平成 24 年度特別研究報告書

フェロモンを用いたセンサネットワーク経路再探索方法の改良

龍谷大学　理工学部　情報メディア学科

学籍番号　T090405　坂田 直樹

指導教員　三好 力　教授

内容梗概

センサーネットワークを構築するのにエージェントを蜂に見立て、蜂のフェロモンをヒントに、エージェントが関係するフェロモンを 3 種類使って制御を行う手法がボストン大学の鈴木らによって提案されている。しかし経路中のリンクが切断するという、重大な問題が発生したとき既存手法の制御だけではデータが消失するなどの不具合が起こる場合がある。そこでフェロモンの種類を増やし、新たな制御の可能性について考え、研究に取り組んだ。

提案手法として特定のリンクが切断したとき帯域の大きいリンクを通るように制御できるフェロモンを導入し、リンクが復旧後は元の状態に戻すようにした。実験を行った結果、提案手法はデータ送受信の失敗は起こらないことが確認でき、既存手法の欠点を改良することができた。また総イベント数の実験の結果、リンク切断中は提案手法のほうがより多く迂回するためイベント数が増加するが、リンク復旧後は切断時の迂回の影響が残る既存手法のほうがイベント数が多くなることが確認できた。

目次

第 1 章　はじめに

1.1　研究背景

センサーネットワークは、小型のデータベースシステムであるセンサノードから構成される。ユーザはセンサーネットワーク全体と通信を行い、センサーネットワークに問合せを送信することができる【1】。 また、昆虫たちの意思疎通となるフェロモンの研究も進み様々なアルゴリズムが解明されてきた。例としてスズメバチに特有な行動が生態学的研究により明らかになっている。このような行動が、多くの化学物質によって引き起こされることが、最近の生化学的な研究から明らかになってきた。体外に放出（分泌）することにより、同種の別の個体に特有な反応を起こさせる。情報の伝達をする化学物質をフェロモンと呼び、社会性昆虫特有の行動を引き起こす役割を担っている【2】。これらの特性を合わせた手法がある。蜂がフェロモンを体外に分泌し仲間の蜂にさまざまな行動を起こすように呼びかける習性と同じように、センサーネットワーク上に存在するエージェントがフェロモンを分泌させてネットワークを構築するという手法が提案されている。本研究はボストン大学の鈴木教授が知能情報ファジイ学会で発表した【3】がきっかけとなっている。センサーネットワークを構築するのに蜂のフェロモンを用いて、エージェントを蜂に見立て、エージェントに関係するフェロモンを 3 種類説明していただいた。

1.2　研究目的

鈴木教授が発表した【3】には、経路が消失してしまった後に作用するフェロモンが少なかった。既存技術だけではリンク切断後に、細かな制御を行うことができない。経路が消失するという、重大な問題が発生したとき、既存技術の制御だけでは不具合が起こるのではないだろうかと考えた。そこでフェロモンの種類を増やし、新たな制御の可能性について考え、研究に取り組んだ。

1.3　論文の流れ

第 1 章では本研究の概論について述べる。第 2 章では本実験での基本用語の説明を行う。第 3 章では【3】の既存技術の説明を行う。第 4 章では問題点の指摘と提案手法を述べる。第 5 章では本実験での結果を記載した。第 6 章では考察を行った。

第 2 章　基本用語の説明

2.1　パケットについて

コンピュータ通信において、送信先のアドレスなどの制御情報を付加されたデータの小さなまとまりのことをいい、データをパケットに分割して送受信する通信方式をパケット通信と呼ぶ。データを多数のパケットに分割して送受信することにより、ある 2 地点間の通信に途中の回線が占有されることがなくなり、通信回線を効率良く利用することができる。また、柔軟に経路選択が行えるため、一部に障害が出ても他の回線で代替できるという利点もある【4】。

2.2　キューについて

先に入力したデータが先に出力されるという特徴をもつ、データ構造の一種のことをいう。ちょうど遊園地の乗り物待ちのような構造になっており、データを入れるときは新しいデータが最後尾につき、データを出すときは一番古いデータが優先して出てくる。このように、「最初に入った物が最初に出てくる」というデータの入出力方式は「First In First Out」を略して「FIFO」と呼ばれる。キューは何かの処理を待たせる際によく使われる構造で、たとえば共有の印刷待ち、CPU の計算待ちなどがキュー構造で処理されている。なお、キューとは逆に、最後に入力したデータが先に出力される方式を「First In Last Out」を略して「FILO」と呼ぶ。

2.3　ドロップテイルと輻輳について

ネットワークが混雑している状態を「輻輳(ふくそう：Congestion)」という。センサーネットワークにおけるふくそうとは、キューの中からパケットをネットワーク上に転送するよりも、到着するパケットの数のほうが多く、どんどんキューにパケットがたまっていく状態を指す。

ふくそうが発生すると、キューにパケットがたまっていくが、キューは有限である。キューがパケットでいっぱいになってしまうと、新しく到着したパケットはキューに入りきらず破棄される。これをドロップテイル(Drop Tail)という。ふくそう状態になるということは、ドロップテイルされパケットが失われることをいう【5】。図 1 にドロップテイルの図を記す。

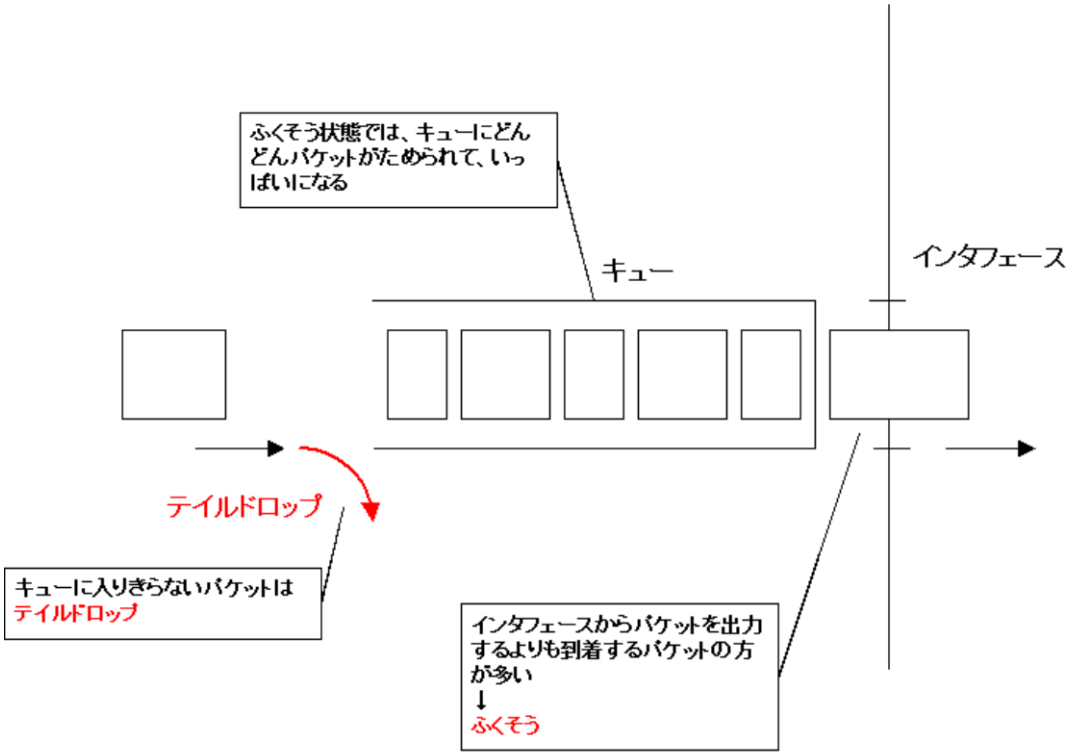

図 テイルドロップ

図 1 ドロップテイルに関する図

2.4 伝搬遅延時間と帯域について

伝送系を電波または電気信号が通過するのに要する時間。送信端から受信端まで信号が進む時間の差が一定の時間以内でないと受信端での信号の再生が難しくなる。ケーブル内を平行して伝送してきたデータを受信端でバッファメモリに収容するとき、受信端において一つのデータの流れとしてとらえるためには、一定の正確な時間内にデータを受信する必要がある。そのために、遅延時間差に限界値をもうける必要がある。また、帯域とは通信速度のことを表す。この値が大きいとデータを一度に多く送ることができる【6】。

第 3 章　フェロモンに関する既存技術

3.1　昆虫たちのフェロモン

3.1.1　情報科学物質について

今回の研究に関係する「フェロモン」は情報科学物質と呼ばれる物のうちのひとつである。情報科学物質は他種の昆虫や植物などとのコミュニケーションの上に成り立っているものである。これらは放出する側に利益をもたらすもの、受け取る側に利益をもたらすもの、双方に利益をもたらすものなど様々である。以下に情報科学物質をまとめた表を記した。

