修　士　論　文

Potential–based DTN Routing System の評価と改善

Evaluation and Improvement of Potential–based DTN Routing System

指導教員　　江崎 浩 教授

東京大学大学院 情報理工学系研究科
電子情報学専攻

氏　名　　48-086412　下忠 健一

提　出　日　　2010年2月9日

梗概

Delay (or Disruption) Tolerant Networks(DTN) は，不安定な接続関係や大きな遅延を伴うネットワーク環境で効率よくメッセージを配送する手法として，近年注目されている．惑星間通信のように遅延が大きい環境や，ノードの密度が疎なセンサネットワークなどではエンドツーエンドのセッションを張ることができないため，従来の TCP/IP では通信を行うことができない．そこで，トポロジ変化に強く，メッセージ配送の冗長性を低く抑えつつ高い配送率を実現できる DTN ルーティング手法として PEAR(Potential-based Entropy Adaptive Routing) が提案された．しかし，ノードのバッファサイズおよび通信速度を考慮した実験が行われていないため，実環境に即した状況下での性能評価が行われていない．

本研究では，ノードのバッファサイズおよび通信速度が有限である環境下で PEAR の性能を調べ，実環境における PEAR の特性を評価した．さらに，他の DTN ルーティングプロトコルと性能の比較を行った．その結果，PEAR は実環境で有用であり，他のルーティングプロトコルよりも優れていることが示された．

また，これまではメッセージ TTL を適切に設定する方法がなく決め打ちで設定していたが，任意のメッセージ配送率を実現するために必要最小限の TTL を設定する方法を提案し，それによりバッファ使用量を削減できることを示した．さらに，配送遅延を低減するために PEAR におけるポテンシャル値の構築方法を改良し，その効果を検証した．

また，PEAR を応用して DTN 環境で IP ネットワーキングを可能にし，DTN と通常のインターネットの融合を可能にする IP over DTN を提案し，その有用性を示した．

目次

図目次

表目次

第 1 章

序論

1.1 はじめに

インターネットの目覚ましい発展と普及により，人々の生活や産業は飛躍的な進化を遂げた．また，近年は半導体技術の向上により電子部品の高性能化・小型化・低価格化が進み，コンピュータのみならずさまざまなデジタル機器がインターネットに接続可能になり，利便性が向上している．また，電子機器の発展により気象センサや画像センサ，GPS 端末などの各種センサが登場した．これらのセンサも，IP をサポートして IP ネットワークに接続し，遠隔地のセンサデータをネットワークを経由して取得したいというのは自然な発想である．しかし，さまざまな位置にある膨大な数のセンサ機器を全てインターネットに接続することは，容易なことではない．センサ機器が移動する場合も考慮すると，イーサネットや Wi–Fi ではなく GSM や 3G などの携帯電話通信網を使うことが考えられるが，これはインフラが整備された地域でのみ使用可能であり，特に発展途上国などネットワークインフラの整備が進んでいない地域では使用することができない．

そこで，ネットワークインフラのない環境下でメッセージ配送を行うために DTN の技術が有望視されている．DTN とは Delay (or Disruption) Tolerant Networks のことであり，無線センサネットワークのみならず，MANET，惑星間通信，潜水艦間通信など，遅延が大きく，接続関係が不安定でトポロジが頻繁に変化するネットワーク環境を指す．従来の TCP/IP を用いた通信では，エンドツーエンドでセッションを張るために安定したネットワーク環境が必要であり，DTN 環境での仕様は不可能である．DTN では，エンドツーエンドでセッションを張らない手法を用いることで，効率のよいメッセージルーティングを行うことができる．

DTN のルーティング手法は，Epidemic Routing，Potential–based Routing，Link–state Routing に分類することができる．Epidemic Routing は通信可能なノードに次々とメッセージをコピーしていく方式であり，最も速くメッセージを配送することが可能であるが，その分リソース消費も大きい．Potential–based Routing は，ノード間の接続関係やトポロジに応じてポテンシャル値を与え，それに応じて経路を決める手法であり，Epidemic Routing よりもリソース消費を抑えることができる．Link–state Routing は，トポロジ変化が少ないネットワークにおいて，宛先までの到着率や遅延時間をリンクコストとして定義し，少ないコストで宛先へ到達できる経路を選ぶ手法である．

本研究では，DTN ルーティングアルゴリズムの中でもトポロジ変化に強く，メッセージ配送の冗長

性を低く抑えつつ高い配送率を実現できる PEAR(Potential-based Entropy Adaptive Routing) に着目し，実環境で PEAR を運用するための性能評価を行う．また，メッセージ TTL を適切に設定する手法を提案するとともに，配送遅延を低減するためのポテンシャル構築方法を提案する．

実環境では，ノードのバッファサイズと通信速度が有限であるが，PEAR はそれらを考慮した評価が行われていない．バッファサイズが有限であると，バッファオーバーフローを起こした際にメッセージをドロップする順序によって配送性能が変化するので，どのようなバッファ管理方式を用いるかが重要となる．通信速度が有限であると，メッセージを転送する順序が配送性能に影響を与えるので，転送順序を考慮する必要がある．また，これまではメッセージ TTL を適切に設定する方法がなく，充分大きなメッセージ TTL を決め打ちで与えていた．しかし，メッセージ TTL が大きすぎると必要以上にバッファを占有するという問題が生じ，メッセージ TTL が小さすぎると配送率が低下するという問題が生じる．そこで，バッファ管理方式およびメッセージ転送方式が配送性能に与える影響を調べ，最も良いバッファ管理方式とメッセージ転送方式の組み合わせを得るとともに，メッセージ TTL を適切に設定する手法を提案してその効果を検証する．

さらに，PEAR を応用して DTN 環境で IP ネットワーキングを可能にし，DTN と通常のインターネットの融合を可能にする IP over DTN を提案する．従来の DTN では，インターネット上のサーバにデータをアップロードしたい場合は，オーバーレイネットワークを構築してその上でメッセージを伝送する必要があるが，オーバーレイネットワークをインターネット上に構築するのは非常に大変な作業であり，導入や運用のコストが大きく現実的ではない．そこで，DTN 上で IP ネットワーキングが可能になれば，アプリケーションは IP パケットによってデータを伝送できるので，IP ルーターと NAPT によって DTN とインターネットを接続することが可能となる．本研究では，IP over DTN を実機に実装し，実環境で実験をして評価を行い，IP over DTN の有用性を示す．

1.2　本論文の構成

第 2 章では，本研究に関連する技術や研究についての紹介を行う．主に DTN ルーティングプロトコルに関する研究や，実環境での運用の際に考慮しなければならないノードのバッファ管理法およびメッセージ転送方法に関する研究，DTN ルーティングでの課題や想定するシナリオについて述べる．第 3 章では，バッファ管理方法の理論や転送方法の理論，メッセージの TTL に関する理論や，評価軸について述べる．第 4 章では，DTN シミュレータの実装や行った実験について述べる．第 5 章では，実験の結果について述べる．第 6 章では，実験結果の考察および他のルーティングプロトコルとの比較などについて述べる．第 7 章では，IP over DTN に関して提案アーキテクチャや行った実験について述べる．第 8 章では，本研究のまとめと結論を述べる．

第 2 章

関連技術・研究

2.1 DTN ルーティング方式

DTN のアーキテクチャーは Burleigh ら [1] と Kevin Fall[2] によって提案され, internet engineering task force は RFC4338[3] で DTN アーキテクチャーを発表した．これらのアーキテクチャーでは，DTN フレームワークをオーバーレイネットワークとして利用し，メッセージを APDU で配送するバンドル層を導入した．このアプローチは，遅延が大きくパケットロス率が高いネットワークで TCP がうまく機能しないことから生まれた [23]．エンドツーエンドのセッションをホップごとのセッションに分割することで，エンドツーエンドのスループットが向上するとされている．

2.1.1 Epidemic Routing

Epidemic Routing (感染型ルーティング)[4] は，Vahdat らによって提案された．出会ったノード同士で保持しているメッセージのコピーを，互いに相手ノードの中に作ることによって，メッセージを伝播させ，最終的に宛先ノードにたどり着かせる方式である．数学的には，ノードを n，メッセージを m，n の近隣ノードを $nbr(n)$ と表現すると，次の操作で規定される．

$$\forall k \in nbr(k),\ Copy_{n \to k}(m) \tag{2.1}$$

ここで $Copy_{n \to k}(m)$ は，n にある m のコピーを k に作成する操作である．これにより，メッセージ m は，図 2.1 のように周囲のノードに感染しながら伝播していく．

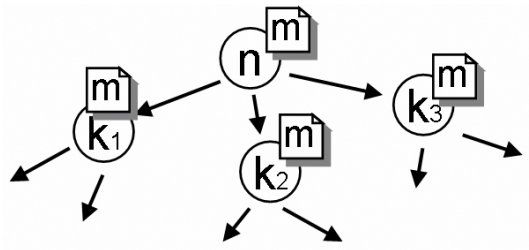

図 2.1　Epidemic Routing によるメッセージ伝搬

Epidemic Routing は， Flooding もしくは Gossiping と呼ばれることもある．出会ったノードすべてに対しメッセージのコピーを作成していくため，余分なパスへメッセージ転送する分，コストが大きくなるが，最小時間で宛先までメッセージを配送することが可能である．

Epidemic Routing の考え方に基づくルーティング手法には，Spray and Wait[5], MaxProp[6] などの派生系がある．

Spray and Wait 方式は，メッセージのソースノードが，時間の経過と共に出会う各ノードにメッセージコピーを作成していく (Spray)．Spray の段階は決められた数のコピーが作成されると終了する．次に，メッセージコピーを持ったノードが移動し，宛先ノードに出会うまで待つ (Wait)．こうしてメッセージを持ったノードが宛先ノードに出会うことで，メッセージを受け渡すことが達成できるルーティング方式である．単純な Epidemic 方式と比べ，Spray 時のみのコピーで済むため，通信コストを低く抑えることができる．この手法は，ノードの行動パターンがランダムで広範囲にわたる場合には効率が良いが，社会的行動パターンの上では，うまく配信できない場合がある [7]．Spray and Wait 方式は，2 ホップで宛先に到達することから，Two–Hop Forwarding 方式 [8] に分類される．

MaxProp 方式は，優先度付きキューを用いて，優先度の高いメッセージから近隣ノードにコピーを作成していく．ここで用いられる優先度は，メッセージの宛先ごとに与えられ，宛先までの近さを表し，概念的には次に述べる Utility とほぼ同じものである．DTN 環境においては，通信帯域及び接続時間が限られているため，すべてのメッセージを交換できない場合がある．このような状況下でも，優先度を高くラベル付けされたメッセージから優先的にコピーすることで，効率的な配信を可能にしている．

2.1.2　Utility–Based Routing

Utility–Based Routing は，各ノードに，宛先までの到達性を現す指標 (=Utility 値) を持たせ，到達性の高い方向へメッセージを転送することで，最終的に宛先へメッセージを届ける方式である．

ノード n における宛先 d への Utility 値を，$U^d(n)$ と書くことにすると，d へのメッセージの転送先は，

$$nexthop^d(n) = \left\{k \mid k \in nbr\,(n) \wedge U^d\,(k) - U^d\,(n) > \alpha\right\} \tag{2.2}$$

で表される．ここで，$nbr(n)$ は，n の近隣ノードの集合で，α は閾値 (> 0) である．

Utility–Based Routing は Potential–Based Routing[9][10][11] や Gradient Routing[12] と同種のものと考えることができる．これらはノードに与えたポテンシャル値で構成される勾配を考え，その勾配を下るようにメッセージを配送する方式である．

これらのルーティング手法の特徴としては，近隣ノードとの関係性だけで転送先を決めることができるため，ネットワークのトポロジ変化によって重大な影響を受けないことである [13][14]．

Utility–Based Routing は，エンドツーエンドのパスを作ることに対し自由性を与えている．その代償に，転送先の選択が必ずしも最適であるとは限らない．Utility 算出モデルが，実際の環境にどの程度，近いかがルーティングの性能を左右する．

Utility の生成方法として，ここでは，History–Based Protocol と Context–Aware Routing(CAR) について述べる．

History–Based Protocol は，ZebraNet[15] において，野生動物に付着された無線センサから，無線

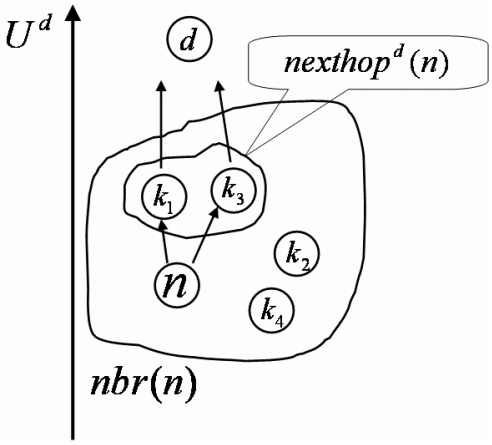

図 2.2　Utility–Based Routing の原理

端末を持ち運ぶ研究者の手によって，動物の場所履歴などの情報を簡単に基地局に収集するために考案されたプロトコルである．宛先 (=基地局) までの接続性 L を考え，宛先の近隣にいるときは，定期的に L に 1 を加える．その他の状況下では，定期的に L を 1 減少させる．メッセージは，L の小さいノードから大きいノードに Unicast 方式で転送される．この方式は，宛先ノードに近いノードが，再び宛先ノードと接触する確率が高い行動パターンの時に効率よく働く．Utility 値に，過去の接続関係を指標として利用した例である．

CAR[16] は，過去の接続関係だけでなく，バッテリ残量などの情報も考慮して Utility 値を計算し，メッセージの転送を行う．一般に，システムのいくつかの側面を属性として記述し，それらに対して Utility 値を与える方式を採用することで，統括的なフレームワークとして利用できるとしている．

2.1.3　Link–State Routing

物理的なネットワークトポロジ (=接続の対向となるノード) が変化しない DTN 環境に対して，Link–State 型のルーティング方式が提案されている．Link–State 型では，リンクコストの決め方がメッセージ配送のパフォーマンスを左右する．コスト情報は，ネットワーク全体に広げる必要があるが，DTN 環境においては，ここに時間がかかるため，比較的変化のある指標をコストとして採用することは望ましくない．

ここでは Link–State 型のルーティングを扱った [17] で取り上げられている，Maximum Delivery Probability(MDP), Minimum Expected Delay(MED) について解説する．

Maximum Delivery Probability (MDP)

ノード a, b 間の接続関係の統計を取ることで，リンク接続確率が $P(a, b)$ と求まったとする．このとき，コストを $1/P(a, b)$ とすることで，計算される最短経路は，平均的な配送確率が最大になる．一般

的には，$P(a,b)$ は，

$$P(a,b) = \frac{\sum (time\, when\, link\, a, b\, is\, up)}{time\, window} \tag{2.3}$$

として計算できる．PRoPHET[18] では $P(a,b)$ を漸化的に求める方法を提案した．

Minimum Expected Delay (MED)

Expected Delay とは，ノード間メッセージ転送の待ち時間である [19]．コストとして，ノード a にあるメッセージが b に渡されるまで待たされる平均時間，

$$W(a,b) = \frac{\sum_{i=1}^{m} d_i^2}{2T} \tag{2.4}$$

を採用する．ここで，T は time window，d_i は T 内で i 番目に切断されていた時間である．MED では，メッセージの平均配送時間が最小となるようにルーティングが行われる．

2.1.4 Potential–Based Entropy Adaptive Routing

Potential–Based Entropy Adaptive Routing (PEAR) は，トポロジ変化に強く，メッセージ配送の冗長性を低く抑えつつ高い配送率を実現できるルーティング手法である．

物理レベルで接続されたネットワークグラフ $G = (N, E)$ を考える．グラフ G 上のノード $n(\in N)$ に対し，$nbr(n)$ で n の近隣ノードの集合を表すことにする．いまノード n において，メッセージの宛先ノード $d(\in N)$ について考える．時刻 t における d 宛の n におけるポテンシャル値を定義し，これを $V^d(n,t)$ と表記する．$V^d(n,t)$ はスカラー値であり，それぞれの宛先 d に対して独立な値を持つ．

Potential–Based Routing (PBR)

時刻 t において n に存在する宛先 d とラベルづけされたメッセージの集合 $M^d(n,t)$ に対し，近隣ノード $k \in nbr(n)$ 方向に働く力 $F_k^d(n,t)$ を次のように定義する．

$$F_k^d \equiv -\left\{V^d(k,t) - V^d(n,t)\right\} \tag{2.5}$$

ここで，$M^d(n,t)$ の転送先 $nexthopd(n,t)$ を，次の規則で与えるのが，PBR による転送規則である．

$$\forall k \in nbr(n),\ \ F_k^d(n,t) \le \alpha \ \rightarrow\ nexthop^d(n,t) = \Phi \tag{2.6}$$

$$\exists k \in nbr(n),\ \ F_k^d(n,t) > \alpha \ \rightarrow\ nexthop^d(n,t) = \left\{k | F_k^d(n,t) = \max_{k \in nbr(n)} F_k^d(n,t)\right\} \tag{2.7}$$

ここで α は正の定数で，転送スレッシュホルドと呼ばれる．，ノードに働く力が α 以下の時，メッセージの転送先は存在しないことを意味し (Φ は空集合)，ノードに働く力が α を越えれば，最大の力を与える近隣ノードを転送先とすることを意味する．この規則で転送を繰り返し行うことで，ポテンシャル値が最も低い宛先 d に，最終的にメッセージが配送される．

図 2.3 に PBR におけるメッセージ転送および配送の様子を示す．この図では，あるネットワーク G の物理的なノード間の接続関係を維持しながら，ノードの持つポテンシャル値を縦にとっている．この状態においては，n の近隣ノード k_1, k_2 で最大の力を与えるものは k_1 であるから，n にあるメッセー

ジは，k_1 に転送される．同様のことが繰り返し転送先で行われることで徐々に宛先 d へ近づき最終的には届く．

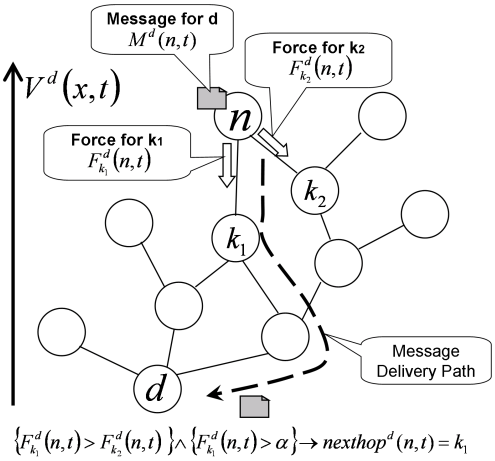

図 2.3　Potential–Based Routing (PBR)

PBR のメリットは，近隣ノードとの関係だけでメッセージの転送先を計算できる点である．グローバルなネットワークトポロジに関する情報と必ずしも同期を取らなくてもメッセージの配送が可能である．

ポテンシャル生成規則

PBR では，ポテンシャルが与えられていれば，前述の方法に従い，次の転送先を計算することができる．ポテンシャルは下記の漸化式により自律的に生成される．

$$V^d(n,t+1) = V^d(n,t) + D \min_{k \in nbr(n)} \left\{0, V^d(k,t) - V^d(n,t)\right\} + \rho \tag{2.8}$$

$$\forall d \in N \ \forall t \ \ V^d(d,t) = 0 \tag{2.9}$$

$$\forall n \in N \ \ V^d(d,0) = 0 \tag{2.10}$$

これは $V^d(n,t)$ に対して，自分の持つポテンシャルよりも低い最小のポテンシャルを持つノードが近隣にあれば，「そのノードとのポテンシャル差分に定数 D $(0 < D < 1)$ をかけたもの」の分だけ自分のポテンシャルを下げ，「定数 ρ $(0 < \rho < D)$」分だけ自分のポテンシャルを上げるようにするということである．また，宛先が自分自身であるならばポテンシャル値は 0，開始時のポテンシャル値は 0，という拘束条件を伴う．

このような形式でポテンシャルを定義すれば，$V^d(n,t)$ の性質を力学的な分析から明らかにすることができる．以下では，3 種類のネットワーク状態においてポテンシャルがどのような形を取るかを述べる．

- 物理的に接続された単一ネットワークの場合 (図 2.4)
  ポテンシャル値は収束し，距離ベクトル型のルーティングと等価な状態になる．図 2.4 左側の

ネットワークに対して生成される，宛先 $d = n6$ へのポテンシャル $V^{n6}(n,t)$ の収束値を図 2.4 の右側に示す．$n6$ からホップ数が増えるごとにポテンシャル値は増加し，メッセージはポテンシャルの低い方向へ配送される．

- ネットワークが物理的に分裂した場合 (図 2.5)
  ネットワークが二つに分裂すると，宛先ノードが属さないネットワーク・ノードのポテンシャル値が増加を開始する．一般に，分裂したネットワークを $G_1 = (N_1, E_1), G_2 = (N_2, E_2)$ とし，G_1 に宛先ノードが存在する場合 (i.e., $d \in N_1$ であるとき)，N_2 のノードのポテンシャルは下記に収束する (c は定数).

$$\forall n \in N_2 \;\; V^d(n,t) \to \rho t + c \;\; (t \to \infty) \tag{2.11}$$

  図 2.5 の例では，$\{n0, ..., n3\}$ のポテンシャルは ρ の速度で上昇する．このとき，$\{n0, ..., n3\}$ にあるメッセージは，各ノードのメッセージバッファに蓄えられる．
- 物理的に分裂した二つのネットワークが再結合する場合 (図 2.6)
  図 2.6 では，$n1 - n5$ 間のリンクが接続されたことにより，二つのネットワークが再結合している．ポテンシャルの低い方のネットワークに接続したノードから順にポテンシャルが降下し，図 2.6 のようなポテンシャルのカーブが生ずる．ここを伝わって $\{n0, ..., n3\}$ のメッセージプールに蓄えられていたメッセージが一気に流れ出る．

ポテンシャルは宛先への到達性を表す，ある種の指標である．「ポテンシャルがより小さい」と「宛先ノードによりよくメッセージを配送できる」が等価となるネットワーク環境であれば，この手法は最大のパフォーマンスを発揮する．

下記の状況が仮定できる環境 (i.e., ノードの接続関係，行動パターン) であれば，式 2.8 から生成されるポテンシャル形状は，メッセージ配送に関して最適な形となる．

- 宛先 d の存在するネットワークから離れているよりは，接続されている方がメッセージは早く配送される環境
- より近い過去に d の存在するネットワークから離れたものが，より近い未来に再接続される確率が高い環境

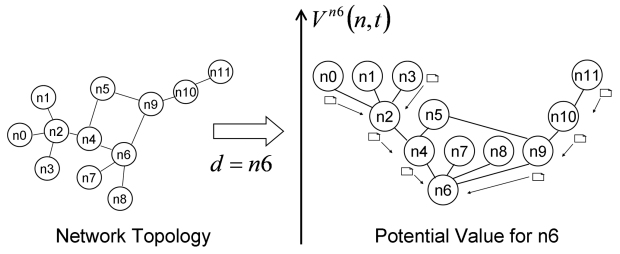

図 2.4　物理的に接続されたネットワークにおける宛先ノード $d = n6$ へのポテンシャルの形状とメッセージの流れ

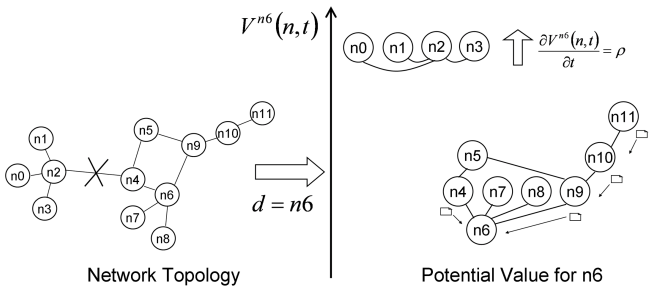

図 2.5　$n2 - n4$ 間のリンクが切断され，二つのネットワークに分離した場合のポテンシャル形状

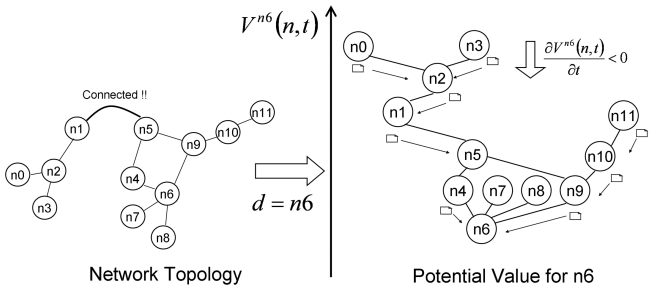

図 2.6　$n1 - n5$ 間のリンクが接続されたことにより，二つのネットワークが再結合する場合

第 3 章

理論

3.1 バッファ管理方式

ノードのバッファサイズおよび通信速度が有限であるときは，メッセージの転送順序（以下，転送方式）およびバッファがオーバーフローした際のドロップ順序（以下，バッファ管理方式）によってルーティングの性能が変化する．

バッファ管理方式は “Early Deletion” と “Late Deletion” の 2 種類に分類することができる．Early Deletion ではバッファがオーバーフローする前にメッセージを削除し，Late Deletion ではバッファがオーバーフローした際にメッセージを削除する．

3.1.1 Early Deletion

Time–Based TTL

Time–Based TTL はメッセージの生存可能時間 (=TTL) をメッセージ生成時に与え，時間経過とともに TTL を減少させていき，TTL が 0 以下になったらメッセージを削除する方式である．

Hop–Based TTL

Hop–Based TTL はメッセージの転送可能回数 (=TTL) をメッセージ生成時に定め，メッセージが他のノードに転送される毎に TTL を減少させていき，TTL が 0 以下になったらメッセージを削除する方式である．

フィードバックによる消去

メッセージが宛先ノードへ届くと，宛先ノードは他のノードに対して配送完了の情報をフィードバックし，他のノードは配送済みのメッセージを削除する方式である．

3.1.2 Late Deletion

Last in First out (LIFO)

LIFO はバッファがフルになった際に，他のノードからのメッセージを受信しないという方式である．メッセージの転送回数が少なくなるため，ノード間のパケット伝送量を抑えることができる．

First in First out (FIFO)

FIFO は，バッファーオーバーフローが生じた場合に，先に受信したメッセージから順に削除する方式である．短時間に多数のメッセージを受信した場合に，一度も他のノードへ転送されずに削除されるメッセージが生じてしまうという欠点を有する．

Oldest–Drop

Oldest–Drop とは，最も古いメッセージ，すなわち TTL の残りが最も少ないメッセージを最初に削除する方式である．古いメッセージは既に他のノードへコピーされネットワーク中に広がっていると考えられるため，削除しても配送への影響が少ないという考え方に基づく方式である．

Youngest–Drop

Youngest–Drop は，Oldest–Drop とは逆の考え方に基づいている．TTL の残りが少ないメッセージはすぐに TTL が 0 以下になり削除されてしまうので，宛先へ届けるためには優先的に転送する必要がある．一方，TTL の残りが多いメッセージは，他のノードから宛先へ配送できる余地があるため，削除しても配送への影響が少ないという考え方に基づく方式である．

Priority–Drop

Priority–Drop は，メッセージを送信するアプリケーションが優先度を付けてメッセージを送信している場合に，優先度の低いメッセージから削除することで，優先度の高いメッセージはバッファに残し配送率を維持しようとする方式である．

