平成 24 年度　特別研究報告書

アドホックネットワークにおける
低消費電力経路探索法の検討

龍谷大学　理工学部　情報メディア学科

T090440　濱口　圭介

指導教員　三好　力　教授

内容梗概

近年，携帯機器の通信方法として無線通信が存在する．この通信方法の 1 つに基地局を介さずに端末(ノード)同士で通信を行い，中継ノードをつないで通信範囲を広げるマルチホップ通信がある．この通信方法を利用し MANET(モバイル・アドホック・ネットワーク)を構築する．この MANET 構築の際，ルーティングプロトコルの 1 つである拡張リング法が存在する．しかし，拡張リング法では中継ノードが多くなればなるほどパケット送信の重複が起きている．さらに，その重複の分だけ送信元ノードと送信元ノードに近いノードは消費電力が生じてしまうという問題点がある．そこで本研究ではパケット送信の重複を無くす手法を提案し，拡張リング法と提案手法の比較実験により検証を行った．

比較実験はフィールドサイズを変化させた場合の，拡張リング法と提案手法の送信回数，経路探索時間を比較した．その結果，経路探索時間はサイズに関係なく拡張リング探策が先に探索を終了することがわかった．送信回数ではサイズが狭くノードが密集している場合では拡張リング探索が良く，サイズが広く中継ノードが増えれば増えるほど，提案手法が良いことがわかった．

目次

第１章　はじめに

近年，携帯電話の普及に伴って携帯端末は現在人口を超える契約数となり，誰もが利用するものとなっている．表１．１は 2012 年９月での携帯機器契約台数を示している．

表１．１　事業者別月末契約数　電気通信事業者協会発表[1]

グループ	2012 年９月
ＮＴＴドコモ	60786600
ａｕ	36110400
ソフトバンク	30461200
ウィルコム	4813700
UQ コミュニケーションズ	3624100
ＷＣＰ	378800

携帯機器の通信方法は大きく分け有線，無線通信が存在する．この無線通信の１つに基地局を介さず，携帯端末同士で通信を行うことのできる機能が存在する．極短距離ではあるが消費電力の少なく指向性のある赤外線通信，超低消費電力の無線通信手法のシングルモード Bluetooth，超広帯域無線通信 UMB(Ultra Wide Band)などがある．赤外線通信は，障害物がない環境で利用可能であり，通信距離は約１メートル以下である．Bluetooth は 2．4GHz の電波を使用した通信を行い，送信電力量によって変化し，標準では半径約 10 メートル以内での通信である．Bluetooth 搭載端末同士での通信エリアをピコネットと言い，ピコネット内には送信元であるマスター端末が存在し，マスター端末と複数の端末が通信を行うことができる．また，携帯やスマートフォンなどの小型端末に幅広く対応しているため，実用的である．UMB は３．１ GHz～10．6GHz の幅広い周波数帯を用いて通信を行う．送信出力は，家庭用テレビやパソコン等の一般の電子機器等が発生する雑音レベルの約 500 分の 1 以下という、非常に小さな出力で通信が可能である。しかし，通信半径に問題があり広範囲での通信には非対応である．図１．１で示すのは無線環境における通信速度と通信距離の関係のグラフである．黄色の範囲では，2．4GHz 以下の電波を使った通信方法であり，携帯端末同士が独自で通信を行う範囲である．上記で示した赤外線，Bluetooth が該当する．緑の範囲では，主に LAN で構築されるネットワークであり，家庭で使われる PC などの無線接続の範囲を示している．オレンジの範囲は LAN を拡張したもので，WAN と呼ばれ LAN 同士を接続することでネットワークを広範囲に広げることができる．黄緑の範囲は基地局を経由する通信方法であり，通信距離は基地局を利用することでどこの場所の相手とでも通信を可能としている．本論文で取り扱う通信範囲は黄色の範囲である．

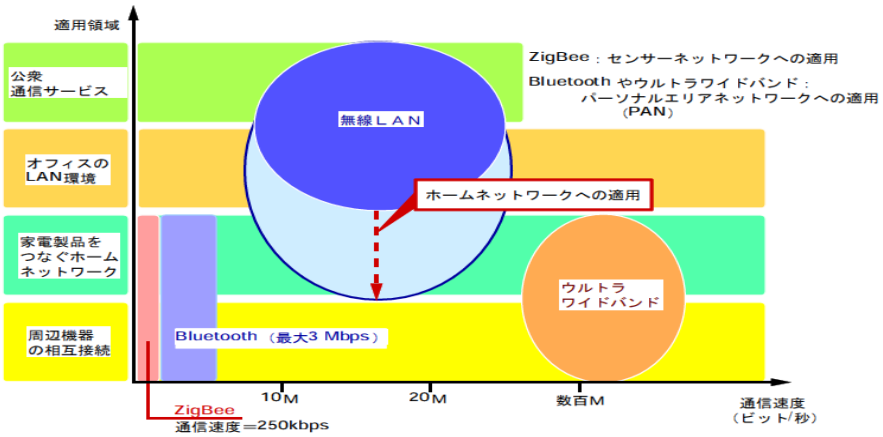


図１．１　無線環境における通信速度と領域の関係[2]

以上が，主に端末同士の１対１の通信を可能としたものである．しかしこれらには通信可能範囲があり，その距離を越えての通信は不可能となってしまう．

この問題点を解決する手法として，その通信範囲を拡大するマルチホップ無線ネットワークというものが存在する．通信可能範囲内に通信先が存在しない場合，通信可能範囲内の他のノードを利用し，そのノードを中継ノードとして通信半径を拡大していくというものである．

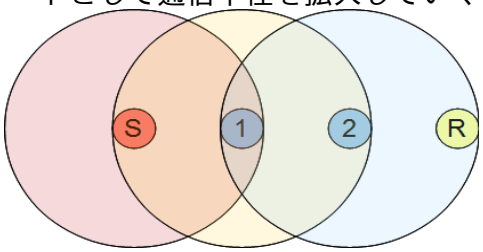


図１．２　マルチホップ無線通信のイメージ

図１．２はマルチホップ無線通信のイメージである．S(Sender)送信元，R(Receiver)送信先とし，中継ノードとして，ノード１，２が存在する．それぞれのノードを中心とした色付きの円形を通信範囲と考える．S と R が通信を行いたいが，それぞれの通信範囲内に存在しない場合を想定している．初めに S は R と通信するために通信範囲内のノード１へ，R に送りたい情報を送信する．次にノード１は R と通信するために通信範囲内のノード２へ情報を転送する．最後にノード２は通信範囲内にいる R へ情報を転送することで，R との通信が可能となる．通信範囲内のノードを中継ノードとして利用することで，マルチホップ無線ネットワークを拡大することができる．

マルチホップ通信の形態として，本論文で扱うアドホック通信というものがあり，構築されるネットワークをモバイル・アドホック・ネットワーク(MANET)という．

1．１　MANET とは

パケット通信(ネット検索，メールの送受信など)では固定設置されている基地局がそれぞれの携帯端末との通信の中継を行っているが，MANET ではそれらの基地局やアクセスポイントを介さずに携帯端末同士が独自に通信を行いデータのやり取りをすることができる．この MANET を利用することで，緊急災害時の救助時やセンサネットワークへの期待ができる．また，このアドホック通信での課題は通信経路の保持，経路の決定方法が難しいことである．携帯端末が移動可能であり，ネットワークの環境が変化しやすい(突如電源が切れたり，新しい端末の参加など)ためである．また，携帯端末の問題としての電力容量がある．MANET では，外部から電力を供給する手段がない場合，各端末のバッテリーのみで稼動することになる．そのためほかの端末との中継ノードになったり，データの送受信によって電力を消費してしまい，限られた電力を使い果たすとバッテリー切れを起こし構築されたネットワークから離脱してしまう．下図は移動型の携帯端末同士が通信局を経由することなく送信側から受け側へとネットワークを構築しているイメージである．

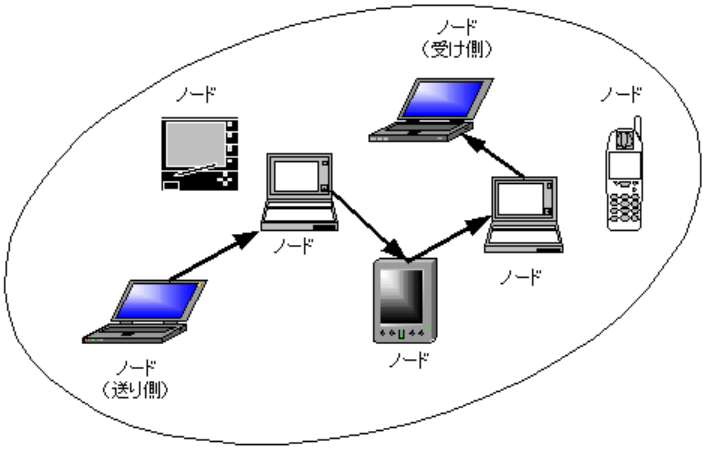


図 1．3　アドホックネットワークのイメージ[3]

1．2　MANET への期待

近年普及し続ける携帯端末(特に携帯電話，スマートフォン)により，MANET への期待が高まっていくと考えられる．一般社会への実用化については，先日起こった東日本巨大地震における通信基地局の崩壊や不通の問題への対処法である．実際には，災害の影響ピーク時の基地局との不通が NTT ドコモは最大 6720 局，KDDI(au)は 3800 局，ソフトバンクでは 3780 局であった．2 日以内で復旧した基地局もあるが，全基地局復旧までには時間がかかってしまった．臨時で移動型の基地局を設置するなどの対策も行っていたが，実際には不通状態の基地局をカバーできるほどの数ではなかった．今回の災害では地震による被害より津波による被害がほとんどであったが，地震による被害を想定すると，建物の崩壊などにより生き埋め状態になる可能性がある．これらの被災者が例え携帯端末を所持していたとしても，基地局との通信が不通であれば通信手段がなくなってしまう．そこで考えられる対処法のひとつが MANET による通信である．これを利用することで例え地上からの通信に届かない場所で埋まっていても中継ノードをたどることで広範囲の探索を行える．先日の大地震の影響で活断層の変化における更なる震災の想定が出てきたなか，携帯端末をより良く活用し，今後に期待できる手法であると考えられる．