表1　情報科学物質の一覧

フェロモン	同種間に作用する	
アロモン	他種間に作用	放出者に利益をもたらす
カイロモン		受容者に利益をもたらす
シノモン		双方に利益をもたらす
アニュモン		無生物から放出される物質

3.1.2 フェロモンとは

フェロモンの定義とは次のことを指す。「ある特定の生物種の個体から分泌され、同種の他固体に生理的な影響を与え、同種の受け手に情報として特異な行動を起こさせるすべての化合物のこと。」

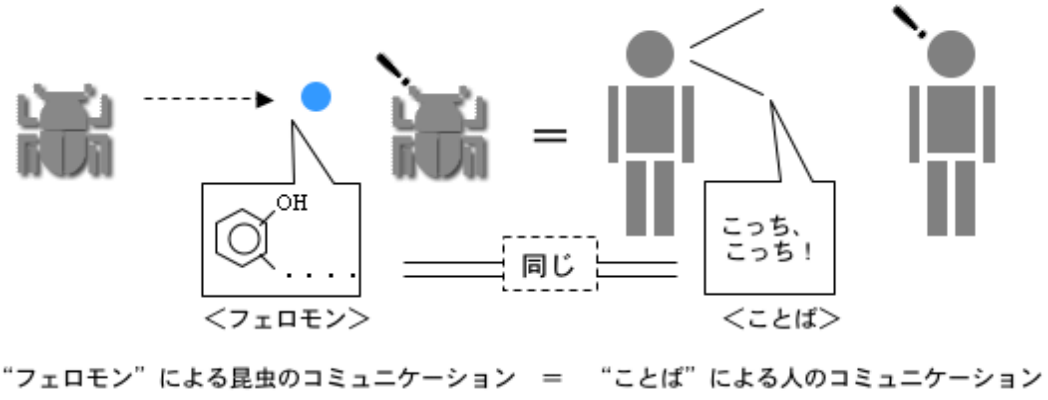

図 2　昆虫と人間のコミュニケーション方法

3.1.3 フェロモンの名前の由来

1959 年カイコガの研究をしていた 2 つのグループがあった。

・カールソンとプデナント

・カールソンとリュッシャー

この二つのグループが 2.1.2 に記した 「ある特定の生物種の個体から分泌され、同種の他固体に生理的な影響を与え、同種の受け手に情報として特異な行動を起こさせるすべての化合物のこと。」をフェロモンと名づけた。名前の理由はギリシャ語の「運ぶ」という意味の「フェライン」と、「刺激する」という意味の「ホルマン」を合わせたものである。

3.1.4 フェロモンの種類

女王フェロモン

女王バチより分泌されるフェロモン。働きバチの産卵を抑制させ、新しい女王バチの育成に関与するフェロモンである。

警報フェロモン（攻撃フェロモン）

働きバチが放出する毒液の中に含まれているフェロモン。ミツバチの針は人を刺すと針とともに毒嚢が動物の皮膚に残る。残った毒嚢の周りにある筋肉は収縮と弛緩を繰り返して針の先から毒液を送り続け、ちぎれた毒嚢の周辺から攻撃フェロモンが出てコロニーの他のハチを攻撃へと向かわせる。

道しるべフェロモン

働きバチから分泌されるフェロモン。巣へもどるための道しるべに使われる。

集合フェロモン

働きバチから分泌されるフェロモン。オスのハチを巣の入り口に誘導する。

性フェロモン

雌が分泌して同種の雄を引きつけるものと，逆に雄が分泌して雌を引きつけるものがある。集合フェロモンと相違点は、その届く距離（誘引距離）が大きく異なる。

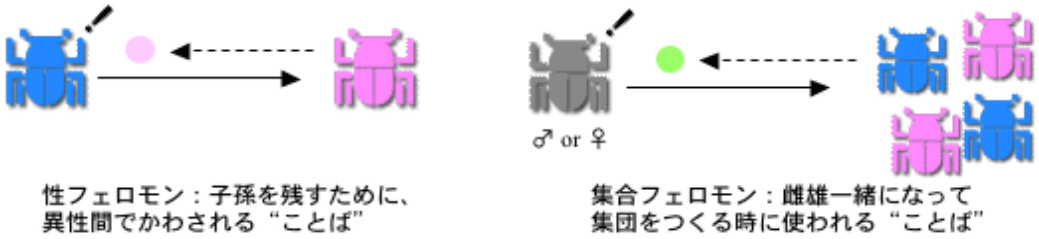

図 3　性フェロモンと集合フェロモンの違い

3.2　フェロモンを用いたセンサーネットワークに関する既存技術

3.2.1 ハチとセンサーネットワーク

ハチの構造と行動は単純なものが多い。これらを組み合わせて用いたセンサーネットワークのアルゴリズムを以下に記す。

・エージェント（ハチ）　　・・・　各アプリ＝エージェントのグループ
・ミドルウェアプラットフォーム（花）　・・・　センサーの読み取りの値、ノードの状態など
・サーバ（蜂の巣）

エージェントについて

センサーデータを、エージェントを用いて収集する技術が研究されている。その一つとしてエージェントのデータを収集するための経路を、フェロモンを使用して決定し、動的変化（経路の消失など）が起こった場合はエージェントがフェロモンを分泌し、状況に対応した行動を起こすようにする手法がある。

エージェントの動作

エージェントはセンサーデータを取得するためにノードを通る。エージェントはデータの収集などを行うとき、経路はフェロモンを使って決める。動的変化（経路の消失など）が起こった場合はエージェントがフェロモンを分泌し、状況に対応した行動を起こすようにする。

フェロモンの種類

フェロモンがエージェントに与える効果は3つある。以下にエージェントに与えるフェロモンの種類と影響を表にして記す。

表 2 フェロモンの種類と効果

フェロモン	重み	どのような動きをするか
ベースステーション	高	短いパスのベースステーションに向けエージェント移動
	低	長いパスのベースステーションに向けエージェント移動
移行	高	エージェントは、他のエージェントがよく使用したパスを優先する
	低	エージェントは、他のエージェントが使用していないパスを優先する
警告	高	エージェントは、以前に失敗したパスを優先する
	低	エージェントは、以前に失敗したパスを避けるようにする

3.2.2 ベースステーションフェロモン

ベースステーションフェロモンは濃度でベースステーションまでの距離を知ることができる。

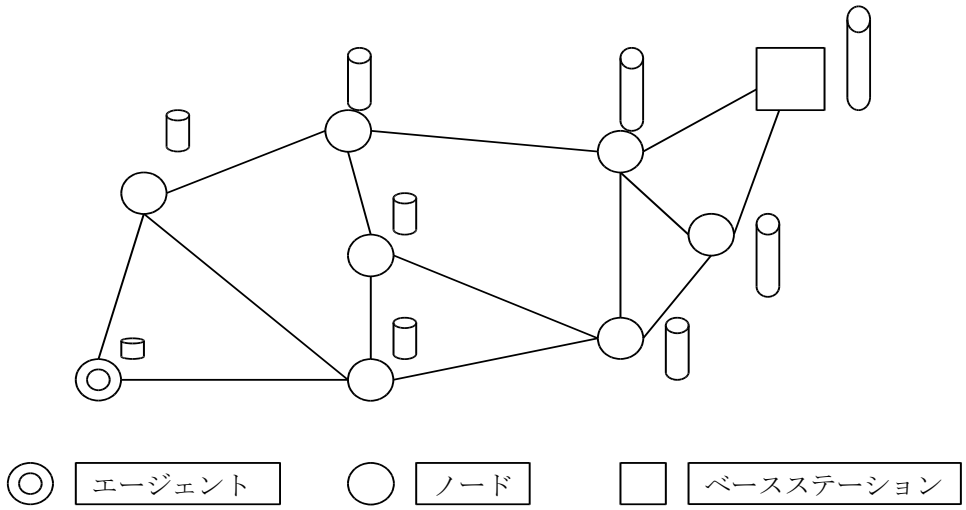

図 4 ノードとベースステーションフェロモンの関係

ベースステーションフェロモンはベースステーションから定期的に個別のノードにフェロモンを伝番していく。図 4 のように濃度が高いほどベースステーションに近くなり、低いほどベースステーションから遠いということである。このベースステーションフェロモンの重要度を高くした場合、短いパスのベースステーションに向けて移動するようになる。反対に重要度を低くした場合は長いパスのベースステーションに向けて移動する。ベースステーションまでの道のりを遠く設定すると、ベースステーションに近いところを使いまくらなくなるので、別段悪くなるわけではない。エージェントはこのフェロモンの濃度勾配を利用してベースステーションに移行していく。