Most Forwarded First Drop

Most Forwarded First Drop は，メッセージをネットワーク中に広げることで配送率を高めようとする方式である．転送回数の多いメッセージは既にネットワーク中に広がっていると考えられるので，転送回数の多いメッセージから削除することで，転送回数の少ないメッセージに転送の機会を与えてネットワーク中に広げようとする方式である．

Most Favorably Forwarded First Drop

Most Favorably Forwarded First Drop は，PRoPHET のように各ノードが宛先ノードまでの配送率 P を計算している場合に，FP 値

$$FP = FP_{old} + P \tag{3.1}$$

が最も高いメッセージから削除する方式である．この FP 値は，転送された回数が多いほど，かつ転送先ノードから宛先へ届く確率が高いほど大きくなるので，宛先に配送される可能性が高いメッセージから削除されることになる．すなわち，宛先へ届く可能性が低いメッセージをバッファに残すことで，他のノードへ転送される機会を与えて配送率を高めようとする方式である．この方式は，重み付きの Most Forwarded First Drop と見なすことができる．

Least Probability First Drop

Least Probability First Drop は，Most Favorably Forwarded First Drop とは逆に，宛先への配送率が最も低いメッセージから削除する方式である．この方式では，宛先への配送率が低いメッセージは保持していても宛先へ届く可能性は低いと考え，宛先への配送率が高いメッセージを保持して確実に届けようとするものである．

3.2 メッセージ転送方式

ノード間の通信速度が有限であり，また，通信が途中で途絶えることがある状況下では，全てのメッセージを転送できない場合があるので，メッセージを送信する順序が重要になる．

Last in First out (LIFO)

LIFO は，後から受信したメッセージを先に転送する方式であり，後から受信したメッセージはまだ他のノードへあまり転送されていないと考えられるので，先に転送してメッセージをネットワーク中に広げようとする方式である．

First in First out (FIFO)

FIFO は，先に受信したメッセージを先に転送する方式であり，先に受信したにもかかわらずまだバッファ中に残っているメッセージは何らかの理由で宛先へ届きにくいということなので，そのようなメッセージを先に転送して早く宛先へとどけようとするものである．

Oldest First Out

Oldest First Out は，最も古いメッセージすなわち最も TTL の残りが少ないメッセージを先に転送する方式である．最も TTL の残りが少ないメッセージは最も早く消滅してしまうので，先に転送して早く宛先へとどけようとするものである．

Least Forwarded Fisrt Out

Least Forwarded Fisrt Out は，メッセージをネットワーク中に広げることで配送率を高めようとする方式である．転送回数の多いメッセージは既にネットワーク中に広がっており宛先ノードへ到着できる可能性が高まっているので，転送回数が少なくまだネットワーク中に広がっていないメッセージを優先的に転送することで，メッセージの配送率を高めようとする方式である．

PRoPHET では各ノードから宛先ノードへの到達確率 P が計算されるので，これを用いて以下のような転送方式をとることができる．ノード A がノード B と出会い，メッセージの宛先を D とする．$P(X,Y)$ はノード X から宛先 Y への到達確率を表す．

GRTRSort

GRTRSort では，$P(B,D) - P(A,D)$ の値が大きい順に，$P(B,D) > P(A,D)$ の場合のみ転送する．これは，ノード A よりもノード B の方がメッセージを宛先へ届けられる確率が高い場合のみメッセージを転送し，転送の順序は，転送することによって配送率がより向上する順とするものである．

GRTRMax

GRTRMax は，GRTRSort と同様に $P(B,D) > P(A,D)$ の場合のみ転送するが，順序は $P(B,D)$ の大きい順とするものである．これは，転送先のノードから宛先に届く確率が最も高いメッセージから転送するものである．

3.3 メッセージ TTL の設定

これまではメッセージ TTL の明確な設定方法がなく，ノードの行動シナリオに応じて決め打ちで TTL を設定していた．しかし，ノードの行動シナリオがあらかじめ分かっていない場合や，行動パターンが途中で変化する場合は，この方法では適切な TTL を与えることができない．TTL が小さすぎるとメッセージ配送率が低くなるという問題が生じ，TTL が大きすぎるとバッファ占有率が高くなるという問題がある．すなわち，メッセージ配送率とバッファ占有率はメッセージ TTL に関してトレードオフの関係にある．従って，メッセージ配送率とバッファ占有率のどちらを優先するかは運用ポリシー次第であるが，送信ノードが現在のメッセージ配送率を知ることができれば，運用ポリシーに応じてメッセージ TTL を適切に調整することができる．

送信ノードに配送率を知らせるために，宛先ノードからメッセージ到着の acknowledgement を用いる以下の方法を提案する．まず，送信ノードはメッセージ送信時に，それまでに送信したメッセージの総数をヘッダに付加する．受信ノードがメッセージを受信すると，メッセージの送信数を知ることができる．複数のメッセージを受信したり，あるいはメッセージの送信順と異なる順序で受信することもあるので，その場合は最も大きな値の送信数を現在のメッセージ送信数として採用する．受信ノードは受信したメッセージの総数を保持しておき，

$$[\text{メッセージ配送率}] = \frac{[\text{受信したメッセージの総数}]}{[\text{得られたメッセージ送信総数}]}$$

によって現在のメッセージ配送率を計算する．送信されたメッセージが受信ノードに届くまでには順序が入れ替わる可能性があるので，得られるメッセージ送信数は必ずしも正確ではないが，この配送率をメッセージ到着の acknowledgement に付加して送信ノードに送り返すことで，送信ノードに配送率を知らせることができる．なお，メッセージ TTL の値によっては acknowledgement が送信ノードに届

く前に TTL が 0 になってメッセージが削除されてしまう場合がある．そこで，TTL が十分大きいメッセージを定期的に送信することにより，確実に配送率を送信ノードに届けられるようにする．

3.4 ポテンシャル構築の改良

PEAR では，宛先ノードへ近いノードほど低いポテンシャルを持ち，宛先ノードから遠ざかるにしたがって高いポテンシャルを持つ（図 3.1）．リンクが切れて宛先ノードへの接続性を失ったノード群はポテンシャルが上昇することにより宛先ノードから遠ざかっていることを表すが，その際に群内のノードはポテンシャルが上昇しながら同一の値となる（図 3.2）．ポテンシャルが同一になるということは，宛先ノードへの到達可能性が同一であるということを意味するため，群内でのメッセージ配送を行うことができなくなる．また，切れたリンクが再びつながるとポテンシャルが元に戻るが，元に戻るまでの間はメッセージを配送することができない（図 3.3）．

そこで，リンクが切れている間は，リンクが切れる直前の順序を維持することにより，リンクが切れている間も群内でのメッセージ配送を可能にすることができる（図 3.4）．また，再びリンクがつながった際もすぐにメッセージ配送が可能となる（図 3.5）．具体的には自身のポテンシャルの計算方法を，近隣ノードに自分より低いポテンシャルを持つノードがあり，そのポテンシャルが上昇中で，かつ自分のポテンシャルとの差が ρ/D 以下であれば式 3.2 とすることで過去の順序を維持し，それ以外の場合は従来のポテンシャル計算式と同じ式 3.3 とすることでトポロジ変化に対応する．

$$V^d(n,t+1) = \min_{k \in nbr(n)} \left\{0, V^d(k,t)\right\} + \frac{\rho}{D} \tag{3.2}$$

$$V^d(n,t+1) = V^d(n,t) + D \min_{k \in nbr(n)} \left\{0, V^d(k,t) - V^d(n,t)\right\} + \rho \tag{3.3}$$

これにより配送遅延の短縮が期待されるが，リンクが切れる前と異なるノードとリンクが繋がった場合は，リンクが切れる前と異なる順序のポテンシャルを形成する必要があるため，配送遅延が増加することが予想される．

3.5 モビリティモデル

Random Waypoint

Random Waypoint はノードの位置を一様分布でランダムに決定するモビリティモデルである [22]．ノードの移動先の位置を一様分布に従うように選び，その位置までの移動速度 v を決めて移動し，到着したらガウス分布に従う停止時間 T_{stop} だけその場にとどまるという動作を繰り返す行動パターンである．

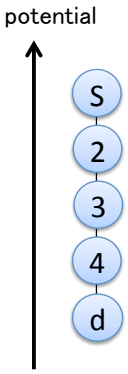

図 3.1　接続されている状態のポテンシャル

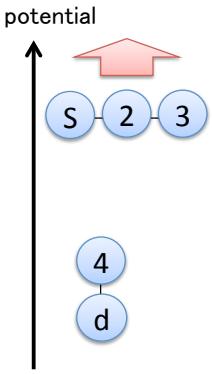

図 3.2　リンクが切れた状態のポテンシャル

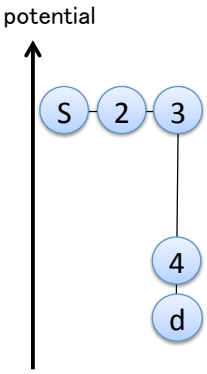

図 3.3　再びリンクが繋がった直後のポテンシャル

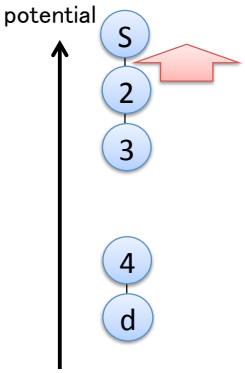

図 3.4　リンクが切れた状態のポテンシャル

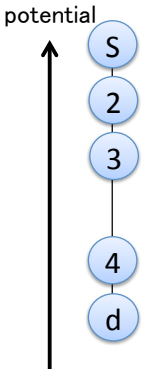

図 3.5　再びリンクが繋がった直後のポテンシャル

Levy–Walk

Levy–Walk は，ブラウン運動には従わない粒子の拡散を表現するために作られたモデルである．移動距離 l と停止時間 Δt_p を

$$f_X(x) = \frac{1}{2\pi}\int_{-\infty}^{+\infty} e^{-itx-|ct|^\alpha}\,dt$$

に従う乱数で決定し，移動方向 θ は一様乱数，移動速度 v は $v = \dfrac{l^\rho}{k}$ で決定する．ただし c, α, ρ, k は定数である．$\alpha < 2$ のとき，$f_X(x)$ は $f_X(x) = \dfrac{1}{|x|^{1+\alpha}}$ と近似できる．

3.6　課題と評価軸

PEAR は DTN 上で効率よくメッセージ配送を行うために開発されたルーティングアルゴリズムであるが，これまではノードのバッファサイズが無限大，ノード間の通信速度が無限大という状況を仮定

した条件下での評価しか行われていない．そこで本研究では，ノードのバッファサイズおよび通信速度が有限である状況下で PEAR の評価を行う．以下の 3 つの状況において，それぞれ評価を行う．

- 通信速度は無限大で，バッファサイズをのみが有限の場合
- バッファサイズは無限大で，通信速度のみが有限の場合
- バッファサイズと通信速度の両方が有限の場合

評価する項目は，

- メッセージ配送率
- 配送遅延
- 配送冗長度
- 伝送量
- バッファ使用量

とする．また，メッセージ TTL がメッセージ配送率およびバッファ占有率に与える影響を調べるとともに，送信ノードが適切なメッセージ TTL を設定できるか検証する．さらに，ポテンシャル構築の改良により配送遅延がどのような影響を受けるかを，改良が有効に機能するモビリティと，そうでないモビリティにおいて評価する．

第 4 章

実装・実験

4.1 DTN シミュレータの構築

ノードのバッファサイズやノード間の通信速度がルーティング性能に与える影響を調べるために，バッファサイズや通信速度を変えてそれぞれの場合の結果を比較する必要がある．影響を正しく評価するためには，他の条件を同一にして実験を行う必要がある．しかし，実機を用いた実験では，ノードの行動パターンや電波状況を毎回同一にすることは不可能であり，正しく評価することができない．そこで，実機ではなくシミュレーションによって実験を行うことにより，同一の条件下で繰り返し実験を行うことができる．

4.1.1 シミュレータに必要な仕様

シミュレータに求められる仕様を以下に挙げる．

- ノードモビリティの入力
- PEAR の再現
- ログ情報の出力

ノードモビリティの入力

ノードモビリティの入力方法としてまず考えられるのは，ノードの位置情報を入力する方法である．各時刻におけるノードの位置を与え，電波の届く距離をあらかじめ定めておくことにより，ノード間の接続関係をシミュレータ上で作り出すことができる．しかし，ノードモビリティをコンピュータ上で合成して入力する場合はこの方法で良いが，実機での実験を再現したい場合には問題が生じる．実機でノードの位置情報を正確に記録するには，GPS を用いて位置を記録することが最も現実的であるが，GPS には欠測や誤測があり，地下など GPS の電波が届かない状況下では計測できないという問題点がある．さらに，電波が届く距離は，周囲の障害物や建物の構造，電磁的ノイズの影響などがあり，一定でない．したがって，この方法では実機での実験を正確に再現することは難しい．

そこで，コンタクトパターンによるノードモビリティの入力を採用することにした．コンタクトパターンとは，ノード間の接続関係を表す情報である．具体的には，各ノードが一定時間毎にブロード

キャストでパケットを送信し，あるノードが他のノードからこのパケットを受信した際に受信時刻と相手ノードを記録した情報が，コンタクトパターンである．これは，実際に電波が届いたかどうかを表す情報なので，実機でのノードの接続関係を正確に再現することができる．また，ノードモビリティを合成する場合も，このコンタクトパターンを合成するのは容易である．

4.1.2　実装の構成

シミュレータは Java で実装した．以下のクラスで構成した．

- DTNSimulator クラス
  メインクラス．メインメソッドから他のクラスを操作し，シミュレーション全体の実行・制御を行う．
- Logger クラス
  ログメッセージを受け取り，ファイルへ出力するクラス．
- LogReader クラス
  コンタクトパターンを読み込み，どのノードとどのノードか通信可能かを判断するクラス．
- Message クラス
  メッセージのクラス．TTL やメッセージ ID，データを保持する．
- Node クラス
  ノードのインターフェースクラス．インターフェースのインプリメンテーションを変えれば，PEAR だけでなく他のルーティングプロトコルを動作させることができる．
- NodeReader クラス
  ノードの個数およびノード ID を管理するクラス．
- SingleCopyNode クラス
  Node クラスのインプリメンテーション．PEAR を実装したもの．ソースコードは付録 A.1 を参照．
- Time クラス
  時刻を管理するクラス．現在時刻を保持し，Time Unit 毎に時間を進める．

4.2　実験シナリオ

図 4.1 に示すように，送信ノード 1 個，宛先ノード 1 個の間を，18 個の中継ノードが 6 個ずつ 3 つのグループに分かれて行動するというシナリオで実験を行った．ノードは Levy–Walk モデルに従って移動する．ただし，x 軸方向の移動は，領域内を往復する動きとなるように制約を加えた Levy-Walk モデルを適用した．Levy–Walk のパラメータは，$\alpha = 1.5$, $k = 18.72$, $\rho = 0.79$ とした．ノードの移動速度の平均値は 0.76[m/s] であった．時刻 0[s] から 8000[s] の間，送信ノードは 10 秒ごとにメッセージを 1 個送信し，時刻 10000[s] でシミュレーションを終了した．

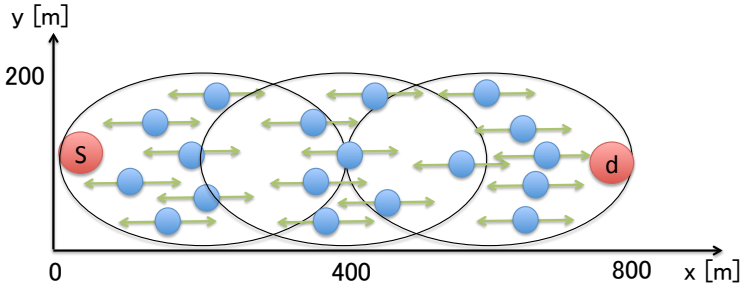

図 4.1　実験に用いたノードの行動シナリオ

バッファサイズおよび通信速度の影響

以下のそれぞれの場合について実験を行い，ルーティング性能を評価した．

- メッセージ TTL が十分大きく，バッファサイズが有限で，通信速度が無限大の場合
- メッセージ TTL が十分大きく，バッファサイズが無限大で，通信速度が有限の場合
- メッセージ TTL が十分大きく，バッファサイズが有限で，通信速度が有限の場合

通信速度は，送信ノードでのメッセージ生成速度とノード間でのメッセージ転送速度の比で表した相対値を用いた．

$$\text{通信速度} = \frac{\text{メッセージ転送速度}}{\text{メッセージ生成速度}}$$

バッファサイズは 10[messages]〜160[messages] まで変え，通信速度は 0.5〜10.0 まで変えて実験を行った．

メッセージ TTL の影響とその自動設定の効果

メッセージ TTL の設定がメッセージ配送率とバッファ使用量に与える影響を調べるため，バッファサイズ無限大・通信速度無限大の条件下でメッセージ TTL を 100[s]〜4000[s] まで変えてメッセージ配送率とバッファ使用量の変化を調べた．さらに，送信ノードがメッセージ TTL を調節することにより配送率とバッファ使用量を制御した場合の効果を調べた．

ポテンシャル構築の改良による配送遅延の変化

ポテンシャル構築の改良が配送遅延に与える影響を調べるため，改良に対して有利なモビリティと，不利なモビリティそれぞれに関して，合成したモビリティと実際のモビリティで改良による配送遅延の変化を調べた．

