修 士 論 文

アドホックネットワークのための
マルチルートルーティングプロトコルの開発

静岡大学大学院 工学研究科 電気電子工学専攻
和田研究室

5073-0154　杉本 孝広

平成 21 年 2 月 9 日

目次

1 序論

1.1 研究の背景

1.1.1 モバイル・アドホックネットワーク

モバイル・アドホックネットワーク (MANET, Mobile Ad-hoc Network)[1] とは、移動通信端末によりアドホックに形成され、基地局などの固定的な通信インフラストラクチャや、集中管理機構に依存することなく構築されるネットワークのことである。

携帯電話や公衆無線 LAN などの現在一般的な移動体通信では、移動端末は固定インフラに接続された基地局を介して通信を行っている。しかし、MANET では基地局のような存在を必ずしも必要とせず、対等な関係にある端末群により、P2P なマルチホップネットワークを構成している。

下記に MANET の特徴を挙げる。

1. 固定的なネットワークインフラに依存しない
2. 集中管理機構がない
3. マルチホップネットワーク
4. 移動無線端末により構成 (ネットワークトポロジが動的に変化)

このようなネットワーク形態は、災害時のバックアップ用通信などのインフラが使用できない場合や、ITS における車車間通信などでの利用が研究されている。また、ユビキタスネットワークを実現するためのネットワーク形態としても注目されている。

1.1.2 ルーティング (経路制御)

マルチホップネットワークにおいては、データは端末同士による複数段の中継、すなわちマルチホップ中継により転送される。その際、転送経路を適切に決定する経路制御、すなわちルーティングが適切に行われることが重要となる。有線ネットワークにおいては、RIP[5]、OSPF[13]、BGP[16] の 3 つのプロトコルが広く普及しているが、これらのプロトコルは MANET においては機能しない。

MANET は、移動端末によりネットワークが構成されるため、トポロジ、すなわちネットワーク構造が頻繁に変化する。また、各ホップの通信が無線リンクであり、有線に比べて不安定である。さらに、電力・計算機・電波帯域などの資源も限られている。MANET を機能させるためには、そのような MANET の特性に合わせたルーティングプロトコルの開発が必要である。現在までに提案されている代表的な MANET 向けルーティングプロトコルとして DSR[9]、AODV[15]、OLSR[4] などがあり、現在標準化に向けた作業が進んでいる。

1.1.3 ルートダイバーシティ

データの伝送路として、複数の経路が利用できる場合、それらの経路を同時に利用して経路の品質劣化を補い、通信効率を上げる手法が考えられる。複数の経路を用い、その多様性 (ダイバーシティ) を利用して通信を行う方法を「ルートダイバーシティ」と呼ぶ [20]。

マルチホップネットワークでこの手法を用いるためには、複数経路の構築に対応したルーティングプロトコルが必要となる。2.4.1 に述べるように、複数経路の構築を目指したプロトコルはこれまでにもいくつか提案されているが、それらは経路消失時の代替や負荷分散を目的としたものであり、ルートダイバーシティを指向

したものではない。

1.1.4 ネットワークシミュレータ ns-2

ns-2[2] は、ネットワーク研究向けのイベントシミュレータである。指定した環境下でのネットワークの振る舞いを調べたり、新規に開発したプロトコルの評価に用いることが出来る。各レイヤのプロトコルが多数実装済みであり、世界中の研究者に利用されている。

現行バージョンの ns-2 には、CMU Monarch プロジェクト [3] により開発された無線シミュレーション機能が取り込まれており、無線通信分野、特に MANET 向けルーティングプロトコルの評価では多く利用されている。

また、ns-2 はオープンソースソフトウェアであり、GPL(GNU General Public License) など[*1]のライセンスの元でリリースされている。従って、無償で利用可能であり、改変・再配布も行うことが出来る。MANET のシミュレーションに用いることのできるその他のネットワークシミュレータとしては、QualNet(旧 GloMoSim)、OPNET などがあるが、いずれも商用アプリケーションである。

1.2 本研究の概要および目的

本研究では、MANET における通信性能を複数経路の同時利用により向上させることを目的とし、そのために必要な複数経路に対応したルーティングプロトコルの開発を行う。

複数経路を考慮したルーティングプロトコルは、これまで複数提案されており、それらのプロトコルでは、切断された経路の再構築に要する時間・パケット数の削減や、負荷の分散でなどを目的としている。一方、本研究では、ルートダイバーシティの観点から、複数のルートを一度に用い、通信の信頼性を向上させることを目指す。

本論文においては、複数経路に対応したルーティングプロトコルを AODV を元に開発し、そのプロトコルをネットワークシミュレータ ns-2 上で実装する。実装したプロトコルを用いて、単一経路、負荷分散的経路利用などと比較し、複数経路同時利用による通信性能の変化を確認する。

1.3 本論文の構成

本論文では、まず第 2 章で MANET の特徴とマルチルート通信について解説する。次に、第 3 章でルーティングプロトコルの開発について述べ、第 4 章で開発したプロトコルのシミュレーションによる検証結果を示す。最後に、第 5 章で本研究を総括し、更なる研究課題について述べる。

[*1] 修正 BSD ライセンス等、GPL と互換性のあるライセンスでリリースされたコードも含まれている。

2　モバイル・アドホックネットワーク

2.1　モバイル・アドホックネットワーク (MANET)

モバイル・アドホックネットワーク (MANET, Mobile Ad-hoc Network)[1] は、移動する無線端末のみによって自律的に構築されるネットワークである。

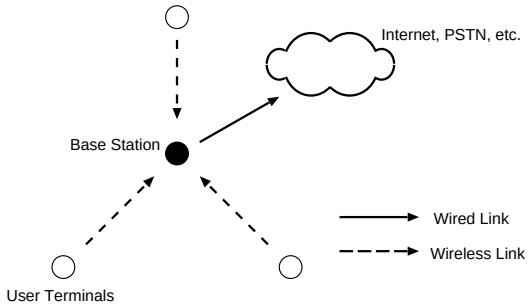

図 1　固定インフラを伴う移動体通信の模式図

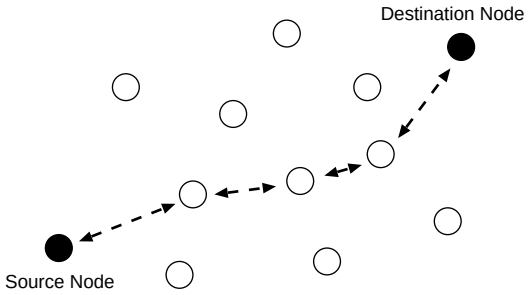

図 2　MANET の模式図

MANET においては、基地局などの固定的な通信インフラストラクチャや、集中管理機構に依存することなく、端末相互の通信を行うことが可能となる。携帯電話や公衆無線 LAN などの現在一般的な移動体通信では、図 1 のように、移動端末は固定インフラに接続された基地局を介して端末同士や他のネットワークとの通信を行っている。しかし、MANET では、基地局のような存在を必ずしも必要とせず、図 2 のように、対等な関係にある端末群がバケツリレー的にデータを転送するマルチホップネットワークを構成する。また、全てのノードが移動可能な状態にあるため、ネットワークのトポロジ (構造) の変化に追従できるような仕組みを備えている。

このようなネットワーク形態は、災害などにより既設インフラが破壊された場合のバックアップとしての利用や、ITS (Intelligent Transport Systems) における車車間通信などでの利用が研究されている。また、ユビキタスネットワークを実現するためのネットワーク形態としても注目されている。

以上の特徴をまとめると、下記のようになる。

1. 固定的なネットワークインフラに依存しない
2. 集中管理機構がない
3. マルチホップネットワーク
4. 移動無線端末により構成 (ネットワークトポロジが動的に変化)

2.2　ルーティング (経路制御)

MANET のようなマルチホップネットワークでは、送信元・送信先の間の中継経路は固定されておらず、何らかの仕組みを用いて経路を構築する必要がある。また、構築しうる経路は複数ある場合があり、それらの中から最適な経路を選択する必要がある。この経路決定をルーティング (経路制御) と呼び、その機能を担うプロトコルをルーティングプロトコルと呼ぶ。

2.2.1 有線ネットワークにおけるルーティング

現在稼働しているマルチホップネットワークの中で最大のものはインターネットである。有線を中心とした現在のインターネットでは、RIP(Routing Information Protocol)[5]、OSPF(Open Shortest Path First)[13]、BGP(Border Gateway Protocl)[16] の 3 つのプロトコルが普及しており、ネットワークの規模・構成・運用方法により使い分けられている。RIP・OSPF は、IGPs(Interior Gateway Protocols) と呼ばれ、AS(Autonomous System, 自律システム)[*2]内で使用される。BGP は、EGPs(Exterior Gateway Protocols) と呼ばれ、AS 間のルーティングに使用される。

このうち、IGPs である RIP, OSPF で使用されている手法は、MANET におけるルーティングプロトコルの基礎となっている。

■RIP　RIP(Routing Information Protocol) は、最古のルーティングプロトコルであり、現在でも広く利用されている。RIP は、距離ベクトル (Distance Vector) 型のプロトコルと呼ばれ、相手先ネットワークの方向 (インターフェース番号) と距離 (ホップ数) の情報を元にルーティングを行っている。各ルータは、隣接するルータ[*3]から送られてきた情報に、自分が保持している情報を付加して再送信する。この動作を繰り返し行うことによりネットワークにある各ルータへの経路とそこまでの距離を知ることが出来る。RIP では、経路表 (ルーティングテーブル) がネットワーク上をそのまま流通することになる。

■OSPF　OSPF(Open Shortest Path First) は、リンク状態 (Link State) 型のプロトコルであり、大規模なネットワークでは機能しないという RIP の問題点を解決することができように設計されている。リンク状態型のルーティングでは、各ルータが、自分の接続しているネットワークの情報をネットワーク全体に配信する。これらの情報を収集し、各ルータはネットワーク全体の構造 (ネットワークトポロジ) を把握、トポロジデータベースを作成する。その情報を元に Dijkstra の Shortest Path First アルゴリズムを使用し、最短経路を算出、ルーティングテーブルを作成する。ネットワーク情報の全体への配信は、保持している情報の変化があった時に差分情報として送信される。

RIP、OSPF は、中継するルータがルートを決定する役割を担うが、これ以外にも、送信元が相手先までの経路を決定するソースルーティング (Source Routing) がある。送信元がネットワーク全体の構造を把握していることが前提であるが、中継ルータの負荷を減らすことが出来る。IP(Internet Protocol) ではルーティングプロトコルが作成するルーティングテーブルと、ソースルーティングによる経路の両方が利用できるように設計されている。

2.2.2 移動無線環境における問題

2.1 に述べたような MANET が実現できれば、その場に合わせたネットワークを柔軟かつ自律的に構成することが可能である。そのためには有線ネットワーク同様にルーティングプロトコルを用いてネットワークを構築する必要がある。しかし MANET では、干渉、減衰、ノードの移動・消滅等のさまざまな不安定要因が存在するため、有線ネットワークのルーティングプロトコルをそのまま用いても機能しない。また、電力や帯域などの資源が限られており、それらを浪費せずに利用することも考慮する必要がある。

[*2] インターネットの管理単位で、同一の運用ルールで稼働しているネットワークシステムのこと。例えば静岡大学の学内ネットワークは 24268 番の AS である。

[*3] ルーティングを行うコンピュータ。MANET においては、参加している各ノードがルーティングを行うが、インターネットにおいてはネットワークの各セグメントの出入口に設置されたコンピュータがルーティングを行うのが一般的である。

不安定な状況を作り出す要因のモデルとして、フェーディングやシャドウィングがある。

■マルチパス・フェーディング　複数の経路から伝搬し位相のずれた同じ電波を受信することにより受信レベルが下がる自波干渉 (マルチパス) の一種で、直接波と反射波の位相差によって、信号の受信レベルが時変動する現象である。

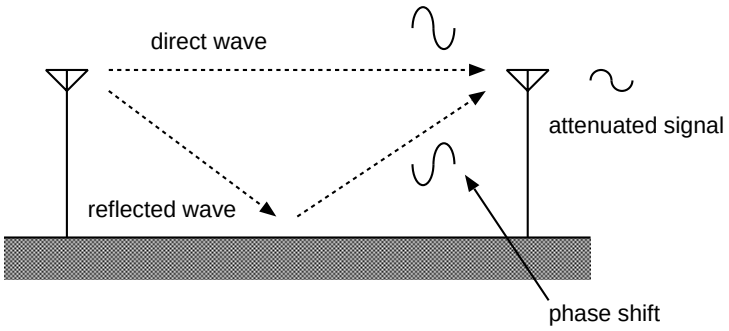

図 3　マルチパス・フェーディング

■シャドウィング　歩行者や自動車などが受信機と送信機の間の伝搬経路上に入り込んでしまい、受信機が送信機から見て電波の影（シャドウ）に入ってしまうことにより信号の受信レベルが一時的に低下してしまう現象である。

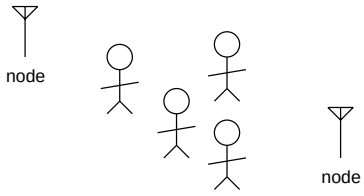

図 4　シャドウィング

2.3　MANET におけるルーティング

2.1 で挙げたような特性を持つ MANET を、2.2.2 で述べたものをはじめとする干渉・妨害が存在する環境下で構築するためには、特別に工夫したプロトコルが必要である。有線ネットワークや基地局までのシングルホップ無線通信を前提とした既存のプロトコルは正常に動作しない。そのため、そのような環境下でも動作するように考えられた多数のプロトコルが提案されている ([18] などを参照)。本節では、現在提案されている種々のプロトコルの基礎となっている DSR(Dynamic Source Routing) および AODV(Ad-hoc On-demand Distance Vector) と、MANET のルーティングプロトコルとして前出の 2 つと並んで研究が進んでいる OLSR(Optimized Link State Routing) を紹介する。

MANET のルーティングプロトコルは、経路構築のタイミングにより 2 つに分類できる。データを送信する必要が生じたときに初めてルーティング動作を行うリアクティブ型と、事前に制御メッセージを交換し経路表を構築しておくプロアクティブ型である。DSR および AODV はリアクティブ型に分類され、OLSR はプ

ロアクティブ型に分類される。以下、この分類に沿って解説する。

2.3.1 リアクティブ型プロトコル

まず、DSR (Dynamic Source Routing)[9] は、ソースルーティングを採用しているプロトコルであり、送信元がルーティングを行うことになる。しかし、送信元は最初の時点では経路情報を保持していない。通信が要求されると、Route Discovery(ルート探索) を開始し、RREQ(Route REQuest) パケットを一斉送信する。この動作はパケットをネットワーク上に洪水のように流すためフラッディングと呼ばれる。このパケットを受信したノードは、宛先が自ノードでない場合には、RREQ パケットに自身を経由したことを書き込み、さらにとなりのノードへ転送する。この作業を繰り返すことによりネットワーク全体を探索することが出来る。なお、同じ RREQ パケットを受信した場合には破棄することになっている。受信したパケットが自ノード宛である場合には、RREP(Route REPly) メッセージを送信元に向けて返信する。また、経路構築時間を短縮し、経路制御パケットを削減するため、自ノードのルートキャッシュに宛先ノードへのルートが存在する場合も、そこまでのルートを追加して返信する。この際、RREQ パケットには経由してきたノードが書き込まれているので、その経路を逆にたどることにより送信元へ到達することが出来る。返ってきた RREP パケットの情報を元に、送信元はデータを送信することになる。

一方、AODV (Ad-hoc On-demand Distance Vector)[15] は、距離ベクトル型のプロトコルである。RREQ および RREP パケットを使用した経路探索については DSR と同様であるが、経路は中継する各ノードが保持している。各ノードでは、どの宛先ノードに到達するにはどのノードに転送すれば何ホップで到達するかという情報を保持しており、この情報を各ホップで順にたどることによりデータの送受信が可能となる。

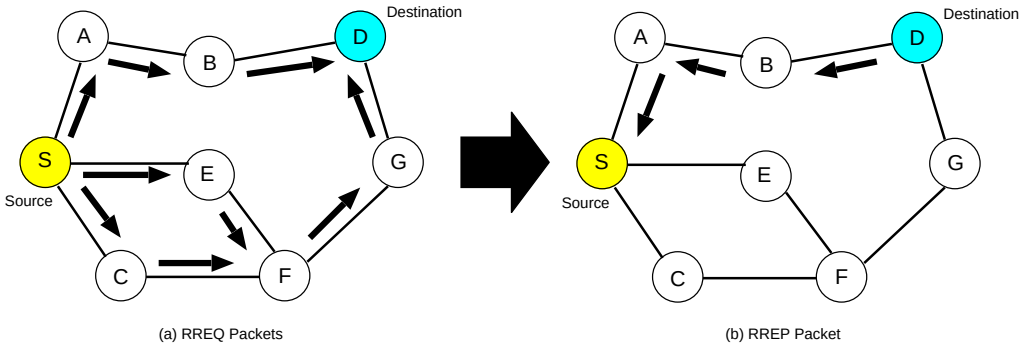

図 5　RREQ, RREP を利用した経路構築

リアクティブ型プロトコルは、送信の必要が生じたときに初めて経路を構築するため、ネットワーク構成の変化に比較的早く対応できるが、送信時のオーバーヘッドは大きくなる。

2.3.2 プロアクティブ型プロトコル

プロアクティブ型プロトコルの代表例である OLSR (Optimized Link State Routing)[4] は、有線でも使用されているリンク状態アルゴリズムを MANET 向けに最適化したものである。OLSR では、接続情報 (リンク状態) の配信のために、MPR (Multipoint Relay) と呼ばれるネットワーク全体にフラッディングを行うために必要な最低限の中継ノード群を選定し、それらのみに配信を行わせることにより、フラッディングの効率化を図っている。また、各 MPR ノードが配信する接続情報は、そのノードを MPR として選んだノードとの

間の接続情報のみの断片的なものである。

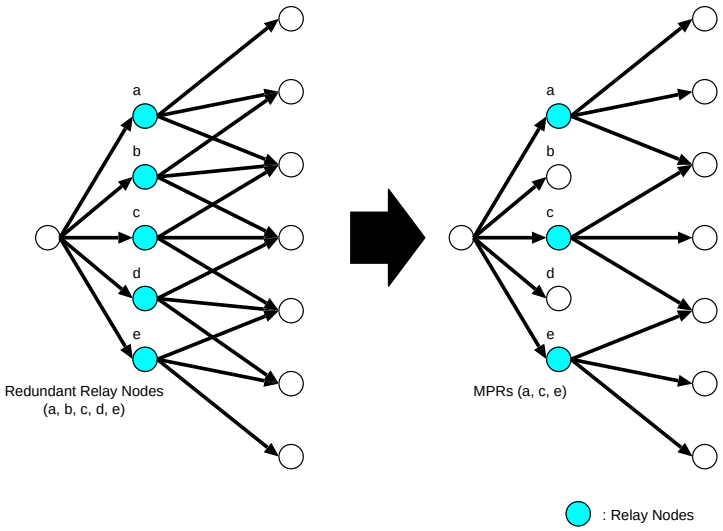

図 6　MPR 集合を利用したフラッディング

OLSR は、MPR の手法が上手く機能する大規模・高密度のネットワークに適する。事前に経路を構築しておくためルーティングオーバーヘッドは小さいが、密度が低かったり、トポロジが短時間に大幅に変化するネットワークには適さない。

2.4　マルチルート通信

マルチホップネットワークでは、取りうるパケットの中継経路は 1 つのみではなく、複数の経路を確立することが可能である。そこで、不安定な通信を、複数の経路を利用して安定化することが考えられる。複数の経路を構築することにより、それらの経路を

- 代替経路としての利用
- 負荷分散
- 帯域の集約によるスループット向上
- 冗長性の向上

のために利用することが可能となる (図 7)。

2.4.1　マルチルートルーティング

複数の経路を構築するためのルーティングプロトコルとしては、これまでに様々なものが提案されている。主なプロトコルを下記に挙げる。

■ソースルーティング

1. MultipathDSR [14]
2. SMR (Split Multipath Routing) [11]

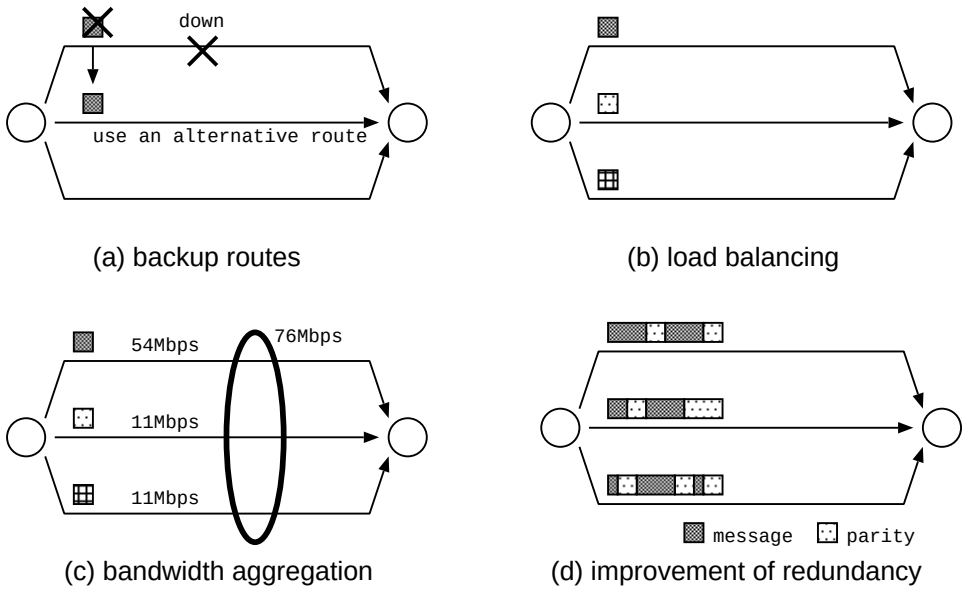

図 7　マルチルート通信

3. MSR (Multipath Source Routing) [17]

上記のプロトコルでは、DSR と同様に送信元で中継経路をすべて把握している。これにより、相互に独立した disjoint な経路の構築が容易になるが、経路を記録するためのパケットヘッダのサイズが増大するという欠点がある。

■距離ベクトル型ルーティング

1. AOMDV (Ad hoc On-demamand Multipath Distance Vector) [12]
2. MNH (Multiple Next Hops) [8]
3. MRAODV (Multiple-Route AODV) [6]
4. AODV-BR (AODV with Backup Routes) [10]

上記のプロトコルはすべて AODV を元に拡張がなされている。

以上のプロトコルではすべて、破棄していた RREQ の再利用という手法を用いている。この方法は、現在提案されているリアクティブ型プロトコルでのマルチパス構築の基本的手法であると言える。また、MANET におけるルーチングの課題である頻繁な経路消失への対応 (再構築に伴うフラッディングの削減を含む) として、現在の複数経路型プロトコルの主なコンセプトである代替経路の確保という手法は有効であると考えられる。

また、複数経路の構築において、disjoint とするかどうかという視点があることが分かる。disjoint である場合、各経路の切断の発生は独立に発生するため、同時に複数経路が切断される可能性は低くなるが、disjoint な場合の代替経路はそうでない場合の経路よりも長くなる傾向になる。また、複数の経路に負荷を分散させることを考えた場合、disjoint でない経路では、リンクまたはノードが共通する部分に負荷が集中することが考

えられる。

2.4.2 ルートダイバーシティ

これまで、上位レイヤの視点から予備経路・負荷分散を目的に MANET におけるマルチルート通信が考えられてきたが、下位レイヤでの重要な技術である誤り訂正符号を用いた手法も提案されれている [19][20]。

複数の経路が利用可能である場合に、各経路を同時に利用してパケットを送信することが考えられる。このとき、いくつかの経路リンクの性能が悪くとも、通信性能を維持できる。この場合、パケット数が増えるけれども、様々な効果によって厳しい環境下で信頼性のある通信を達成ことが可能となる。これを、「ルートダイバーシティ」という。

ルートダイバーシティ効果を強化するため方法として、前方誤り訂正符号 (FEC) を用いることが可能である。FEC にはターボ符号や LDPC(Low Density Parity Check, 低密度パリティ検査) 符号などがある。

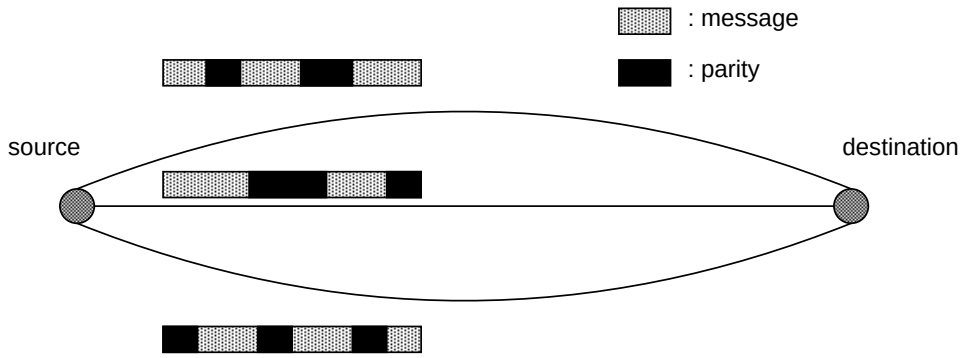

図 8　ルートダイバーシティの例 (FEC を用いた情報の分散)

3　マルチルートルーティングプロトコルの開発

本章では、MANET においてマルチルート通信を行うために必要な、複数経路に対応したルーティングプロトコルの開発を行う。まず、3.1 で、今回開発するプロトコルの基礎になる AODV(Ad-hoc On-demand Distance Vector)[15] の経路構築動作について再度解説し、その動作における複数経路構築の可能性について述べる。次に、3.2 で、その可能性を生かして複数経路の構築を実現するために必要なプロトコルの修正内容について整理する。最後に、3.3 で、修正したプロトコルの評価のために必要な、ネットワークシミュレータ ns-2 上での実装作業の詳細について述べる。

3.1　AODV と複数経路ルーティング

本研究では、AODV を拡張し、複数の経路を構築できるプロトコルの開発を行う。2.4.1 で述べたように、AODV のような、RREQ パケットのフラッディング (一斉配信、図 9) と RREP パケットの返信 (図 10) によりルートの探索と構築を行うプロトコルでは、原理的に複数経路を構築できる可能性を有しているが、利用されていない (図 11)。

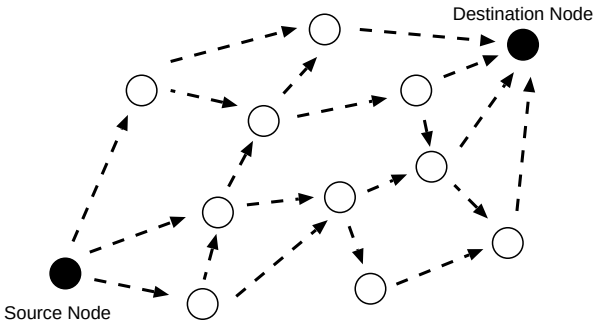

図 9　RREQ のフラッディング

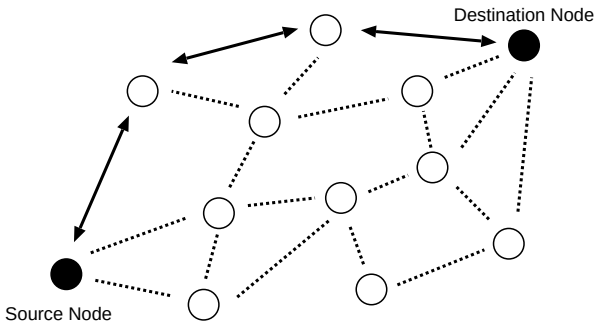

図 10　RREP の返信

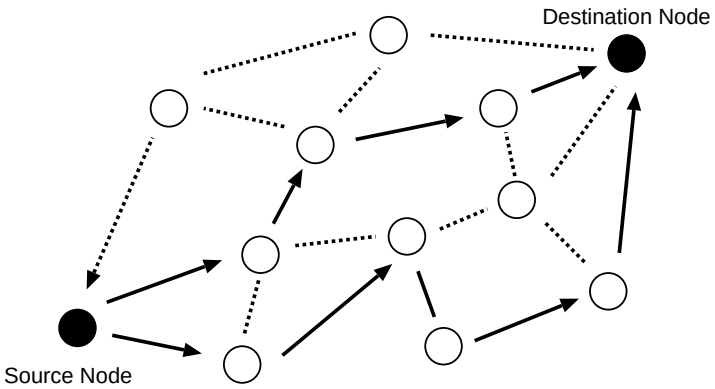

図 11　図 10 で無視されている経路の例

単一経路の場合の経路構築動作は次のような手順により行われる。

1. 送信ノードで送信する必要のあるパケットが生じる。
2. 送信ノードは、パケット送信先までの経路が未知である場合、RREQ パケットを隣接ノードすべてにブロードキャストする。
3. RREQ を受け取った中継ノードは、既に同じ RREQ を受け取っていないか確認する。
   (a) 既に受け取っている場合は破棄し、再ブロードキャストしない。
   (b) まだ受け取っていない場合は、ルーティングテーブルに中継ノード → 送信ノードのへ経路を記録した上でさらにブロードキャストする。
4. ブロードキャストの繰り返しにより、RREQ はやがて受信ノードに到達する。
5. RREQ を受け取った受信ノードは、既に同じ RREQ を受け取っていないか確認する。
   (a) 既に受け取っている場合は破棄し、何もしない。
   (b) まだ受け取っていない場合は、ルーティングテーブルに送信ノードへの経路を記録した上で、自分宛に RREQ を中継してきたノードに向かって RREP パケットを送信する。
6. RREP を受け取った中継ノードは、RREQ 受信時にルーティングテーブルに記録した情報に従って送信ノードに向けて RREP を転送する。
7. RREP が送信ノードに到達し、送信ノードは受信ノードまでの経路の構築完了をを確認する。
8. 送信ノードはデータ送信を開始する。

上記の単一経路での経路構築手順に対して、複数経路を構築するための修正を行う方法としては、2 番目以降の RREQ・RREP をルート構築に利用する方法が考えられる。具体的には、以下の修正を行えば、AODV と互換性のある形で複数経路の構築が可能である。

- 上記手順の 5 番における、受信ノードでの重複 RREQ の破棄をやめる。
- 送信ノードにおいても 2 番目以降の RREP を処理できるようにする。
- 上記 2 項目の修正による情報を格納するために、ルーティングテーブルに経路を複数保持できるように改める。
- データパケットの送信時に複数経路を適切に選択・あるいは同時に利用するようにする。

3.2 マルチルートルーティングプロトコルの仕様

3.1 を元に、本研究で行う AODV プロトコルに対する修正を整理する。修正のポイントは下記の通りである。

1. 送信ノード・受信ノードにおいて 2 個目以降の RREQ・RREP を受信し、その情報を利用するようにする (3.2.1)
2. 中継ノードは通常の AODV と同様の動作をさせる (3.2.1)
3. ルーティングテーブルに項目を追加し、送信ノード・受信ノードにおいて追加項目を利用して送受信を行う (3.2.2, 3.2.3)

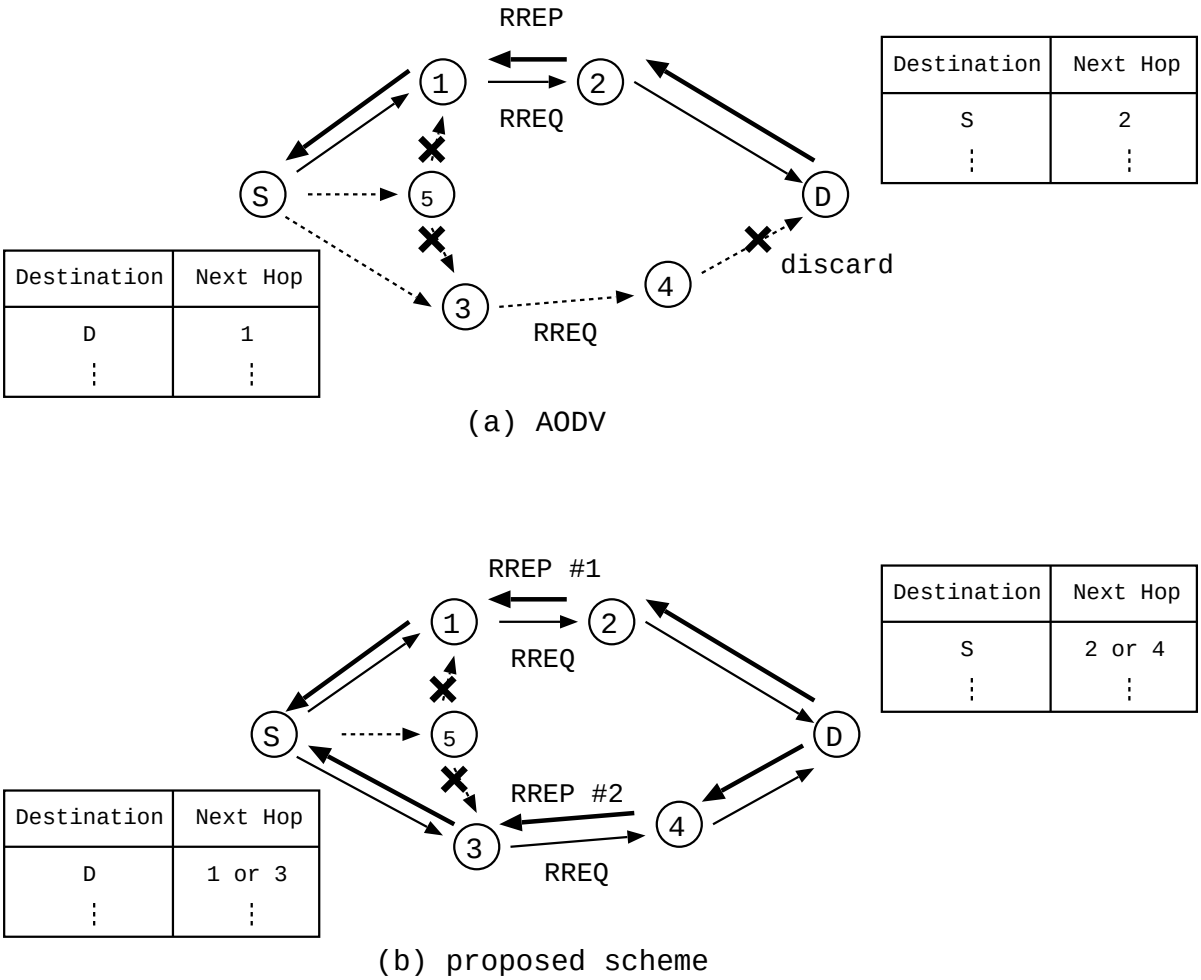

図 12　マルチルート実現のための修正

3.2.1　**ルーティングパケットの取扱い**

各ノードにおける RREQ、RREP 等のパケットの処理は、パケット中継時には通常の AODV と同様であり、パケット送受信時の処理についてのみ修正を行う必要がある。

RREQ については、重複して受信したものは破棄するという条件を受信ノードについてのみ改め、

- 自らが RREQ の宛先である
- 複数経路用ルーティングテーブルに空きがある

場合には重複であっても受信し、複数経路用ルーティングテーブルのエントリに追加することとする。

RREP については、通常の AODV でも重複して受信した場合には良い経路であれば受け入れて、現在の経路を上書きすることになっている。今回の修正では、受け入れて上書きするだけでなく、複数経路用ルーティングテーブルに現在の経路も残しつつさらに新たな経路のエントリを作成することとする。

3.2.2 ルーティングテーブルの構成

通常の AODV のルーティングテーブルの 1 エントリに記録される情報は下記の通りである。

- 宛先 IP アドレス
- 宛先シーケンス番号
- 有効宛先シーケンス番号フラグ
- 状態フラグ（有効, 無効, 修復可能, 修復中）
- ネットワークインターフェース
- ホップ数（宛先ノードに到達するまでに必要なホップ数）
- 次ホップ
- Precursor List
- 経路の有効期限

拡張版の AODV では、複数の経路を扱うために下の項目を追加する。但し、これらの項目を利用するのは送信ノード・受信ノードのみであり、中継ノードでは従来どおり上記の項目のみに基づいてパケットを処理する。