第2章　ルーティングプロトコル

2．1　ルーティングプロトコルとは

ルーティングプロトコルとは，2つ以上のネットワークを接続するルータ同士間の，情報やり取りを行う際に用いる複数のノードを選択し，経路設計するための通信規約である．また，ルーティングプロトコルは主に3つに分類(プロアクティブ型，リアクティブ型，ハイブリッド型)することができ，それぞれを以下に示す．

プロアクティブ型では，新しい経路情報を常に保持しておき通信要求があった場合に即時に通信をすることができる．常に経路情報を保持しているので電力コストがかかってしまう．主に小規模ネットワークで用いられる．代表プロトコルとして，OLSR(Optimized Link State Routing)，TBRPF(Topology Dissemination Based on Reverse-Path Forwarding)，DSDV(Destination Sequence Distance Vector)，LANMAR(Landmark Routing Protocol)．FSR(Fisheye State Routing Protocol)，IARP(Intrazone Routing Protocol)などがある．

リアクティブ型では，通信要求があった場合に経路情報確保のためにパケットを送信し通信を開始する．電力コストは低いが通信に時間がかかってしまう．主に大規模ネットワークで用いられる．代表プロトコルとして，AODV(Adhoc On-Demand Distance Vector)，DSR(Dynamic Source Routing)，IERP(Interzone Routing Protocol)などがある．

ハイブリッド型では，プロアクティブ型，リアクティブ型のそれぞれの欠点を補ったプロトコルである．距離が近いノードとの通信はプロアクティブ型のアルゴリズムを用い，距離が開くノードとの通信はリアクティブ型のアルゴリズムを用いる．代表プロトコルとして，ZHLS(Zone-based Hierarchical Link State)がある．

2．2　MANETで用いる既存ルーティングプロトコル

MANETでは携帯端末での通信なので，通信状況が著しく変化してしまう．このことより，静的な環境でのインターネット用のルーティングプロトコルは対応できず，動的なMANET向けのルーティングプロトコルを必要とする．本論文では，MANETにおける2つの既存ルーティングプロトコル，メッセージフラッティングと拡張リング法について紹介する．

2．2．1　メッセージフラッティング

メッセージフラッティングとは，送信元から送信先へとメッセージを送信する際に，送信先ノードまで中継する全ノードへパケットを送信するという手法である．設定値までを探索するTTL(TTL = Time To Live(目的地までのホップ数))という値を設定することが一般であり，設定値までの範囲に送信先のノードが発見できれば接続完了となる．隣接ノード全体に送信することによって接続にかかる時間は早く，信頼性の高いネットワーク構築が可能である．

しかし，実際に送信先のノードが何ホップくらいかを判断することは困難であり，TTL値を高めれば送信先ノード発見の信頼度は高いが，その分送信パケットが膨大になりコストも増加してしまう．また，TTL値を低く設定してしまうと送信先ノード発見の信頼度は低くなり，最悪接続不可能の状況に陥ってしまう．このことから，特にネットワーク環境が狭い環境で使用されている．

2．2．2　拡張リング法

拡張リング法とは2．2．1節で述べたメッセージフラッティングの問題点を改善したものである．メッセージフラッティングとは異なりTTL値を変数にすることによって，送信先ノードを確実に発見でき，コスト面では最低限のノードでしか探索を行わないためMobile対応のルーティングとして使用されている．移動型のノードだが，通信時間，接続時間は十分に短いものとすると，ノードを固定して経路設計を考えることができる．送信元ノードをS，送信先ノードをRとする．Rが隣接ノードからパケットを受信すると，Sまでの経路情報通りに返信メッセージを送る．この返信をSが受信することで全体的な経路探索終了となる．今回の例では1ホップ(隣接ノード間の距離)間の接続完了までの時間は全て同じと考え，ノード間に障害物や電波送信における抵抗は考えないものとする．図2．1，2．2，2．3にて，拡張リング法動作例を示す．

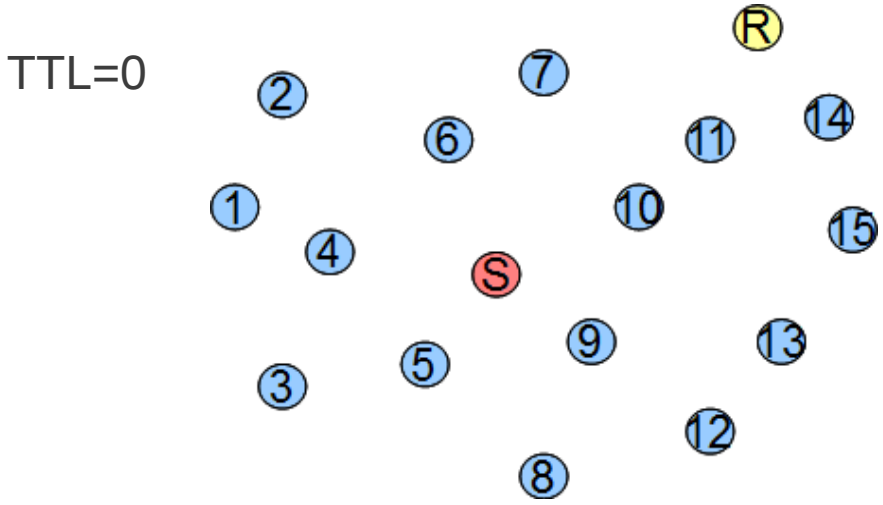


図2．1　ランダムに配置したノードの例

図2．1では．ランダムに配置された計17個の携帯端末(ノード)があるとし，SとRが通信をしようと想定している．TTLは通信を行っていないので0である．

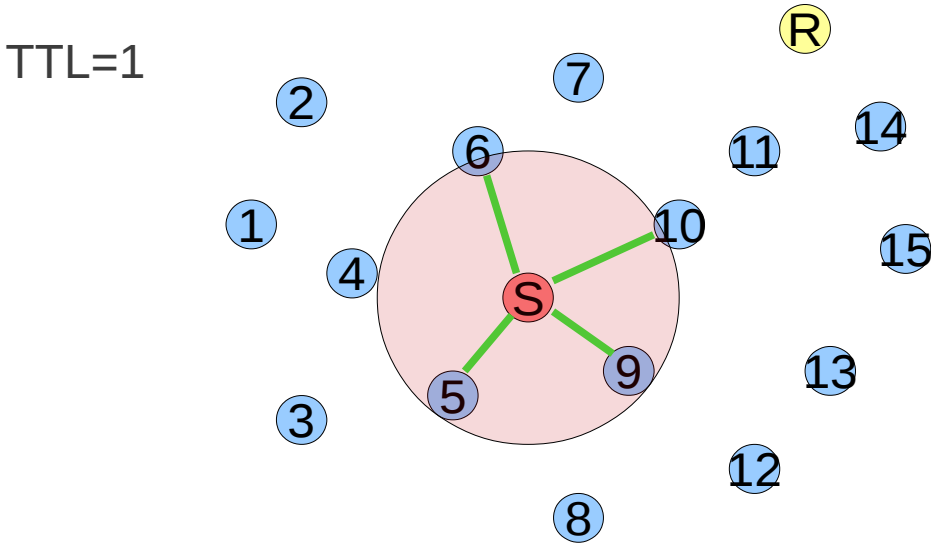


図2．2　隣接ノード探索例

1．Sから指定半径範囲内(今回では赤色半透明の円内)にノードが存在するかを探索する．例では該当ノードが5，6，9，10が存在しており，SはTTL=1でパケットを送信し，隣接ノードがRであるかを探索する．図2．2における緑の線が今回新たな経路である．送信回数合計は送信元Sの発信=1であり，SはTTL=1内にRがあった場合に返信にかかる1ホップ分を加えた，2ホップ分の待ち時間でRへの経路があるかを判断できる．

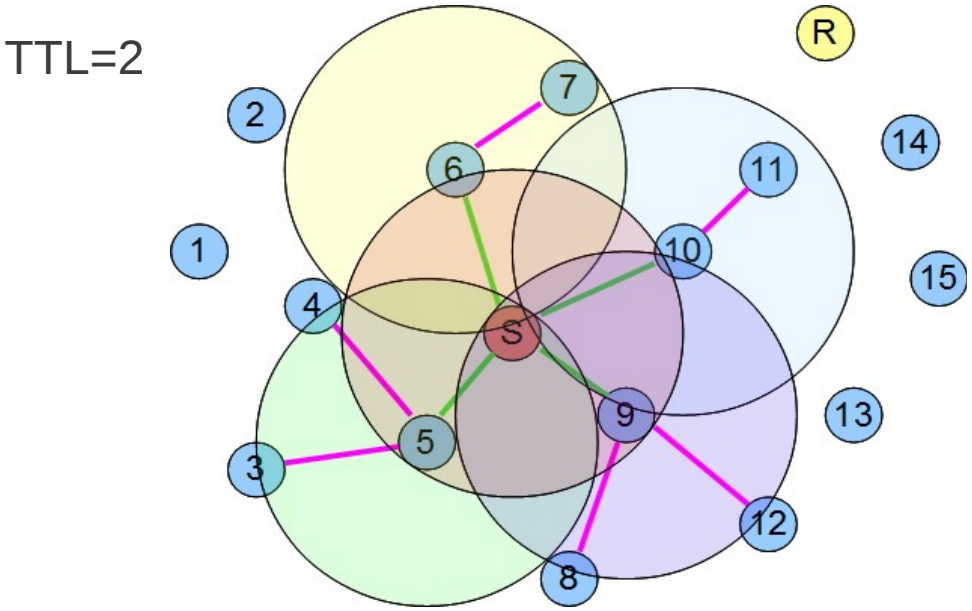


図2．3　隣接ノード探索例

2．上記1の結果，Rへの経路が発見されなかったため，再度Sより隣接するノードへTTL=2でパケットを送信する．Sからのパケットを送信し，ノード5，6，9，10が受け取ると，それらノードはTTL=1でパケットを送信し，隣接ノードがRであるかを探索する．例ではノード5にノード3，4が，ノード6にノード7が，ノード9にノード8，12が，ノード10にノード11が存在している．図2．3における紫の線が今回新たな経路である．送信回数合計は送信元Sのの隣接ノードへの送信+ノード5，6，9，10の発信=1+4=5であり，SはTTL=2内にRがあった場合に返信にかかる2ホップ分を加えた，4ホップ分の待ち時間でRへの経路があるかを判断できる．