第 5 章

結果

5.1 バッファサイズおよびバンド幅が与える影響

メッセージ TTL が十分大きく，バッファサイズが有限で，通信速度が無限大の場合

メッセージ TTL が十分大きい場合に，通信速度を無限大としたときのバッファサイズと性能の関係は次のようになった．

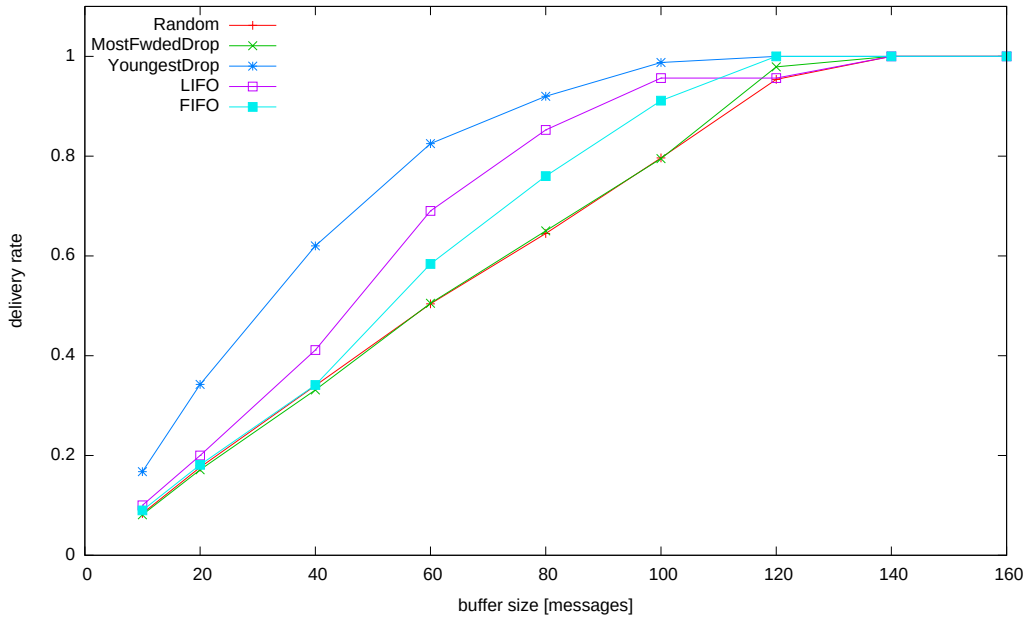

図 5.1 バッファサイズ有限の場合の平均配送率

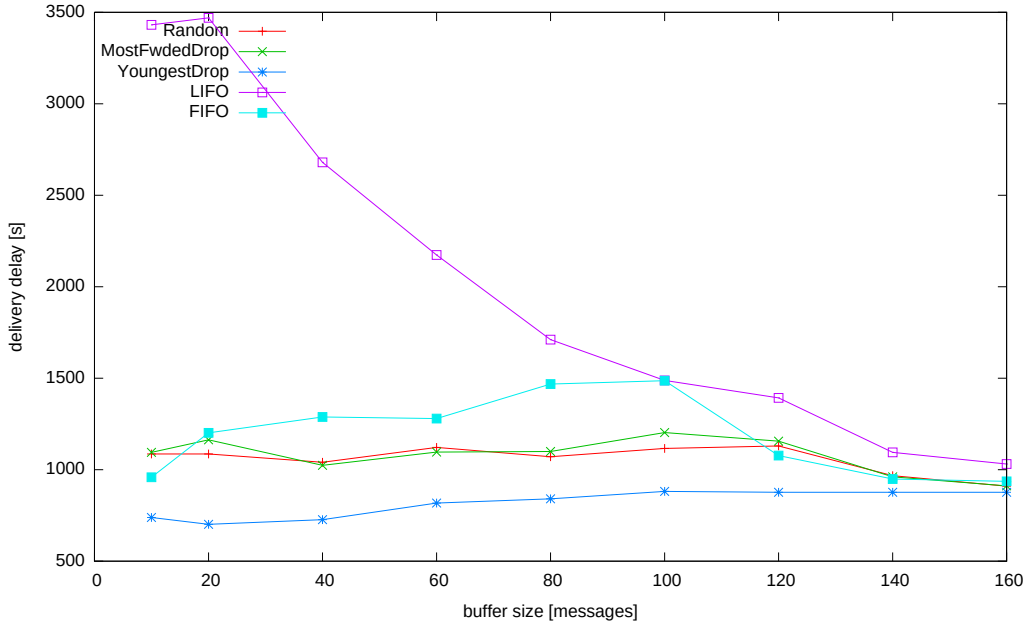

図 5.2　バッファサイズ有限の場合の平均配送遅延

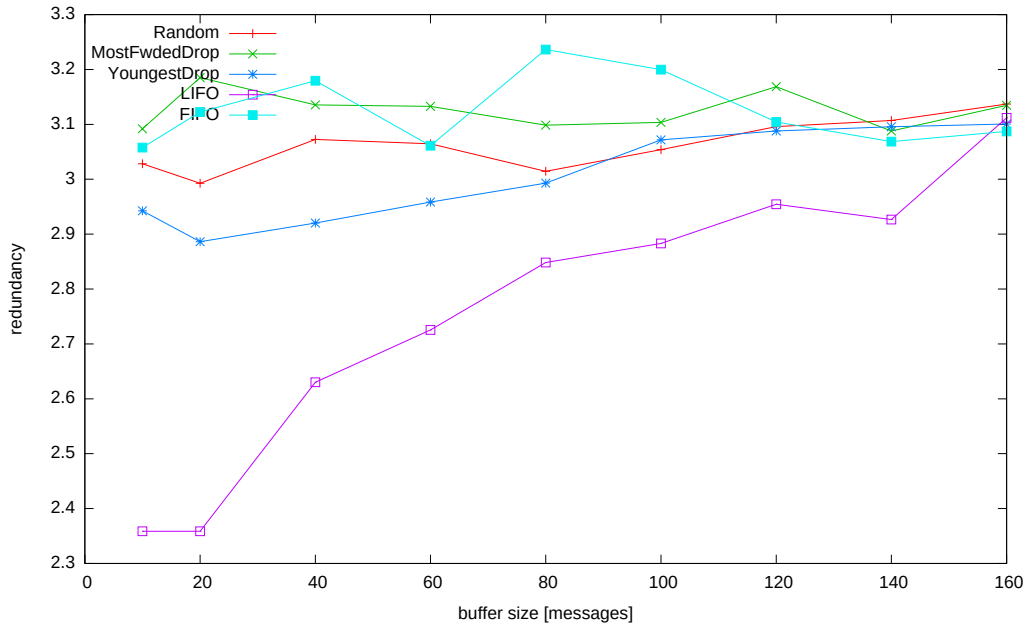

図 5.3　バッファサイズ有限の場合の平均配送冗長度

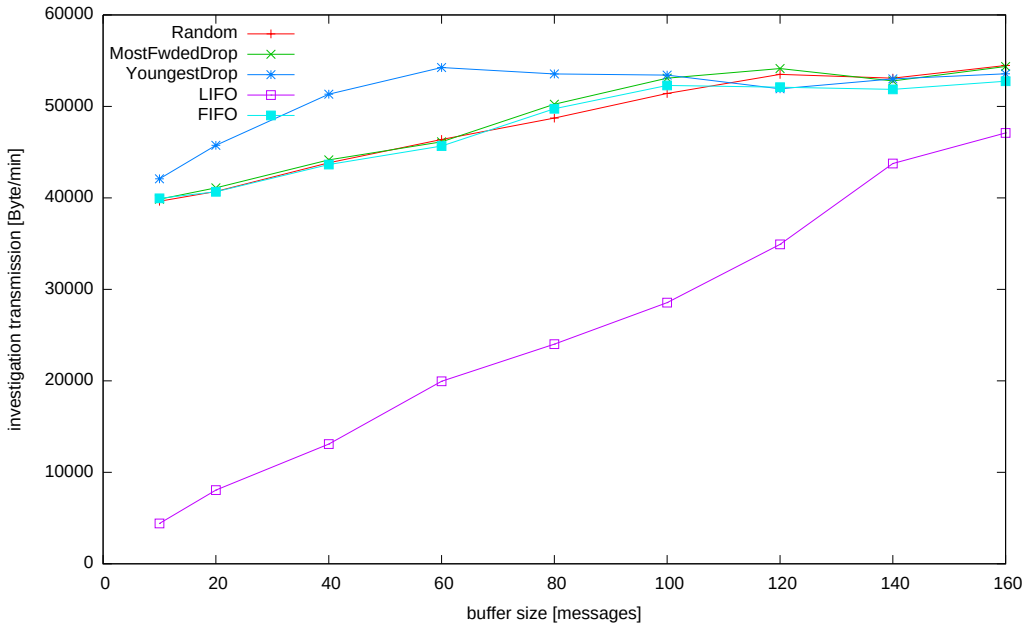

図 5.4　バッファサイズ有限の場合の investigation パケット伝送量

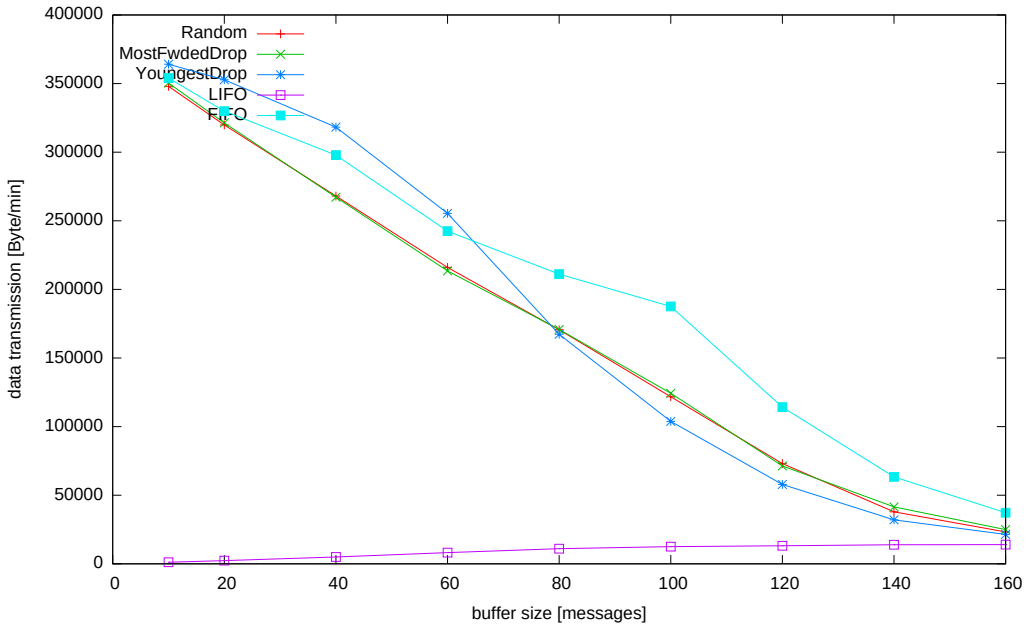

図 5.5　バッファサイズ有限の場合のデータパケット伝送量

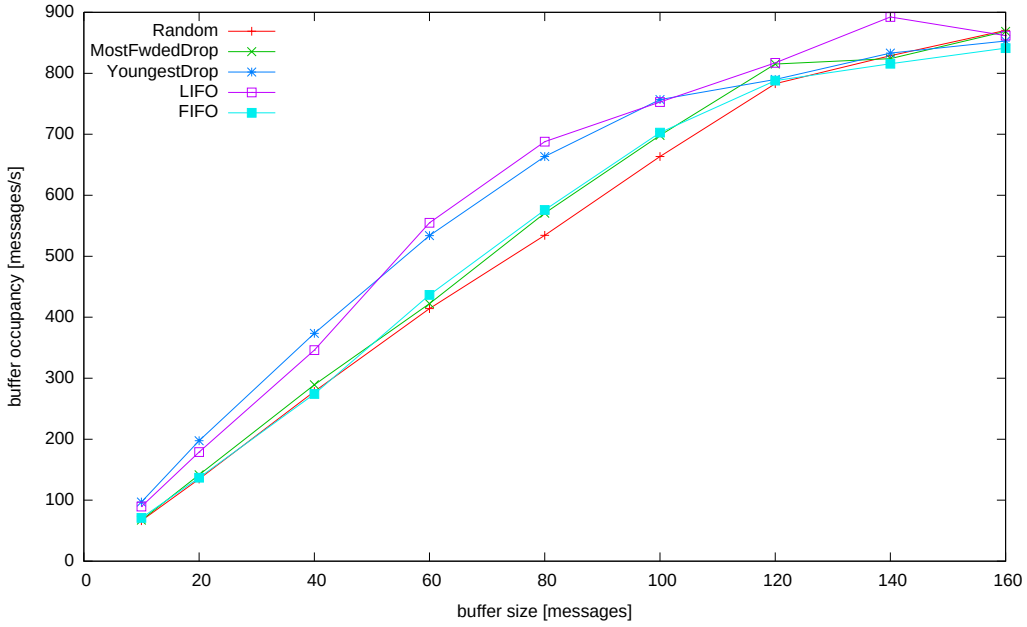

図 5.6　バッファサイズ有限の場合のバッファ占有量（データ本体）

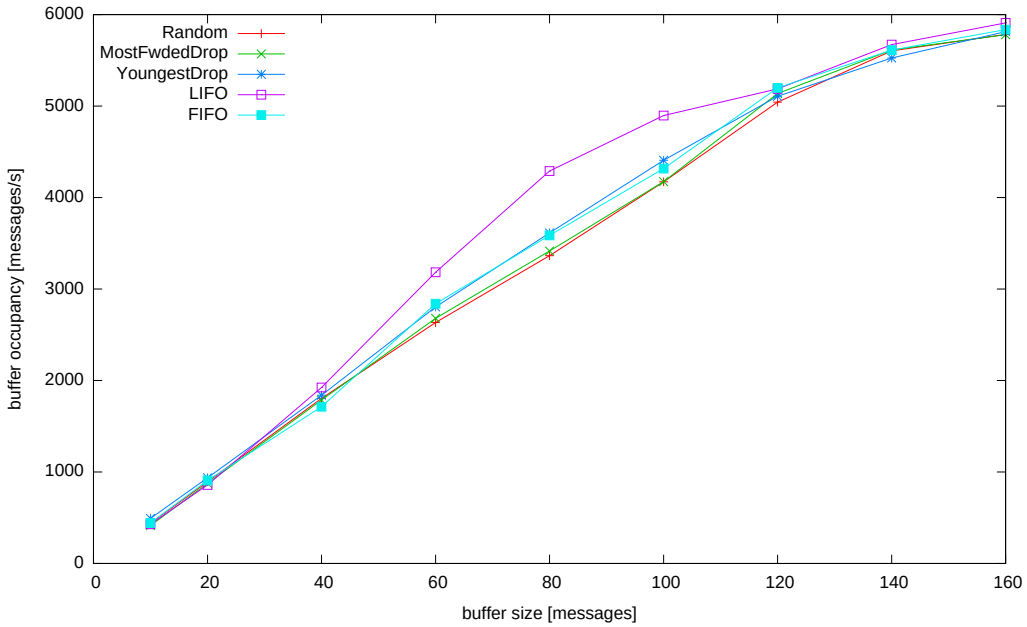

図 5.7　バッファサイズ有限の場合のバッファ占有量（ヘッダのみ）

メッセージ TTL が十分大きく，バッファサイズが無限大で，通信速度が有限の場合

メッセージ TTL が十分大きい場合に，バッファサイズを無限大としたときの通信速度と性能の関係は次のようになった．

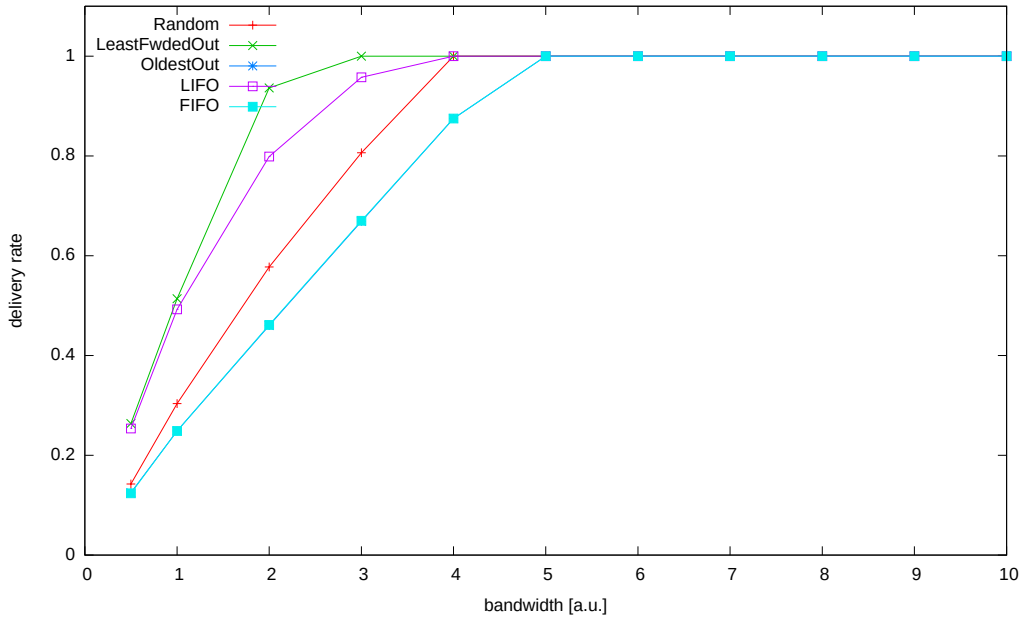

図 5.8　通信速度有限の場合の平均配送率

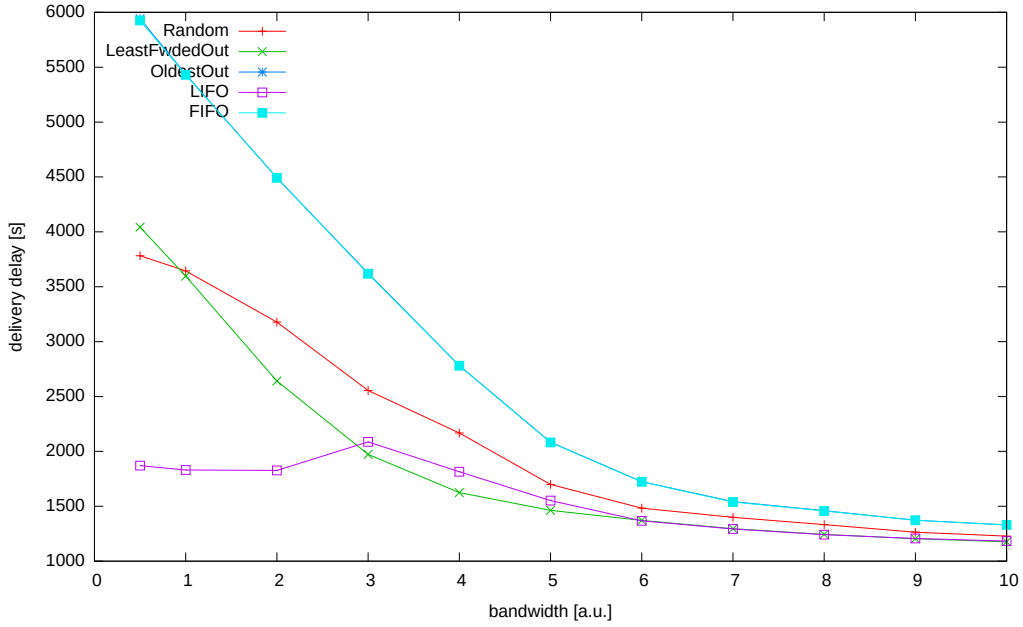

図 5.9　通信速度有限の場合の平均配送遅延

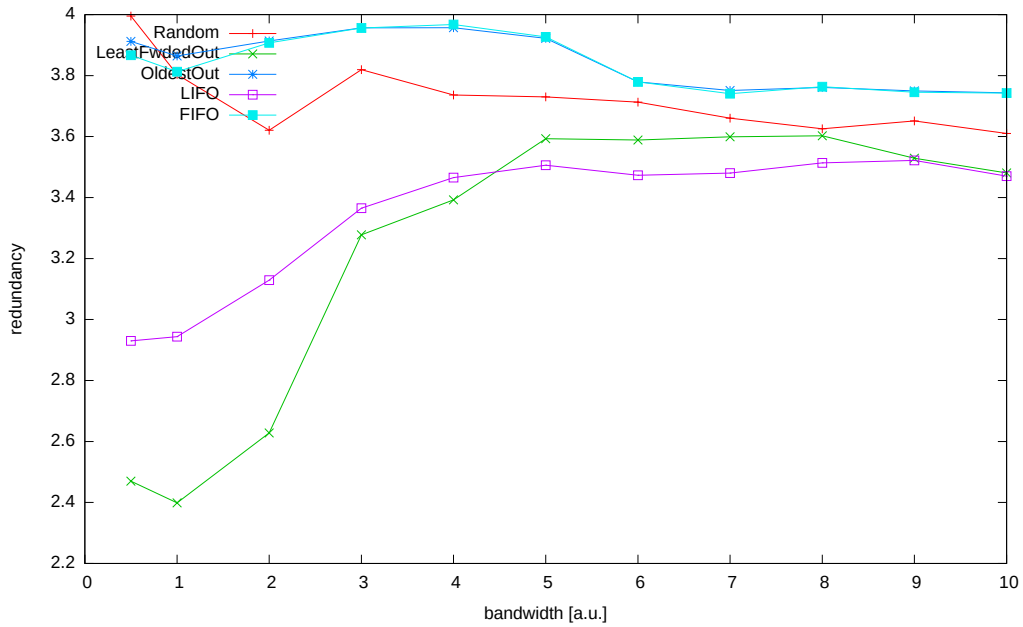

図 5.10　通信速度有限の場合の平均配送冗長度

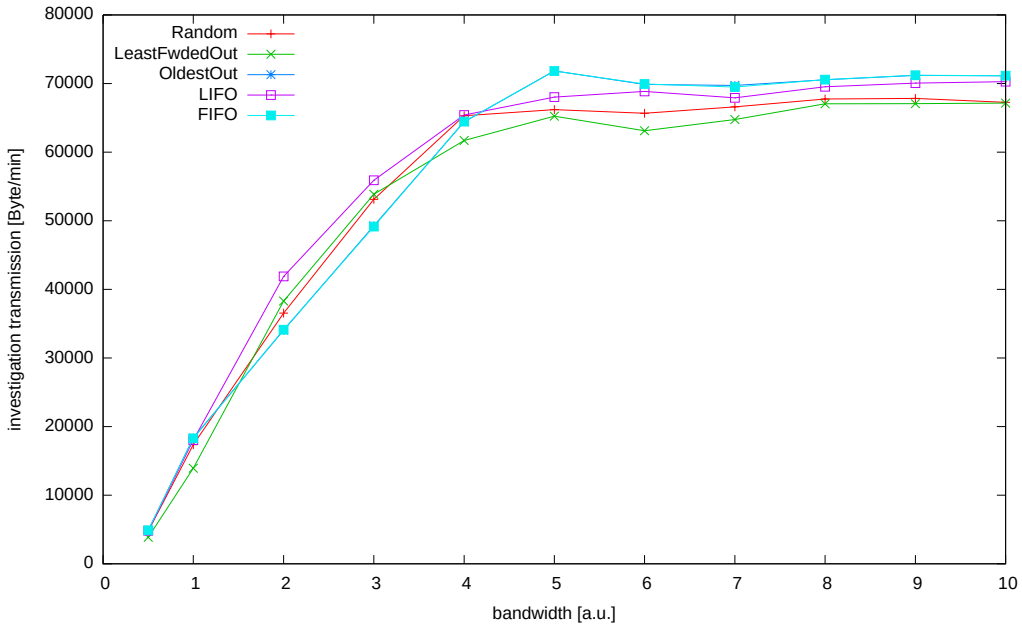

図 5.11　通信速度有限の場合の investigation パケット伝送量

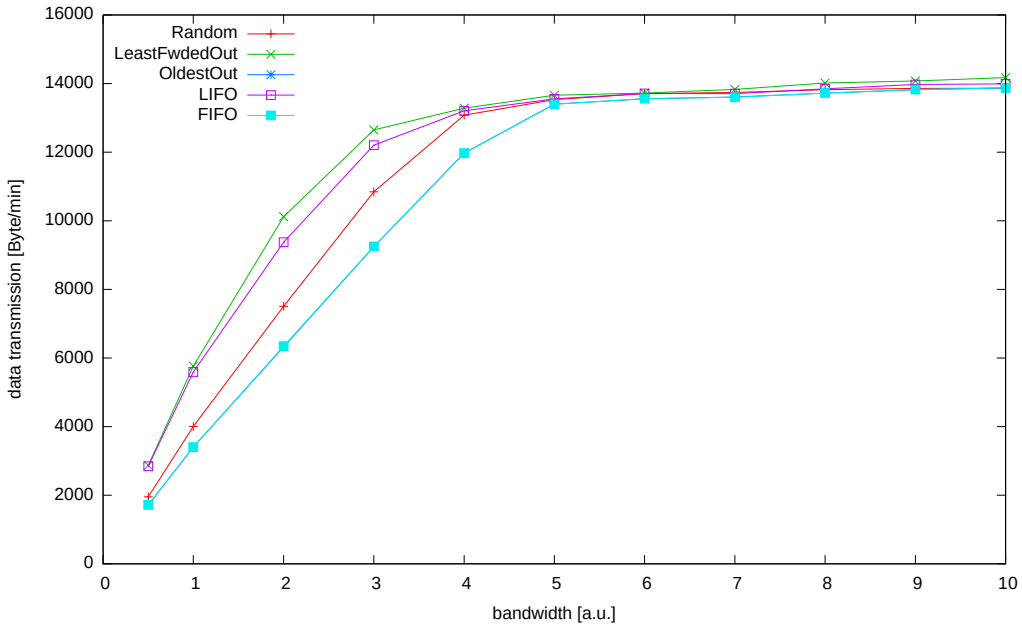

図 5.12　通信速度有限の場合のデータパケット伝送量

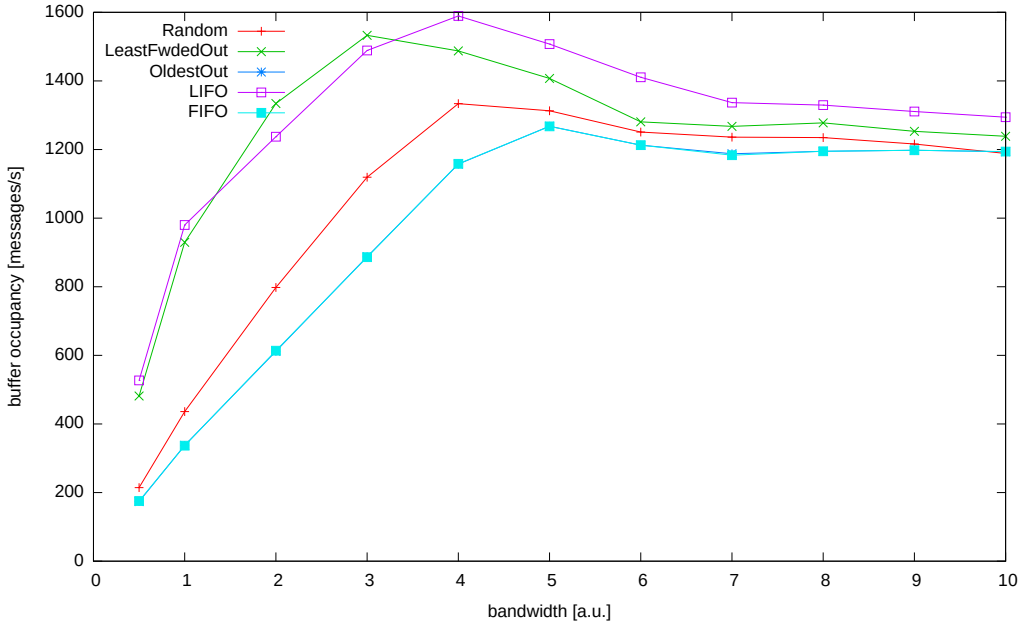

図 5.13　通信速度有限の場合のバッファ占有量（データ本体）

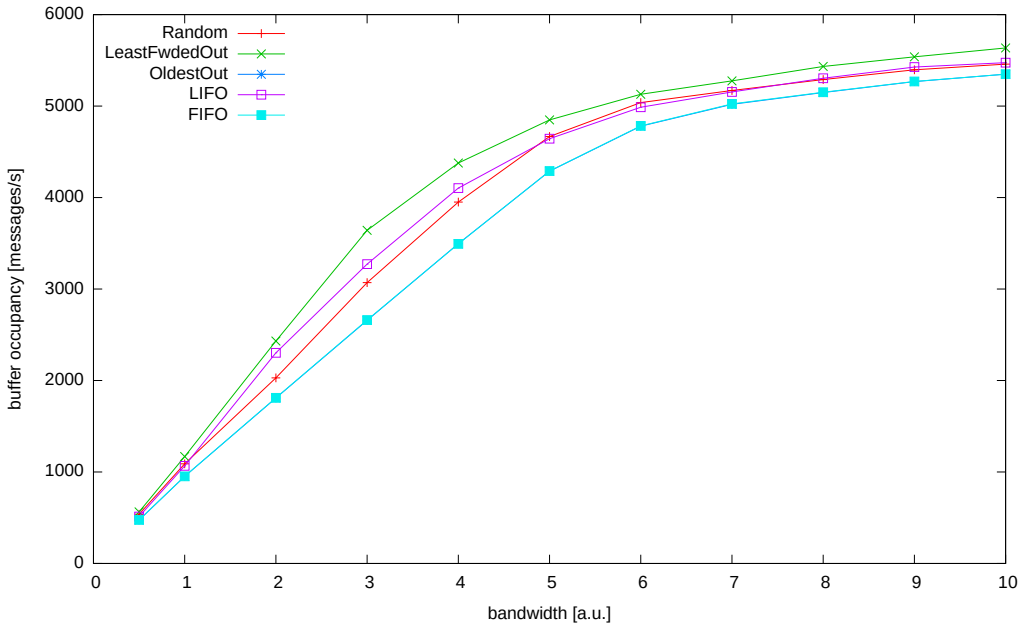

図 5.14　通信速度有限の場合のバッファ占有量（ヘッダのみ）

メッセージ TTL が十分大きく，バッファサイズが有限で，通信速度が有限の場合

メッセージ TTL が十分大きい場合に，通信速度を 2.0 としたときのバッファサイズと性能の関係は次のようになった．

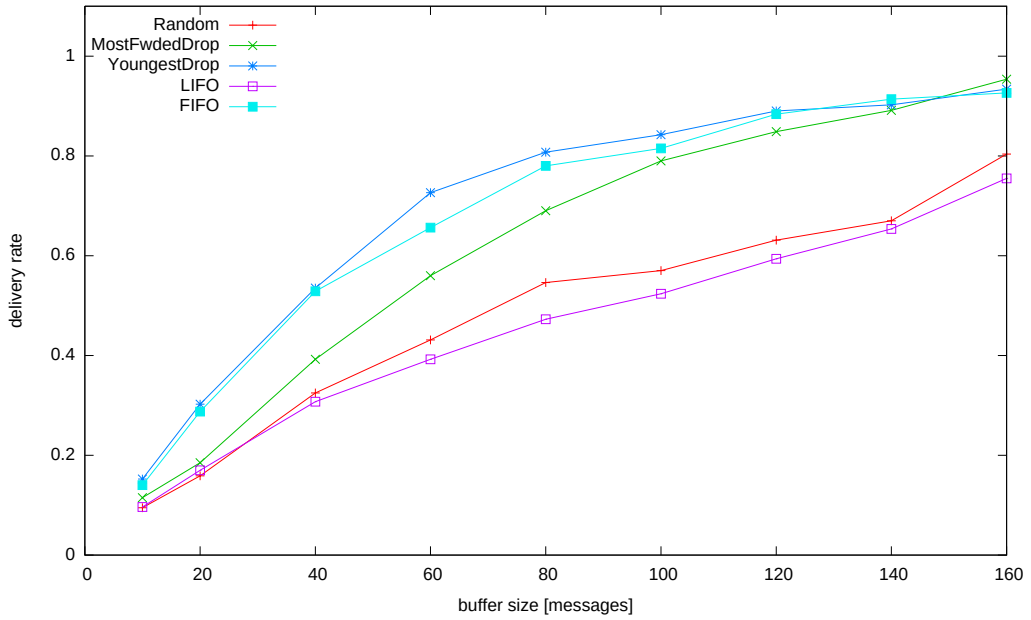

図 5.15　通信速度 2.0 の場合の平均配送率

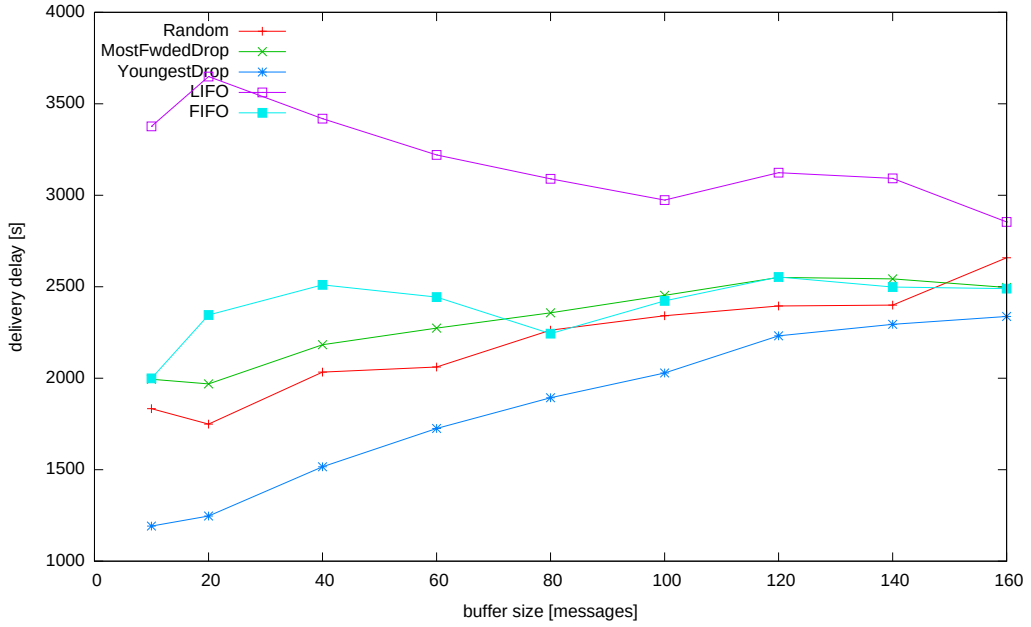

図 5.16　通信速度 2.0 の場合の平均配送遅延

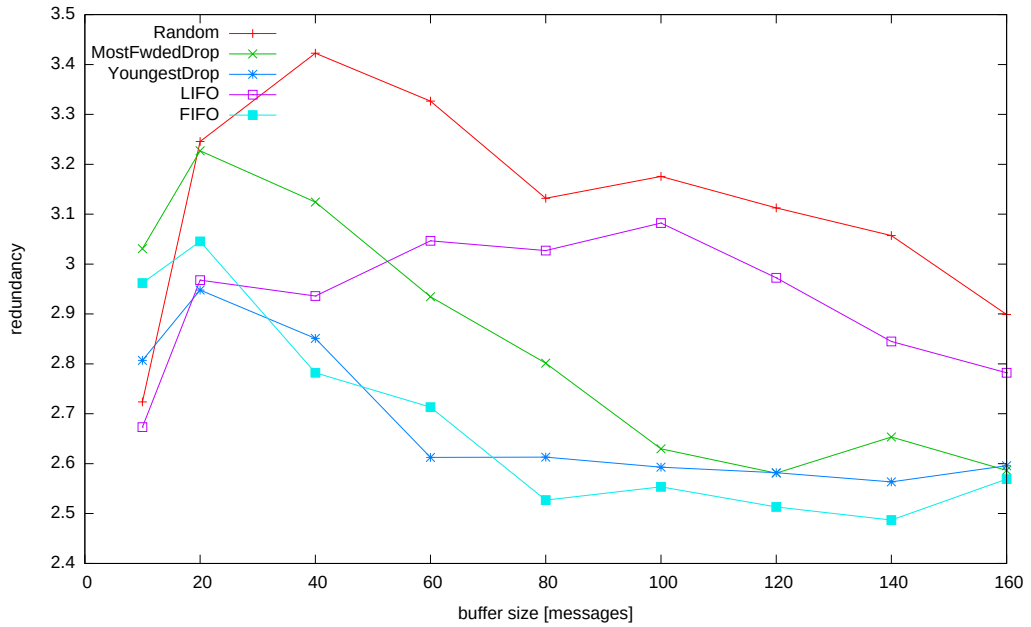

図 5.17　通信速度 2.0 の場合の平均配送冗長度

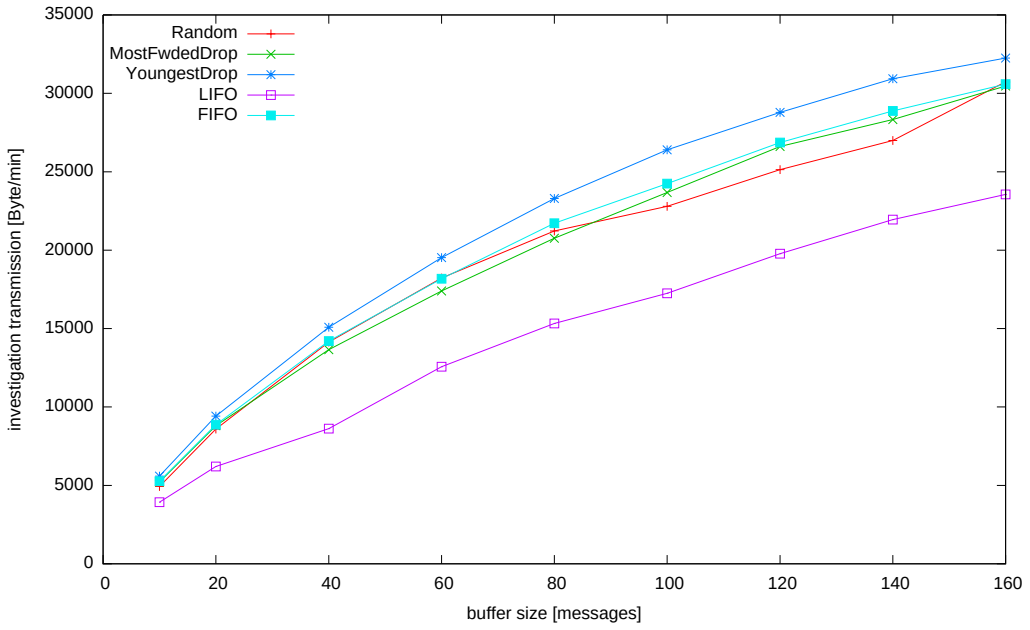

図 5.18　通信速度 2.0 の場合の investigation パケット伝送量

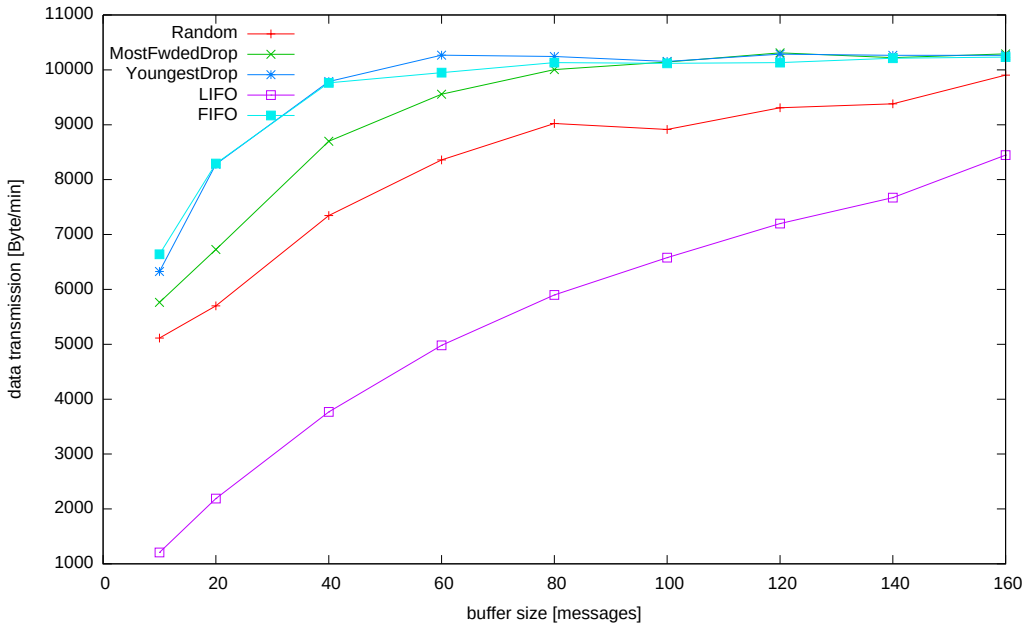

図 5.19　通信速度 2.0 の場合のデータパケット伝送量

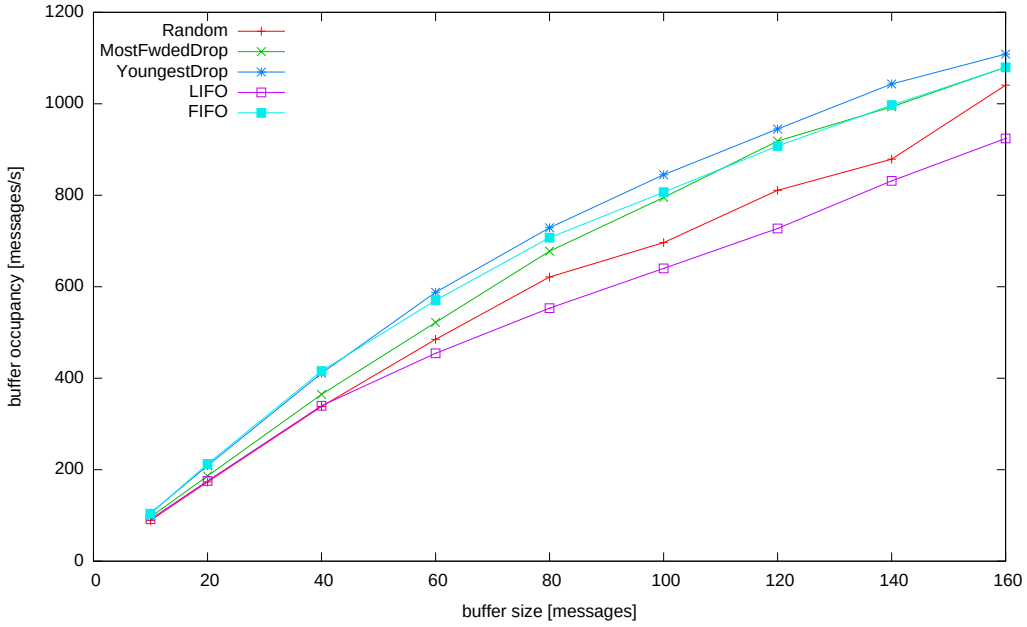

図 5.20　通信速度 2.0 の場合のバッファ占有量（データ本体）

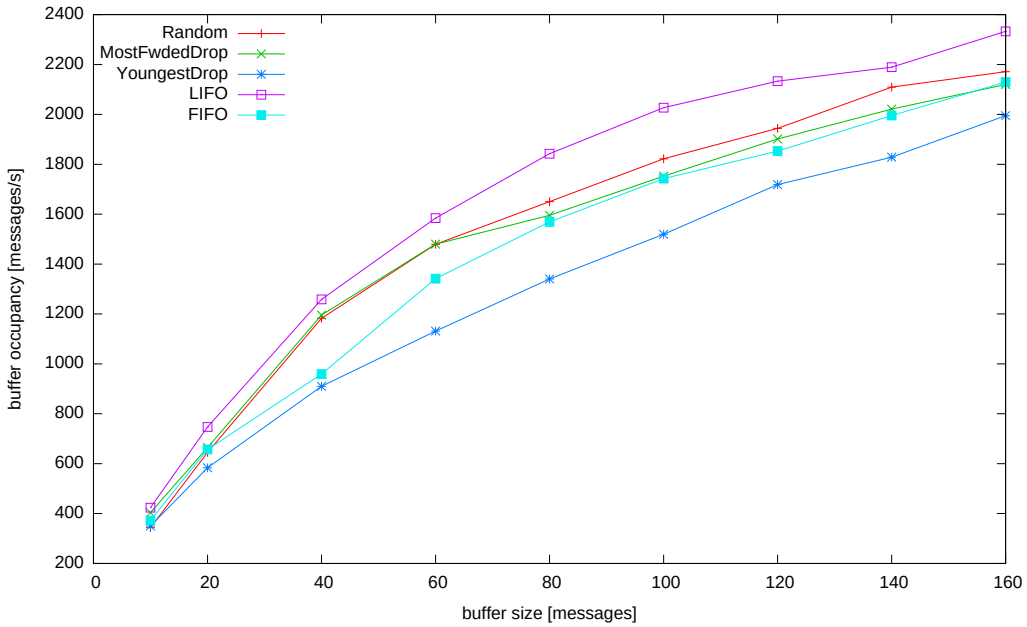

図 5.21　通信速度 2.0 の場合のバッファ占有量（ヘッダのみ）

5.2　メッセージ TTL による配送率とバッファ使用量の変化

メッセージ TTL を変化させたときの配送率とバッファ使用量は図 5.22 のようになった．メッセージ TTL が小さくなるにしたがってバッファ使用量は少なくなっているが，メッセージ配送率はメッセージ TTL が 2200[s] 以上であれば 100% を保っている．メッセージ TTL が 2200[s] であれば，4000[s] のときと比べバッファ使用量が 15% 減少し，メッセージ TTL が 1600[s] であればメッセージ配送率はおよそ 90% になるが，バッファ使用量は 26% 減少した．

また，配送率を 99% に近づけるようにメッセージ TTL を調節したところ，メッセージ TTL の平均は 2252[s] となり，メッセージ配送率は 98.0% となり，バッファ使用量はメッセージ TTL が十分大きい場合に比べ 12.9% 減少した．配送率を 90% に近づけるようにメッセージ TTL を調節したところ，メッセージ TTL の平均は 1936[s] となり，メッセージ配送率は 91.3% となり，バッファ使用量はメッセージ TTL が十分大きい場合に比べ 21.2% 減少した．

これらのことから，メッセージ TTL を適切に設定すると高い配送率を維持しながらバッファ使用量を削減することが可能であり，送信ノードがメッセージ配送率を把握できるようにすることで適切なメッセージ TTL を設定することができたと言える．

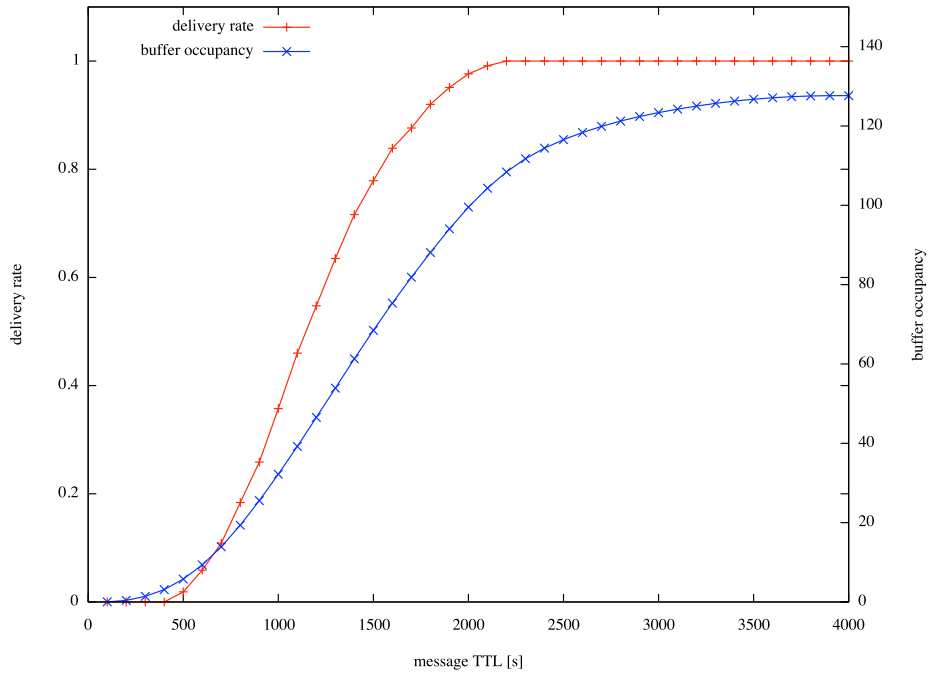

図 5.22　メッセージ TTL による配送率とバッファ使用量の変化

表 5.1　メッセージ TTL による配送率とバッファ使用量の変化

設定した TTL[s]	配送率 [%]	バッファ使用量 [%](TTL が 4000[s] のときを 100% とした場合)
4000	100	100
2200	100	85
1900	95	74
1760	90	67
1600	85	48
1546	80	44

表 5.2　メッセージ TTL を自動設定した場合の配送率とバッファ使用量

目標配送率 [%]	平均 TTL [s]	実際の配送率 [%]	バッファ使用量 [%](TTL が 4000[s] のときを 100% とした場合)
99	2252	98.0	87.1
90	1936	91.3	78.8
80	1729	78.5	57.7

5.3　ポテンシャル構築の改良による配送遅延の変化