- 経路 1 の次ホップ, ホップ数
- 経路 2 の次ホップ, ホップ数
- …
- 経路 n の次ホップ, ホップ数

但し、n は構築する経路の数である。これらの経路のいずれか、あるいはすべてを同時に利用するかを送信時に選択できるようにする。

3.2.3 ルーティングテーブルの利用

各ノードでは、パケット送信時にルーティングテーブルを参照し、送信先に対応する中継ノードの情報を得ることになる。

送信ノードにおいては、利用可能な複数の経路を何らかの方法で選択する必要がある。今回は、ルーティングテーブルが参照された時に経路をラウンドロビン (複数の経路を順繰りに利用する) に返すようにすることとする。また、上位レイヤからの指示により指定経路を利用出来るようなレイヤ間連携の仕組みを合わせて実装することとする。

中継ノードにおいては、拡張した項目は参照せず、通常の AODV と同様の項目を用いてパケットを中継することとする。

3.3 シミュレータ上での実装

3.2 の仕様に基づいたプロトコルの評価を行うため、ネットワークシミュレータ上でプロトコルを実装し、動作や性能を検証する。

3.3.1 ネットワークシミュレータ ns-2

ns-2[2] は、ネットワーク研究向けのイベントシミュレータである。指定した環境下でのネットワークの振る舞いを調べたり、新規に開発したプロトコルの評価に用いることが出来る。各レイヤのプロトコルが多数実装済みであり、世界中の研究者に利用されている。

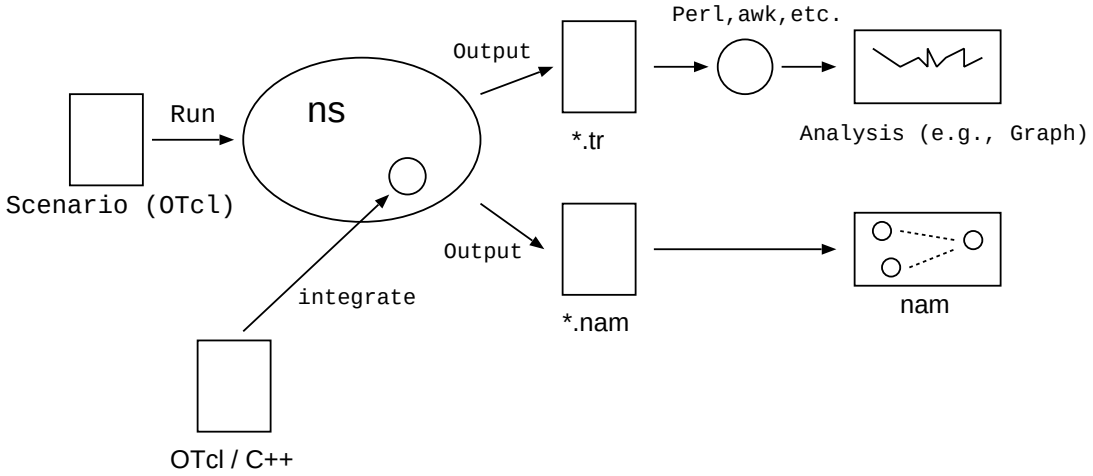

図 13　ns-2 によるシミュレーション

現行バージョンの ns-2 には、CMU Monarch プロジェクト [3] により開発された無線シミュレーション機能が取り込まれており、無線通信分野、特に MANET 向けルーティングプロトコルの評価では多く利用されている。また、ns-2 はオープンソースソフトウェアであり、GPL(GNU General Public License) などのライセンスの元でリリースされている。従って、無償で利用可能であり、改変・再配布も行うことが出来る。従って、実装済のプロトコルを改変したり、独自のプロトコルやアプリケーションを組み込み、その成果を自由に公開することが可能である。

MANET のシミュレーションに用いることのできるその他のネットワークシミュレータとしては、QualNet(旧 GloMoSim)、OPNET などがあるが、いずれも商用アプリケーションである。本研究では、ns-2 を利用して、プロトコルの評価を行う。

3.3.2 実装の詳細

ns-2 には、既に AODV が標準で組み込まれているため、ルーティングプロトコルについてはこのエージェントを拡張して今回の拡張版プロトコルを実装することとする。また、拡張版 AODV と連携して動作し、任意のルートへ任意のデータを割り当てるためのアプリケーションを併せて開発する。

■ルーティングテーブルの修正　3.2 で述べたルーティングテーブルの変更に合わせるように、ルーティングテーブルエントリクラスに複数の経路を記憶するための変数とそれらを扱うメンバ関数を追加する。主要なメンバ変数を表 1 に、該当部分のソースコードをプログラム 1 に示す。関連するソースコードの全文は、A.2.4、A.2.5 に示す。

表 1　ルーティングテーブルエントリクラスの主要メンバ変数一覧

追加	メンバ名	説明
	rt_dst	送信先アドレス
	rt_seqno	送信先シーケンス番号
	rt_hops	ホップ数
	rt_nexthop	次ホップのアドレス
	rt_pclist	precursor リスト
	rt_expire	エントリ有効期限
	rt_flags	ルートの状態 (フラグ)
*	rt_m_hops[ROUTE_COUNT]	マルチルートのホップ数 (ROUTE_COUNT 個の配列)
*	rt_m_nexthop[ROUTE_COUNT]	マルチルートの次ホップアドレス (ROUTE_COUNT 個の配列)
*	rt_routes	マルチルート保持経路数
*	rt_counter	ラウンドロビン切替用 カウンタ

プログラム 1　ルーティングテーブルエントリクラスの主要部分

```
        nsaddr_t        rt_dst;                 // 送信先アドレス
        u_int32_t       rt_seqno;               // シーケンス番号
        /* u_int8_t     rt_interface; */
        u_int16_t       rt_hops;                // ホップ数 (hop count)
        int             rt_last_hop_count;      // last valid hop count
        nsaddr_t        rt_nexthop;             // 次ホップのIPアドレス (next hop IP address)
        /* list of precursors */
        aodv_precursors rt_pclist;
        double          rt_expire;              // エントリ有効期限 (when entry expires)
        u_int8_t        rt_flags;               // ルートの状態(フラグ)

#define RTF_DOWN 0              // ルート無効(DOWN)
#define RTF_UP 1                // ルート有効(UP)
#define RTF_IN_REPAIR 2         // ルート修復中

#ifdef AODV_MULTIROUTE
        /*
         * マルチルート用追加フラグ
         */
#define RTF_WAIT 3              // 複数ルート構築待ち状態(WAIT) データパケット以外に対しては
                                // RTF_UPと同等

        /*
         * マルチルート拡張関連メンバ変数・関数
         */
#define ROUTE_COUNT 2           // 使用するルート数
//#define MULTIROUTE_TIMEOUT 5  // 複数ルート構築を待つ期限
        /* マルチルート用ルーティングテーブル */
        u_int16_t       rt_m_hops[ROUTE_COUNT];         // ホップ数 (hop count)
        nsaddr_t        rt_m_nexthop[ROUTE_COUNT];      // 次ホップのIPアドレス
                                                        // (next hop IP address)
        /* ルート切替のための変数 */
        int             rt_routes;                      // 保持しているルートの数
        int             rt_counter;                     // ルート選択カウンタ
        double          rt_m_create;                    // エントリが生成された時間
        /* マルチルート用ルーティングテーブルの操作を行うメンバ関数 */
        bool            rt_m_add(nsaddr_t nexthop, u_int16_t hops);     // ルート追加
        bool            rt_m_delete(nsaddr_t nexthop);  // ルート削除
        void            rt_m_rotate(void);              // ルート切替
        int             rt_m_lookup(nsaddr_t nexthop);  // nexthopからルートをlookup
        void            rt_m_dump();                    // デバッグ用
#endif // AODV_MULTIROUTE
```

■パケット処理関数の修正　ルーティングパケット (RREQ、RREP)・データパケットを処理する関数を、仕様変更に合わせて修正する。関連するコードの全文は A.2.2、A.2.3 を参照。

RREQ **受信時** (recvRequest)　受信ノードでは重複したものも受信するように修正し、マルチルート用ルーティングテーブルの経路数が指定個数になるまで受け付けるようにする。

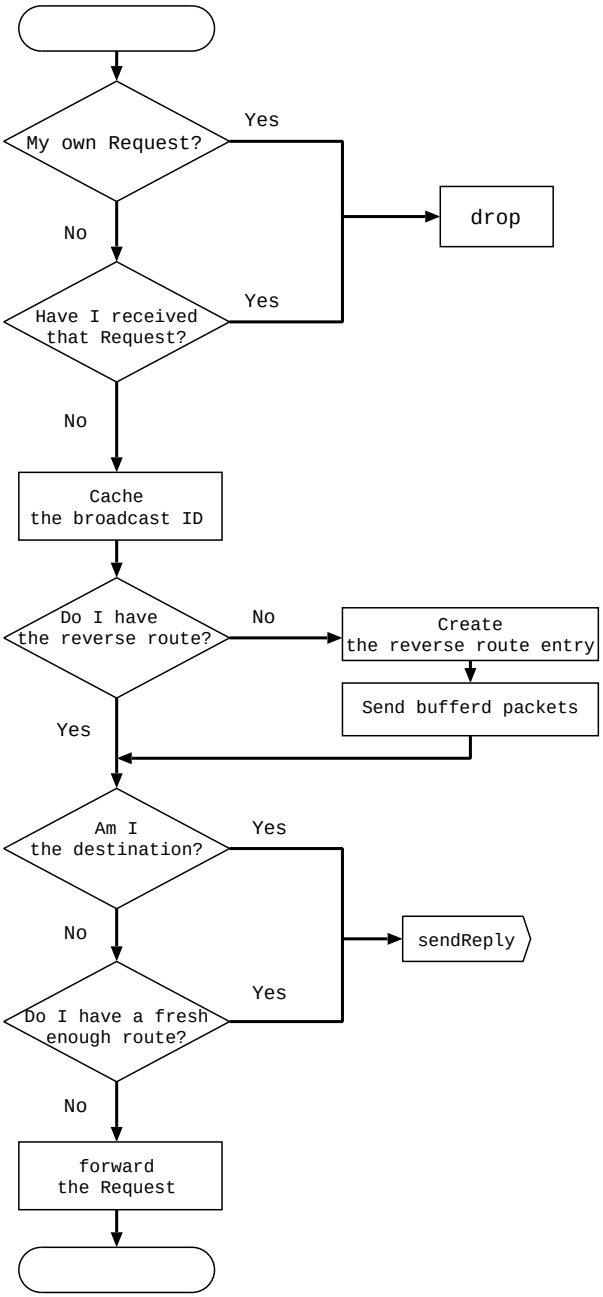

図 14　recvRequest (通常)

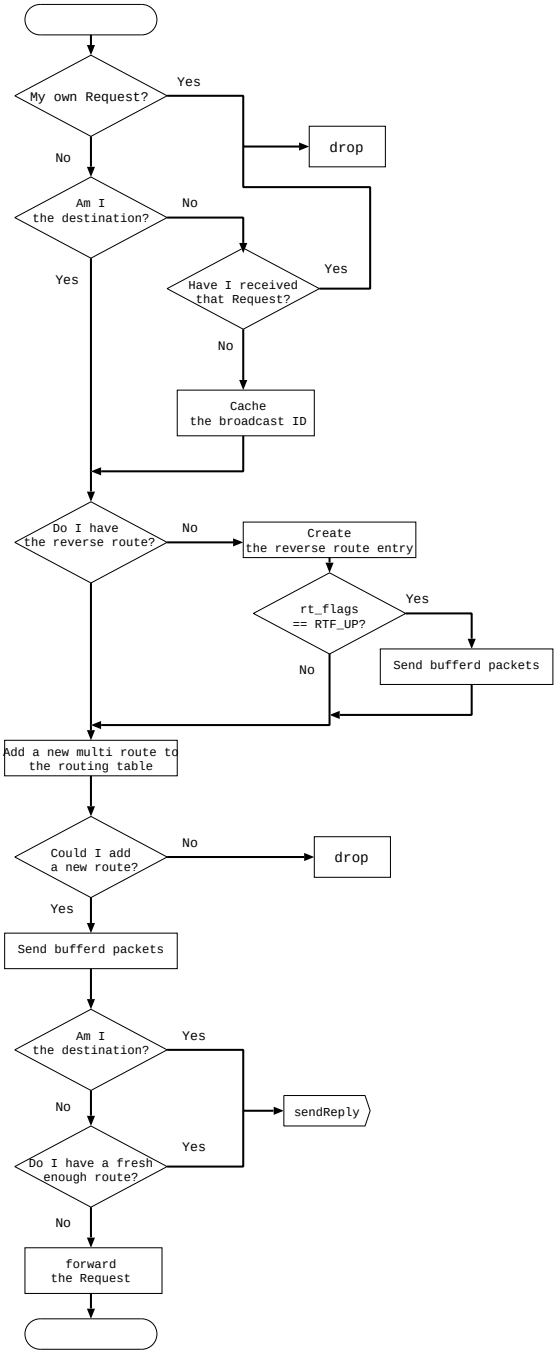

図 15　recvRequest (マルチルート修正)

RREP **送信時** (sendReply)　自分が受信ノードであるならば、RREP の返信先を直近に追加されたルートにする。これにより RREQ を送ってきた方向に向けて RREP を返すことが可能になる。それ以外では通常のルーティングテーブルを参照し、通常どおりのルートに返信する。

RREP **受信時** (recvReply)　自分が送信した RREQ への最初の RREP ならば、通常のルーティングテーブルのエントリを作成すると同時に、マルチルート用ルーティングテーブルにも経路を追加するようにする。2 番目以降の RREP の場合は、マルチルート用ルーティングテーブルに空きがあれば受け付け、そちらのルーティングテーブルに追加するようにする。

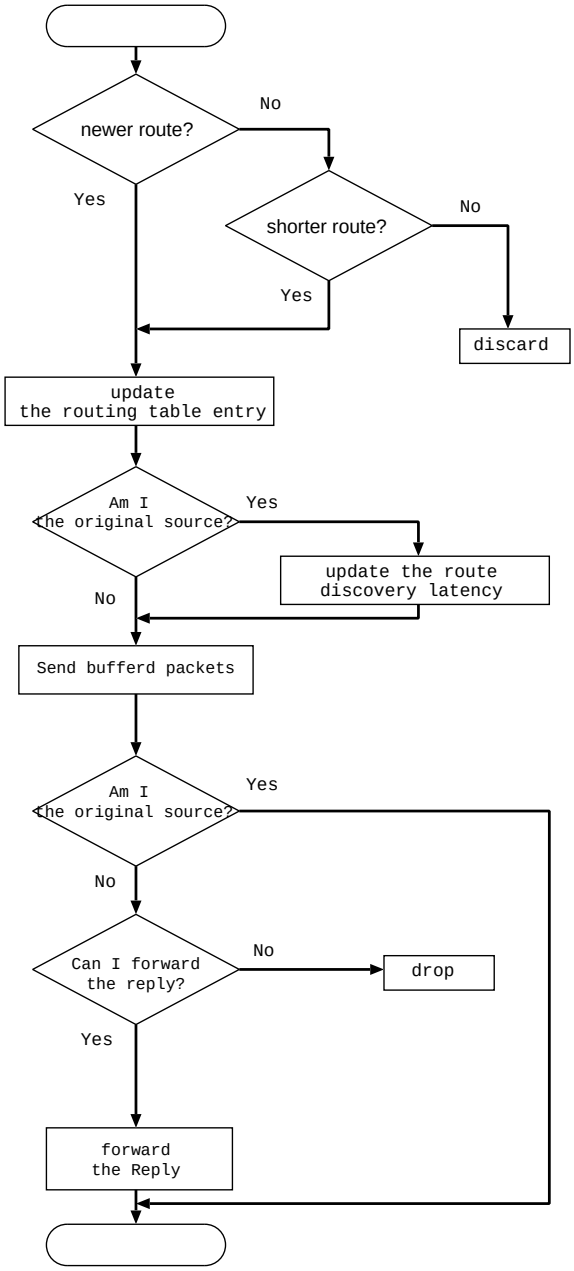

図 16　recvReply (通常)

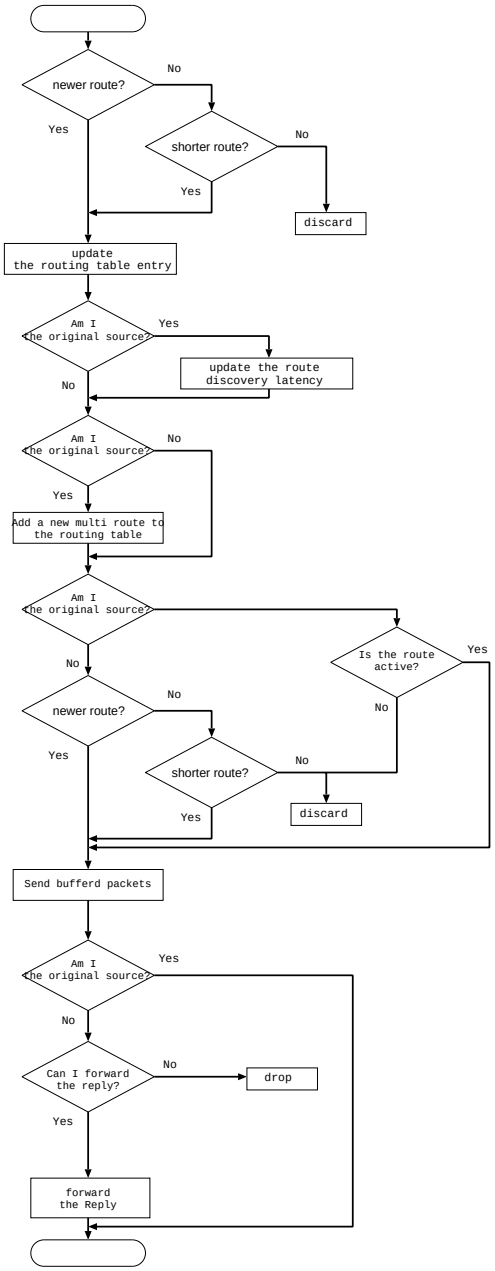

図 17　recvReply (マルチルート修正)

リンク切断時 (handle_link_failure)　通常のルーティングテーブルエントリ削除処理に加えて、マルチルートのルーティングテーブルからも切断されたリンクを含む経路を削除するようにする。

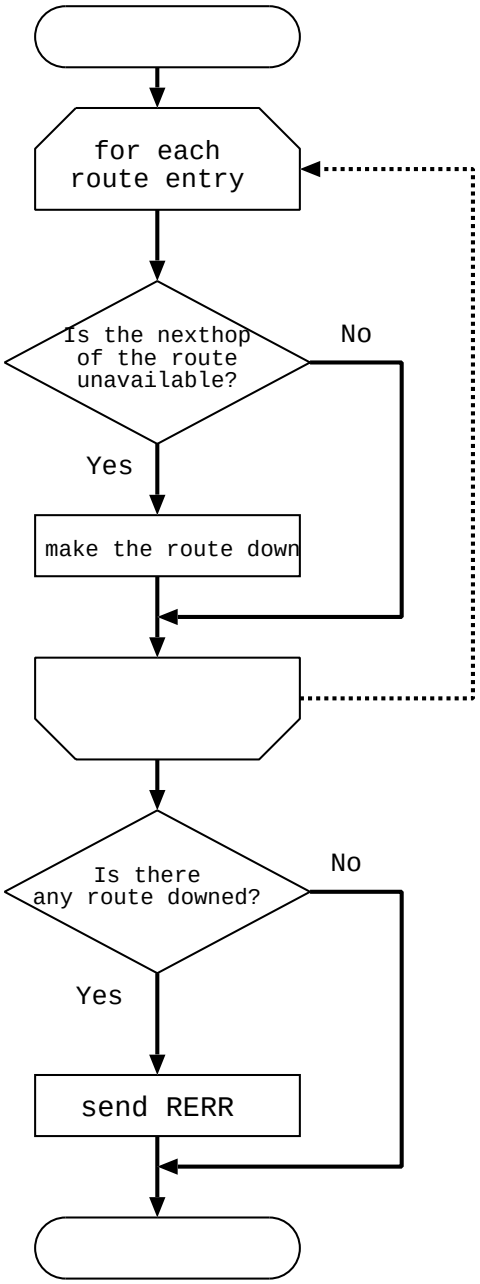

図 18　handle_link_failure (通常)

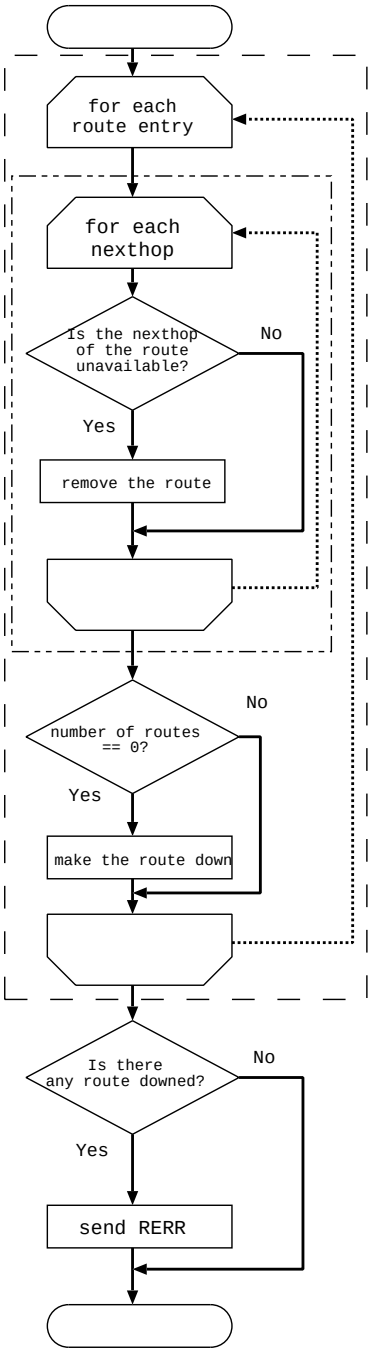

図 19　handle_link_failure (マルチルート修正)

パケット転送 (forward)　送信ノードでは、上位レイヤでパケットに記録した経路に対して転送するようにする。UserData が組み込まれていない場合には、ルーティングテーブルにあるカウンタに従い、保持しているルートを順繰りに使用してパケットを送信する。

プログラム 2　パケット転送関数のルート指定関連部分

```
#ifndef AODV_MULTIROUTE
    ch->next_hop_ = rt->rt_nexthop;
#else // AODV_MULTIROUTE
    if (ch->ptype() != PT_AODV && ih->saddr() == here_.addr_) {
      // 自分が送信したパケットの場合
      // XXX: データパケット送信先の変更はここ

      assert(rt->rt_flags == RTF_UP);
      assert(rt->rt_routes != 0);

/*
 * Application/UserDataが組み込まれているとき
 */
#ifdef ns_userdata_h

        // UserDataで割り当てた経路をパケットから取り出す
        int userpath = ((UserData*) p->userdata())->path();
        //fprintf(stderr, "MULTIROUTE: %s: userdata->path() = %d\n", __FUNCTION__, userpath);
        // 現在の経路数を元に経路割当を決め直す
        if (userpath >= rt->rt_routes) {
                // 割り当てた経路 Noが保持経路数より多い場合は、カウンタの値を使う
                userpath = rt->rt_counter;
        }

#ifdef DEBUG
      fprintf(stderr, "MULTIROUTE: %s: %d sent data packet to %d via %d(#%d) at %f\n",
                __FUNCTION__, index, rt->rt_dst,
                rt->rt_m_nexthop[userpath], userpath,
                CURRENT_TIME);
        //rt->rt_m_dump();
#endif // DEBUG
        ch->next_hop_ = rt->rt_m_nexthop[userpath];

/*
 * Application/UserDataが組み込まれていないとき
 */
#else // ns_userdata_h

#ifdef DEBUG
        fprintf(stderr, "MULTIROUTE: %s: %d sent data packet to %d via %d(#%d) at %f\n",
                __FUNCTION__, index, rt->rt_dst,
                rt->rt_m_nexthop[rt->rt_counter], rt->rt_counter,
                CURRENT_TIME);
        rt->rt_m_dump();
#endif // DEBUG
        ch->next_hop_ = rt->rt_m_nexthop[rt->rt_counter];

#endif // ns_userdata_h

        rt->rt_m_rotate();
 } else {
```

```
        // その他のパケット：通常どおり
#ifdef DEBUG
      if (ch->ptype() != PT_AODV) {
         fprintf(stderr, "MULTIROUTE: %s: %d forwards data packet to %d via %d at %f\n",
                 __FUNCTION__, index, rt->rt_dst,
                 rt->rt_nexthop,
                 CURRENT_TIME);
      }
#endif // DEBUG
      ch->next_hop_ = rt->rt_nexthop;
   }
#endif // AODV_MULTIROUTE
```

■データ転送アプリケーションの開発　上記の拡張版 AODV と連携して動作し、任意のデータを任意の経路に送信するためのアプリケーションを開発する。これにより、複数経路の利用方法による通信性能の違いを調べることが可能になるとともに、将来的に行う予定である FEC を考慮したシミュレーションに用いることが可能になる。

ns-2 の下位レイヤでは、パケットをサイズで扱っている。しかし、上位レイヤからのデータをパケットオブジェクトに格納したままで、下位レイヤからのサイズ基準での処理を行わせるための仕組みは存在している。従って、任意のデータを送受信するためには、用意したファイルを読み取り、内部のパケット形式に格納し、下位レイヤの処理を行わせ、転送され受信ノードのアプリケーション層に渡された時点で格納したデータを読み取り、ファイルに書き出す処理を行うという仕組みを実装することになる。

しかし、アプリケーション層で単純に送信処理を実装すると、経路構築完了前にパケット送信処理が大量に行われ、リンク層のキューが溢れてしまう問題が生じてしまう。これは、リアクティブ型のルーティングプロトコルでは、送信するデータが生じてからルート構築を行うためであり、アプリケーション層側はその間待機させなければならないと考えられる。そこで、データの送信を開始しようとした時点でルーティングエージェントに経路構築を要求し構築完了まで待機するようにし、構築が完了した時点でアプリケーション層に通知し、送信を開始させることにする。この動作の流れを図 20 に示す。

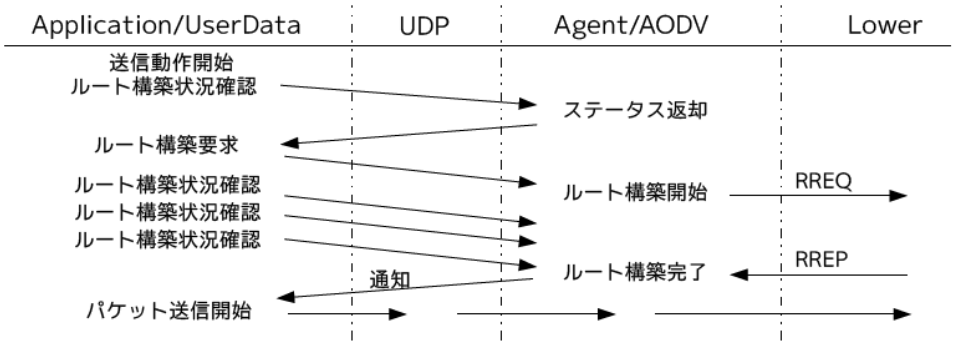

図 20　拡張版 AODV と UserData の連携

また、上記の問題の解決に合わせて、同じレイヤ間連携の仕組みを利用してパケットの転送経路をアプリケーション層側から指示できるようにする。送信時にアプリケーション層側からルーティングテーブルを参照できるようにして、その情報を元に経路を決定しパケットに選択した経路の情報を追加するようにする。その後、ルーティングエージェントのパケット転送関数でその情報を参照しその経路に該当する次ホップにパケッ

トを送信するようにすることにより、任意の経路へのパケット送信が可能となる。
　各経路への割り当て方法については、下記の方式を実装し、シミュレーションシナリオでどの方式を使用するか指定できるようにする。

round-robin　各経路を順繰りに使用
uniform　各経路を等確率で選択して使用
hop-weighted　少ないホップ数の経路に高い確率でパケットが割り当てられるようにして使用
clone　各経路に同一パケットをコピーして送信
short-only　経路の中で最も短いものだけを選択して送信

4　マルチルートルーティングプロトコルの特性評価

本章では、3 章で ns-2 に対して実装したマルチルートのルーティングプロトコルを用いて、次の 2 つの場合についてシミュレーションを行う。

1 つ目の 4.1 節では、頻繁に変化する環境下におけるマルチルート修正版プロトコルの振る舞いを確認するため、ノードが移動しネットワークトポロジが変化する中で通信する場合について、スループットを比較するシミュレーションを行う。

2 つ目の 4.2 節では、マルチルート通信の性能を確認するため、移動しないノードをランダムに配置して、伝送方法を変化させてシミュレーションを行い、伝送方法によるスループットとパケット到達率の違いを比較する。

4.1　ノードが移動する場合

4.1.1　シミュレーションシナリオ

4 つのノードが図 21(a) から (b) の状態に等速移動しながら通信する場合について、スループットの変化を調べる。

図 21　ノード移動シミュレーション

パケットの送信元はノード 0、送信先はノード 1 とする。通信内容は TCP(Transmission Control Protocol) をトランスポート層のプロトコルとして利用する FTP(File Transfer Protocol) トラフィック、または UDP(User Datagram Protocol) を利用する 800kbps の CBR(Constant Bit Rate) トラフィックの 2 種類とする。シミュレーション時間は 150 秒。通信パラメータは表 2 の通りとする。

表 2　通信パラメータ

無線伝搬モデル	TwoRayGround または Shadowing
MAC 層の種類	IEEE 802.11
IFQ の長さ	50 パケット
ルーティングプロトコル	AODV または マルチルート修正版 AODV
マルチルートでの経路数	2
マルチルート利用方法	ラウンドロビン

ここで、表中の IFQ は InterFace Queue の略で、データリンク層のキューを指す。

なお、このシミュレーションでは前章で説明した UserData アプリケーションは用いず、ns-2 に標準で組み込まれている FTP・CBR トラフィック生成エージェントを利用した。FTP は双方向の通信であり、再送や輻輳[*4]制御機能を有した TCP をトランスポート層のプロトコルとして使用している。一方、CBR では、煩雑な制御を行わず送信するだけの UDP を利用している。従って、パケットの到達が保証できないかわりに、送信するパケットが生じたら生じた順に随時送信することがことができる。実ネットワークにおいては音声や映像のストリーミングなどに利用されている。

4.1.2　スループットの変化

スループットの時変動をグラフにしたものを図 22 から図 25 に示す。

4.1.3　考察

図 22 から 24 までのグラフから、単一経路の場合に比べて複数経路の場合は通信開始が遅くなっているということが読み取れる。これは、2 つの経路を確立するためには、2 番目の RREP を待つ必要があり時間がかかるためであると考えられる。しかし、2 番目に構築される経路が最初に構築される経路よりもホップ数が少なく条件が良いため、経路構築に時間はかかっているが一旦複数経路で通信を開始した後はスループットが向上している。

Shadowing 環境下における UDP 通信 (図 25) では他のグラフと違い、通信開始が遅れていないが、単一経路の場合よりもスループットが低下している。これは、状態の悪い通信路を選んで経路を構築しているが、その経路の状況が悪化しているにもかかわらず使い続けているためであると考えられる。TCP(図 24) の場合のグラフからは、当初 UDP と同じ経路を構築したが、状況の悪化に伴い一度切断し、しばらく経路再構築動作をした後、よりよい経路で通信を再開したことが読み取れる。図 25 の場合、当初構築された経路の 2 つのルートが非常に近く、CSMA/CA による衝突回避が働いてしまうためにマルチルートの方がスループットが低下してしまっていると考えられる。TCP を用いている場合には、このような状況では輻輳とみなされ TCP 側で一旦コネクションを切断するため、よい条件で経路再構築を行うチャンスが生まれたと考えられる。

[*4] ふくそう。通信の混雑。

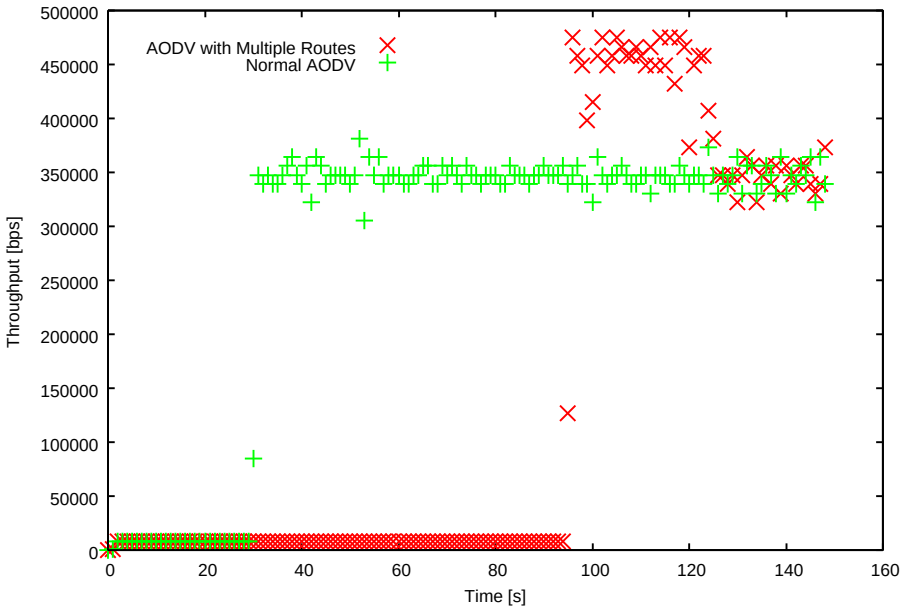

図 22　ノード移動シナリオでのスループット (FTP, TwoRayGround)

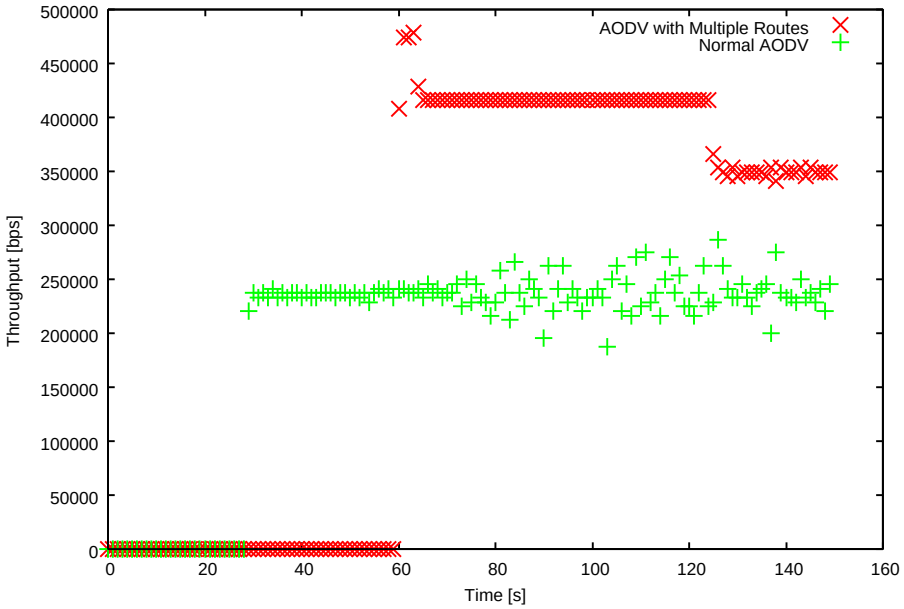

図 23　ノード移動シナリオでのスループット (CBR, TwoRayGround)

