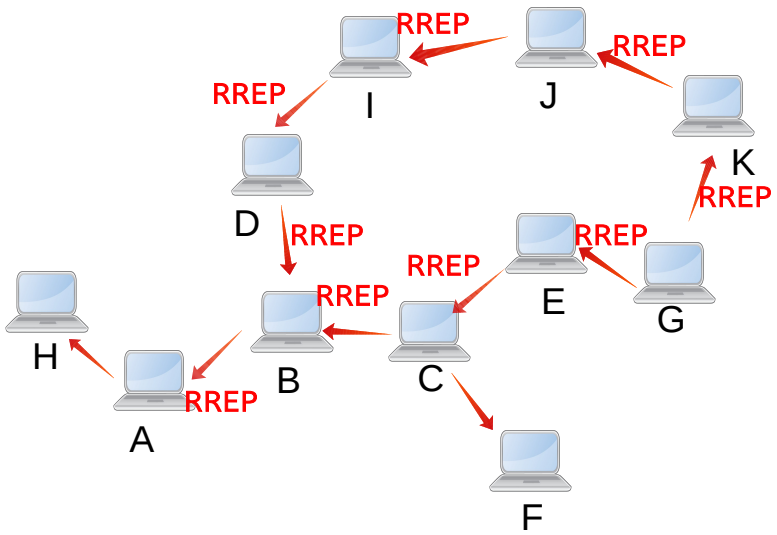

図 3：要求に対する返答の動作図

図３ではRREPについて示す。図２でRREQを送信端末Ａから宛先端末Ｇに届けた経路を保持し、保持した経路を使用しRREPをＧからＡに送信する。

この動作を行うと、通信経路は２つ構築される。構築される通信経路は前述した、A→B→C→E→GとA→B→D→I→J→K→Gの2つの経路である。既存ルーティングプロトコルでは、通信が確立されるまでの時間経過によって経路を１つに限定する。経過時間は、通信元端末から宛先端末までのホップ数に依存する。以上から、既存ルーティングプロトコルではホップ数の最も小さい経路が最適経路として確立される。

今回の例では、A→B→C→E→GとA→B→D→I→J→K→Gの2つの経路のホップ数がそれぞれ４と５になることから、A→B→C→E→Gが最適経路と判断される。

第 3 章　提案手法

3.1　既存手法の問題点

AODV ルーティングプロトコルでは、構築した通信経路が切断された場合、通信切断箇所を修正するのではなく再度始めから経路を組み直すので通信に遅延が発生し、バッテリー消費が増える点がある。

また端末の移動速度などは考慮していないので、速く動く端末を経路内に含んだまま経路構築を行った場合無線通信が不可な場所に移動してしまい、通信が切断され再度の経路組み直しになるため、通信が安定して行えない状況になる点がある。

3.2　提案手法

既存手法である AODV の速度を考慮していないという部分に注目しルーティングプロトコルの拡張を行う。

AODV は構築した経路を確立させる場合、経路構築までの経過時間（ホップ数）に依存する形である。この経路確立の際に、中間端末の移動速度から構築した経路に低速端末が多く含まれている経路を主経路に確定する。この動作により、高速で動く端末が選ばれにくくなり、端末移動による通信経路の切断の頻度を低下させ構築した通信経路が安定すると考えた。

上記の機能を実装したルーティングプロトコルの動作を下記に記す。

経路構築までの手順は上記した AODV ルーティングプロトコルと同様である。変更した点は、通信経路を確立させる際に経過時間（ホップ数）ではなく、各端末から送信可能な端末の中から最も移動速度の遅いものを選択していくことで最終的な経路を確立させるという点である。以下に詳細を記す。

