修士論文

アドホックネットワークにおけるTCP通信のグッドプットを改善する早期再送

Early Re-transmission for Improving TCP Goodput over Ad Hoc Networks

平成26年2月6日提出

指導教員　若原 恭　　教授

東京大学工学系研究科　電気系工学専攻

37-126435 小原 知也

概要

近年の車車間通信によるネットワークの考え方や自然災害の際の災害救助への注目により、アドホックネットワークの利用が考えられる場面が増えてきている。そのような多彩な環境の上では、信頼性ある通信が求められる場合もありTCPの利用が考えられるが、アドホックネットワーク上でTCPを用いた場合、グッドプットが大きく低下してしまうことが知られている。この主原因は、ノードの動きによるリンクの途切れと、マルチホップ無線通信によるパケット衝突にあるが、エンドノード間の情報交換のみではこれらの具体的情報を得ることは難しい。そこで、早期のルート再セットアップ、パケット再送を行うクロスレイヤー手法を提案し、TCPのグッドプットの向上に有効であることを示した。

目 次

表 目 次

図 目 次

第1章　序論

1.1　背景と目的

アドホックネットワークとは、通信を行う際に基地局などに依存せず、移動端末によって形成される自己構成型のネットワークである。アドホックネットワークにより通信を行なっている様子を図 1.1 に示す。

アドホックネットワークの特徴としては、

- 無線マルチホップネットワーク
- モビリティー（全てのノードが動きまわる)
- 固定ネットワークインフラが存在しない

などが挙げられる。この固定インフラが使えない場面でも用いることができるという特徴を生かして、災害などでインフラが正常に稼働しない場面やセルラーネットワークの基地局などのインフラが混雑している場合の利用が考えられる。また、地域性を重視する情報の場合には、近くの場所に早く情報を伝えられるアドホックネットワークを利用することも考えられる。例えば以下のような適用例が考えられる。

- 高速道路などで事故が起こった場合に事故現場の映像を周りに拡散して情報を伝える
- 混雑した場所や地下などで周りと相互通信を行う
- 店の周りで宣伝情報を受け取る

ここで高速道路の事故情報伝達について考えると、事故現場から得られる緊急性かつ重要性の高い情報を確実に伝えたい、映像配信により状況をより詳細に伝えたい、といったような要求が考えられる。そこで、TCP を用いることで信頼性ある通信を保証し、また映像配信を早く伝達するのに耐えられる十分なスループットを確保する必要がある。しかし、TCP のスループットはアドホックネットワークでは大きく低下してしまうため、そのスループット改善を行うことが本研究の目的である。

アドホックネットワークについての研究では、MAC 層のプロトコルやルーティングプロトコルなど、単一レイヤーのアプローチについては既に多くの研究が行われている。本研究では、トランスポート層のプロトコルとして TCP を用いた場合の AODV(アドホックネットワークに適したルーティングプロトコル）への影響を加味したクロスレイヤーのアプローチで性能向上を行うことを考える。

TCP とは、end-to-end で信頼できるデータ転送を行うトランスポート層のプロトコルであり、現在最も普及しているプロトコルであるといえる。しかし、もともと TCP は有線ネットワークに最適化されて構成されていたため、近年無線ネットワークが普及してくるにつれて様々な問題が顕在化している。それに伴い、TCP の輻輳制御方式では、TCP Tahoe[9]、TCP Reno、 TCP NewReno[6] のように変更が加えられてきたが、それでも解決していな

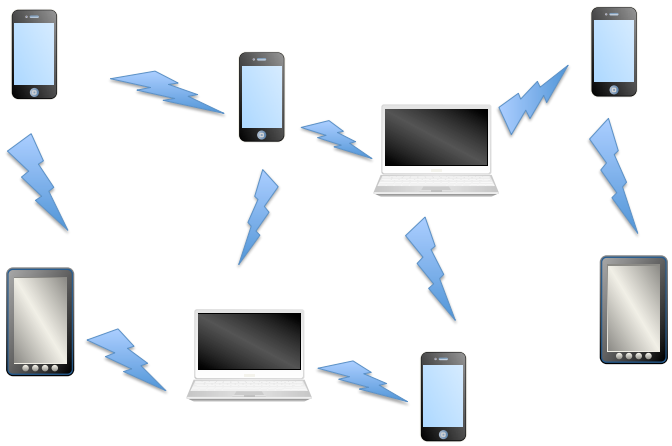

図 1.1. アドホックネットワーク

い問題はまだ多く残っている。そのうえ、アドホックネットワークにおいてはそのような問題の影響が大きくなってしまう場合がある。

TCP では、パケットロスが生じた場合にその原因が輻輳であるという前提になっており、これは有線ネットワークにおいては概ね正しいのだが、無線ネットワークにおいては正しいとは言えない。例えば、無線ネットワークではリンク間の無線品質に応じてある程度リンクエラーが発生してしまうことは避けられない。しかし、このリンクエラーによりパケットロスが生じた場合に、TCP ではそれをどこかで輻輳が起こったと認識して不必要に輻輳ウィンドウサイズを減少させてしまい、結果としてスループットの低下を招いてしまうのである。更に、本研究で想定しているアドホックネットワークでは、ノードモビリティーによりリンクが途切れ、route failure と呼ばれる、ルートの消失による通信障害が発生する可能性もあるため、影響が大きくなってしまう。また、パケットロスの原因としては、リンクエラーの他にも、MAC 層の衝突、route failure などが考えられ、これらも TCP では輻輳と区別することができない。また、fast retransmit パケットのロスが生じた場合にも、輻輳は起こっていないにもかかわらず再送タイムアウトとなってしまい、輻輳ウィンドウサイズの劇的な低下を引き起こすことになる [18]。

TCP において最も大きく輻輳ウィンドウサイズが低下するのは、RTO(Retransmission Time Out) が生じた場合である。これはつまり再送が行われるまでの時間であり、この時間はこれまでの通信の RTT を用いて推測することにより設定される。しかし、最小の RTO というものが設定されており、これが実際の RTT に対して大きめの値であることが多いため、輻輳ウィンドウサイズの低下だけでなく、無駄な時間を大幅にロスする可能性がある。また、再送と輻輳ウィンドウサイズの低下が行われるタイミングとしては他にも、duplicate ack と呼ばれる重複した 3 つの ack を受け取った場合がある。つまり、パケットロスが発生

したが duplicate ack を受け取ることができなかった場合には非常に大きな時間を待たなければならないことになる。

アドホックネットワークで TCP を用いた場合、グッドプットが大きく低下してしまう。ここでグッドプットとは、アプリケーションが実際に到達できるスループットのことである。例えば、トランスポート層で再送が行われた場合にそれ以前のパケットも正常に送受信が行われていた場合には二重にパケットが送られることになり、通常のスループット計測ではこれを二重に計測してしまい、アプリケーションから見た本来のスループットとは異なるものとなってしまうことがある。それに対してグッドプット計測では、重複したパケットに対しては一度目しか計測を行わないことにより本来の性能を測ることのできる評価基準となっている。アドホックネットワークにおける TCP でグッドプットが低下してしまう主原因は、ノードの動きによるリンクの途切れと、マルチホップ無線通信によるパケット衝突にある。エンドノード間での情報交換のみではこれらの具体的情報を得ることは難しいため、例えば一時的にリンクが切断した場合に TCP ではタイムアウトまでの大きな待ち時間が必要となり、更にタイムアウトにより輻輳ウィンドウが大幅に低下してしまう。そこで本研究では、他のレイヤーの情報を用いるクロスレイヤアプローチにより TCP のグッドプットを改善する。

1.2 位置づけ

本研究は、アドホックネットワーク上で AODV と TCP を併用した場合を対象として、TCP の RTO によるリソースの浪費、不適切な輻輳ウィンドウサイズの低下、という二つの問題を解決してスループットの向上を計る。本研究の手法は、アドホックネットワーク上でルーティングプロトコルとして AODV を用いた場合を想定しているが、リアクティブ型のルーティングプロトコルに対しては適用可能である。

1.3 本稿の構成

本稿では、以下の構成で本研究について述べていく。2 節では、アドホックネットワークにおけるルーティングプロトコルとして現在最もよく知られている AODV について概説し、関連研究について述べる。3 節では、アドホックネットワークにおいて TCP を用いた場合に発生する問題を解決するための手法を提案する。4 節では、実験方法について延べ、その結果について評価・考察を行う。最後に、5 節でまとめ、今後の課題について述べる。

第2章　関連研究

2.1　Ad Hoc On-demand Distance Vector

これまでのルーティング方式では、各ルータが定期的に経路情報を交換し、ネットワーク全体のトポロジを管理する方式が多く取られてきた。しかし、アドホックネットワークでは通信帯域幅が十分でないことやノードモビリティーにより頻繁にルートが変化することからそういった方式は非効率である。

そこで注目されたのがオンデマンド型のルーティングプロトコルであり、アドホックネットワークでは Ad Hoc On-demand Distance Vector(AODV)[16] というプロトコルが多く用いられている。このルーティングプロトコルでは、通信要求があった時のみ経路を確立するため、不必要なルートを確立するための制御メッセージを省くことができ、効率が良い。AODV ルーティングプロトコルの概要について以下に示す。

AODV では、主に RREQ(Route Request) メッセージ, RREP(Route Reply) メッセージ, RERR(Route Error) メッセージの 3 種類の制御メッセージが使われる。