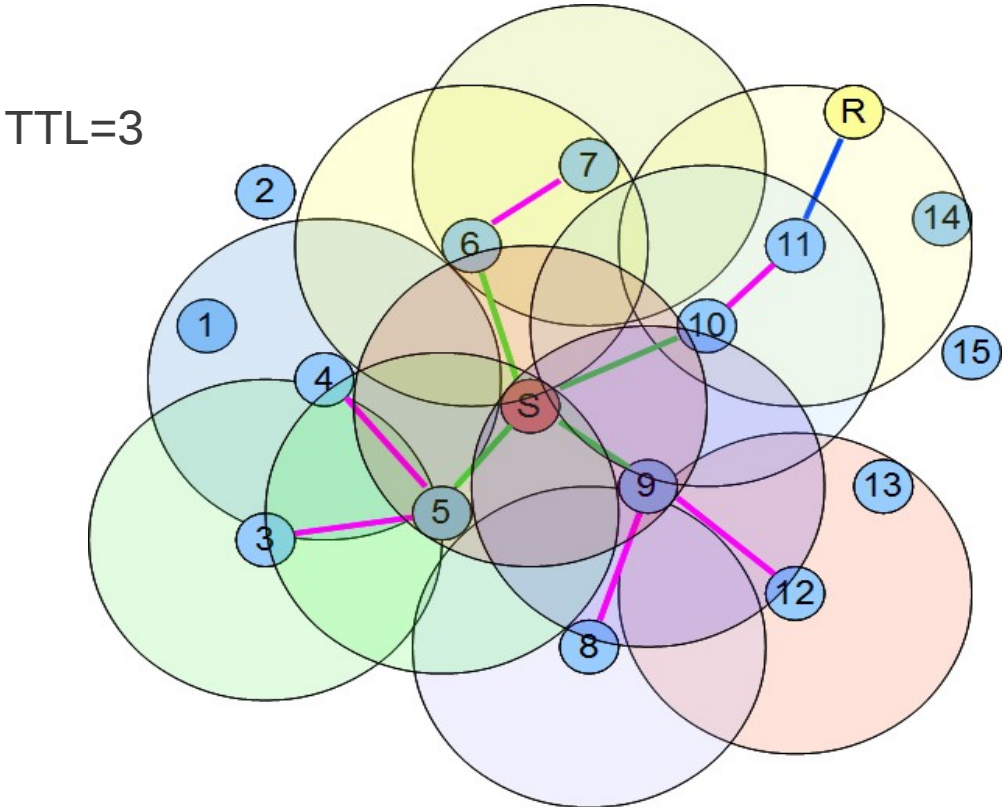


図2．4　隣接ノード探索例

3．上記2の結果，Rへの経路が発見されなかったため，再度Sより隣接するノードへパケットをTTL=3で送信する．Sからのパケット送信し，ノード5，6，9，10はTTL=2でパケットを送信し，上記2の探索結果の隣接ノードがパケットを受け取ると，TTL=1でパケットを送信し，隣接ノードにRが存在するかを探索する．例ではノード4にノード1が，ノード11にノード14，Rが，ノード12にノード13が存在している．図2．4における青の線が今回新たな接続結果である．送信回数合計は送信元の隣接ノードへの送信+ノード5，6，9，10の隣接ノードへの送信+ノード3，4，7，8，11，12の発信=1+4+6=11であり，SはTTL=3内にRがあった場合に返信にかかる3ホップ分を加えた，6ホップ分の待ち時間でRへの経路があるかを判断できる．

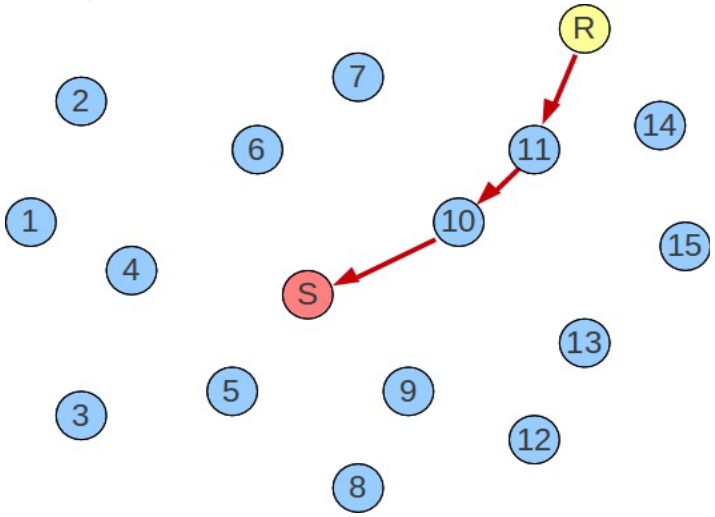


図2．5　拡張リング法の場合のRの返信イメージ

4．上記3の結果，Rへの経路が発見された．図2．5のようにRからの返信がSの到達すると経路探索は終了する．

今回の例ではTTL＝3で接続完了し，送信回数の全合計は1+5+11+3(赤線)=20回，探索時間は12ホップ分である．拡張リング法の経路探索時間はTTL×(TTL+2)で求めることができる．

2．3　研究目的

本研究では MANET 構築時の消費電力量の更なる低下のため，拡張リング法を改良した新しい通信プロトコルの提案を目的とする．

今回拡張リング法のパケット送信回数に着目した．転送，送信回数を減らすことができれば消費電力の節約が期待できる．この改良点では主に，ノード数が増加すればするほど消費電力量の低下が見込める．

2．2 の例で示した拡張リング法では，TTL の増加毎に S から隣接ノードへ再送信している．再送信が多くなればなるほど，送信回数，転送回数は膨れ上がる．また，S の送信回数も多く，電力消費が激しいと考えられる．前節で述べたように電力量が限られた環境での過度の通信はできるだけ避けたい．そこで，この改良点として，隣接ノードとして発見したノードに着目する．R を発見できなかった場合は，S から再送信するのではなく，経路接続できている一番端のノードから転送を開始する．そうすることで，再送信の際の重複は起こらず，さらに電力消費は均等に分散することが期待できる．図 2．5 では重複を表した例である．

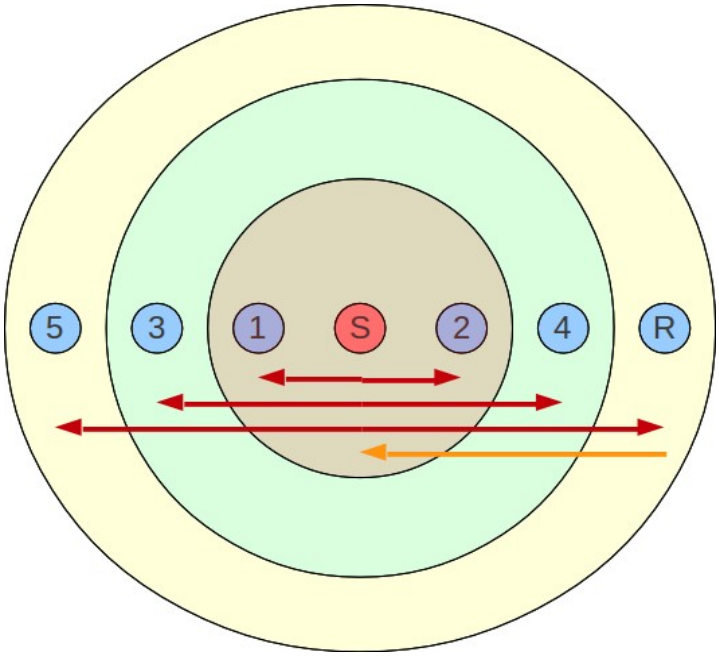


図 2．6　拡張リング法での重複送信のイメージ

図 2．6 では重複をなくした例である．初めにノード 1 とノード 2 を発見し接続したあと，今度はノード 1，2 から隣接ノードの探索を行う．ノード 3，4 も同じように探索を行う，黄色の線は返信と各ノードへの更なる隣接ノードへの転送を阻止するための転送停止パケットである．