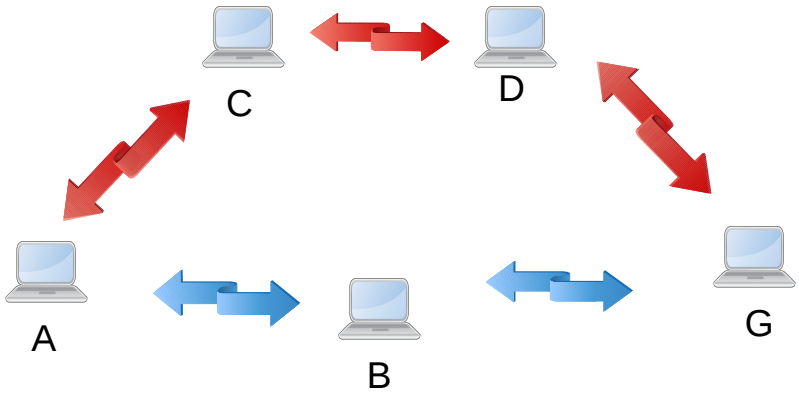

図 4：経路構築後の図

図4では A を送信元端末、G を宛先端末とし赤と青の双方向矢印は経路を表している。各端末が静止している状況であれば A→B→G と経路構築を行えば最適の経路となる。

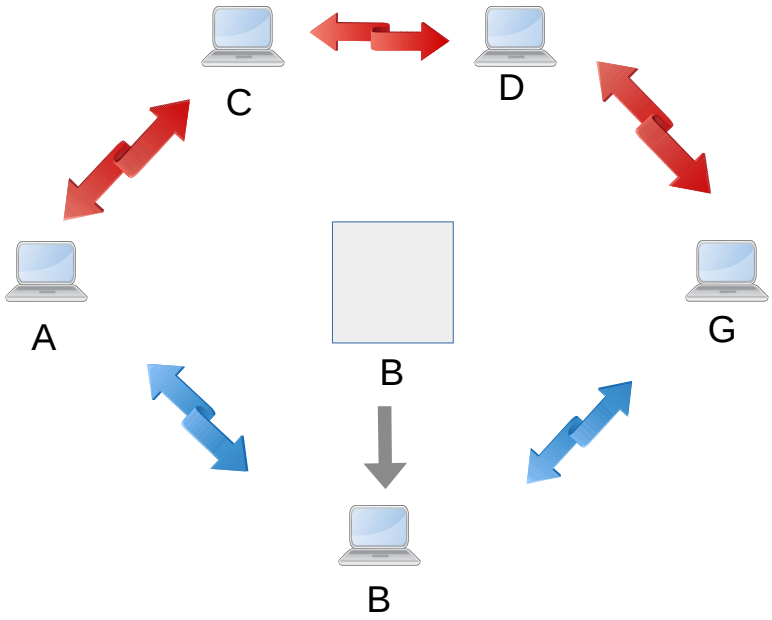

図 5:図 4 から数秒後を想定した図

次に図4から数秒後の各端末の状態を仮定する。図5では端末 B は四角の位置から矢印の指す位置まで移動している。C,D は図4よりほぼ移動していないものとする。

図5の場合、その後に端末 B は経路構築が不可能な位置まで移動する可能性が多分に含まれる。このことから A→B→G の通信経路を最適経路としている場合、数秒後に通信経路が切断される可能性が高い。

次に A→C→D→G の通信経路に焦点を当てる。図4から図5にかけて端末 C と端末 D はほぼ移動しておらずその後も移動しない可能性が高い。以上より A→C→D→G の通信経路は以降も切断されることなく維持できると評価できる。

上記したことから、通信経路を確立する際に各端末の移動速度を考慮することで通信経路を構築後に通信切断の起こりにくい通信経路を最適経路とする。よって、図4、図5を用いた例では A→C→D→G を最適経路として構築し、端末 A から端末 G までの通信を行う。

この機能を実現するため、提案手法では既存手法の RecvRequest 関数、RecvReply 関数に以下の機能を追加する。

1. 次端末に選択している端末が中間端末の中で最も遅い端末であるかの判定
2. 1.が偽の場合、通信可能な端末の中から最も遅い端末を次端末に選択

これにより、移動端末を考慮したルーティングを実装する。

第4章　実験と評価

4.1　実験目的

本実験では、アドホックネットワークの AODV ルーティングプロトコルの拡張を行い構築した通信経路の安定性を評価することを目的とする。

4.2　実験の評価・検証基準

安定性の評価基準として、「送受信したパケットの損失率」「通信の際の平均スループット」の2点に注目する。

1点目のパケット損失率の変化より、時間内で通信経路の安定への影響を評価する。送信元端末は、宛先端末に向けてパケットを送信する。通信経路が構築されている場合、送信したパケットを受け取る。通信経路が構築されていない場合、送信されているパケットは宛先端末に到達せずに破棄される。以上のことから、パケットの損失率の変化より通信経路の安定性を評価する。