ポテンシャル構築の改良によって，配送遅延は表 5.3 のように変化した．合成したモビリティで改良に有利な場合は 14.6% の減少，不利な場合は 3.1% の増加となり，実際のモビリティでは改良に有利な場合は 11.2% の減少，不利な場合は 2.1% の増加となった．実際のモビリティで有利な場合における配送遅延のヒストグラムを図 5.23 と図 5.24 に示す．改良後は，大きな配送遅延のメッセージが減少し，小さな配送遅延のメッセージが増加し，これにより平均配送遅延が低下していることが分かる．また，最大配送遅延も低下していることが分かる．ポテンシャルは，改良前が図 5.25・図 5.26 のようになったのに対し，改良後は図 5.27・図 5.28 のようになった．

表 5.3　ポテンシャル構築の改良による配送遅延の変化

モビリティ		改良前 [s]	改良後 [s]	変化率
合成した	有利	61.5	52.5	-14.6%
モビリティ	不利	256.2	264.1	+3.1%
実際の	有利	108.5	96.4	-11.2%
モビリティ	不利	473.2	483.1	+3.1%

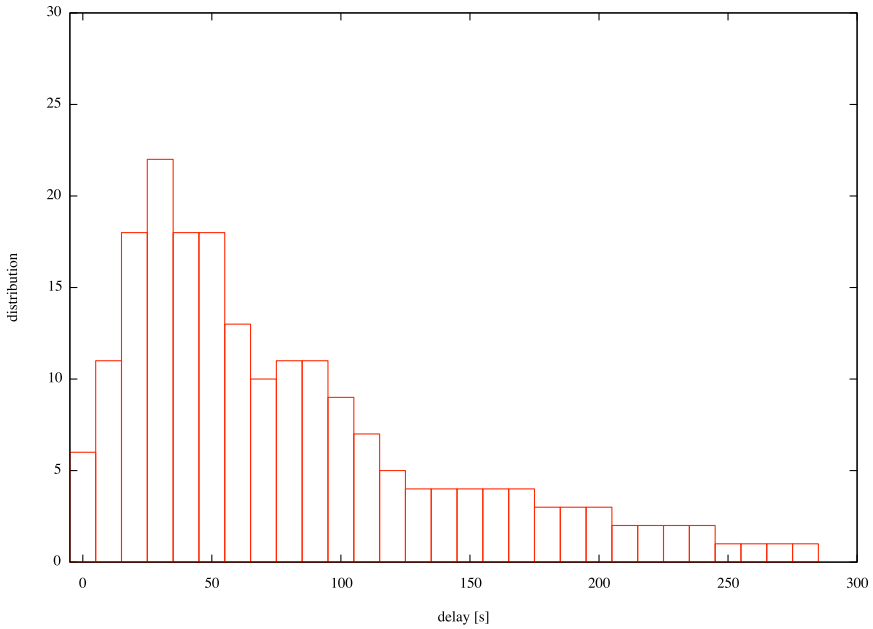

図 5.23　配送遅延のヒストグラム（改良前）

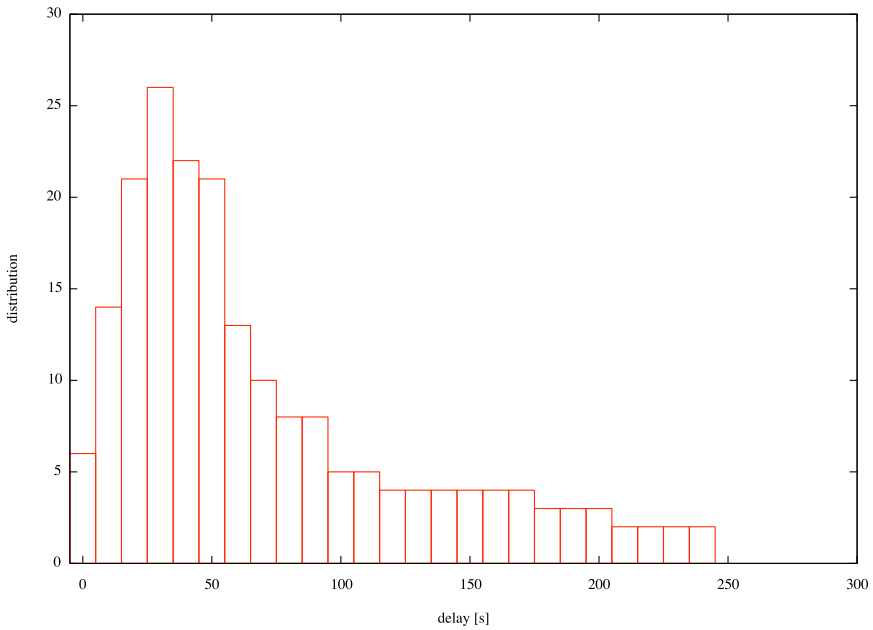

図 5.24　配送遅延のヒストグラム（改良後）

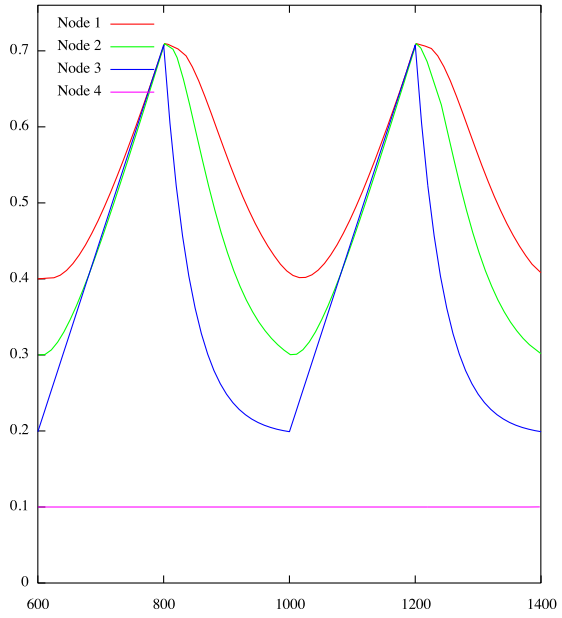

図 5.25　改良前のポテンシャル（改良に有利な場合）

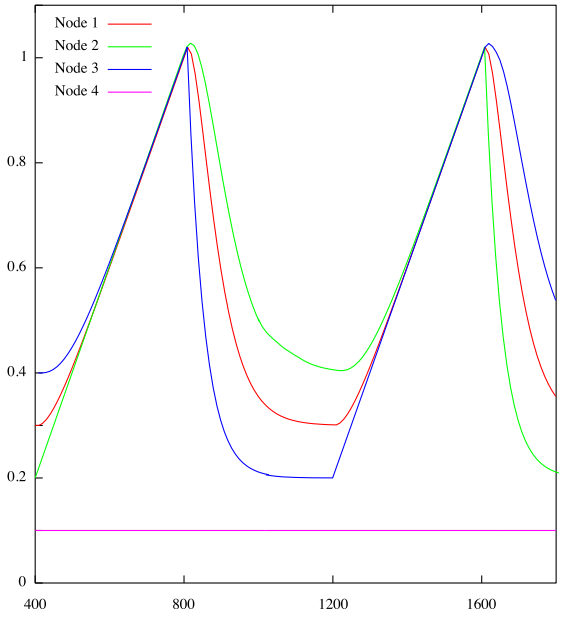

図 5.26　改良前のポテンシャル（改良に不利な場合）

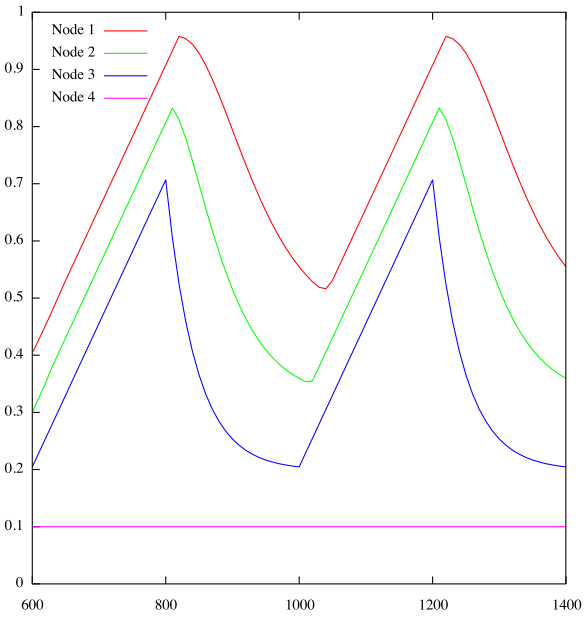

図 5.27　改良後のポテンシャル（改良に有利な場合）

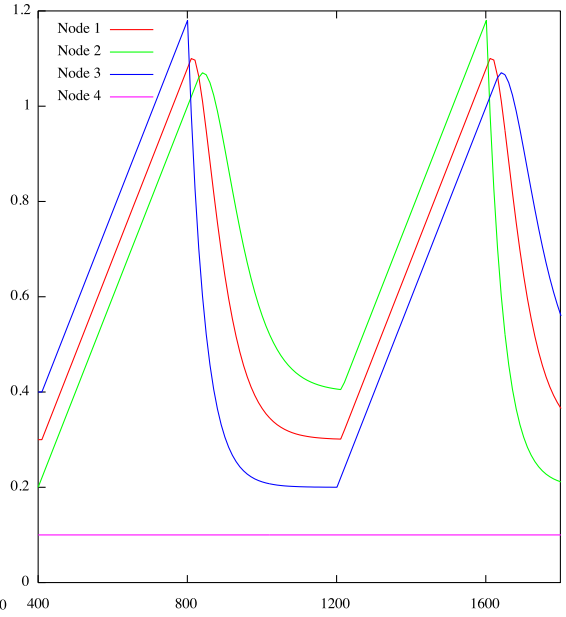

図 5.28　改良後のポテンシャル（改良に不利な場合）

第 6 章

考察

6.1 バッファ管理方式および転送方式の影響

バッファサイズが有限で，通信速度が無限大の場合

通信速度を無限大としたときの，各バッファ管理方式におけるバッファサイズとメッセージ配送率の関係は図 5.1 に示すようになった．全てのバッファサイズにおいて，バッファ管理方式が Youngest Drop のときに配送率が最も高くなった．Youngest Drop は，本来は「TTL の残りが最も多いメッセージは他のノードで転送される機会があるので，TTL の残りが少ないメッセージをバッファに残しておく」という考え方に基づいているが，ここでは十分大きな TTL を用いているので，TTL の残りは転送の機会に影響しない．したがって，TTL の残りが少ないメッセージをバッファに残すことは，先に生成されたメッセージをバッファに残すということを意味し，先に生成されたにもかかわらずまだ宛先へ届いていないメッセージを優先的に保持することで，配送率が高まったということになる．

また，配送遅延も図 5.2 に示すように，全てのバッファサイズにおいて，バッファ管理方法が Youngest Drop のときに最も小さくなった．前述のように，先に生成されたメッセージが優先的に保持されるので，遅延が大きくなるのを防ぐ効果があるということが分かる．なお，LIFO は後から来たメッセージを一切受け付けないため，バッファに空きができるまで新たなメッセージを配送することができず，配送遅延が非常に大きくなったと考えられる．

図 5.3 から分かるように，配送冗長度は LIFO が最も低くなっており，これはメッセージが転送されにくいためであると考えられる．また，Youngest Drop では，バッファサイズによる違いはあるものの配送冗長度は比較的低くなっていることが分かる．

バッファサイズが無限大で，通信速度が有限の場合

バッファサイズを無限大としたときの，各転送方式における通信速度とメッセージ配送率の関係は図 5.8 に示すようになった．いずれの場合も Least Forwarded First Out が最も配送率が高くなった．転送回数が少ないメッセージを優先的に転送することで，全てのメッセージが均等にネットワーク中に広がったため，メッセージ配送率が高くなったと考えられる．逆に，FIFO では毎回同じ順序でメッセージが転送されるので，頻繁に転送されるメッセージとそうでないメッセージが存在するため配送率が低くなったと考えられる．

また，配送遅延は図 5.9 のようになり，通信速度が遅い場合に LIFO の配送遅延が非常に小さくなっ

た．LIFO では後から受け取ったメッセージを先に転送するので，条件によってはメッセージが素早く転送されるためであると考えられる．通信速度が 0.5 または 1.0 の場合は，LIFO と Least Forwarded First Out で配送率がほぼ同じなので，このときは LIFO の方が性能が良いと言える．

配送冗長度は図 5.10 のようになり，通信速度が遅い場合は Least Forwarded First Out のときに小さくなった．前述のように Least Forwarded First Out ではメッセージが均一に転送されるため，過剰な転送が抑制されて冗長度が低く抑えられたと考えられる．

バッファサイズが有限で，通信速度が有限の場合

通信速度を 2.0 としたときの各バッファ管理方式におけるバッファサイズとメッセージ配送率の関係は図 5.15 に示すようになった．通信速度が 2.0 のときは図 5.8 から分かるように配送率は最大で 0.93 なので，全てのバッファサイズで配送率は 0.93 以下となっている．また，通信速度が無限大の場合と同様に，ほぼ全てのバッファサイズで Youngest Drop の場合が最も高い配送率となっているが，次いで FIFO と Most Forwarded First Drop の配送率が高くなっている．Youngest Drop と FIFO における，バンド幅が無限大の場合と 2.0 の場合の配送率を図 6.1〜図 6.2 に示す．

図 6.2 から分かるように，FIFO の場合はバッファサイズが比較的小さいときはバンド幅が無限大よりも 2.0 の方が配送率が高くなった．これは，通信速度 2.0 の場合は，通信速度が無限大の場合と比べノードが新たに受け取るメッセージの数が少ないため，バッファからドロップされるメッセージが少なくなり，バッファに保持されているメッセージが転送される機会が増えて配送率が高くなったと考えられる．従って，通信速度がある程度遅い場合は FIFO や Most Forwarded First Drop によるバッファ管理でも適切であるが，通信速度にかかわらず高い配送率を実現できる Youngest Drop が最も良いバッファ管理方式であると言える．

配送遅延と配送冗長度は，それぞれ図 5.16 および図 5.17 に示すようになった．通信速度が遅いため配送遅延も大きくなっている．配送冗長度は，変化はあるものの全体的には概ね同じような値をとっている．