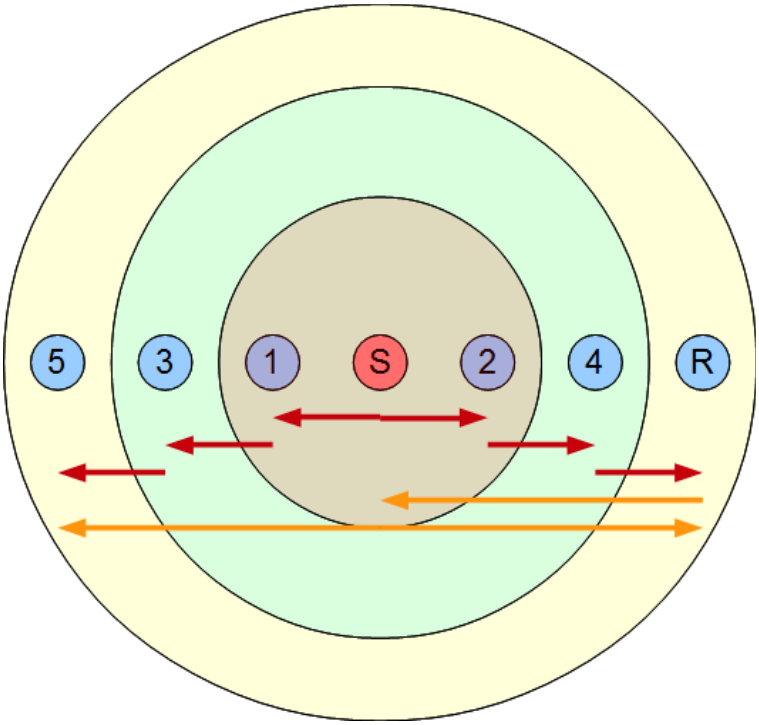


図 2．7　提案手法ののイメージ

第３章　提案手法

3．1　概要

ここでは２．３節の拡張リング法を改良した提案手法について述べる．

拡張リング法では送信先ノードが未発見の状態で送信元ノードからの送信の際，TTL=2 以降現在接続完了している全ノードにパケットを再送信してる．これではノード数が増加すればするほど送信の重複が起こり，コスト的に無駄が生じてしまう．そこで考えられる手法が重複をなくした探索法である．この探索法を行うことによってコスト面でかなりの節約，また，個々のノードへの電力量の偏りが改善できると考えられる．

拡張リング法と同じように通信時間，接続時間は十分に短いものとすると，ノードを固定して経路設計を考えることができる．送信先ノードRが隣接ノードからパケットを受信すると，送信元ノードまでの経路情報通りに送信元ノードに向かって返信メッセージを送る．この返信をSが受信すると送信元ノードはRまでの経路以外の接続している経路へ更なる隣接ノードへの転送をやめさせるため，転送停止パケットを送信し，全体的な経路探索は終了となる．今回の例では1ホップ(隣接ノード間の距離)間の接続完了までの時間は全て同じと考え，ノード間に障害物や電波送信における抵抗は考えないものとする．図3．1，3．2，3．3にて，提案手法の動作例を示す．

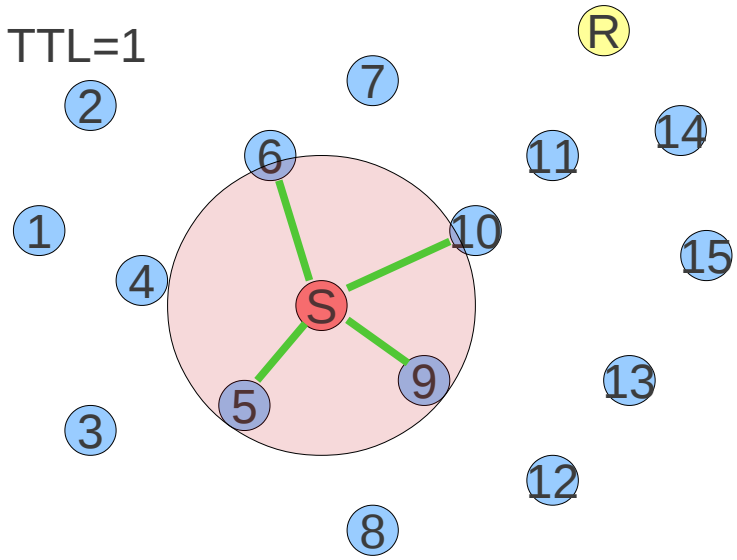


図3．1　隣接ノード探索例

1．既存手法である拡張リング法同様にまず初めには，Sから指定半径範囲内(今回では赤色半透明の円内)のノードを探索する．例では該当ノードが5，6，9，10が存在しており，SはノードへTTL=1でパケットを送信し，隣接ノードRがあるかを探索する．図3．1における緑の線が今回新たな経路である．送信回数合計は送信元Sの発信=1であり，SはTTL=1内にRがあった場合に返信にかかる1ホップ分を加えた，2ホップ分の待ち時間でRへの経路があるかを判断できる．ここで既存技術改良点として，TTL=1でRの返信があった場合に，SはTTL=1内の隣接ノード(今回ではRを含めた5，6，9，10)に転送停止パケットを送るようにする．結果，TTL=1での全体の待ち時間は3ホップ分である．

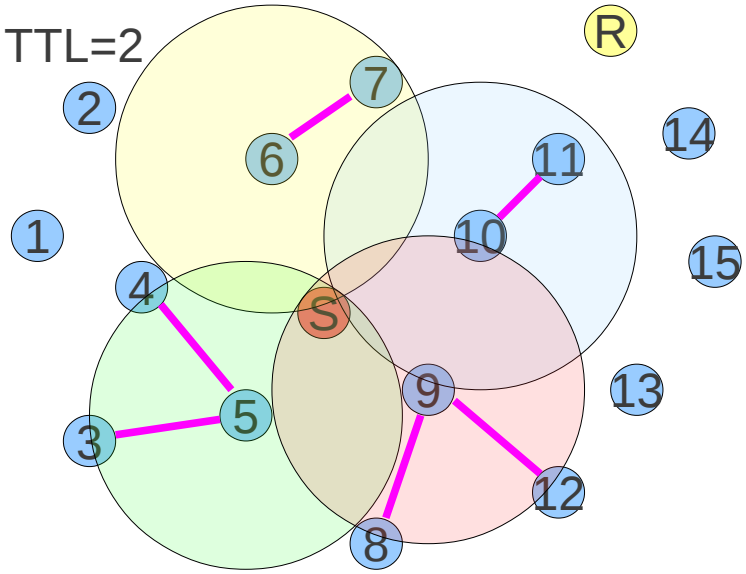


図3．2　隣接ノード探索例

2．上記1の結果，Rへの経路が発見されなかったため，再びRを探索する．ここで，拡張リング法とは違い，Sは待機状態となり，ノード5，6，9，10がRを探索する．例ではノード5にノード3，4が，ノード6にノード7が，ノード9にノード8，12が，ノード10にノード11が存在している．図3．2における紫の線が今回新たな接続結果である．TTL=2での，送信回数の合計は，ノード5，6，9，10の発信=4であり，SはTTL=2内にRがあった場合に返信にかかる2ホップ分を加えた，3ホップ分の待ち時間でRへの経路があるかを判断できる．全体の待ち時間は，転送停止パケット分を含めた計5ホップ分である．

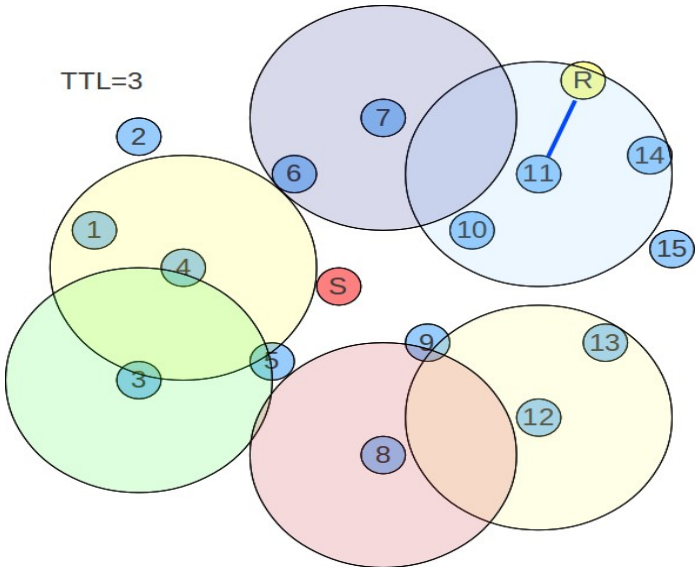


図3．3　隣接ノード探索例

3．上記2の結果，Rへの経路が発見されなかったため，再びRを探索する．今回はノード3，4，7，8，11，12がRを探索する．例ではノード4にノード1が，ノード11にノード14，送信先ノードが，ノード12にノード13が存在している．図3．3における青の線が今回新たな接続結果である．TTL=3での送信回数の合計はノード3，4，7，8，11，12の発信=6であり，待ち時間は，発信のみの1ホップ分である(今回ではRの返信があったため，返信，転送停止パケット分は含めていない)．

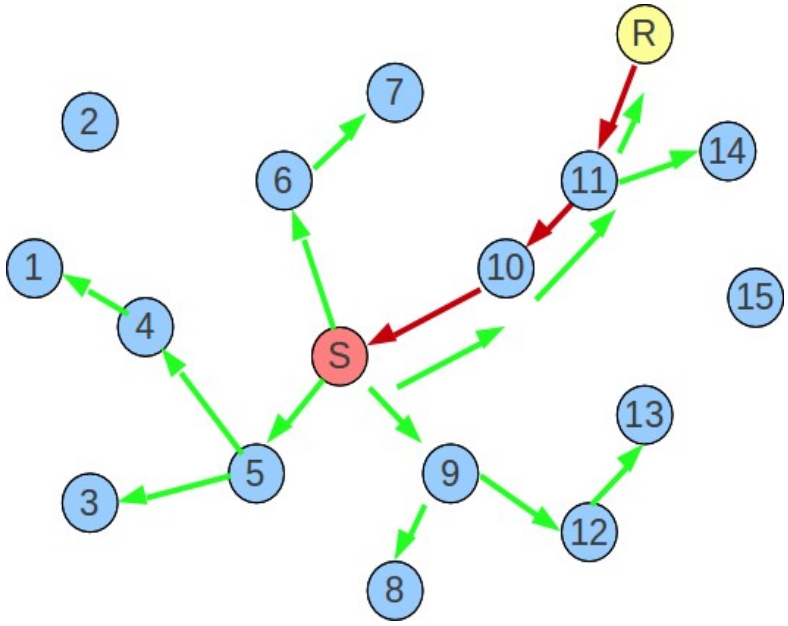


図3．4　探索終了時の全ノードへのパケット送信例

4．上記3の結果，Rへの経路が発見された．図3．5のようにRからの返信(赤線)がSの到達し，転送停止パケット(緑線)が各ノードへ送信されると経路探索は終了する．今回の例ではTTL＝3で接続完了し，送信回数の全合計は1+4+6+3(赤線)+11(緑線)=25回，探索時間は3+5+1+3(赤線)+3(緑線)=15ホップ分である．提案手法の経路探索時間はTTL×(TTL+2)で求めることができる．