2点目の平均スループットより、ネットワークの通信速度を評価する。スループットは、送信元端末から宛先端末までのホップ数に大きく影響される。ホップ数が小さくなれば通信速度は向上する。

以上2点より、拡張した AODV ルーティングプロトコルと既存 AODV ルーティングプロトコルの比較を行う。

4.3　ns3（Network Simulator 3）について

離散イベント駆動型ネットワークシミュレータとして開発された。新しいプロトコルの実装及び、大規模シミュレーション評価をより簡単にするために、開発言語とシナリオ記述言語を C++に統一して、並列分散環境のもとで大規模な実験も行える。このネットワークシミュレータを使い実験を行った。

サンプルで用意されている AODV のルーティングプロトコルのプログラム内に新たな関数を実装し、既存関数内で実装した関数を呼び出す動作を加えた。実装した関数は3つあり、それぞれ以下の動作を行う。

・中間端末の速度を取得しプロトコル内で定義した配列に格納
・通信先を速度の低い端末に変更を指示し、指示した端末が通信可能 かの判別
・端末ごとの通信可能端末のアドレスを二次元配列に格納

4.4　実験で比較するルーティングプロトコル

既存の AODV ルーティングプロトコルと、提案手法を導入した拡張 AODV ルーティングプロトコルで各データの比較を行う。各端末の配置座標、その他各設定を統一し実験を行う。

4.5　実験内容

4.5.1　実験環境

先に述べた既存ルーティングプロトコルと提案手法を組み込んだ拡張ルーティングプロトコルで同じ実験環境、シミュレーション時間、実行回数でデータを取得した。実験環境、シミュレーション時間、実行回数の詳細を以下に示す。

・送信元端末を(0, 125, 0)、宛先端末を(280, 125, 0)の座標に配置(以降固定)
・送信元端末と宛先端末の間にモバイル端末を5つ配置
・中間端末はシミュレーションごとにランダムに配置座標を設定しランダムに移動
・シミュレーションごとにスループットの平均、パケットの損失率を他ファイルに出力
・シミュレーション時間は 100 秒
・シミュレーション回数は 100 回

モバイル端末の制御には ns3 内に備えられている端末制御用のクラスを用いる。以下に詳細を示す。

-WaypointMobilityModel
　シミュレーション開始時の座標とシミュレーション終了時の座標を設定する。シミュレーション時間内で開始時の座標から終了時の座標まで一定の速度で移動する。

-RandomDirection2dMobilityModel
　シミュレーション開始時の座標と移動範囲を定める。端末がシミュレーション内で移動する速度の最大・最小を定める。端末は定められた最大・最小の間の速度で移動する。

本実験では備えられている端末制御用のクラスから上記 2 種のクラスを用いて端末の制御を行った。端末ごとの開始時の座標、終了時の座標は送信元端末と宛先端末の間の座標をランダムに取得させている。上記の内容を図 6 に示す。