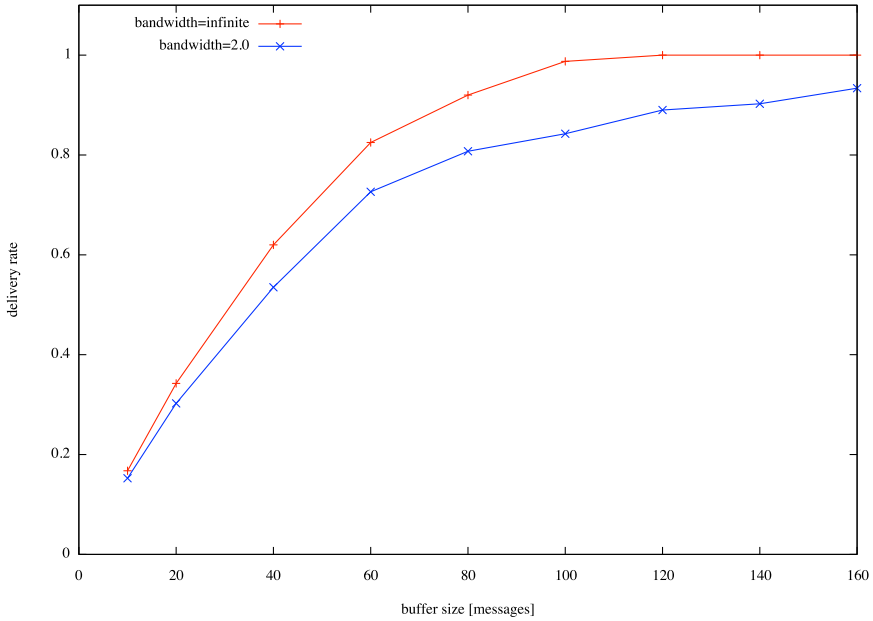

図 6.1　Youngest Drop におけるバンド幅による配送率の変化

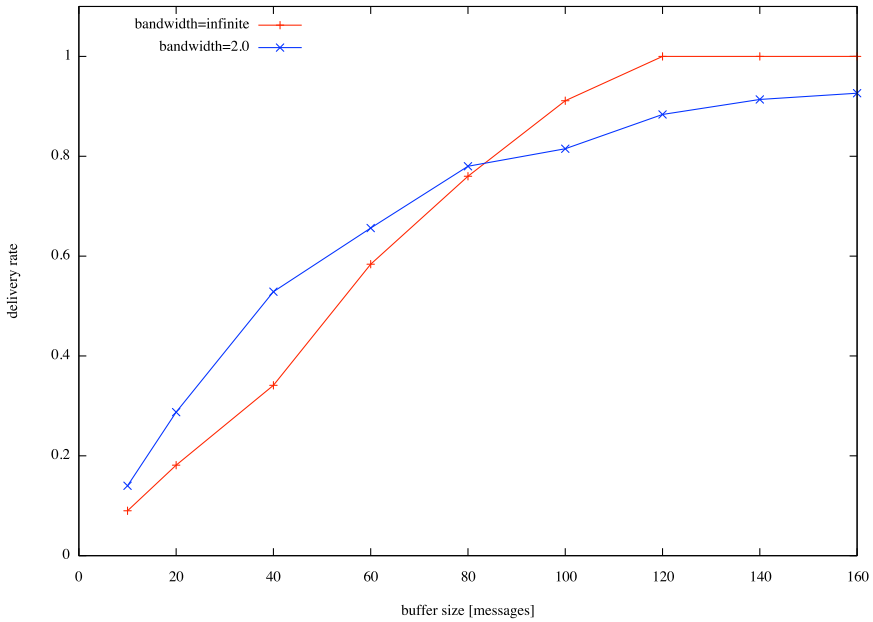

図 6.2　Youngest Drop におけるバンド幅による配送率の変化

6.2　他ルーティングプロトコルとの比較

PEAR と Epidemic および Binary Spray and Wait におけるルーティング性能を比較した．条件は次のようにした．

- バッファ管理方式: Youngest Drop
- 転送方式: Least Forwarded First Out
- バンド幅: 2.0
- バッファサイズ: 10〜160 [messages]

比較した結果を図 6.3〜図 6.9 に示す．図 6.3 はメッセージ配送率の比較で，PEAR の配送率が最も高く，次いで Epidemic の配送率が高い．配送遅延は図 6.4 に示したようになり，Epidemic が最も小さく，PEAR が最も大きい．PEAR が最も配送率が高いことを考慮すると，大きな遅延を許容できる場合は PEAR の方が良いが，PEAR と Epidemic の配送率の差はわずかなので，高い配送率と小さい配送遅延を両立できるのは Epidemic であると言える．配送冗長度は図 6.5 のようになり，バッファサイズが非常に小さい場合以外は PEAR が最も低い．データパケット伝送量は図 6.7 のようになり，Epidemic が最も多く，バッファサイズが小さい場合を除いては PEAR が最も低い．また，データ本体によるバッファ占有量は図 6.8 のようになり，PEAR が最も少なく，Epidemic と Binary Spray and Wait はほぼ同じであるがバッファサイズが大きいときは Binary Spray and Wait が最も多かった．これは，Epidemic の方がメッセージ配送完了の ack が伝わるのが速く，配送済みのメッセージが消去されたために Binary Spray and Wait よりもバッファ占有量が少なくなったと考えられる．

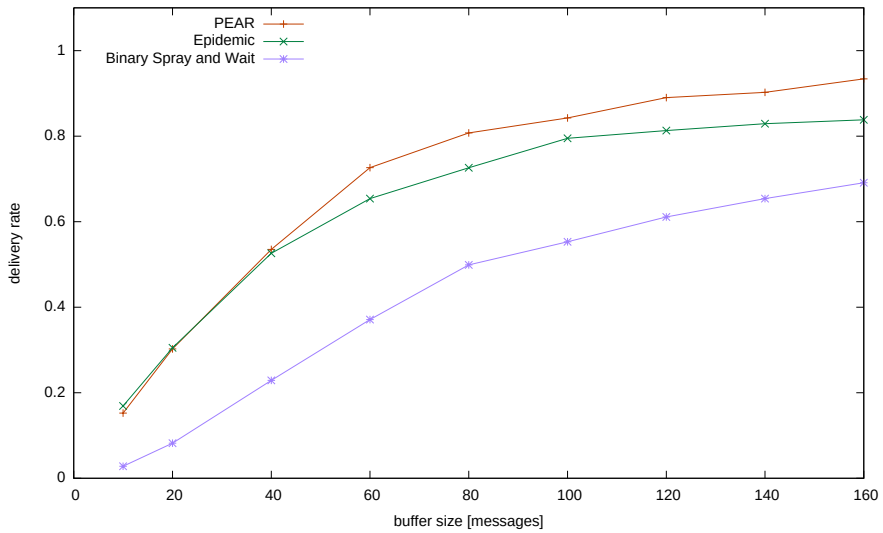

図 6.3　メッセージ配送率の比較

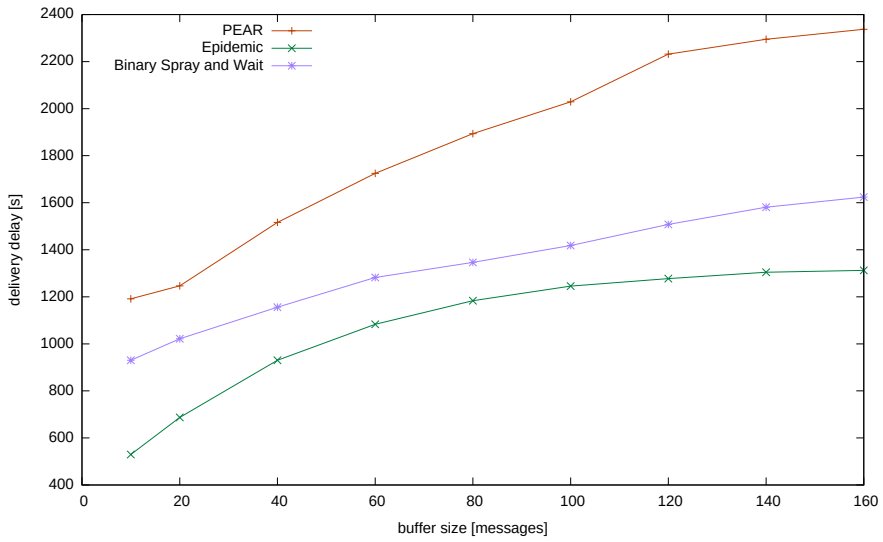

図 6.4　平均配送遅延の比較

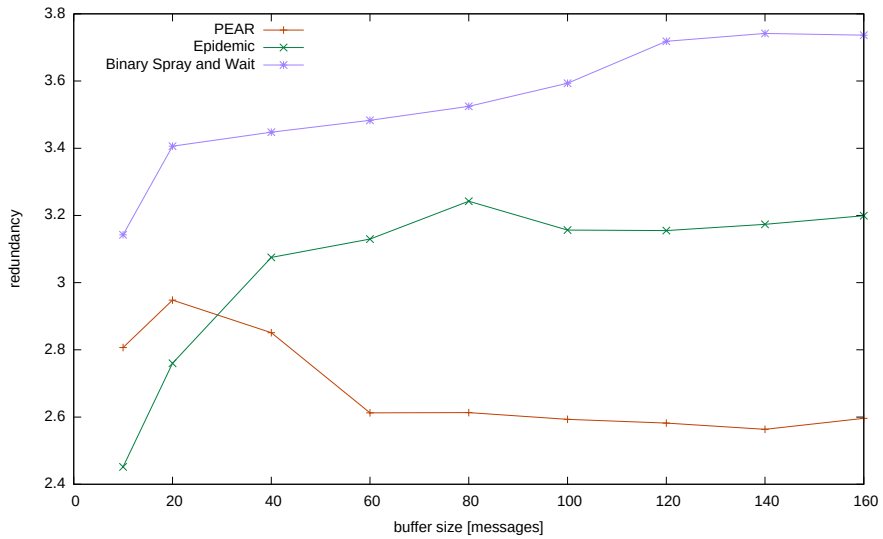

図 6.5　平均配送冗長度の比較

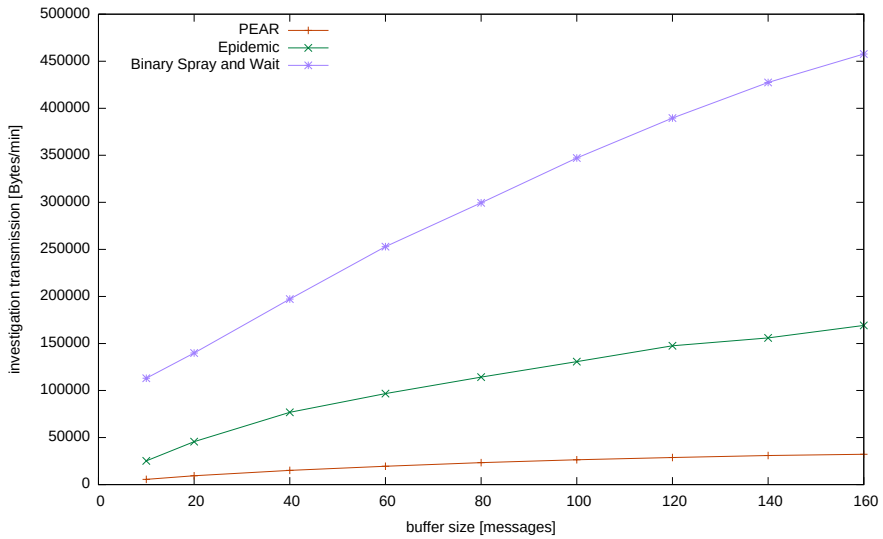

図 6.6　investigation パケット伝送量の比較

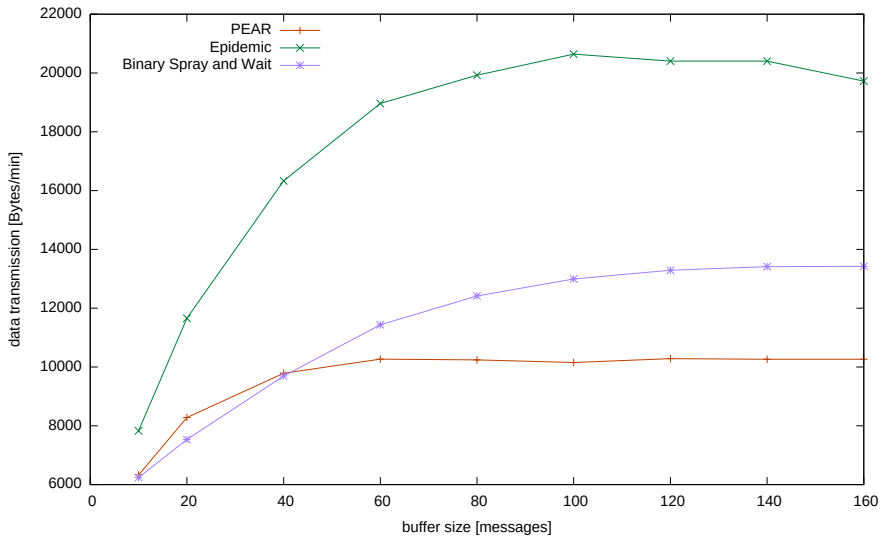

図 6.7　データパケット伝送量の比較

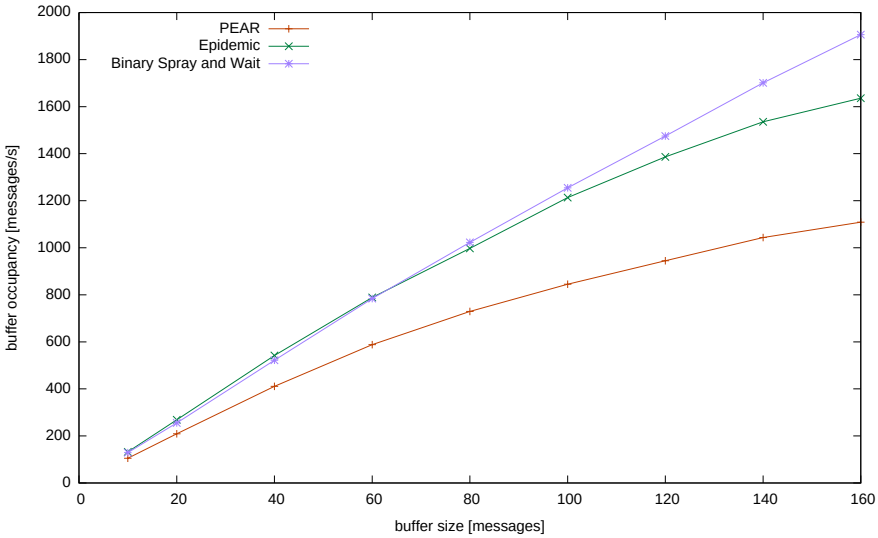

図 6.8　バッファ占有量（データ本体）の比較

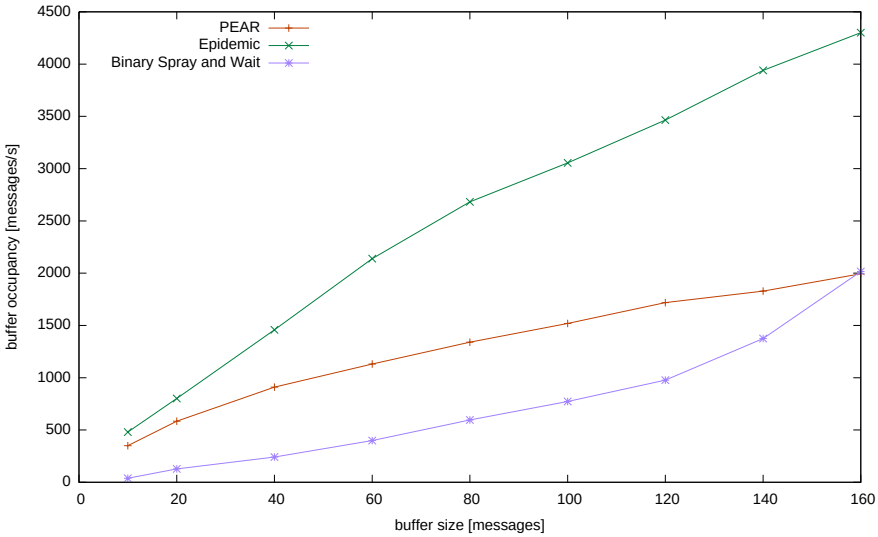

図 6.9　バッファ占有量（ヘッダのみ）の比較

6.3 メッセージ TTL による配送率とバッファ使用量の変化

メッセージ TTL を変化させたときの配送率とバッファ使用量は図 5.22 のようになった．メッセージ TTL が大きくなると，バッファ使用量は増加するものの，増加の仕方は緩やかになり，ほぼ収束した．これは，メッセージが宛先ノードに届いた acknowledgement によってメッセージがバッファから削除されるためである．メッセージ TTL が大きくても，acknowledgement によりバッファから削除されるので，バッファ使用量は増加しない．一方，メッセージ TTL が小さくなると，配送完了前に TTL が 0 になるメッセージが増加し，バッファ使用量が減少するとともにメッセージ配送率が低下した．したがって，適切にメッセージ TTL を設定すれば，配送率を 100% に保ちつつバッファ使用量を削減することができ，配送率の低下を許容すればさらに削減することができることが分かった．

宛先ノードから伝えられた配送率に基づいてメッセージ TTL を調整すると，表 5.2 のようになった．ある程度の誤差はあるものの目標配送率とほぼ同じ配送率を実現することができ，バッファ使用量を削減できていることを確認した．削減量は表 5.1 に示されている値よりは少ないが，これはメッセージ TTL を調整しているため結果的に平均の TTL が表 5.1 の値よりも大きくなったためである．配送率を必要最小限に抑えることでバッファ使用量を削減できることが実際に確認できた．

6.4 ポテンシャル構築の改良

ポテンシャル構築の改良による配送遅延の変化は表 5.3 のようになり，改良が有効に働く場合は合成したモビリティで 14.6% の減少，実際のモビリティで 11.2% の減少となり，一方，不利な場合には合成したモビリティで 3.1% の増加，実際のモビリティで 2.1% の増加にとどまった．不利な場合というのは，リンクが切れる前と再びつながったときとでトポロジが異なる場合であり，そのような場合は新しいトポロジに応じてポテンシャルの順序を変更するため，配送遅延が増加する．しかし，ポテンシャルが降下しながら順序が入れ替わる場合，降下の速度は一定ではなく式 3.3 に従って急速に降下する．そのため，順序の入れ替わりに要する時間は長くはない．したがって，ポテンシャル構築の改良により配送遅延が増加してしまう場合もあるがその影響は小さく，配送遅延が減少するメリットの方が大きいと言える．

ポテンシャルは，改良前が図 5.25・図 5.26 のようになったのに対し，改良後は図 5.27・図 5.28 のようになった．改良前は接続が切れている間にノード 1・2・3 のポテンシャルが上昇しながら等しくなったのに対し，改良後は接続が切れる前の順序を維持しながらポテンシャルが上昇した．改良に対して有利な場合はポテンシャルの順序が入れ替わる必要がないので常に同じ順番を維持しているのに対し，改良に対して不利な場合はポテンシャルの順序が入れ替わるために 20 秒の時間を要したが，前述のように配送遅延への影響は小さかった．

第 7 章

IP over DTN

7.1 IP over DTN の背景

DTN はバンドル層を導入してオーバーレイネットワークを構築することにより，惑星間通信 [1] や発展途上国の村同士での通信 [2]，アドホックネットワーク [26] でのパフォーマンスを向上させようという試みが行われている．しかし，この方法ではインターネット全体の上に新たにオーバーレイネットワークを構築する必要があり (図 7.1)，実現が非常に難しい．

そこで，本研究ではインターネットに容易に導入することができる IP over DTN アーキテクチャーを提案する．IP over DTN では，IP パケットが DTN によって伝送される (7.2)．すなわち，DTN はデータリンク層として機能する．また，上位層のアプリケーションは非同期で動作することを前提としている．一般的な DTN として RFC4383[3] や DTNRG[27] があるが，本研究で用いる DTN フレームワークは，1500 バイト程度の IP パケットを配送することだけを目的とした簡便な設計となっている．したがって，IP over DTN で大きなサイズのデータを送るためには，トランスポート層またはアプリケーション層で分割・再結合を行わなければならない．

遅延が大きく切断が頻繁に発生する環境では，そもそも同期での通信が不可能なので，非同期で通信を行わざるを得ない．IP over DTN では，アプリケーションは非同期で通信を行い，IP ネットワークがもともと備えている非同期性を利用する．したがって，IP over DTN ではアプリケーションは UDP ベースの非同期なプロトコルで通信を行う．Burleigh[1] らは惑星間通信での UDP ベースでの通信を議論したが，ack や再送要求などを行う，同期通信を仮定するものであった．IP マルチキャストでは，多点間での同期が困難であるために TCP ベースでの接続は行わない [24]．この点で IP over DTN も同様であるが，IP マルチキャストはネットワーク全体で管理を行わなければならないため [25]，ローカルな管理ができる IP over DTN は IP マルチキャストより導入が容易である．

IP ネットワークはベストエフォート型のサービスであり，遅延やパケットロスは生じてもよいことになっている．DTN をデータリンク層として用いると，IP パケットは DTN ノードの中に蓄積され，ノードの物理的な移動によって運ばれ，途中で大きな遅延が生じても宛先のネットワークへ転送される．

アプリケーションには同期と非同期の 2 種類の方式がある [28]．同期方式では，双方のアプリケーションが同時にオンラインになっており，相互に通信を行う．一方，非同期方式では，双方のアプリケーションが同時にオンラインになる必要はなく，送信側は受信側がオフラインの状態でもメッセージ

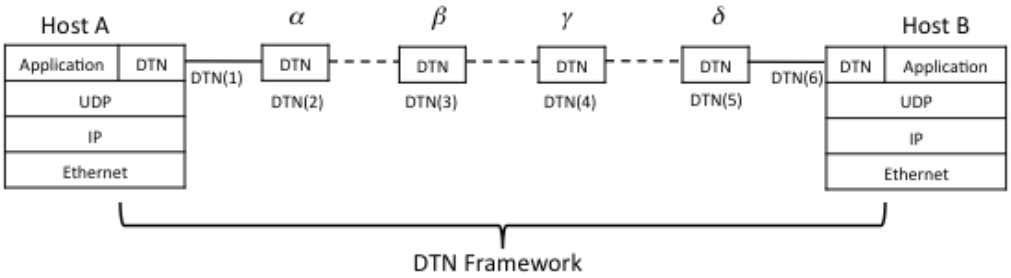

図 7.1　DTN オーバーレイネットワークのプロトコルスタック

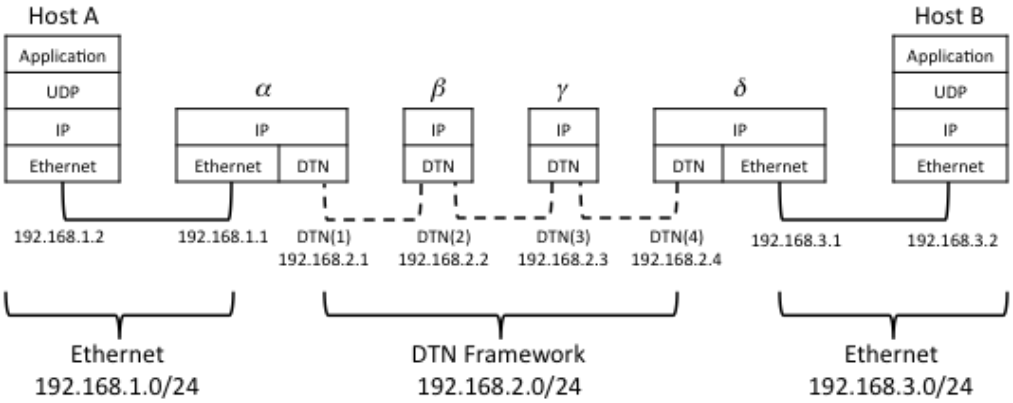

図 7.2　IP over DTN のプロトコルスタック

を送ることができ，受信側は後に送信側がオフラインになった後でもネットワークからメッセージを受け取ることができる．

大きな遅延が生じるネットワークでは，アプリケーションが TCP などの同期方式で通信しようとすると失敗する [29]．すなわち，遅延が大きいネットワークでは，アプリケーションは同期方式ではなく非同期方式をとらなければならない．既存の DTN は，オーバーレイネットワークを作り，エンドツーエンドのセッションをホップごとのセッションに分割することにより非同期で通信を行う．一方，IP over DTN は，IP ネットワークがもともと備えている非同期での転送能力を利用して，エンドツーエンドの同期を避けている．したがって，IP over DTN のアーキテクチャーは UDP などの非同期のプロトコルのみの使用に限定される．この制約は，アプリケーションの設計を大きく制約するものではないと言える．

7.2　IP over DTN の実装

Linux 上で IP over DTN の実装を行った．まずパーソナルコンピュータ上で実装し，動作および性能の検証を行った．次に組み込み CPU ボード上で実装し，小型のモバイルノードを作成した．

7.2.1　IP over DTN のアーキテクチャ

IP パケットを DTN のメッセージとして配送するために，仮想ネットワークインターフェースを用意し，それが受け取った IP パケットを DTN のメッセージでカプセル化して配送するようにした．受信側は，受け取った DTN メッセージから IP パケットを取り出し，仮想ネットワークインターフェースを通してアプリケーションへ渡すようにした．実装の概略を図 7.3 に示す．無線 LAN によって接続されているノード上で DTN フレームワークが動作し，その上に IP ネットワークが構築されている．

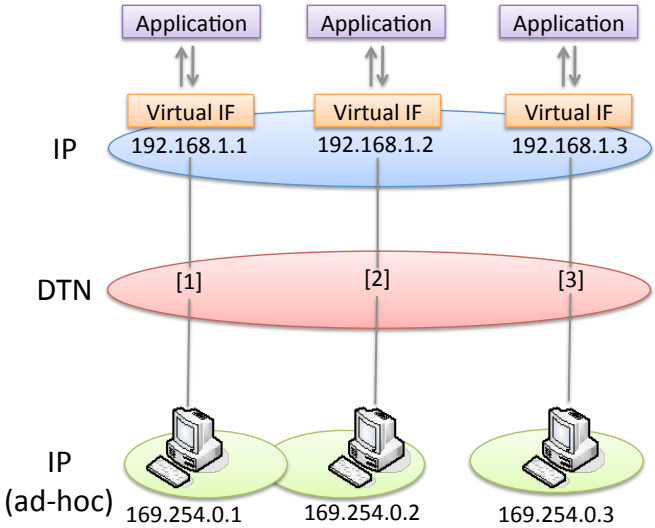

図 7.3　IP over DTN の実装の概略

IP パケットを DTN で配送する際，IP パケットは宛先を IP アドレスで指定するが，DTN は宛先を DTN 専用の ID で指定しなければならない．そのため，各ノードが他のノードにおける IP アドレスと ID の対応関係をあらかじめ知っておかなければ IP パケットを宛先へ届けることができない．そこで，次の手法で IP アドレスと ID の対応関係をネットワーク全体に伝搬させることにした．

IP アドレスと ID の対応関係を保持するためのテーブルを用意し，自分自身の IP アドレスと ID は分かっているので，この情報をテーブルに記録する．次に，他のノードと通信可能になった際に，互いのテーブルを参照して，自分は持っていないが相手は持っている情報をコピーする．すなわち，各ノードは自分が持っていない情報を他のノードからもらい，自分が持っている情報を他のノードに与えることにより，テーブルが保持する情報を増やしていく．これを繰り返すことにより，ネットワーク全体で IP アドレスと ID の関係を共有することができる．この具体例を図 7.4 に示す．初期状態は図 7.4(a) のように自分自身の IP アドレスと ID のみが保持されている．ノード 1 とノード 2 が通信可能になると，図 7.4(b) に示すようにテーブルが更新される．次に，ノード 2 とノード 3 が通信可能になると，図 7.4(c) に示すようにテーブルが更新される．そして，ノード 1 とノード 2 が再び通信可能になると，図 7.4(d) に示すようにテーブルが更新される．このように，ノード 1 とノード 3 は直接通信することができなくても，ノード 2 を介して互いの情報を交換することができる．