新しい送信先への通信要求があった時、RREQ メッセージのブロードキャストを行い、ルートのセットアップを始める。目的の送信先では、RREQ メッセージが届くと RREP メッセージを送信元へユニキャストで送り返す。この RREP メッセージが返って行く時に中間ノードも含めたルート上の全てのノードで、その送信先に送る際に次のホップで送るべきノードが定まることになる。また、RREQ メッセージを送る際に中間ノードがその送信先に対する新しい経路を保持していた場合には、送信先の代わりに RREP を送信元へ送り返すことが許されている。ここで定まった経路は一定時間利用されない場合には削除されることになっている。

いずれかの中間ノードでリンクエラーを検知した場合には、その経路を無効として RERR メッセージをブロードキャストすることになっている。リンクエラーを検知した場合だけではなく、経路がない送信先へのデータパケットを受け取った場合やその送信先への経路を持っている RERR メッセージを受け取った場合にも RERR メッセージが送られる。

RREQ メッセージと RREP メッセージの流れを以下の図で説明する。

2.1.1　RREQ

図 2.1 においてノード s がノード d 宛のデータを送信しようとした場合を考える。ノード s ではデータの送受信の必要が生じたので、ルート探索のために RREQ のブロードキャストを行う。この際に効率化のために徐々にホップ数を増やし、探索範囲を広げてフラッディングさせていく。重複した RREQ は送信されないよう処理される。この過程で中間ノードでも送信元 (ノード s) への経路表を更新していく。こうして徐々にフラッディング範囲が広がりノード d まで最初の RREQ が届いたらノード d は RREQ を返すことになる。

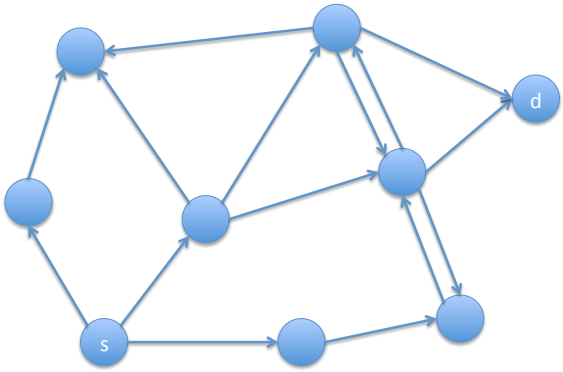

図 2.1. RREQ

2.1.2 RREP

ノード d で RREQ を受け取ったら、今度は図 2.2 のようにユニキャストでノード s 宛に RREP を送り返す。この時、中間ノードでは RREQ の転送時に作成された経路表を用いてノード s に転送を行なっていく。この過程で今度はノード d への経路表が更新されていく。これにより最終的にその時点で最も適切と考えられる双方向のルートが確立される。

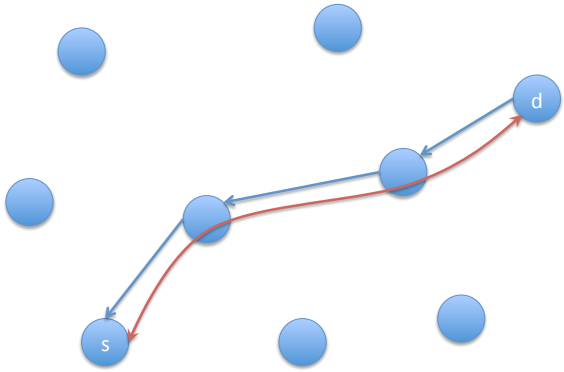

図 2.2. RREP

2.2 TCP

TCP はコネクション指向型の信頼性のあるサービスを提供するプロトコルである。TCP を利用する 2 つのエンドノードは相互に TCP コネクションを確立してからデータの送受信を行う。これをコネクション指向型のプロトコルという。TCP の主な機能は以下のようなものである。

TCP の主な機能

- アプリケーションから送られてきたデータを分割して適切なサイズとする。
- ソースノードではパケットを送信する際にタイマーが設定され、時間内に送信先のノードから正常にパケットが受信されたことを知らせる ACK(確認応答) が返ってこなければパケットの再送を行う。
- ヘッダとデータでチェックサムを利用し、データが転送中に変化していないかどうかを検出する。
- 送信先のノードのバッファ容量に応じたフロー制御を行う。
- 輻輳制御を行う。

2.2.1 輻輳制御方式

TCP の主な機能の一つである輻輳制御方式について解説する。まず例えばあるノードに大量のデータパケットが集まり、そのノードで処理できる量を越えてしまった、いわゆるネットワーク中のどこかで混雑が生じているような状態を輻輳という。特定のノード間の回線のスピードが他の回線のスピードより小さい場合や、あるノードに複数のノードからデータパケットが送られてくるような場合に発生しやすい。TCP ではエンドノードにおいて通信量を制限することにより輻輳の問題に対処している。

スロースタート

TCP ではまずエンドノード間のコネクションが確立されると、ネットワークに送り出すパケットの通信速度を送信先のエンドノードから返される ACK の通信速度と同期させるためにスロースタートアルゴリズムを用いる。スロースタートでは、ソースノードがあるタイミングにネットワーク中に同時に送出させていてよいパケットの数と対応する輻輳ウィンドウサイズ（cwnd とも呼ばれる）を用いる。スロースタートの段階では、送信元のノードで ACK を受け取る度に輻輳ウィンドウサイズを 1 パケット長、つまり 1MSS(Maximum Segment Size) ずつ増加させる。エンドノード間のコネクションが確立されると最初に輻輳ウィンドウは 1MSS で初期化される。まずソースノードは 1 パケットの転送から始め、その ACK を受け取ったら輻輳ウィンドウサイズは 2MSS に増加し、2 つのパケットが送信可能となる。この 2 つのパケットに対する ACK が返ってきたら輻輳ウィンドウサイズは 4MSS へと増やされる。このように輻輳ウィンドウサイズは、実質的には指数関数的に増加していくこととなる。

輻輳回避

スロースタートにはしきい値の ssthresh があり、輻輳ウィンドウサイズがスロースタートの段階で増加していき ssthresh を越えると輻輳回避の段階へと移り、輻輳ウィンドウサイズの増加方法が異なるものとなる。輻輳回避の段階では、ACK の受信ごとに輻輳ウィンドウサイズ (cwnd) を 1/cwnd ずつ増加させる。これは、輻輳ウィンドウサイズが ssthresh 以下の時にはスロースタートとして指数関数的に増加していき、ssthresh を越えると輻輳回避の段階として線形的に増加していくことを示している。コネクションが確定した段階での ssthresh の初期値は 65535 バイトとなっている。

2.2.2 タイムアウトと再送

TCP では、RTO の値に応じたタイムアウト、もしくは設定された一定数（一般には 3 つ）の duplicate ack を受け取った場合にパケットの再送を行うことになっている。ここで、パケットの再送と同時に輻輳ウィンドウサイズも減少させることになるが、その減少方法についてはタイムアウトによるものか duplicate ack によるものかによって異なる。RTO の値の計算方法については後に述べる。ここで、duplicate ack とは要するに重複した同じ ACK のことであり、送信先のエンドノードでシーケンス番号の順番通りでないパケットを受け取った場合に生成される。この duplicate ack の目的は、パケットの順番が想定と違って送られ

てきたことと、次に送られることが期待されるシーケンス番号を送信元のエンドノードに知らせることが目的である。duplicate ack が、パケットの消失によるものなのか、もしくは単にパケットが入れ替わって送られてきただけのものなのかを判断することができないため、一般には 3 つ以上の duplicate ack を受け取った場合には、パケットの消失の影響である可能性が高いとして期待されるシーケンス番号のパケットから再送を行うことになっている。タイムアウトを待つことなく、複数回の duplicate ack によって再送を行うことを Fast Retransmit アルゴリズムと呼ぶ。

タイムアウトとなりパケットの再送が行われる場合には、輻輳ウィンドウサイズは 1MSS に設定され再びスロースタートが始まることとなる。それに対して duplicate ack による再送が行われる場合には、スロースタートは行われずに輻輳回避の段階となる。具体的には以下のようなアルゴリズムとなる。

1. 3 つの duplicate ack を連続して受信した場合
   - ssthresh を現在の輻輳ウィンドウサイズの半分に設定する
   - 消失したと思われるパケットを再送する
   - 輻輳ウィンドウサイズを $ssthresh$ + パケットサイズ × 3 に設定する
2. 更に duplicate ack を受信する度に輻輳ウィンドウサイズをパケットサイズずつ増加させる（パケットが送出される）
3. 新しいシーケンス番号に対応する ACK を受信したら輻輳ウィンドウサイズを ssthresh に設定し輻輳回避の段階とするこれは再送したパケットに対する確認応答を示している

RTO の計算方法

RTO は ACK が到着する度に RTT の測定値を用いて更新される。RTT の測定値 M に対して計算式は以下のようになる。

RTO の計算方法

$$Err = M - A \tag{2.1}$$

$$A \leftarrow A + gErr \tag{2.2}$$

$$D \leftarrow D + h(|Err| - D) \tag{2.3}$$

$$RTO = A + 4D \tag{2.4}$$

ここで、A は平滑化された RTT、つまり直近の RTT の測定値の重みを大きくした平均値である。D は平滑化された平均偏差、Err は実際の測定値と平滑化された RTT の評価値との差分である。係数 g は 1/8, 係数 h は 1/4 に設定される。係数が大きくなると、RTT の変化に対して RTO の変化が激しくなる。