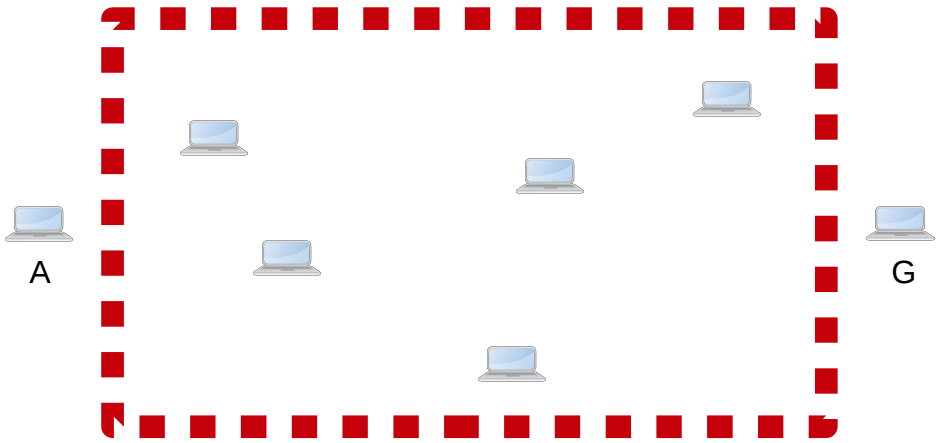

図 6:A を送信元端末、D を宛先端末としたときの実験環境図

図 6 の点線で示した範囲が 5 つのモバイル端末がランダムに動く領域としている。

4.5.2　実装内容

先に述べている提案手法を組み込むために、本実験では既存ルーティングプロセス内に 3 つの

新たな関数を設けた。3 つの関数について以下に示す。

−Speed 関数

int 型で引数に送信端末の Ipv4 アドレスを持つ。モバイル端の Ipv4 アドレスを速度順に並べて記載しているファイルから、Ipv4 アドレスを配列の変数に格納する。その後、格納した端末の Ipv4 アドレスを遅い順に引数で与えられている Ipv4 アドレスと比較する。一致した Ipv4 アドレスが最も遅い端末の場合 true を返し、それ以外は false を返す。

−Sender 関数

int 型で引数に端末番号と Ipv4 アドレスを持つ。引数として送られてきた端末番号のルーティングテーブルに接続指示を出した端末の Ipv4 アドレスが含まれているのかを判定する。含まれていれば true を返し、含まれていなければ false を返す。

−TableWrite 関数

void 型で引数に端末番号と Ipv4 アドレスを持つ。引数として送られてきた端末号ごとのルーティングテーブルを管理する。

上記の動きをする関数をルーティングプロセス内の通信経路の確立の動きをする関数の中に組み込み、宛先端末に到達するまでより遅い端末を経由するという指示を行っている。

実験環境用のプログラムでは、4.4.1 で述べた内容を実装させ、かつ各端末の始点と終点から速度の算出を行い外部ファイルに速度の遅い端末から端末が持つ Ipv4 アドレスを順に書き込んでいく関数を実装している。