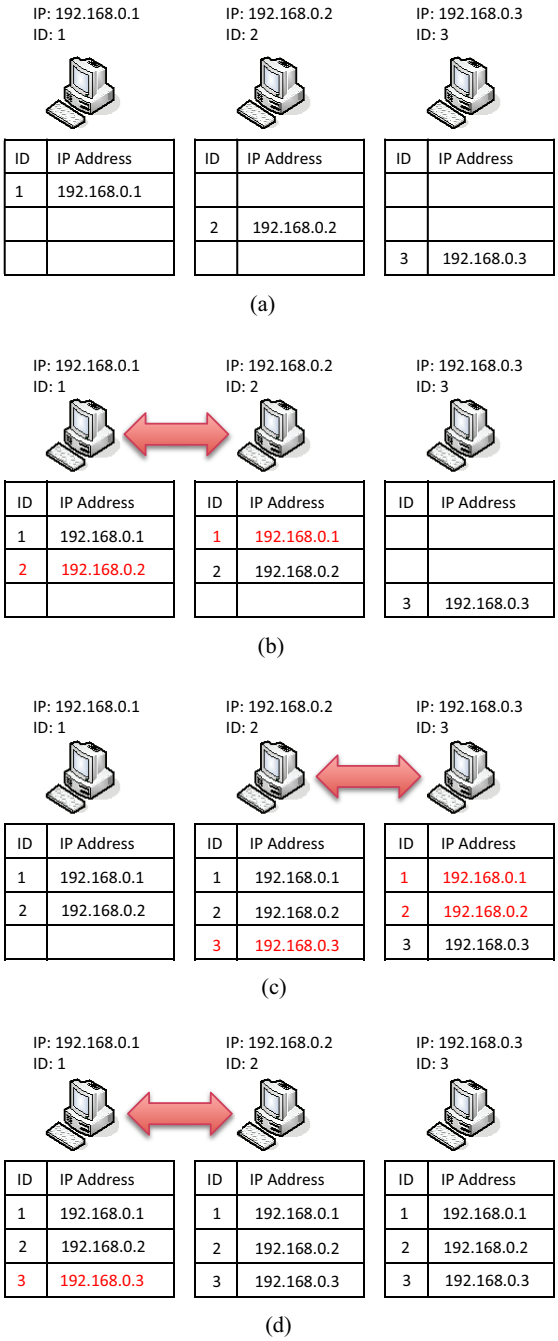

図 7.4　IP–ID 変換テーブル

7.2.2　パーソナルコンピュータ上での実装

7.2.1 節で述べた仕組みをパーソナルコンピュータ上で実装した．OS は Ubuntu Linux 8.04（カーネル 2.6.24-24-generic）を用いた．

実装した IP over DTN の伝送特性を評価するため，ping により Round Trip Time(RTT) を測定した．DTN 環境において，2 台のノードが直接通信可能な場合 (1 hop)，2 台のノードが中継ノード 1 台を経由して繋がっている場合 (2 hops)，2 台のノードが中継ノード 2 台を経由して繋がっている場合 (3 hops) で測定を行った．また，比較のために，無線 LAN でアドホック接続した 2 台間で測定を行った．測定の結果は図 7.5，表 7.1 のようになった．中継ノードの数が増えるに従って RTT も増加していることが分かる．特に中継ノード 2 台 (3 hops) では RTT の最大値が大幅に増加していることが分かった．

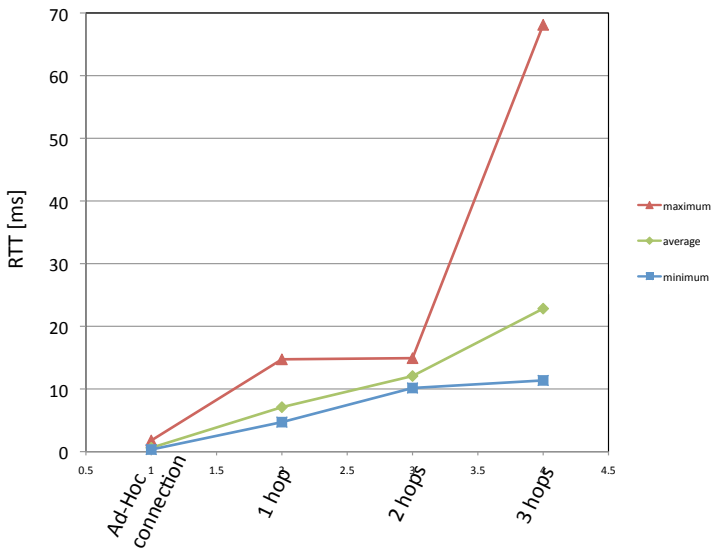

図 7.5　IP over DTN における RTT

表 7.1　IP over DTN における RTT

	RTT [ms]		
	最小	平均	最大
Ad-Hoc	0.332	0.615	1.782
1 hop	4.708	7.096	14.733
2 hops	10.154	12.065	14.917
3 hops	11.363	22.822	68.111

7.2.3　組み込み CPU ボード上での実装

組み込み CPU ボード Armadillo 220[30] に IP over DTN を実装した．Armadillo 220 は ARM9 200MHz CPU を搭載し，Linux が動作する．Linux カーネルは 2.6.12.3-a9-15 を用い，IEEE802.11g の無線 LAN モジュールを USB で接続した．作成したモバイルノードの外観を図 7.6 に示す．

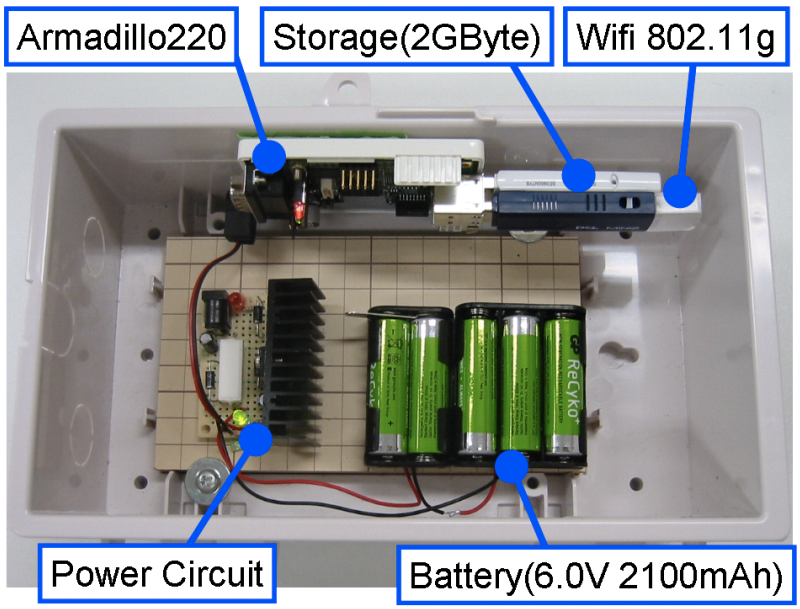

図 7.6　IP over DTN が実装されたモバイルノード

7.3　実験

7.3.1　画像伝送実験

図 7.7 に示すような 5 つのネットワークを構築した．4 つはイーサネット上に構築し，1 つは DTN 上に構築した．それぞれのネットワークは IP ルーターによって接続されている．DTN は 3 つのノード α・β・γ で構成され，IEEE802.11g の無線 LAN のアドホックモードで接続されており，メッセージを交換できるようになっている．ノード α と γ はイーサネットにも接続されており，IP ルーターとしても機能している．ノード α と γ を離れた位置に設置して，その間でノード β を物理的に移動させた．移動の周期は約 10 分とした．

ネットワークに Host A と Host B をインターネットアプリケーションとして接続した．Host B は画像データベースとして機能し，Host A は Host B から画像データを取得するようにした．アプリケーションは大きな遅延を想定して設計し，UDP により実装した．Host A がクエリを送ると，Host B は画像データを 5〜9 個の UDP セグメントに分割して送信する．全体の通信方式は同期方式であるが，クエリの送信と画像データの送信それぞれは非同期方式で行われるようになっている．

1 つのクエリと応答において，実際に得られた IP パケット伝送の様子を 7.8 に示す．青色の矢印はクエリの伝送の様子を表し，赤色の矢印は画像データの伝送の様子を表す．矢印の横の数字は伝送された時刻を表し，括弧内の数字は同時に伝送されたパケットの数を表す．時刻 0[sec] に Host A は Host B に対するクエリを送信し，このクエリはノード α に蓄えられた．時刻 162[sec] においてノード β がノード α と通信可能になり，クエリを受け取った．時刻 447[sec] においてノード β がノード γ と通信可能になり，クエリがノード β からノード γ へ送られ，イーサネットを通じて Host B へ届けられた．同じ時刻に，Host B は画像データを 8 個の UDP セグメントに分割して送信し，ノード γ を経由してノード β に蓄えられた．時刻 766[sec] にノード β が再びノード α と通信可能になり，7 個のパケットが送られ，時刻 776[sec] に残り 1 個のパケットが送られた．最後の 1 個のパケットは時刻 766[sec] にも送信が行われたが，ノード α が受信に失敗したため，10 秒後に再び送信された．このようにして，Host A はクエリを送信してから 776 秒後に画像を取得した．　この実験で Host A は画像データを取得するまでに 776 秒の待機を要したことからも明らかなように，IP over DTN 上で動作するアプリケーションはパケットの配送の遅延を想定しなければならない．したがって，プロトコル設計の早い段階で大きな遅延を考慮しなければならない．しかし，一般的な DTN のアプリケーションも遅延を想定しなければならないので，このことは IP over DTN だけでの問題ではない．

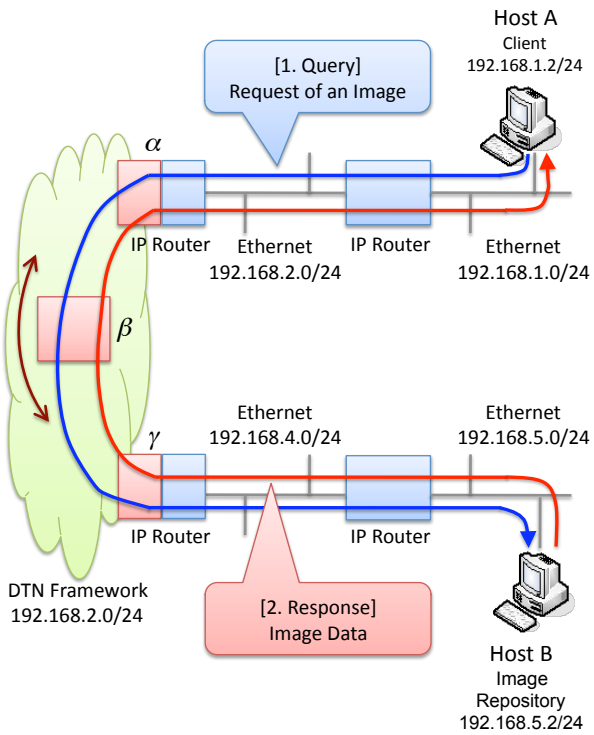

図 7.7　画像伝送実験の概要

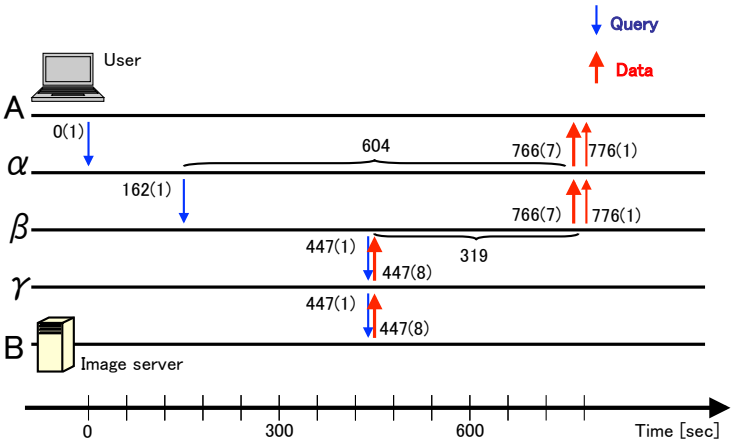

図 7.8　パケット配送の様子

7.3.2　センサデータ回収実験

センサノードからデータを回収する状況を想定した実験を行った．ノードを固定して設置した場合で 2 通り，ノードを移動させた場合で 2 通りの実験を行った．以下，それぞれ (1) StaticA, (2) StaticB, (3) DynamicA, (4) DynamicB と呼ぶ．

実験のネットワーク構成は図 7.9 のようになっており，センサノードが送信したデータが，DTN を経由してゲートウェイノードに届けられる．ゲートウェイノードは通常のネットワーク (133.11.168.0/25) に接続されており，センサノードはネットワーク上のサーバ 133.11.168.120 へ ADTP でメッセージを送信する．センサノードとゲートウェイノードの間の DTN で，PEAR が IP パケットを配送することにより，センサノードのデータがサーバへ届けられる．

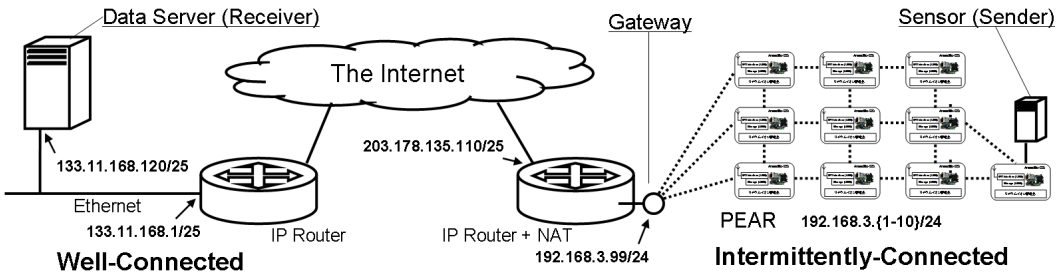

図 7.9　実験のネットワーク構成

ノードは図 7.10 に示すように配置した．StaticA では工学部 2 号館の各階に 1 個ずつノードを固定し，StaticB では各階に 2 個ずつ固定した．DynamicA では 9 個のノードが固定され，1 個のノードが位置 A とゲートウェイノードの間，および位置 B とゲートウェイノードの間を交互に 20 分周期で往復した．DynamicB では 5 個のノードが固定され，5 個のノードが破線で示した範囲内を移動し，実験中に数回ゲートウェイノードの位置へ戻った．

StaticA と StaticB の実験は 12 時間ずつ行い，DynamicA と Dynamic はそれぞれ 3 時間および 1.5 時間行った．実験中，センサノードは 90 秒ごとに 5kByte，15kByte，45kByte のメッセージを生成し，誤り訂正符号を加えると 90 秒間で 120 パケットが生成されるようにした．また，PEAR の設定は $D = 0.2,\ \rho = 0.02,\ \alpha = 0.04$, message TTL=2400[sec] とした．advertisement の TTL は 30[sec] とし，dissemination TTL=1800[sec] とした．この設定では最大で約 3200[entry] のバッファを使用するのに対し，ノードのバッファサイズは 4096[entry] であるので，バッファオーバーフローは起きない設定となっている．

実験で得られたコンタクトラフを図 7.11 に示す．ノード間の線は接続が存在することを表し，太い実線は平均の接続時間が長いことを表す．こから明らかなように，コンタクトグラフは物理的なノードの配置を反映している．したがって，コンタクトグラフからノードの物理的な位置を推測することが可能であるといえる．また，ノードが固定されている場合でも，接続関係は時間変化していることが分かった．なお，実験中に非対称な接続も観測されたが，図 7.11 には示されていない．

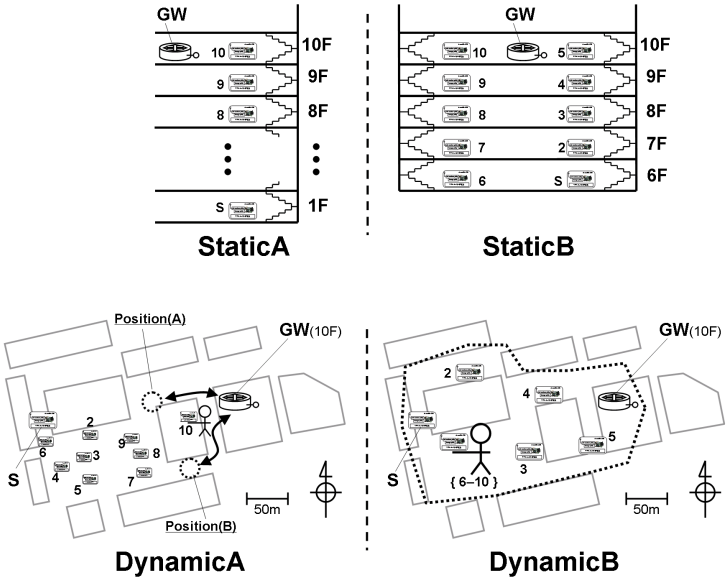

図 7.10　実験のシナリオ

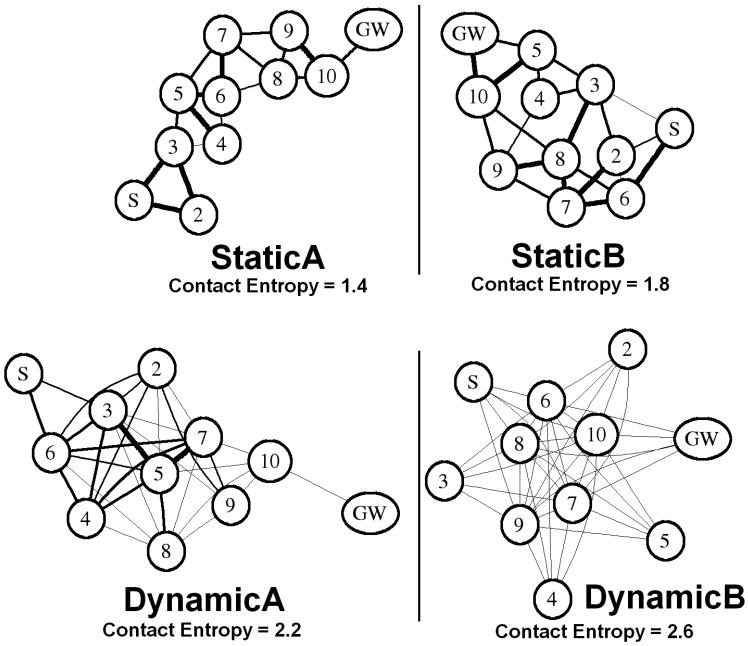

図 7.11　ノード間接続関係

図 7.12 は StaticA，DynamicA，DynamicB における各ノードからゲートウェイノードへのポテンシャルの時間変化を表したグラフである．StaticA ではポテンシャルはほぼ一定であるが，途中近隣ノードとの接続が切れてポテンシャルが上昇している箇所がある．DynamicA ではノード 10 のポテンシャルが常に最も低く，これはノード 10 だけがゲートウェイノードと直接通信可能であることによるものである．DynamicB ではノードの移動が複雑であるため，ポテンシャルも複雑に上下したものとなっている．

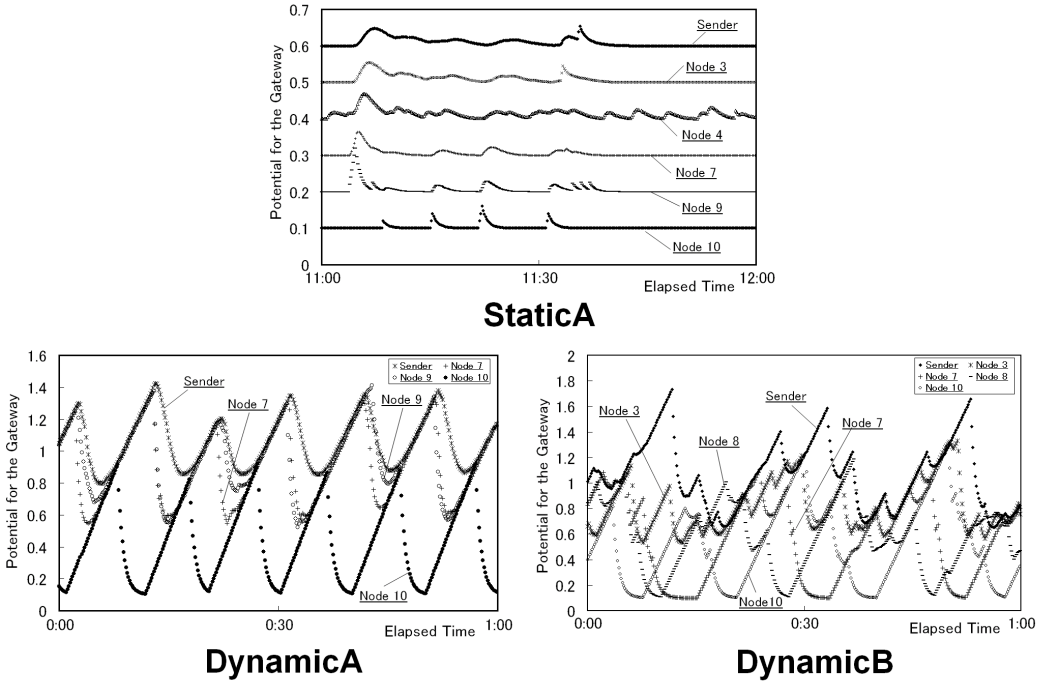

図 7.12　各ノードからゲートウェイノードへのポテンシャル

図 7.13 はセンサノード (S) からゲートウェイノード (GW) への IP パケットの配送パターンを表している．それぞれのグラフは異なる IP パケットを表しており，矢印の横にある数字はゲートウェイノードに到着した時刻を 0[sec] としたときのパケットが配送された時刻を表している．この図に表れているように，配送経路は枝分かれしており，冗長な配送が行われている．これはデータ転送のバッファ占有率のオーバーヘッドを大きくするが，配送率や配送遅延を改善する効果がある．配送の冗長度 R を

$$R = \frac{\text{コピー回数}}{\text{センサからゲートウェイまでのホップ数}}$$

で定義し，平均冗長度を図 7.13 に示した．$R = 1$ のときは配送経路に枝分かれはなく，R が大きくなるにつれてより多くのコピーが作られ，配送率と配送遅延が改善される．この図に示されているように，エントロピーが大きいほど冗長度が高くなることが確認された．

図 7.14 はノード間における advertisement，investigation，データの各パケットの伝送量を表している．伝送量全体に対する制御信号（advertisement および investigation）が占める割合はおよそ 10%〜20% であり，伝送量の大部分はデータ信号が占めていることが分かる．

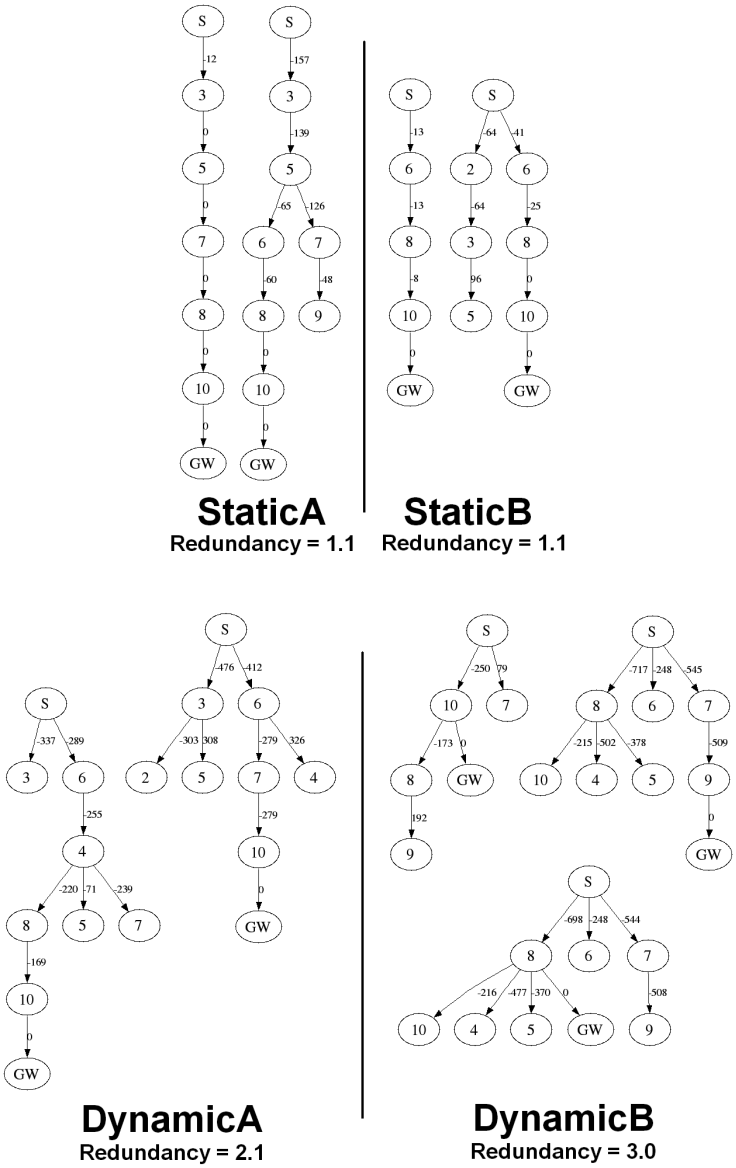

図 7.13　パケット配送パターン

図 7.15 はノードのバッファ占有率を表している．StaticA と StaticB では，占有されているバッファのおよそ 90% 以上はメッセージ本体がクリアされたヘッダのみのエントリであり，ほとんどのメッセージが短時間で配送完了されているため，バッファには配送が完了したことを他のノードに知らせるためにヘッダのみが残されている．一方，DynamicA と DynamicB では，およそ 30%〜50% のエントリがメッセージ本体を保持しており，これらのエントリが他のノードへコピーを作り配送の冗長度が大きくなる．

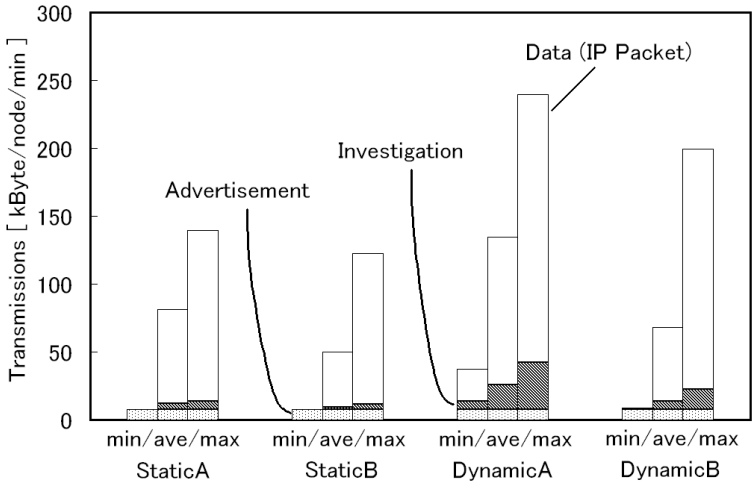

図 7.14　ノード間のパケット伝送量

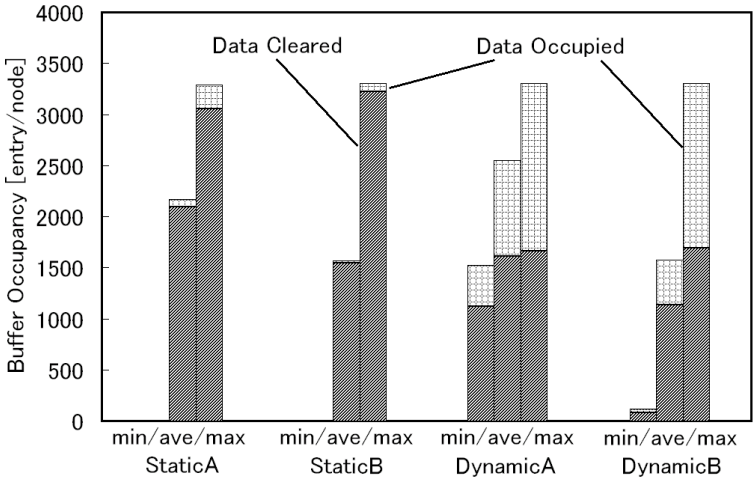

図 7.15　ノードのバッファ占有率

7.3.3　まとめ

IP over DTN では，IP パケットを非同期伝送のユニットとして用いており，データリンク層が対遅延性を与えている．データリンク層として PEAR を用い，実機に実装をして実験を行い，メッセージ配送率や配送遅延，伝送量，バッファ占有量などの結果を分析した．これらの結果から，IP over DTN は，バッファ占有や冗長な配送などのオーバーヘッドを伴うものの，今回の実験シナリオでは 100% のメッセージ到着率を実現しており，実環境のインターネットへの組み込みに適していると言うことができる．