2.2.3 TCP の問題点

TCP の輻輳回避アルゴリズムでは、パケットの消失は送信元と送信先の間のネットワーク上のどこかで輻輳が発生していることを示す信号となるという前提がある。しかし実際には輻輳が発生していない場合でもパケットロスが生じることはあるため、その場合は不必要な輻輳ウィンドウサイズの低下を招くこととなる。

2.2.4 TCP の輻輳ウィンドウサイズの最大値

マルチホップ無線通信で TCP を用いている場合には、輻輳ウィンドウサイズが大きくなりすぎると隠れ端末問題やさらし端末問題による MAC 層の衝突が生じる頻度が大幅に上がりスループットが低下する [7]。図 2.3(a) においてノード A とノード B がお互いの伝播到達範囲内にいないとき、同時に B にデータを送信しようとすると衝突が起きてしまう。これを隠れ端末問題という。これに対処するために RTS/CTS を用いた仮想キャリアセンスが用いて衝突を減らすことができる。しかし、図 2.3(b) のように A,B,C,D のノードが並んでいるとき、ノード A からノード B へデータ送信が行われるとすると、まずノード A からノード B へ RTS-CTS-DATA-ACK ハンドシェイクが行われる。この時、ノード C はノード B から CTS を受け取り NAV 期間となる。この期間の間、ノード C は何も通信することができない。つまりノード A からノード B へのデータ転送が完了するまでノード C は通信を行

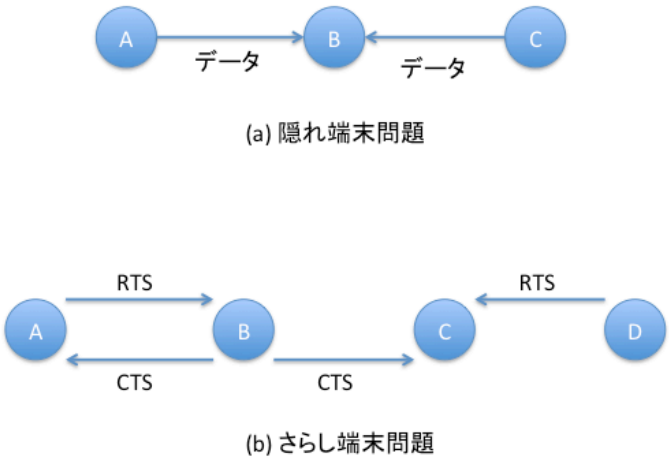

図 2.3. 隠れ端末問題とさらし端末問題

うことができなくなるが、その期間にノード D からノード C へデータ送信が行われようとすると、ノード D はノード C へ RTS を送信する。しかし、ノード C は NAV 期間のために返答することができない。ノード D は数回 RTS を送っても返答が確認できない場合、該当するパケットは破棄されてしまう。この問題が頻発することによりスループットが低下してしまうのである。また、MAC 層の衝突の問題に対してバッファオーバフローによるスループット低下の影響はほとんどない。

図 2.4 によると、ホップ数 7 の時には輻輳ウィンドウサイズの最大値が 3 程度の時に MAC 層の衝突を最も抑えることができ、スループットが最大となっている。ホップ数が 15 の時にも輻輳ウィンドウサイズの最大値は 6 程度で十分である。

2.3 マルチパスルーティング

TCP ではアドホックネットワークにおいてはスループットが悪くなるため、これをマルチパスルーティングにより解決しようという研究がなされてきた。マルチパスルーティングとは、ソースノードとデスティネーションノード間で、複数のルートを保持しておき、もしあるルートで問題が起きた場合にも別のルートを用いて通信を行うことができるというものである。しかし、マルチパスルーティングを用いた TCP のグッドプットを評価すると、ホップ数が大きい場合にはパフォーマンスが向上するが、ホップ数が小さい場合にはパフォーマンスが低下する場合があることが分かった [23]。それにより実際には、ホップ数が大きいものと小さいものが混在するようなネットワークの場合には、全体のグッドプットが低下して

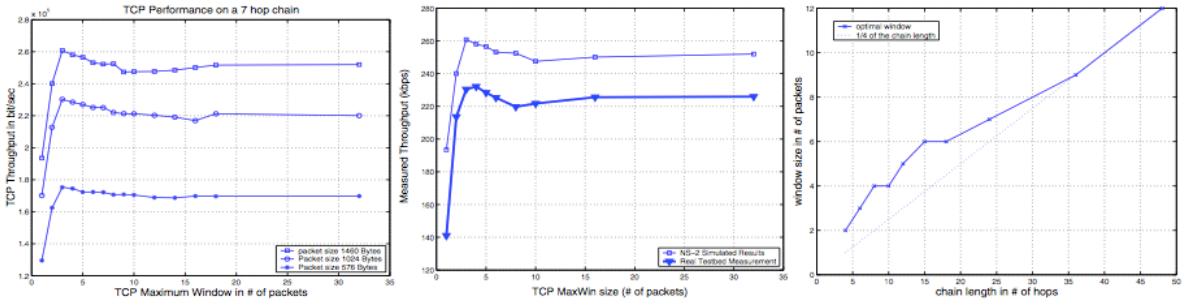

図 2.4. 輻輳ウィンドウサイズの最適値 [Zhenghua Fu, et al, 2003]

しまう場合が多い。ホップ数が小さい場合にパフォーマンスが低下してしまう原因としては以下のようなものがあると考えられる。

- アドホックネットワークでは通常、共有チャネルを用いるため、お互いに干渉してしまう
- ホップ数が多い通信ではパケットロスなどへの対応が遅くなってしまう

また、alternate path routing(一つ目のルートが失敗した場合に次のルートを用いる) を用いた場合と、複数のルートを用いて同時にデータを送信する場合とではあまり性能が変わらなかった。これは、ソースノードやデスティネーションノードにおいては同時に一つの通信しかできないため、結局ここにボトルネックがあるためであると考えられる。

更にマルチパスルーティングプロトコルによって保持されたバックアップのルートは優れたルートでない場合（通信速度が遅い等）がある。ルートのセットアップからしばらくしてリンク切断が起きた時にバックアップのルートを使うのは不適切な場合も多いと考えられる。

2.3.1 Ad hoc On-demand Multipath Distance Vector

Ad hoc On-demand Multipath Distance Vector(AOMDV) とは、AODV をマルチパスルーティングに拡張したルーティングプロトコルである [13]。AOMDV では経路のループを防ぐために新たなフィールドを用意している。このフィールドはある宛先に対する経路のホップ数のうち最大のものを示している。このフィールドの値より小さいホップ数の経路のみを保持する。また、AOMDV ではリンクディスジョイントな経路、つまりルート間でリンクが重複しないような経路を構築する。しかし、リンクディスジョイントな経路であってもソースノードとデスティネーションノードのそれぞれ隣接ノードとの通信に起因するボトルネックはあると考えられる。

第3章　提案手法

3.1　既存手法の問題点

1章で述べたとおり、既存手法の問題点はTCPにおける不適切な輻輳ウィンドウサイズの低下と、TCPのRTOによって生じるリソース利用の非効率な時間帯の発生にある。後の議論を正確にするため、この問題について詳細に解析する。

3.1.1　不適切な輻輳ウィンドウサイズの低下

アドホックネットワークでは、高いビットエラー率やノードモビリティーの影響によりリンクの切断が繰り返されることで、リンクエラーが起こりやすくなっている。リンクエラーが発生すると、TCPではduplicate acks、もしくはRTOによって輻輳ウィンドウサイズを低下させ、パケットの再送を行うことになる。しかし、これでは輻輳が起こっていない場合でも輻輳ウィンドウサイズの不要な低下が発生し、スループットの低下を引き起こす可能性がある。特に、コネクション数が多く、他コネクションに帯域を取られる可能性が高いような状況では、輻輳ウィンドウサイズの上昇が阻害されスループットの低下はより顕著なものになると考えられる。

そこで、提案手法では、輻輳が起こっていない場面での輻輳ウィンドウサイズの大幅な低下を避けることを考える。

3.1.2　TCPのRTOによるリソースの浪費

まず、既存手法の場合について見てみると、正常にパケットが送られている状態でノードの動きによりリンクが途切れてしまった場合には、MAC層で設定されたリトライ回数を超えてしまいパケットロスが発生することとなる。その後、リンクが切れていることによりduplicate acksが送られてはこないため、RTOによる再送が発生するまでスループットがゼロ、つまり無線リソースを全く使えていない状態となる。もし、その途中でノードの移動により新しくルートが確立できる状態になっていたとすると、無駄になっている期間があるということになる。

そこで、提案手法では、RTOより早いタイミングでルートの再セットアップ、パケット再送を行うことにより、この無駄になっている期間を減らし全体のスループットを上げることを試みる。