4.6　実験結果

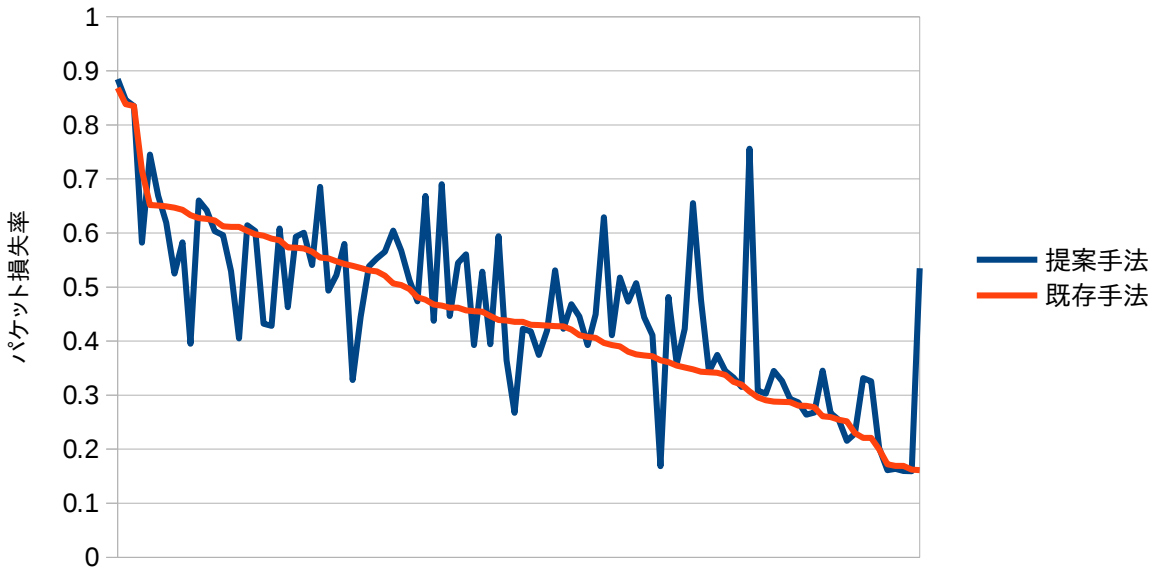

図7：パケット損失率の推移

図7は実験で取得した結果を既存手法の値でソートしたものである。

図7を見ると、既存手法のパケット損失率が大きくなり通信が不安定な状況下では提案手法の損失率が低くなり、通信がより安定していると言える。また、既存手法のパケット損失率が低く比較的端末同士が通信を行い易い状況下では、提案手法の損失率が高くなっている。

提案手法と既存手法を比較すると、既存手法がパケットの損失率が低くなることが多いことが確認できる。

表1：パケット損失率の比較結果

提案手法<既存手法	
真	42
偽	58

パケットの損失率の値が既存手法よりも提案手法が大きくなったのは 58 回で6割程度の傾向が見られる。また、2つの手法の全体の平均パケット損失率は提案手法が 46%、既存手法が 44%となった。

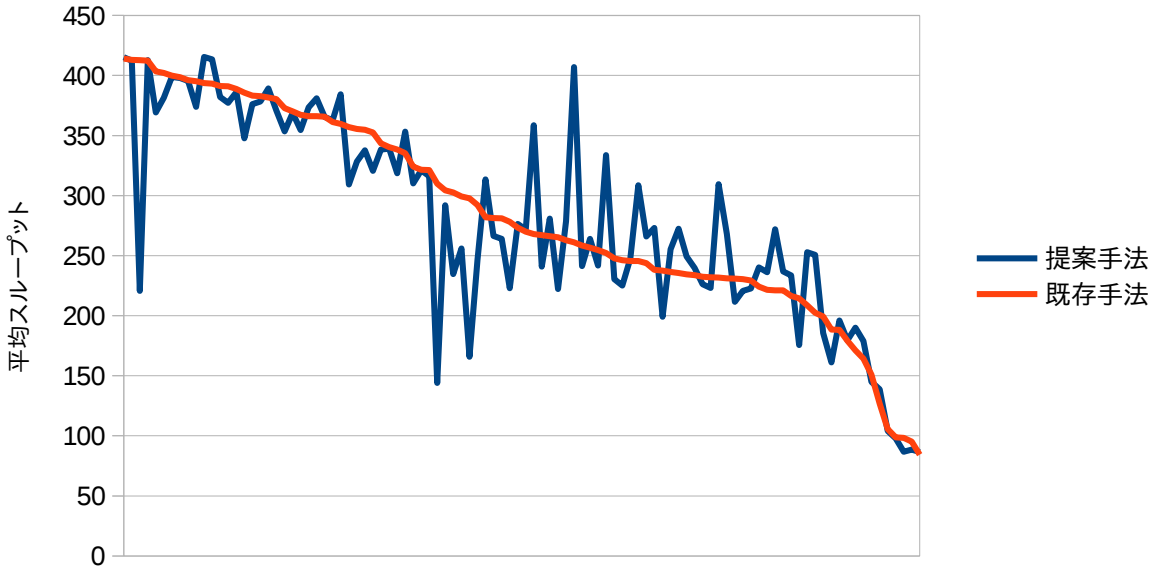

図8：平均スループットの推移

図8は実験で取得した結果を既存手法の値でソートしたものである。

図8を見ると、既存手法の平均スループットが高い状況下では提案手法の平均スループットが提案手法に比べ低くなる。また、提案手法の平均スループットが全体的に低い状況下にはるにつれ提案手法の平均スループットが高くなっていることが判る。

4.7　考察

4.7.1　パケット損失率の推移

実験結果より、全体的にパケット損失率が高くなり通信状態が悪いと判断できる状況下の場合は提案手法がより損失率を抑えた通信経路を構築できている傾向にある。既存手法は中間端末の動きに注目しないため、通信が可能な限界に達している端末でもホップ数が少なければ経路を構築してしまうことで、通信の切断が何度か起こりパケット損失率が上がっていると考えられる。これに対し、提案手法は中間端末の移動速度の遅いものを優先して選択して経路構築することから、1度構築した通信経路が切断されにくくなりパケット損失率の値が低くなっていると考えることができる。