3.2.3 移行フェロモン

移行フェロモンは隣接するノードにエージェントが移行した際にエージェントが放つフェロモンのことをいう。またリンク障害に起因するノードに移動した際にも放ち、エージェントが移行することができなかったノードを参照する。

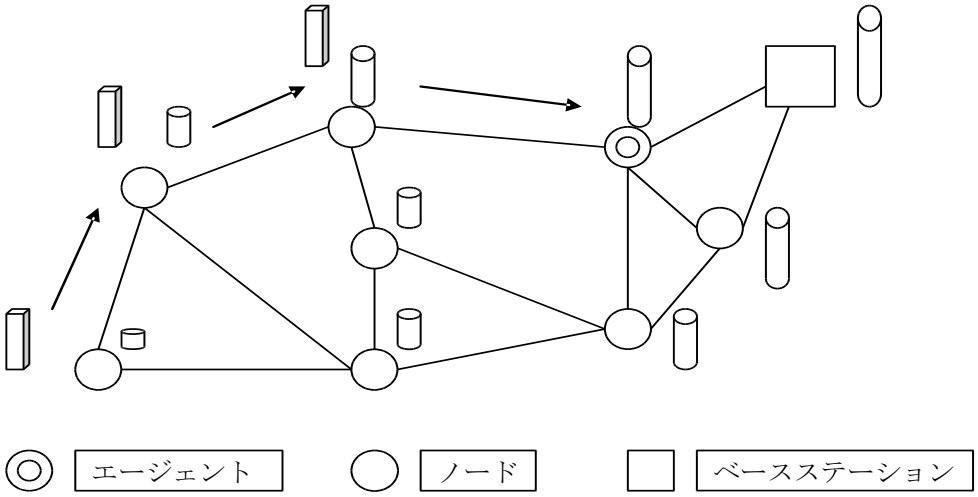

図 5　エージェントと移行フェロモンの関係

図 5 のように、左端のエージェントがベースステーションフェロモンを利用して、ベースステーションに向かって移動した際、エージェントが移行したノード先に移行フェロモンを分泌する。この移行フェロモンの重要度を高くした場合、他のエージェントがよく使用したパスを優先的に使うようになる。反対に移行フェロモンの重要度を低くした場合は、他のエージェントが使用していないパスを優先的に使うようになる。この移行フェロモンは時間が経つと徐々に濃度が下がっていく。

3.2.4 警告フェロモン

警告フェロモンは移行フェロモンと同様、隣接するノードにエージェントが移行に失敗した際にエージェントが放つフェロモンのことをいう。またリンク障害に起因するノードに移動した際にも放ち、エージェントが移行することができなかったノードを参照する。

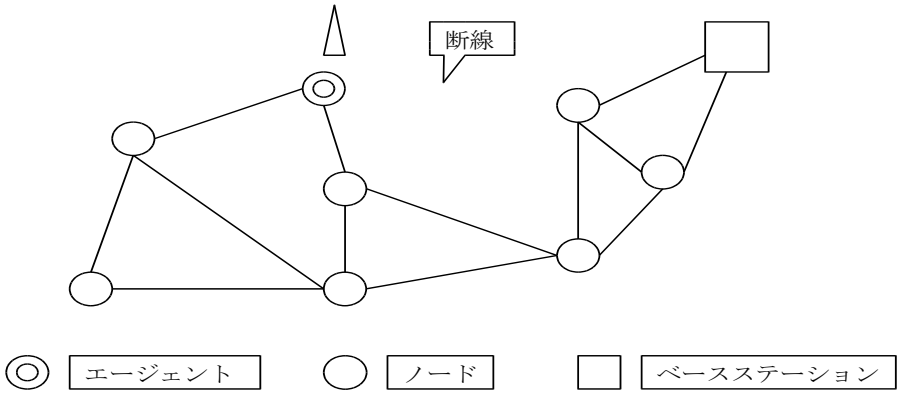

図 6　警告フェロモンを放出する状況

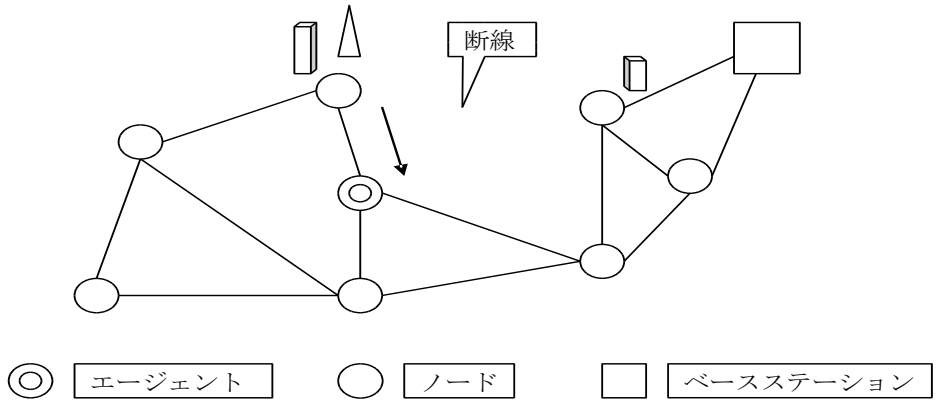

図 7　警告フェロモンとエージェントの動作

図 6 のように移行フェロモンを利用してエージェントが動作していた際にパスが断線し、エージェントがノードに移行することができなかった場合、エージェントがノードに警告フェロモンを放つ。図 7 のように、新たなエージェントが移行フェロモンを利用してルートを選択する際に警告フェロモンを感知した場合、以前通ろうとしていたエージェントのルートを選ばないようにする。

第 4 章　提案手法

4.1　問題点

4.1.1　帯域を無視したルート選択

既存技術では切断リンクが起こると警告フェロモンを発生させて切断リンクをさけるようにする。その後、エージェントはベースステーションフェロモンと移行フェロモンを用いて新たなルートへ移動を試みる。しかし、この方法では切断リンクを避けた後のルートはリンク間の帯域を考慮せず、ベースステーションフェロモンを用いて最短距離のルートを選ぶ。そのため帯域の小さいリンク間のルートを使用した場合、ドロップテイルを起こしてしまう恐れがある。またリンク切断から迂回路までの間にいるエージェントの動作が不安定になる。これは複数のエージェントが経路を同時に探索し、フェロモンの情報の書き換えを行っているからである。

図 8 に示すように既存技術は切断リンクを避けて新たなルートを選択するが、帯域が小さくてドロップテイルが起こったとしても、移行フェロモンの制御でそのルートを選び続ける。よってリンク復旧後も切断リンクを避けてドロップテイルが発生しているルートを選択し続けてしまい、データの消失が起こってしまう。