3．2　送信元ノード，中継ノードの動作アルゴリズム

・送信元ノードSの動作アルゴリズム
nを最大探策パケット数，ホップ数(時間)をt，TTL=iとする．
R探索パケットをTTL=nとTTLhp=1で送信し，時間tをカウントしRの返信を待つ．
Rからの返信があれば，次式でRまでのiを計算する．
$i^2+i-t=0\ (i>0)$　……(注)
解の公式より
$i=(-1+\sqrt{1+4t})/2$
計算結果のTTL=iで転送停止パケットを送信する．
注:提案手法のSからRまでのTTLは，Rからの返信をSが受信した時点の時間t(ホップ数)がわかれば計算することができる．t=2の場合，TTL=1．t=6の場合，TTL=2．t=12の場合，TTL=3．t=20の場合，TTL=4．このようにTTLが1増加するとtは4，6，8，10，12，14，･･･と増加する．よってtの数列は階差数列であり，t=TTL(TTL+1)の式が成り立つ．この式を変形し，t=TTL^2+TTLからTTL^2+TTL-t=0とした．

・中継ノードアルゴリズム
探索パケット（TTL=n，TTLhp=m）を受け取ると，次式の時間tw待機する．
$tw=m\times 2$
待ち時間twの間に転送停止パケットを受信しなかった場合，
探策パケット（TTL=n-1，TTLhp=m+1）を送信する．
転送停止パケットを受け取ると，受け取ったTTL=iをTTL=i-1にして転送停止パケットを送信する.もしTTL=1の場合送信しない．

第 4 章　実験

4．1　実験概要

本実験では既存手法である拡張リング法と提案手法を実装し，それぞれの性能の比較を行う．

性能比較対象は電力消費につながるメッセージ送信回数，アドホックネットワーク接続完了までにかかった時間(ホップ数)とする．フィールドサイズを変化させたパターンを複数用意しそれぞれでも比較し，実験結果を元に考察を行う．

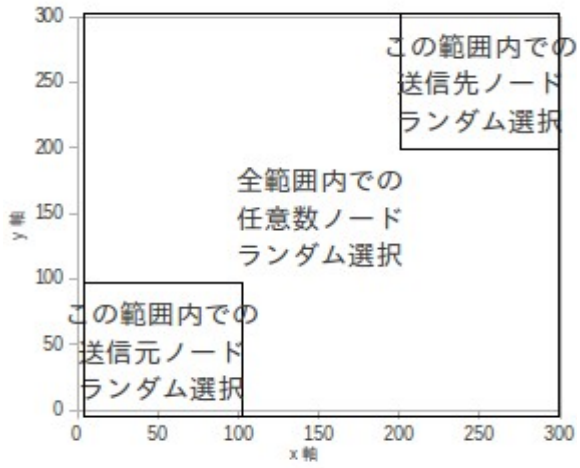


図 4．1．1　プログラムでのノード分布状況

4．2　プログラムアルゴリズム

既存手法．提案手法の共通プログラムアルゴリズムについて説明する．

4．2．1　拡張リング法プログラムアルゴリズム

1，ノード 125 個分の座標を用意する．(x，y)=(0～200，0～200)とする．

2，送信先 R と送信元 S のノードは図 4．1．1 のように配置する．

3，ノード自身が探索できる半径 r を設定する．

4，S から r 以内のノードを探索する．ユークリッド距離を使用．

5，r 内にいたノードを隣接ノードとして考えノード 1 つ毎に送信数 send(変数)を 1 増やす．
　この時探索したノードは使用済みとし，再探索されないようにする．

6，S の隣接ノードをすべて探索したら，TTL を 1 とする．(以降 1 ずつ増加)

7，もし r 内に R が存在した場合，探索を終了し，現在の send 数を出力する．

8，7 が成り立たなかった時，send を 1 増やし，S の隣接ノードが r 以内のノードを探索する．

9，r 内にいたノードを更なる隣接ノードとして，7 の条件が成り立つかを判断する．

10，4～9 を繰り返し行い，send の合計，送信時間を求める．

4．2．2　提案手法プログラムアルゴリズム

拡張リング法同様に行うが，send と送信時間の求め方を変更する．

1～7 は同様

8，7 が成り立たなかった時，S の隣接ノードが r 以内のノードを探索する．

9，r 内にいたノードを更なる隣接ノードとして，7 の条件が成り立つかを判断する．

10，4～9 を繰り返し行い，送信数，送信時間を求める．

4．3　実験結果

実験結果を図 4．1～図 4．5，表 4．1～4．3 に示す．図 4．1～図 4．3，表 4.1，表 4．2 は送信回数のグラフである．

図 4．1 はノード数 125 でフィールドサイズ 200×200 での送信数の結果である．このサイズは広範囲，疎らなノードの配置を想定している．実行回数 100 回での結果をまとめている．全体的に拡張リング法は提案手法と比べ，ばらつきが激しい．x 座標 3 では送信回数が拡張リング法は約 165 に対して，提案手法は約 70 である．x 座標 56 では送信回数が拡張リング法は約 195 に対して，提案手法は約 65 である．これより，TTL が多くなればなるほど，提案手法が良いと考えられる．逆に，x 座標 1 では送信回数が拡張リング法は約 25 に対して，提案手法は約 40 である．x 座標 81 では送信回数が拡張リング法は約 10 に対して，提案手法は約 20 である．これより，TTL が少なければ，拡張リング法が良いと考えられる．

表 4．1 はノード数 125 でフィールドサイズ 200×200 での送信数の平均である．拡張リング法は 46．71，提案手法は 34．30 となった．拡張リング法はノード配置状況で送信数が約 10～200 の変化があったため，実行数の増加に伴い平均値の変動も多いと考えられる．提案手法は，拡張リング法程のバラつきはなく，安定した送信回数となった．

図 4．2 はノード数 125 でフィールドサイズ 50×50 での送信数の結果である．このサイズは狭い範囲，ノードが密集する配置を想定している．実行回数 100 回での結果をまとめている．図 4．1 とは大きく違い，すべての実行時で拡張リング法が提案手法を下回る結果となった．これはサイズが小さく，ノードが密集しているため，TTL が小さい値でも送信回数が多くなってしまうからである．このサイズでは拡張リング法が良いと考えられる．

表 4．2 はノード数 125 でフィールドサイズ 50×50 での送信数の平均である．拡張リング法は 104．44，提案手法は 180．18 となった．約 2 倍の差があり，ノード密集環境での提案手法は非効率的な結果となった．

図 4.3 はノード数 125 でフィールドサイズ 40×40～200×200 での送信数の結果である．フィールドサイズ N×N の時の実行回数 100 回で平均値をとり，結果をまとめている．x 座標約 85 を境に提案手法と拡張リング法の送信回数の逆転が起きた．サイズが大きくなればなるほど，拡張リング法は増大していき，提案手法は約 100 に収縮する結果となった．また，x 座標約 40 以下からは両手法とも送信回数にあまり変化が見られなかった．これは，サイズが 40×40 以下からはノードの配置に変化が起きにくくなり， TTL=1～3 で R を発見できてしまうためと考えられる．x 座標 230 を最大値にしているのは，これ以上サイズを広げると R を発見できなくなってしまうためである．