一方、パケット損失率が全体的に低くなり通信が安定していると考えられる状況下では、提案手法よりも既存手法の値のほうが優れた結果が得られている。提案手法でよりパケット損失率が高くなってしまった理由として、通信端末の選択の優先度が関係していると考えられる。低速端末で通信することを優先したことで余分な端末通信を行っている考察できる。

4.7.2　平均スループットの推移

実験結果より、平均スループットが高くなる場合既存技術が提案手法に比べより効率良く通信経路を構築することができると判断できる。この結果から「平均スループットが高い」即ち「通信を行い易い状況」では端末の移動速度を考慮することより経路のホップ数をより少なくすることに注目した経路構築が良いと考えられる。よって、提案手法が既存手法に比べ値が低くなった原因として、低速端末を優先して繋いだことで無駄な中間端末を経路内に組み込みホップ数を大きくしてしまったと考えられる。

一方、平均スループットが全体的に低くなってくると提案手法がより高い平均スループットの値を出している傾向になっている。これは 4.7.1 で述べた既存手法と提案手法の経路構築の手法の違いによって起こるものと考えられる。

第5章　まとめ

本論文では既存ルーティングプロトコルと提案手法を用いた拡張ルーティングプロトコルの性能の比較を行いより安定したルーティングプロトコルの提案を行った。

「パケット損失率」、「平均スループット」ともに特定の条件下で優劣が変化している。それぞれを単独で通信経路を繋ぐことでどのような状況に弱いのかが明確に出来た。このことから、「低速端末を選択し経路構築を行う提案手法」と「ホップ数の大小で経路構築を行う既存手法」では端末の配置状況、移動速度によりその状況の最適手法が変わると言える。

今後の課題として、本研究での提案手法と既存手法の長所を引き出し通信を安定させるために、両手法を併せ持ったルーティングプロトコルの開発が挙げられる。2つの手法で経路を構築させより安定性の高いものを主経路とするルーティングプロトコルならばさらに性能の向上が望めると考察する。そのために、構築した経路の評価関数を作成し判別させることが必要である。

謝辞

本研究関し日々ご指導頂きました三好力教授に深く感謝申し上げます。また、研究過程で多くの助言や討論をして頂きました同三好研究室の皆様に感謝申し上げます。

参考文献

[1] 銭　飛著,ns3 によるネットワークシミュレーション

[2] ns-3
http://www.nsnam.org

[3] GitHub
https://github.com/dtaht/ns-3-codel-dev

[4] 誰もが電話を"携帯”するまでの歴史
http://next.rikunabi.com/tech/docs/ct_s03600.jsp?p=001430

[5] C-K. Toh 著；構造計画研究所訳,
アドホックモバイルワイヤレスネットワーク：プロトコルとシステム

[6] 東京電機大学 理工学部 情報システム工学科 長谷部 顕司、梅村 慎　　吾、檜垣 博章
「複数経路を用いた安定なメッセージ配送のための
アドホックルーティングプロトコル」