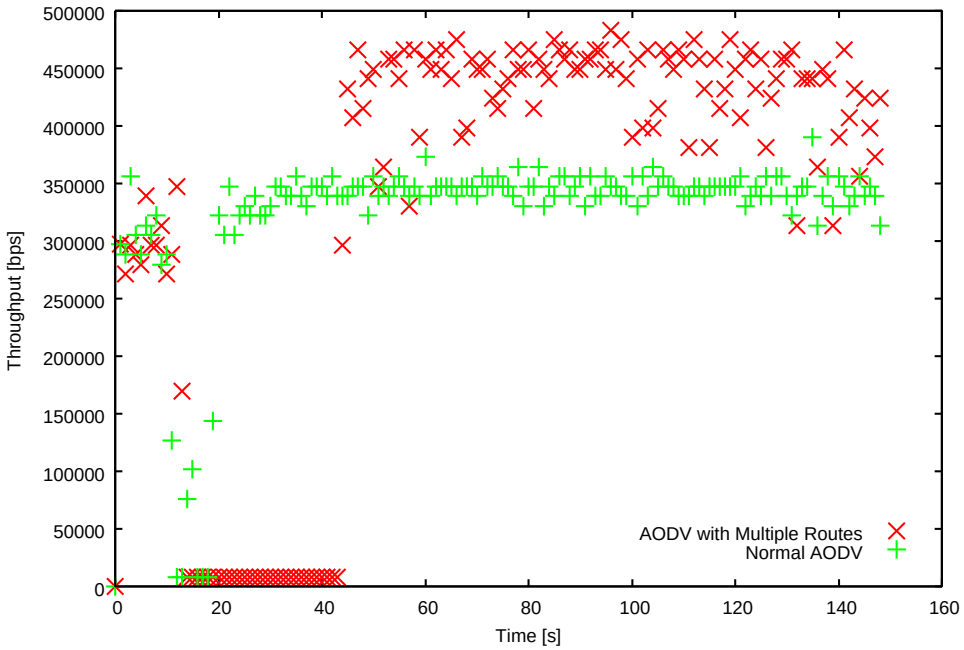

図 24　ノード移動シナリオでのスループット (FTP, Shadowing)

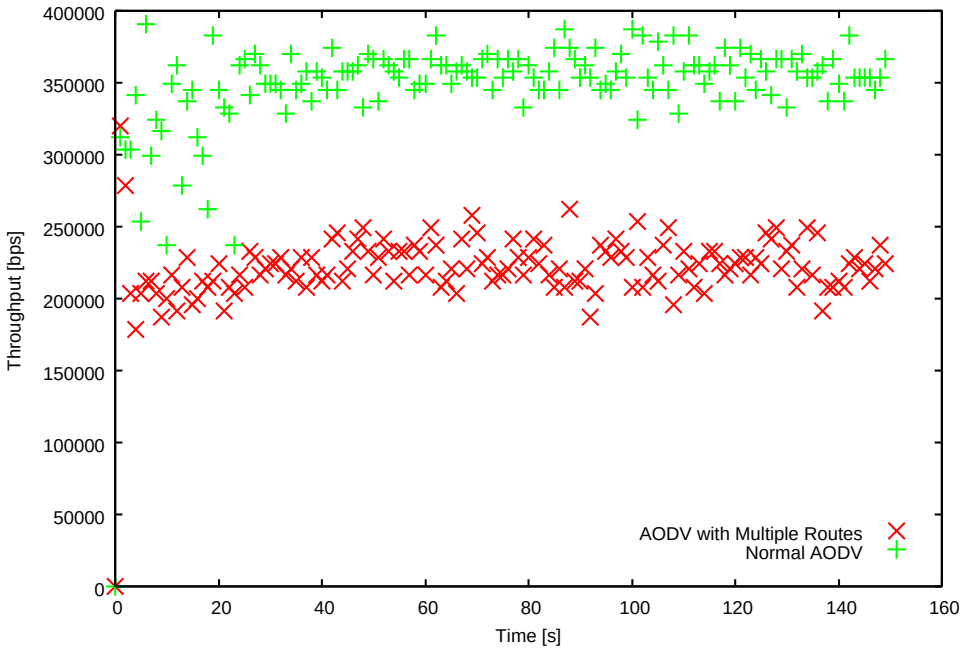

図 25　ノード移動シナリオでのスループット (CBR, Shadowing)

4.2 ランダム配置

4.2.1 シミュレーション諸元

ノードを図 26 のように 40 ノードを 1000x1000 の領域に対してランダムに配置し、ノード 0 とノード 1 の間で UDP 通信を行い、パケット配信率とスループットを調べる。

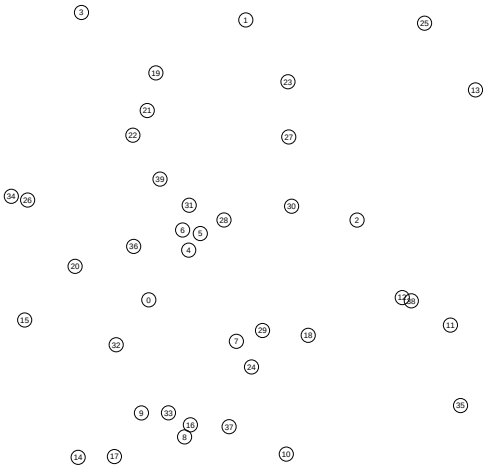

図 26　ランダム配置

送信ビットレートを変化させてパケット到達率を算出し、スループットの時変動も調べる。また、UserData アプリケーションを用い、複数経路の利用方法として下記の 5 通りを試す。

- ラウンドロビン (round-robin)
- 一様分布乱数 (uniform)
- ホップ数の少ないものを優先 (hop-weighted)
- 各経路に複製して送信 (clone)
- 最短経路のみ利用 (short-only)

シミュレーション時間は上限 2000 秒で、3MB の画像ファイルを送信し、送信が終了し次第終了する。通信パラメータは表 3 の通り。

表 3　通信パラメータ

無線伝搬モデル	Shadowing
MAC 層の種類	IEEE 802.11
IFQ の長さ	50 パケット
ルーティングプロトコル	AODV または マルチルート修正版 AODV
パケットサイズ (ヘッダ除く)	500 kbytes

4.2.2 パケット配信率

送信ビットレートを変化させたときのパケット配信率のグラフを図 27 に示す。

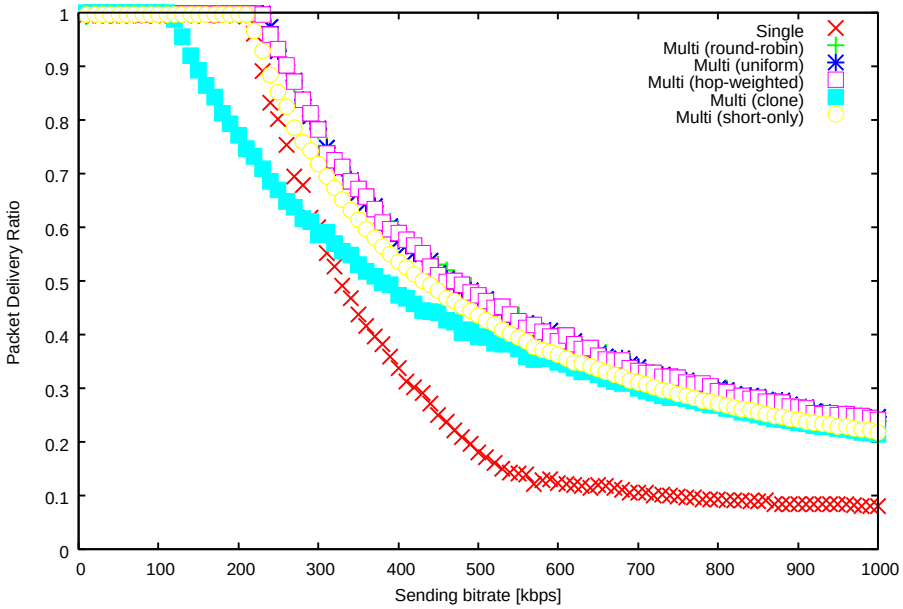

図 27　ランダム配置 パケット配信率

また、Shadowing の深さ (標準偏差 std_db_) を、初期値 4.0 から 10.0 に変更したときのパケット配信率のグラフを図 28 に示す。

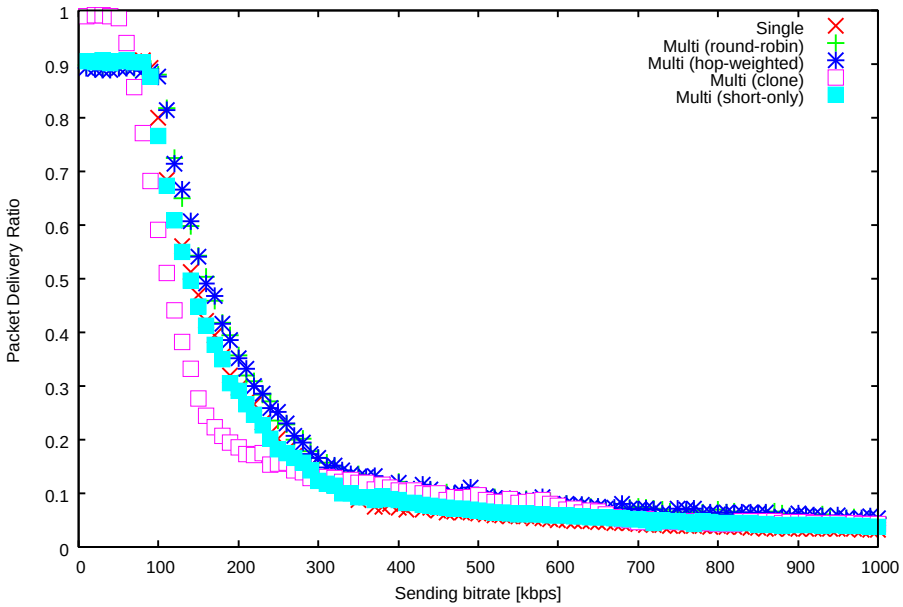

図 28　ランダム配置 パケット配信率 (std_db_ = 10.0)

4.2.3 スループット

送信ビットレート 200kbps のときのスループットの時変動グラフを図 29 に、400kbps のときのグラフを図 30 に示す。

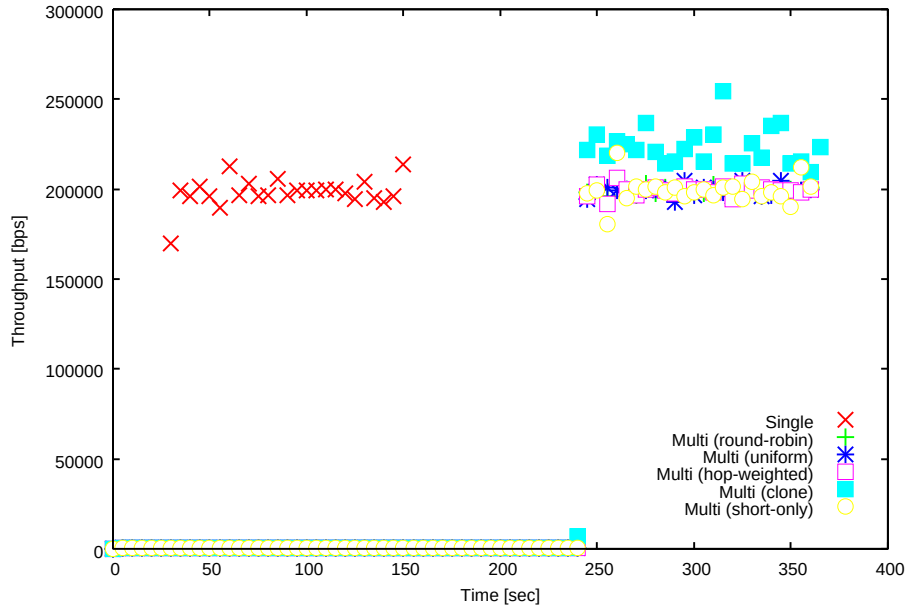

図 29　ランダム配置 スループット (送信ビットレート 200kbps)

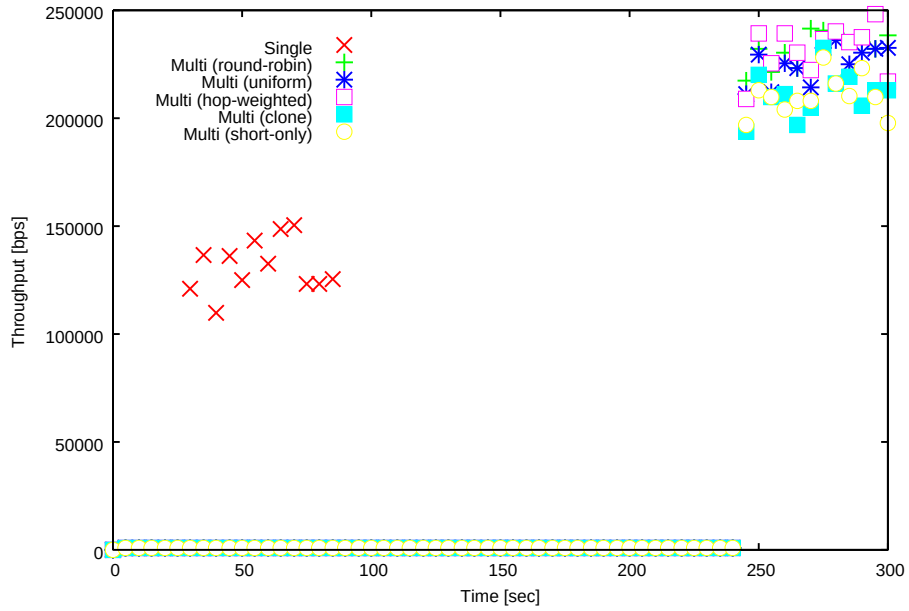

図 30　ランダム配置 スループット (送信ビットレート 400kbps)

4.2.4 考察

図 27 では、単一経路 (Single および short-only) の場合に比べて、複数経路の場合ではパケットドロップなしで送信できるビットレートが上昇している。このとき、IFQ 溢れによるドロップが減っていることから、複数経路になり中継ノードの IFQ の合計容量が増加したことによる負荷分散効果で送信ビットレートをより上げられていると考えられる。また、複数経路の中では唯一 clone の著しく性能が下がっているが、これは現在の UserData では同一パケットを 2 パケット送信することになってしまっているからであると考えられる。この問題は、無線がブロードキャストであることを利用して、同一パケットを複数のノードに送信するリンク層での工夫 [7] などにより解決できると考えられる。

ここで、clone について、図 28 の Shadowing を深くした場合のシミュレーションにおいて、送信ビットレート 100kbps までの領域において、clone のみパケット配信率がよい領域が存在する。これは、複数経路に同一パケットを送信することにより、Shadowing による変動をダイバーシティにより吸収できているのではないかと考えられる。よって、clone は、上述の問題点を解消出来ればより良い特性を達成できるのではないかと考えられる。

スループットの側面から検討すると、図 29 より、4.1 で見られた経路構築の遅延を無視すれば、clone を除いて、複数経路でも単一経路と同等のスループットであることが分かる。clone はこのとき図 27 よりパケット配信率は低下しているがスループットの側面から見ると他よりよい性能が出ていることが分かる。また、図 30 より、送信スループットを上げていくとマルチルートによるものの方がより高いスループットを出せていることが分かる。これは、short-only でも同様に性能が向上していることから、2 番目に構築されるルートの方が環境が良いためであると考えられる。

5 結論

5.1 まとめ

本研究では、AODV を元にしたマルチルートルーティングプロトコルを開発、ネットワークシミュレータの ns-2 上に実装し、その性能について検証した。

マルチルートルーティングプロトコルの利用により、これまで利用されていなかったより良い経路を活用できることによりスループットが向上した。また、パケットが複数経路に分散されることにより中継ノードのキューに余裕が出来、パケット配信率の向上が見られた。

さらに、同一パケットを複数経路に複製して送信する方法について、今回は著しく性能が低かったが、Shadowing の影響を打ち消すことができることが確認できた。また、性能の低さについては、今回の実装では手を加えなかった、データリンク層以下の低いレイヤで工夫することにより向上することが予想できる。

5.2 今後の課題

5.2.1 物理層・データリンク層との連携

物理層・データリンク層との連携により無線の特性を活用し、複製送信のパフォーマンスを向上させるプロトコルを検討する。無線通信は電波が届く範囲であれば複数ノードが同時受信可能であり、物理層レベルではブロードキャストである。したがって、このブロードキャストな特性を生かすようにデータリンク層やネットワーク層 (ルーティングプロトコル) が連携して動作すれば今回生じた複製送信における問題点が解消出来、性能が向上することが期待できる。

5.2.2 前方誤り訂正符号 (FEC) の適用

誤り訂正符号を適用し、情報の分散の効果を向上させることができるか検証する。複数経路伝送に合わせた前方誤り訂正符号 (FEC) を用い、情報ビットとパリティビットを各経路に適切に分散させることにより、ルートダイバーシティ効果の向上が期待できる。

参考文献

[1] Mobile Ad-hoc Networks Charter. http://www.ietf.org/html.charters/manet-charter.html.

[2] The network simulator - ns-2. http://www.isi.edu/nsnam/ns/.

[3] Rice monarch project extension to ns-2. http://www.monarch.cs.rice.edu/cmu-ns.html.

[4] T. Clausen and P. Jacquet. RFC3626: Optimized Link State Routing Protocol (OLSR), 2003. http://www.ietf.org/rfc/rfc3626.txt.

[5] C. Hedrick. RFC1058: Routing Information Protocol, Jun. 1988. http://www.ietf.org/rfc/rfc1058.txt.

[6] Hiroaki Higaki and Shingo Umeshima. Multiple-route ad hoc on-demand distance vector (MRAODV) routing protocl. In *Proc. of IEEE 18th International Parallel and Distributed Processing Symposium*, p. 237, Apr. 2004.

[7] Shweta Jain and Samir R. Das. Exploiting path diversity in the link layer in wireless ad hoc networks. In *Proceedings of the Sixth IEEE International Symposium on World of Wireless Mobile and Multimedia Networks*, pp. 22–30, 2005.

[8] Mimg-Hong Jiang and Rong-Hong Jan. An efficient multiple paths routing protocol for ad-hoc networks. In *Proc. of IEEE International Conference on Information Networking*, pp. 544–549, 2001.

[9] David B. Johnson, David A. Maltz, and Yih-Chun Hu. The dynamic source routing protocol for mobile ad hoc networks (DSR), 2004. http://www.ietf.org/internet-drafts/draft-ietf-manet-dsr-10.txt.

[10] Sung-Ju Lee and Mario Gerla. AODV-BR: Backup routing in ad hoc networks. In *Proc. of IEEE Wireless Communication and Networksing Conference*, pp. 1311–1316, Sep. 2000.

[11] Sung-Ju Lee and Mario Gerla. Split multipath rouitng with maximaly disjoint paths in ad hoc networks. In *Conference Record of IEEE International Conference on Communications*, Vol. 10, pp. 3201–3205, Jun. 2001.

[12] Mahesh K. Marina and Samir R. Das. On-demand multipath distance vector in ad hoc networks. In *Proc. of IEEE International Conference on Networs Protocols*, pp. 14–23, Nov. 2001.

[13] J. Moy. RFC2328: OSPF Version 2, Apr. 1998. http://www.ietf.org/rfc/rfc2328.txt.

[14] Asis Nasipuri and Samir R. Das. On-demand multipath routing for mobile ad hoc networks. In *Proc. of IEEE Eight International Conference on Computer Communications and Networks*, pp. 64–70, Oct. 1999.

[15] C. Perkins, E. Belding-Royer, and S. Das. RFC3561: Ad hoc On-Demand Distance Vector (AODV) Routing, 2003. http://www.ietf.org/frc/rfc3561.txt.

[16] Y. Rekhter and T. Li. RFC1771: A Border Gateway Protocol 4 (BGP-4), Mar. 1995. http://www.ietf.org/rfc/rfc1771.txt.

[17] Lei Wang, Yantai Shu, Miao Dong, Oliver W.W. Yang, and Lianfang Zhang. Adaptive multipath source routing in ad hoc networks. In *Proc. of IEEE International Conference on Communications*, pp. 867–871, Jun. 2001.

[18] 阪田史郎, 青木秀憲, 間瀬憲一. アドホックネットワークと無線 LAN メッシュネットワーク. 電子情報通

信学会論文誌 B, Vol. J89-B, No. 6, pp. 811–823, Jun. 2006.
[19] 中川信之, 鯉江尚央, 岡田啓, 山里敬也, 片山正昭. マルチホップ無線ネットワークにおける複数経路を利用した誤り訂正手法. 2003 年電子情報通信学会通信ソサエティ大会, p. 496, Sep. 2003.
[20] 御倉悠, 和田忠浩. アドホックネットワークにおける誤り訂正符号を用いたルートダイバーシチ特性に関する一検討. 電子情報通信学会技術研究報告 CS2006-18, pp. 25–30, May 2006.

付録 A　ns-2 マルチルートデータ転送拡張

A.1　ネットワークシミュレータ ns-2

A.1.1　ns-2 の構成

ns-2 は、C++ と OTcl (Object Tcl) の 2 つのプログラミング言語を用いて開発されている。それぞれ、実行速度を要求する部分は事前にコンパイルし実行バイナリを生成しておく C++ で開発、記述性が必要となるシミュレーションシナリオなどの部分にはインタプリタ言語である OTcl を用いる、という役割分担がある。

C++・OTcl は共にオブジェクト指向言語であり、ns-2 内のネットワーク要素やプロトコルはオブジェクトとして表現される。C++ 側のオブジェクトと OTcl 側のオブジェクトは TclCL により関連付けられ、C++ 側・OTcl 側の双方から利用することを可能にしている。

A.1.2　本研究における ns-2 の拡張

本研究では、

1. AODV プロトコルエージェント (Agent/AODV) のマルチルート化修正
2. 新規アプリケーション Application/UserData の作成

を行った。修正の詳細は 3 章を参照。それぞれのソースコードは A.2 に掲載している。

A.1.3　本研究における ns-2 利用の問題点

本研究において障害となった ns-2 の問題点を下記に述べる。

■OTcl を採用していること　OTcl は、Tcl(Tool Command Language) に対してオブジェクト指向に対応するように拡張したものであるが、現在 ns-2 以外では利用されていない。そのため、参考資料が少なく、IDE(統合開発環境) も OTcl をサポートしていない。従って、OTcl を習得することは初学者にとっては容易ではなく、たとえ習得したとしても ns-2 のような大規模プロジェクトの全容を把握するために必要なツール類が入手・利用できない。

ns の次期バージョンである ns-3 では OTcl の代わりに Python を採用して開発が進められており、ns-3 に移行すればこの問題は解消されると考えられる。但し、ns-2 との互換性はないため、本研究で開発したものを含む、ns-2 におけるコード資産を利用するためには相応の移植作業が必要になる。

■ドキュメントが未整備かつメンテナンスされていないこと　ns-2 には、マニュアルは存在してるが、現行バージョンに記述の更新が追いついておらず、コードを読むまで正確な状況を掴むことが出来ない。特に、MANET 向けルーティングのシミュレーションについてはドキュメント化自体行われていない。シナリオを書いてシミュレーションを動かす程度であれば授業ノート等が Web 上に公開されているが、実装済みのプロトコルを改変する場合には困難が伴う。

また、C++ 側、Tcl 側にまたがる膨大なクラス構造を持っているにもかかわらず、それらの全容を把握できる最新のドキュメントが存在していない。IDE やソースコード解析ツールも、OTcl に対応しているものはほとんど存在しない (Tcl 用が一部使える場合もあるが、制限があることが多い)。

■物理層モデルが簡略化されていること　本研究は元々、誤り訂正符号・再送制御などの物理層・データリンク層での手法を、ネットワーク層の技術であるルーティングと連携させるクロスレイヤを指向したものであった (参考：[20])。しかし、このような低レイヤの研究を行う視点から見ると、ns-2 のネットワークモデルは物理層・データリンク層が簡略化されすぎている。ns-2 におけるパケットはパケット長を元に扱われており、実際のデータが ns-2 内のネットワークを直接流れることはない。従って、本研究ではアプリケーション層のパケットのレベルで実装せざるを得なかった。

変調方式・誤り訂正符号化方式による誤り率の違いを設定するモジュールも存在するが、パケット内で何ビット誤って、何ビット以上誤ったらパケットを受信できなかったことにする、というモデルであり、誤り訂正符号のシミュレーションを低いレイヤで実現するためには大規模な改変が必要である。

A.2　マルチルートデータ転送コード

以下に、本研究で作成した ns-2 用マルチルート修正版 AODV ルーティングプロトコル、UserData アプリケーション、および関連スクリプトのソースコードを示す。

A.2.1　README

```
--------------------------------------------------
 ns-2 マルチルートデータ転送拡張 (2009/01/21 版)

  Wada Laboratory, Shizuoka University
  Takahiro Sugimoto <taka_s@keyaki.kokage.cc>

  Last Modified: 2009/02/04 17:56:59
--------------------------------------------------

==================
 プログラム概要
==================

  [マルチルート拡張版 AODV]
    ns-2 に標準で組み込まれている AODV エージェントを改変し、複数の経路を構築できる
  ようにしたものです。

  [UserData]
    ns-2 上の UDP パケットに任意のデータを載せて通信させることができるアプリケーシ
  ョンです。

==================
 ディレクトリ構成
==================

 /
 +- README                      -- このファイル
 +- aodv/                       -- マルチルート拡張版 AODV
 +- userdata/                   -- データ転送アプリケーション (UserData)
 |  +- userdata.{h, cc}         --  UserData メインコード
 |  +- userdata_file.{h, cc}    --  UserData ファイル I/O コード
 +- scripts/                    -- シミュレーション用スクリプト
    +- demo.tcl                 --  テスト用スクリプト
    +- demo.in                  --  テストスクリプト用入力データ (BMP 画像)
```

```
   +- *.pl                      --  ログ分析スクリプト

==================
 開発環境
==================

 Fedora 8 (gcc4, i386-redhat-linux) + ns-2.32

==================
 組み込み方法
==================

 [ UserData + マルチルート拡張 AODV ] (連携 ON)

   1. ns-2 のディレクトリ内にある aodv ディレクトリを削除します。
   2. このアーカイブ内にある aodv ディレクトリ、userdata ディレクトリを ns-2 のディレ
      クトリにコピーします。
   3. ./configure スクリプトを実行し、Makefile を生成します。
   4. 生成しした Makefile の「DEFINE」に
        -DAODV_MULTIROUTE -DAODV_USERDATA_CONNECT
      を、「OBJ_CC」に
        userdata/userdata.o userdata/userdata_file.o \
      を追加します。
   5. make します。

 [ UserData + シングルルート AODV ] (連携 ON)

   1. ns-2 のディレクトリ内にある aodv ディレクトリを削除します。
   2. このアーカイブ内にある aodv ディレクトリ、userdata ディレクトリを ns-2 のディレ
      クトリにコピーします。
   3. ./configure スクリプトを実行し、Makefile を生成します。
   4. 生成しした Makefile の「DEFINE」に
        -DAODV_USERDATA_CONNECT
      を、「OBJ_CC」に
        userdata/userdata.o userdata/userdata_file.o \
      を追加します。
   5. make します。

 [ UserData のみ ]

   1. userdata ディレクトリを ns-2 のディレクトリにコピーします。
   2. ./configure スクリプトを実行し、Makefile を生成します。
   3. 生成しした Makefile の「OBJ_CC」に
        userdata/userdata.o userdata/userdata_file.o \
      を追加します。
   4. make します。

  ※ この場合、送信開始直後にパケットの大量ドロップが発生します。

 [ マルチルート拡張 AODV のみ ]

   1. ns-2 のディレクトリ内にある aodv ディレクトリを削除します。
   2. このアーカイブ内にある aodv ディレクトリを ns-2 のディレクトリにコピーします。
   3. ./configure スクリプトを実行し、Makefile を生成します。
   4. 生成しした Makefile の「DEFINE」に
        -DAODV_MULTIROUTE
      を追加します。
```

5. make します。

[シングルルート AODV のみ]

1. ns-2 のディレクトリ内にある aodv ディレクトリを削除します。
2. このアーカイブ内にある aodv ディレクトリを ns-2 のディレクトリにコピーします。
3. ./configure スクリプトを実行し、Makefile を生成します。
4. make します。

※ 機能的には ns-2 標準の AODV と同じ（はず）ですが、デバッグ出力が一部異なります。

==========
利用方法
==========

・具体例は script/demo.tcl を参考にしてください。

・Application/UserData、UserDataFile、UserDataLog の引数は次の通りです。

[UserData]

set userdata [new Application/UserData]
--- 新たな UserDataApp インスタンス $userdata を生成します。

$userdata attach-agent <agent object>
--- Agent に $userdata をアタッチします。

$userdata attach-file-in <UserDataFile>
--- $userdata に UserDataFile オブジェクト <UserDataFile> を入力先としてアタッチします。

$userdata attach-file-out <UserDataFile>
--- $userdata に UserDataFile オブジェクト <UserDataFile> を出力先としてアタッチします。

$userdata attach-file-sorted <UserDataFile>
--- $userdata に UserDataFile オブジェクト <UserDataFile> を並べ替え済出力先としてアタッチします。

$userdata attach-log <UserDataLog>
--- $userdata に UserDataLog オブジェクト <UserDataLog> をログ出力先としてアタッチします。

$userdata set-interval <interval>
--- 送信間隔を <interval> に設定します。
設定しない場合には、userdata.h で DEFINE した INTERVAL の値になります。

$userdata start
--- 送信を開始/再開します。

$userdata wait
--- 送信を中断します。

$userdata stop
--- 送信を終了します。attach-file-sorted されている場合にはここで並べ替え済みデータの書き出しが行われます。

以下は AODV_MULTIROUTE を有効にしてコンパイルした場合のみ利用可能です。

$userdata attach-ragent <routing agent object>
--- $userdata にルーティングエージェントをアタッチします。

$userdata set-multiroute-scheme {round-robin|uniform|hop-weighted|clone}
--- 複数ルートへの割り当て方式を設定します。
round-robin : 各経路を番号順に繰り替えして使用します。
uniform : 各経路を等確率で選択して使用します。
hop-weighted : 短い経路に高い確率でパケットが割り当てられるようにして使用します。

```
            clone         : 各経路に同一パケットをコピーして送信します。
            short-only    : 経路の中で最も短いものだけを選択して送信します。

    [UserDataFile]
      set userfile [new UserDataFile]
        --- 新たな UserDataFile インスタンス $userfile を生成します。
       $userfile setfile <filename> <r/w>
        --- ファイルと読み書きモードを設定します。
       $userdata set-unitlen <length>
        --- 一回に読むデータの長さ（＝パケットサイズ）を指定します。

    [UserDataLog]
      set userlog [new UserDataLog]
        --- 新たな UserDataLog インスタンス $userfile を生成します。
      $userfile setfile <filename>
        --- ファイルを指定します。

==================
 備考
==================

  ・AODV_USERDATA_CONNECT なしの状態で UserData を使用すると、送信開始直後のパケット
    が大量にドロップする問題が発生します。
  ・AODV のコードは修正 BSD ライセンスの適用を受けます。コード冒頭の著作権表示は削除
    しないようにしてください。
  ・テスト用データ（demo.in）の出典: http://sipi.usc.edu/database/index.html

  ・「warning: no class variable UserDataFile::debug_」という表示が出ますが、動作
    には支障ありません。気になるのであれば、ns-default.tcl に次のように追記してく
    ださい。
       UserDataFile set debug_ false
       UserDataLog set debug_ false
```