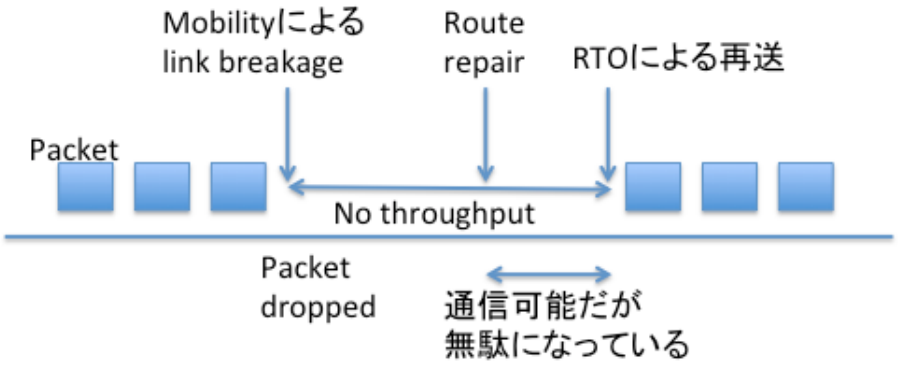

図 3.1. 既存手法

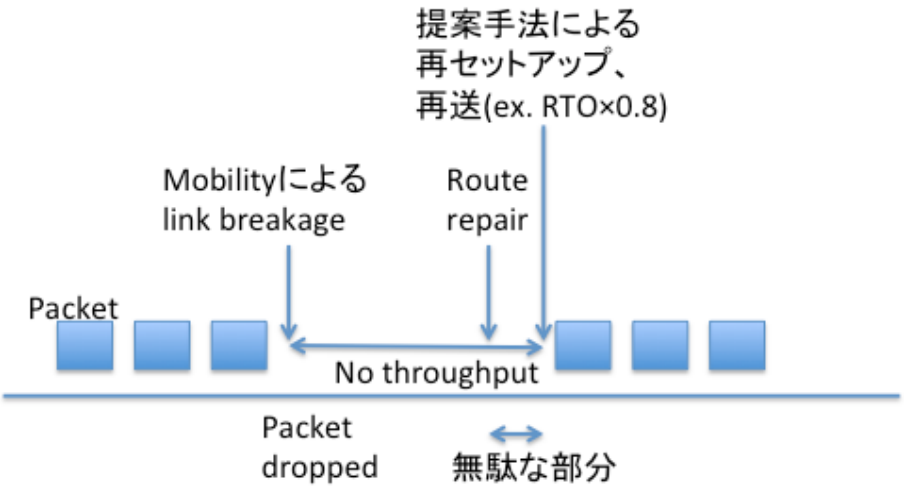

図 3.2. 提案手法

3.2 早期のルート再セットアップ、パケット再送

TCP コネクションのエンドノード間のルート上のあるノードが移動したことによりリンクが切断された場合には、MAC 層でリンク切れを検知するが、トランスポート層ではパケットロスとみなされる。既存手法では、図 3.1 の通り、その後タイムアウト（RTO）による再送が発生するまで通信が行われない。そこで提案手法では、図 3.2 のように、RTO より早いタイミングでルートを再セットアップしパケット再送を行うことにより、無駄な時間を短くすることでグッドプットを向上させる。更に、前節で述べたように、輻輳が生じていないのにタイムアウトになると、輻輳ウィンドウの不要な低下を招きグッドプットが低下する。提案手法では、このような輻輳ウィンドウの不要な低下も抑える。

送信者がパケットを送ってからある一定時間経過した後に ACK が返ってきていない場合には、そのルートが途切れている可能性は高いと考えられる。そこでそれ以上の時間を待つことは無駄な時間となってしまい、スループットの低下を招く。その上、それによって RTO や duplicate ack が生じ、輻輳ウィンドウサイズの低下を引き起こす可能性も高くなってしまう。

そこで提案手法では、ルートが切れている可能性が高いと判断したら、ルートの再セットアップを行うタイミングを早め即座に再送を行うことで、前述の問題の影響を和らげることを考える。この場合の再送では、従来の TCP による再送と違い、輻輳ウィンドウサイズの低下を行わない。リンクが途切れやすいアドホックネットワーク上では、従来の TCP では輻輳ウィンドウサイズを低下させるまでに通信を行う機会が少なすぎると考える。つまり、従来の TCP では使えるルートが残っており通信の回復を望めるような状態であっても輻輳ウィンドウサイズを低下させてしまう場合が多い考えられる。そこで提案手法では、輻輳ウィンドウサイズの低下が起こってしまう前に、回復させられる通信はなるべく回復させることを目指す。マルチパスルーティングを用いた手法では、バックアップのルートを保持していたとしてもそのルートはセットアップ時に存在していたルートに過ぎない。ノードの移動が激しいアドホックネットワークでは、ルートのセットアップからしばらく時間が経つと、そのバックアップのルートが物理的に存在しているとは限らない上、そのルートがその時点で効率の良いルートとは限らない。そこで提案手法では、ルートが切れている可能性が高いと判断できるタイミングでルートの再セットアップを行う。提案手法でルートの再セットアップ、再送を行うトリガーとなる条件を以下に挙げ、二つの方式を提案する。

3.2.1 提案手法 1

$RTO \times \alpha (0 < \alpha < 1)$ の期間の間、ソースノードが正常な ACK を受け取らなかった場合、ルートを再セットアップし、受信が確認できていないデータパケットから再送を始める。この際に、RTO とは違い輻輳ウィンドウは減少させない。この再送を行っても RTO までに正常な ACK を受けとれなかった場合には、通常と同じ処理を行う。

RTO は RTT に大してある程度余裕を持って設定されているが、実際には正常な ACK の到達する度に再送タイマーがリセットされるため、正常な ACK の到達間隔に対する RTO の値の大きさがより重要である。多くの場合、正常な ACK の到達間隔に対して RTO は大きめの値になっているので、α の値を小さくしすぎなければ正常な通信が行われている状態に提案手法による再セットアップ、再送が誤作動する危険は高くない。RTT が大きくなると正常な ACK の到達間隔も大きくなる傾向がある。つまり、RTO と正常な ACK の到達間

隔にも関係があると考えられるので、RTO の定数倍に再セットアップ、再送を行うことで性能改善が見込める。

3.2.2 提案手法 2

提案手法 1 で、一定期間正常な ACK を受け取らなかった場合の再セットアップ開始タイミングを $RTO \times \alpha, RTO \times \beta (0 < \beta < \alpha < 1)$ の二つに拡張する。そして、$RTO \times \beta$ のタイマーの時間となった場合には、$RTO \times \alpha > t$ が成立すれば再セットアップと再送を行い、$RTO \times \alpha$ のタイマーの時間となった場合には無条件に再セットアップと再送を行う。ここで、t は正常な ACK の到着間隔 (ackgap) を用い、以下のように計算する。

t の計算方法

$$t = K \times (sgap + 4 \times gapvar) \ (K : \text{予め定めた係数}) \tag{3.1}$$

$$sgap \leftarrow 0.875 \times sgap + 0.125 \times ackgap \tag{3.2}$$

$$gapvar \leftarrow 0.75 \times gapvar + 0.25 \times |sgap - ackgap| \tag{3.3}$$

t の計算には、RTT の観測値を用いた RTO の計算の考え方を用いている。よって t の値は、正常な ACK の到達間隔の平均値を重み付けして計算したものと対応しており、更に平均偏差を用いているため、到達間隔のぶれが大きいほど大きい値となる。$RTO \times \alpha > t$ の条件は、新しい正常な ACK の到達間隔の観測値がこれまでの到達間隔の傾向から大きく外れているかどうかを調べているのである。$RTO \times \alpha > t$ を満たす時、つまり正常な ACK の到達間隔がこれまでの傾向より大きくずれた時には、ルートの状況が変わり切断している可能性が高いと考えられるため、提案手法により再セットアップ、再送を行っても問題がない。提案手法 1 では通常の通信状態にも影響を与える可能性を考慮すると小さすぎる α の値を用いることはできず性能改善に限界があったが、提案手法 2 を用いることでルートが途切れている可能性が高い場合のみ更に早期の再セットアップ、再送が可能となるため、性能をより向上させられると考えられる。

3.2.3 提案手法のアルゴリズム

また、提案手法の再送の条件では、少なくとも RTO となってしまう前にルートの再セットアップ後のルートを往復できるだけの余裕が必要となる。また、再送までの時間を RTT

より小さくしてしまうと、まだそのルートで正常に送られているパケットが存在するにも関わらずルートの再セットアップを始めてしまう可能性が高くなってしまう。これらを踏まえると、再送までの時間は以下の式を満たす必要がある。

$$RTT\ in\ the\ previous\ route\ <\ \text{再送までの時間}\ <\ RTO - (route\ setup\ time + RTT\ in\ the\ new\ route) \tag{3.4}$$

提案手法では、TCP には変更を加えておらず、実装上の変更を送信元のノードのみに行うだけで済むため、実装が容易であり従来の TCP との互換性があるという利点がある。問題点としては、従来の手法では RTO が起こらないような通信であった場合でも、提案手法によりルートの再セットアップを行うことにより逆にスループットの低下を招いてしまう可能性があるということである。この問題を防ぐためにも、適切なパラメータの選別が必要となる。

提案手法について、具体的なアルゴリズムは以下のようなものである。

提案手法 1

1. 正常な ACK が届く
2. 　timer reset
3.
4. 条件: $RTO \times \alpha$ timer expire
5. 　ルートの再セットアップ
6. 　　セットアップ完了したら
7. 　　TCP で送受信が確認できていないシーケンス番号の
   　　若いパケットから再送