付録

```
Copyright (c) 2009 IITP RAS
--------- aodv-routing-protocol.cc ----------
Ipv4Address node[9][20];
Ipv4Address reset = "255.0.0.0";
int flag = 0;
int count[9] = {};

-----------------------------------------------------
int
RoutingProtocol::Speed(Ipv4Address sender)
{
  ifstream fp("Speed.txt");
  int w = 0;
  int number;
  while((fp >> speed[w]) != 0)
    {
      w++;
    }
  for(int i = w + 1;i >= 0;i--)
    {
      if(speed[i] == sender)
        number = i;
    }
  if(number == 0)
    return (true);
  return(false);
}

int
RoutingProtocol::Sender(uint32_t                n_number,
Ipv4Address sender)
{
  for(int j = 0;j < 20;j++)
    {
      if(node[n_number][j] == sender)
        {
          return(true);
        }
    }
  return(false);
}

void
RoutingProtocol::TableWrite(uint32_t                n_number,
Ipv4Address sender)
{
  node[n_number][count[n_number]] = sender;
  count[n_number] = count[n_number] + 1;
  return;
}
-----------------------------------------------------
void
RoutingProtocol::RecvRequest          (Ptr<Packet>          p,
Ipv4Address receiver, Ipv4Address src)
{

.......

 RoutingTableEntry toOrigin;
  if (!m_routingTable.LookupRoute (origin, toOrigin))
    {
       cout << " Node:   " << m_ipv4->GetObject<Node> ()-
>GetId () << " ------ " << src << endl;

-----------------------------------------------------
     flag = Speed(src);
     if(flag != 1)
       {
         for(int i = 0;i < 5;i++)
           {
              if(Sender(m_ipv4->GetObject<Node> ()->GetId
(), speed[i]) == 1)
                {
                  src = speed[i];
                  break;
                }
           }
       }
     cout << endl;
-----------------------------------------------------

     Ptr<NetDevice> dev = m_ipv4-     >GetNetDevice
(m_ipv4- >GetInterfaceForAddress (receiver));
                    RoutingTableEntry          newEntry
            (/*device=*/       dev,       /*dst=*/       origin,
            /*validSeno=*/ true,
             /*seqNo=*/          rreqHeader.GetOriginSeqno
(),
          /*iface=*/ m_ipv4->GetAddress              (m_ipv4-
>GetInterfaceForAddress
            (receiver), 0),
             /*hops=*/ hop,
```

```
        /*nextHop*/ src,
          /*timeLife=*/ Time ((2 *       NetTraversalTime
2 * hop *
                    NodeTraversalTime)));
          m_routingTable.AddRoute   (newEntry);
   }
  .........
}

void
RoutingProtocol::RecvReply (Ptr<Packet> p, Ipv4Address
receiver, Ipv4Address sender)
{

   ...........

 if (IsMyOwnAddress (rrepHeader.GetOrigin ()))
   {
     if (toDst.GetFlag () == IN_SEARCH)
      {
        m_routingTable.Update (newEntry);
        m_addressReqTimer[dst].Remove ();
        m_addressReqTimer.erase (dst);
      }
     m_routingTable.LookupRoute (dst, toDst);
     SendPacketFromQueue (dst, toDst.GetRoute ());
-----------------------------------------------------
     flag = Speed(sender);
     if(flag != 1)
      {
        for(int i = 0;i < 5;i++)
         {
            if(Sender(m_ipv4->GetObject<Node> ()->GetId
(), speed[i]) == 1)
             {
               toDst.SetNextHop(speed[i]);
               break;
             }
         }
      }
-----------------------------------------------------
     return;

  ............

}
```

```
Copyright (c) 2009 IITP RAS
--------- Environment.cc ----------
void
NetSim::ConfigureDataLinkLayer(bool                verbose,
StringValue phyMode, double dist)
{

  ...........

-----------------------------------------------------
      Ptr<WaypointMobilityModel> m_mob;
      m_mob = CreateObject<WaypointMobilityModel> ();
      mn->AggregateObject (m_mob);
        Waypoint wpt_start   (Seconds (0.0), Vector (x[0],
y[0], 0.0));
      m_mob->AddWaypoint  (wpt_start);
       Waypoint wpt_stop    (Seconds (SIM_STOP), Vector
(x[1], y[1], 0.0));
      m_mob->AddWaypoint  (wpt_stop);
          Ptr<RandomDirection2dMobilityModel>
m_mob2;
                                          m_mob2       =
CreateObject<RandomDirection2dMobilityModel> ();
      m_mob2->GetTypeId();
      m_mob2->SetPosition(Vector(70,50,0));
      mn2->AggregateObject (m_mob2);

      Ptr<WaypointMobilityModel> m_mob3;
        m_mob3 = CreateObject<WaypointMobilityModel>