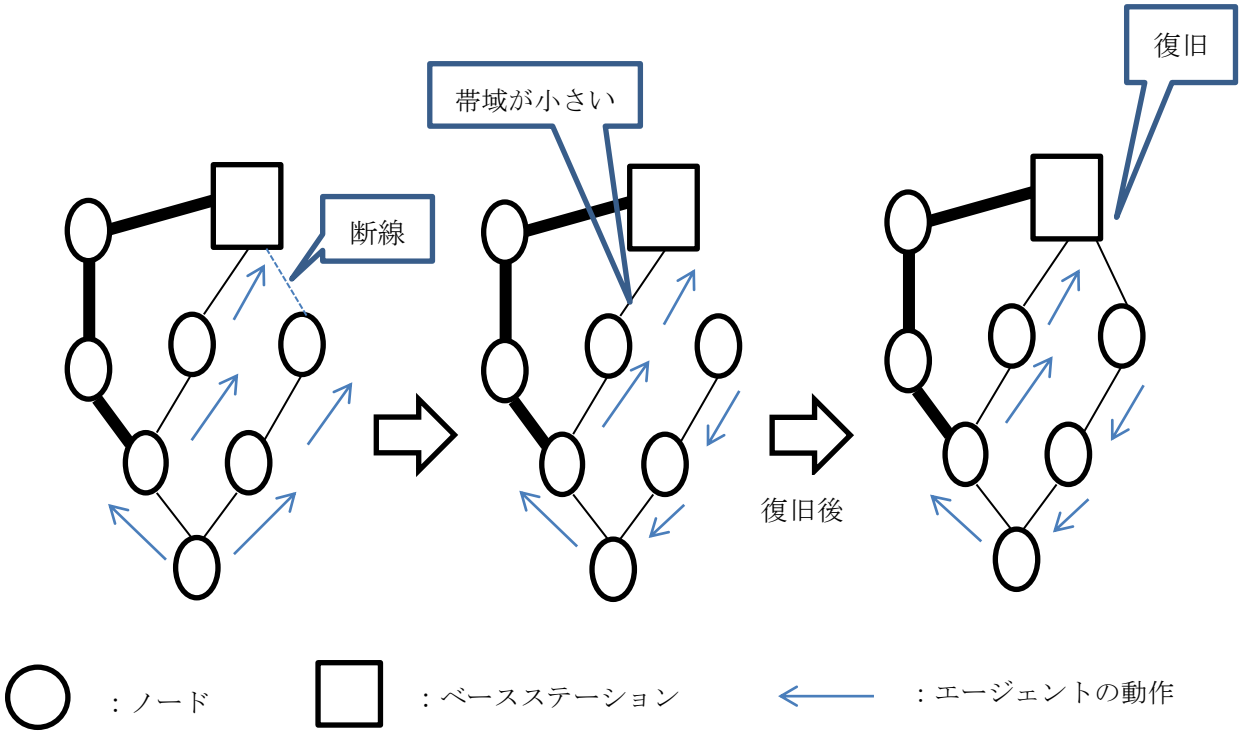

図 8　既存手法の問題点

4.2 提案手法

帯域の小さいリンク間を使用してドロップテイルを起こすことを避けるため、あるリンク間が切断したとき、帯域の大きいリンク間を通るように制御できる新たなフェロモンを導入する。たとえば特定の重要なリンクが切断した場合、リンク間の復旧が行われるまでエージェントを帯域が大きいルートに誘導する。リンク間の復旧後は断線が起こる以前の状態に戻すようにする。この制御を行うフェロモンを集合フェロモンと呼ぶ。以下に図 9 に切断中の集合フェロモンの例を示す。

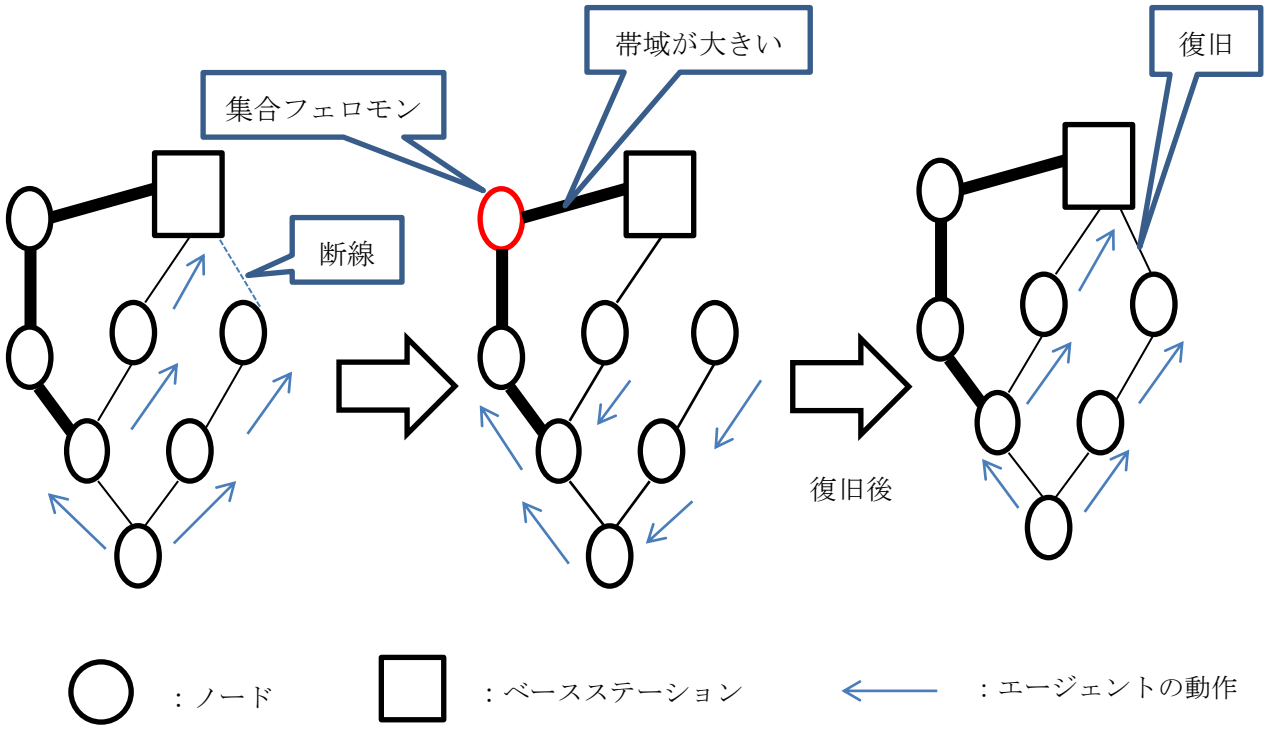

図 9　提案手法である切断中の集合フェロモンでの動作

第 5 章　実験と評価

5.1　実験目的

提案手法と既存手法を比較し、リンク切断に対するデータの送受信の失敗数と総イベント数の関係を評価すること。

5.2　実験準備

今回の実験では NS-2 を用いたネットワークでシミュレートする方法をとった。NS-2 はネットワークシミュレータなので、実際にノードなどの機器をそろえる必要がなく、ノード間のリンクの設定も容易である。以下に NS-2 の詳細を記す。

5.2.1　NS-2とは

NS とは、カリフォルニア大学バークレイ校で開発されたネットワークシミュレータのことで、NS-2 はバージョン 2 をあらわす。NS-2 は C++と Otcl で書かれたオブジェクト指向のネットワークシミュレータでローカル・広域ネットワークをシミュレートするのに役立つ。既存のモジュールを組み合わせてシミュレーションを行うには Otcl のスクリプトを書けば良い。また、新たなモジュールは C++で追加する。【7】

5.2.2　nam について

nam(network animator)はトレースファイルをもとにシミュレーション内容をアニメーション化するツールのことをいう。 ns-2 は Otcl などで記述したスクリプトに従いトレースファイルを出力する。以下に nam でアニメーション化した図を記す。【8】