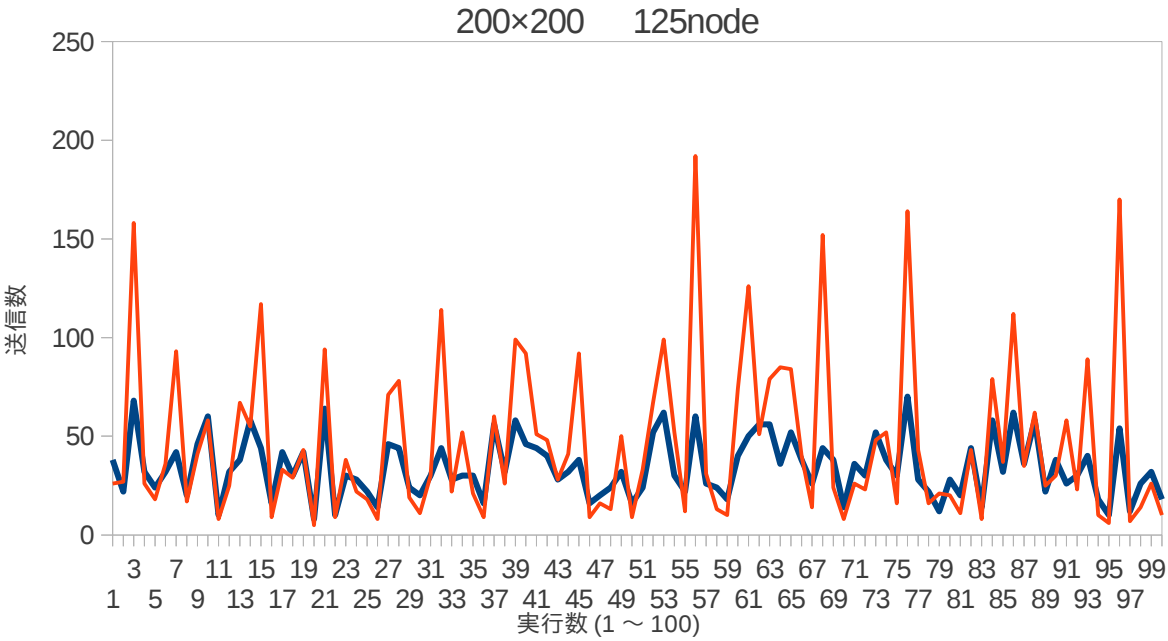


図 4．1　200×200 ノード数 125 実行回数 100 回における結果 青線提案 赤線拡張リング

表 4.1 ノード数 125 (x, y)=(200, 200)における送信平均と分散値

	拡張リング法	提案手法
平均	46.71	34.3
分散	1654.51	223.55

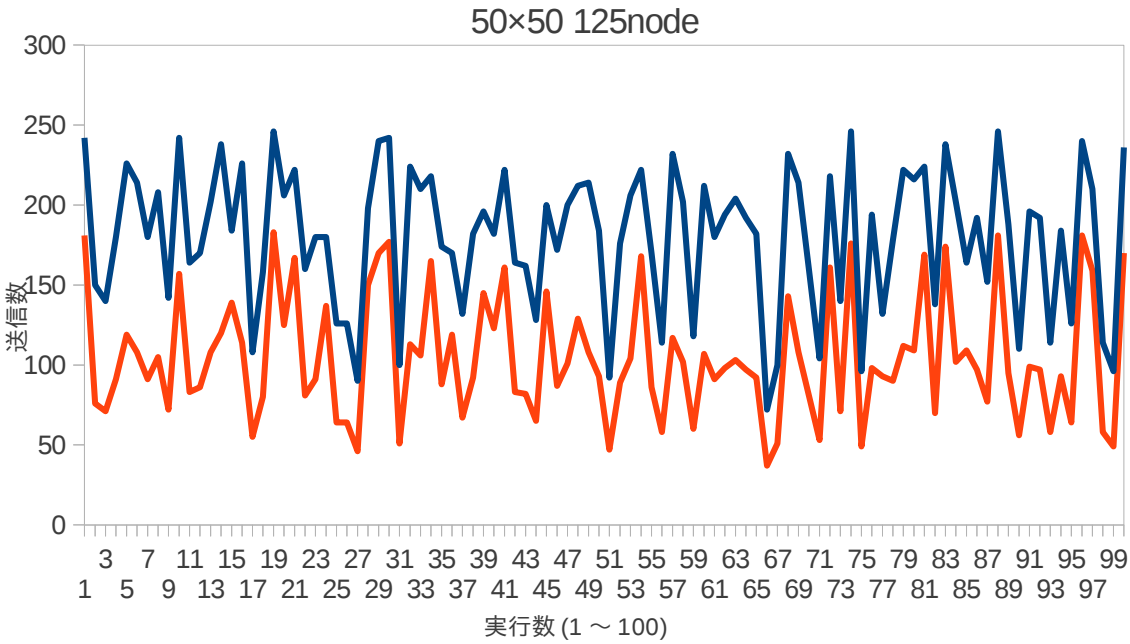


図 4．2　50×50 ノード数 125 実行回数 100 回における結果 青線提案 赤線拡張リング

表 4.2 ノード数 125 (x, y)=(50, 50)におけ送信平均と分散値

	拡張リング法	提案手法
平均	104.44	180.18
分散	1467.53	1946.17

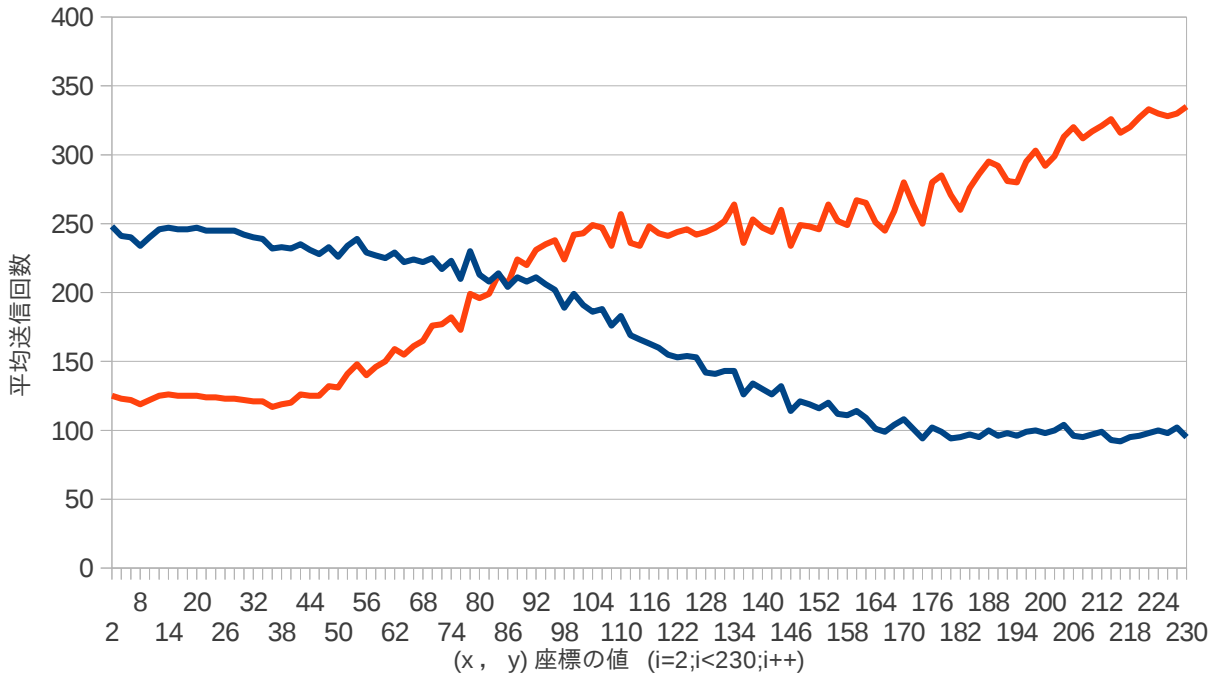


図 4．3　N×N(2～230)　ノード数 125　実行回数 100 における送信回数平均値の結果

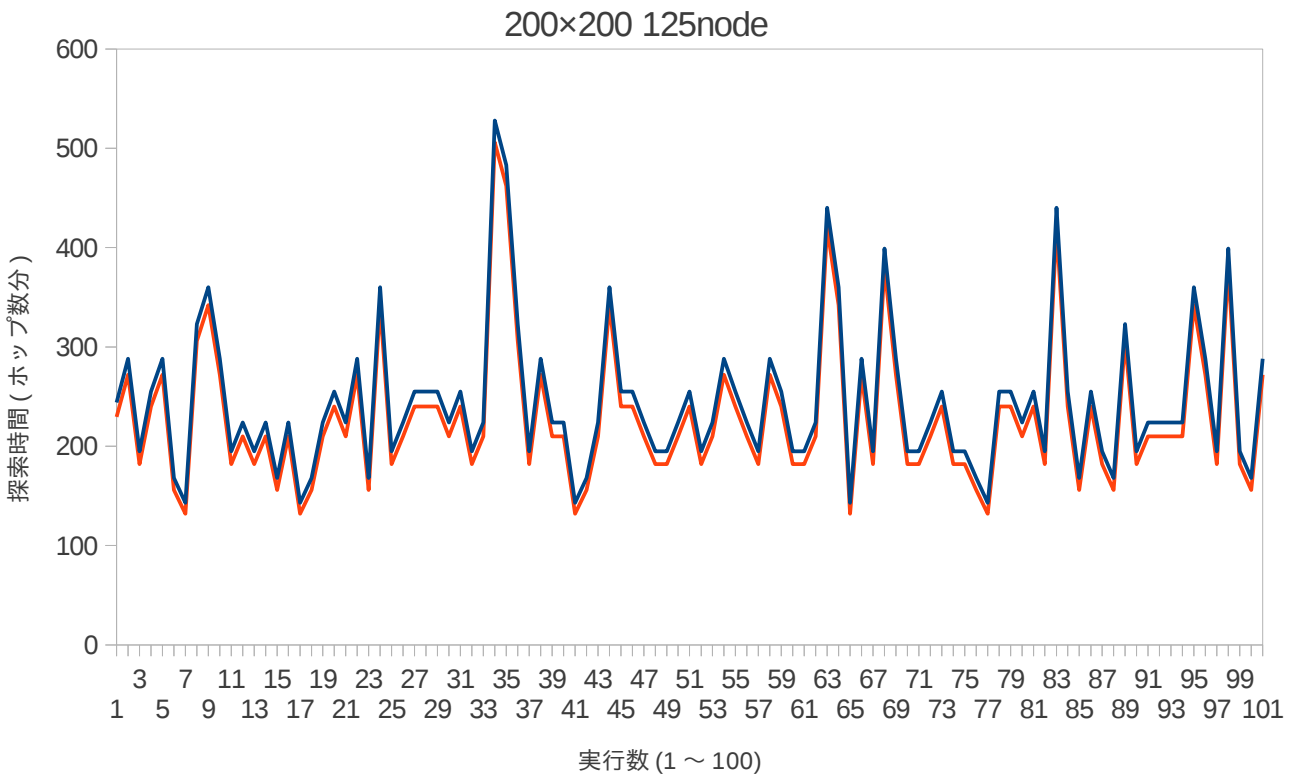
青線提案手法　赤線拡張リング

図 4．4，図 4．5，表 4．3 は時間のグラフである．

図 4．4 はノード数 125 でフィールドサイズ 200×200 での時間(ホップ数)の結果である．実行回数 100 回での結果をまとめている．全ての実行時で拡張リングが先に探索終了の結果となった．このことより，全体の待ち時間は拡張リング法が短いとわかる．