A.2.2 aodv/aodv.h

```
/*
Copyright (c) 1997, 1998 Carnegie Mellon University.  All Rights
Reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice,
this list of conditions and the following disclaimer in the documentation
and/or other materials provided with the distribution.
3. The name of the author may not be used to endorse or promote products
derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE AUTHOR ``AS IS'' AND ANY EXPRESS OR
IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES
OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED.
IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS;
OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
```

```
WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR
OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

The AODV code developed by the CMU/MONARCH group was optimized and tuned by Samir Das and Mahesh Marina,
University of Cincinnati. The work was partially done in Sun Microsystems.

*/

/*
 * AODV マルチルート拡張 (UserData 連携 ver.)
 * Last Modified: 2008/12/11 18:44:49
 */

#ifndef __aodv_h__
#define __aodv_h__

//#include <agent.h>
//#include <packet.h>
//#include <sys/types.h>
//#include <cmu/list.h>
//#include <scheduler.h>

#include <cmu-trace.h>
#include <priqueue.h>
#include <aodv/aodv_rtable.h>
#include <aodv/aodv_rqueue.h>
#include <classifier/classifier-port.h>

/*
  Allows local repair of routes
*/
//#define AODV_LOCAL_REPAIR

/*
  Allows AODV to use link-layer (802.11) feedback in determining when
  links are up/down.
*/
#define AODV_LINK_LAYER_DETECTION

/*
  Causes AODV to apply a "smoothing" function to the link layer feedback
  that is generated by 802.11.  In essence, it requires that RT_MAX_ERROR
  errors occurs within a window of RT_MAX_ERROR_TIME before the link
  is considered bad.
*/
#define AODV_USE_LL_METRIC

/*
  Only applies if AODV_USE_LL_METRIC is defined.
  Causes AODV to apply omniscient knowledge to the feedback received
  from 802.11.  This may be flawed, because it does not account for
  congestion.
*/
//#define AODV_USE_GOD_FEEDBACK
```

```
class AODV;

/* ns-2.32 標準値 */
#define MY_ROUTE_TIMEOUT        10                              // 100 seconds
#define ACTIVE_ROUTE_TIMEOUT    10                              // 50 seconds
#define REV_ROUTE_LIFE          6                               // 5  seconds
#define BCAST_ID_SAVE           6                               // 3 seconds
/* */
/* コメント準拠の数値
#define MY_ROUTE_TIMEOUT        100                             // 100 seconds
#define ACTIVE_ROUTE_TIMEOUT    50                              // 50 seconds
#define REV_ROUTE_LIFE          5                               // 5  seconds
#define BCAST_ID_SAVE           3                               // 3 seconds
*/

// No. of times to do network-wide search before timing out for
// MAX_RREQ_TIMEOUT sec.
#define RREQ_RETRIES            3
// timeout after doing network-wide search RREQ_RETRIES times
#define MAX_RREQ_TIMEOUT        10.0 //sec

/* Various constants used for the expanding ring search */
#define TTL_START     5
#define TTL_THRESHOLD 7
#define TTL_INCREMENT 2

// This should be somewhat related to arp timeout
#define NODE_TRAVERSAL_TIME     0.03             // 30 ms
#define LOCAL_REPAIR_WAIT_TIME  0.15 //sec

// Should be set by the user using best guess (conservative)
#define NETWORK_DIAMETER        30             // 30 hops

// Must be larger than the time difference between a node propagates a route
// request and gets the route reply back.

//#define RREP_WAIT_TIME     (3 * NODE_TRAVERSAL_TIME * NETWORK_DIAMETER) // ms
//#define RREP_WAIT_TIME     (2 * REV_ROUTE_LIFE)  // seconds
#define RREP_WAIT_TIME         1.0  // sec

#define ID_NOT_FOUND    0x00
#define ID_FOUND        0x01
//#define INFINITY        0xff

// The followings are used for the forward() function. Controls pacing.
#define DELAY 1.0           // random delay
#define NO_DELAY -1.0       // no delay

// think it should be 30 ms
#define ARP_DELAY 0.01      // fixed delay to keep arp happy


#define HELLO_INTERVAL          1               // 1000 ms
#define ALLOWED_HELLO_LOSS      3               // packets
#define BAD_LINK_LIFETIME       3               // 3000 ms
#define MaxHelloInterval        (1.25 * HELLO_INTERVAL)
#define MinHelloInterval        (0.75 * HELLO_INTERVAL)
```

```
// 和田研独自修正関係
//（ここではなく、コンパイルオプションでの設定を推奨します）
// ** マルチルート関連の機能を有効にする **
//#define AODV_MULTIROUTE
// ** Application/UserData 用のルート構築要求機能を有効にする **
//#define AODV_USERDATA_CONNECT

#ifdef AODV_USERDATA_CONNECT
#include "../userdata/userdata.h"
#endif // AODV_USERDATA_CONNECT

/*
  Timers (Broadcast ID, Hello, Neighbor Cache, Route Cache)
*/
class BroadcastTimer : public Handler {
public:
        BroadcastTimer(AODV* a) : agent(a) {}
        void    handle(Event*);
private:
        AODV    *agent;
        Event   intr;
};

class HelloTimer : public Handler {
public:
        HelloTimer(AODV* a) : agent(a) {}
        void    handle(Event*);
private:
        AODV    *agent;
        Event   intr;
};

class NeighborTimer : public Handler {
public:
        NeighborTimer(AODV* a) : agent(a) {}
        void    handle(Event*);
private:
        AODV    *agent;
        Event   intr;
};

class RouteCacheTimer : public Handler {
public:
        RouteCacheTimer(AODV* a) : agent(a) {}
        void    handle(Event*);
private:
        AODV    *agent;
        Event   intr;
};

class LocalRepairTimer : public Handler {
public:
        LocalRepairTimer(AODV* a) : agent(a) {}
        void    handle(Event*);
private:
        AODV    *agent;
```

```
        Event   intr;
};


/*
  Broadcast ID Cache
*/
class BroadcastID {
        friend class AODV;
 public:
        BroadcastID(nsaddr_t i, u_int32_t b) { src = i; id = b;  }
 protected:
        LIST_ENTRY(BroadcastID) link;
        nsaddr_t        src;
        u_int32_t       id;
        double          expire;         // now + BCAST_ID_SAVE s
};

LIST_HEAD(aodv_bcache, BroadcastID);


/*
  The Routing Agent
*/
class AODV: public Agent {

  /*
   * make some friends first
   */

        friend class aodv_rt_entry;
        friend class BroadcastTimer;
        friend class HelloTimer;
        friend class NeighborTimer;
        friend class RouteCacheTimer;
        friend class LocalRepairTimer;

 public:
        AODV(nsaddr_t id);

        void            recv(Packet *p, Handler *);

        /*
         *  Application/UserData 用インターフェース
         */
#ifdef AODV_MULTIROUTE
        // ルート保持数を返す
        int             rt_count(nsaddr_t dst);
        // 各ルートのホップ数を返す
        int             rt_hops(nsaddr_t dst, int path_id);
        // ルート保持可能数を返す
        int             rt_count_max() { return ROUTE_COUNT; }
#endif // AODV_MULTIROUTE
#ifdef AODV_USERDATA_CONNECT
        // 強制ルート構築
        void            force_rt_request(nsaddr_t dst, UserDataApp *callback);
        int             rt_request_status() { return rt_request_status_; }
```

```
 protected:
        void            userdata_callback(nsaddr_t dst);
        void            userdata_fail_callback(nsaddr_t dst);
 private:
        UserDataApp     *userdata_callback_;
        int             rt_request_status_;
        nsaddr_t        userdata_dst;
#endif // AODV_USERDATA_CONNECT

 protected:
        int             command(int, const char *const *);
        int             initialized() { return 1 && target_; }

        /*
         * Route Table Management
         */
        void            rt_resolve(Packet *p);
        void            rt_update(aodv_rt_entry *rt, u_int32_t seqnum,
                                  u_int16_t metric, nsaddr_t nexthop,
                                  double expire_time);
        void            rt_down(aodv_rt_entry *rt);
#ifdef AODV_MULTIROUTE
        // rt->rt_flags を RTF_UP に変更
        //  元の実装では rt_update を行った時点で RTF_UP となるが、マルチルート版では
        //  一旦 RTF_WAIT になった後、ルート数が揃った段階で rt_up により RTF_UP にする。
        void            rt_up(aodv_rt_entry *rt);
#endif // AODV_MULTIROUTE
        void            local_rt_repair(aodv_rt_entry *rt, Packet *p);
 public:
        void            rt_ll_failed(Packet *p);
        void            handle_link_failure(nsaddr_t id);
 protected:
        void            rt_purge(void);

        void            enque(aodv_rt_entry *rt, Packet *p);
        Packet*         deque(aodv_rt_entry *rt);

        /*
         * Neighbor Management
         */
        void            nb_insert(nsaddr_t id);
        AODV_Neighbor*       nb_lookup(nsaddr_t id);
        void            nb_delete(nsaddr_t id);
        void            nb_purge(void);

        /*
         * Broadcast ID Management
         */

        void            id_insert(nsaddr_t id, u_int32_t bid);
        bool            id_lookup(nsaddr_t id, u_int32_t bid);
        void            id_purge(void);

        /*
         * Packet TX Routines
         */
        void            forward(aodv_rt_entry *rt, Packet *p, double delay);
```

```
void            sendHello(void);
void            sendRequest(nsaddr_t dst);

void            sendReply(nsaddr_t ipdst, u_int32_t hop_count,
                          nsaddr_t rpdst, u_int32_t rpseq,
                          u_int32_t lifetime, double timestamp);
void            sendError(Packet *p, bool jitter = true);

/*
 * Packet RX Routines
 */
void            recvAODV(Packet *p);
void            recvHello(Packet *p);
void            recvRequest(Packet *p);
void            recvReply(Packet *p);
void            recvError(Packet *p);

/*
 * History management
 */

double          PerHopTime(aodv_rt_entry *rt);


nsaddr_t        index;                  // IP Address of this node
u_int32_t       seqno;                  // Sequence Number
int             bid;                    // Broadcast ID

aodv_rtable         rthead;                 // routing table
aodv_ncache         nbhead;                 // Neighbor Cache
aodv_bcache          bihead;                 // Broadcast ID Cache

/*
 * Timers
 */
BroadcastTimer  btimer;
HelloTimer      htimer;
NeighborTimer   ntimer;
RouteCacheTimer rtimer;
LocalRepairTimer lrtimer;

/*
 * Routing Table
 */
aodv_rtable          rtable;
/*
 *  A "drop-front" queue used by the routing layer to buffer
 *  packets to which it does not have a route.
 */
aodv_rqueue         rqueue;

/*
 * A mechanism for logging the contents of the routing
 * table.
 */
Trace           *logtarget;
```

```
        /*
         * A pointer to the network interface queue that sits
         * between the "classifier" and the "link layer".
         */
        PriQueue        *ifqueue;

        /*
         * Logging stuff
         */
        void            log_link_del(nsaddr_t dst);
        void            log_link_broke(Packet *p);
        void            log_link_kept(nsaddr_t dst);

        /* for passing packets up to agents */
        PortClassifier *dmux_;

};

#endif /* __aodv_h__ */
```