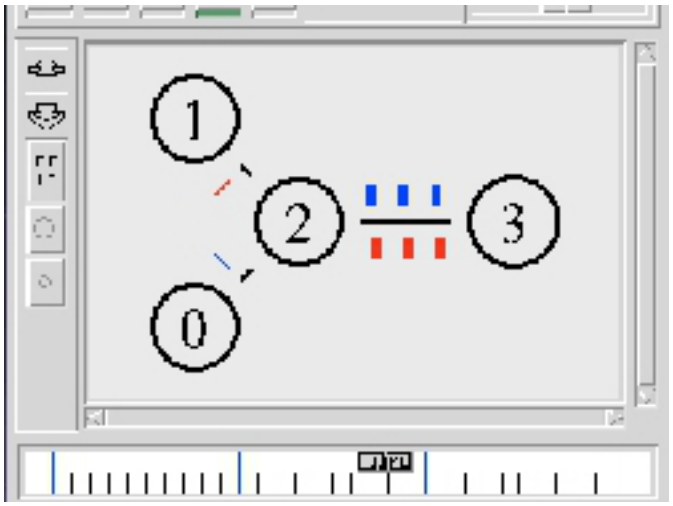

図 10　nam のアニメーション画面

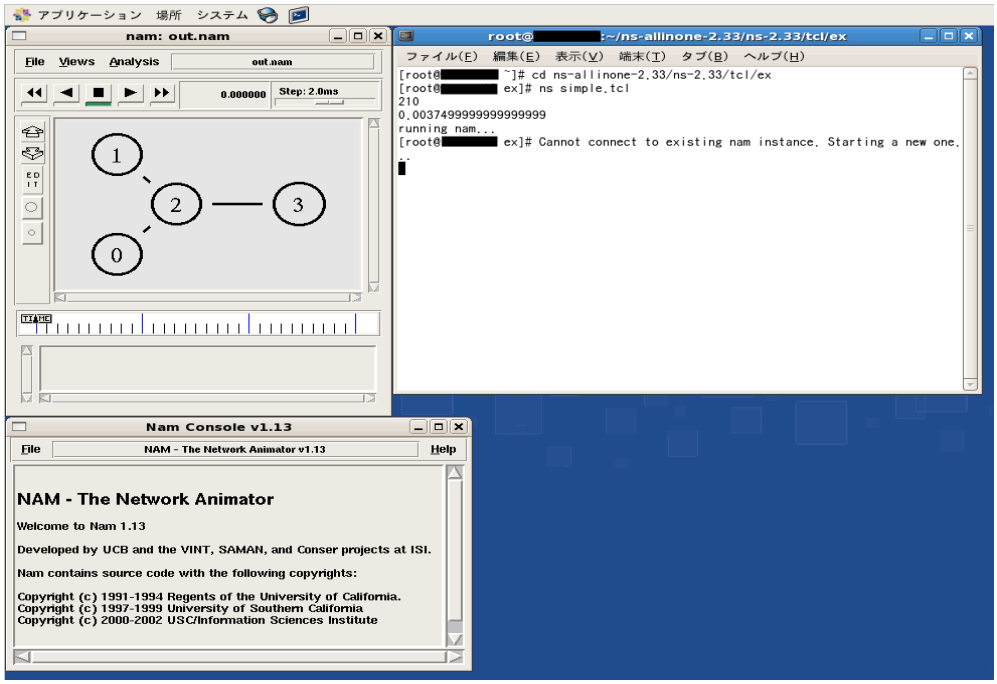

図 11　nam を起動させたときの画面

図 10、図 11 のようにネットワーク図やパケットの流れを視覚化することが可能になっている。シミュレーションを行うには、コンポーネントを C++で記述し、ネットワークトポロジやノードの定義を Otcl 言語で記述するスタイルとなっている。これは実行速度の速い C++と、単純な記述方式の Otclのメリットを組み合わせて高速な実行と手軽なネットワーク定義を実現するためである。 図 10 では青のデータを Node0 から Node2 を介して Node3 に送信している。赤のデータは Node3から Node2 を介して Node1へ送信を行っているのである。このように、送信ノードや中継ノード、受信ノードの設定や通信速度や伝版遅延など、細かな設定をすることもできる。

5.2.3　生成されるファイルについて

NS-2 は Tcl 言語を用いてシミュレーションのシナリオファイルを作成する。そのシナリオファイルをシミュレータに実行させる。UDPを使う場合シミュレーション結果としてout.tr とout.namというファイルに出力される。out.tr というファイルには全てのパケットの転送情報が書き込まれている。個々のパケットがどのように転送されたのかを知ることができる。このファイルを「トレースファイル」と呼ぶ【5】。out.nam というファイルは結果をアニメーションとして表示させるためのデータが書き込まれている。このファイルを nam でアニメーションソフトに実行させると、パケットが転送される様子が表示され上記のように通信の様子を直感的に理解することができる。

シミュレーションで TCP を使う場合のみ、上記で説明したファイル以外に out.tcp というファイルが出力される。out.tcpは1度に送出できるデータ量や再送動作など、各パラメータの時間変化が記録されている。このファイルは TCP の輻輳制御の様子を出力したものである。out.tcp を Gnuplot などでグラフ化することにより、輻輳制御の様子を簡単に知ることができる。このファイルを「TCP のトレースファイル」と呼ぶ。

5.2.4　Tcl 言語について

Tcl/Tk は、スクリプト言語 Tcl と GUI ツールキット Tk からなる 非常に強力な GUI スクリプティング環境である。現在、各種オペレーティングシステム上で動作する。Tcl 言語は、コマンド行のみで構造化文法をフォローしてしまう非常にシンプルな文法を特徴とする。Tk はクロスプラットフォームな GUI 環境としても有名で、Tcl 言語に限らず、Perl、Python、Ruby などの言語環境からも Tk を利用できる【9】。

5.3 評価方法

既存手法と提案手法のデータの送受信の失敗数と各リンク間のイベントの総数を測定し評価した。

各リンク間でのイベントとは以下の 4 つである。

① パケットをキューに入れるイベント

② パケットをキューから出すイベント

③ ノード間のリンクを通りパケットが次のノードに達するイベント

④ パケットはキューに入らずドロップテイルしてしまうイベント

①、②、③の合計を各リンク間のイベントの総数、④を送受信の失敗数とした。総イベント数が多いほどベースステーションまでの経路が長く、データの送受信の失敗数が多いほどドロップテイルが多く発生しデータが消失していることを表している。

5.4 実験方法

今回実験したセンサーネットワークは 5.3.4 で説明した NS-2 で動作する TCL 言語を用いて構築した。実験の流れは以下のとおりである。

① あるリンク間を意図的に断線させる。

② 断線後は新たなルートが決定される。

③ データの送受信が再び行われる。

④ ある時間が経つと切断リンクが復旧する。

⑤ 切断時間を 1s、2s、3sと変える。

上記の実験を繰り返し行ってデータを収集した。本実験はセンサーのデータ値をベースステーションに高速で送り続けることに適したUDPを想定して行った。ネットワークは以下の図のように構築した。

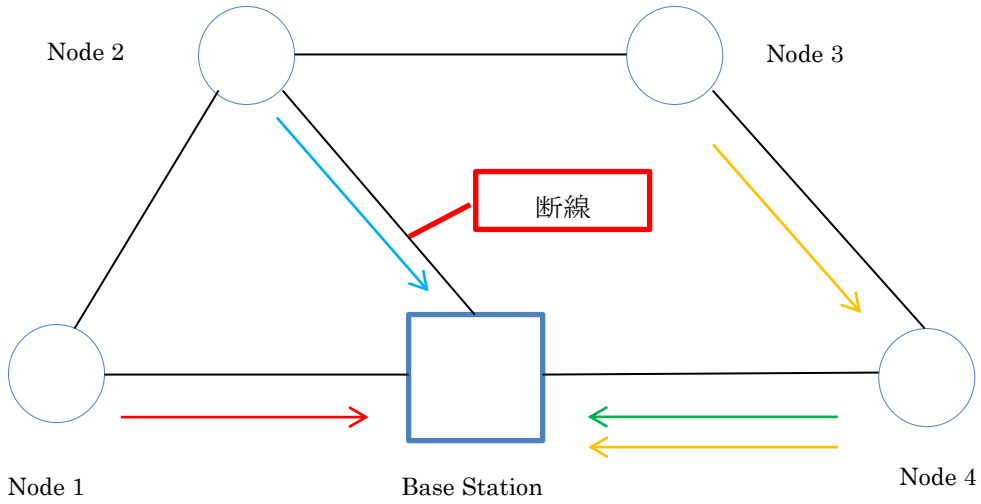

図 12　UDP の場合の実験動作図

提案手法ではリンク切断が起こったとき帯域の大きいリンクへエージェントを誘導するよう、集合フェロモンを設定した。各ノード間のリンクの設定は以下の表にまとめた。

表 3　実験でのパラメータ

ノード数	4
伝搬遅延	2ms
切断時間	1s, 2s, 3s
起動時間	4.5s
切断リンク間	Base Station ～Node2

表 4　ノード間の帯域

ノード間	帯域
Node1～BaseStation	0.8　Mbps
Node2～BaseStation	1.5　Mbps
Node4～BaseStation	1.8　Mbps
Node1～Node2	0.7　Mbps
Node2～Node3	1　Mbps
Node3～Node4	1.4　Mbps

図 12 のようなネットワークと表 3、表 4 のパラメータを設定した条件で実験を行い、切断時間、提案手法と既存手法の各リンクのデータの送受信の失敗数と総イベント数を出した。各切断時間でのリンク切断と復旧の様子を図 13 に示す。

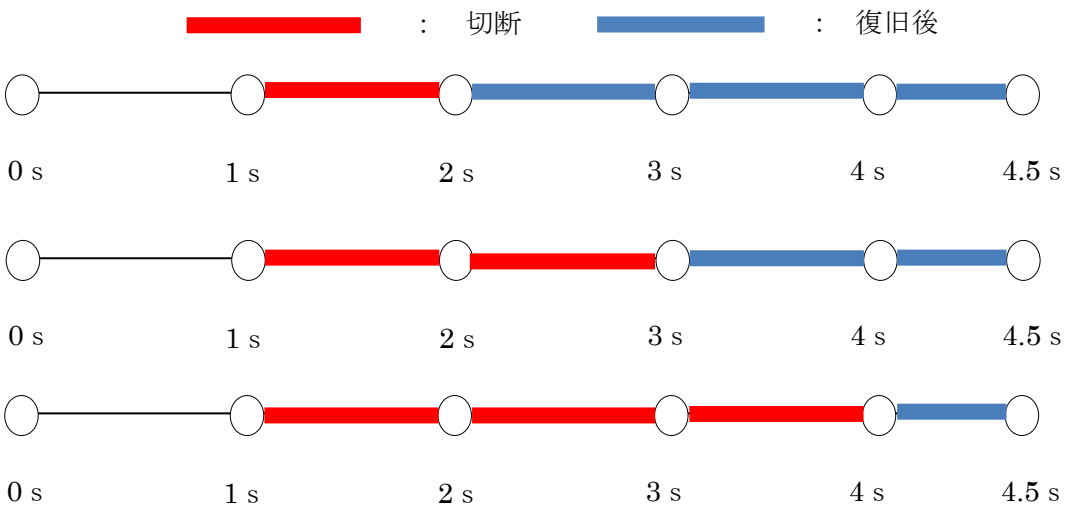

図 13　リンク切断と復旧

5.5 データの送受信の失敗数に関する実験結果

提案手法と既存手法で実験を行い、図 13 の切断中のデータの送受信の失敗数を以下の表にまとめた。

表 5　切断時間におけるデータの送受信の失敗数

手法	切断時間(s)	失敗数
提案手法	1	0
	2	0
	3	0
既存手法	1	9
	2	64
	3	123

表 5 よりデータの送受信の失敗数は既存手法より提案手法のほうが、失敗数が少ない結果となった。提案手法は帯域が大きいリンクにエージェントを誘導したので切断時間に関係なく、ドロップテイルが発生しなかった。既存手法はベースステーションまで最短経路を通るが、帯域が小さいリンクを使用したためドロップテイルが発生した。また切断時間が長いほど失敗数が増加するという結果になった。