表 4．3 はノード数 125 でフィールドサイズ 200×200 での時間の平均である．拡張リング法は 230，提案手法は 240 となった．このサイズでは約 10 ホップ分の時間差が生じることがわかった．通信時間に関してはあまり差がないと考えられる．

図 4．5 はノード数 125 でフィールドサイズ 40×40～200×200 での時間の結果である．フィールドサイズ N×N の時の実行回数 100 回で平均値をとり，結果をまとめている．全ての実行時で拡張リングが先に探索終了している．また，TTL が増える分だけ拡張リングと提案との差が広がっていくことがわかる．これらの値の差は，提案手法の時間 - 拡張リング法の時間=TTL であり，提案手法の時間は TTL 分だけ拡張リング法より遅れて探策終了となる．また，この計算結果は図 4．5 のグラフと一致したため予測通りの結果を求めることができた．



15 図 4. 4　200×200 ノード数 125 実行回数 100 回における結果 青線提案 赤線拡張リング

表 4.3 ノード数 125　(x, y)=(200, 200)における時間平均と分散値

	拡張リング法	提案手法
平均	229.56	244.06
分散	5001.23	5313.13

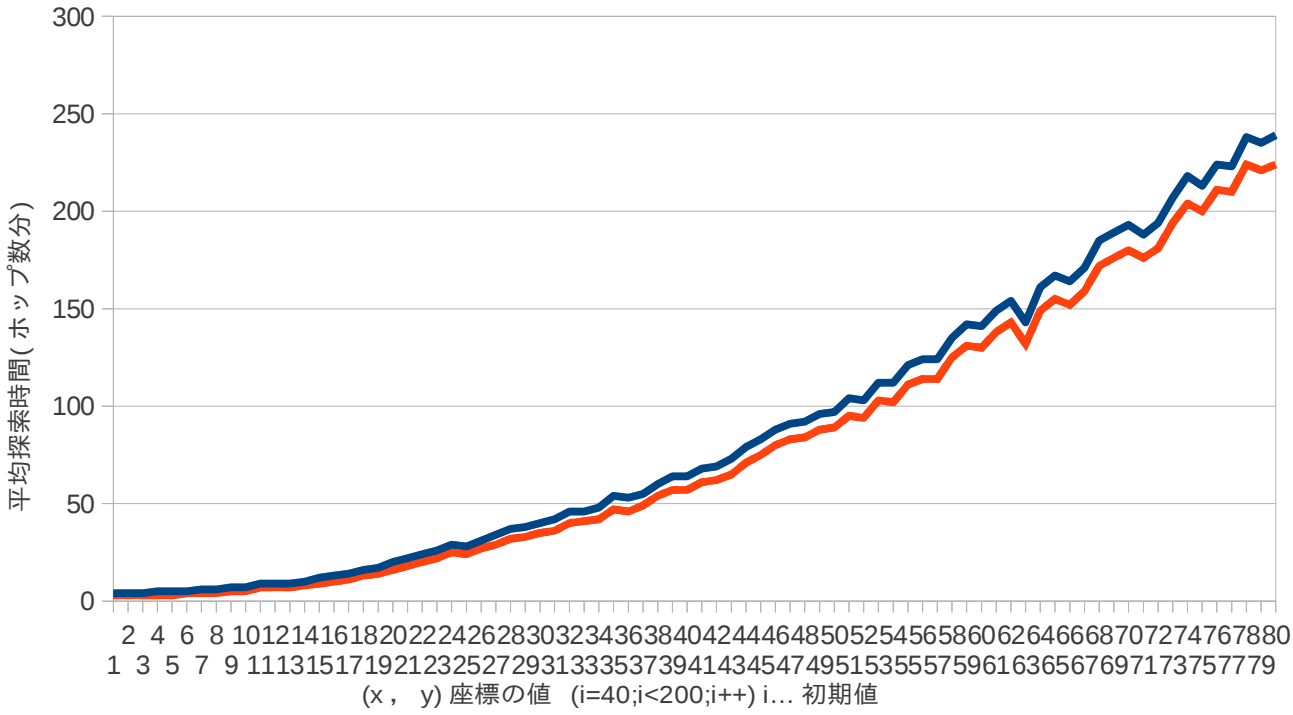


図 4. 5　N×N(40～200) ノード数 125 実行回数 100 回における時間平均値の結果

青点線提案 赤点線拡張リング

4．4　考察

図 4．3 の結果よりサイズが約 85 を境に拡張リング法と提案手法との送信回数の逆転が起こっている．これは，サイズが狭く，ノードが密集すればするほど，拡張リング法が提案手法よりも下回ったと考えられる．逆に，サイズが広く，ノードが疎らになればなるほど，拡張リング法が提案手法よりも上回ったと考えられる．これ以下の様に考えると説明できる．

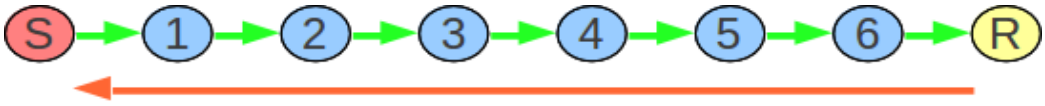


図 4．6　ノードが疎らな場合での経路探索例

まず最もノードが疎らな場合を考える．隣接ノードとして 1 ノードずつ検出された場合経路設計の結果は図 4．6 のようになる．この場合拡張リング法での経路設計完了までにかかる送信数は(1+2+3+4+5+6+7)緑線+(7)R の返信赤線によって合計 35 の送信回数で経路設計完了となる．提案手法の場合(1+1+1+1+1+1+1)緑線+(1+1+1+1+1+1+1)R の返信赤線+(7)各ノードへの転送停止パケットによって合計 21 の送信回数で経路設計完了となる．

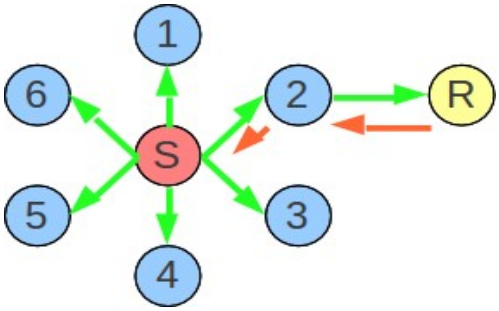


図 4．7　ノードが密集しているの場合での拡張リング法例

次に，ノードが密集した場合を考える．送信元の隣接ノードとして周囲に複数ノードを検出し場合その 2 ホップ先に送信先がある場合の経路設計の結果は図 4．7 のようになる．この場合拡張リング法での経路設計完了までにかかる送信数は(1)S の発信+(1+1+1+1+1+1)1～6 の発信緑線+(2)R からの返信赤線によって合計 10 の送信回数で経路設計完了となる．

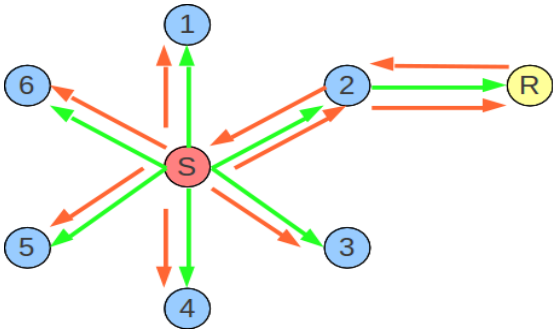


図 4．8　ノード数が密集しているの場合での提案手法例

提案手法の場合は図 4.8 で示すように，(1)S の発信+(1+1+1+1+1+1)ノード 1～6 の発信緑線+(2)R の返信赤線+(7)各ノードへの転送停止パケットによって合計 16 の送信回数で経路設計完了となる．

このことより図 4．3 の(約 85×85～)結果はサイズが広範囲なことにより図 4．6 に近いノード配置となるため，送信回数が提案手法が拡張リング法よりも下回ったと考えられる．また，図 4．3 の(約～85×85)結果はサイズが狭く，ノード間が密集しているため，図 4．7，図 4．8 に近いノード配置となるため，送信回数が提案手法が拡張リング法よりも上回ったと考えられる．

第 5 章　おわりに

本論文では普及されている携帯端末を使って MANET を構築する技術である，マルチホップ通信のルーティングプロトコルの 1 つである拡張リング法のアルゴリズムを改良することによって，更なる消費電力の低下を検証した．拡張リング法ではパケット送信の際に重複が起きている．これは中継ノードが多くなればなるほど消費電力が増える．更に，送信元ノードを含めた隣接ノードに集中して消費電力が生じ，バッテリー切れの問題も考えられた．そこで提案手法では，探策パケットの重複を起こさないようにノードを制御する手法を考えた．拡張リング法と提案手法をプログラミングし，送信回数，接続までのターン数を比較することでそれぞれの特徴を調べた．

実験結果よりノードが密集している環境でのネットワーク構築は拡張リング法によるルーティグが良く．送信先ノードと送信元ノードの間が広がり，中継ノードが増えれば増えるほど提案手法が良いと考えられる．経路探索時間は TTL 値の分だけ提案手法がかかり，TTL が大きくなればなるほど，拡張リング法と提案手法の差が広がっていくと考えられる．

第 2 章で述べたようにネットワーク構築の際に既存ルーティングプロトコルにハイブリッド型という手法が存在する．この手法をアドホックネットワークにも採用することによって，一定の距離でルーティングプロトコルを使い分けることができる．今回の実験結果から考えられることはノード密集地帯と想定できる環境では拡張リング法，ノード配置が疎らと想定できる環境では提案手法を採用することによって，それぞれの利点を持ち合わせるが出来，更なる消費電力の低コスト化を望めると考えられる．

謝辞

本研究を進めるにあたり，様々なご指導を頂きました三好力教授に深く感謝いたします．
また，研究室の皆様からも様々な知識，示唆を頂き感謝します．

参考文献

[1]事業者別月末契約数　電気通信事業者協会発表
http://www.tca.or.jp/database/2012/03/

[2]無線環境における通信速度と領域の関係
http://www.venture.nict.go.jp/trend/sensor/2_1.html

[3]アドホックネットワークのイメージ
yamachan.shse.u-hyogo.ac.jp/...2/wirelesstrans.pdf

[4] 濱口圭介、三好力、アドホックネットワークの低消費電力経路探策法の検討、情報処理学会第 75 回全国大会（2013 発表予定).