A.2.3 aodv/aodv.cc

```
/*
Copyright (c) 1997, 1998 Carnegie Mellon University.  All Rights
Reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice,
this list of conditions and the following disclaimer in the documentation
and/or other materials provided with the distribution.
3. The name of the author may not be used to endorse or promote products
derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE AUTHOR ``AS IS'' AND ANY EXPRESS OR
IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES
OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED.
IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS;
OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR
OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

The AODV code developed by the CMU/MONARCH group was optimized and tuned by Samir Das and Mahesh Marina,
University of Cincinnati. The work was partially done in Sun Microsystems. Modified for gratuitous replies
by Anant Utgikar, 09/16/02.

*/

/*
 * AODV マルチルート拡張 (UserData 連携 ver.)
 * Last Modified: 2008/12/11 18:48:14
```

```
 */

//#include <ip.h>

#include <aodv/aodv.h>
#include <aodv/aodv_packet.h>
#include <random.h>
#include <cmu-trace.h>
//#include <energy-model.h>

#define max(a,b)        ( (a) > (b) ? (a) : (b) )
#define CURRENT_TIME    Scheduler::instance().clock()

#define DEBUG
//#define ERROR

#ifdef DEBUG
static int extra_route_reply = 0;
static int limit_route_request = 0;
static int route_request = 0;
#endif


/*
  TCL Hooks
*/


int hdr_aodv::offset_;
static class AODVHeaderClass : public PacketHeaderClass {
public:
        AODVHeaderClass() : PacketHeaderClass("PacketHeader/AODV",
                                              sizeof(hdr_all_aodv)) {
          bind_offset(&hdr_aodv::offset_);
        }
} class_rtProtoAODV_hdr;

static class AODVclass : public TclClass {
public:
        AODVclass() : TclClass("Agent/AODV") {}
        TclObject* create(int argc, const char*const* argv) {
          assert(argc == 5);
          //return (new AODV((nsaddr_t) atoi(argv[4])));
          return (new AODV((nsaddr_t) Address::instance().str2addr(argv[4])));
        }
} class_rtProtoAODV;


int
AODV::command(int argc, const char*const* argv) {
  if(argc == 2) {
  Tcl& tcl = Tcl::instance();

    if(strncasecmp(argv[1], "id", 2) == 0) {
      tcl.resultf("%d", index);
      return TCL_OK;
    }
```

```
    if(strncasecmp(argv[1], "start", 2) == 0) {
      btimer.handle((Event*) 0);

#ifndef AODV_LINK_LAYER_DETECTION
      htimer.handle((Event*) 0);
      ntimer.handle((Event*) 0);
#endif // LINK LAYER DETECTION

      rtimer.handle((Event*) 0);
      return TCL_OK;
     }
  }
  else if(argc == 3) {

    if(strcmp(argv[1], "index") == 0) {
      index = atoi(argv[2]);
      return TCL_OK;
    }

    else if(strcmp(argv[1], "log-target") == 0 || strcmp(argv[1], "tracetarget") == 0) {
      logtarget = (Trace*) TclObject::lookup(argv[2]);
      if(logtarget == 0)
        return TCL_ERROR;
      return TCL_OK;
    }
    else if(strcmp(argv[1], "drop-target") == 0) {
    int stat = rqueue.command(argc,argv);
      if (stat != TCL_OK) return stat;
      return Agent::command(argc, argv);
    }
    else if(strcmp(argv[1], "if-queue") == 0) {
    ifqueue = (PriQueue*) TclObject::lookup(argv[2]);

      if(ifqueue == 0)
        return TCL_ERROR;
      return TCL_OK;
    }
    else if (strcmp(argv[1], "port-dmux") == 0) {
        dmux_ = (PortClassifier *)TclObject::lookup(argv[2]);
        if (dmux_ == 0) {
                fprintf (stderr, "%s: %s lookup of %s failed\n", __FILE__,
                argv[1], argv[2]);
                return TCL_ERROR;
        }
        return TCL_OK;
    }
  }
  return Agent::command(argc, argv);
}

/*
   Constructor
*/

AODV::AODV(nsaddr_t id) : Agent(PT_AODV),
                          btimer(this), htimer(this), ntimer(this),
```

```
                        rtimer(this), lrtimer(this), rqueue() {

        index = id;
        seqno = 2;
        bid = 1;

        LIST_INIT(&nbhead);
        LIST_INIT(&bihead);

        logtarget = 0;
        ifqueue = 0;

#ifdef AODV_USERDATA_CONNECT
        rt_request_status_ = 0;
#endif // AODV_USERDATA_CONNECT

}

/*
  Timers
*/

void
BroadcastTimer::handle(Event*) {
  agent->id_purge();
  Scheduler::instance().schedule(this, &intr, BCAST_ID_SAVE);
}

void
HelloTimer::handle(Event*) {
   agent->sendHello();
   double interval = MinHelloInterval +
               ((MaxHelloInterval - MinHelloInterval) * Random::uniform());
   assert(interval >= 0);
   Scheduler::instance().schedule(this, &intr, interval);
}

void
NeighborTimer::handle(Event*) {
  agent->nb_purge();
  Scheduler::instance().schedule(this, &intr, HELLO_INTERVAL);
}

void
RouteCacheTimer::handle(Event*) {
  agent->rt_purge();
#define FREQUENCY 0.5 // sec
  Scheduler::instance().schedule(this, &intr, FREQUENCY);
}

void
LocalRepairTimer::handle(Event* p)  {  // SRD: 5/4/99
aodv_rt_entry *rt;
struct hdr_ip *ih = HDR_IP( (Packet *)p);

   /* you get here after the timeout in a local repair attempt */
   /*   fprintf(stderr, "%s\n", __FUNCTION__); */
```

```
    rt = agent->rtable.rt_lookup(ih->daddr());

#ifndef AODV_MULTIROUTE
    if (rt && rt->rt_flags != RTF_UP) {
#else // AODV_MULTIROUTE
    if (rt && (rt->rt_flags != RTF_UP || rt->rt_flags != RTF_WAIT)) {
#endif // AODV_MULTIROUTE
    // route is yet to be repaired
    // I will be conservative and bring down the route
    // and send route errors upstream.
    /* The following assert fails, not sure why */
    /* assert (rt->rt_flags == RTF_IN_REPAIR); */

      //rt->rt_seqno++;
      agent->rt_down(rt);
      // send RERR
#ifdef DEBUG
      fprintf(stderr,"Node %d: Dst - %d, failed local repair\n",index, rt->rt_dst);
#endif
    }
    Packet::free((Packet *)p);
}


/*
   Broadcast ID Management  Functions
*/


void
AODV::id_insert(nsaddr_t id, u_int32_t bid) {
BroadcastID *b = new BroadcastID(id, bid);

 assert(b);
 b->expire = CURRENT_TIME + BCAST_ID_SAVE;
 LIST_INSERT_HEAD(&bihead, b, link);
}

/* SRD */
bool
AODV::id_lookup(nsaddr_t id, u_int32_t bid) {
BroadcastID *b = bihead.lh_first;

 // Search the list for a match of source and bid
 for( ; b; b = b->link.le_next) {
   if ((b->src == id) && (b->id == bid))
     return true;
 }
 return false;
}

void
AODV::id_purge() {
BroadcastID *b = bihead.lh_first;
BroadcastID *bn;
```

```
double now = CURRENT_TIME;

 for(; b; b = bn) {
   bn = b->link.le_next;
   if(b->expire <= now) {
     LIST_REMOVE(b,link);
     delete b;
   }
 }
}

/*
  Helper Functions
*/

double
AODV::PerHopTime(aodv_rt_entry *rt) {
int num_non_zero = 0, i;
double total_latency = 0.0;

 if (!rt)
   return ((double) NODE_TRAVERSAL_TIME );

 for (i=0; i < MAX_HISTORY; i++) {
   if (rt->rt_disc_latency[i] > 0.0) {
      num_non_zero++;
      total_latency += rt->rt_disc_latency[i];
   }
 }
 if (num_non_zero > 0)
   return(total_latency / (double) num_non_zero);
 else
   return((double) NODE_TRAVERSAL_TIME);

}

/*
  Link Failure Management Functions
*/

static void
aodv_rt_failed_callback(Packet *p, void *arg) {
  ((AODV*) arg)->rt_ll_failed(p);
}

/*
 * This routine is invoked when the link-layer reports a route failed.
 */
void
AODV::rt_ll_failed(Packet *p) {
struct hdr_cmn *ch = HDR_CMN(p);
struct hdr_ip *ih = HDR_IP(p);
aodv_rt_entry *rt;
nsaddr_t broken_nbr = ch->next_hop_;

#ifndef AODV_LINK_LAYER_DETECTION
 drop(p, DROP_RTR_MAC_CALLBACK);
```

```
#else

 /*
  * Non-data packets and Broadcast Packets can be dropped.
  */
  if(! DATA_PACKET(ch->ptype()) ||
     (u_int32_t) ih->daddr() == IP_BROADCAST) {
    drop(p, DROP_RTR_MAC_CALLBACK);
    return;
  }
  log_link_broke(p);
        if((rt = rtable.rt_lookup(ih->daddr())) == 0) {
    drop(p, DROP_RTR_MAC_CALLBACK);
    return;
  }
  log_link_del(ch->next_hop_);

#ifdef AODV_LOCAL_REPAIR
  /* if the broken link is closer to the dest than source,
     attempt a local repair. Otherwise, bring down the route. */


  if (ch->num_forwards() > rt->rt_hops) {
    local_rt_repair(rt, p); // local repair
    // retrieve all the packets in the ifq using this link,
    // queue the packets for which local repair is done,
    return;
  }
  else
#endif // LOCAL REPAIR

  {
    drop(p, DROP_RTR_MAC_CALLBACK);
    // Do the same thing for other packets in the interface queue using the
    // broken link -Mahesh
while((p = ifqueue->filter(broken_nbr))) {
     drop(p, DROP_RTR_MAC_CALLBACK);
    }
    nb_delete(broken_nbr);
  }

#endif // LINK LAYER DETECTION
}

void
AODV::handle_link_failure(nsaddr_t id) {
aodv_rt_entry *rt, *rtn;
Packet *rerr = Packet::alloc();
struct hdr_aodv_error *re = HDR_AODV_ERROR(rerr);

 re->DestCount = 0;
 for(rt = rtable.head(); rt; rt = rtn) { // for each rt entry
 /*
  * 切断したリンクを利用した経路が無いかルーティングテーブルの各エントリを調べる。
  *
  * AODV_MULTIROUTE では、次のような処理を行う。
  *  1. マルチルート用ルーティングテーブルの検索・削除処理
```

```
  *  2. 通常のルーティングテーブルに対する検索・削除処理
  *  3. 通常のルーティングテーブルの内容が削除され、かつマルチルート用テーブルに
  *     エントリが残っていたら、マルチルート用のルートの一つをメインに昇格
  *     残っていなかったら RTF_DOWN。
  *
  * ※類似の処理が recvError() にもあり。
  */

        rtn = rt->rt_link.le_next;

        /*
         * マルチルート用ルーティングテーブルに対する処理
         */
#ifdef AODV_MULTIROUTE
        u_int8_t j;
        for (j=0; j<rt->rt_routes; j++) {
                if (rt->rt_m_nexthop[j] == id) {
#ifdef DEBUG
                fprintf(stderr, "AODV %d %f m remove the multi table to %d via %d with %d hop(s)\n",
                                index, CURRENT_TIME, rt->rt_dst, j, rt->rt_m_hops[j]);
#endif // DEBUG
                rt->rt_m_delete(id);
                if( (rt->rt_hops == INFINITY2) && (rt->rt_nexthop != id) ) {
                        // ルーティングパケット関係の後処理。
                        // nexthop == id の時は通常のルーティングテーブルに対する処理のときに行う。
                        assert((rt->rt_seqno%2) == 0);
                        rt->rt_seqno++;
                        re->unreachable_dst[re->DestCount] = rt->rt_dst;
                        re->unreachable_dst_seqno[re->DestCount] = rt->rt_seqno;
#ifdef DEBUG
                        fprintf(stderr, "%s(%f): %d\t(%d\t%u\t%d)\n", __FUNCTION__, CURRENT_TIME,
                                        index, re->unreachable_dst[re->DestCount],
                                        re->unreachable_dst_seqno[re->DestCount], rt->rt_nexthop);
#endif // DEBUG
                        re->DestCount += 1;
                }
           }
        }
#endif // AODV_MULTIROUTE

        /*
         * 通常のルーティングテーブルに対する処理
         */
        if ((rt->rt_hops != INFINITY2) && (rt->rt_nexthop == id) ) {
#ifndef AODV_MULTIROUTE
        assert (rt->rt_flags == RTF_UP);
#else // AODV_MULTIROUTE
        assert (rt->rt_flags == RTF_UP || rt->rt_flags == RTF_WAIT);
#endif // AODV_MULTIROUTE
        assert((rt->rt_seqno%2) == 0);
        rt->rt_seqno++;
        re->unreachable_dst[re->DestCount] = rt->rt_dst;
        re->unreachable_dst_seqno[re->DestCount] = rt->rt_seqno;
#ifdef DEBUG
        fprintf(stderr, "%s(%f): %d\t(%d\t%u\t%d)\n", __FUNCTION__, CURRENT_TIME,
                        index, re->unreachable_dst[re->DestCount],
                        re->unreachable_dst_seqno[re->DestCount], rt->rt_nexthop);
```

```
#endif // DEBUG
        re->DestCount += 1;
#ifndef AODV_MULTIROUTE // マルチルートでは down は保留
        rt_down(rt);
#endif // AODV_MULTIROUTE
        }
        // remove the lost neighbor from all the precursor lists
        rt->pc_delete(id);

        /*
         * マルチルート関連の残りの処理
         */
#ifdef AODV_MULTIROUTE
        if (rt->rt_routes == 0) {
        // マルチルート用ルーティングテーブルのルート数が 0 だったら down
#ifdef DEBUG
                fprintf(stderr, "MULTIROUTE: %s: %d's multiple routing table is empty, down at %f.\n",
                                __FUNCTION__, index, CURRENT_TIME);
#endif // DEBUG
                rt_down(rt);
        } else if (rt->rt_nexthop == id) {
        // エントリ数が 0 でなく、今回削除したのがメインルートなら、サブルートをメインに
#ifdef DEBUG
                fprintf(stderr, "MULTIROUTE: %s: %d's multiple routing table is not empty, salvaged at %f.\n",
                                __FUNCTION__, index, CURRENT_TIME);
                //rt->rt_m_dump();
#endif // DEBUG
                rt->rt_hops = rt->rt_m_hops[0];
                rt->rt_nexthop = rt->rt_m_nexthop[0];
        }
#endif // AODV_MULTIROUTE

 }

 if (re->DestCount > 0) {
        // 削除したエントリーがあったので RERR を転送 (BROADCAST)
#ifdef DEBUG
        fprintf(stderr, "%s(%f): %d\tsending RERR...\n", __FUNCTION__, CURRENT_TIME, index);
#endif // DEBUG
        sendError(rerr, false);
 } else {
        Packet::free(rerr);
 }

}

void
AODV::local_rt_repair(aodv_rt_entry *rt, Packet *p) {
#ifdef DEBUG
  fprintf(stderr,"%s: Dst - %d\n", __FUNCTION__, rt->rt_dst);
#endif
  // Buffer the packet
  rqueue.enque(p);

  // mark the route as under repair
  rt->rt_flags = RTF_IN_REPAIR;
```

```
  sendRequest(rt->rt_dst);

  // set up a timer interrupt
  Scheduler::instance().schedule(&lrtimer, p->copy(), rt->rt_req_timeout);
}

void
AODV::rt_update(aodv_rt_entry *rt, u_int32_t seqnum, u_int16_t metric,
                nsaddr_t nexthop, double expire_time) {

#ifdef DEBUG
        fprintf(stderr, "AODV %d %f t update the master table to %d via %d with %d hop(s)\n",
                index, CURRENT_TIME, rt->rt_dst, nexthop, metric);
#endif // DEBUG

     rt->rt_seqno = seqnum;
     rt->rt_hops = metric;
#ifndef AODV_MULTIROUTE
     rt->rt_flags = RTF_UP;
#else // AODV_MULTIROUTE
        // 既に UP している場合は WAIT させない
        if(rt->rt_flags != RTF_UP) rt->rt_flags = RTF_WAIT;
        rt->rt_m_create = CURRENT_TIME;
#endif // AODV_MULTIROUTE
     rt->rt_nexthop = nexthop;
     rt->rt_expire = expire_time;
}

#ifdef AODV_MULTIROUTE
/* RTF_UP への切替 */
void
AODV::rt_up(aodv_rt_entry *rt) {

#ifdef DEBUG
        fprintf(stderr, "AODV %d %f t RTF_UP for %d\n",
                index, CURRENT_TIME, rt->rt_dst);
#endif // DEBUG

        rt->rt_flags = RTF_UP;
}
#endif // AODV_MULTIROUTE

void
AODV::rt_down(aodv_rt_entry *rt) {
  /*
   *  Make sure that you don't "down" a route more than once.
   */

        if(rt->rt_flags == RTF_DOWN) {
                return;
        }

#ifdef DEBUG
        fprintf(stderr, "AODV %d %f t RTF_DOWN for %d\n",
                index, CURRENT_TIME, rt->rt_dst);
#endif // DEBUG
```

```
        // assert (rt->rt_seqno%2); // is the seqno odd?
        rt->rt_last_hop_count = rt->rt_hops;
        rt->rt_hops = INFINITY2;
        rt->rt_flags = RTF_DOWN;
        rt->rt_nexthop = 0;
        rt->rt_expire = 0;

#ifdef AODV_MULTIROUTE
        rt->rt_m_hops[0] = INFINITY2;
        rt->rt_m_nexthop[0] = 0;
        rt->rt_routes = 0;
        rt->rt_counter = 0;
        rt->rt_m_create = 0;
#endif // AODV_MULTIROUTE

#ifdef AODV_USERDATA_CONNECT
        rt_request_status_ = 0;
#endif // AODV_USERDATA_CONNECT

} /* rt_down function */

/*
  Route Handling Functions
*/

void
AODV::rt_resolve(Packet *p) {
struct hdr_cmn *ch = HDR_CMN(p);
struct hdr_ip *ih = HDR_IP(p);
aodv_rt_entry *rt;

 /*
  *  Set the transmit failure callback.  That
  *  won't change.
  */
 ch->xmit_failure_ = aodv_rt_failed_callback;
 ch->xmit_failure_data_ = (void*) this;
       rt = rtable.rt_lookup(ih->daddr());
 if(rt == 0) {
         rt = rtable.rt_add(ih->daddr());
 }

 /*
  * If the route is up, forward the packet
  */

 if(rt->rt_flags == RTF_UP) {
   assert(rt->rt_hops != INFINITY2);
   forward(rt, p, NO_DELAY);
 }
#ifdef AODV_MULTIROUTE
 /*
  * 追加の RREP パケットを待っている送信元ノードの場合、パケットをバッファする。
  */
 else if (ih->saddr() == index && rt->rt_flags == RTF_WAIT) {
       rqueue.enque(p);
 }
```

```
 /*
  * 中継ノードの場合、RTF_WAIT 状態でも送信する。
  */
 else if (ih->saddr() != index && rt->rt_flags == RTF_WAIT) {
        assert(rt->rt_hops != INFINITY2);
        forward(rt, p, NO_DELAY);
 }
#endif // AODV_MULTIROUTE
 /*
  *  if I am the source of the packet, then do a Route Request.
  */
 else if(ih->saddr() == index) {
   rqueue.enque(p);
   sendRequest(rt->rt_dst);
 }
 /*
  *     A local repair is in progress. Buffer the packet.
  */
 else if (rt->rt_flags == RTF_IN_REPAIR) {
   rqueue.enque(p);
 }

 /*
  * I am trying to forward a packet for someone else to which
  * I don't have a route.
  */
 else {
 Packet *rerr = Packet::alloc();
 struct hdr_aodv_error *re = HDR_AODV_ERROR(rerr);
 /*
  * For now, drop the packet and send error upstream.
  * Now the route errors are broadcast to upstream
  * neighbors - Mahesh 09/11/99
  */

   assert (rt->rt_flags == RTF_DOWN);
   re->DestCount = 0;
   re->unreachable_dst[re->DestCount] = rt->rt_dst;
   re->unreachable_dst_seqno[re->DestCount] = rt->rt_seqno;
   re->DestCount += 1;
#ifdef DEBUG
   fprintf(stderr, "%s: %d sending RERR...\n", __FUNCTION__, index);
#endif
   sendError(rerr, false);

   drop(p, DROP_RTR_NO_ROUTE);
 }

}

void
AODV::rt_purge() {
aodv_rt_entry *rt, *rtn;
double now = CURRENT_TIME;
double delay = 0.0;
Packet *p;
```

```
 /*
  * RouteCacheTimer::handle() から FREQUENCY おきに呼び出される
  */

 for(rt = rtable.head(); rt; rt = rtn) {  // for each rt entry
        rtn = rt->rt_link.le_next;
#ifndef AODV_MULTIROUTE
        if ((rt->rt_flags == RTF_UP) && (rt->rt_expire < now)) {
#else // AODV_MULTIROUTE
        // マルチルート有効時には、RTF_WAIT 中のルートに対しても同様に expire 処理する。
        if ((rt->rt_flags == RTF_UP || rt->rt_flags == RTF_WAIT) && (rt->rt_expire < now)) {
#endif // AODV_MULTIROUTE
                // if a valid route has expired, purge all packets from
                // send buffer and invalidate the route.
                assert(rt->rt_hops != INFINITY2);
                while((p = rqueue.deque(rt->rt_dst))) {
#ifdef DEBUG
                        fprintf(stderr, "%s: calling drop()\n",
                                        __FUNCTION__);
#endif // DEBUG
                        drop(p, DROP_RTR_NO_ROUTE);
                }
                rt->rt_seqno++;
                assert (rt->rt_seqno%2);
                rt_down(rt);
        }
#ifdef AODV_MULTIROUTE
#ifdef MULTIROUTE_TIMEOUT
        // MULTIROUTE_TIMEOUT が指定されている時には、指定本数のルートが構築できていなくても、
        // 現在あるルートだけで RTF_UP にする。
        else if ((rt->rt_flags == RTF_WAIT) && (rt->rt_m_create + MULTIROUTE_TIMEOUT  < now)) {

#ifdef DEBUG
                fprintf(stderr, "MULTIROUTE: %s: %d's RTF_WAIT for %d timeout at %f\n",
                        __FUNCTION__, index, rt->rt_dst, CURRENT_TIME);
#endif // DEBUG
                rt_up(rt);

#ifdef AODV_USERDATA_CONNECT
                userdata_callback(rt->rt_dst);
#endif // AODV_USERDATA_CONNECT
                while((p = rqueue.deque(rt->rt_dst))) {
                        forward (rt, p, delay);
                        delay += ARP_DELAY;
                }

        }
#endif // MULTIROUTE_TIMEOUT
#endif // AODV_MULTIROUTE
        else if (rt->rt_flags == RTF_UP) {
                // If the route is not expired,
                // and there are packets in the sendbuffer waiting,
                // forward them. This should not be needed, but this extra
                // check does no harm.
                assert(rt->rt_hops != INFINITY2);
                while((p = rqueue.deque(rt->rt_dst))) {
                        forward (rt, p, delay);
```

```
                        delay += ARP_DELAY;
                }
        }
        else if (rqueue.find(rt->rt_dst)) {
                // If the route is down and
                // if there is a packet for this destination waiting in
                // the sendbuffer, then send out route request. sendRequest
                // will check whether it is time to really send out request
                // or not.
                // This may not be crucial to do it here, as each generated
                // packet will do a sendRequest anyway.

#ifdef DEBUG
                fprintf(stderr, "%s: %d calling sendRequest(%d)\n", __FUNCTION__, index, rt->rt_dst);
#endif // DEBUG

                sendRequest(rt->rt_dst);
        }
#ifdef AODV_USERDATA_CONNECT
        else {
                userdata_fail_callback(rt->rt_dst);
        }
#endif // AODV_USERDATA_CONNECT

 }

}

/*
  Packet Reception Routines
*/

void
AODV::recv(Packet *p, Handler*) {
struct hdr_cmn *ch = HDR_CMN(p);
struct hdr_ip *ih = HDR_IP(p);

 assert(initialized());
 //assert(p->incoming == 0);
 // XXXXX NOTE: use of incoming flag has been depracated; In order to track direction of pkt flow, direction_
 in hdr_cmn is used instead. see packet.h for details.

 if(ch->ptype() == PT_AODV) {
   ih->ttl_ -= 1;
   recvAODV(p);
   return;
 }


 /*
  *  Must be a packet I'm originating...
  */
if((ih->saddr() == index) && (ch->num_forwards() == 0)) {
 /*
  * Add the IP Header
  */
   ch->size() += IP_HDR_LEN;
```

```
   // Added by Parag Dadhania && John Novatnack to handle broadcasting
   if ( (u_int32_t)ih->daddr() != IP_BROADCAST)
     ih->ttl_ = NETWORK_DIAMETER;
}
 /*
  *  I received a packet that I sent.  Probably
  *  a routing loop.
  */
else if(ih->saddr() == index) {
   drop(p, DROP_RTR_ROUTE_LOOP);
   return;
 }
 /*
  *  Packet I'm forwarding...
  */
 else {
 /*
  *  Check the TTL.  If it is zero, then discard.
  */
   if(--ih->ttl_ == 0) {
     drop(p, DROP_RTR_TTL);
     return;
   }
 }
// Added by Parag Dadhania && John Novatnack to handle broadcasting
 if ( (u_int32_t)ih->daddr() != IP_BROADCAST)
   rt_resolve(p);
 else
   forward((aodv_rt_entry*) 0, p, NO_DELAY);
}


void
AODV::recvAODV(Packet *p) {
 struct hdr_aodv *ah = HDR_AODV(p);

 assert(HDR_IP (p)->sport() == RT_PORT);
 assert(HDR_IP (p)->dport() == RT_PORT);

 /*
  * Incoming Packets.
  */
 switch(ah->ah_type) {

 case AODVTYPE_RREQ:
   recvRequest(p);
   break;

 case AODVTYPE_RREP:
   recvReply(p);
   break;

 case AODVTYPE_RERR:
   recvError(p);
   break;

 case AODVTYPE_HELLO:
```

```
   recvHello(p);
   break;

 default:
   fprintf(stderr, "Invalid AODV type (%x)\n", ah->ah_type);
   exit(1);
 }

}


void
AODV::recvRequest(Packet *p) {
struct hdr_ip *ih = HDR_IP(p);
struct hdr_aodv_request *rq = HDR_AODV_REQUEST(p);
aodv_rt_entry *rt;

/*
 *  アドレス参照方法一覧
 *
 *   * 送信元           rq->rq_src
 *   |
 *   * 前ホップ         ih->saddr()
 *   |
 *   x 自ノード         index
 *   |
 *   * 次ホップ
 *   |
 *   * 送信先           rq->rq_dst
 */

#ifdef DEBUG
 fprintf(stderr, "AODV %d %f r RREQ #%d from %d to %d via %d\n",
                index, CURRENT_TIME, rq->rq_bcast_id, rq->rq_src, rq->rq_dst, ih->saddr());
#endif // DEBUG

 /*
  * Drop if:
  *      - I'm the source
  *      - I recently heard this request.
  *
  * drop させる条件:
  *      - 自分が送信元の場合
  *      - その RREQ を受信したことがある場合
  *
  * ** AODV_MULTIROUTE においては、受信元では 2 番目の条件を無視する
  */

 if(rq->rq_src == index) {
        // 自分の送信した RREQ の場合
#ifdef DEBUG
        //fprintf(stderr, "%s: %d got my own REQUEST\n", __FUNCTION__, index);
        fprintf(stderr, "AODV %d %f i RREQ #%d discarded own\n",
                index, CURRENT_TIME, rq->rq_bcast_id);
#endif // DEBUG
        Packet::free(p);
        return;
```

```
 }

#ifndef AODV_MULTIROUTE
 // 重複 RREQ は drop させる
 if (id_lookup(rq->rq_src, rq->rq_bcast_id)) {
#else // AODV_MULTIROUTE
 // AODV_MULTIROUTE では、受信ノードのみ重複 RREQ を破棄しない
 if (rq->rq_dst != index && id_lookup(rq->rq_src, rq->rq_bcast_id)) {
#endif // AODV_MULTIROUTE

#ifdef DEBUG
        //fprintf(stderr, "%s: %d discarding request\n", __FUNCTION__, index);
        fprintf(stderr, "AODV %d %f i RREQ #%d discarded dup\n",
                        index, CURRENT_TIME, rq->rq_bcast_id);
#endif // DEBUG

        Packet::free(p);
        return;
 }

 /*
  * Cache the broadcast ID (broadcast ID を記憶する)
  */
#ifdef AODV_MULTIROUTE
 if ( !id_lookup(rq->rq_src, rq->rq_bcast_id)) {        // 二重挿入回避
#endif // AODV_MULTIROUTE
 id_insert(rq->rq_src, rq->rq_bcast_id);
#ifdef AODV_MULTIROUTE
 }
#endif // AODV_MULTIROUTE

 /*
  * We are either going to forward the REQUEST or generate a
  * REPLY. Before we do anything, we make sure that the REVERSE
  * route is in the route table.
  *
  * RREQ の転送または、RREP の生成を行う。
  * どちらを行う場合でも、ルーティングテーブルに逆経路が確実に存在する状態にする。
  */
 aodv_rt_entry *rt0; // rt0 is the reverse route

   rt0 = rtable.rt_lookup(rq->rq_src);
   if(rt0 == 0) { /* if not in the route table */
   // create an entry for the reverse route.
     rt0 = rtable.rt_add(rq->rq_src);
   }

   rt0->rt_expire = max(rt0->rt_expire, (CURRENT_TIME + REV_ROUTE_LIFE));

   if ( (rq->rq_src_seqno > rt0->rt_seqno ) ||
        ((rq->rq_src_seqno == rt0->rt_seqno) &&
         (rq->rq_hop_count < rt0->rt_hops)) ) {
   // If we have a fresher seq no. or lesser #hops for the
   // same seq no., update the rt entry. Else don't bother.
       rt_update(rt0, rq->rq_src_seqno, rq->rq_hop_count, ih->saddr(),
              max(rt0->rt_expire, (CURRENT_TIME + REV_ROUTE_LIFE)) );
     if (rt0->rt_req_timeout > 0.0) {
```

```
     // Reset the soft state and
     // Set expiry time to CURRENT_TIME + ACTIVE_ROUTE_TIMEOUT
     // This is because route is used in the forward direction,
     // but only sources get benefited by this change
       rt0->rt_req_cnt = 0;
       rt0->rt_req_timeout = 0.0;
       rt0->rt_req_last_ttl = rq->rq_hop_count;
       rt0->rt_expire = CURRENT_TIME + ACTIVE_ROUTE_TIMEOUT;
     }

     /* Find out whether any buffered packet can benefit from the
      * reverse route.
      * May need some change in the following code - Mahesh 09/11/99
      */
#ifndef AODV_MULTIROUTE
     assert (rt0->rt_flags == RTF_UP);
#else // AODV_MULTIROUTE
     assert (rt0->rt_flags == RTF_UP || rt0->rt_flags == RTF_WAIT);
     if (rt0->rt_flags == RTF_UP) {     // 念のため
#endif // AODV_MULTIROUTE
     Packet *buffered_pkt;
     while ((buffered_pkt = rqueue.deque(rt0->rt_dst))) {
       if (rt0 && (rt0->rt_flags == RTF_UP)) {
        assert(rt0->rt_hops != INFINITY2);
         forward(rt0, buffered_pkt, NO_DELAY);
       }
     }
#ifdef AODV_MULTIROUTE
     }
#endif // AODV_MULTIROUTE
   }
   // End for putting reverse route in rt table
   // 逆経路を追加する処理終了

#ifdef AODV_MULTIROUTE
        /*
         *  マルチルート用ルーティングテーブルへの経路追加処理
         */
        if ( rq->rq_dst != index ) {
                // 宛先ノードではない場合：追加処理はしない
#ifdef DEBUG
        //      fprintf(stderr, "MULTIROUTE: %s: %d skip request #%d\n",
        //              __FUNCTION__, index, rq->rq_bcast_id);
        fprintf(stderr, "AODV %d %f m RREQ #%d ignored table nondst\n",
                index, CURRENT_TIME, rq->rq_bcast_id);
#endif // DEBUG
        } else if (! rt0->rt_m_add(ih->saddr(), rq->rq_hop_count)) {
                // ルーティングテーブルが満杯 / 既に同じルートが存在する場合：drop させる
#ifdef DEBUG
        //      fprintf(stderr, "MULTIROUTE: %s: %d discarding request #%d, routing table is full or duplicate
 route.\n",
        //              __FUNCTION__, index, rq->rq_bcast_id);
        fprintf(stderr, "AODV %d %f m RREQ #%d discarded table full/dup\n",
                index, CURRENT_TIME, rq->rq_bcast_id);
#endif // DEBUG
                Packet::free(p);
                return;
```

```
        } else {
                // 一つ上の条件式で経路を追加できた場合：経路数が ROUTE_COUNT に達していれば経路を UP
                if(rt0->rt_routes >= ROUTE_COUNT) {
                        rt_up(rt0);
                        assert(rt0->rt_flags == RTF_UP);
                        // 溜まっていたパケットの送信
                        Packet *buf_pkt;
                        while ((buf_pkt = rqueue.deque(rt0->rt_dst))) {
                                if (rt0 && (rt0->rt_flags == RTF_UP)) {
                                        assert(rt0->rt_hops != INFINITY2);
                                        forward(rt0, buf_pkt, NO_DELAY);
                                }
                        }
                }
#ifdef DEBUG
        //      fprintf(stderr, "MULTIROUTE: %s: %d add a new route to %d via %d with hop count %d\n",
        //                      __FUNCTION__, index, rq->rq_src, ih->saddr(), rq->rq_hop_count);
                fprintf(stderr, "AODV %d %f m RREQ #%d add table to %d via %d with %d hop(s)\n",
                        index, CURRENT_TIME, rq->rq_bcast_id, rq->rq_src, ih->saddr(), rq->rq_hop_count);
                //rt0->rt_m_dump();
#endif // DEBUG
        }
#endif // AODV_MULTIROUTE

 /*
  * We have taken care of the reverse route stuff.
  * Now see whether we can send a route reply.
  */

 rt = rtable.rt_lookup(rq->rq_dst);

 // First check if I am the destination ..

 if(rq->rq_dst == index) {

//#ifdef DEBUG
//   fprintf(stderr, "%d - %s: destination sending reply\n",
//                   index, __FUNCTION__);
//#endif // DEBUG


   // Just to be safe, I use the max. Somebody may have
   // incremented the dst seqno.
   seqno = max(seqno, rq->rq_dst_seqno)+1;
   if (seqno%2) seqno++;

   sendReply(rq->rq_src,           // IP Destination
             1,                    // Hop Count
             index,                // Dest IP Address
             seqno,                // Dest Sequence Num
             MY_ROUTE_TIMEOUT,     // Lifetime
             rq->rq_timestamp);    // timestamp

   Packet::free(p);
 }

 // I am not the destination, but I may have a fresh enough route.
```

```
 else if (rt && (rt->rt_hops != INFINITY2) &&
                (rt->rt_seqno >= rq->rq_dst_seqno) ) {

   //assert (rt->rt_flags == RTF_UP);
   assert(rq->rq_dst == rt->rt_dst);
   //assert ((rt->rt_seqno%2) == 0);    // is the seqno even?
   sendReply(rq->rq_src,
             rt->rt_hops + 1,
             rq->rq_dst,
             rt->rt_seqno,
             (u_int32_t) (rt->rt_expire - CURRENT_TIME),
             //             rt->rt_expire - CURRENT_TIME,
             rq->rq_timestamp);
   // Insert nexthops to RREQ source and RREQ destination in the
   // precursor lists of destination and source respectively
   rt->pc_insert(rt0->rt_nexthop); // nexthop to RREQ source
   rt0->pc_insert(rt->rt_nexthop); // nexthop to RREQ destination

#ifdef RREQ_GRAT_RREP

   sendReply(rq->rq_dst,
             rq->rq_hop_count,
             rq->rq_src,
             rq->rq_src_seqno,
             (u_int32_t) (rt->rt_expire - CURRENT_TIME),
             //             rt->rt_expire - CURRENT_TIME,
             rq->rq_timestamp);
#endif

// TODO: send grat RREP to dst if G flag set in RREQ using rq->rq_src_seqno, rq->rq_hop_counT

// DONE: Included gratuitous replies to be sent as per IETF aodv draft specification. As of now,
G flag has not been dynamically used and is always set or reset in aodv-packet.h --- Anant Utgikar, 09/16/02.

        Packet::free(p);
 }
 /*
  * Can't reply. So forward the  Route Request
  * (RREP を返せないので、RREQ を転送する)
  */
 else {
#ifdef DEBUG
        fprintf(stderr, "AODV %d %f s RREQ #%d forward\n",
                index, CURRENT_TIME, rq->rq_bcast_id);
#endif // DEBUG
   ih->saddr() = index;
   ih->daddr() = IP_BROADCAST;
   rq->rq_hop_count += 1;
   // Maximum sequence number seen en route
   if (rt) rq->rq_dst_seqno = max(rt->rt_seqno, rq->rq_dst_seqno);
   forward((aodv_rt_entry*) 0, p, DELAY);
 }

}
```

```
void
AODV::recvReply(Packet *p) {
struct hdr_cmn *ch = HDR_CMN(p);
struct hdr_ip *ih = HDR_IP(p);
struct hdr_aodv_reply *rp = HDR_AODV_REPLY(p);
aodv_rt_entry *rt;
char suppress_reply = 0;
double delay = 0.0;

/*
 * ・この関数内にある、英語と日本語が併記されているコメントは、オリジナルのコードに
 *   記載されていたコメントを杉本が和訳したものです。
 * ・アドレス参照方法一覧
 *   dst, src はデータパケット転送方向が基準なので注意。rt 系は rt_lookup 後から使用可。
 *
 *     * RREP 送信先（データ送信元）       (rt->rt_dst), ih->daddr()
 *     |
 *     * 次ホップ                          rp->rp_src, (rt->rt_nexthop)
 *     |
 *     x 自ノード                          index
 *     |
 *     * 前ホップ                          ih->saddr()
 *     |
 *     * RREP 送信元（データ送信先）       rp->rp_dst
 */

#ifdef DEBUG
 //fprintf(stderr, "%d - %s: received a REPLY from %d via %d\n", index, __FUNCTION__, rp->rp_dst, rp->rp_src);
 fprintf(stderr, "AODV %d %f r RREP from %d via %d for RREQ %f\n",
                index, CURRENT_TIME, rp->rp_dst, rp->rp_src, rp->rp_timestamp);
#endif // DEBUG

 /*
  *  Got a reply. So reset the "soft state" maintained for
  *  route requests in the request table. We don't really have
  *  have a separate request table. It is just a part of the
  *  routing table itself.
  *  RREP を受信したため、リクエストテーブル内の RREQ に関する"soft state"をリセット
  *  する。この実装では、独立したリクエストテーブルはなく、ルーティングテーブルの
  *  一部になっている。
  */
 // Note that rp_dst is the dest of the data packets, not the
 // the dest of the reply, which is the src of the data packets.
 // rp_dst はデータパケットの送信先を示している。
 // RREP の送信先、すなわちデータパケットの送信元ではないことに注意。

 rt = rtable.rt_lookup(rp->rp_dst);

 /*
  *  If I don't have a rt entry to this host... adding
  *  ルーティングテーブルに送信先に対するエントリが存在していなければ追加する。
  */
 if(rt == 0) {
        rt = rtable.rt_add(rp->rp_dst);
 }

 /*
```

```
  * Add a forward route table entry... here I am following
  * Perkins-Royer AODV paper almost literally - SRD 5/99
  * 送信方向のルーティングテーブルエントリ更新処理。
  * 以下の処理は Perkins-Royer による AODV の論文にほぼ従っている。
  *（訳注：マルチルート修正により論文とは異なっていることに注意）
  */

 if ( (rt->rt_seqno < rp->rp_dst_seqno) ||   // newer route
      ((rt->rt_seqno == rp->rp_dst_seqno) &&
       (rt->rt_hops > rp->rp_hop_count)) ) { // shorter or better route

        // Update the rt entry
        rt_update(rt, rp->rp_dst_seqno, rp->rp_hop_count,
                rp->rp_src, CURRENT_TIME + rp->rp_lifetime);

        // reset the soft state
        rt->rt_req_cnt = 0;
        rt->rt_req_timeout = 0.0;
        rt->rt_req_last_ttl = rp->rp_hop_count;

        if (ih->daddr() == index) { // If I am the original source
                // Update the route discovery latency statistics
                // rp->rp_timestamp is the time of request origination

                rt->rt_disc_latency[(unsigned char)rt->hist_indx] = (CURRENT_TIME - rp->rp_timestamp)
                                                / (double) rp->rp_hop_count;
                // increment indx for next time
                rt->hist_indx = (rt->hist_indx + 1) % MAX_HISTORY;
        }

#ifdef AODV_MULTIROUTE
 } // マルチルート関連処理挿入のため、if 括弧をここで一旦閉じる。
 else {
        suppress_reply = 1;
#ifdef DEBUG
        fprintf(stderr, "%s: %d suppressed reply at %f\n", __FUNCTION__, index, CURRENT_TIME);
#endif // DEBUG
 }
        if(ih->daddr() == index) {
        /*
         * 自分が RREQ 送信元ならば、マルチルート用ルーティングテーブルへルートを追加。
         */
                if (! rt->rt_m_add(rp->rp_src, rp->rp_hop_count)) {
#ifdef DEBUG
                        //fprintf(stderr, "MULTIROUTE: %s: %d failed to add a route to %d via %d at %f\n",
                        //              __FUNCTION__, index,
                        //              rp->rp_dst, rp->rp_src,
                        //              CURRENT_TIME);
                        fprintf(stderr, "AODV %d %f m failed to add route to %d via %d\n",
                                        index, CURRENT_TIME, rp->rp_dst, rp->rp_src);
#endif // DEBUG
                } else {
                        if (rt->rt_routes >= ROUTE_COUNT) {
                                rt_up(rt);
                        }
#ifdef DEBUG
                        fprintf(stderr, "MULTIROUTE: %s: %d add a new route to %d via %d with hop count %d at %f\n",
```

```
                                        __FUNCTION__, index,
                                        rp->rp_dst, rp->rp_src, rp->rp_hop_count,
                                        CURRENT_TIME);
                        rt->rt_m_dump();
#endif // DEBUG
                }
        }

 // 一旦切った if 文の条件をここで復活させる（送信元の例外を追加）
 if ( ( ih->daddr() != index && (       // 自分が p の宛先ではない かつ
        (rt->rt_seqno < rp->rp_dst_seqno) ||   // 新しいルート（シーケンス番号が若い）または
        ((rt->rt_seqno == rp->rp_dst_seqno) &&
        (rt->rt_hops > rp->rp_hop_count))      // 同じシーケンス番号で、ホップ数が少ない良いルート
 ) ) || (
        (ih->daddr() == index) &&       // 自分が p の宛先 かつ
        (rt->rt_flags == RTF_UP)                // ルートが UP している場合
 ) ) {
#endif // AODV_MULTIROUTE

        /*
         * Send all packets queued in the sendbuffer destined for
         * this destination.
         * キュー内にある、今回追加したルート向けのパケットを全て送信する。
         * XXX - observe the "second" use of p.
         */
#ifdef DEBUG
        fprintf(stderr, "%s: %d send all packet in sendbuffer at %f\n", __FUNCTION__, index, CURRENT_TIME);
#endif //DEBUG
#ifdef AODV_USERDATA_CONNECT
        userdata_callback(rt->rt_dst);
#endif // AODV_USERDATA_CONNECT
        Packet *buf_pkt;
        while((buf_pkt = rqueue.deque(rt->rt_dst))) {
                if(rt->rt_hops != INFINITY2) {
                        assert (rt->rt_flags == RTF_UP);
                        // Delay them a little to help ARP. Otherwise ARP
                        // may drop packets. -SRD 5/23/99
                        forward(rt, buf_pkt, delay);
                        delay += ARP_DELAY;
                }
        }
 }
#ifndef AODV_MULTIROUTE
 else {
        suppress_reply = 1;
#ifdef DEBUG
        fprintf(stderr, "%s: %d suppressed reply at %f\n", __FUNCTION__, index, CURRENT_TIME);
#endif // DEBUG
 }
#endif // AODV_MULTIROUTE

 /*
  * If reply is for me, discard it.
  */
 if(ih->daddr() == index || suppress_reply) {
        Packet::free(p);
 }
```

```
 /*
  * Otherwise, forward the Route Reply.
  */
 else {
        // Find the rt entry
        aodv_rt_entry *rt0 = rtable.rt_lookup(ih->daddr());
        // If the rt is up, forward
        if(rt0 && (rt0->rt_hops != INFINITY2)) {
                assert (rt0->rt_flags == RTF_UP);
                rp->rp_hop_count += 1;
                rp->rp_src = index;
                forward(rt0, p, NO_DELAY);
                // Insert the nexthop towards the RREQ source to
                // the precursor list of the RREQ destination
                rt->pc_insert(rt0->rt_nexthop); // nexthop to RREQ source

#ifdef DEBUG
//     fprintf(stderr, "%s: %d forwarding Route Reply\n", __FUNCTION__, index);
                fprintf(stderr, "AODV %d %f f RREP forward to %d via %d\n",
                        index, CURRENT_TIME, rt0->rt_dst, rt0->rt_nexthop);
#endif // DEBUG

        }
        else {
                // I don't know how to forward .. drop the reply.

#ifdef DEBUG
                fprintf(stderr, "%s: %d dropping Route Reply\n", __FUNCTION__, index);
#endif // DEBUG

                drop(p, DROP_RTR_NO_ROUTE);
        }
 }
}


void
AODV::recvError(Packet *p) {
struct hdr_ip *ih = HDR_IP(p);
struct hdr_aodv_error *re = HDR_AODV_ERROR(p);
aodv_rt_entry *rt;
u_int8_t i;
Packet *rerr = Packet::alloc();
struct hdr_aodv_error *nre = HDR_AODV_ERROR(rerr);

#ifdef DEBUG
 fprintf(stderr, "AODV %d %f r RERR %d\n",
                index, CURRENT_TIME, re->DestCount);
#endif // DEBUG

 nre->DestCount = 0;
 for (i=0; i<re->DestCount; i++) { // For each unreachable destination
 /*
  * RERR に含まれる経路がルーティングテーブルに無いか、各エントリを調べる。
  *
  * AODV_MULTIROUTE では、次のような処理を行う。
  *  1. マルチルート用ルーティングテーブルの検索・削除処理
```

```
  *  2. 通常のルーティングテーブルに対する確認・削除処理
  *  3. 通常のルーティングテーブルの内容が削除され、かつマルチルート用テーブルに
  *     エントリが残っていたら、マルチルート用のルートの一つをメインに昇格。
  *     残っていなかったら RTF_DOWN。
  *
  * ** 類似の処理が handle_link_failure() にもあり。
  */

        rt = rtable.rt_lookup(re->unreachable_dst[i]);

        /*
         * マルチルート用ルーティングテーブルに対する処理
         */
#ifdef AODV_MULTIROUTE
        if ( rt && (rt->rt_seqno <= re->unreachable_dst_seqno[i]) ) {
                u_int8_t j;
                for (j=0; j<rt->rt_routes; j++) {
                        if (rt->rt_m_nexthop[j] == ih->saddr()) {
                                assert((rt->rt_seqno%2) == 0);
#ifdef DEBUG
                                fprintf(stderr, "%s(%f): %d\t(%d\t%u\t%d)\t(%d\t%u\t%d)\n", __FUNCTION__,CURRENT_TIME,
                                                index, rt->rt_dst, rt->rt_seqno, rt->rt_nexthop,
                                                re->unreachable_dst[i],re->unreachable_dst_seqno[i],
                                                ih->saddr());
#endif // DEBUG
                                rt->rt_seqno = re->unreachable_dst_seqno[i];
#ifdef DEBUG
                                fprintf(stderr, "AODV %d %f t route #%d for %d removed\n",
                                        index, CURRENT_TIME, j, rt->rt_dst);
#endif // DEBUG
                                rt->rt_m_delete(ih->saddr());
                        }
                }
        }
#endif // AODV_MULTIROUTE

        /*
         * 通常のルーティングテーブルに対する処理
         */
        if ( rt && (rt->rt_hops != INFINITY2) &&
                (rt->rt_nexthop == ih->saddr()) &&
                (rt->rt_seqno <= re->unreachable_dst_seqno[i]) )
        {
                assert(rt->rt_flags == RTF_UP);
                assert((rt->rt_seqno%2) == 0); // is the seqno even?

#ifdef DEBUG
                fprintf(stderr, "%s(%f): %d\t(%d\t%u\t%d)\t(%d\t%u\t%d)\n", __FUNCTION__,CURRENT_TIME,
                                index, rt->rt_dst, rt->rt_seqno, rt->rt_nexthop,
                                re->unreachable_dst[i],re->unreachable_dst_seqno[i],
                                ih->saddr());
#endif // DEBUG

                rt->rt_seqno = re->unreachable_dst_seqno[i];
#ifndef AODV_MULTIROUTE
                rt_down(rt);
#else // AODV_MULTIROUTE
```

```
                if (rt->rt_routes == 0) {
                        // マルチルート用ルーティングテーブルも空なら rt_down()
#ifdef DEBUG
                        fprintf(stderr, "MULTIROUTE: %s: %d's multiple routing table is empty, down at %f.\n",
                                        __FUNCTION__, index, CURRENT_TIME);
#endif // DEBUG
                        rt_down(rt);
                } else {
                        // マルチルート用ルーティングテーブルに経路が残っていたら復活
#ifdef DEBUG
                        fprintf(stderr, "MULTIROUTE: %s: %d's multiple routing table is not empty, salvaged at %f.\n",
                                        __FUNCTION__, index, CURRENT_TIME);
                        //rt->rt_m_dump();
#endif // DEBUG
                        rt->rt_hops = rt->rt_m_hops[0];
                        rt->rt_nexthop = rt->rt_m_nexthop[0];
                }
#endif // AODV_MULTIROUTE

                // Not sure whether this is the right thing to do
                Packet *pkt;
                while((pkt = ifqueue->filter(ih->saddr()))) {
                        drop(pkt, DROP_RTR_MAC_CALLBACK);
                }

                // if precursor list non-empty add to RERR and delete the precursor list
                if (!rt->pc_empty()) {
                        nre->unreachable_dst[nre->DestCount] = rt->rt_dst;
                        nre->unreachable_dst_seqno[nre->DestCount] = rt->rt_seqno;
                        nre->DestCount += 1;
                        rt->pc_delete();
                }
        }
 }

 if (nre->DestCount > 0) {
        // 削除したエントリがあったので RERR を転送 (BROADCAST)
#ifdef DEBUG
        fprintf(stderr, "%s(%f): %d\t sending RERR...\n", __FUNCTION__, CURRENT_TIME, index);
#endif // DEBUG
        sendError(rerr);
 } else {
        Packet::free(rerr);
 }

 // 受信した RERR は用済み
 Packet::free(p);
}


/*
   Packet Transmission Routines
*/

void
AODV::forward(aodv_rt_entry *rt, Packet *p, double delay) {
struct hdr_cmn *ch = HDR_CMN(p);
```

```
struct hdr_ip *ih = HDR_IP(p);

 if(ih->ttl_ == 0) {

#ifdef DEBUG
  fprintf(stderr, "%s: calling drop()\n", __PRETTY_FUNCTION__);
#endif // DEBUG

  drop(p, DROP_RTR_TTL);
  return;
 }

 if (ch->ptype() != PT_AODV && ch->direction() == hdr_cmn::UP &&
        ((u_int32_t)ih->daddr() == IP_BROADCAST)
                || (ih->daddr() == here_.addr_)) {
        dmux_->recv(p,0);
        return;
 }

 if (rt) {
#ifndef AODV_MULTIROUTE // MULTIROUTE: rt_flags は RTF_WAIT である場合がある
   assert(rt->rt_flags == RTF_UP);
#endif // AODV_MULTIROUTE
   rt->rt_expire = CURRENT_TIME + ACTIVE_ROUTE_TIMEOUT;
#ifndef AODV_MULTIROUTE
   ch->next_hop_ = rt->rt_nexthop;
#else // AODV_MULTIROUTE
   if (ch->ptype() != PT_AODV && ih->saddr() == here_.addr_) {
     // 自分が送信したパケットの場合
     // XXX: データパケット送信先の変更はここ

     assert(rt->rt_flags == RTF_UP);
     assert(rt->rt_routes != 0);

/*
 * Application/UserData が組み込まれているとき
 */
#ifdef ns_userdata_h

        // UserData で割り当てた経路をパケットから取り出す
        int userpath = ((UserData*) p->userdata())->path();
        //fprintf(stderr, "MULTIROUTE: %s: userdata->path() = %d\n", __FUNCTION__, userpath);
        // 現在の経路数を元に経路割当を決め直す
        if (userpath >= rt->rt_routes) {
                // 割り当てた経路 No が保持経路数より多い場合は、カウンタの値を使う
                userpath = rt->rt_counter;
        }

#ifdef DEBUG
     fprintf(stderr, "MULTIROUTE: %s: %d sent data packet to %d via %d(#%d) at %f\n",
                 __FUNCTION__, index, rt->rt_dst,
                 rt->rt_m_nexthop[userpath], userpath,
                 CURRENT_TIME);
        //rt->rt_m_dump();
#endif // DEBUG
        ch->next_hop_ = rt->rt_m_nexthop[userpath];
```

```
/*
 * Application/UserData が組み込まれていないとき
 */
#else // ns_userdata_h

#ifdef DEBUG
        fprintf(stderr, "MULTIROUTE: %s: %d sent data packet to %d via %d(#%d) at %f\n",
                __FUNCTION__, index, rt->rt_dst,
                rt->rt_m_nexthop[rt->rt_counter], rt->rt_counter,
                CURRENT_TIME);
        rt->rt_m_dump();
#endif // DEBUG
        ch->next_hop_ = rt->rt_m_nexthop[rt->rt_counter];

#endif // ns_userdata_h

        rt->rt_m_rotate();
 } else {
        // その他のパケット：通常どおり
#ifdef DEBUG
     if (ch->ptype() != PT_AODV) {
        fprintf(stderr, "MULTIROUTE: %s: %d forwards data packet to %d via %d at %f\n",
                 __FUNCTION__, index, rt->rt_dst,
                 rt->rt_nexthop,
                 CURRENT_TIME);
     }
#endif // DEBUG
     ch->next_hop_ = rt->rt_nexthop;
   }
#endif // AODV_MULTIROUTE
   ch->addr_type() = NS_AF_INET;
   ch->direction() = hdr_cmn::DOWN;       //important: change the packet's direction
 }
 else { // if it is a broadcast packet
   // assert(ch->ptype() == PT_AODV); // maybe a diff pkt type like gaf
   assert(ih->daddr() == (nsaddr_t) IP_BROADCAST);
   ch->addr_type() = NS_AF_NONE;
   ch->direction() = hdr_cmn::DOWN;       //important: change the packet's direction
 }

if (ih->daddr() == (nsaddr_t) IP_BROADCAST) {
 // If it is a broadcast packet
   assert(rt == 0);
   /*
    *  Jitter the sending of broadcast packets by 10ms
    */
   Scheduler::instance().schedule(target_, p,
                                  0.01 * Random::uniform());
 }
 else { // Not a broadcast packet
   if(delay > 0.0) {
     Scheduler::instance().schedule(target_, p, delay);
   }
   else {
   // Not a broadcast packet, no delay, send immediately
     Scheduler::instance().schedule(target_, p, 0.);
   }
```

```
 }

}


void
AODV::sendRequest(nsaddr_t dst) {
// Allocate a RREQ packet
Packet *p = Packet::alloc();
struct hdr_cmn *ch = HDR_CMN(p);
struct hdr_ip *ih = HDR_IP(p);
struct hdr_aodv_request *rq = HDR_AODV_REQUEST(p);
aodv_rt_entry *rt = rtable.rt_lookup(dst);

 assert(rt);

 /*
  *  Rate limit sending of Route Requests. We are very conservative
  *  about sending out route requests.
  */

 if (rt->rt_flags == RTF_UP) {
   assert(rt->rt_hops != INFINITY2);
   Packet::free((Packet *)p);
   return;
 }

 if (rt->rt_req_timeout > CURRENT_TIME) {
   Packet::free((Packet *)p);
   return;
 }

 // rt_req_cnt is the no. of times we did network-wide broadcast
 // RREQ_RETRIES is the maximum number we will allow before
 // going to a long timeout.

 if (rt->rt_req_cnt > RREQ_RETRIES) {
   rt->rt_req_timeout = CURRENT_TIME + MAX_RREQ_TIMEOUT;
   rt->rt_req_cnt = 0;
 Packet *buf_pkt;
   while ((buf_pkt = rqueue.deque(rt->rt_dst))) {
       drop(buf_pkt, DROP_RTR_NO_ROUTE);
   }
   Packet::free((Packet *)p);
   return;
 }

//#ifdef DEBUG
//   fprintf(stderr, "(%2d) - %2d sending Route Request, dst: %d\n",
//                    ++route_request, index, rt->rt_dst);
//#endif // DEBUG

 // Determine the TTL to be used this time.
 // Dynamic TTL evaluation - SRD

 rt->rt_req_last_ttl = max(rt->rt_req_last_ttl,rt->rt_last_hop_count);
```

```
if (0 == rt->rt_req_last_ttl) {
// first time query broadcast
  ih->ttl_ = TTL_START;
}
else {
// Expanding ring search.
  if (rt->rt_req_last_ttl < TTL_THRESHOLD)
    ih->ttl_ = rt->rt_req_last_ttl + TTL_INCREMENT;
  else {
  // network-wide broadcast
    ih->ttl_ = NETWORK_DIAMETER;
    rt->rt_req_cnt += 1;
  }
}

// remember the TTL used  for the next time
rt->rt_req_last_ttl = ih->ttl_;

// PerHopTime is the roundtrip time per hop for route requests.
// The factor 2.0 is just to be safe .. SRD 5/22/99
// Also note that we are making timeouts to be larger if we have
// done network wide broadcast before.

rt->rt_req_timeout = 2.0 * (double) ih->ttl_ * PerHopTime(rt);
if (rt->rt_req_cnt > 0)
  rt->rt_req_timeout *= rt->rt_req_cnt;
rt->rt_req_timeout += CURRENT_TIME;

// Don't let the timeout to be too large, however .. SRD 6/8/99
if (rt->rt_req_timeout > CURRENT_TIME + MAX_RREQ_TIMEOUT)
  rt->rt_req_timeout = CURRENT_TIME + MAX_RREQ_TIMEOUT;
rt->rt_expire = 0;

//#ifdef DEBUG
// fprintf(stderr, "(%2d) - %2d sending Route Request, dst: %d, tout %f ms\n",
//               ++route_request,
//               index, rt->rt_dst,
//               rt->rt_req_timeout - CURRENT_TIME);
//#endif        // DEBUG


// Fill out the RREQ packet
// ch->uid() = 0;
ch->ptype() = PT_AODV;
ch->size() = IP_HDR_LEN + rq->size();
ch->iface() = -2;
ch->error() = 0;
ch->addr_type() = NS_AF_NONE;
ch->prev_hop_ = index;          // AODV hack

ih->saddr() = index;
ih->daddr() = IP_BROADCAST;
ih->sport() = RT_PORT;
ih->dport() = RT_PORT;

// Fill up some more fields.
rq->rq_type = AODVTYPE_RREQ;
```

```
 rq->rq_hop_count = 1;
 rq->rq_bcast_id = bid++;
 rq->rq_dst = dst;
 rq->rq_dst_seqno = (rt ? rt->rt_seqno : 0);
 rq->rq_src = index;
 seqno += 2;
 assert ((seqno%2) == 0);
 rq->rq_src_seqno = seqno;
 rq->rq_timestamp = CURRENT_TIME;

#ifdef DEBUG
        fprintf(stderr, "AODV %d %f s RREQ #%d to %d tout at %f\n",
                index, CURRENT_TIME, rq->rq_bcast_id, rt->rt_dst, rt->rt_req_timeout);
#endif  // DEBUG

 Scheduler::instance().schedule(target_, p, 0.);

}

void
AODV::sendReply(nsaddr_t ipdst, u_int32_t hop_count, nsaddr_t rpdst,
                u_int32_t rpseq, u_int32_t lifetime, double timestamp) {
Packet *p = Packet::alloc();
struct hdr_cmn *ch = HDR_CMN(p);
struct hdr_ip *ih = HDR_IP(p);
struct hdr_aodv_reply *rp = HDR_AODV_REPLY(p);
aodv_rt_entry *rt = rtable.rt_lookup(ipdst);

//#ifdef DEBUG
//fprintf(stderr, "sending Reply from %d at %.2f\n", index, Scheduler::instance().clock());
//#endif // DEBUG
 assert(rt);

 rp->rp_type = AODVTYPE_RREP;
 //rp->rp_flags = 0x00;
 rp->rp_hop_count = hop_count;
 rp->rp_dst = rpdst;
 rp->rp_dst_seqno = rpseq;
 rp->rp_src = index;
 rp->rp_lifetime = lifetime;
 rp->rp_timestamp = timestamp;

 // ch->uid() = 0;
 ch->ptype() = PT_AODV;
 ch->size() = IP_HDR_LEN + rp->size();
 ch->iface() = -2;
 ch->error() = 0;
 ch->addr_type() = NS_AF_INET;
#ifndef AODV_MULTIROUTE
 ch->next_hop_ = rt->rt_nexthop;
#ifdef DEBUG
 fprintf(stderr, "AODV %d %f s RREP from %d via %d for RREQ %f\n",
                index, CURRENT_TIME, rpdst, ch->next_hop_, timestamp);
#endif // DEBUG
#else // AODV_MULTIROUTE
 if(rpdst == index) {
        // 宛先ノード：最後に追加されたルートへ返信
```

```
        ch->next_hop_ = rt->rt_m_nexthop[rt->rt_routes - 1];
#ifdef DEBUG
        fprintf(stderr, "AODV %d %f s RREP from %d via %d (route #%d) for RREQ %f\n",
                        index, CURRENT_TIME, rpdst, ch->next_hop_, rt->rt_routes - 1, timestamp);
        //rt->rt_m_dump();
#endif // DEBUG
 } else {
        // 中継ノード：通常どおり
        ch->next_hop_ = rt->rt_nexthop;
#ifdef DEBUG
        fprintf(stderr, "AODV %d %f f RREP from %d via %d for RREQ %f\n",
                        index, CURRENT_TIME, rpdst, ch->next_hop_, timestamp);
        //rt->rt_m_dump();
#endif // DEBUG
 }
#endif // AODV_MULTIROUTE
 ch->prev_hop_ = index;          // AODV hack
 ch->direction() = hdr_cmn::DOWN;

 ih->saddr() = index;
 ih->daddr() = ipdst;
 ih->sport() = RT_PORT;
 ih->dport() = RT_PORT;
 ih->ttl_ = NETWORK_DIAMETER;

 Scheduler::instance().schedule(target_, p, 0.0);

}

void
AODV::sendError(Packet *p, bool jitter) {
struct hdr_cmn *ch = HDR_CMN(p);
struct hdr_ip *ih = HDR_IP(p);
struct hdr_aodv_error *re = HDR_AODV_ERROR(p);

#ifdef ERROR
//fprintf(stderr, "sending Error from %d at %.2f\n", index, Scheduler::instance().clock());
 fprintf(stderr, "AODV %d %f s RERR\n",
                index, CURRENT_TIME);
#endif // DEBUG

 re->re_type = AODVTYPE_RERR;
 //re->reserved[0] = 0x00; re->reserved[1] = 0x00;
 // DestCount and list of unreachable destinations are already filled

 // ch->uid() = 0;
 ch->ptype() = PT_AODV;
 ch->size() = IP_HDR_LEN + re->size();
 ch->iface() = -2;
 ch->error() = 0;
 ch->addr_type() = NS_AF_NONE;
 ch->next_hop_ = 0;
 ch->prev_hop_ = index;          // AODV hack
 ch->direction() = hdr_cmn::DOWN;       //important: change the packet's direction

 ih->saddr() = index;
 ih->daddr() = IP_BROADCAST;
```

```
 ih->sport() = RT_PORT;
 ih->dport() = RT_PORT;
 ih->ttl_ = 1;

 // Do we need any jitter? Yes
 if (jitter)
        Scheduler::instance().schedule(target_, p, 0.01*Random::uniform());
 else
        Scheduler::instance().schedule(target_, p, 0.0);

}


/*
   Neighbor Management Functions
*/

void
AODV::sendHello() {
Packet *p = Packet::alloc();
struct hdr_cmn *ch = HDR_CMN(p);
struct hdr_ip *ih = HDR_IP(p);
struct hdr_aodv_reply *rh = HDR_AODV_REPLY(p);

#ifdef DEBUG
//fprintf(stderr, "sending Hello from %d at %.2f\n", index, Scheduler::instance().clock());
        fprintf(stderr, "AODV %d %f s HELO\n",
                index, CURRENT_TIME);
#endif // DEBUG

 rh->rp_type = AODVTYPE_HELLO;
 //rh->rp_flags = 0x00;
 rh->rp_hop_count = 1;
 rh->rp_dst = index;
 rh->rp_dst_seqno = seqno;
 rh->rp_lifetime = (1 + ALLOWED_HELLO_LOSS) * HELLO_INTERVAL;

 // ch->uid() = 0;
 ch->ptype() = PT_AODV;
 ch->size() = IP_HDR_LEN + rh->size();
 ch->iface() = -2;
 ch->error() = 0;
 ch->addr_type() = NS_AF_NONE;
 ch->prev_hop_ = index;          // AODV hack

 ih->saddr() = index;
 ih->daddr() = IP_BROADCAST;
 ih->sport() = RT_PORT;
 ih->dport() = RT_PORT;
 ih->ttl_ = 1;

 Scheduler::instance().schedule(target_, p, 0.0);
}


void
AODV::recvHello(Packet *p) {
```

```
//struct hdr_ip *ih = HDR_IP(p);
struct hdr_aodv_reply *rp = HDR_AODV_REPLY(p);
AODV_Neighbor *nb;

#ifdef DEBUG
        fprintf(stderr, "AODV %d %f r HELO from %d\n",
                index, CURRENT_TIME, rp->rp_dst);
#endif // DEBUG

 nb = nb_lookup(rp->rp_dst);
 if(nb == 0) {
   nb_insert(rp->rp_dst);
 }
 else {
   nb->nb_expire = CURRENT_TIME +
                   (1.5 * ALLOWED_HELLO_LOSS * HELLO_INTERVAL);
 }

 Packet::free(p);
}

void
AODV::nb_insert(nsaddr_t id) {
AODV_Neighbor *nb = new AODV_Neighbor(id);

 assert(nb);
 nb->nb_expire = CURRENT_TIME +
                (1.5 * ALLOWED_HELLO_LOSS * HELLO_INTERVAL);
 LIST_INSERT_HEAD(&nbhead, nb, nb_link);
 seqno += 2;             // set of neighbors changed
 assert ((seqno%2) == 0);
}


AODV_Neighbor*
AODV::nb_lookup(nsaddr_t id) {
AODV_Neighbor *nb = nbhead.lh_first;

 for(; nb; nb = nb->nb_link.le_next) {
   if(nb->nb_addr == id) break;
 }
 return nb;
}


/*
 * Called when we receive *explicit* notification that a Neighbor
 * is no longer reachable.
 */
void
AODV::nb_delete(nsaddr_t id) {
AODV_Neighbor *nb = nbhead.lh_first;

 log_link_del(id);
 seqno += 2;     // Set of neighbors changed
 assert ((seqno%2) == 0);
```

```
 for(; nb; nb = nb->nb_link.le_next) {
   if(nb->nb_addr == id) {
     LIST_REMOVE(nb,nb_link);
     delete nb;
     break;
   }
 }

 handle_link_failure(id);

}


/*
 * Purges all timed-out Neighbor Entries - runs every
 * HELLO_INTERVAL * 1.5 seconds.
 */
void
AODV::nb_purge() {
AODV_Neighbor *nb = nbhead.lh_first;
AODV_Neighbor *nbn;
double now = CURRENT_TIME;

 for(; nb; nb = nbn) {
   nbn = nb->nb_link.le_next;
   if(nb->nb_expire <= now) {
     nb_delete(nb->nb_addr);
   }
 }

}

/*
 * Application/UserData 用インターフェース
 */
#ifdef AODV_MULTIROUTE
// ルート数を返す
int AODV::rt_count(nsaddr_t dst) {

        aodv_rt_entry *rt;

        rt = rtable.rt_lookup(dst);
        if (rt == 0) {
                // 該当ルートが無い
                return 0;
        } else {
                // 該当ルートがあった
                return rt->rt_routes;
        }

}
// 各ルートのホップ数を返す
int AODV::rt_hops(nsaddr_t dst, int path_id) {

        aodv_rt_entry *rt;

        rt = rtable.rt_lookup(dst);
```

```
        if (rt == 0) {
                // 該当エントリが無い
                return 0;
        } else {
                // 該当エントリがあった
                if (path_id < rt->rt_routes) {
                        // 該当経路がある
                        return rt->rt_m_hops[path_id];
                } else {
                        // 該当経路がない
                        return 0;
                }
        }
}
#endif // AODV_MULTIROUTE
#ifdef AODV_USERDATA_CONNECT
// 強制ルート構築
void AODV::force_rt_request(nsaddr_t dst, UserDataApp *callback) {

        aodv_rt_entry *rt;

#ifdef DEBUG
        fprintf(stderr, "UserData called AODV::force_rt_request for %d\n", dst);
#endif // DEBUG

        userdata_callback_ = callback;
        userdata_dst = dst;
        rt_request_status_ = 1;

        rt = rtable.rt_lookup(dst);
        if(rt == 0) {
                rt = rtable.rt_add(dst);
        }
        sendRequest(rt->rt_dst);

}
// UserData への通知
void AODV::userdata_callback(nsaddr_t dst) {

        if (dst == userdata_dst) {
                rt_request_status_ = 2;
                userdata_callback_->aodv_callback();
        }

}
// ルート構築失敗
void AODV::userdata_fail_callback(nsaddr_t dst) {

        if (dst == userdata_dst) {
                // もう一回 force_rt_request を呼ばせる
                rt_request_status_ = 0;
        }

}
#endif // AODV_USERDATA_CONNECT
```