5.6 総イベント数に関する実験結果

提案手法と既存手法で実験を行い、図 13 の切断中の総イベント数を以下の図にまとめた。

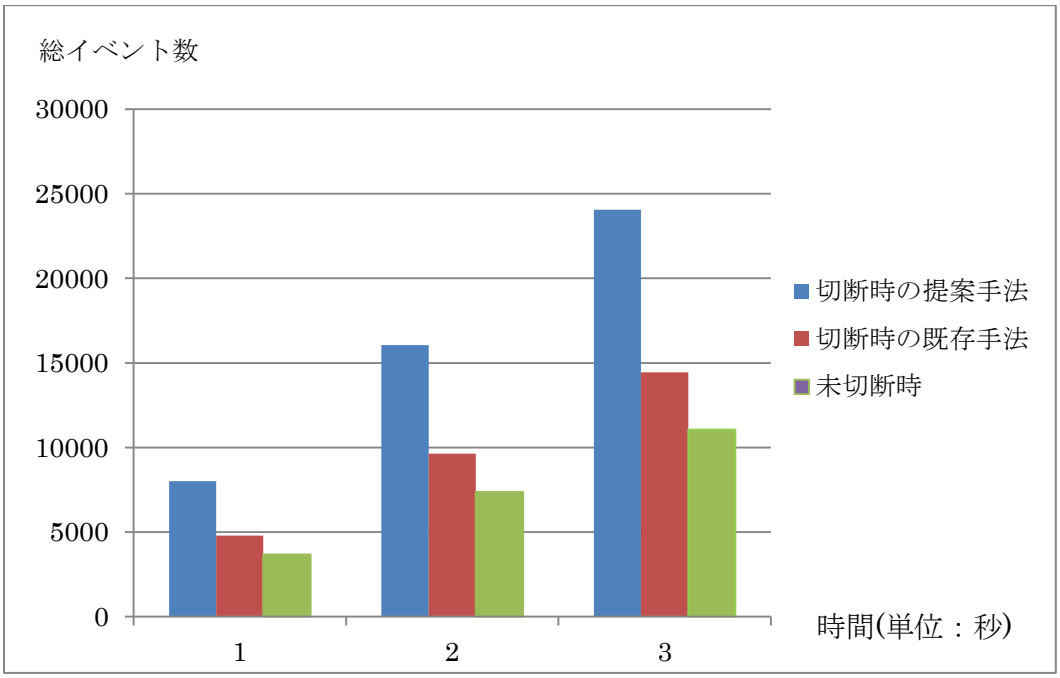

図 14　各手法の切断時と未切断時のグラフ

図 14 を見ると、未切断時より既存手法と提案手法の総イベント数が多い。これはリンク切断によって経路が増え、迂回しているからである。提案手法と既存手法を比べると切断中の総イベント数は提案手法が多い結果になった。つまりベースステーションまでのルートは、既存手法より提案手法のほうが迂回しており、迂回することで各リンク間のイベントが増え、総イベント数が増加する結果となった。

表 6　各手法の切断時における総イベント数

切断時間(秒)	提案手法	既存手法	未切断時
1	8018	4816	3684
2	16036	9632	7368
3	24054	14448	11052

表 6 より、切断時の総イベント数は未切断時と同様、切断時間で等倍しているということがわかる。

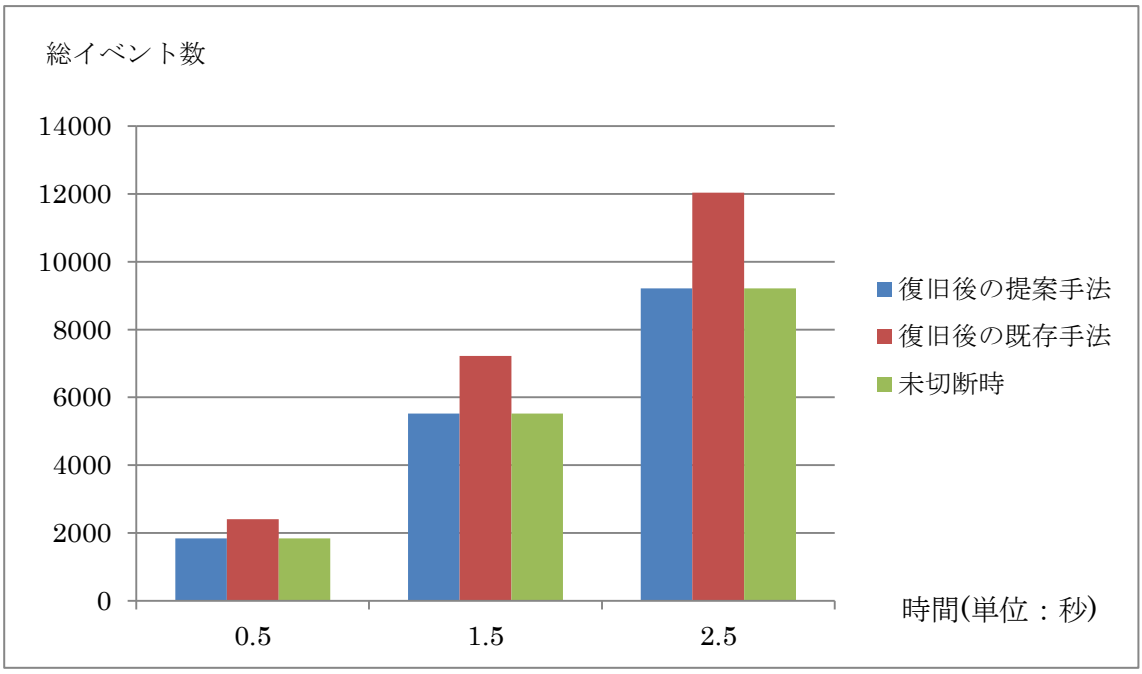

図 15　各手法の復旧後の総イベント数の増加量

表 7　各手法の復旧後の総イベント数の増加量

復旧後の時間(秒)	提案手法	既存手法	未切断時
0.5	1842	2408	1842
1.5	5526	7224	5526
2.5	9210	12040	9210

図 15 と表 7 は図 13 中の復旧後の総イベント数を表している。提案手法の復旧後は切断が起こる前と同じリンク間を経由しているので、未切断時と同じイベント数になっている。しかし既存手法は未切断時や既存手法と比べて総イベント数が多い。これは復旧後もリンク切断時に使用した新たなルートを通り続けているからである。既存手法の新たなリンク間は表 5 より、データの送受信の失敗が起こっている。そのため復旧後もデータの送受信の失敗が起こるという問題が発生している。このことから既存手法より提案手法のほうが復旧後の問題が少ないことがわかった。

第 6 章　考察

今回の実験結果を考察する。提案手法は、リンク切断がある間は指定したノードに全エージェントが経由してくるが、リンク復旧後は切断前の状態に戻る。既存手法はリンク復旧後も切断中と同じ経路を利用し続ける。それにより帯域の小さいリンクを使いづける場合があり、データの送受信の失敗数が増加する傾向がある。このことから帯域が一部小さいリンク間がある場合、提案手法を用いたほうが良いと考えられる。逆に全リンク間の帯域が大きい場合、既存手法を用いたほうが良い。なぜならば総イベント数が少ないリンクを使用するので、ベースステーションまで最短距離によるデータの送受信が行われるからである。

おわりに

本論文では、既存技術の 3 種類のフェロモンを用いた手法の欠点である、帯域を考慮していないルート再探索の手法を改良した。提案手法として特定のリンク間が切断したとき帯域の大きいリンク間を通るように制御できるフェロモンを導入し、リンク間が復旧後は元の状態に戻すようにした。