();
      mn3->AggregateObject (m_mob3);
       Waypoint wpt_start3   (Seconds (0.0), Vector (x[2],
y[2], 0.0));
      m_mob3->AddWaypoint  (wpt_start3);
       Waypoint wpt_stop3    (Seconds (SIM_STOP), Vector
(x[3], y[3], 0.0));
      m_mob3->AddWaypoint  (wpt_stop3);

      ..........

-----------------------------------------------------

}

  ............
-----------------------------------------------------
```

```
void
NetSim::Speed()
{
        double road;
        double mn_speed[5][2];
        double rr = 2.0;
        int j = 0;
        int number = 4;
        int tmp_node;
        double tmp_speed;
        for(int i = 0;i < 5;i++)
         {
           road = (x[2*j+1]-x[2*j]) * (x[2*j+1]-x[2*j])
                  +(y[2*j+1]-y[2*j]) * (y[2*j+1]-y[2*j])
                  +(0-0);
           if(i == 0 || i > 1)
            {
              mn_speed[i][1] = sqrt(road)/SIM_STOP;
              mn_speed[i][0] = number;
              number++;
              j++;
            }
           if(i == 1)
            {
              mn_speed[i][1] = rr;
              mn_speed[i][0] = number;
              number++;
            }
         }

        for(int i = 0;i < 5;i++)
         {
           for(int k = 0;k < 5;k++)
           {
           if(mn_speed[i][1] > mn_speed[k][1])
               {
                 tmp_speed = mn_speed[k][1];
                 mn_speed[k][1] = mn_speed[i][1];
                 mn_speed[i][1] = tmp_speed;
                 tmp_node = mn_speed[k][0];
                 mn_speed[k][0] = mn_speed[i][0];
                 mn_speed[i][0] = tmp_node;
               }
           }
         }
        Ipv4Address address[9];
        ofstream outputfile("Speed.txt");
        if(outputfile.fail()){  // if(!fout)でもよい。
          cout << "出力ファイルをオープンでき            ません" <<
endl;
          return;
        }

        ifstream fp("SetAddress.txt");
        if(fp.fail()){  // if(!fin)でもよい。
          cout << "入力ファイルをオープンできません" << endl;
          return;
        }
        int i = 0;
        while((fp >> address[i])!=0){
          i++;
        }
        for(int i = 4;i >= 0;i--)
           outputfile << address[int(mn_speed[i][0])] << endl;
}

-------------------------------------------------------
int
main (int argc, char *argv[])
{
              NetSim sim;
-------------------------------------------------------

        ofstream file("X_Value.txt", ios::app);
        if(file.fail()){
          cout << "出力ファイルをオープンできません" << endl;
          return(false);
        }
        ofstream ff("Y_Value.txt", ios::app);
        if(ff.fail()){
          cout << "出力ファイルをオープンできません" << endl;
          return(false);
        }
        srand((unsigned)time(NULL));
        for(int i = 0;i < 8;i++)
         {
           x[i] = rand() % (270 - 10 + 1) + 10;
           file << x[i] << endl;
         }
        file << endl;
        for(int i = 0;i < 8;i++)
         {
```

```
        y[i] = rand() % (300 - 0 + 1) + 0;
        ff << y[i] << endl;
      }
    ff << endl;
    double s_packet;
    double r_packet;

------------------------------------------------------

    ..........

------------------------------------------------------
 ofstream output("result.txt", ios::app);
    if(output.fail()){  // if(!fin)でもよい。
      cout << "入力ファイルをオープンできません" << endl;
      return(false);
    }
          ofstream   throughput("result_  throughput.txt",
ios::app);
    if(throughput.fail()){  // if(!fin)でもよい。
      cout << "入力ファイルをオープンできません" << endl;
      return(false);
    }
    ofstream  loss("result_ loss.txt", ios::app);
    if(loss.fail()){  // if(!fin)でもよい。

    cout << "入力ファイルをオープンできません" << endl;
      return(false);
    }

    s_packet = double(sim.totalSent);
    r_packet = double(sim.totalReceived);
     output << "Total packets sent    : " << sim.totalSent
<< endl;
            output  <<  "Total  packets  received:  "  <<
sim.totalReceived << endl;
        output << "Average throughput     : " << sum /
(double( SIM_STOP / TH_INTERVAL)) << endl;
       output << "Packet loss rate       : " << r_packet /
s_packet << endl;
    output << endl;

          throughput  <<  sum  /  (double(  SIM_STOP  /
TH_INTERVAL)) << endl;

    loss << 1 - (r_packet / s_packet) << endl;
}
```