A.2.4 aodv/aodv_rtable.h

```
/*
Copyright (c) 1997, 1998 Carnegie Mellon University.  All Rights
Reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice,
this list of conditions and the following disclaimer in the documentation
and/or other materials provided with the distribution.
3. The name of the author may not be used to endorse or promote products
derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE AUTHOR ``AS IS'' AND ANY EXPRESS OR
IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES
OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED.
IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS;
OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR
OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

The AODV code developed by the CMU/MONARCH group was optimized and tuned by Samir Das and Mahesh Marina,
University of Cincinnati. The work was partially done in Sun Microsystems.
*/

/*
 * AODV マルチルート拡張（仮実装）
 * Last Modified: 2008/10/22 23:53:10
 */

#ifndef __aodv_rtable_h__
#define __aodv_rtable_h__

#include <assert.h>
#include <sys/types.h>
#include <config.h>
#include <lib/bsd-list.h>
#include <scheduler.h>

#define CURRENT_TIME    Scheduler::instance().clock()
#define INFINITY2        0xff

/*
   AODV Neighbor Cache Entry
*/
class AODV_Neighbor {
        friend class AODV;
        friend class aodv_rt_entry;
 public:
        AODV_Neighbor(u_int32_t a) { nb_addr = a; }
```

```
 protected:
        LIST_ENTRY(AODV_Neighbor) nb_link;
        nsaddr_t        nb_addr;
        double          nb_expire;      // ALLOWED_HELLO_LOSS * HELLO_INTERVAL
};

LIST_HEAD(aodv_ncache, AODV_Neighbor);

/*
   AODV Precursor list data structure
*/
class AODV_Precursor {
        friend class AODV;
        friend class aodv_rt_entry;
 public:
        AODV_Precursor(u_int32_t a) { pc_addr = a; }

 protected:
        LIST_ENTRY(AODV_Precursor) pc_link;
        nsaddr_t        pc_addr;        // precursor address
};

LIST_HEAD(aodv_precursors, AODV_Precursor);


/*
  Route Table Entry
*/

class aodv_rt_entry {
        friend class aodv_rtable;
        friend class AODV;
        friend class LocalRepairTimer;
 public:
        aodv_rt_entry();
        ~aodv_rt_entry();

        void            nb_insert(nsaddr_t id);
        AODV_Neighbor*  nb_lookup(nsaddr_t id);

        void            pc_insert(nsaddr_t id);
        AODV_Precursor* pc_lookup(nsaddr_t id);
        void            pc_delete(nsaddr_t id);
        void            pc_delete(void);
        bool            pc_empty(void);

        double          rt_req_timeout;         // when I can send another req
        u_int8_t        rt_req_cnt;             // number of route requests

 protected:
        LIST_ENTRY(aodv_rt_entry) rt_link;

        nsaddr_t        rt_dst;                 // 送信先アドレス
        u_int32_t       rt_seqno;               // シーケンス番号
        /* u_int8_t     rt_interface; */
        u_int16_t       rt_hops;                // ホップ数 (hop count)
```

```
        int             rt_last_hop_count;      // last valid hop count
        nsaddr_t        rt_nexthop;             // 次ホップの IP アドレス (next hop IP address)
        /* list of precursors */
        aodv_precursors rt_pclist;
        double          rt_expire;              // エントリ有効期限 (when entry expires)
        u_int8_t        rt_flags;               // ルートの状態 (フラグ)

#define RTF_DOWN 0              // ルート無効 (DOWN)
#define RTF_UP 1                // ルート有効 (UP)
#define RTF_IN_REPAIR 2         // ルート修復中

#ifdef AODV_MULTIROUTE
        /*
         * マルチルート用追加フラグ
         */
#define RTF_WAIT 3              // 複数ルート構築待ち状態 (WAIT) データパケット以外に対しては RTF_UP と同等

        /*
         * マルチルート拡張関連メンバ変数・関数
         */
#define ROUTE_COUNT 2           // 使用するルート数
//#define MULTIROUTE_TIMEOUT 5  // 複数ルート構築を待つ期限
        /* マルチルート用ルーティングテーブル */
        u_int16_t       rt_m_hops[ROUTE_COUNT];         // ホップ数 (hop count)
        nsaddr_t        rt_m_nexthop[ROUTE_COUNT];      // 次ホップの IP アドレス (next hop IP address)
        /* ルート切替のための変数 */
        int             rt_routes;                      // 保持しているルートの数
        int             rt_counter;                     // ルート選択カウンタ
        double          rt_m_create;                    // エントリが生成された時間
        /* マルチルート用ルーティングテーブルの操作を行うメンバ関数 */
        bool            rt_m_add(nsaddr_t nexthop, u_int16_t hops);     // ルート追加
        bool            rt_m_delete(nsaddr_t nexthop);  // ルート削除
        void            rt_m_rotate(void);              // ルート切替
        int             rt_m_lookup(nsaddr_t nexthop);  // nexthop からルートを lookup
        void            rt_m_dump();                    // デバッグ用
#endif // AODV_MULTIROUTE

        /*
         *  Must receive 4 errors within 3 seconds in order to mark
         *  the route down.
        u_int8_t        rt_errors;      // error count
        double          rt_error_time;
#define MAX_RT_ERROR            4       // errors
#define MAX_RT_ERROR_TIME       3       // seconds
         */

#define MAX_HISTORY     3
        double          rt_disc_latency[MAX_HISTORY];
        char            hist_indx;
        int             rt_req_last_ttl;        // last ttl value used
        // last few route discovery latencies
        // double               rt_length [MAX_HISTORY];
        // last few route lengths

        /*
         * a list of neighbors that are using this route.
         */
```

```
        aodv_ncache          rt_nblist;
};


/*
  The Routing Table
*/

class aodv_rtable {
 public:
        aodv_rtable() { LIST_INIT(&rthead); }

        aodv_rt_entry*       head() { return rthead.lh_first; }

        aodv_rt_entry*       rt_add(nsaddr_t id);
        void                 rt_delete(nsaddr_t id);
        aodv_rt_entry*       rt_lookup(nsaddr_t id);

 private:
        LIST_HEAD(aodv_rthead, aodv_rt_entry) rthead;
};

#endif /* _aodv__rtable_h__ */

/* vim: set fenc=utf-8 ff=unix */
```

A.2.5 aodv/aodv_rtable.cc

```
/*
Copyright (c) 1997, 1998 Carnegie Mellon University.  All Rights
Reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice,
this list of conditions and the following disclaimer in the documentation
and/or other materials provided with the distribution.
3. The name of the author may not be used to endorse or promote products
derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE AUTHOR ``AS IS'' AND ANY EXPRESS OR
IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES
OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED.
IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS;
OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR
OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

The AODV code developed by the CMU/MONARCH group was optimized and tuned by Samir Das and Mahesh Marina,
University of Cincinnati. The work was partially done in Sun Microsystems.
*/
```

```
/*
 * AODV マルチルート拡張（仮実装）
 * Last Modified: 2008/10/30 18:45:20
 */

#include <aodv/aodv_rtable.h>
//#include <cmu/aodv/aodv.h>

/*
  The Routing Table
*/

aodv_rt_entry::aodv_rt_entry()
{
int i;

 rt_req_timeout = 0.0;
 rt_req_cnt = 0;

 rt_dst = 0;
 rt_seqno = 0;
 rt_hops = rt_last_hop_count = INFINITY2;
 rt_nexthop = 0;
 LIST_INIT(&rt_pclist);
 rt_expire = 0.0;
 rt_flags = RTF_DOWN;

 /*
 rt_errors = 0;
 rt_error_time = 0.0;
 */

#ifdef AODV_MULTIROUTE
        for(i=0; i < ROUTE_COUNT; i++) {
                rt_m_hops[i] = INFINITY2;
                rt_m_nexthop[i] = 0;
        }
        rt_routes = 0;
        rt_counter = 0;
        rt_m_create = 0.0;
#endif // AODV_MULTIROUTE

 for (i=0; i < MAX_HISTORY; i++) {
   rt_disc_latency[i] = 0.0;
 }
 hist_indx = 0;
 rt_req_last_ttl = 0;

 LIST_INIT(&rt_nblist);

}


aodv_rt_entry::~aodv_rt_entry()
{
AODV_Neighbor *nb;
```

```
 while((nb = rt_nblist.lh_first)) {
   LIST_REMOVE(nb, nb_link);
   delete nb;
 }

AODV_Precursor *pc;

 while((pc = rt_pclist.lh_first)) {
   LIST_REMOVE(pc, pc_link);
   delete pc;
 }

}


void
aodv_rt_entry::nb_insert(nsaddr_t id)
{
AODV_Neighbor *nb = new AODV_Neighbor(id);

 assert(nb);
 nb->nb_expire = 0;
 LIST_INSERT_HEAD(&rt_nblist, nb, nb_link);

}


AODV_Neighbor*
aodv_rt_entry::nb_lookup(nsaddr_t id)
{
AODV_Neighbor *nb = rt_nblist.lh_first;

 for(; nb; nb = nb->nb_link.le_next) {
   if(nb->nb_addr == id)
     break;
 }
 return nb;

}


void
aodv_rt_entry::pc_insert(nsaddr_t id)
{
        if (pc_lookup(id) == NULL) {
        AODV_Precursor *pc = new AODV_Precursor(id);

                assert(pc);
                LIST_INSERT_HEAD(&rt_pclist, pc, pc_link);
        }
}


AODV_Precursor*
aodv_rt_entry::pc_lookup(nsaddr_t id)
{
```

```
AODV_Precursor *pc = rt_pclist.lh_first;

 for(; pc; pc = pc->pc_link.le_next) {
   if(pc->pc_addr == id)
        return pc;
 }
 return NULL;

}

void
aodv_rt_entry::pc_delete(nsaddr_t id) {
AODV_Precursor *pc = rt_pclist.lh_first;

 for(; pc; pc = pc->pc_link.le_next) {
   if(pc->pc_addr == id) {
     LIST_REMOVE(pc,pc_link);
     delete pc;
     break;
   }
 }

}

void
aodv_rt_entry::pc_delete(void) {
AODV_Precursor *pc;

 while((pc = rt_pclist.lh_first)) {
   LIST_REMOVE(pc, pc_link);
   delete pc;
 }
}

bool
aodv_rt_entry::pc_empty(void) {
AODV_Precursor *pc;

 if ((pc = rt_pclist.lh_first)) return false;
 else return true;
}

/*
  The Routing Table
*/

aodv_rt_entry*
aodv_rtable::rt_lookup(nsaddr_t id)
{
aodv_rt_entry *rt = rthead.lh_first;

 for(; rt; rt = rt->rt_link.le_next) {
   if(rt->rt_dst == id)
     break;
 }
 return rt;
```

```
}

void
aodv_rtable::rt_delete(nsaddr_t id)
{
aodv_rt_entry *rt = rt_lookup(id);

 if(rt) {
   LIST_REMOVE(rt, rt_link);
   delete rt;
 }

}

aodv_rt_entry*
aodv_rtable::rt_add(nsaddr_t id)
{
aodv_rt_entry *rt;

 assert(rt_lookup(id) == 0);
 rt = new aodv_rt_entry;
 assert(rt);
 rt->rt_dst = id;
 LIST_INSERT_HEAD(&rthead, rt, rt_link);
 return rt;
}

#ifdef AODV_MULTIROUTE
/*
 * マルチルート用ルーティングテーブルへのルート追加
 */
bool
aodv_rt_entry::rt_m_add(nsaddr_t nexthop, u_int16_t hops)
{
        if ( rt_routes >= ROUTE_COUNT || rt_m_lookup(nexthop) != -1 ) {
                /*
                 *  1) ルーティングテーブルが満杯
                 *  2) 同じルートが既にルーティングテーブル内にある
                 *  場合には追加せずに false を返す。
                 */
#ifdef DEBUG
                if(rt_routes >= ROUTE_COUNT) {
                        fprintf(stderr, "MULTIROUTE: %s: %d full at %f\n",
                                __FUNCTION__, index, CURRENT_TIME);
                }
                if(rt_m_lookup(nexthop) != -1) {
                        fprintf(stderr, "MULTIROUTE: %s: %d duplicate at %f\n",
                                __FUNCTION__, index, CURRENT_TIME);
                }
#endif // DEBUG
                return false;
        } else {
                rt_routes++;
                rt_m_nexthop[rt_routes - 1] = nexthop;
                rt_m_hops[rt_routes - 1] = hops;
                return true;
        }
```

```
}

/*
 * マルチルート用ルーティングテーブルからのルート削除
 */
bool
aodv_rt_entry::rt_m_delete(nsaddr_t nexthop)
{
        int i, m_rt = -1;

        m_rt = rt_m_lookup(nexthop);    // nexthop からルート番号を調べる
        if(m_rt == -1) return false;    // 削除対象が見つからなかった

        /* 対象エントリの削除 */
        for(i=0;i<ROUTE_COUNT;i++) {
                // 対象よりも後ろのエントリで上書き
                if(i>m_rt) {
                        rt_m_hops[i-1] = rt_m_hops[i];
                        rt_m_nexthop[i-1] = rt_m_nexthop[i];
                }
        }
        rt_routes--;
        // rt_counter が存在しないルート番号を指していたら rt_routes の値にする
        if(rt_counter > rt_routes) rt_counter = rt_routes;

        return true;

}

/*
 * ルートの切り替え（ラウンドロビン用）
 */
void
aodv_rt_entry::rt_m_rotate(void)
{
        rt_counter++;
        if(rt_counter >= rt_routes) {
                rt_counter = 0;
        }
}

/*
 * マルチルート用ルーティングテーブルの lookup（nexthop →ルート番号）
 */
int
aodv_rt_entry::rt_m_lookup(nsaddr_t nexthop)
{
        int i, m_rt = -1;       // ルートが見つからなければ -1 を返す

        for(i=0; i<rt_routes; i++) {
                if(rt_m_nexthop[i] == nexthop) {
                        m_rt = i;
                }
        }

        return m_rt;
}
```

```
/*
 * DEBUG: マルチルート用ルーティングテーブルの中身を表示
 */
void
aodv_rt_entry::rt_m_dump()
{
        int i;

        for(i=0; i<rt_routes; i++) {
                fprintf(stderr, "MULTIROUTE: %s: [#%d] %d -- %d hops\n",
                         __FUNCTION__, i, rt_m_nexthop[i], rt_m_hops[i]);
        }
        fprintf(stderr, "MULTIROUTE: %s: number of routes is %d, current route is #%d, state is ",
                         __FUNCTION__, rt_routes, rt_counter);
        switch (rt_flags) {
                case RTF_UP:
                        fprintf(stderr, "RTF_UP\n");
                        break;
                case RTF_DOWN:
                        fprintf(stderr, "RTF_DOWN\n");
                        break;
                case RTF_IN_REPAIR:
                        fprintf(stderr, "RTF_IN_REPAIR\n");
                        break;
                case RTF_WAIT:
                        fprintf(stderr, "RTF_WAIT\n");
                        break;
                detault:
                        fprintf(stderr, "unknown(%d)\n", rt_flags);
                        break;
        }
}
#endif //AODV_MULTIROUTE

/* vim: set fenc=utf-8 ff=unix */
```

A.2.6 userdata/userdata.h

```
/* vim: set fenc=utf-8 ff=unix */

/*
 * Application/UserData : ユーザ定義データの送受信
 *
 * Wada Laboratory, Shizuoka University
 * Takahiro Sugimoto    <taka_s@keyaki.kokage.cc>
 *
 * Last Modified: 2008/12/19 17:30:14
 */

#ifndef ns_userdata_h
#define ns_userdata_h

#include <stdarg.h>
#include "app.h"
#include "userdata_file.h"
```

```
#define CURRENT_TIME Scheduler::instance().clock()

// デバッグメッセージ出力制御
#define UD_DEBUG

// 送信間隔の初期値 … set-interval で変更可能
#define INTERVAL 0.01

// 修正済 AODV エージェントと連携するかどうか
// ここではなく、コンパイルオプションでの設定を推奨します。
// ** マルチルート関連の連携機能を有効にする **
//#define AODV_MULTIROUTE
// ** ルート構築要求関連の連携機能を有効にする **
//#define AODV_USERDATA_CONNECT

// オブジェクトへのポインタをメンバ変数に入れるために先に宣言
class UserData;
class UserDataApp;

// AODV 関係
// AODV_MULTIROUTE/AODV_USERDATA_CONNECT のいずれかが define されていたら、
// USE_AODV を define する。
#ifdef AODV_MULTIROUTE
#define USE_AODV
#endif // AODV_MULTIROUTE
#ifdef AODV_USERDATA_CONNECT
#define USE_AODV
#endif // AODV_USERDATA_CONNECT

#ifdef USE_AODV
#include "../aodv/aodv.h"
#endif // USE_AODV

/* ファイル送信タイマー */
class UserDataTimer : public TimerHandler {
        /*
         * TimerHandler の仕様については common/timer-handler.{h,cc} を参照。
         *
         * マニュアルでは expire が virtual double expire() となっているが、
         * 開発に使用しているバージョン (2.32) では virtual void expire() と
         * なっていることに注意。Timer の reschedule には resched() を使用
         * する (timer-handler.h の 1.98-99 を参照のこと)。
         */
 public:
        UserDataTimer(UserDataApp *a) : TimerHandler() {
                a_ = a;
        }
 protected:
        virtual void expire(Event *e);
        UserDataApp *a_;
};

/* データ送受信アプリーケーションクラス */
class UserDataApp : public Application {
 public:
        // コンストラクタ/デストラクタ
        UserDataApp();
```

```
        ~UserDataApp();
        // 送信処理 (Timer から呼び出される)
        void send();
        // 変数取得
        int status() {
                // 送受信の状態
                return status_;
        }
        double delay() {
                // 送信周期
                return interval_;
        }
        Agent* agent() {
                return agent_;
        }
        AODV* ragent() {
                // ルーティングエージェントへのポインタ
                return ragent_;
        }
#ifdef AODV_USERDATA_CONNECT
        // AODV からのルート構築完了通知
        void aodv_callback();
#endif // AODV_USERDATA_CONNECT
        // UserDataApp 間通信 (see "class Process" in ns-process.{h,cc})
        virtual void process_data(int size, AppData* data);
        virtual void send_data(int size, AppData* data = 0) {
                // 相手先の process_data() を呼び出す。UserDataApp では使用しない。
                //if (target_) target_->process_data(size, data);
                abort();
        }
 protected:
        // Tcl インタプリタ
        virtual int command(int argc, const char*const* argv);
        // パケット受信
        //  - Application の子クラスなので必要。
        //  - 実際の受信処理は process_data が行う。
        void recv(Packet* pkt, Handler*);
 private:
        // 送受信状態変更
        void start();
        void stop();
        void wait();
        void restart();
        // 送信周期
        double interval_;
        // 送受信の状態
        int status_;
#define UD_S_NULL 0
#define UD_S_SEND 1
#define UD_S_WAIT 2
#define UD_S_STOP 3
        // パケット番号
        int packetID_;
        // アタッチされた UserDataFile オブジェクトへのポインタ
        UserDataFile* filein_;
        UserDataFile* fileout_;
        UserDataFile* filesorted_;
```

```
        // Timer
        UserDataTimer timer_;
#ifdef USE_AODV
        // アタッチされたルーティングエージェントへのポインタ
        AODV* ragent_;
#endif // USE_AODV
#ifdef AODV_MULTIROUTE
        // マルチルートへのパケット割り当て方式
        int multiroute_scheme_;
#define UD_ROUNDROBIN 0
#define UD_UNIFORM 1
#define UD_HOPWEIGHTED 2
#define UD_CLONE 3
#define UD_SHORTONLY 4
        // UD_ROUNDROBIN 用カウンタ
        int multiroute_rr_counter_;
#endif // AODV_MULTIROUTE
        // ログ関係
        UserDataLog* logout_;
        // 受信データ再構築用一時キャッシュ
        struct cache {
                struct cache* previous;
                unsigned char* cached_data;
                int id;
                int size;
                struct cache* next;
                void init(struct cache* prev) {
                        previous = prev;
                        cached_data = NULL;
                        id = 0;
                        size = 0;
                        next = NULL;
                }
                void setdata(unsigned char* data, int data_id, int data_size) {
                        cached_data = new unsigned char[data_size];
                        memcpy(cached_data, data, data_size);
                        id = data_id;
                        size = data_size;
                }
        };
        cache* cache_entrypoint;
        int cache_fillsize;
        bool cache_insert(unsigned char* data, int id, int size);
        void cache_flush();
        int flushed_flag;
};

/* ADU for UserDataApp */
class UserData : public PacketData {
        /*
         *  ADU (Application-level Data Unit) については、common/ns-process.{h,cc}
         *  および、マニュアル pp.341-343 "Web cache as an application" を参照。
         *  ここで継承している PacketData は common/packet.{h,cc} で、AppData の
         *  子クラスとして宣言されている。
         */
 public:
        // コンストラクタ: 新規生成時
```

```
        UserData(int sz) : PacketData(sz) {
                // 変数を初期化
                path_ = 0;
                packetID_ = 0;
        }
        // コンストラクタ: コピー時
        UserData(UserData& d) : PacketData(d) {
                // 変数もコピー
                path_ = d.path_;
                packetID_ = d.packetID_;
        }
        // 変数のやりとり
        virtual int path() { return path_; }
        void path(int id) { path_ = id; }
        virtual int packetID() { return packetID_; };
        void packetID(int id) { packetID_ = id; }

        // パケットサイズを返す
        // TODO: 追加した情報分のサイズを足す
        //virtual int size() const { return datalen_; }

        // コピー関数のオーバーライド
        virtual UserData* copy() { return new UserData(*this); }
 private:
        // 割当先ルート (AODV_MULTIROUTE 有効時に利用)
        int path_;
        // パケット番号
        int packetID_;
};

#endif // ns_userdata_h
```