提案手法はデータ送受信の失敗数の実験により、ドロップテイルによるデータの消失が少なくなることが確認でき、既存手法の欠点を改良することができた。また総イベント数の実験を行い、リンク間のイベント数の増加量を計測した。総イベント数が多いほどベースステーションまで遠回りの迂回をしていることがわかるからである。総イベント数の実験により、ベースステーションまで迂回するほど総イベント数が増加することが確認できた。

本研究により、帯域が一部小さいリンク間がある場合、提案手法を用いたほうが良いと考えられる。反対にリンク間の帯域が十分大きい場合は既存手法を用いたほうが良いと考えられる。なぜならば総イベント数が少ない最短距離でデータの送受信を行うからである。このように状況に応じて手法を使い分けることでデータの消失が起こらない経路を取ることが考えられる。

謝辞

今回の実験、研究は三好力教授、三好研究室の方たちの協力があって行うことができました。プログラムの書き方や動作を行った際の不備な点の指摘など、様々な助言を頂きありがとうございました。

参考文献

【1】情報通信ビジネスのトレンド：センサーネットワーク
http://www.venture.nict.go.jp/trend/sensor/index.html
【2】フェロモンとは何か？
http://light.kakiko.com/sionta/Feromo.htm
【3】 鈴木純一 “Evolutionary Approaches to Gain Self -* Propreties in Wireless sensor Network”
【4】パケットとは
http://e-words.jp/w/E38391E382B1E38383E38388.html
【5】ドロップテイルと輻輳について
http://www.n-study.com/network/congestion.htm
【6】伝播遅延とは
http://kotobank.jp/word/%E4%BC%9D%E6%90%AC%E9%81%85%E5%BB%B6%E6%99%82%E9%96%93
【7】NS-2 について
http://ns2.kano4.com/what-is-ns2.php
【8】NS-2 の基礎
https://sites.google.com/site/106cvi/ns2/base
【9】NS-2 の学習方法
http://www1.bbiq.jp/wibu/craft/learning-ns2.html
【10】 坂田直樹、三好力、“フェロモンを利用したセンサーネットワーク経路再探索方法の改良”、情報処理学会第75回全国大会（2013 発表予定）.