提案手法 2

1. 正常な ACK が届く
2. timer reset
3. $sgap \leftarrow 0.875 \times sgap + 0.125 \times ackgap$
4. $gapvar \leftarrow 0.75 \times gapvar + 0.25 \times |sgap - ackgap|$
5.
6. 条件 1: $RTO \times \beta$ timer expire かつ $RTO \times \beta > t$
7. $t = K \times (sgap + 4 \times gapvar)$ (K:予め定めた係数)
8. ルートの再セットアップ
9. セットアップ完了したら
10. TCP で送受信が確認できていないシーケンス番号の若いパケットから再送
11.
12. 条件 2: $RTO \times \alpha$
13. ルートの再セットアップ
14. セットアップ完了したら
15. TCP で送受信が確認できていないシーケンス番号の若いパケットから再送

3.3 提案手法の悪影響

提案手法では早期の再セットアップ、再送を行うため、エンドノード間のルートが存在していないタイミングで再セットアップ、再送が行われた場合には他のフローに影響を与えると考えられる。例えば、図 3.3 のようなトポロジーにおいて、ノード 2 がノード 1 の電波到達範囲外へと出て行った場合を考える。その場合、TCP flow1 では通信が途切れてしまい、いくらか時間が経つとノード 0 は提案手法による再セットアップ、再送を始めることとなる。まずは RREQ メッセージを送信してルートの確立を試みるが、ルートは途切れているため当然 RREP メッセージは返ってこない。よって、ルートが確立せずにデータパケットの再送は行われてないため、他フローに与える影響は RREQ メッセージのブロードキャストによるもののみとなる。図で表示されているノードのうちでは、ノード 0, ノード 1, ノード 5, ノード 6 による 4 回のブロードキャストが行われることになるが、TCP フローに対する性能低下としては軽微なものである。

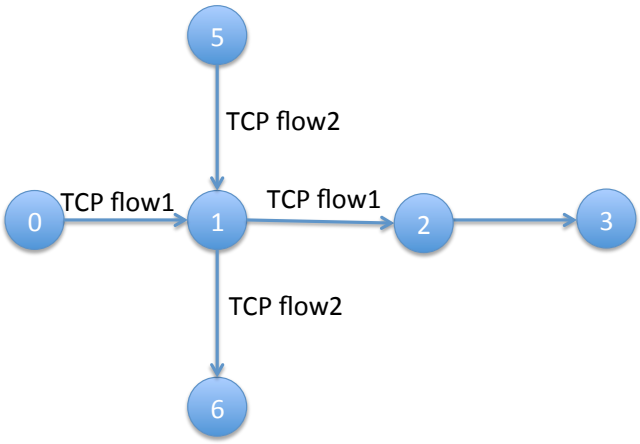

図 3.3. 提案手法の悪影響: トポロジー

第4章　実験と評価

4.1　実験環境

実験環境を表 4.1 に示す。

また、以下のグラフ上でエラーバーは信頼区間 95%を示す。「パケット数の差」は既存手法「TCP Newreno」で送受信できたパケット数よりも、提案手法で多く送受信することができたパケットの数である。

トポロジーとしては主に直線上にノードが並んだトポロジーを用いる。TCP フローが一つの場合には直線上のトポロジーについての結果を示せば十分だと考える。フローが複数になった場合は別のトポロジーも考える必要がある。また、ルートリカバリー（エンドノード間でルーティングプロトコルによりルートがセットアップされているかに関わらず物理的にルートが存在している状態）のタイミングとしては、ノードの移動によるリンク切断と同時にルートリカバリーが完了している場合と、長時間ルートリカバリーが完了しない場合について考えれば十分である。前者では提案手法による性能向上の程度を計測することができ、後者では提案手法による性能低下の程度を計測することができる。提案手法による性能低下については 3 章で述べた。ルートリカバリーがこれらとは違うタイミングで完了した場合には、既存手法、提案手法、共にその時点での RTO の値に応じて結果が大きく変わると考えられるが、ルートリカバリーのタイミングと RTO の値とは基本的には独立している。アプリケーションの傾向やノードの動く環境状況を用いてルートリカバリーのタイミングに合わせたパラメータの最適化により性能を向上させる手法は考えられるが、本研究では考えるところではない。

4.2　提案手法の有効性に関する予備検討

提案手法の有効性を示すため、ns-2[14] を用いたシミュレーションを行った。まず直線上の 3 個のノードから構成されるトポロジー (図 4.1) で、実験開始 10s 後に 2hop の TCP 通信を開始する。次に 50s 後に他のノードの電波到達範囲から外れるように中間ノードを移動し、同時に新たなノードを中間ノードの元の位置へ移動する。パラメータは、提案手法 1 では $\alpha = 0.5$、提案手法 2 では $\alpha = 0.8$, $\beta = 0.1$, $K = 10$ とした。結果を図 4.2, 4.3 に示す。提案手法 1,2 ともに RTO を避け、輻輳ウィンドウの最小値への低下を回避できており、リンクが途切れた際のグッドプットの低下を抑えている。

具体的には、回線速度が 54Mbps、パケット長が 1500 バイトの場合、既存手法と比較して、提案手法 1 では 618 パケット、提案手法 2 では 1200 パケット多く受信できている。このパケット数は、それぞれ提案手法 1 では 1s、提案手法 2 では 2s 分の通信量に対応する。

表 4.1. 実験環境

Parameter	Value
Simulator	ns-2 ver2.35
MAC Protocol	IEEE 802.11g
Number of Channel	1
Transmission Range	250m
Propagation Model	TwoRayGround
Interface Queue Type	DropTail/PriQueue
Queue Size	10000
RTS/CTS	あり
Routing Protocol	AODV
Transport Protocol	TCP NewReno
Number of Connections	1
Packet size	1460bytes
Cwnd upper limit	5MSS〜
Initial cwnd	1MSS
Initial ssthreshold	65535MSS
minimum RTO	0.2s
maximum RTO	60s
Generate Rate	FTP
Simulation area	5000m × 5000m
Speed	10m/s

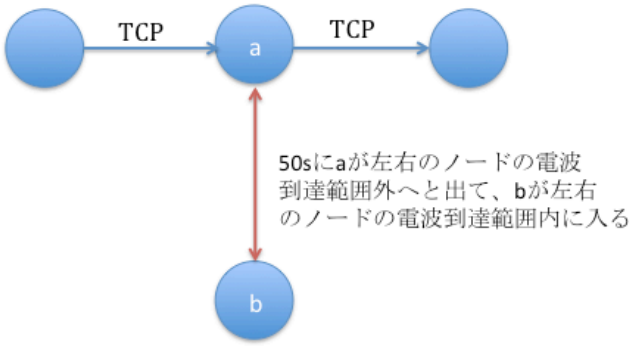

図 4.1. トポロジー 1

これはシミュレーション時間には依存しない値であり、既存手法では、定常状態から一度のリンク切れが発生するごとにこれだけのグッドプットの損失を重ねていくことになる。

輻輳ウィンドウサイズの変化を比較すると、既存手法では RTO が発生したことで輻輳ウィンドウサイズが 1MSS まで低下してしまっているが、提案手法では一度 duplicate acks により輻輳ウィンドウサイズが半分に減少してしまっているものの RTO の発生は避けることができている。これによりこの実験においては t の制約条件の上限を満たしているということがいえる。また、輻輳ウィンドウサイズが 1MSS に近い小さな値になると他コネクションとの兼ね合いにより輻輳ウィンドウサイズの上昇が遅れることも考えられるので、輻輳ウィンドウサイズは平均値よりも最小値の方に着目すべきであると考えられる。そのため、この結果は他コネクションの存在によってはより有用な結果になる可能性がある。

また、このような環境の実験では、輻輳ウィンドウサイズの最大値の設定に影響を受けることが分かった。アドホックネットワークでは、適切な輻輳ウィンドウサイズの値があり、それに応じて輻輳ウィンドウサイズの最大値を設定すべきだという報告がある [7]。

4.3 ホップ数の変化の影響を踏まえた実験環境パラメータの調整

MAC 層の RTS の設定された再送回数を越えてしまいリンクレイヤーディテクションが働き、パケットロスが発生する。これは NAV 状態のノードに対して RTS を複数回送ろうとしたノードで発生する。その結果、duplicate acks や RTO が発生することになってしま