付録

```
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#define NODE 50/*ノード数*/
#define radius 30/*半径*/
#define N_START 0 /* 送信元 */
#define N_END 19 /* 送信先 */
#define map 50
#define START 20
#define END 20

struct status{//ノード割り当てよう
  int number;
  int x;/*x 座標*/
  int y;/*y 座標*/
  int used;
};
struct route{//ノード接続よう
  int connect[NODE];//ノード連結格納
  int kyori;//隣接連結ノードとの距離
  int send;//送信回数
  int xx;
  int yy;
};

int main(){
  FILE *kakutyoufile;
  FILE *teianfile;
  FILE *kakutyoufile1;
  FILE *teianfile1;
  FILE *kakutyoufile2;
  FILE *teianfile2;
  FILE *heikinfile;
  FILE *heikinfile1;
  FILE *heikinfile2;
  FILE *heikinfile3;
  int a,b,c,d,e,A,B;
  int i,j;//座標
  int k;
  int l,m,n;//元隣接ノード
  int x;
  int null_a,null_b;
  int v,w,y,z;
  int p,q;//tateyoko 増分
  int amount=0;
  int amount_kyori=0;
  int reject_kyori=0, reject_send=0;
  int yoko,tate;
  int send_a;
  int send_b;
  int send_c;
  int send_d;
  int send_amount=0;
  int kyori_a;
  int kyori_b;
  int kyori_c;
  int kyori_d;
  int kyori_amount;
  int reply_send=0,reply_kyori=0;
  int kakutyou_turn = 0,teiann_turn =0;
      int   heikin_t_s=0,   heikin_t_t=0,   heikin_k_s=0,
heikin_k_t=0;
  struct status node[NODE];
  struct route code[NODE][NODE];//構造の配列宣言
  srand((unsigned) time(NULL));/* 乱数系列の変更 */
  /*100×100 の座標にノードをランダムに配置*/
  /*ランダムに配置したノードの x[0]y[0]を送信元，x[NODE-2]y[NODE-2]
を送信先に決定*/

  heikinfile = fopen("拡張回数平均.txt", "w");
  heikinfile1 = fopen("拡張時間平均.txt", "w");
  heikinfile2 = fopen("提案回数平均.txt", "w");
  heikinfile3 = fopen("提案時間平均.txt", "w");
 for(B=0;B<100;B++){
  for(A=0;A<100;A++){
 RESTART:
    for (i=1;i<NODE;i++){
    node[i].number = i;//ノード番号
    node[i].x = rand()%(map+B);
    node[i].y = rand()%(map+B);
    node[i].used = 0;
    }

  node[0].number = 0;//ノード番号
  node[0].x = 0;
  node[0].y = 0;
  node[0].used = 0;

  for(yoko=0;yoko<NODE;yoko++){
    for(tate=0;tate<NODE;tate++){
      code[tate][yoko].connect[0] = -1;
      code[tate][yoko].kyori = 0;
      code[tate][yoko].send = 0;
      code[tate][yoko].xx = -1;
      code[tate][yoko].yy = -1;
    }
```

```
  }


  code[0][0].connect[0] = node[0].number;

  code[0][0].connect[1] = -1;

  code[0][0].kyori = 0;

  code[0][0].send = 0;

  code[0][0].xx = node[0].x;

  code[0][0].yy = node[0].y;


   //printf("初期送信先ノードは(%d,%d)\n初期送信元ノードは(%d,%d)\n",node[0].x,node[0].y,node[NODE-2].x,node[NODE-2].y);

  for(tate=0;tate<NODE;tate++){

    amount = 0;

    p = 0;

    q = 1;

    for(yoko=0;yoko<NODE;yoko++){

      if(code[tate][yoko].xx == -1) break;

      for(j=1;j<NODE;j++){//ノード1～NODEまでの検索

          if(radius*radius     >     ((node[j].x-code[tate][yoko].xx)*(node[j].x-code[tate][yoko].xx)

                                      +(node[j].y-code[tate][yoko].yy)*(node[j].y-code[tate][yoko].yy))){//ユークリッド距離での隣接ノード検索

            if(node[j].used == 1) continue;

            node[j].used = 1;

            if(p == 0){

              for(a=0;a<NODE;a++){//1

                if(code[tate][yoko].connect[a]==-1){

                   for(e=0;e<a;e++){

                         code[tate+1][yoko].connect[e]    = code[tate][yoko].connect[e];

                   }

                   code[tate+1][yoko].connect[a]=node[j].number;//ノード番号のみの配列

                   code[tate+1][yoko].connect[a+1]=-1;

                   code[tate+1][yoko].kyori   =   code[tate][yoko].kyori

                                      +(node[j].x-code[tate][yoko].xx)*(node[j].x-code[tate][yoko].xx)

                                      +(node[j].y-code[tate][yoko].yy)*(node[j].y-code[tate][yoko].yy);

                   code[tate+1][yoko].send    =  code[tate][yoko].send +2;

                   code[tate+1][yoko].xx     = node[j].x;//現ルートの最端座表に更新

                   code[tate+1][yoko].yy    = node[j].y;

                   p = 1;

                   amount = 1;

                   if(j==NODE-2){//送信先発見

                     reply_send = code[tate+1][yoko].send;

                           reply_kyori    =    code[tate+1][yoko].kyori;

                     goto FLAG;//探索終了

                   }

                   break;

                }

              }//1

            }

            else{//4

              for(b=0;b<NODE;b++){

                if(code[tate][yoko].connect[b]==-1){

                   for(d=0;d<b;d++){

                        code[tate+1][yoko+q].connect[d]   = code[tate][yoko].connect[d];

                   }

                   code[tate+1][yoko+q].connect[b]=node[j].number;//ノード番号のみの配列

                   code[tate+1][yoko+q].connect[b+1]=-1;

                   code[tate+1][yoko+q].kyori  =  code[tate][yoko].kyori

                                      +(node[j].x-code[tate][yoko].xx)*(node[j].x-code[tate][yoko].xx)

                                      +(node[j].y-code[tate][yoko].yy)*(node[j].y-code[tate][yoko].yy);

                   code[tate+1][yoko+q].send   = code[tate][yoko].send +1;

                   code[tate+1][yoko+q].xx                =  node[j].x;//現ルートの最端座表に更新

                   code[tate+1][yoko+q].yy    = node[j].y;

                   q++;

                   if(j==NODE-2){//送信先発見

                        reply_send  =  code[tate+1][yoko+q-1].send;

                       reply_kyori  =  code[tate+1][yoko+q-1].kyori;

                     goto FLAG;//探索終了

                   }

                   break;

                }

              }//4

            }

            else{

              if(j==NODE-1){//括弧5

                if(amount==0){//tate内でもう隣接ノードがなかった場合

                   reject_kyori = reject_kyori + code[tate][yoko].kyori;

                   reject_send  =  reject_send  +  code[tate][yoko].send;

                //for(k=yoko;k<NODE;k++){//経路の削除

                //  code[tate][k]=code[tate][k+1];

                //  if(code[tate][k+1].xx==-1){

                //    break;

                //  }

                //}

                /*for(c=0;c<NODE;c++){

                   code[tate][NODE-1].connect[c] = -1;

                   if(code[tate][NODE-1].connect[c+1]     = -1); break;

                   }

                   code[tate][NODE-1].kyori = 0;

                   code[tate][NODE-1].send = 0;
```

```
                code[tate][NODE-1].xx = -1;
                code[tate][NODE-1].yy = -1;
                if(tate==1&&yoko==0){//全 tate で隣接ノードがなかった場合
                goto FLAG;//終了
                }
                }*/
            }//括弧５
          }
        }
    }
   }
  }
  FLAG:
  if(j==NODE-2){//送信先発見できた場合
    for(send_a=0;send_a<NODE;send_a++){
     for(send_b=1;send_b<NODE;send_b++){
         send_amount   =   send_amount   +   code[send_b][send_a].send;
         if(code[send_b][send_a].send == 0) break;
     }
    }
    send_amount = send_amount + reply_send;

    kakutyou_turn = (tate+1)*((tate+1)+1);

    heikin_k_s = heikin_k_s + send_amount;
    heikin_k_t = heikin_k_t + kakutyou_turn;
    send_amount = 0;
    kakutyou_turn =0;

    for(send_c=0;send_c<NODE;send_c++){
     for(send_d=1;send_d<NODE;send_d++){
         if(code[send_d][send_c].send == 0){
            send_amount  =  send_amount  +  code[send_d-1][send_c].send;
            if(code[send_d][send_c].send == 0) break;
         }
      }
    }
    send_amount = send_amount + send_amount;
    teiann_turn = (tate+1)*((tate+1)+2);

    heikin_t_s = heikin_t_s + send_amount;
    heikin_t_t = heikin_t_t + kakutyou_turn;
    send_amount = 0;
    teiann_turn =0;
  }
  else{
    //printf("ルーティング失敗\n\n");
    goto RESTART;
  }
  }
  fprintf(heikinfile,"%d\n",heikin_k_s%100);
  fprintf(heikinfile1,"%d\n",heikin_k_t%100);
  fprintf(heikinfile2,"%d\n",heikin_t_s%100);
  fprintf(heikinfile3,"%d\n",heikin_t_t%100);
  heikin_k_s=0;
  heikin_k_t=0;
  heikin_t_s=0;
  heikin_t_t=0;
 }
    fclose(heikinfile);
    fclose(heikinfile1);
    fclose(heikinfile2);
    fclose(heikinfile3);
  return(0);
}
```