付録

ソースコード

```
set ns [new Simulator]
$ns rtproto DV
#色番号 0 に青, 1 に赤, 2 に白を設定する. 色番号と35行目や52行目で設定したエージェ ントの class が対応しているため, このシミュレーションの場合は udp0 が出すパ ケットは青, udp1 が出すパケットは赤, tcp が出すパケットは白で表示される.
$ns color 0 red
$ns color 1 green
$ns color 2 blue
$ns color 3 yellow
# nam のトレースファイルをオープン
set nf [open out.nam w]
$ns namtrace-all $nf
#ノードの配置
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
set f [open out.tr w]
$ns trace-all $f
set nf [open out.nam w]
$ns namtrace-all $nf
#ノードのリンク、データ転送速度　ノード n0 と n2 の間に帯域が 5Mbps で伝搬遅延が 2ms の全二重(Full-Duplex)のリンクを作成している. 更に, n0 と n2 に DropTail 型の出力キュー　を作成している
$ns duplex-link $n0 $n1 0.8Mb 2ms DropTail
$ns duplex-link $n0 $n2 1.5Mb 2ms DropTail
$ns duplex-link $n2 $n3 1Mb 2ms DropTail
$ns duplex-link $n0 $n4 1.8Mb 2ms DropTail

$ns duplex-link $n1 $n2 0.7Mb 2ms DropTail
$ns duplex-link $n3 $n4 1.4Mb 2ms DropTail
#リンクを表示する際の傾きを設定する. この設定では, n0-n2 のリンクは右上がり, n1-n2 のリンクは右下がり, n2-n3 のリンクは水平に表示される.
$ns duplex-link-op $n0 $n2 orient right-up
$ns duplex-link-op $n1 $n2 orient right-down
$ns duplex-link-op $n2 $n3 orient right
#n2-n3 のリンクのキューに溜るパケットが上(0.5π[rad]=90)方向に伸びるように設定する.
$ns duplex-link-op $n2 $n3 queuePos 0.5
$ns duplex-link-op $n3 $n4 queuePos 0.5
$ns duplex-link-op $n0 $n4 orient right
#TCP 通信において送信側である TCP エージェント tcp と, 受信側で Ack　を返す機能だけを持つ TCPSink エージェント sink が作成され, それぞれノード n0 と n3 に設置される. TCP のバージョンは Tahoe. 56 行目で, tcp と sink が相互に通信するように設定される. tcp から出力されるパケットの class は 2 であるため, これに応答する形で sink が返す Ack パケットの class も 2 となる.
set udp0 [new Agent/UDP]
$ns attach-agent $n1 $udp0
$udp0 set class_ 0
set cbr0 [new Application/Traffic/CBR]
$cbr0 attach-agent $udp0
set udp1 [new Agent/UDP]
$ns attach-agent $n2 $udp1
$udp1 set class_ 1
set cbr1 [new Application/Traffic/CBR]
$cbr1 attach-agent $udp1

set udp2 [new Agent/UDP]
$ns attach-agent $n3 $udp2
$udp2 set class_ 2
```

```
set cbr2 [new Application/Traffic/CBR]
$cbr2 attach-agent $udp2
set udp3 [new Agent/UDP]
$ns attach-agent $n4 $udp3
$udp3 set class_ 3
set cbr3 [new Application/Traffic/CBR]
$cbr3 attach-agent $udp3
set null0 [new Agent/Null]
$ns attach-agent $n0 $null0
set null1 [new Agent/Null]
$ns attach-agent $n0 $null1
set null2 [new Agent/Null]
$ns attach-agent $n0 $null2
set null3 [new Agent/Null]
$ns attach-agent $n0 $null3
$ns connect $udp0 $null0
$ns connect $udp1 $null1
$ns connect $udp2 $null2
$ns connect $udp3 $null3
#シミュレーション開始から0.1秒後にftpが送信を開始するように設定.
$ns at 0.1 "$cbr0 start"
$ns at 0.1 "$cbr1 start"
$ns at 0.1 "$cbr2 start"
$ns at 0.1 "$cbr3 start"
#シミュレーション開始から1.35秒後に，TCPエージェントtcpをノードn0から， TCPSinkエージェントsinkをノードn3から取り外す. これによって送受信を強制的に停止させる. なお，正常に通信を終了させるためには"stop"コマンドを用いるのが一般的であり，その場合は$ns at 1.2 "$ftp stop"と記述する.
$ns at 4.5 "$cbr0 stop"
$ns at 4.5 "$cbr1 stop"
$ns at 4.5 "$cbr2 stop"
$ns at 4.5 "$cbr3 stop"
$ns at 5 "finish"
proc finish {} {
global ns f nf
#ファイルに未出力のトレースをファイルに全て書き出す. これをやらないとトレースの内容が途中で切れてしまう可能性がある.
$ns flush-trace
close $f
close $nf
puts "running nam..."
exec nam out.nam &
exit 0
}
$ns run

set ns [new Simulator]
$ns rtproto DV
#色番号0に青，1に赤，2に白を設定する. 色番号と35行目や52行目で設定したエージェ ントのclassが対応しているため，このシミュレーションの場合はudp0が出すパ ケットは青，udp1が出すパケットは赤，tcpが出すパケットは白で表示される.
$ns color 0 red
$ns color 1 green
$ns color 2 blue
$ns color 3 yellow
# namのトレースファイルをオープン
set nf [open out.nam w]
$ns namtrace-all $nf
set quef [open queue.tr w]
#ノードの配置
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
set f [open out.tr w]
```

```
$ns trace-all $f
set nf [open out.nam w]
$ns namtrace-all $nf
#ノードのリンク、データ転送速度　ノード n0 と n2 の間に帯域が 5Mbps で伝搬遅延が 2ms の全二重 (Full-Duplex)のリンクを作成している. 更に, n0 と n2 に DropTail 型の出力キュー を作成している
$ns duplex-link $n0 $n1 0.8Mb 2ms DropTail
$ns duplex-link $n0 $n2 1.5Mb 2ms DropTail
$ns duplex-link $n2 $n3 1Mb 2ms DropTail
$ns duplex-link $n0 $n4 1.8Mb 2ms DropTail
$ns duplex-link $n1 $n2 0.7Mb 2ms DropTail
$ns duplex-link $n3 $n4 1.4Mb 2ms DropTail
$ns trace-queue $n4 $n0 $quef
#リンクを表示する際の傾きを設定する. この設定では,
n0-n2 のリンクは右上がり, n1-n2 のリンクは右下がり,
n2-n3 のリンクは水平に表示される.
$ns duplex-link-op $n0 $n2 orient right-up
$ns duplex-link-op $n1 $n2 orient right-down
$ns duplex-link-op $n2 $n3 orient right
#n2-n3 のリンクのキューに溜るパケットが上(0.5π[rad]=90)方向に伸びるように設定する.
$ns duplex-link-op $n2 $n3 queuePos 0.5
$ns duplex-link-op $n3 $n4 queuePos 0.5
$ns duplex-link-op $n0 $n4 orient right
#TCP 通信において送信側である TCP エージェント tcp と, 受信側で Ack を返す機能だけを持つ TCPSink エージェント sink が作成され, それぞれノード n0 と n3 に設置される. TCP のバージョンは Tahoe. 56 行目で, tcp と sink が相互に通信するように設定される. tcp から出力されるパケットの class は 2 であるため, これに応答する形で sink が返す Ack パケットの class も 2 となる.
set udp0 [new Agent/UDP]
$ns attach-agent $n1 $udp0
$udp0 set class_ 0
set cbr0 [new Application/Traffic/CBR]
$cbr0 attach-agent $udp0
set udp1 [new Agent/UDP]
$ns attach-agent $n2 $udp1
$udp1 set class_ 1
set cbr1 [new Application/Traffic/CBR]
$cbr1 attach-agent $udp1
set udp2 [new Agent/UDP]
$ns attach-agent $n3 $udp2
$udp2 set class_ 2
set cbr2 [new Application/Traffic/CBR]
$cbr2 attach-agent $udp2
set udp3 [new Agent/UDP]
$ns attach-agent $n4 $udp3
$udp3 set class_ 3
set cbr3 [new Application/Traffic/CBR]
$cbr3 attach-agent $udp3
set null0 [new Agent/Null]
$ns attach-agent $n0 $null0
set null1 [new Agent/Null]
$ns attach-agent $n0 $null1
set null2 [new Agent/Null]
$ns attach-agent $n0 $null2
set null3 [new Agent/Null]
$ns attach-agent $n0 $null3
$ns connect $udp0 $null0
$ns connect $udp1 $null1
$ns connect $udp2 $null2
$ns connect $udp3 $null3
$ns rtmodel-at 1 down $n0 $n2
$ns rtmodel-at 1 down $n0 $n1
$ns rtmodel-at 4 up $n0 $n2
$ns rtmodel-at 4 up $n0 $n1
#シミュレーション開始から 0.1 秒後に ftp が送信を開始するように設定.
$ns at 0.1 "$cbr0 start"
$ns at 0.1 "$cbr1 start"
```

```
$ns at 0.1 "$cbr2 start"
$ns at 0.1 "$cbr3 start"

#シミュレーション開始から 1.35 秒後に，TCP エージェント tcp をノード n0 から， TCPSink エージェント sink をノード n3 から取り外す．これによって送受信を強制的に停止させる．なお，正常に通信を終了させるためには"stop"コマンドを用いるのが一般的であり，その場合は$ns at 1.2 "$ftp stop"と記述する．
$ns at 4.5 "$cbr0 stop"
$ns at 4.5 "$cbr1 stop"
$ns at 4.5 "$cbr2 stop"
$ns at 4.5 "$cbr3 stop"
$ns at 5 "finish"
proc finish {} {
global ns f nf
#ファイルに未出力のトレースをファイルに全て書き出す．これをやらないとトレースの内容が途中で切れてしまう可能性がある．
$ns flush-trace
close $f
close $nf
close $quef
puts "running nam..."
exec nam out.nam &
exit 0
}
$ns run


set ns [new Simulator]
$ns rtproto DV
#色番号 0 に青，1 に赤，2 に白を設定する．色番号と 35 行目や 52 行目で設定したエージェ ントの class が対応しているため，このシミュレーションの場合は udp0 が出すパ ケットは青，udp1 が出すパケットは赤，tcp が出すパケットは白で表示される．
$ns color 0 red
$ns color 1 green
$ns color 2 blue
$ns color 3 yellow
# nam のトレースファイルをオープン
set nf [open out.nam w]
$ns namtrace-all $nf
set quef [open queue.tr w]
#ノードの配置
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
set f [open out.tr w]
$ns trace-all $f
set nf [open out.nam w]
$ns namtrace-all $nf
#ノードのリンク、データ転送速度 ノード n0 と n2 の間に帯域が 5Mbps で伝搬遅延が 2ms の全二重 (Full-Duplex)のリンクを作成している．更に，n0 と n2 に DropTail 型の出力キュー を作成している
$ns duplex-link $n0 $n1 0.8Mb 2ms DropTail
$ns duplex-link $n0 $n2 1.5Mb 2ms DropTail
$ns duplex-link $n2 $n3 1Mb 2ms DropTail
$ns duplex-link $n0 $n4 1.8Mb 2ms DropTail
$ns duplex-link $n1 $n2 0.7Mb 2ms DropTail
$ns duplex-link $n3 $n4 1.4Mb 2ms DropTail
$ns trace-queue $n4 $n0 $quef
#リンクを表示する際の傾きを設定する．この設定では，n0-n2 のリンクは右上がり，n1-n2 のリンクは右下がり，n2-n3 のリンクは水平に表示される．
$ns duplex-link-op $n0 $n2 orient right-up
$ns duplex-link-op $n1 $n2 orient right-down
```

```
$ns duplex-link-op $n2 $n3 orient right
#n2-n3 のリンクのキューに溜るパケットが上(0.5π[rad]=90)方向に伸びるように設定する.
$ns duplex-link-op $n2 $n3 queuePos 0.5
$ns duplex-link-op $n3 $n4 queuePos 0.5
$ns duplex-link-op $n0 $n4 orient right
#TCP 通信において送信側である TCP エージェント tcp と, 受信側で Ack を返す機能だけを持つ TCPSink エージェント sink が作成され, それぞれノード n0 と n3 に設置される. TCP のバージョンは Tahoe. 56 行目で, tcp と sink が相互に通信するように設定される. tcp から出力されるパケットの class は 2 であるため, これに応答する形で sink が返す Ack パケットの class も 2 となる.
set udp0 [new Agent/UDP]
$ns attach-agent $n1 $udp0
$udp0 set class_ 0
set cbr0 [new Application/Traffic/CBR]
$cbr0 attach-agent $udp0
set udp1 [new Agent/UDP]
$ns attach-agent $n2 $udp1
$udp1 set class_ 1
set cbr1 [new Application/Traffic/CBR]
$cbr1 attach-agent $udp1
set udp2 [new Agent/UDP]
$ns attach-agent $n3 $udp2
$udp2 set class_ 2
set cbr2 [new Application/Traffic/CBR]
$cbr2 attach-agent $udp2
set udp3 [new Agent/UDP]
$ns attach-agent $n4 $udp3
$udp3 set class_ 3
set cbr3 [new Application/Traffic/CBR]
$cbr3 attach-agent $udp3
set null0 [new Agent/Null]
$ns attach-agent $n0 $null0
set null1 [new Agent/Null]
$ns attach-agent $n0 $null1
set null2 [new Agent/Null]
$ns attach-agent $n0 $null2
set null3 [new Agent/Null]
$ns attach-agent $n0 $null3
$ns connect $udp0 $null0
$ns connect $udp1 $null1
$ns connect $udp2 $null2
$ns connect $udp3 $null3
$ns rtmodel-at 1 down $n0 $n2
$ns rtmodel-at 4 up $n0 $n2
#シミュレーション開始から 0.1 秒後に ftp が送信を開始するように設定.
$ns at 0.1 "$cbr0 start"
$ns at 0.1 "$cbr1 start"
$ns at 0.1 "$cbr2 start"
$ns at 0.1 "$cbr3 start"
#シミュレーション開始から 1.35 秒後に, TCP エージェント tcp をノード n0 から, TCPSink エージェント sink をノード n3 から取り外す. これによって送受信を強制的に停止させる. なお, 正常に通信を終了させるためには"stop"コマンドを用いるのが一般的であり, その場合は$ns at 1.2 "$ftp stop"と記述する.
$ns at 4.5 "$cbr0 stop"
$ns at 4.5 "$cbr1 stop"
$ns at 4.5 "$cbr2 stop"
$ns at 4.5 "$cbr3 stop"
$ns at 5 "finish"
proc finish {} {
global ns f nf
ファイルに未出力のトレースをファイルに全て書き出す. これをやらないとトレースの内容が途中で切れてしまう可能性がある.
$ns flush-trace
close $f
close $nf
```

```
close $quef
puts "running nam..."
exec nam out.nam &
exit 0
}
$ns run
```