A.2.7 userdata/userdata.cc

```
/* vim: set fenc=utf-8 ff=unix */

/*
 * Application/UserData : ユーザ定義データの送受信
 *
 * Wada Laboratory, Shizuoka University
 * Takahiro Sugimoto    <taka_s@keyaki.kokage.cc>
 *
 * Last Modified: 2009/01/21 19:36:40
 */

#include <string.h>
#include "packet.h"
#include "userdata.h"

#ifdef AODV_MULTIROUTE
#include "rng.h"
#endif // AODV_MULTIROUTE

/* Tcl Hooks */
static class UserDataClass : public TclClass {
 public:
        UserDataClass() : TclClass("Application/UserData") {}
```

```
        TclObject* create(int, const char*const*) {
                return (new UserDataApp);
        }
} class_userdata;

/*
 *  "UserDataTimer" Class
 *            -- パケット送信タイマー
 */
void UserDataTimer::expire(Event *e) {

        // agent に送信要求
        if (a_->status() != UD_S_STOP) {
                a_->send();
        }

        // 次の expire を schedule
        resched(a_->delay());

}

/*
 *  "UserDataApp" Class
 *            -- UserData アプリケーション
 */
/* コンストラクタ */
UserDataApp::UserDataApp() : timer_(this) {

        // 変数の初期化
        target_ = (UserDataApp*) NULL;
        status_ = UD_S_NULL;
        interval_ = INTERVAL;
        packetID_ = 0;
        filein_ = (UserDataFile*) NULL;
        fileout_ = (UserDataFile*) NULL;
        filesorted_ = (UserDataFile*) NULL;
        logout_ = (UserDataLog*) NULL;
        cache_entrypoint = NULL;
        flushed_flag = 0;
#ifdef AODV_MULTIROUTE
        ragent_ = 0;
        multiroute_scheme_ = UD_UNIFORM;
        multiroute_rr_counter_ = 0;
#endif // AODV_MULTIROUTE

}
/* デストラクタ */
UserDataApp::~UserDataApp() {

        // XXX: 作っても呼ばれないようだ

}

/* Tcl コマンドインタプリタ */
int UserDataApp::command(int argc, const char*const* argv) {

        /*
```

```
         * Tcl 側から UserDataApp に渡された引数はここで処理されます。
         *
         * コマンド一覧:
         *  set userdata [new Application/UserData]
         *    --- 新たな UserDataApp インスタンス $userdata を生成します。
         *        (※ C++ 側でこの処理を担当するのは Tcl Hooks)
         *  $userdata attach-agent <agent object>
         *    --- Agent に $userdata をアタッチします。
         *  $userdata attach-file-in <UserDataFile>
         *    --- $userdata に UserDataFile オブジェクト <UserDataFile> を入力先としてアタッチします。
         *  $userdata attach-file-out <UserDataFile>
         *    --- $userdata に UserDataFile オブジェクト <UserDataFile> を出力先としてアタッチします。
         *  $userdata attach-file-sorted <UserDataFile>
         *    --- $userdata に UserDataFile オブジェクト <UserDataFile> を並替済出力先としてアタッチします。
         *  $userdata attach-log <UserDataLog>
         *    --- $userdata に UserDataLog オブジェクト <UserDataLog> をログ出力先としてアタッチします。
         *  $userdata set-interval <interval>
         *    --- 送信間隔を <interval> に変更します。変更しない場合は初期値 INTERVAL の値が使われます。
         *  $userdata start
         *    --- 送信を開始/再開します。
         *  $userdata wait
         *    --- 送信を中断します。
         *  $userdata stop
         *    --- 送信を終了します。attach-file-sorted されている場合にはここで並べ替え済みデータの書き出しが行われま
す。
         *
         *  $userdata attach-ragent <routing agent object>
         *    --- $userdata にルーティングエージェントをアタッチします (AODV_MULTIROUTE 有効時のみ)
         *  $userdata set-multiroute-scheme {round-robin|uniform|hop-weighted|clone}
         *    --- 複数ルートへの割り当て方式を設定します。
         *         round-robin  : 各経路を番号順に繰り替えして使用します。
         *         uniform      : 各経路を等確率で選択して使用します。
         *         hop-weighted : 短い経路に高い確率でパケットが割り当てられるようにして使用します。
         *         clone        : 各経路に同一パケットをコピーして送信します。
         *         short-only   : 経路の中で最も短いものだけを選択して送信します。
         */

        Tcl& tcl = Tcl::instance();

        if (argc == 2) {        // 引数 1 個
                /* start: 送信開始 */
                if (strcmp(argv[1], "start") == 0) {
#ifdef UD_DEBUG
                        printf("%s: start\n", __FUNCTION__);
#endif // UD_DEBUG
                        if (status_ == UD_S_WAIT) {
                                restart();
                        } else {
                                start();
                        }
                        return (TCL_OK);
                }
                /* wait: 送信中断 */
                else if (strcmp(argv[1], "wait") == 0) {
#ifdef UD_DEBUG
                        printf("%s: wait\n", __FUNCTION__);
#endif // UD_DEBUG
```

```
                        wait();
                        return (TCL_OK);
                }
                /* stop: 送信終了 */
                else if (strcmp(argv[1], "stop") == 0) {
#ifdef UD_DEBUG
                        printf("%s: stop\n", __FUNCTION__);
#endif // UD_DEBUG
                        stop();
                        return (TCL_OK);
                }
        } else if (argc == 3) { // 引数 2 個
                /* attach-agent : エージェントにアタッチ */
                if (strcmp(argv[1], "attach-agent") == 0) {
                        agent_ = (Agent*) TclObject::lookup(argv[2]);
                        if (agent_ == 0) {
                                // 指定されたエージェントが見つからない
                                tcl.resultf("no such agent %s", argv[2]);
                                return(TCL_ERROR);
                        }
                        agent_->attachApp(this);
#ifdef UD_DEBUG
                        printf("%s: attach-agent %s: succeeded\n", __FUNCTION__, argv[2]);
#endif // UD_DEBUG
                        return (TCL_OK);
                }
                /* set-interval : 送信間隔の設定*/
                else if (strcmp(argv[1], "set-interval") == 0) {
                        interval_ = atof(argv[2]);
#ifdef UD_DEBUG
                        printf("%s: set-interval %f\n", __FUNCTION__, interval_);
#endif // UD_DEBUG

                        return (TCL_OK);
                }
#ifdef USE_AODV
                /* attach-ragent : ルーティングエージェントのアタッチ */
                else if (strcmp(argv[1], "attach-ragent") == 0) {
                        ragent_ = (AODV*) TclObject::lookup(argv[2]);
                        if (ragent_ == 0) {
                                tcl.resultf("no such routing agent %s", argv[2]);
                                return(TCL_ERROR);
                        }
#ifdef UD_DEBUG
                        printf("%s: attach-ragent %s: succeeded\n", __FUNCTION__, argv[2]);
#endif // UD_DEBUG
                        return (TCL_OK);
                }
#else // USE_AODV
                else if (strcmp(argv[1], "attach-ragent") == 0) {
                        tcl.resultf("%s: attach-ragent %s: ignoring (disabled by compiling options)\n", __FUNCTION__,
argv[2]);
                        return (TCL_OK); // 無視して処理を継続させる
                }
#endif // USE_AODV
                /* attach-file-in: 入力ファイルのアタッチ */
                else if (strcmp(argv[1], "attach-file-in") == 0) {
```

```
                        filein_ = (UserDataFile*) TclObject::lookup(argv[2]);
                        if (filein_ == 0) {
                                // 指定された UserDataFile オブジェクトが見つからない
                                tcl.resultf("no such object %s", argv[2]);
                                return(TCL_ERROR);
                        }
                        if (filein_->mode() != 1) {
                                // ファイルが read モードではない
                                tcl.resultf("object %s is not read mode", argv[2]);
                                return(TCL_ERROR);
                        }
#ifdef UD_DEBUG
                        printf("%s: attach-file-in %s: succeeded\n", __FUNCTION__, argv[2]);
#endif // UD_DEBUG
                        return (TCL_OK);
                }
                /* attach-file-out: 出力ファイルのアタッチ */
                else if (strcmp(argv[1], "attach-file-out") == 0) {
                        fileout_ = (UserDataFile*) TclObject::lookup(argv[2]);
                        if (fileout_ == 0) {
                                // 指定された UserDataFile オブジェクトが見つからない
                                tcl.resultf("no such object %s", argv[2]);
                                return(TCL_ERROR);
                        }
                        if (fileout_->mode() != 2) {
                                // ファイルが write モードではない
                                tcl.resultf("object %s is not write mode", argv[2]);
                                return(TCL_ERROR);
                        }
#ifdef UD_DEBUG
                        printf("%s: attach-file-out %s\n", __FUNCTION__, argv[2]);
#endif // UD_DEBUG
                        return (TCL_OK);
                }
                /* attach-file-sorted: 並替済出力ファイルのアタッチ */
                else if (strcmp(argv[1], "attach-file-sorted") == 0) {
                        filesorted_ = (UserDataFile*) TclObject::lookup(argv[2]);
                        if (filesorted_ == 0) {
                                // 指定された UserDataFile オブジェクトが見つからない
                                tcl.resultf("no such object %s", argv[2]);
                                return(TCL_ERROR);
                        }
                        if (filesorted_->mode() != 2) {
                                // ファイルが write モードではない
                                tcl.resultf("object %s is not write mode", argv[2]);
                                return(TCL_ERROR);
                        }
#ifdef UD_DEBUG
                        printf("%s: attach-file-sorted %s\n", __FUNCTION__, argv[2]);
#endif // UD_DEBUG
                        return (TCL_OK);
                }
                /* attach-log: ログファイルのアタッチ */
                else if (strcmp(argv[1], "attach-log") == 0) {
                        logout_ = (UserDataLog*) TclObject::lookup(argv[2]);
                        if (logout_ == 0) {
                                // 指定された UserDataLog オブジェクトが見つからない
```

```
                        tcl.resultf("no such object %s", argv[2]);
                        return(TCL_ERROR);
                }
#ifdef UD_DEBUG
                printf("%s: attach-log %s\n", __FUNCTION__, argv[2]);
#endif // UD_DEBUG
                return (TCL_OK);
        }
#ifdef AODV_MULTIROUTE
        /* set-multiroute-scheme: 複数ルートへのパケット割り当て方式の設定 */
        else if (strcmp(argv[1], "set-multiroute-scheme") == 0) {
                if (strcmp(argv[2], "round-robin") == 0) {
                        // ラウンドロビン
                        multiroute_scheme_ = UD_ROUNDROBIN;
                        return (TCL_OK);
                } else if (strcmp(argv[2], "uniform") == 0) {
                        // 等確率
                        multiroute_scheme_ = UD_UNIFORM;
                        return (TCL_OK);
                } else if (strcmp(argv[2], "hop-weighted") == 0) {
                        // 短ルート優先
                        multiroute_scheme_ = UD_HOPWEIGHTED;
                        return (TCL_OK);
                } else if (strcmp(argv[2], "clone") == 0) {
                        // 同一パケットコピー
                        multiroute_scheme_ = UD_CLONE;
                        return (TCL_OK);
                } else if (strcmp(argv[2], "short-only") == 0) {
                        // 最短ルートのみ
                        multiroute_scheme_ = UD_SHORTONLY;
                        return (TCL_OK);
                } else {
                        // 方式名が無効
                        tcl.resultf("invalid multiroute-scheme (%s)\n", argv[2]);
                        return (TCL_ERROR);
                }
        }
#else // AODV_MULTIROUTE
        else if (strcmp(argv[1], "set-multiroute-scheme") == 0) {
                tcl.resultf("%s: set-multiroute-scheme %s: ignoring (disabled by compiling options)\n",
__FUNCTION__, argv[2]);
                return (TCL_OK); // 無視して処理を継続させる
        }
#endif // AODV_MULTIROUTE

        }

        return(Application::command(argc, argv));

}

/* 送信処理（Timer から呼び出される） */
void UserDataApp::send() {

        assert(status_ != UD_S_STOP);

        unsigned char *buf; int buf_len;
```

```
        UserData* data;
#ifdef AODV_MULTIROUTE
        int path_count, path_id, i, j, tmp, tmp2;
        RNG rng;
#endif // AODV_MULTIROUTE

#ifdef AODV_USERDATA_CONNECT
        /* フライング抑止 */
        if (ragent_->rt_request_status() == 0) {
                ragent_->force_rt_request(agent_->daddr(), this);
                wait();
                return;
        } else if (ragent_->rt_request_status() == 2) {
                if (status_ == UD_S_WAIT) {
                        restart();
                }
        } else {
                return;
        }
#endif // AODV_USERDATA_CONNECT

        /* UserData 自身の送信モードチェック */
        if (status_ != UD_S_SEND) {
                return;
        }

        /* パケットの生成 */
        // データの読み込み
        buf = new unsigned char[filein_->unitlen()];
        buf_len = filein_->get_next(buf);
        if (buf_len == -1 || buf_len == 0) {
                // 読み込み失敗 (-1) or ファイル終了 (0)
                stop();
                delete buf;
                return;
        }
        // AppData オブジェクトの生成
        data = new UserData(buf_len);
        memcpy(data->data(), buf, buf_len);
#ifndef AODV_MULTIROUTE
        delete buf;
#else // AODV_MULTIROUTE
        if (multiroute_scheme_ != UD_CLONE) {
                // UD_CLONE 時は buf を温存
                delete buf;
        }
#endif // AODV_MULTIROUTE
        packetID_++; data->packetID(packetID_);

#ifdef AODV_MULTIROUTE

        /* ルート割り当ての決定 */
        if (ragent_ != 0) {
                // ルート数の取得
                path_count = ragent_->rt_count(agent_->daddr());
                // ルート決定
                switch (multiroute_scheme_) {
```

```
case UD_ROUNDROBIN:
        // ラウンドロビン
        if (multiroute_rr_counter_ >= path_count) {
                multiroute_rr_counter_ = 0;
        }
        path_id = multiroute_rr_counter_;
        multiroute_rr_counter_++;
        break;
case UD_UNIFORM:
        // 等確率
        path_id = rng.uniform(path_count);
        break;
case UD_HOPWEIGHTED:
        // 短ルート優先
        path_id = -1; tmp = 0;
        for (i=0; i<path_count; i++) {
                tmp += ragent_->rt_hops(agent_->daddr(), i);
        }
        // ルーレット選択
        tmp2 = rng.uniform(tmp) + 1;
        for (i=0; i<path_count; i++) {
                if (tmp2 <= ragent_->rt_hops(agent_->daddr(), i)) {
                        path_id = i;
                        break;
                } else {
                        tmp2 -= ragent_->rt_hops(agent_->daddr(),i );
                }
        }
        if (path_id == -1) {
                fprintf(stderr, "UserDataApp::send(): UD_HOPWEIGHED failed\n");
                abort();
        }
        break;
case UD_CLONE:
        // 同一パケットコピー
        // とりあえず今のデータについては最初のルートへ、他のルートは後で送信
        path_id = 0;
        break;
case UD_SHORTONLY:
        // 最短ルートのみ
        path_id = 0;
        if (path_count == 1) {
                // ルートが 1 本だけならそれを使う
                break;
        } else {
                tmp = ragent_->rt_hops(agent_->daddr(), 0);
        }
        for (i=1; i<path_count; i++) {
                // 最短ルートを探す
                if (tmp > ragent_->rt_hops(agent_->daddr(), i)) {
                        path_id = i; tmp = ragent_->rt_hops(agent_->daddr(), i);
                }
        }
        break;
default:
        fprintf(stderr, "UserDataApp::send: invalid route selection scheme %d\n", multiroute_scheme_);
        abort();
```

```
                        break;
                }
                // UserData に割り当て情報を入れる
                data->path(path_id);
        } else {
                // ルーティングエージェントがアタッチされていない
                // 0 を入れておく
                data->path(0);
        }

#else // AODV_MULTIROUTE

        // マルチルート OFF：0 を入れておく
        data->path(0);

#endif // AODV_MULTIROUTE

#ifdef UD_DEBUG
        fprintf(stderr, "UserDataApp::send #%d, length %d, path %d at %f\n",
                data->packetID(), data->size(), data->path(), CURRENT_TIME);
#endif // UD_DEBUG

        /* 送信 */
        // ログに記録
        if (logout_ != 0) {
                logout_->write("s", CURRENT_TIME, data->packetID(), data->path(), data->size());
        }
        // 送信処理
        assert(agent_ != 0);
        agent_->sendmsg(data->size(), data);

#ifdef AODV_MULTIROUTE

        if (multiroute_scheme_ == UD_CLONE) {
                // UD_CLONE: 2 番目以降のルートへの送信
                UserData *data[path_count];

                for (i=1; i<path_count; i++) {

                        data[i] = new UserData(buf_len);
                        memcpy(data[i]->data(), buf, buf_len);
                        data[i]->packetID(packetID_);
                        data[i]->path(i);
                        agent_->sendmsg(data[i]->size(), data[i]);
                }

                delete buf;
        }

#endif // AODV_MULTIROUTE

}

/* 送信状態変更 */
void UserDataApp::start() { // 送信開始

        status_ = UD_S_SEND;
```

```
#ifdef UD_DEBUG
        fprintf(stderr, "UserDataApp: started.\n");
#endif // UD_DEBUG

        // Timer の開始
        timer_.sched(interval_);

}
void UserDataApp::wait() { // 送信中断

        status_ = UD_S_WAIT;

#ifdef UD_DEBUG
        fprintf(stderr, "UserDataApp: wait.\n");
#endif // UD_DEBUG

}
void UserDataApp::stop() { // 送信終了

        status_ = UD_S_STOP;

#ifdef UD_DEBUG
        fprintf(stderr, "UserDataApp: stopped.\n");
#endif // UD_DEBUG

        if (filesorted_ != 0) {
                // データ再構築キャッシュ書き出し
                cache_flush();
        }

}
void UserDataApp::restart() { // 送信再開

        status_ = UD_S_SEND;

#ifdef UD_DEBUG
        fprintf(stderr, "%s: UserDataApp::status_ is changed to %d\n", __FUNCTION__, status_);
#endif // UD_DEBUG

}

/* 受信データの処理 */
void UserDataApp::process_data(int size, AppData* appdata)
{
        UserData* data = (UserData*) appdata;

#ifdef UD_DEBUG
        fprintf(stderr, "UserDataApp::process_data #%d, length %d at %f\n",
                data->packetID(), data->size(), CURRENT_TIME);
#endif // UD_DEBUG

        // ログに記録
        if (logout_ != 0) {
                logout_->write("r", CURRENT_TIME, data->packetID(), data->path(), data->size());
        }
```

```
        // UserDataFile::write へデータを渡す
        fileout_->write(data->data(), data->size());

        // データ再構築キャッシュに挿入
        if (filesorted_ != 0) {
                cache_insert(data->data(), data->packetID(), data->size());
        }

}

/* 受信データ再構築キャッシュ：データ挿入 */
bool UserDataApp::cache_insert(unsigned char* data, int id, int size) {

        cache *current_node, *tmp;

        // TODO: リスト走査の高速化

        /*
         * 連結リストの構造
         * struct cache {
         *      struct cache* previous;
         *      unsigned char* cached_data;
         *      int id;
         *      int size;
         *      struct cache* next;
         *      void init(struct cache* prev) {
         *              previous = prev;
         *              cached_data = NULL;
         *              id = 0;
         *              size = 0;
         *              next = NULL;
         *      }
         *      void setdata(unsigned char* data, int data_id, int data_size) {
         *              cached_data = new unsigned char[data_size];
         *              memcpy(cached_data, data, data_size);
         *              id = data_id;
         *              size = data_size;
         *      }
         * };
         * cache* cache_entrypoint;
         */

        // 最初の insert
        if (cache_entrypoint == NULL) {
                cache_entrypoint = new cache;
                cache_entrypoint->init(NULL);
                cache_entrypoint->setdata(data, id, size);
                cache_fillsize = size;  // drop 穴埋めのために記憶

                assert(cache_entrypoint != NULL);
                return true;
        }

        // 2 回目以降
        current_node = cache_entrypoint;
        while(current_node != NULL) {
                if(current_node->id == id) {
```

```
                        // 重複パケット
                        return false;
                } else if(current_node->id < id && current_node->next == NULL) {
                        // リスト末端まで到達 … 末尾に追加
                        current_node->next = new cache;
                        current_node->next->init(current_node);
                        current_node->next->setdata(data, id, size);
#ifdef UD_DEBUG
                        fprintf(stderr, "%s: insert #%d at the last of list, %x\n",
                                __FUNCTION__, id, current_node->next);
#endif // UD_DEBUG
                        return true;
                } else if(current_node->id > id) {
                        // 追加するデータよりも新しいデータに到達 … 一つ前に追加
                        tmp = new cache;
                        tmp->init(current_node->previous);
                        tmp->setdata(data, id, size);
                        tmp->next = current_node;
                        if(current_node->previous == NULL) { cache_entrypoint = tmp; }
                        current_node->previous->next = tmp;
#ifdef UD_DEBUG
                        fprintf(stderr, "%s: insert #%d in front of %d, %x\n",
                                __FUNCTION__, id, current_node->id, tmp);
#endif // UD_DEBUG
                        return true;
                }
                current_node = current_node->next;
        }

        // 追加失敗
        return false;

}
/* 受信データ再構築キャッシュ：データ書き出し */
void UserDataApp::cache_flush() {

        if (flushed_flag == 1) {
                // XXX: 多数回の flush に正しく対応できていないのでエラー扱いにしておく
                fprintf(stderr, "%s: Sorry, you can't flush the cache twice.\n", __FUNCTION__);
        }

        cache *current_node, *prev_node;
        unsigned char *fill;
        int prev_id;
        int i;

        prev_id = 0;
        current_node = cache_entrypoint;
        while(current_node != NULL) {

#ifdef UD_DEBUG
                fprintf(stderr, "%s: flushing cache #%d\n",
                        __FUNCTION__, prev_id + 1);
#endif // UD_DEBUG

                if (current_node->id > prev_id + 1) {
                        // データ抜けの取扱い
```

```
#ifdef UD_DEBUG
                        fprintf(stderr, "%s: #%d not found\n",
                                __FUNCTION__, prev_id + 1, current_node->id);
#endif // UD_DEBUG
                        // XXX: 0 で穴埋め … 要再検討
                        // XXX: データ長は最初に到着したパケットと同じにしておく
                        fill = new unsigned char[cache_fillsize];
                        for (i=0; i<cache_fillsize; i++) { fill[i] = 0; }
                        filesorted_->write(fill, cache_fillsize);
                        delete [] fill;
                        prev_id++; continue;
                }

                // 書き込み処理
                filesorted_->write(current_node->cached_data, current_node->size);

                // ノードの削除処理
                prev_id = current_node->id;
                prev_node = current_node;
                current_node = current_node->next;
                if (current_node != NULL) { current_node->previous = NULL; }
                cache_entrypoint = current_node;
                delete [] prev_node->cached_data;
                delete prev_node;
        }

        flushed_flag = 1;

}

/* パケット受信（未使用）*/
void UserDataApp::recv(Packet* pkt, Handler*)
{
        /*
         *  UserDataApp は Application の子クラスなので recv が必要。
         */

        fprintf(stderr, "UserDataApp::recv shouldn't be called\n");
        abort(); // このメンバ関数は使用しない:

}

#ifdef AODV_USERDATA_CONNECT
void UserDataApp::aodv_callback()
{
        if (status_ == UD_S_WAIT) {
                restart();
        }
}
#endif // AODV_USERDATA_CONNECT
```

A.2.8 userdata/userdata_file.h

```
/* vim: set fenc=utf-8 ff=unix */

/*
 * Application/UserData : ユーザ定義データの送受信
```

```
 *
 * Wada Laboratory, Shizuoka University
 * Takahiro Sugimoto    <taka_s@keyaki.kokage.cc>
 *
 * Last Modified: 2008/11/30 21:32:52
 */

#ifndef ns_userdata_file_h
#define ns_userdata_file_h

#include "object.h"

// 一回に読む量の初期値 … set-unitlen で変更可能。Agent 側のパケットサイズ上限に注意。
#define UNITLEN 512

/* ファイル処理クラス */
class UserDataFile : public NsObject {
 public:
        // コンストラクタ / デストラクタ
        UserDataFile();
        ~UserDataFile();
        // 入出力インターフェース
        int get_next(unsigned char* data);
        void write(unsigned char* data, int datalen_);
        int mode() {
                return mode_;
        }
        int unitlen() {
                return unitlen_;
        }
 protected:
        // Tcl インタプリタ
        virtual int command(int argc, const char*const* argv);
        // パケット受信 (NsObject の子クラスなので必要、実際には使用しない)
        void recv(Packet*, Handler*);
 private:
        // ファイル名
        char *file_;
        // ファイルポインタ
        FILE *fp_;
        // ファイルの状態
        int mode_;              // ファイルの読み書きモード 0:Null 1:Read 2:Write
        int unitlen_;           // 一回の読み書きで処理する長さ
        int infile_pos_;        // Read モード用カウンタ
};

/* ログファイルクラス */
class UserDataLog : public NsObject {
 public:
        // コンストラクタ / デストラクタ
        UserDataLog();
        ~UserDataLog();
        // 入力インターフェース
        void write(const char* action, double time, int packetid, int path, int size);
 protected:
        // Tcl インタプリタ
        virtual int command(int argc, const char*const* argv);
```

```
        // パケット受信 (NsObject の子クラスなので必要、実際には使用しない)
        void recv(Packet*, Handler*);
 private:
        // ファイル名
        char *file_;
        // ファイルポインタ
        FILE *fp_;
};

#endif // ns_userdata_file_h
```

A.2.9 userdata/userdata_file.cc

```
/* vim: set fenc=utf-8 ff=unix */

/*
 * Application/UserData : ユーザ定義データの送受信
 *
 * Wada Laboratory, Shizuoka University
 * Takahiro Sugimoto    <taka_s@keyaki.kokage.cc>
 *
 * Last Modified: 2008/12/08 07:56:08
 */

#include <string.h>
#include "userdata_file.h"

/* Tcl Hooks */
static class UserDataFileClass : public TclClass {
 public:
        UserDataFileClass() : TclClass("UserDataFile") {}
        TclObject* create(int, const char* const*) {
                return (new UserDataFile);
        }
} class_userdatafile;
static class UserDataLogClass : public TclClass {
 public:
        UserDataLogClass() : TclClass("UserDataLog") {}
        TclObject* create(int, const char* const*) {
                return (new UserDataLog);
        }
} class_userdatalog;

/*
 *  "UserDataFile" Class
 *              -- UserData 用ファイルクラス
 */
/* コンストラクタ */
UserDataFile::UserDataFile() {

        // 変数の初期化
        mode_ = 0;
        unitlen_ = UNITLEN;
        infile_pos_ = 0;
        file_ = (char*) NULL;
        fp_ = (FILE*) NULL;
```

```
}
/* デストラクタ */
UserDataFile::~UserDataFile() {

        // XXX: 呼ばれていない？

        if (fp_ != NULL) {
                // ファイルが open されていたらここで閉じる
                fclose(fp_);
        }

}

/* Tcl コマンドインタプリタ */
int UserDataFile::command(int argc, const char*const* argv) {

        /*
         * Tcl 側から UserDataFile に渡された引数はここで処理されます。
         *
         * コマンド一覧:
         *  set userfile [new UserDataFile]
         *    --- 新たな UserDataFile インスタンス $userfile を生成します。
         *        (※ C++ 側でこの処理を担当するのは Tcl Hooks)
         *  $userfile setfile <filename> <r/w>
         *    --- ファイルと読み書きモードを設定します。
         *  $userdata set-unitlen <length>
         *    --- 一回に読むデータの長さを指定します。
         *
         */

        Tcl& tcl = Tcl::instance();

        if (argc == 3) { // 引数 2 個
                /* set-unitlen: 一回に読むデータの長さを指定 */
                if (strcmp(argv[1], "set-unitlen") == 0) {
                        unitlen_ = atoi(argv[2]);
#ifdef UD_DEBUG
                        printf("%s: set-unitlen %f\n", __FUNCTION__, unitlen_);
#endif // UD_DEBUG
                        return (TCL_OK);
                }
        } else if (argc == 4) { // 引数 3 個
                /* setfile: ファイルの指定 */
                if (strcmp(argv[1], "setfile") == 0) {
                        file_ = new char[strlen(argv[2])+1];
                        strcpy(file_, argv[2]);
                        if (strcmp(argv[3], "r") == 0) {
                                mode_ = 1;
                        } else if (strcmp(argv[3], "w") == 0) {
                                mode_ = 2;
                        } else {
                                tcl.resultf("invalid file mode %s for %s", argv[3], argv[2]);
                                return (TCL_ERROR);
                        }
                        return (TCL_OK);
                }
        }
```

```
        return(NsObject::command(argc, argv));

}

/* 次のメッセージを取得 */
int UserDataFile::get_next(unsigned char* data)
{

        int i, length = 0;
        unsigned char* pos = data;

        // 前処理
        if (mode_ != 1) {
                // ファイルが read モードではない
                fprintf(stderr, "%s: %s is not readable (mode %d).\n", __FUNCTION__, file_, mode_);
                return -1;
        }
        if (fp_ == NULL) {
                // まだファイルが open されていないので開く
                if ((fp_ = fopen(file_, "rb")) == NULL) {
                        // open に失敗
                        fprintf(stderr, "%s: failed to open %s\n", __FUNCTION__, file_);
                        return -1;
                }
        }

        // 読み込み処理
        for (i=0; i<unitlen_; i++) {
                if (fread(pos, sizeof(unsigned char), 1, fp_) >= 1) {
                        length++; pos++;
                } else {
                        // ファイル終端またはエラー
                        break;
                }
        }

        return length;

}

/* 受信したメッセージを書き込む */
void UserDataFile::write(unsigned char* data, int datalen_)
{

        // 前処理
        if (mode_ != 2) {
                // ファイルが write モードではない
                fprintf(stderr, "%s: %s is not writeable (mode %d).\n", __FUNCTION__, file_, mode_);
                abort();
        }
        if (fp_ == NULL) {
                // まだファイルが open されていないので開く
                if ((fp_ = fopen(file_, "wb")) == NULL) {
                        // open に失敗
                        fprintf(stderr, "%s: failed to open %s\n", __FUNCTION__, file_);
                        abort();
```

```
                }
        }

        // 書き込み処理
        fwrite(data, sizeof(unsigned char), datalen_, fp_);

}

/* パケット受信（未使用）*/
void UserDataFile::recv(Packet*, Handler*)
{
        /*
         *  UserDataFile は NsObject の子クラスなので recv が必要。
         */

        fprintf(stderr, "UserDataFile::recv shouldn't be called\n");
        abort(); // このメンバ関数は使わない
}

/*
 *  "UserDataLog" Class
 *              -- UserData 用ログファイルクラス
 */
/* コンストラクタ */
UserDataLog::UserDataLog() {

        // 変数の初期化
        file_ = (char*) NULL;
        fp_ = (FILE*) NULL;

}
/* デストラクタ */
UserDataLog::~UserDataLog() {

        if (fp_ != NULL) {
                // ファイルが open されていたらここで閉じる
                fclose(fp_);
        }
}

/* Tcl コマンドインタプリタ */
int UserDataLog::command(int argc, const char*const* argv) {

        /*
         * Tcl 側から UserDataLog に渡された引数はここで処理されます。
         *
         * コマンド一覧:
         *  set userlog [new UserDataLog]
         *    --- 新たな UserDataLog インスタンス $userfile を生成します。
         *  $userfile setfile <filename>
         *    --- ファイルを指定します。
         *
         */

        if (argc == 3) { // 引数 2 個
                /* setfile: ファイルの指定 */
                if (strcmp(argv[1], "setfile") == 0) {
```

```
                        file_ = new char[strlen(argv[2])+1];
                        strcpy(file_, argv[2]);
                        return (TCL_OK);
                }
        }

        return(NsObject::command(argc, argv));

}

/* ログを書き込む */
void UserDataLog::write(const char* action, double time, int packetid, int path, int size)
{

        // 前処理
        if (fp_ == NULL) {
                // まだファイルが open されていないので開く
                if ((fp_ = fopen(file_, "w")) == NULL) {
                        // open に失敗
                        fprintf(stderr, "%s: failed to open %s\n", __FUNCTION__, file_);
                        abort();
                }
        }

        // 書き込むデータの整形
        char buf[100];
        sprintf(buf, "%s %f %d %d %d\n", action, time, packetid, path, size);

        // 書き込み処理
        fputs(buf, fp_);

}

/* パケット受信（未使用）*/
void UserDataLog::recv(Packet*, Handler*)
{
        /*
         *  UserDataLog は NsObject の子クラスなので recv が必要。
         */

        fprintf(stderr, "UserDataLog::recv shouldn't be called\n");
        abort(); // このメンバ関数は使わない
}
```

A.2.10 scripts/demo.tcl

```
# vim: set fenc=utf-8 ff=unix

# Application/UserData バイナリ送受信テスト
#
# Wada Laboratory, Shizuoka University
# Takahiro Sugimoto     <taka_s@keyaki.kokage.cc>
#
# Last Modified: 2008/12/17 18:07:57

#
```

```
# 設定用変数
#

# -- Phy / Mac --
set val(chan)   Channel/WirelessChannel         ;# 通信路の種類
set val(netif)  Phy/WirelessPhy                 ;# Network Interface の種類
set val(prop)   Propagation/Shadowing           ;# 無線伝搬モデル
set val(ant)    Antenna/OmniAntenna             ;# アンテナモデル
set val(mac)    Mac/802_11                      ;# MAC の種類
set val(ifq)    Queue/DropTail/PriQueue         ;# Interface Queue(IFQ) の種類
set val(ifqlen) 50                              ;# IFQ の長さ
set val(ll)     LL                              ;# リンク層の種類
# -- Routing --
set val(rp)     AODV                            ;# ルーティングプロトコル
# -- Network Topology / Topography --
set val(n)      4                               ;# ノード数
set val(x)      500                             ;# Topography(地形) の x 方向
set val(y)      500                             ;# Topography(地形) の y 方向
# -- Other stuff --
set val(simid)  demo                            ;# シミュレーション識別名
set val(infile) $val(simid).in                  ;# 入力ファイル名
set val(outfile)        $val(simid).out         ;# 出力ファイル名
set val(sorted) $val(simid).sorted              ;# 並替済出力ファイル名
set val(logout) $val(simid)_ud.log              ;# UserData ログファイル名
set val(interval) 0.01                          ;# 送信間隔
set val(unitlen) 128                            ;# メッセージ長
set val(scheme) round-robin             ;# 経路割当: ラウンドロビン
#set val(scheme)        uniform                 ;# 経路割当: 一様分布
#set val(scheme)        hop-weighted            ;# 経路割当: ホップ数利用
#set val(scheme)        clone                   ;# 経路割当: 各経路にコピー
#set val(scheme)        short-only              ;# 経路割当: 最短経路のみ

#
# シナリオ
#

# シミュレータの初期化
set ns [new Simulator]

# トレースファイルの初期化
set tracefd [open $val(simid).tr w]
$ns trace-all $tracefd
#$ns use-newtrace
set namtrace [open $val(simid).nam w]
$ns namtrace-all-wireless $namtrace $val(x) $val(y)

# 入力・出力ファイルの初期化
set input [new UserDataFile]
$input setfile $val(infile) r
set output [new UserDataFile]
$output setfile $val(outfile) w
set sorted [new UserDataFile]
$sorted setfile $val(sorted) w
set logout [new UserDataLog]
$logout setfile $val(logout)

# Topography の初期化
```

```
set topo [new Topography]
$topo load_flatgrid $val(x) $val(y)

# God の生成
create-god $val(n)

# ノードの生成
set chan_1_ [new $val(chan)]
$ns node-config -adhocRouting $val(rp) \
                -llType $val(ll) \
                -macType $val(mac) \
                -ifqType $val(ifq) \
                -ifqLen $val(ifqlen) \
                -antType $val(ant) \
                -propType $val(prop) \
                -phyType $val(netif) \
                -topoInstance $topo \
                -agentTrace ON \
                -routerTrace ON \
                -macTrace ON \
                -movementTrace ON \
                -channel $chan_1_
for {set i 0} {$i < $val(n)} { incr i } {
        set node_($i) [$ns node]
}

# Agent・Application のアタッチ
set a(0) [new Agent/UDP]
set a(1) [new Agent/UDP]
$ns attach-agent $node_(0) $a(0)
$ns attach-agent $node_(1) $a(1)
$ns connect $a(0) $a(1)
set userdata(0) [new Application/UserData]
set userdata(1) [new Application/UserData]
$userdata(0) attach-agent $a(0)
$userdata(0) attach-file-in $input
$userdata(0) attach-log $logout
$userdata(0) set-interval $val(interval)
$input set-unitlen $val(unitlen)
$userdata(1) attach-agent $a(1)
$userdata(1) attach-file-out $output
$userdata(1) attach-file-sorted $sorted
$userdata(1) attach-log $logout
# クロスレイヤ連携
$userdata(0) attach-ragent [$node_(0) set ragent_]
$userdata(1) attach-ragent [$node_(1) set ragent_]
# ルート割り当て方式
$userdata(0) set-multiroute-scheme $val(scheme)

# ノードの位置の設定
# X: 横, Y: 縦
#  (2) ---- (1)
#   |        |
#   |        |
#  (0) ---- (3)
#
$node_(0) set X_ 100.0
```

```
$node_(0) set Y_ 0.0
$node_(0) set Z_ 0.0

$node_(1) set X_ 340.0
$node_(1) set Y_ 230.0
$node_(1) set Z_ 0.0

$node_(2) set X_ 100.0
$node_(2) set Y_ 230.0
$node_(2) set Z_ 0.0

$node_(3) set X_ 340.0
$node_(3) set Y_ 0.0
$node_(3) set Z_ 0.0

for {set i 0} {$i < $val(n)} {incr i} {
        $ns initial_node_pos $node_($i) 30
}

# イベントの設定
$ns at 1.0 "$userdata(0) start"
$ns at 500.0 "$userdata(0) stop"
$ns at 500.1 "$userdata(1) stop"
proc finish {} {
        global ns tracefd namtrace

        $ns flush-trace
        close $tracefd
        close $namtrace

        exit 0
}
$ns at 500.2 "finish; $ns halt"

# 実行
$ns run
```

A.2.11 scripts/throughput.pl

```
#!/usr/bin/perl

# スループット計算ツール（トレースファイル使用）
#
# Wada Laboratory, Shizuoka University
# Takahiro Sugimoto     <taka_s@keyaki.kokage.cc>
#
# Last Modified: 2008/12/19 16:29:44

use strict;
use warnings;
use Getopt::Std;

# ** 引数処理
# 既定値
my $FLOW = 'udp';
my $SRC = '_0_';
my $DST = '_1_';
```

```
my $INTERVAL = '1.0';
# Getopt で処理
my %opts = ();
getopts('t:i:d:s:d:f:bh', \%opts);
if (exists $opts{'h'}) {
        &usage;
        exit 0;
}
unless (exists $opts{'t'}) {
        printf STDERR "トレースファイルが指定されていません\n";
        &usage;
        exit 1;
}
if (exists $opts{'f'}) {
        $FLOW = $opts{'f'};
}
if (exists $opts{'i'}) {
        $INTERVAL = $opts{'i'};
}
if (exists $opts{'s'}) {
        $SRC = $opts{'s'};
}
if (exists $opts{'d'}) {
        $DST = $opts{'d'};
}

# ** 集計
open (FILE, "< $opts{'t'}") or die "トレースファイル $opts{'t'} が開けません。($!)";

my $sum_recv = 0; my $throughput = 0; my $clock = 0;
my @tcp_packets = ();
while (<FILE>) {

        my @line = split(' ', $_);
        #  0: 送信 (s) / 受信 (r)
        #  1: イベント発生時間 [s]
        #  2: ノード番号
        #  3: イベント発生レイヤ (IFQ, MAC, AGT, RTR 等)
        #  5: シーケンス番号
        #  6: フローの種類 (ACK, RTS, CTS, tcp, cbr 等)
        #  7: パケットサイズ [bytes]
        # 17: TCP シーケンス番号 (TCP の場合)

        if ($FLOW eq 'tcp' && $line[6] eq 'tcp') {

                # TCP データパケットの場合 … 重複したものを削除して tcp_packets に記憶、スループット計算は後で
                $line[17] =~ s/\[//; $line[18] =~ s/\]//;
                if ($line[0] eq 'r' && $line[2] eq $DST && $line[3] eq 'AGT' && $line[6] eq 'tcp') {
                        my $drop = 0;
                        # 重複確認
                        foreach (@tcp_packets) {
                                if ( $_->[1] == $line[17] ) {
                                        $drop = 1;
                                        last;
                                }
                        }
                        if ($drop != 1) {
```

```
                                my $ref = [$line[1], $line[17], $line[7]];      # 時間，シーケンス番号，サイ
ズ
                                push @tcp_packets, $ref;
                        }
                }

        } elsif ($FLOW ne 'tcp') {

                # TCP 以外 … スループットを逐次計算
                if ($line[1] - $clock > $INTERVAL) {
                        if (exists $opts{'b'}) {
                                $throughput = $sum_recv / $INTERVAL;
                        } else {
                                $throughput = $sum_recv * 8 / $INTERVAL;
                        }
                        print "$clock $throughput\n";
                        $clock = $clock + $INTERVAL;
                        $sum_recv = 0;
                }
                if ($line[0] eq 'r' && $line[2] eq $DST && $line[3] eq 'AGT' && $line[6] eq $FLOW) {
                        # 受信時
                        $sum_recv = $sum_recv + $line[7];
                }

        }

}
if ($FLOW eq 'tcp') {

        # TCP … @tcp_packets に格納されているデータからスループットを計算
        my $sum_recv = 0; my $throughput = 0; my $clock = 0;
        foreach (@tcp_packets) {
                if ($_->[0] - $clock > $INTERVAL) {
                        if (exists $opts{'b'}) {
                                $throughput = $sum_recv / $INTERVAL;
                        } else {
                                $throughput = $sum_recv * 8 / $INTERVAL;
                        }
                        print "$clock $throughput\n";
                        $clock = $clock + $INTERVAL;
                        $sum_recv = 0;
                }
                $sum_recv = $sum_recv + $_->[2];
        }

} else {

        # TCP 以外 … 逐次計算の最後の 1 回
        if (exists $opts{'b'}) {
                $throughput = $sum_recv / $INTERVAL;
        } else {
                $throughput = $sum_recv * 8 / $INTERVAL;
        }
        print "$clock $throughput\n";

}
```

```
# ** 後処理
close (FILE);

# ** 関数: Usage の出力
sub usage {
        printf STDERR <<_OUT_;

Usage: throughput [-t TRACEFILE] [-h]
                  [-f FLOW] [-i INTERVAL] [-s SRC] [-d DST]

Where:
  -t FILE       トレースファイル FILE を指定します。
  -f FLOW       処理対象となるフローを指定します。(既定値：$FLOW)
  -i INTERVAL   集計間隔を指定します。(既定値：$INTERVAL)
  -s SRC        送信元ノードを指定します。(既定値：$SRC)
  -d DST        送信先ノードを指定します。(既定値：$DST)
  -b            出力をバイト毎秒に変更します。(指定のない時は bps で出力)

  -h            このメッセージを出力して終了します。

_OUT_
}

__END__
```

A.2.12 scripts/ud_stat.pl

```
#!/usr/bin/perl

# Application/UserData 用 パケット受信状況解析ツール
#
# Wada Laboratory, Shizuoka University
# Takahiro Sugimoto     <taka_s@keyaki.kokage.cc>
#
# Last Modified: 2008/12/12 12:39:37

use strict;
use warnings;

# 引数確認
if (@ARGV < 1) {
        # 引数なし
        printf STDERR "Usage: ud_stat [log file]\n";
        exit 1;
}

# ファイルを開く
open (DATA, "<" . $ARGV[0]) or die "$!: open failed";

# ファイル読み込みとデータ処理
my $count = 0; my $total = 0; my $dup = 0; my $prev = 0;my @rawdata; my @map = ();
while (<DATA>) {

        @rawdata = split(' ');
        #  0: 送信 (s) / 受信 (r)
        #  1: イベント発生時間 [s]
        #  2: パケット番号
```

```
        #  3: 割当先ルート
        #  4: パケットサイズ [bytes]

        if ($rawdata[0] eq "s") {
                # パケット送信時
                my $ref = [ $rawdata[2], 0]; # パケット番号，受信状況（初期値 0）
                push @map, $ref;
                $total++;
        } elsif ($rawdata[0] eq "r") {
                # パケット受信時
                my $status;
                # 通し番号の記録と順番チェック
                if ($rawdata[2] > $prev) {
                        # 正常な場合
                        $status = 1;
                } else {
                        # 通し番号が逆転している場合
                        $status = 2;
                }
                # 状況の記録
                #foreach (@map) {
                foreach (reverse (@map)) {
                        if ( $_->[0] == $rawdata[2] ) {
                                if ( $_->[1] == 0 ) {
                                        $_->[1] = $status;
                                        $count++;
                                        last;
                                } else {
                                        $dup++;
                                }
                        }
                }
                $prev = $rawdata[2];
        } else {
                print STDERR "invalid log entry!!\n";
                exit 1;
        }

}

# 処理結果の並べ替え
my @sorted = sort { $a->[0] <=> $b->[0] } @map;

# 解析結果表示
#  0 (x) = ドロップ
#  1 (o) = 正常に受信
#  2 (*) = 順番が狂って受信
my $ratio = $count / $total * 100;
print STDOUT "$count of $total packets delivered ($ratio %).\n\n";
print STDOUT "Block Map:\n";
print STDOUT "|--------|---------|---------|---------|---------|---------|---------|---------|\n";
my $i = 0;
foreach (@sorted) {
        $i++;
        if ($i%80 == 1 && $i != 1) { print STDOUT "\n"; }
        if ($_->[1] == 0) { print STDOUT "x"; }
        elsif ($_->[1] == 1) { print STDOUT "o"; }
```

```
        elsif ($_->[1] == 2) { print STDOUT "*"; }
}
print STDOUT "\n|--------|---------|---------|---------|---------|---------|---------|---------|\n";

__END__
```

** 補足説明

ログファイル (UserDataLog) の例

```
-----
s 28.860000 1135 1 128
r 28.865381 1135 1 128
s 28.870000 1136 0 128
r 28.874359 1136 0 128
s 28.880000 1137 1 128
r 28.885321 1137 1 128
s 28.890000 1138 1 128
r 28.895361 1138 1 128
-----
```

[0] = 送信 (s) / 受信 (r)
[1] = 時間 [s]
[2] = パケット ID : userdata->packetID()
[3] = 割当先経路 : userdata->path()
送信元のルーティングテーブルにおける経路番号であることに注意。
[4] = サイズ [bytes] : userdata->size()
Application 層でのパケットサイズ。下位レイヤのヘッダは含まれていない。