第 8 章

まとめ

8.1 結論

本研究では，実環境に即した状況下でバッファ管理方式および転送方式が PEAR に与える影響をシミュレーションによって調べ，実環境において PEAR に最適なバッファ管理方式と転送方式の組み合わせを示した．

バッファサイズのみが有限の場合は，バッファ管理方式が Youngest Drop のときにメッセージ配送率が最も高くなった．転送方式はバッファ管理方式が FIFO または LIFO の場合のみ影響を及ぼした．通信速度のみが有限の場合は，転送方式が Least Forwarded First Out のときにメッセージ配送率が最も高くなり，バッファ管理方式による違いはなかった．バッファサイズおよび通信速度の双方が有限の場合は，片方のみが有限の場合と同様に，配送率はバッファ管理方式が Youngest Drop，転送方式が Least Forwarded First Out のときに最も高くなった．また，この組み合わせのときには配送遅延が比較的小さくなり，配送冗長度も低く抑えられることが分かった．また，実環境に即した状況下で PEAR と Epidemic Routing および Binary Spray and Wait Routing の性能を比較し，配送率・配送冗長度・パケット伝送量・バッファ使用量において PEAR が他のルーティングプロトコルより優れていることを示した．さらに，メッセージ TTL が配送率およびバッファ使用量に与える影響を調べ，適切なメッセージ TTL を設定することでバッファ使用量を削減する手法を提案した．そして，PEAR におけるポテンシャルの構築方法を改良し，配送遅延を低減する手法を提案した．

また，DTN とインターネットを接続するための新しいアーキテクチャ IP over DTN を構築し，実機を用いて実際に動作することを確認し，有用性を示した．IP over DTN を用いることで，DTN 上で IP パケットを伝送することができた．IP パケットは大きな遅延を伴って伝送されても，アプリケーションが非同期な通信を前提に設計されていれば，IP によって対遅延性を持った通信ができることが確認できた．さらに，IP ルーターおよび NAPT と組み合わせることで，オーバーレイネットワークを構築することなく DTN とインターネットを接続することができることを確認した．

8.2 今後の課題

今後の課題としては，ノード毎にバッファサイズやバンド幅が異なる場合の影響の考慮や，バッテリー残量などの他の要素の考慮が挙げられる．本研究では，全てのノードが同一の条件（バッファサイ

ズ・バンド幅）を持っていると仮定したが，実際には不均一である場合が多い．さらに，バッテリー残量などを考慮することで，より実際の状況を再現することができる．また，さまざまなモビリティパターンで実験を行い，モビリティ依存性を解析する必要がある．そのためには，モビリティモデルの構築や，実際のバッテリー残量をシミュレータ上で再現する手法が必要となる．

さらに，実機に実装して実環境で実験を行い，本研究での結果や改善を検証する必要がある．

謝辞

修士の研究活動では，本論文の執筆以外にも多くの事を学び多くの人からご指導とご支援を頂きました．ここに江崎研究室での研究生活への感謝の意を表します．

研究を進めるにあたり，研究内容のみならず研究に対する心構えなども助言を下さった江崎浩教授に深く感謝いたします．

的確な助言と温かい激励を下さった土本康生博士，ネットワークに関してご指導下さった山本成一博士，金海好彦氏，土井裕介氏に感謝致します．研究に対する真摯な姿勢でご指導して下さった藤田祥氏に感謝いたします．

本論文の執筆にあたって，研究の方向性や詳細についてご指導下さり，また議論や質問に長い時間お付き合い下さった落合秀也氏に深く感謝を申し上げます．

運用ができ研究室のネットワークを支えてくれる浅井大史氏，研究ができいろいろと助言をくれる肥村洋輔氏，趙越氏，川上雄也君，杉田毅博君，Sathita Kaveevivitchai さん，Leela-Amornsin Lertluck 君，唐明シン氏，Luciano Aparicio 氏，呉和賢君，川口紘典君，本舘拓也君，David Jageberg 君，Jonas Johansson 君に感謝致します．

研究しやすい環境の整備や事務的なサポートをしていただいた高橋富美氏，田坂佳苗氏に感謝致します．

最後に，研究生活や学生生活においていろいろなご指導，ご支援を下さった全ての皆様に感謝致します．

2010 年 2 月 9 日

下忠 健一

参考文献

[1] S. Burleigh, A. Hooke, L. Torgerson, K. Fall, V. Cerf, B. Durst, K. Scott, and H. Weiss, “Delay-tolerant networking: an approach to interplanetary internet”, IEEE Communications Magazine, **41**, 6, pp. 128-136 (2003).

[2] D. Demmer and K. Fall, “DTLSR: Delay tolerant routing for developing regions”, ACM NSDR (2007).

[3] V. Cerf, S. Burleigh, A. Hooke, L. Torgerson, R. Durst, K. Scott, K. Fall, and H. Weiss, “FC4838: delay tolerant networking architecture” (2007).

[4] A. Vahdat and D. Becker, “Epidemic routing for partially–connected ad–hoc networks”, Tech report, Duke University (2000).

[5] J. Hahner, C. Becker and K. Rothermel, “A protocol for data dissemination in frequently partitioned mobile ad hoc networks”, IEEE ISCC (2003).

[6] J. Burgess, B. Gallagher, D. Jensen and B. N. Levine, “MaxProp: routing for vehicle-based disruption-tolerant networks”, IEEE INFOCOM (2006).

[7] T. Spyropoulos, T. Turletti and K. Obraczka, “Utility-based message replication for intermittently connected heterogeneous networks”, IEEE WoWMoM (2007).

[8] A. Panagakis, A. Vaios and I. Stavrakakis, “Study of two-hop message spreading in DTNs”, IEEE WiOpt (2007).

[9] A. Basu, A. Lin and S. Ramanathan, “Routing using potentials: a dynamic traffic–aware routing algorithm”, ACM SIGCOMM (2003).

[10] P. Kumar, J. Kuri, P. Nuggehalli, M. Strasser, M. May and B. Plattner, ”Connectivity–aware routing in sensor networks”, IEEE SENSERCOMM (2007).

[11] H. Liu, Z. Zhang, J. Srivastava, V. Firoiu and B. Decleene, ”PWave: a multi–source multi–sink anycast routing framework for wireless sensor networks”, LNCS, **4479**, pp. 179-—190 (2007).

[12] R. D. Poor, ”Gradient routing in ad hoc networks”, Tech report, MIT (2000).

[13] Y. Ganjali and N. McKeown “Routing in a highly dynamic topology”, IEEE SECON (2005).

[14] G. G. Chen, J. W. Branch and B. K. Szymanski, “Self–selective routing for wireless ad hoc networks”, IEEE WiMob (2005).

[15] P. Juang, H. Oki, Y. Wang, M. Martonosi, L. Peh and D. Rubenstein, “Energy-efficient computing for wildlife tracking: design tradeoffs and early experiences with ZebraNet”, ACM SIGPLAN Notices (2002).

[16] M. Musolesi, S. Hailes and C. Mascolo, “Adaptive routing for intermittently connected mobile ad hoc networks”, IEEE WoWMoM (2005).

[17] C. Chen and Z. Chen, “Evaluating contacts for routing in highly partitioned mobile networks”, ACM MobiOpp (2007).

[18] A. Lindgren, A. Doria and O. Schelen, “Probablistic routing in intermittently connected networks”, LNCS, **3126**, pp. 239-—254 (2004).

[19] S. Jain, K. Fall and R. Patra, “Routing in a delay tolerant network”, ACM SIGCOMM Workshop (2004).

[20] H. Ochiai and H. Esaki. “Mobility entropy and message routing in community-structured delay tolerant networks”. ACM AINTEC, pp. 93-102 (2008).

[21] A.Lindgren, K.S.Phanse, “Evaluation of queuing policies and forwarding strategies for routing in intermittently connected networks”, Proc. of IEEE COMSWARE (2006).

[22] Christian Bettstetter, Giovanni Resta, Paolo Santi, “The node distribution of the random waypoint mobility model for wireless ad hoc networks”, IEEE Transactions on Mobile Computing, **2**, 3, (2003).

[23] T. V. Lakshman and U. Madhow, “The performance of TCP/IP for networks with high bandwidth-delay products and random loss”, IEEE/ACM Transactions on Networking, **5**, 3, pp. 336-350 (1997).

[24] S. Floyd, V. Jacobson, C.-G. Liu, S. McCanne, and L. Zhang, “A reliable multicast framework for light-weight sessions and application level framing”, IEEE/ACM Transactions on Networking, **5**, 6, pp. 784-803 (1997).

[25] C. Diot, B. N. Levine, B. Lyles, H. Kassem, and D. Balensiefen, “Deployment issues for the IP multicast service and architecture”, IEEE Network, **14**, pp. 78-88 (2000).

[26] J. Ott, D. Kutscher, and C. Dwertmann, “Integrating DTN and MANET routing”, ACM SIGCOMM workshop, pp. 221-228 (2006).

[27] “DTN research group”, http://www.dtnrg.org/.

[28] P. T. Eugster, P. A. Felber, R. Guerraoui, and A. M. Kermarrec,“The many faces of publish/-subscribe”, ACM Computing Surveys, **35**, 2, pp. 114–131 (2003).

[29] R. C. Durst, P. D. Feighery, and K. L. Scott, “why not use the standard internet suite for the interplanetary internet”, http://www.ipnsig.org/reports/TCP_IP.pdf.

[30] Armadillo 220. http://www.atmark-techno.com/.

発表文献

[1] 落合秀也, 下忠健一, 江崎浩：”IP over DTN: Large-Delay Asynchronous Packet Delivery in the Internet”, E-DTN (2009).

[2] 下忠健一, 落合秀也, 江崎浩：”Evaluation and Improvement of DTN Routing Algorithm”, The third AsiaFI Winter School (2010). [発表予定]

[3] 下忠健一, 落合秀也, 江崎浩：”バッファ管理／転送方式が DTN 配送性能に与える影響”, 信学総合大会 (2010). [発表予定]

[4] 落合秀也, 下忠健一, 江崎浩：”DTIPN: Delay Tolerant IP Networking for Opportunistic Network Applications”, MobiOpp (2010). [発表予定]