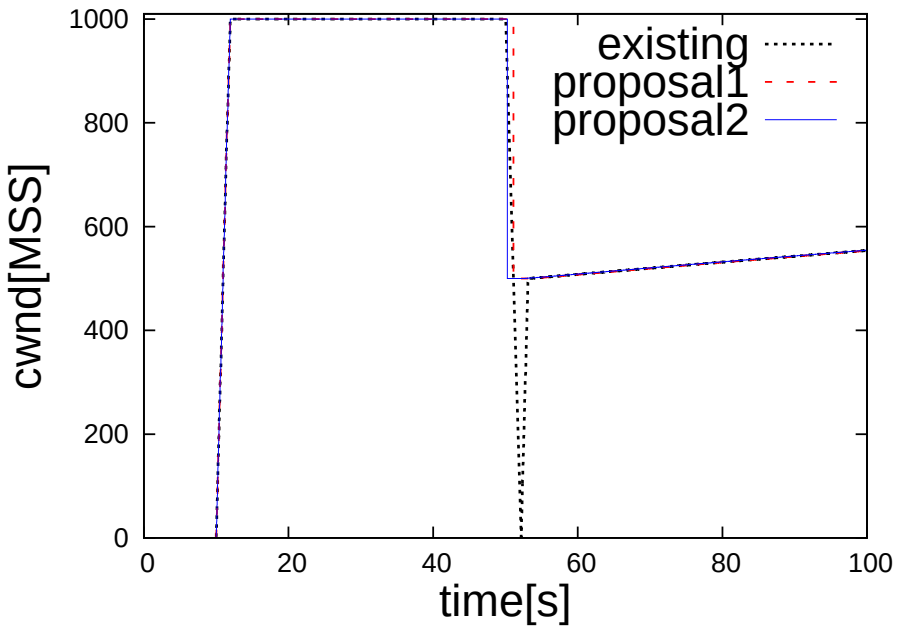

図 4.2. 輻輳ウィンドウサイズ

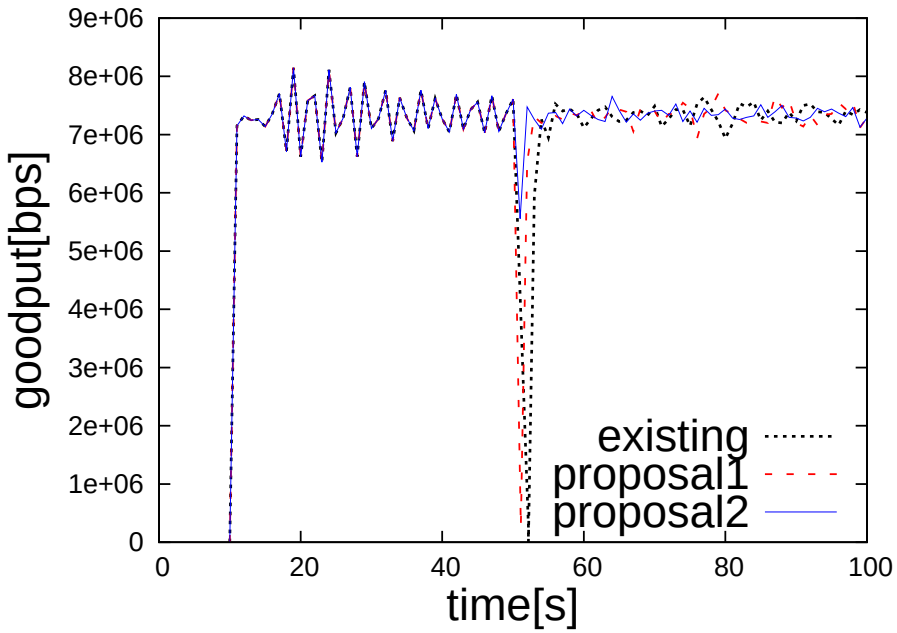

図 4.3. グッドプット

うため、輻輳ウィンドウサイズが激しく上下してしまう。これは輻輳ウィンドウサイズの最大値が大きい場合により顕著となる。図 2.4 より輻輳ウィンドウサイズの最大値はホップ数が 10 程度までなら 5 以下に設定すべきである。そこで、以下の実験では輻輳ウィンドウサイズの最大値を 5 として実験を行う。

輻輳ウィンドウサイズの最大値を必要以上に大きくした場合には既存手法では性能が低下するが、その原因としては、ノードのキューが空になる頻度の違いによると考えられる。ノードのキューが空になる頻度が高すぎても無線リソースを有効に使うことができずスループットが低下してしまうが、ノードのキューが常に埋まっているような状態になると隠れ端末問題やさらし端末問題による MAC 層の衝突が起こりやすくなる。提案手法の場合にも頻繁に発生する MAC 層の衝突をルートの途切れだと判断してしまい不必要な再セットアップ、再送が生じることにより性能が大幅に低下してしまった。輻輳ウィンドウサイズは大きすぎても小さすぎても性能が低下してしまうため、適切な値に設定する必要がある。

複数のチャネルを用いると衝突の可能性を減らすことができるが、通常、アドホックネットワークでは共有チャネルが前提となっていること、また現在のモバイル機器で複数チャネルに対応できる機器が少ないことから単一の共有チャネルを用いるという前提で実験を行う。また、MAC 層のリトライ回数に変更を加えることにより、ホップ数が大きくなった場合の MAC 層の衝突によるパケットロスを減らすこともできるが、リトライ回数を大きくすることで RTT や ACK の間隔のぶれが大きくなるため、提案手法には適さなかった。

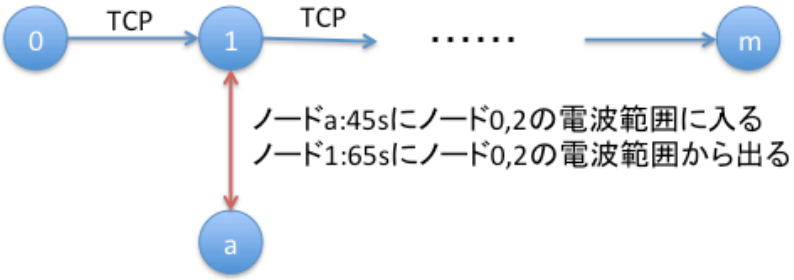

図 4.4. トポロジー 2

4.4 提案手法のパラメータ調整

直線上のトポロジーに対して提案手法のパラメータを変えて実験を行い、既存手法と比べて最終的に何パケット多く送受信が完了したか比較することによりパラメータ調整を行った。ここで、「送受信が完了した」とはソースノードで適切に ACK を受け取り終えている、ということである。200s のシミュレーションをシードを変えて数百回程度実験を行った。シードを変えることによりバックオフ初期値が変わりパケット通信の順序が変わることになる。その結果、タイムアウトが起きる可能性にも大きな影響が出る。

本節で用いるトポロジーを、図 4.4 に示す。このような直線上のトポロジーで 2 ホップ〜15 ホップのうち、いくつかのホップ数に対して実験を行った。それぞれのノード間の距離は 200m である。15 ホップまで実験を行うことで、メッシュネットワークでは 10 × 10 程度の環境まで適応できると思われる。また VANET でも 100m 程度に 1 ホップと考えると、1km 以上先までの情報伝達が可能となるので、車の衝突を回避するための事故情報の伝播を行うには十分である。

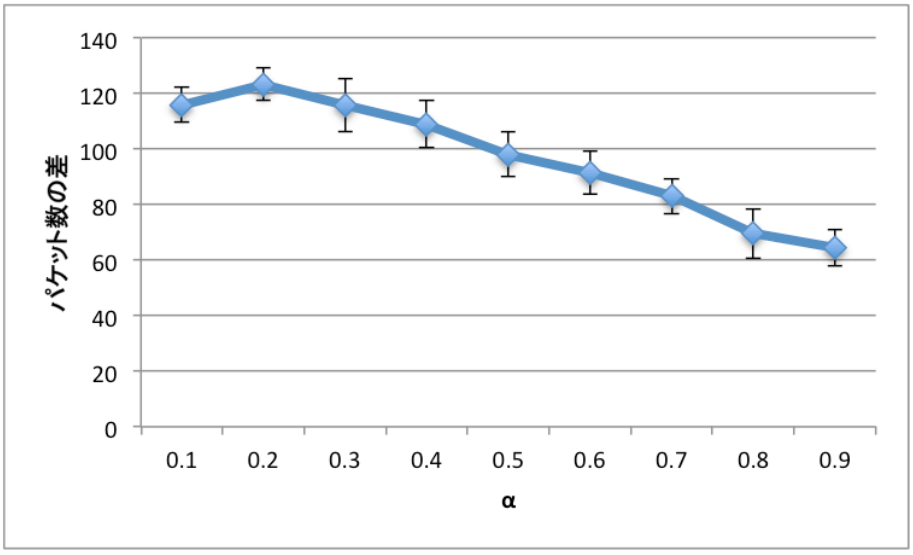

図 4.5. 提案手法 1 ホップ数 2 ($\alpha = [0.1, 0.9]$)

4.4.1 提案手法 1 の再送タイミングの調整

(a) ホップ数 2 の場合

既存手法に比べて提案手法がどれだけ多くパケットの送受信を行うことができたか、結果を図 4.5 に示す。これによると $\alpha = 0.2$ の時に最も多くのパケットを送受信できている。α が大きくなるにしたがって既存手法との差異が小さくなっているが、これは α が小さくなるほどより早期の再セットアップが行われることになるので、その分だけ多くのパケットができたということを示している。また、$\alpha = 1$ に近づいても既存手法とのパケット数の差異は 0 より大きいところに位置しているが、これは RTO の最小値の影響と考えられる。ホップ数が少ない場合には実装の変数内の RTO の値は最小値より小さい値を取っているために、RTO $\times$ 0.9 の場合であってもより性能の向上が見られる。