A.2.13 scripts/ud_throughput.pl

```
#!/usr/bin/perl

# Application/UserData 用 スループット計算ツール
#
# Wada Laboratory, Shizuoka University
# Takahiro Sugimoto     <taka_s@keyaki.kokage.cc>
#
# Last Modified: 2008/12/11 17:45:21

use strict;
use warnings;
use Getopt::Std;

# ** 引数処理
# 既定値
my $INTERVAL = 1.0;
# Getopt で処理
my %opts = ();
getopts('t:i:bh', \%opts);
if (exists $opts{'h'}) {
        &usage;
        exit 0;
}
unless (exists $opts{'t'}) {
        printf STDERR "ログファイルが指定されていません\n";
        &usage;
```

```
        exit 1;
}
if (exists $opts{'i'}) {
        $INTERVAL = $opts{'i'};
}

# ** 集計
open (FILE, "< $opts{'t'}") or die "ログファイル $opts{'t'} が開けません。($!)";

my $sum_recv = 0; my $throughput = 0; my $clock = 0;
my @checked_packets = ();
while (<FILE>) {

        my @line = split(' ', $_);
        #  0: 送信 (s) / 受信 (r)
        #  1: イベント発生時間 [s]
        #  2: パケット番号
        #  3: 割当先ルート
        #  4: パケットサイズ [bytes]

        if ((exists $opts{'d'}) && !(exists $opts{'s'}) ) {

                # 重複・遅延パケットもスループットに算入 … 随時スループットを計算
                if ($line[1] - $clock > $INTERVAL) {
                        if (exists $opts{'b'}) {
                                $throughput = $sum_recv / $INTERVAL;
                        } else {
                                $throughput = $sum_recv * 8 / $INTERVAL;
                        }
                        print "$clock $throughput\n";
                        $clock = $clock + $INTERVAL;
                        $sum_recv = 0;
                }
                if ($line[0] eq 'r') {
                        # 受信時
                        $sum_recv = $sum_recv + $line[4];
                }

        } else {

                # 重複・遅延パケットの両方、または片方を除外 … 一旦 checked_packets に記憶し、スループット計算は後で
                my $current_id = 0;
                if ($line[0] eq 'r') {
                        my $drop = 0;
                        if (exists $opts{'d'}) {
                                # 重複確認
                                foreach (@checked_packets) {
                                        if ( $_->[1] == $line[2] ) {
                                                $drop = 1;
                                                last;
                                        }
                                }
                        }
                        if (!exists $opts{'s'}) {
                                # 遅延確認
                                if ( $line[2] < $current_id ) {
                                        $drop = 1;
```

```
                                } else {
                                        $current_id = $line[2];
                                }
                        }
                        if ($drop != 1) {
                                my $ref = [$line[1], $line[2], $line[4]];       # 時間，シーケンス番号，サイ
ズ
                                push @checked_packets, $ref;
                        }
                }

        }

}
if (!(exists $opts{'d'}) || (exists $opts{'s'})) {

        # パケットチェックあり … @checked_packets に格納されているデータからスループットを計算
        my $sum_recv = 0; my $throughput = 0; my $clock = 0;
        foreach (@checked_packets) {
                if ($_->[0] - $clock > $INTERVAL) {
                        if (exists $opts{'b'}) {
                                $throughput = $sum_recv / $INTERVAL;
                        } else {
                                $throughput = $sum_recv * 8 / $INTERVAL;
                        }
                        print "$clock $throughput\n";
                        $clock = $clock + $INTERVAL;
                        $sum_recv = 0;
                }
                $sum_recv = $sum_recv + $_->[2];
        }

} else {

        # パケットチェックなし … 逐次計算の最後の 1 回
        if (exists $opts{'b'}) {
                $throughput = $sum_recv / $INTERVAL;
        } else {
                $throughput = $sum_recv * 8 / $INTERVAL;
        }
        print "$clock $throughput\n";

}

# ** 後処理
close (FILE);

# ** 関数: Usage の出力
sub usage {
        printf STDERR <<_OUT_;

Usage: ud_throughput [-t TRACEFILE] [-i INTERVAL] [-b] [-h]

Where:
  -t FILE       ログファイル FILE を指定します。
  -i INTERVAL   集計間隔を指定します。(既定値：1.0)
  -b            出力をバイト毎秒に変更します。(指定のない時は bps で出力)
```

```
  -s            遅れて到着したパケットをスループットに算入しません。（指定のないときは算入）
  -d            重複したパケットも除外せずスループットに算入します。（指定のないときは算入しません）

  -h            このメッセージを出力して終了します。

_OUT_
}

__END__
```

** 補足説明

ログファイル (UserDataLog) の例

```
-----
s 28.860000 1135 1 128
r 28.865381 1135 1 128
s 28.870000 1136 0 128
r 28.874359 1136 0 128
s 28.880000 1137 1 128
r 28.885321 1137 1 128
s 28.890000 1138 1 128
r 28.895361 1138 1 128
-----
```

[0] = 送信 (s) / 受信 (r)
[1] = 時間 [s]
[2] = パケット ID : userdata->packetID()
[3] = 割当先経路 : userdata->path()
送信元のルーティングテーブルにおける経路番号であることに注意。
[4] = サイズ [bytes] : userdata->size()
Application 層でのパケットサイズ。下位レイヤのヘッダは含まれていない。

A.3 本研究で使用したシミュレーションシナリオ

A.3.1 ノード移動シミュレーション

TwoRayGround のシナリオを掲載。Shadowing のシミュレーションでは val(prop) を Propagation/Shadowing に変更のこと。

■TCP の場合

```
# 4nodes-moving_tcp.tcl - 適当に 4 ノードが移動
# original: http://www-sop.inria.fr/mistral/personnel/Eitan.Altman/ns.htm
#
# Last Modified: 2008/11/25 22:17:37

# Define options
set val(chan)           Channel/WirelessChannel    ;# channel type
set val(prop)           Propagation/TwoRayGround   ;# radio-propagation model
set val(netif)          Phy/WirelessPhy            ;# network interface type
set val(mac)            Mac/802_11                 ;# MAC type
set val(ifq)            Queue/DropTail/PriQueue    ;# interface queue type
set val(ll)             LL                         ;# link layer type
set val(ant)            Antenna/OmniAntenna        ;# antenna model
set val(ifqlen)         50                         ;# max packet in ifq
```

```
set val(nn)             4                          ;# number of mobilenodes
set val(rp)             AODV                       ;# routing protocol
set val(x)              500                        ;# X dimension of topography
set val(y)              500                        ;# Y dimension of topography
set val(stop)           150                        ;# time of simulation end

set ns            [new Simulator]
set tracefd       [open 4nodes-moving_tcp.tr w]
set namtrace      [open 4nodes-moving_tcp.nam w]

$ns trace-all $tracefd
$ns namtrace-all-wireless $namtrace $val(x) $val(y)

# set up topography object
set topo       [new Topography]

$topo load_flatgrid $val(x) $val(y)

create-god $val(nn)

#
#  Create nn mobilenodes [$val(nn)] and attach them to the channel.
#

# configure the nodes
        $ns node-config -adhocRouting $val(rp) \
                        -llType $val(ll) \
                        -macType $val(mac) \
                        -ifqType $val(ifq) \
                        -ifqLen $val(ifqlen) \
                        -antType $val(ant) \
                        -propType $val(prop) \
                        -phyType $val(netif) \
                        -channelType $val(chan) \
                        -topoInstance $topo \
                        -agentTrace ON \
                        -routerTrace ON \
                        -macTrace ON \
                        -movementTrace ON

        for {set i 0} {$i < $val(nn) } { incr i } {
                set node_($i) [$ns node]
        }

# Provide initial location of mobilenodes
$node_(0) set X_ 5.0
$node_(0) set Y_ 5.0
$node_(0) set Z_ 0.0

$node_(1) set X_ 490.0
$node_(1) set Y_ 285.0
$node_(1) set Z_ 0.0

$node_(2) set X_ 170.0
$node_(2) set Y_ 260.0
$node_(2) set Z_ 0.0
```

```
$node_(3) set X_ 350.0
$node_(3) set Y_ 25.0
$node_(3) set Z_ 0.0

# Generation of movements
$ns at 10.0 "$node_(0) setdest 250.0 250.0 3.0"
$ns at 15.0 "$node_(1) setdest 45.0 285.0 5.0"
$ns at 110.0 "$node_(0) setdest 480.0 300.0 5.0"
$ns at 1.0 "$node_(3) setdest 250.0 380.0 5.0"

# Set a TCP connection between node_(0) and node_(1)
set tcp [new Agent/TCP/Newreno]
$tcp set class_ 2
set sink [new Agent/TCPSink]
$ns attach-agent $node_(0) $tcp
$ns attach-agent $node_(1) $sink
$ns connect $tcp $sink
set ftp [new Application/FTP]
$ftp attach-agent $tcp
$ns at 1.0 "$ftp start"
$ns at $val(stop) "$ftp stop"

# Define node initial position in nam
for {set i 0} {$i < $val(nn)} { incr i } {
        # 30 defines the node size for nam
        $ns initial_node_pos $node_($i) 30
}

# Telling nodes when the simulation ends
for {set i 0} {$i < $val(nn) } { incr i } {
    $ns at $val(stop) "$node_($i) reset";
}

# ending nam and the simulation
$ns at $val(stop) "$ns nam-end-wireless $val(stop)"
$ns at $val(stop) "stop"
$ns at 150 "puts \"end simulation\"; $ns halt"
proc stop {} {
    global ns tracefd namtrace
    $ns flush-trace
    close $tracefd
    close $namtrace
}

$ns run
```

■UDP の場合

```
# 4nodes-moving_udp.tcl - 適当に 4 ノードが移動
# original: http://www-sop.inria.fr/mistral/personnel/Eitan.Altman/ns.htm
#
# Last Modified: 2008/12/08 17:05:45

# Define options
set val(chan)           Channel/WirelessChannel    ;# channel type
set val(prop)           Propagation/TwoRayGround   ;# radio-propagation model
set val(netif)          Phy/WirelessPhy            ;# network interface type
```

```
set val(mac)            Mac/802_11                  ;# MAC type
set val(ifq)            Queue/DropTail/PriQueue     ;# interface queue type
set val(ll)             LL                          ;# link layer type
set val(ant)            Antenna/OmniAntenna         ;# antenna model
set val(ifqlen)         50                          ;# max packet in ifq
set val(nn)             4                           ;# number of mobilenodes
set val(rp)             AODV                        ;# routing protocol
set val(x)              500                         ;# X dimension of topography
set val(y)              500                         ;# Y dimension of topography
set val(stop)           150                         ;# time of simulation end

set ns            [new Simulator]
set tracefd       [open 4nodes-moving_udp.tr w]
set namtrace      [open 4nodes-moving_udp.nam w]

$ns trace-all $tracefd
$ns namtrace-all-wireless $namtrace $val(x) $val(y)

# set up topography object
set topo       [new Topography]

$topo load_flatgrid $val(x) $val(y)

create-god $val(nn)

#
#  Create nn mobilenodes [$val(nn)] and attach them to the channel.
#

# configure the nodes
        $ns node-config -adhocRouting $val(rp) \
                        -llType $val(ll) \
                        -macType $val(mac) \
                        -ifqType $val(ifq) \
                        -ifqLen $val(ifqlen) \
                        -antType $val(ant) \
                        -propType $val(prop) \
                        -phyType $val(netif) \
                        -channelType $val(chan) \
                        -topoInstance $topo \
                        -agentTrace ON \
                        -routerTrace ON \
                        -macTrace ON \
                        -movementTrace ON

        for {set i 0} {$i < $val(nn) } { incr i } {
                set node_($i) [$ns node]
        }

# Provide initial location of mobilenodes
$node_(0) set X_ 5.0
$node_(0) set Y_ 5.0
$node_(0) set Z_ 0.0

$node_(1) set X_ 490.0
$node_(1) set Y_ 285.0
$node_(1) set Z_ 0.0
```

```
$node_(2) set X_ 170.0
$node_(2) set Y_ 260.0
$node_(2) set Z_ 0.0

$node_(3) set X_ 350.0
$node_(3) set Y_ 25.0
$node_(3) set Z_ 0.0

# Generation of movements
$ns at 10.0 "$node_(0) setdest 250.0 250.0 3.0"
$ns at 15.0 "$node_(1) setdest 45.0 285.0 5.0"
$ns at 110.0 "$node_(0) setdest 480.0 300.0 5.0"
$ns at 1.0 "$node_(3) setdest 250.0 380.0 5.0"

# Set a UDP connection between node_(0) and node_(1)
set udp [new Agent/UDP]
set null [new Agent/Null]
$ns attach-agent $node_(0) $udp
$ns attach-agent $node_(1) $null
$ns connect $udp $null
set cbr [new Application/Traffic/CBR]
$cbr attach-agent $udp
#$cbr set packetSize_ 250
#$cbr set rate_ 100Kb
$cbr set packet_size_ 500
$cbr set interval_ 0.01
$ns at 1.0 "$cbr start"
$ns at $val(stop) "$cbr stop"

# Define node initial position in nam
for {set i 0} {$i < $val(nn)} { incr i } {
        # 30 defines the node size for nam
        $ns initial_node_pos $node_($i) 30
}

# Telling nodes when the simulation ends
for {set i 0} {$i < $val(nn) } { incr i } {
    $ns at $val(stop) "$node_($i) reset";
}

# ending nam and the simulation
$ns at $val(stop) "$ns nam-end-wireless $val(stop)"
$ns at $val(stop) "stop"
$ns at 150 "puts \"end simulation\"; $ns halt"
proc stop {} {
    global ns tracefd namtrace
    $ns flush-trace
    close $tracefd
    close $namtrace
}

$ns run
```

A.3.2 ランダム配置シミュレーション

シェルスクリプト＋ OTcl によるシナリオ本体と、パケット配信率算出スクリプト、ノード配置データで構成。このシナリオは大量のログファイル・トレースファイルを作成するのでディスクの空き容量に注意すること。シングルルートのシミュレーション実行が冗長なのでその部分を効率的にすれば若干の削減が望める。

■シナリオ本体

```
#!/bin/sh

# シミュレーション作業の自動化 for マルチルート利用方法検証
# Last Modified: 2009/01/05 18:25:20

# ** 設定 **
# シミュレーション名
SIMNAME='random2-auto-shadowing'
# パラメータ
SCHEMES='round-robin uniform hop-weighted clone short-only'
P_SIZE='500'
R_START='10'
R_STEP='10'
R_END='1000'
# プログラム
NSDEFAULT='/home/sugimoto/ns-2.32-private/user/ns'
NSLOCAL='/home/sugimoto/ns-2.32-private/useraodv/ns'
UD_DELIVERY='./ud_delivery'

# ** シナリオ **
function ns_run() {

        # ns_run [1:ns path] [2:scheme] [3:rate] [4:simid]

        INTERVAL="`echo "scale=5; (8*${P_SIZE})/($3*1000)" | bc -l`"

        $1 <<EOL
#
# 設定用変数
#

# -- Phy / Mac --
set val(chan)   Channel/WirelessChannel         ;# 通信路の種類
set val(netif)  Phy/WirelessPhy                 ;# Network Interface の種類
set val(prop)   Propagation/Shadowing           ;# 無線伝搬モデル
set val(ant)    Antenna/OmniAntenna             ;# アンテナモデル
set val(mac)    Mac/802_11                      ;# MAC の種類
set val(ifq)    Queue/DropTail/PriQueue         ;# Interface Queue(IFQ) の種類
set val(ifqlen) 50                              ;# IFQ の長さ
set val(ll)     LL                              ;# リンク層の種類
# -- Routing --
set val(rp)     AODV                            ;# ルーティングプロトコル
# -- Network Topology / Topography --
set val(n)      40                              ;# ノード数
set val(x)      1000                            ;# Topography(地形) の x 方向
set val(y)      1000                            ;# Topography(地形) の y 方向
# -- Other stuff --
set val(simid)  $4                      ;# シミュレーション識別名
```

```
#set val(infile)        \$val(simid).in                 ;# 入力ファイル名
set val(infile) 2.2.05.bmp
set val(outfile)        \$val(simid).out                ;# 出力ファイル名
set val(sorted) \$val(simid).sorted             ;# 出力ファイル名
set val(logout) \$val(simid)_ud.log             ;# 出力ファイル名
set val(interval) ${INTERVAL}                           ;# 送信間隔
set val(unitlen) ${P_SIZE}                              ;# メッセージ長
#set val(scheme)        round-robin             ;# 経路割当: ラウンドロビン
#set val(scheme)        uniform                 ;# 経路割当: 一様分布
#set val(scheme)        hop-weighted            ;# 経路割当: ホップ数利用
#set val(scheme)        clone                   ;# 経路割当: 各経路にコピー
#set val(scheme)        short-only              ;# 経路割当: 最短経路のみ
set val(scheme) $2
#set val(ss)            random.dat              ;# ノード配置
set val(ss)             random-1000x1000-40.dat

#
# シナリオ
#

# シミュレータの初期化
set ns [new Simulator]

# トレースファイルの初期化
set tracefd [open \$val(simid).tr w]
\$ns trace-all \$tracefd
#\$ns use-newtrace
set namtrace [open \$val(simid).nam w]
\$ns namtrace-all-wireless \$namtrace \$val(x) \$val(y)

# 入力・出力ファイルの初期化
set input [new UserDataFile]
\$input setfile \$val(infile) r
set output [new UserDataFile]
\$output setfile \$val(outfile) w
set sorted [new UserDataFile]
\$sorted setfile \$val(sorted) w
set logout [new UserDataLog]
\$logout setfile \$val(logout)

# Topography の初期化
set topo [new Topography]
\$topo load_flatgrid \$val(x) \$val(y)

# God の生成
create-god \$val(n)

# ノードの生成
set chan_1_ [new \$val(chan)]
\$ns node-config -adhocRouting \$val(rp) \
                 -llType \$val(ll) \
                 -macType \$val(mac) \
                 -ifqType \$val(ifq) \
                 -ifqLen \$val(ifqlen) \
                 -antType \$val(ant) \
                 -propType \$val(prop) \
                 -phyType \$val(netif) \
```

```
                -topoInstance \$topo \
                -agentTrace ON \
                -routerTrace ON \
                -macTrace ON \
                -movementTrace ON \
                -channel \$chan_1_
for {set i 0} {\$i < \$val(n)} { incr i } {
        set node_(\$i) [\$ns node]
}

# Agent・Application のアタッチ
set a(0) [new Agent/UDP]
set a(1) [new Agent/UDP]
\$ns attach-agent \$node_(0) \$a(0)
\$ns attach-agent \$node_(1) \$a(1)
\$ns connect \$a(0) \$a(1)
set userdata(0) [new Application/UserData]
set userdata(1) [new Application/UserData]
\$userdata(0) attach-agent \$a(0)
\$userdata(0) attach-file-in \$input
\$userdata(0) attach-log \$logout
\$userdata(0) set-interval \$val(interval)
\$input set-unitlen \$val(unitlen)
\$userdata(1) attach-agent \$a(1)
\$userdata(1) attach-file-out \$output
\$userdata(1) attach-file-sorted \$sorted
\$userdata(1) attach-log \$logout
# クロスレイヤ連携
\$userdata(0) attach-ragent [\$node_(0) set ragent_]
\$userdata(1) attach-ragent [\$node_(1) set ragent_]
# ルート割り当て方式
\$userdata(0) set-multiroute-scheme \$val(scheme)

# ノードの初期位置の設定
source \$val(ss)

for {set i 0} {\$i < \$val(n)} {incr i} {
        \$ns initial_node_pos \$node_(\$i) 30
}

# イベントの設定
\$ns at 1.0 "\$userdata(0) start"
\$ns at 2000.0 "\$userdata(0) stop"
\$ns at 2000.1 "\$userdata(1) stop"
proc finish {} {
        global ns tracefd namtrace

        \$ns flush-trace
        close \$tracefd
        close \$namtrace

        exit 0
}
\$ns at 2000.2 "finish; \$ns halt"

# 実行
\$ns run
```

```
EOL
}

# ** 処理 **
# clean : 出力結果消去
if [ "$1" = "clean" ]; then
        rm -f ${SIMNAME}*.tr ${SIMNAME}*.nam.gz ${SIMNAME}*.log
        rm -f ${SIMNAME}*_delivery.txt
        rm -f ${SIMNAME}*.obj ${SIMNAME}*.png
        rm -f ${SIMNAME}*.out ${SIMNAME}*.sorted
        exit
fi

# logclean : ログ消去（結果データは残す）
if [ "$1" = "logclean" ]; then
        rm -f ${SIMNAME}*.tr ${SIMNAME}*.nam.gz ${SIMNAME}*.log
        rm -f ${SIMNAME}*.out ${SIMNAME}*.sorted
        exit
fi

# namonly : nam 以外のログを残さない
if [ "$1" = "namonly" ]; then
        NAMONLY=1
else
        NAMONLY=0
fi

# 各方式ごとに実行
for scheme in $SCHEMES
do

        echo "${scheme} start."

        # 前回の記録を削除
        rm -f ${SIMNAME}_${scheme}_*_delivery.txt

        # スループットを変化させて実行
        for rate in `seq ${R_START} ${R_STEP} ${R_END}`
        do

                echo " ${rate} runnning..."

                # 標準版 ns-2 での処理実行
                #ns_run ${NSDEFAULT} ${scheme} ${rate} ${SIMNAME}_${scheme}_${rate}_default >
${SIMNAME}_${scheme}_${rate}_default.log 2>&1
                ns_run ${NSDEFAULT} ${scheme} ${rate} ${SIMNAME}_${scheme}_${rate}_default > /dev/null 2>&1
                gzip -f ${SIMNAME}_${scheme}_${rate}_default.nam
                echo -n "${rate}        " >> ${SIMNAME}_${scheme}_default_delivery.txt
                ${UD_DELIVERY} -t ${SIMNAME}_${scheme}_${rate}_default_ud.log >>
${SIMNAME}_${scheme}_default_delivery.txt

                # 修正版 ns-2 での処理実行
                #ns_run ${NSLOCAL} ${scheme} ${rate} ${SIMNAME}_${scheme}_${rate} >
${SIMNAME}_${scheme}_${rate}.log 2>&1
                ns_run ${NSLOCAL} ${scheme} ${rate} ${SIMNAME}_${scheme}_${rate} > /dev/null 2>&1
                gzip -f ${SIMNAME}_${scheme}_${rate}.nam
                echo -n "${rate}        " >> ${SIMNAME}_${scheme}_delivery.txt
```

```
                ${UD_DELIVERY} -t ${SIMNAME}_${scheme}_${rate}_ud.log >> ${SIMNAME}_${scheme}_delivery.txt

                # namonly の場合、nam 以外のログを消去
                if [ ${NAMONLY} -eq 1 ]; then
                        rm -f ${SIMNAME}_${scheme}_${rate}*.tr
                        rm -f ${SIMNAME}_${scheme}_${rate}*.log
                        rm -f ${SIMNAME}_${scheme}_${rate}*.out
                        rm -f ${SIMNAME}_${scheme}_${rate}*.sorted
                fi

        done

        # グラフ生成（"EOL"の行まではインデントしないこと）
        gnuplot <<EOL
set terminal tgif
set output "${SIMNAME}_${scheme}_delivery.obj"
set xlabel "Sending bitrate [kbps]"
set ylabel "Packet Delivery Ratio"
plot "${SIMNAME}_${scheme}_default_delivery.txt" title "Single (${scheme})", "${SIMNAME}_${scheme}_delivery.txt"
 title "Multi (${scheme})"
set terminal png
set output "${SIMNAME}_${scheme}_delivery.png"
set xlabel "Sending bitrate [kbps]"
set ylabel "Packet Delivery Ratio"
plot "${SIMNAME}_${scheme}_default_delivery.txt" title "Single (${scheme})", "${SIMNAME}_${scheme}_delivery.txt"
 title "Multi (${scheme})"
EOL

done
```

■パケット配信率算出スクリプト

```
#!/usr/bin/perl

# Application/UserData 用 パケット配信率 の算出
#
# Wada Laboratory, Shizuoka University
# Takahiro Sugimoto     <taka_s@keyaki.kokage.cc>
#
# Last Modified: 2008/12/22 19:33:08

use strict;
use warnings;
use Getopt::Std;

# ** 引数処理
# Getopt で処理
my %opts = ();
getopts('t:h', \%opts);
if (exists $opts{'h'}) {
        &usage;
        exit 0;
}
unless (exists $opts{'t'}) {
        printf STDERR "ログファイルが指定されていません\n";
        &usage;
        exit 1;
```

```
}

# ** ファイルを開く
open (FILE, "< $opts{'t'}") or die "トレースファイル $opts{'t'} が開けません。($!)";

# ** パケット到着状況の記録
my @packets = ();
while (<FILE>) {
        my @line = split(' ', $_);
        #  0: 送信 (s) / 受信 (r)
        #  1: イベント発生時間 [s]
        #  2: パケット番号
        #  3: 割当先ルート
        #  4: パケットサイズ [bytes]

        if ($line[0] eq 's') {
                # 送信時
                my $dup = 0;
                foreach (reverse @packets) {
                        if ($_->[0] == $line[2]) {
                                $dup = 1;
                                last;
                        }
                }
                if ($dup != 1) {
                        my $ref = [$line[2], $line[4], 0];      # シーケンス番号，サイズ，受信フラグ
                        push @packets, $ref;
                }
        }
        if ($line[0] eq 'r') {
                # 受信時
                foreach (reverse @packets) {
                        if ($_->[0] == $line[2]) {
                                $_->[2] = 1;
                                last;
                        }
                }
        }

}

# ** 結果の集計と出力
my $sum_send = 0; my $sum_recv = 0; my $pdr = 0;
foreach (@packets) {
        $sum_send++;
        if ($_->[2] == 1) {
                $sum_recv++;
        }
}
if ($sum_send > 0) {
        $pdr = $sum_recv / $sum_send;
        #print "$sum_recv / $sum_send = $per\n";
        print "$pdr\n";
} else {
        printf STDERR "パケットが送信されませんでした。\n";
}
```

```
# ** 後処理
close (FILE);

# ** 関数: Usage の出力
sub usage {
        printf STDERR <<_OUT_;

Usage: delivery [-t TRACEFILE] [-h]

Where:
  -t FILE       トレースファイル FILE を指定します。

  -h            このメッセージを出力して終了します。

_OUT_
}

__END__

** 補足説明

ログファイル (UserDataLog) の例
-----
s 28.860000 1135 1 128
r 28.865381 1135 1 128
s 28.870000 1136 0 128
r 28.874359 1136 0 128
s 28.880000 1137 1 128
r 28.885321 1137 1 128
s 28.890000 1138 1 128
r 28.895361 1138 1 128
-----

[0] = 送信 (s) / 受信 (r)
[1] = 時間 [s]
[2] = パケット ID     : userdata->packetID()
[3] = 割当先経路      : userdata->path()
      送信元のルーティングテーブルにおける経路番号であることに注意。
[4] = サイズ [bytes] : userdata->size()
      Application 層でのパケットサイズ。下位レイヤのヘッダは含まれていない。
```

■ノード配置データ

```
# gen-random -x 1000 -y 1000 -z 0 -n 40
# Generated at 2008/11/24 16:21:41

$node_(0) set X_ 302.6
$node_(0) set Y_ 383.6
$node_(0) set Z_ 0.0

$node_(1) set X_ 508.6
$node_(1) set Y_ 977.7
$node_(1) set Z_ 0.0

$node_(2) set X_ 744.6
$node_(2) set Y_ 552.9
$node_(2) set Z_ 0.0
```

```
$node_(3) set X_ 159.7
$node_(3) set Y_ 993.7
$node_(3) set Z_ 0.0

$node_(4) set X_ 387.3
$node_(4) set Y_ 489.1
$node_(4) set Z_ 0.0

$node_(5) set X_ 412.0
$node_(5) set Y_ 524.3
$node_(5) set Z_ 0.0

$node_(6) set X_ 374.4
$node_(6) set Y_ 531.9
$node_(6) set Z_ 0.0

$node_(7) set X_ 488.5
$node_(7) set Y_ 295.5
$node_(7) set Z_ 0.0

$node_(8) set X_ 378.5
$node_(8) set Y_ 92.5
$node_(8) set Z_ 0.0

$node_(9) set X_ 286.9
$node_(9) set Y_ 143.4
$node_(9) set Z_ 0.0

$node_(10) set X_ 594.5
$node_(10) set Y_ 56.1
$node_(10) set Z_ 0.0

$node_(11) set X_ 942.7
$node_(11) set Y_ 329.8
$node_(11) set Z_ 0.0

$node_(12) set X_ 840.5
$node_(12) set Y_ 388.2
$node_(12) set Z_ 0.0

$node_(13) set X_ 995.9
$node_(13) set Y_ 829.3
$node_(13) set Z_ 0.0

$node_(14) set X_ 152.6
$node_(14) set Y_ 49.2
$node_(14) set Z_ 0.0

$node_(15) set X_ 39.1
$node_(15) set Y_ 340.8
$node_(15) set Z_ 0.0

$node_(16) set X_ 390.8
$node_(16) set Y_ 118.1
$node_(16) set Z_ 0.0
```

```
$node_(17) set X_ 229.7
$node_(17) set Y_ 50.8
$node_(17) set Z_ 0.0

$node_(18) set X_ 640.9
$node_(18) set Y_ 308.2
$node_(18) set Z_ 0.0

$node_(19) set X_ 317.7
$node_(19) set Y_ 865.3
$node_(19) set Z_ 0.0

$node_(20) set X_ 146.1
$node_(20) set Y_ 454.7
$node_(20) set Z_ 0.0

$node_(21) set X_ 299.2
$node_(21) set Y_ 785.4
$node_(21) set Z_ 0.0

$node_(22) set X_ 268.8
$node_(22) set Y_ 733.1
$node_(22) set Z_ 0.0

$node_(23) set X_ 598.0
$node_(23) set Y_ 846.5
$node_(23) set Z_ 0.0

$node_(24) set X_ 520.6
$node_(24) set Y_ 241.0
$node_(24) set Z_ 0.0

$node_(25) set X_ 887.9
$node_(25) set Y_ 970.9
$node_(25) set Z_ 0.0

$node_(26) set X_ 45.4
$node_(26) set Y_ 595.5
$node_(26) set Z_ 0.0

$node_(27) set X_ 599.7
$node_(27) set Y_ 729.4
$node_(27) set Z_ 0.0

$node_(28) set X_ 462.1
$node_(28) set Y_ 553.0
$node_(28) set Z_ 0.0

$node_(29) set X_ 544.0
$node_(29) set Y_ 318.5
$node_(29) set Z_ 0.0

$node_(30) set X_ 605.7
$node_(30) set Y_ 582.3
$node_(30) set Z_ 0.0

$node_(31) set X_ 388.3
```

```
$node_(31) set Y_ 584.3
$node_(31) set Z_ 0.0

$node_(32) set X_ 233.4
$node_(32) set Y_ 288.2
$node_(32) set Z_ 0.0

$node_(33) set X_ 344.3
$node_(33) set Y_ 143.6
$node_(33) set Z_ 0.0

$node_(34) set X_ 10.9
$node_(34) set Y_ 603.4
$node_(34) set Z_ 0.0

$node_(35) set X_ 964.3
$node_(35) set Y_ 159.0
$node_(35) set Z_ 0.0

$node_(36) set X_ 270.6
$node_(36) set Y_ 497.0
$node_(36) set Z_ 0.0

$node_(37) set X_ 473.0
$node_(37) set Y_ 114.2
$node_(37) set Z_ 0.0

$node_(38) set X_ 859.7
$node_(38) set Y_ 381.4
$node_(38) set Z_ 0.0

$node_(39) set X_ 326.3
$node_(39) set Y_ 639.9
$node_(39) set Z_ 0.0
```

■ランダムノード配置生成スクリプト

```
#!/usr/bin/perl

# ランダムな配置の生成
# Last Modified: 2008/11/22 03:06:02

use strict;
use warnings;
use Getopt::Std;

# ** 引数処理
my $XRANGE = 0; my $YRANGE = 0; my $ZRANGE = 0; my $NUM = 0;
my %opts = ();
getopts('x:y:z:n:h', \%opts);
if (exists $opts{'h'}) {
        &usage;
        exit 0;
}
unless (exists $opts{'x'} && exists $opts{'y'} && exists $opts{'n'}) {
        printf STDERR "引数 ";
        unless (exists $opts{'x'}) { printf STDERR "x "; };
```

```
        unless (exists $opts{'y'}) { printf STDERR "y "; };
        unless (exists $opts{'n'}) { printf STDERR "n "; };
        printf STDERR "が指定されていません\n";
        &usage;
        exit 1;
} else {
        $XRANGE = $opts{'x'}; $YRANGE = $opts{'y'};
        $NUM = $opts{'n'};
        if (exists $opts{'z'}) {
                $ZRANGE = $opts{'z'};
        }
}
if ( ($XRANGE * 10 + 1) * ($YRANGE * 10 + 1) * ($ZRANGE * 10 + 1) <= $NUM ) {
        printf STDERR "ノードを重複無く配置できません（ノード数: %d，取り得る座標の数: %d)。\n", $NUM,
($XRANGE * 10 + 1) * ($YRANGE * 10 + 1) * ($ZRANGE * 10 + 1);
        exit 1;
}

# ** 座標の生成
my @nodes = ();
for (my $i = 0; $i < $NUM; $i++) {

        my $ref; my $flag = 0;
        my $tmp_x = 0; my $tmp_y = 0; my $tmp_z = 0;

        while ($flag == 0) {

                # 座標を生成
                $tmp_x = &trunc(rand $XRANGE, 1);
                $tmp_y = &trunc(rand $YRANGE, 1);
                if ($ZRANGE != 0) {
                        $tmp_z = &trunc(rand $ZRANGE, 1);
                } else {
                        $tmp_z = 0.0;
                }

                # 重複チェック
                $flag = 1;
                foreach(@nodes) {
                        if ( $tmp_x == $_->[0] && $tmp_y == $_->[1] && $tmp_z == $_->[2] ) {
                                $flag = 0;
                        }
                }
        }

        # リファレンスを格納
        $ref = [$tmp_x, $tmp_y, $tmp_z];
        push @nodes, $ref;
}

# ** 出力
print "# gen-random -x $XRANGE -y $YRANGE -z $ZRANGE -n $NUM\n";
printf "# Generated at %s\n\n", &date;
for (my $i = 0; $i < $NUM; $i++) {
        printf "\$node_(%d) set X_ %.1f\n", $i, $nodes[$i]->[0];
        printf "\$node_(%d) set Y_ %.1f\n", $i, $nodes[$i]->[1];
        printf "\$node_(%d) set Z_ %.1f\n", $i, $nodes[$i]->[2];
```

```
        print "\n";
}

# ** Usage 出力関数
sub usage {
        printf STDERR <<_OUT_;

Usage: gen-random [-x RANGE] [-y RANGE] [-n NUM] [-h]

Where:
  -x            X 座標の最大値を指定します。
  -y            Y 座標の最大値を指定します。
  -z            Z 座標の最大値を指定します。(既定値：0)
  -n            ノード数を指定します。

  -h            このメッセージを出力して終了します。

_OUT_
}

# ** 日付取得関数
sub date {
    (my $sec, my $min, my $hour, my $mday, my $month, my $year) = localtime(time);
    my $d = sprintf("%04d/%02d/%02d %02d:%02d:%02d", $year+1900, $month, $mday, $hour, $min, $sec);
    return $d;
}

# ** 指定桁以下切り捨て
# 参考: http://www2u.biglobe.ne.jp/~MAS/perl/waza/trunc.html
sub trunc {
        my $val = shift;        # 切り捨てる数
        my $col = shift;        # 小数点以下のどこまで残すか
        my $r = 10 ** $col;
        if ($val > 0) {
                return int($val * $r) / $r;
        } else {
                my $tmp = $val * $r;
                if ($tmp == int($tmp)) {
                        return $tmp / $r;
                } else {
                        return int($tmp - 1) / $r;
                }
        }
}

__END__
```

謝辞

本研究で利用させていただいた有用なネットワークシミュレータ ns-2 の開発に携わった皆様に感謝致します。また、本研究を進めるにあたり大変お世話になりました和田研究室・石橋研究室のみなさまに深く感謝いたします。

最後に、本研究を進めるにあたり、多大なるご指導・ご助言を賜わりました和田忠浩先生・石橋功至先生に心より感謝いたします。

著作権について

AODV 関連コードの著作権について

本論文の A.2.2 節から A.2.5 節までにかけて掲載されている AODV 関連のソースコードは、CMU Monarch プロジェクトの成果物を元に改変したものであり、下記の条件のもとで改変・再配布を行うことができます。

WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.