付録 A

DTN シミュレータのソースコード

ソースコード A.1　PEAR でルーティングを行うプログラム

```
// 2009-08-04 Updated by Kenichi Shimotada

package org.livee.dtn.sim;

public class SingleCopyNode implements Node {

  private String m_ID;
  public double D; // originally 0.2
  public double R; // originally 0.02
  public double alpha; // modified to 0.08 from 0.0 (2008-05-10) // modified to 0.04 from 0.08(2
009-08-23)
  public int AdvValidTerm; // originally 60
  public int initial_gttl;
  public int MessageBufferSize;
  public int bandWidth;
  public boolean dynamicTTL;

  public static final int fwdMng_Random=1;
  public static final int fwdMng_LeastFwded=2;
  public static final int fwdMng_Oldest=3;
  public static final int fwdMng_LIFO=4;
  public static final int fwdMng_FIFO=5;
  private int fwdMngMethod=fwdMng_LeastFwded;

  public static final int bufMng_Random=6;
  public static final int bufMng_MostFwded=7;
  public static final int bufMng_Youngest=8;
  public static final int bufMng_LIFO=9;
  public static final int bufMng_FIFO=10;
  private int bufMngMethod=bufMng_Youngest;

  public final int TIME_UNIT=10;
  private java.util.ArrayList<String> neighborNodes_recv=new java.util.ArrayList<String>();
  private java.util.ArrayList<String> neighborNodes_send=new java.util.ArrayList<String>();

  private static java.util.HashSet<String> GTTLExpiredMsgs=new java.util.HashSet<String>();
  private static java.util.HashSet<String> DeliveredMsgs=new java.util.HashSet<String>();
  private static java.util.HashSet<String> ExpiredDeliveredMsgs=new java.util.HashSet<String>();

  private static Object mutex_MsgsSet=new Object();
  private static int numMsgSent=0;
  private static Object mutex_numMsgSent=new Object();
  private static long trnsAdvertisement=0;
  private static long trnsInvestigation=0;
  private static long trnsData=0;
  private static Object mutex_trns=new Object();
  private static java.util.Hashtable<Integer,Integer> pdiffHist=new java.util.Hashtable<Integer,
Integer>();
  private static Object mutex_pdiffHist=new Object();
  private final double pdiffHist_step=0.005;
  private final double pdiffHist_shift=pdiffHist_step/2.0;
  private int dropped=0;
```

```
  private int averageGTTL=0;

  public SingleCopyNode(String id, String params, int _fwdMngMethod, int _bufMngMethod){
    m_ID=id;
    m_Time=Time.time();
  //  m_LastTime=-10;
    m_LastTime=m_Time-10;
    initRouteConstruction();
    initMsgForwarding();
  //  start();

    D=Double.parseDouble(params.split("D=")[1].split(",")[0]);
    R=Double.parseDouble(params.split("R=")[1].split(",")[0]);
    alpha=Double.parseDouble(params.split("a=")[1].split(",")[0]);
    AdvValidTerm=Integer.parseInt(params.split("v=")[1].split(",")[0]);
    if(params.split("g=")[1].split(",")[0].equals("MAX_VALUE")||params.split("g=")[1].split(",")
[0].startsWith("inf")){
      initial_gttl=Integer.MAX_VALUE;
    }else{
      initial_gttl=Integer.parseInt(params.split("g=")[1].split(",")[0]);
    }
    if(params.split("b=")[1].split(",")[0].equals("MAX_VALUE")||params.split("b=")[1].split(",")
[0].startsWith("inf")){
      MessageBufferSize=Integer.MAX_VALUE;
    }else{
      MessageBufferSize=Integer.parseInt(params.split("b=")[1].split(",")[0]);
    }
    if(params.split("w=")[1].split(",")[0].equals("MAX_VALUE")||params.split("w=")[1].split(",")
[0].startsWith("inf")){
      bandWidth=Integer.MAX_VALUE;
    }else{
      bandWidth=Integer.parseInt(params.split("w=")[1].split(",")[0]);
    }
    if(params.split("dynamicTTL=")[1].split(",")[0].equals("true")||params.split("dynamicTTL=")[
1].split(",")[0].startsWith("yes")){
      dynamicTTL=true;
    }else{
      dynamicTTL=false;
    }

    fwdMngMethod=_fwdMngMethod;
    bufMngMethod=_bufMngMethod;

    Logger.logMessage(m_ID+java.io.File.separator+"BufferOccupancy","#time before now TTLexpired
 delivered totalEntries withBody headerOnly dropped");
  }

  public String getNodeID(){
    return m_ID;
  }

  private long m_LastTime;
  private long m_Time;

  public void run_01(){
    m_Time=Time.time();
    routeConstructionProc();
    broadcastPotential();
    decrementTTL();
    dropped=0;
    rxCount=0;
    txCount=0;
  }

  public void run_02(){
    msgForwardingProc();
  }

  public void run_03(){
    m_NewMessageList=new java.util.ArrayList<String>(m_MessageBuffer.keySet());

    logPotential();
    if(Time.ready()){
      logFT();
      logTransmissions();
      logDeliveryRate();
```

```
      calcPdiffHist();
      logBufferOccupancy();
    }

    m_LastTime=m_Time;

    calcAverageGTTL();
    System.out.println("avg remaining gttl="+averageGTTL);
  }

/*  public void run(){

    while(true){

      try{
        Thread.sleep(100);
      }catch(Exception e){

      }

      m_Time=Time.time();
      if(m_Time>=m_LastTime+TIME_UNIT){
        m_Time=((m_LastTime/TIME_UNIT)+1)*TIME_UNIT;
        try{
          routeConstructionProc();
          decrementTTL();
          msgForwardingProc();
          logPotential();
          logFT();
        }catch(Exception e){
          e.printStackTrace();
        }
        m_LastTime=m_Time;
      }
    }

  }
*/

  private void calcAverageGTTL(){
    java.util.Iterator<Message> itr=m_MessageBuffer.values().iterator();
    int sum=0;
    int count=0;
    while(itr.hasNext()){
      Message m=itr.next();
      if(m.message_body!=null&&m.seq.contains("test")==false){
        sum+=m.gttl;
        count++;
      }
    }
    if(count!=0){
      averageGTTL=sum/count;
    }else{
      averageGTTL=0;
    }
  }


  // For Constructing Potential
  /**
   * @uml.property  name="m_NeighborTTL"
   * @uml.associationEnd  qualifier="key:java.lang.Object java.lang.Integer"
   */
  private java.util.Map<String,Integer> m_NeighborTTL;
  /**
   * @uml.property  name="m_NeighborPotentials"
   * @uml.associationEnd  qualifier="key:java.lang.Object java.util.Map<String,double[]>"
   */
  private java.util.Map<String,java.util.Map<String,double[]>> m_NeighborPotentials;
  /**
   * @uml.property  name="m_ThisPotential"
   * @uml.associationEnd  qualifier="key:java.lang.Object doub[]"
   */
  private java.util.Map<String,double[]> m_ThisPotential;
  private void initRouteConstruction(){
    m_NeighborTTL=new java.util.Hashtable<String,Integer>();
```

```
    m_NeighborPotentials=new java.util.Hashtable<String,java.util.Map<String, double[]>>();
    m_ThisPotential=new java.util.Hashtable<String,double[]>();
  }

  private void routeConstructionProc(){

    // Delete Obsolete Neighbors and List All the Destinations at the same time
    java.util.ArrayList<String> destinations=new java.util.ArrayList<String>();
    Object[] keys=m_NeighborTTL.keySet().toArray();

    for(int i=0;i<keys.length;i++){
      Integer ttl=m_NeighborTTL.get(keys[i]);
      if(ttl>=0){
        m_NeighborTTL.put((String)keys[i], ttl-TIME_UNIT);
      }else{
        m_NeighborTTL.remove(keys[i]);
        m_NeighborPotentials.remove(keys[i]);
      }

      //synchronized(m_NeighborPotentials){
      java.util.Map<String,double[]> p=m_NeighborPotentials.get(keys[i]);
      if(p!=null){
        Object[] dests=p.keySet().toArray();
        for(int j=0;j<dests.length;j++){
          String dest=(String)dests[j];
          if(!destinations.contains(dest)){
            destinations.add(dest);
          }
        }
      }
      //}
    }

    // This
    java.util.Iterator<String> itr_thisPotential=m_ThisPotential.keySet().iterator();
    while(itr_thisPotential.hasNext()){
      String dest=itr_thisPotential.next();
      if(!destinations.contains(dest)){
        destinations.add(dest);
      }
    }

    // Calculate the Next Potentials
    Object[] dests=destinations.toArray();
    for(int d=0;d<dests.length;d++){
      double min_p=Double.MAX_VALUE;
      double min_p_grad=Double.MAX_VALUE;
      String min_ID="";
      for(int k=0;k<keys.length;k++){
        java.util.Map<String,double[]> p=m_NeighborPotentials.get(keys[k]);
        if(p==null){ continue;  }

        double[] pp=p.get(dests[d]);
        if(pp!=null){
      //    pp[0]+=pp[1];
          if(min_p>pp[0]){
            min_p=pp[0];
            min_p_grad=pp[1];
            min_ID=(String)keys[k];
          }
        }
      }

      double[] pp=m_ThisPotential.get(dests[d]);
      if(pp!=null){
        double[] next_p=new double[2];
        if(pp[0]>min_p){
          if(min_p_grad<0.0001){ // not increasing
            next_p[0]=pp[0]+D*(min_p-pp[0])+R;
          }else{ // increasing
            if(pp[0]-min_p>R/D){
              next_p[0]=pp[0]+D*(min_p-pp[0])+R;
            }else{
              next_p[0]=min_p+R/D;
            }
          }
```

```
281           next_p[1]=next_p[0]-pp[0];
282           m_ThisPotential.put((String)dests[d],next_p);
283         }else{
284           next_p[0]=pp[0]+R;
285           next_p[1]=next_p[0]-pp[0];
286           m_ThisPotential.put((String)dests[d],next_p);
287         }
288       }else{
289         double[] next_p=new double[2];
290         next_p[0]=R+min_p; // added 2008-05-10
291         next_p[1]=0;
292         m_ThisPotential.put((String)dests[d],next_p);
293       }
294     }
295     double[] next_p=new double[2];
296     next_p[0]=0.0; next_p[1]=0.0;
297     m_ThisPotential.put(m_ID, next_p);
298
299     // begin Logging --- the size of potential table
300     //java.util.Iterator<String> itr=m_ThisPotential.keySet().iterator();
301     //while(itr.hasNext()){
302     //  String dest=itr.next();
303     //  String potential=Double.toString(m_ThisPotential.get(dest)[0]);
304     //  Logger.logMessage("PotentialTable", m_Time+","+m_ID+","+dest+","+potential);
305     //}
306     // end Logging
307   }
308
309   private void broadcastPotential(){
310     // Broadcast the Potentials
311     StringBuilder sb=new StringBuilder();
312     int size=m_ThisPotential.size();
313     sb.append(m_ID);
314     sb.append(",");
315     sb.append(size);
316     sb.append(",");
317     Object[] dests=m_ThisPotential.keySet().toArray();
318     for(int i=0;i<dests.length;i++){
319       sb.append(dests[i]);
320       sb.append(",");
321       double[] p=m_ThisPotential.get(dests[i]);
322       sb.append(p[0]);
323       sb.append(",");
324       sb.append(p[1]);
325       sb.append(",");
326     }
327     String bcast_message=sb.toString();
328
329     Node[] ns=NodeLocator.findNeighborNodes_send(this);
330     if(ns!=null){
331       for(int i=0;i<ns.length;i++){
332         ns[i].recvBroadcastMessage(bcast_message);
333       }
334
335       if(Time.ready()){
336         synchronized(mutex_trns){
337           trnsAdvertisement+=(6+8*m_ThisPotential.size());
338         }
339       }
340     }
341   }
342
343   public void recvBroadcastMessage(String data) {
344     String[] datum=data.split(",");
345
346     if(datum.length>2){
347       String neighbor=datum[0];
348       int size=Integer.parseInt(datum[1]);
349
350       m_NeighborTTL.put(neighbor, AdvValidTerm);
351
352       java.util.Map<String,double[]> potentials=m_NeighborPotentials.get(neighbor);
353       if(potentials==null){
354         potentials=new java.util.Hashtable<String,double[]>();
355         //synchronized(m_NeighborPotentials){  // cause dead locks
356           m_NeighborPotentials.put(neighbor, potentials);
```

```
357         //}
358       }
359
360       for(int i=0;i<size*3;i+=3){
361         try{
362           String dest=datum[i+2];
363           double[] potential=new double[2];
364           potential[0]=Double.parseDouble(datum[i+3]);
365           potential[1]=Double.parseDouble(datum[i+4]);
366           potentials.put(dest, potential);
367         }catch(Exception e){
368           e.printStackTrace();
369         }
370       }
371     }
372
373   }
374
375   public int getPotentialTableSize(){
376     return m_ThisPotential.size();
377   }
378
379   // DecrementTTL
380   private void decrementTTL(){
381     Object[] msg_id;
382
383     msg_id=m_MessageBuffer.keySet().toArray();
384     for(int i=0;i<msg_id.length;i++){
385       Message m=m_MessageBuffer.get(msg_id[i]);
386       //System.out.println("Decrement: "+msg_id[i]+","+m.gttl+","+m.ttl+" @"+m_ID);
387       if(m.isForwarded==true){
388         m.ttl-=TIME_UNIT;
389       }
390       m.gttl-=TIME_UNIT;
391       if(m.gttl<=0){
392         removeMessageFromBuffer((String)msg_id[i]);
393         if(Time.ready()){
394           synchronized(mutex_MsgsSet){
395             GTTLExpiredMsgs.add(m.seq);
396           }
397         }
398       }
399     }
400
401     msg_id=m_DeliveredMessageBuffer.keySet().toArray();
402     for(int i=0;i<msg_id.length;i++){
403       Message m=m_DeliveredMessageBuffer.get(msg_id[i]);
404       m.gttl-=TIME_UNIT;
405       if(m.gttl<=0){
406         m_DeliveredMessageBuffer.remove((String)msg_id[i]);
407         if(Time.ready()){
408           synchronized(mutex_MsgsSet){
409             GTTLExpiredMsgs.add(m.seq);
410             ExpiredDeliveredMsgs.add(m.seq);
411           }
412         }
413       }
414     }
415
416   }
417
418   private void removeMessageFromBuffer(String msg_id){
419     m_MessageBuffer.remove(msg_id);
420     m_MessageForwardCount.remove(msg_id);
421     m_MessageReceivedTime.remove(msg_id);
422   }
423
424   // For Messaging
425   /**
426    * @uml.property  name="m_MessageBuffer"
427    * @uml.associationEnd  qualifier="key:java.lang.Object org.livee.dtn.sim.Message"
428    */
429   private java.util.Map<String,Message> m_MessageBuffer;
430   private java.util.Hashtable<String,Message> m_DeliveredMessageBuffer=new java.util.Hashtable<S
431 tring,Message>();
432   private java.util.Hashtable<String,Long> m_MessageReceivedTime=new java.util.Hashtable<String,
```

```
433 Long>();
434   private java.util.Hashtable<String,Long> m_MessageDeliveredTime=new java.util.Hashtable<String
435 ,Long>();
436   private java.util.Hashtable<String,Integer> m_MessageForwardCount=new java.util.Hashtable<Stri
437 ng,Integer>();
438   private java.util.ArrayList<String> m_NewMessageList;
439   private void initMsgForwarding(){
440     m_MessageBuffer=new java.util.Hashtable<String,Message>();
441     m_NewMessageList=new java.util.ArrayList<String>();
442   }
443
444   private int txCount=0;
445   private void msgForwardingProc(){
446     if(m_NewMessageList.isEmpty()){
447       return;
448     }
449
450     // forwarding method
451     java.util.ArrayList<java.util.Map.Entry> msgs=null;
452
453     if(fwdMngMethod==fwdMng_Random){ // random (depends on hashtable's algorithm)
454       msgs=new java.util.ArrayList<java.util.Map.Entry>(m_MessageBuffer.entrySet());
455
456     }else if(fwdMngMethod==fwdMng_LeastFwded){ // forward count
457       msgs=new java.util.ArrayList<java.util.Map.Entry>(m_MessageForwardCount.entrySet());
458       java.util.Collections.sort(msgs, new java.util.Comparator(){
459         public int compare(Object obj1, Object obj2){
460           java.util.Map.Entry ent1=(java.util.Map.Entry)obj1;
461           java.util.Map.Entry ent2=(java.util.Map.Entry)obj2;
462           Integer val1=(Integer)ent1.getValue();
463           Integer val2=(Integer)ent2.getValue();
464           return val1.compareTo(val2);
465         }
466       });
467
468     }else if(fwdMngMethod==fwdMng_Oldest){ // gttl
469       msgs=new java.util.ArrayList<java.util.Map.Entry>(m_MessageBuffer.entrySet());
470       java.util.Collections.sort(msgs, new java.util.Comparator(){
471         public int compare(Object obj1, Object obj2){
472           java.util.Map.Entry ent1=(java.util.Map.Entry)obj1;
473           java.util.Map.Entry ent2=(java.util.Map.Entry)obj2;
474           Integer val1=((Message)ent1.getValue()).gttl;
475           Integer val2=((Message)ent2.getValue()).gttl;
476           return val1.compareTo(val2);
477         }
478       });
479
480     }else if(fwdMngMethod==fwdMng_LIFO){ // LIFO (received time descending)
481       msgs=new java.util.ArrayList<java.util.Map.Entry>(m_MessageReceivedTime.entrySet());
482       java.util.Collections.sort(msgs, new java.util.Comparator(){
483         public int compare(Object obj1, Object obj2){
484           java.util.Map.Entry ent1=(java.util.Map.Entry)obj1;
485           java.util.Map.Entry ent2=(java.util.Map.Entry)obj2;
486           Long val1=(Long)ent1.getValue();
487           Long val2=(Long)ent2.getValue();
488           return val1.compareTo(val2)*-1; // *-1 for descending sort
489         }
490       });
491
492     }else if(fwdMngMethod==fwdMng_FIFO){ // FIFO (received time ascending)
493       msgs=new java.util.ArrayList<java.util.Map.Entry>(m_MessageReceivedTime.entrySet());
494       java.util.Collections.sort(msgs, new java.util.Comparator(){
495         public int compare(Object obj1, Object obj2){
496           java.util.Map.Entry ent1=(java.util.Map.Entry)obj1;
497           java.util.Map.Entry ent2=(java.util.Map.Entry)obj2;
498           Long val1=(Long)ent1.getValue();
499           Long val2=(Long)ent2.getValue();
500           return val1.compareTo(val2);
501         }
502       });
503     }
504
505     java.util.ArrayList<String> nexthop_refuse=new java.util.ArrayList<String>();
506     java.util.Iterator<java.util.Map.Entry> itr_msgs=msgs.iterator();
507     while(itr_msgs.hasNext() && txCount<bandWidth){
508       Message m=m_MessageBuffer.get(itr_msgs.next().getKey());
```

```
      if(m.isDelivered){
        continue;
      }

      if(m_NewMessageList.contains(m.seq)==false){
        continue;
      }

      double[] local_p=m_ThisPotential.get(m.dest);
      if(local_p==null){
        continue;
      }

      java.util.Hashtable<String,Double> pdiffTable=new java.util.Hashtable<String,Double>();

      //synchronized(m_NeighborPotentials){
      //  java.util.Iterator<String> itr=m_NeighborPotentials.keySet().iterator();
      java.util.ArrayList<String> keys=new java.util.ArrayList<String>(m_NeighborPotentials.keyS
et());
      java.util.Iterator<String> itr=keys.iterator();
      while(itr.hasNext()){
        String neighbor=itr.next();
        java.util.Map<String,double[]> potentials=m_NeighborPotentials.get(neighbor);

        if(nexthop_refuse.contains(neighbor)){
          continue; // omit refusing node
        }

        double[] neigh_p=potentials.get(m.dest);
        if(neigh_p==null){
          continue;
        }

      //  if(neigh_p[1]<0.001){ // not incresing
          if(local_p[0]-neigh_p[0]>alpha){
            pdiffTable.put(neighbor,local_p[0]-neigh_p[0]);
          }
      /*  }else{ // increasing
          if(local_p[0]-neigh_p[0]>alpha&&local_p[0]-neigh_p[0]>R/D*0.8){
            pdiffTable.put(neighbor,local_p[0]-neigh_p[0]);
          }
        }*/
      }
      //}

      if(pdiffTable.isEmpty()){
        continue;
      }
      java.util.ArrayList<java.util.Map.Entry> pdiffEntries=new java.util.ArrayList<java.util.Ma
p.Entry>(pdiffTable.entrySet());
      java.util.Collections.sort(pdiffEntries, new java.util.Comparator(){
        public int compare(Object obj1, Object obj2){
          java.util.Map.Entry ent1=(java.util.Map.Entry)obj1;
          java.util.Map.Entry ent2=(java.util.Map.Entry)obj2;
          Double val1=(Double)ent1.getValue();
          Double val2=(Double)ent2.getValue();
          return val1.compareTo(val2)*-1; // *-1 for descending sort
        }
      });

      java.util.Iterator<java.util.Map.Entry> itr_nexthop=pdiffEntries.iterator();
      while(itr_nexthop.hasNext()){
        String neighbor_max=(String)itr_nexthop.next().getKey();

        if(Time.ready()){
          synchronized(mutex_trns){
            trnsInvestigation+=5;
          }
        }

        if(getNeighborNodes_send().contains(neighbor_max)==false||getNeighborNodes_recv().contai
ns(neighbor_max)==false){
          break;
        }

        Node next_hop_candidate=NodeLocator.getNodeFromID(neighbor_max);
```

```
585         long hasTest=next_hop_candidate.hasTest(m.seq,m.dest);
586         if(Time.ready()){
587           synchronized(mutex_trns){
588             trnsInvestigation+=12;
589           }
590         }
591
592         if(hasTest==NOT_HAVE && m.ttl>0){
593           //  if(m.ttl>DISSEMINATION_MODE_TTL){
594           //    m.ttl=DISSEMINATION_MODE_TTL;
595           //  }
596           if(next_hop_candidate.postMessage(m_ID,m.serialize())==true){
597             txCount++;
598             m.isForwarded=true;
599             m_MessageForwardCount.put(m.seq,m_MessageForwardCount.get(m.seq)+1);
600
601             if(Time.ready()){
602               synchronized(mutex_trns){
603                 trnsData+=167;
604               }
605             }
606
607             // for counting total_replication
608             RuntimeAnalysis.messageCopied();
609
610             if(Time.ready()){
611               Logger.logMessage(m_ID+java.io.File.separator+"MM_copy_send", m_Time+","+m_ID+","+
612 neighbor_max+","+m.dest+","+m.src+","+m.seq);
613             }
614           }
615         }else if(hasTest==ALREADY_HAVE){
616           ; // do nothing
617         }else if(hasTest==REFUSE){
618           nexthop_refuse.add(neighbor_max);
619           dropped++;
620           if(Time.ready()){
621             Logger.logMessage(m_ID+java.io.File.separator+"MM_drop",m_Time+","+m_ID+","+m.dest+"
622 ,"+m.src+","+m.seq+",LIFO");
623           }
624         //  ; // do nothing
625           continue; // try another candidate
626         }else if(hasTest>0){ // DELIVERED
627           m.isDelivered=true;
628           //  m.ttl=EXPIRATION_OF_DELIVERY_NOTIFICATION;
629           m.message_body=null;
630           m_DeliveredMessageBuffer.put(m.seq,m);
631           m_MessageDeliveredTime.put(m.seq,hasTest);
632
633           Long receivedTime;
634           if((receivedTime=m_MessageReceivedTime.get(m.seq))!=null){
635             avgDeliveryTime.add((int)(hasTest-receivedTime));
636             if(avgDeliveryTime.size()>avgDeliveryTimeSize){
637               avgDeliveryTime.removeFirst();
638             }
639           }
640
641           removeMessageFromBuffer(m.seq);
642         }
643
644         break;
645       }
646     }
647
648     m_NewMessageList.clear();
649
650
651   }
652
653   public void postAck(String data) {
654
655   }
656
657   private int rxCount=0;
658   public boolean postMessage(String from,String data) {
659     Message m=new Message();
660     m.deserialize(data);
```

```
  /*  if(count_for_ttl>=10 && averageGTTL!=0){
      if(avgDeliveryTime/count_for_ttl*3>averageGTTL/3){
        m.ttl=avgDeliveryTime/count_for_ttl*3;
      }else{
        m.ttl=averageGTTL/3;
      }
    }else{*/
      m.ttl=Integer.MAX_VALUE;
  //  }

    if(m_ID.equals(m.dest)){
      //Logger.logMessage("MessageRecv", m_Time+","+m.seq+","+m_ID+","+m.src);
      //Logger.logMessage("MessageDelivered", m_Time+","+m_ID+","+m.src+","+(m_Time-Long.parseLong(m.time))); // dest, src, delivery_time
      //System.out.println(m_ID+","+src+","+(m_Time-Long.parseLong(time)));  // dest, src, delivery_time
      m.isDelivered=true;
      recvMessage(m.src,m.message_body);
      m.message_body=null;
      m_DeliveredMessageBuffer.put(m.seq,m);
      m_MessageDeliveredTime.put(m.seq,m_Time);

      RuntimeAnalysis.messageReceived(m.seq);

      if(Time.ready()){
        Logger.logMessage(m_ID+java.io.File.separator+"MM_copy_recv",m_Time+","+from+","+m_ID+","+m.dest+","+m.src+","+m.seq);
      }

      if(Time.ready()){
        synchronized(mutex_MsgsSet){
          DeliveredMsgs.add(m.seq);
        }
      }

    }else if(m_ID.equals(m.src)){
      m_MessageBuffer.put(m.seq, m);
      m_NewMessageList.add(m.seq);
      m_MessageForwardCount.put(m.seq,0);
      m_MessageReceivedTime.put(m.seq,m_Time);
    }else{
      if((rxCount++)>bandWidth){
        return false;
      }

      m_MessageBuffer.put(m.seq, m);
      m_NewMessageList.add(m.seq);
      m_MessageForwardCount.put(m.seq,0);
      m_MessageReceivedTime.put(m.seq,m_Time);
      if(Time.ready()){
        Logger.logMessage(m_ID+java.io.File.separator+"MM_copy_recv",m_Time+","+from+","+m_ID+","+m.dest+","+m.src+","+m.seq);
      }

      // random (depends on hashtable's algorithm)
      if(bufMngMethod==bufMng_Random){
        Object[] keys=m_MessageBuffer.keySet().toArray();
        for(int i=0;i<keys.length&&m_MessageBuffer.size()>MessageBufferSize;i++){
          removeMessageFromBuffer((String)keys[i]);
        }
      }

      // FIFO Buffer Management
      if(bufMngMethod==bufMng_FIFO){
        java.util.ArrayList<java.util.Map.Entry> msgs=new java.util.ArrayList<java.util.Map.Entry>(m_MessageReceivedTime.entrySet());
        java.util.Collections.sort(msgs, new java.util.Comparator(){
          public int compare(Object obj1, Object obj2){
            java.util.Map.Entry ent1=(java.util.Map.Entry)obj1;
            java.util.Map.Entry ent2=(java.util.Map.Entry)obj2;
            Long val1=(Long)ent1.getValue();
            Long val2=(Long)ent2.getValue();
            return val1.compareTo(val2);
          }
        });
        for(int i=0;i<msgs.size()&&m_MessageBuffer.size()>MessageBufferSize;i++){
```

```
737             Message m_drop=m_MessageBuffer.get(msgs.get(i).getKey());
738             if(m_drop.seq.endsWith("test")){
739               continue;
740             }
741
742             removeMessageFromBuffer(m_drop.seq);
743             dropped++;
744             if(Time.ready()){
745               Logger.logMessage(m_ID+java.io.File.separator+"MM_drop",m_Time+","+m_ID+","+m_drop.d
746 est+","+m_drop.src+","+m_drop.seq+",GTTL_ascending");
747             }
748           }
749         }
750
751         // Forward Count Buffer Management
752         if(bufMngMethod==bufMng_MostFwded){
753           java.util.ArrayList<java.util.Map.Entry> msgs=new java.util.ArrayList<java.util.Map.Entr
754 y>(m_MessageForwardCount.entrySet());
755           java.util.Collections.sort(msgs, new java.util.Comparator(){
756             public int compare(Object obj1, Object obj2){
757               java.util.Map.Entry ent1=(java.util.Map.Entry)obj1;
758               java.util.Map.Entry ent2=(java.util.Map.Entry)obj2;
759               Integer val1=(Integer)ent1.getValue();
760               Integer val2=(Integer)ent2.getValue();
761               return val1.compareTo(val2)*-1; // *-1 for descending sort
762             }
763           });
764           for(int i=0;i<msgs.size()&&m_MessageBuffer.size()>MessageBufferSize;i++){
765             Message m_drop=m_MessageBuffer.get(msgs.get(i).getKey());
766             if(m_drop.seq.endsWith("test")){
767               continue;
768             }
769
770             removeMessageFromBuffer(m_drop.seq);
771             dropped++;
772             if(Time.ready()){
773               Logger.logMessage(m_ID+java.io.File.separator+"MM_drop",m_Time+","+m_ID+","+m_drop.d
774 est+","+m_drop.src+","+m_drop.seq+",GTTL_ascending");
775             }
776           }
777         }
778
779         // GTTL Buffer Management
780         if(bufMngMethod==bufMng_Youngest){
781           java.util.ArrayList<java.util.Map.Entry> msgs=new java.util.ArrayList<java.util.Map.Entr
782 y>(m_MessageBuffer.entrySet());
783           java.util.Collections.sort(msgs, new java.util.Comparator(){
784             public int compare(Object obj1, Object obj2){
785               java.util.Map.Entry ent1=(java.util.Map.Entry)obj1;
786               java.util.Map.Entry ent2=(java.util.Map.Entry)obj2;
787               Integer val1=((Message)ent1.getValue()).gttl;
788               Integer val2=((Message)ent2.getValue()).gttl;
789               return val1.compareTo(val2);
790             }
791           });
792           for(int i=0;i<msgs.size()&&m_MessageBuffer.size()>MessageBufferSize;i++){
793             Message m_drop=m_MessageBuffer.get(msgs.get(i).getKey());
794             if(m_drop.seq.endsWith("test")){
795               continue;
796             }
797
798             removeMessageFromBuffer(m_drop.seq);
799             dropped++;
800             if(Time.ready()){
801               Logger.logMessage(m_ID+java.io.File.separator+"MM_drop",m_Time+","+m_ID+","+m_drop.d
802 est+","+m_drop.src+","+m_drop.seq+",GTTL_ascending");
803             }
804           }
805         }
806
807       }
808
809       return true;
810     }
811
812     public void recvMessage(String from,String message_body){
```

```
813     //  System.out.println(m_ID+" received from "+from+"; "+message_body);
814     }
815
816     private java.util.LinkedList<Integer> avgDeliveryTime=new java.util.LinkedList<Integer>();
817     private int avgDeliveryTimeSize=100;
818     private int count_for_msgID=0;
819     public void sendMessage(String dest,String message_body){
820     //  sendMessageWithID(dest,message_body,java.util.UUID.randomUUID().toString());
821       sendMessageWithID(dest,message_body,m_ID+"-"+(++count_for_msgID));
822 /*    java.util.UUID uuid=java.util.UUID.randomUUID();
823       Message m=new Message();
824       m.dest=dest;
825       m.src=m_ID;
826       m.seq=uuid.toString();
827       m.time=Long.toString(m_Time);
828       m.message_body=message_body;
829       //Logger.logMessage("MessageSend", m_Time+","+uuid.toString()+","+dest+","+m_ID);
830       //synchronized(m_MessageBuffer){
831       postMessage(m_ID,m.serialize());
832       //}
833       Logger.logMessage(m_ID+java.io.File.separator+"MM_send",m_Time+","+m_ID+","+dest+","+m.seq+"
834 ,"+message_body.length());
835       synchronized (mutex_numMsgSent) {
836         numMsgSent++;
837       } */
838     }
839
840     public void sendMessageWithID(String dest,String message_body,String msgID){
841       Message m=new Message();
842       m.dest=dest;
843       m.src=m_ID;
844       m.seq=msgID;
845       m.time=Long.toString(m_Time);
846       m.message_body=message_body;
847       if(dynamicTTL==true && avgDeliveryTime.size()>=1){
848         int sum=0;
849         java.util.Iterator<Integer> itr=avgDeliveryTime.iterator();
850         while(itr.hasNext()){
851           sum+=itr.next().intValue();
852         }
853         m.gttl=(int)(sum/avgDeliveryTime.size()*2.5);
854       }else{
855         m.gttl=initial_gttl;
856       }
857       //Logger.logMessage("MessageSend", m_Time+","+uuid.toString()+","+dest+","+m_ID);
858       //synchronized(m_MessageBuffer){
859       postMessage(m_ID,m.serialize());
860       //}
861       Logger.logMessage(m_ID+java.io.File.separator+"MM_send",m_Time+","+m_ID+","+dest+","+m.seq+"
862 ,"+message_body.length()+","+m.gttl);
863       synchronized (mutex_numMsgSent) {
864         numMsgSent++;
865       }
866       System.out.println("gttl="+m.gttl);
867     }
868
869     int testMessageCount=0;
870     public void sendTestMessage(String dest,String message_body){
871       Message m=new Message();
872       m.dest=dest;
873       m.src=m_ID;
874       m.seq=m_ID+"-"+(++count_for_msgID)+"-"+(++testMessageCount)+"-test";
875       m.time=Long.toString(m_Time);
876       m.message_body=message_body;
877       m.gttl=Integer.MAX_VALUE;
878       postMessage(m_ID,m.serialize());
879       Logger.logMessage(m_ID+java.io.File.separator+"MM_send",m_Time+","+m_ID+","+dest+","+m.seq+"
880 ,"+m.message_body.length()+","+m.gttl);
881       synchronized (mutex_numMsgSent) {
882         numMsgSent++;
883       }
884     }
885
886     public long hasTest(String msg_id,String msg_dest){
887       if(m_DeliveredMessageBuffer.get(msg_id)!=null){
888         return m_MessageDeliveredTime.get(msg_id).longValue(); // DELIVERED
```

```
    }

    Message m=m_MessageBuffer.get(msg_id);
    if(m==null){ // NOT_HAVE
      // LIFO Buffer Management
      if(bufMngMethod==bufMng_LIFO){
        if(msg_dest.equals(m_ID)==false&&m_MessageBuffer.size()>=MessageBufferSize){
          return REFUSE;
        }
      }

      return NOT_HAVE;

    }else{ // ALREADY_HAVE
      return ALREADY_HAVE;
    }
  }

  // recv
  public void setNeighborNodes_recv(String[] id){
    neighborNodes_recv=new java.util.ArrayList<String>(java.util.Arrays.asList(id));
  }

  public void printNeighborNodes_recv(){
    java.util.Iterator<String> itr=neighborNodes_recv.iterator();
    while (itr.hasNext()) {
      System.out.print(itr.next() + " ");
    }
    System.out.println();
  }

  public java.util.ArrayList<String> getNeighborNodes_recv(){
    return neighborNodes_recv;
  }

  // send
  public void setNeighborNodes_send(String[] id){
    neighborNodes_send=new java.util.ArrayList<String>(java.util.Arrays.asList(id));
  }

  public void printNeighborNodes_send(){
    java.util.Iterator<String> itr=neighborNodes_send.iterator();
    while (itr.hasNext()) {
      System.out.print(itr.next() + " ");
    }
    System.out.println();
  }

  public java.util.ArrayList<String> getNeighborNodes_send(){
    return neighborNodes_send;
  }

  public String[] temp_find_send(Node[] nodes){
    java.util.ArrayList<String> foundNodes=new java.util.ArrayList<String>();
    for(int i=0;i<nodes.length;i++){
      if(nodes[i].getNodeID().equals(m_ID)){
        continue;
      }
      if(nodes[i].getNeighborNodes_recv().contains(m_ID)){
        foundNodes.add(nodes[i].getNodeID());
      }
    }
    return (String[])foundNodes.toArray(new String[0]);
  }

  // log potential
  public void logPotential(){
    java.util.Iterator<String> itr=m_ThisPotential.keySet().iterator();
    Logger.logMessage(m_ID+java.io.File.separator+"potential_table", "======    POTENTIAL TABLE
of Simulation   ======");
    Logger.logMessage(m_ID+java.io.File.separator+"potential_table", "id,"+m_ID);
    Logger.logMessage(m_ID+java.io.File.separator+"potential_table", "time,"+m_Time);
    Logger.logMessage(m_ID+java.io.File.separator+"potential_table", "size,"+m_ThisPotential.siz
e());
    Logger.logMessage(m_ID+java.io.File.separator+"potential_table", "-- potential vector --");
    while(itr.hasNext()){
```

```
965         String dest=itr.next();
966         String potential=Double.toString(m_ThisPotential.get(dest)[0]);
967         Logger.logMessage(m_ID+java.io.File.separator+"potential_table", dest+","+potential);
968       }
969     }
970
971     // log forwarding table
972     public void logFT(){
973
974       java.util.Iterator<String> itr_nodes=m_ThisPotential.keySet().iterator();
975       while(itr_nodes.hasNext()){
976         String dest=itr_nodes.next();
977
978         Double max_potential_diff=alpha;
979         String neighbor_max=null;
980
981         double[] local_p=m_ThisPotential.get(dest);
982         if(local_p!=null){
983           //synchronized(m_NeighborPotentials){
984           java.util.Iterator<String> itr=m_NeighborPotentials.keySet().iterator();
985           while(itr.hasNext()){
986             String neighbor=itr.next();
987
988             java.util.Map<String,double[]> potentials=m_NeighborPotentials.get(neighbor);
989             if(potentials==null){
990               continue;
991             }
992
993             double[] neigh_p=potentials.get(dest);
994             if(neigh_p==null){
995               continue;
996             }
997
998             if(max_potential_diff<local_p[0]-neigh_p[0]){
999               max_potential_diff=local_p[0]-neigh_p[0];
1000              neighbor_max=neighbor;
1001            }
1002          }
1003          //}
1004        }
1005
1006        if(neighbor_max!=null){
1007          Logger.logMessage(m_ID+java.io.File.separator+"FT_state",m_Time+","+m_ID+","+dest+",1,"+
1008  neighbor_max);
1009        }else{
1010          Logger.logMessage(m_ID+java.io.File.separator+"FT_state",m_Time+","+m_ID+","+dest+",0");
1011
1012        }
1013
1014      }
1015    }
1016
1017    // log Transmissions
1018    public void logTransmissions(){
1019      synchronized(mutex_trns){
1020        Logger.logMessage("Transmissions",m_Time+", "+m_ID+" Advertisement="+trnsAdvertisement+"[B
1021  ytes] Investigation="+trnsInvestigation+"[Bytes] Data="+trnsData+"[Bytes] total="+(trnsAdvertise
1022  ment+trnsInvestigation+trnsData)+"[Bytes]");
1023      }
1024    }
1025
1026    // log DeliveryRate
1027    public void logDeliveryRate(){
1028      synchronized(mutex_MsgsSet){
1029        double DeliveryRate1;
1030        if(numMsgSent!=0){
1031          DeliveryRate1=(double)DeliveredMsgs.size()/(double)numMsgSent;
1032        }else{
1033          DeliveryRate1=0;
1034        }
1035
1036        double DeliveryRate2;
1037        if(DeliveredMsgs.size()+GTTLExpiredMsgs.size()!=0){
1038          DeliveryRate2=(double)DeliveredMsgs.size()/(double)(DeliveredMsgs.size()+GTTLExpiredMsgs
1039  .size());
1040        }else{
```

```
1041          DeliveryRate2=0;
1042        }
1043        Logger.logMessage("DeliveryRate",m_Time+", "+m_ID+", Sent="+numMsgSent+" Delivered="+Deliv
1044 eredMsgs.size()+" GTTLExpired="+GTTLExpiredMsgs.size()+" ExpiredDelivered="+ExpiredDeliveredMsgs
1045 .size()+" DeliveryRate1="+DeliveryRate1+" DeliveryRate2="+DeliveryRate2);
1046      }
1047    }
1048
1049    // calc potential difference histogram
1050    public void calcPdiffHist(){
1051      java.util.Iterator<String> itr_nodes=m_ThisPotential.keySet().iterator();
1052      while(itr_nodes.hasNext()){
1053        String dest=itr_nodes.next();
1054        if(dest.equals("99")==false){
1055          continue;
1056        }
1057
1058        double max_potential_diff=0;
1059
1060        double[] local_p=m_ThisPotential.get(dest);
1061        if(local_p!=null){
1062          //synchronized(m_NeighborPotentials){
1063          java.util.Iterator<String> itr=m_NeighborPotentials.keySet().iterator();
1064          while(itr.hasNext()){
1065            String neighbor=itr.next();
1066
1067            java.util.Map<String,double[]> potentials=m_NeighborPotentials.get(neighbor);
1068            if(potentials==null){
1069              continue;
1070            }
1071
1072            double[] neigh_p=potentials.get(dest);
1073            if(neigh_p==null){
1074              continue;
1075            }
1076
1077            if(max_potential_diff<local_p[0]-neigh_p[0]){
1078              max_potential_diff=local_p[0]-neigh_p[0];
1079            }
1080          }
1081          //}
1082        }
1083
1084        if(max_potential_diff!=0){
1085          synchronized (mutex_pdiffHist) {
1086            Integer key=(int)((max_potential_diff+pdiffHist_shift)/pdiffHist_step);
1087            if(pdiffHist.get(key)==null){
1088              pdiffHist.put(key,1);
1089            }else{
1090              pdiffHist.put(key,pdiffHist.get(key)+1);
1091            }
1092          }
1093        }
1094
1095      }
1096    }
1097
1098    // log potential difference histogram
1099    public void logPdiffHist(){
1100      Object[] keys=pdiffHist.keySet().toArray();
1101      java.util.Arrays.sort(keys);
1102      for(int i=0;i<keys.length;i++){
1103        Logger.logMessage("pdiffHist",((Integer)keys[i]*pdiffHist_step)+" "+pdiffHist.get(keys[i])
1104 );
1105      }
1106    }
1107
1108    // log Buffer Occupancy
1109    public void logBufferOccupancy(){
1110      int delivered=m_DeliveredMessageBuffer.size();
1111      int before=0, now=0, expired=0;
1112      int IDs=0;
1113      Object[] msgs=m_MessageBuffer.values().toArray();
1114      for(int i=0;i<msgs.length;i++){
1115        Message m=(Message)msgs[i];
1116        if(m.ttl<=0){
```

```
1117          expired++;
1118        }else if(m.isForwarded==true){
1119          now++;
1120        }else if(m.isForwarded==false){
1121          before++;
1122        }
1123        IDs+=Integer.parseInt(m.seq.split("-")[1]);
1124      }
1125      if(msgs.length!=0){
1126        IDs/=msgs.length;
1127      }
1128      Logger.logMessage(m_ID+java.io.File.separator+"BufferOccupancy",m_Time+" "+before+" "+now+"
1129 "+expired+" "+delivered+" "+(before+now+expired+delivered)+" "+(before+now)+" "+(expired+deliver
1130 ed)+" "+dropped+" "+IDs);
1131    }
1132
1133 }
```