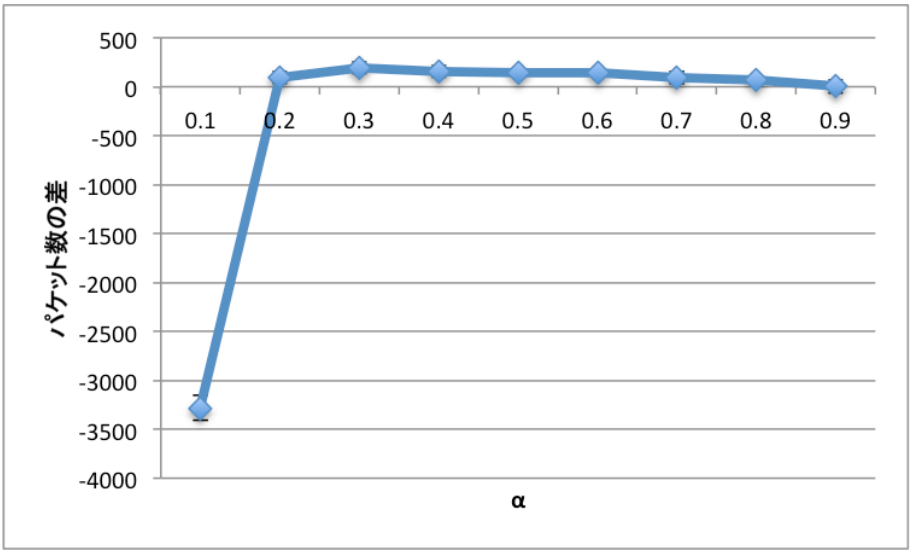

図 4.6. 提案手法 1 ホップ数 5 ($\alpha = [0.1, 0.9]$)

(b) ホップ数 5 の場合

図 4.6, 図 4.7 によると、$\alpha = 0.1$ の場合は既存手法よりも大きく性能が低下している。$\alpha = 0.2$ の場合には、既存手法よりは良い結果となっているが、$\alpha = 0.3$ の場合が極大となっておりそれに比べると性能が悪い。ホップ数 2 の場合と比べるとちょうど α の値が 0.1 ずれたものと近くなっている。つまり、ホップ数が大きくなるにしたがって必要とされる α の値は大きくなると考えられる。

この原因としてはまずホップ数が大きくなることにより RTT や ackgap が大きくなることによると考えられる。それにより RTO も大きくはなるのだが、計算式により RTT や ackgap の方がホップ数の影響を大きく受けるので、あまりに小さな α の値だと不必要な再セットアップ、再送を行ってしまい性能の低下を招いてしまう。もう一つの原因としては、ホップ数が増えることで隠れ端末問題やさらし端末問題により MAC 層の衝突が起こってしまうことにあると考えられる。MAC 層の衝突が起こることで実際にはルートの切断は起きていないのに提案手法による再セットアップ、再送が不必要に行われることが多くなる。更に、隠れ端末問題の影響でパケットスケジューリングにもぶれが生じやすくなり ackgap のぶれも大きくなると考えられる。これらのことより、ホップ数が 2 の場合に比べると α を大きめの値にする必要がある。

提案手法による再セットアップ、再送を行ったものの RTO になってしまった確率を表 4.2 に示す。これによると、α の値が大きくなるにつれて RTO になる確率は小さくなるのだが、提案手法による再セットアップ、再送が行われた場合の利益が α が小さくなるにつれて大きくなるので結果としては α を大きくしすぎると性能があまり向上しない。

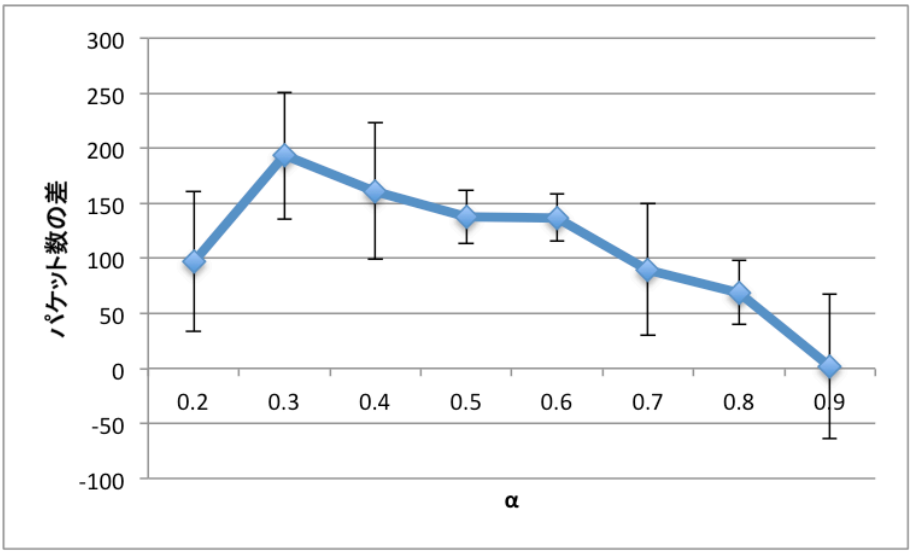

図 4.7. 提案手法 1 ホップ数 5 ($\alpha = [0.2, 0.9]$)

表 4.2. 提案手法 1　ホップ数 5　RTO の確率

α	RTO になる確率
0.1	37%
0.2	32.3%
0.3	33%
0.4	34%
0.5	31%
0.6	33%
0.7	27%
0.8	31.3%
0.9	24%

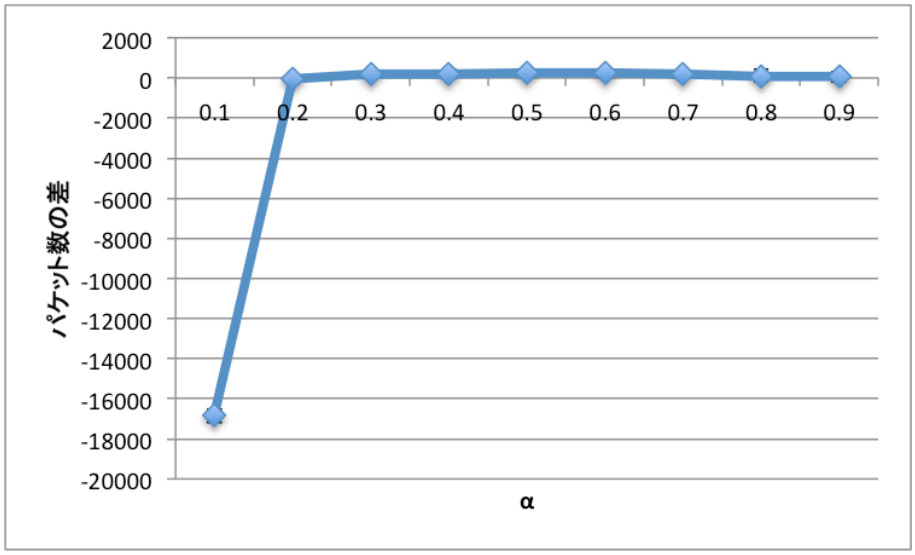

図 4.8. 提案手法 1 ホップ数 15 ($\alpha = [0.1, 0.9]$)

(c) ホップ数 15 の場合

図 4.8, 図 4.9 を見ると、ホップ数が 5 の場合と同様に $\alpha = 0.1$ の場合には既存手法より大きく性能が低下してしまっている。最も性能の良い α の値としては、少しの差ではあるが $\alpha = 0.6$ 程度と考えられる。

また、200s のシミュレーションでルートの切断が一度だけ起こったとすると、ホップ数が 2 の場合には提案手法 1 により既存手法で送受信できた全パケット数から 0.11%の増加が見られた。ホップ数が 5 の場合には提案手法 1 により既存手法で送受信できた全パケット数から 0.42%の増加、ホップ数が 15 の場合には提案手法 1 により既存手法で送受信できた全パケット数から 0.80%の増加が見られた。これはルートの切断が一度しか起こらなかったと仮定した場合の増加率であり、複数回のルート切断が見られればその度にこれらの量の送受信パケットの増加が見込める。この結果から見ると提案手法 1 ではホップ数が大きい場合の方がパケットの増加率は大きいと考えられる。

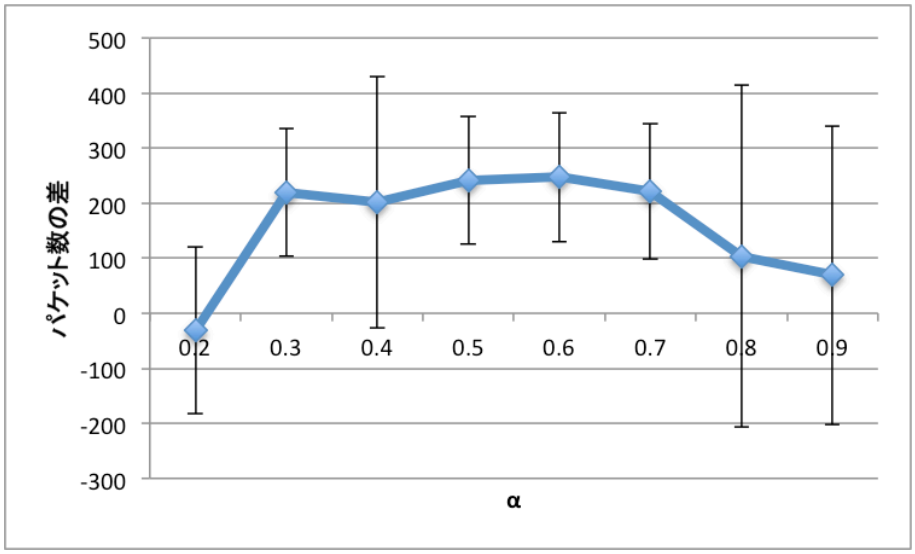

図 4.9. 提案手法 1 ホップ数 15 ($\alpha = [0.2, 0.9]$)

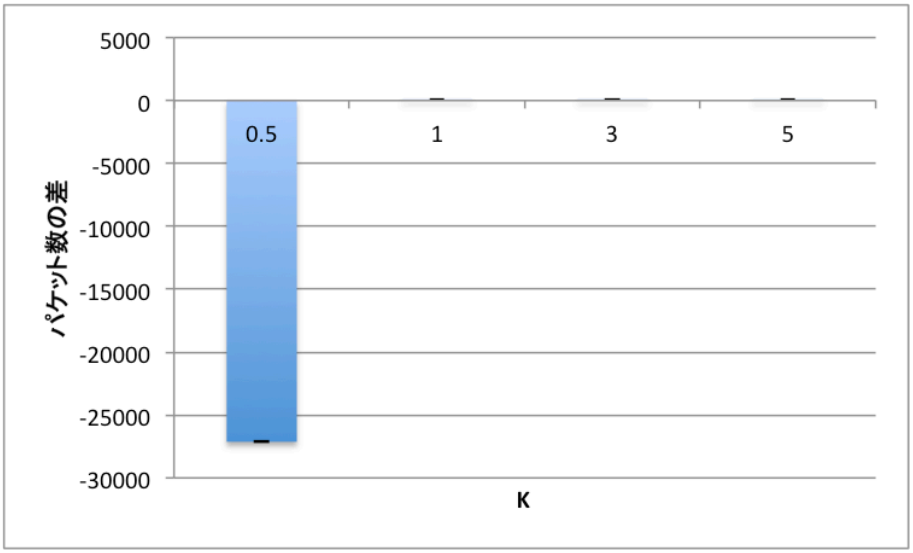

図 4.10. 提案手法 2 ホップ数 2 ($\alpha = 0.5, \beta = 0.01$)

4.4.2 提案手法 2 の再送タイミングの調整

提案手法 2 では、条件 1 の調整、つまり β や K の値の影響を調べながらパラメータの最適化を行う。条件 2 については、提案手法 1 の場合とほぼ同じ傾向が見られると考えられるので、今回の実験においては $\alpha = 0.5$ とした。

(a) ホップ数 2 の場合

$\alpha = 0.5$ のため、β の値の影響や K の値が小さすぎることにより条件 1 の再セットアップ、再送が悪影響を与えない限り、結果が既存手法より少なくとも 100 パケット程度良くなるはずである。

$\beta = 0.01$ とした場合には、提案手法の再セットアップ、再送のタイミングが非常に早くなりすぎており、通常の通信状態でも何度も条件 1 を満たし再セットアップ、再送を行っていた。ソースノードで新しいパケットが送信される前に、再送のタイミングとなることが多かったため、図 4.10 のようにグッドプットはあまり低下せず結果としては提案手法 1 の場合と同じ程度となっている。

図 4.13 を見ると、$K = 0.5, 1, 3$ の場合には提案手法 1 の場合より良い結果となっている。提案手法 1 において $\alpha = 0.2$ とした場合よりも良い結果となっていることより、適切に $RTO \times 0.1 (RTO \times \beta)$ のタイミングで再セットアップ、再送が行われたことが分かる。提案手法 1 では $\alpha = 0.1$ とした場合には結果が落ちてしまっていたが、提案手法 2 の方式によって必要な場合のみに $RTO \times 0.1$ のタイミングで再セットアップ、再送を行おうとする試みがうまくいったといえる。また、$K = 5$ の場合を見てみると、提案手法 1 で $\alpha = 0.5$ とした場合と同程度の結果である。これは K の値を大きくしすぎたことにより、条件 1 を満たすことがほとんどなくなり提案手法 1 の場合と同じ結果になったと考えられる。つまり、K の値をある程度大きめの値に設定しておくことにより、少なくとも提案手法 1 程度の性能向上を保証することができる。

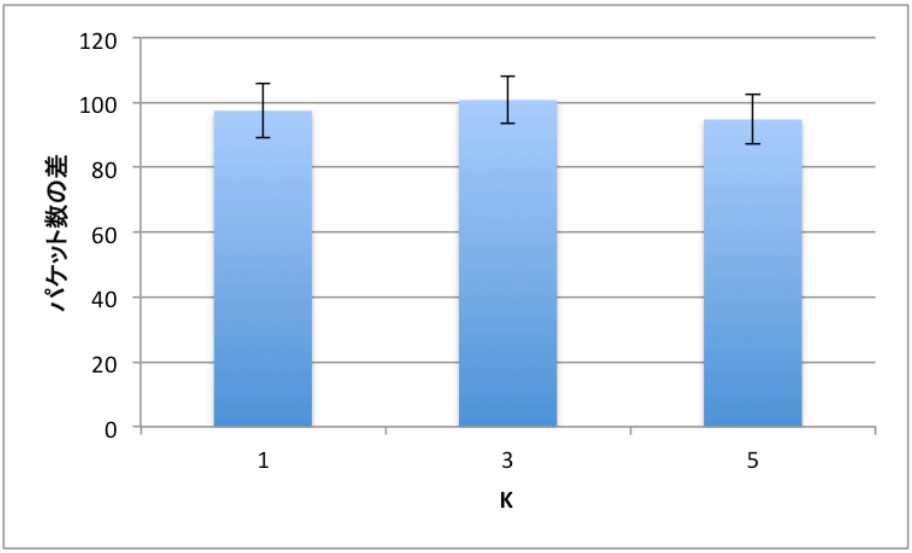

図 4.11. 提案手法 2 ホップ数 2 ($\alpha = 0.5, \beta = 0.01, K = 1, 3, 5$)

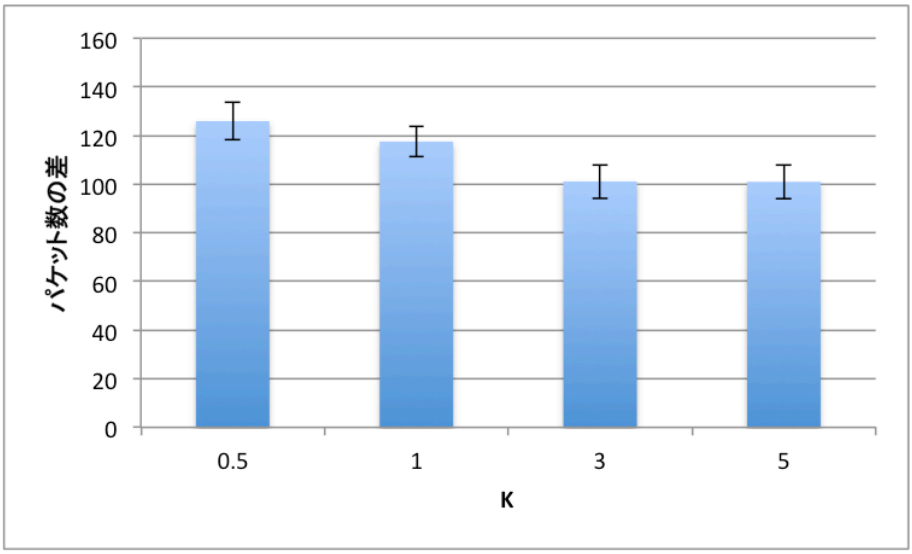

図 4.12. 提案手法 2 ホップ数 2 ($\alpha = 0.5, \beta = 0.05$)

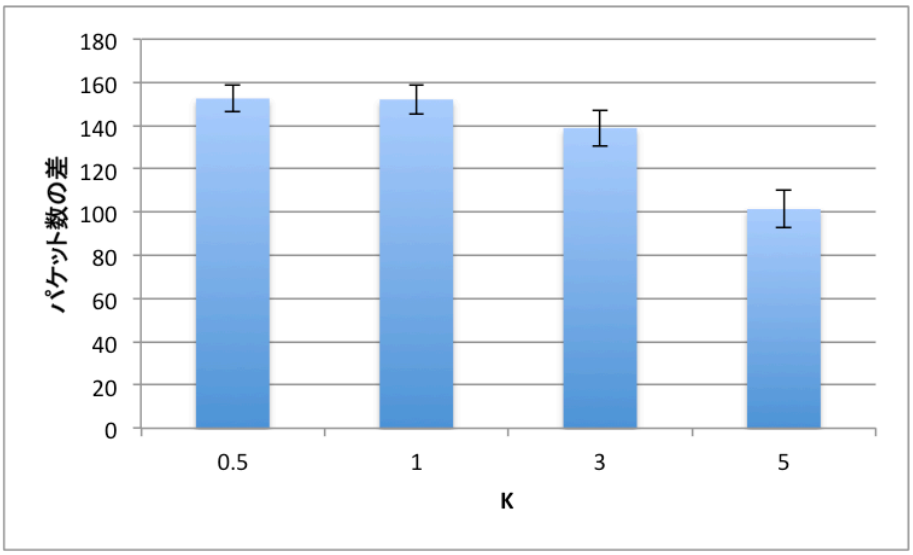

図 4.13. 提案手法 2 ホップ数 2 ($\alpha = 0.5, \beta = 0.1$)

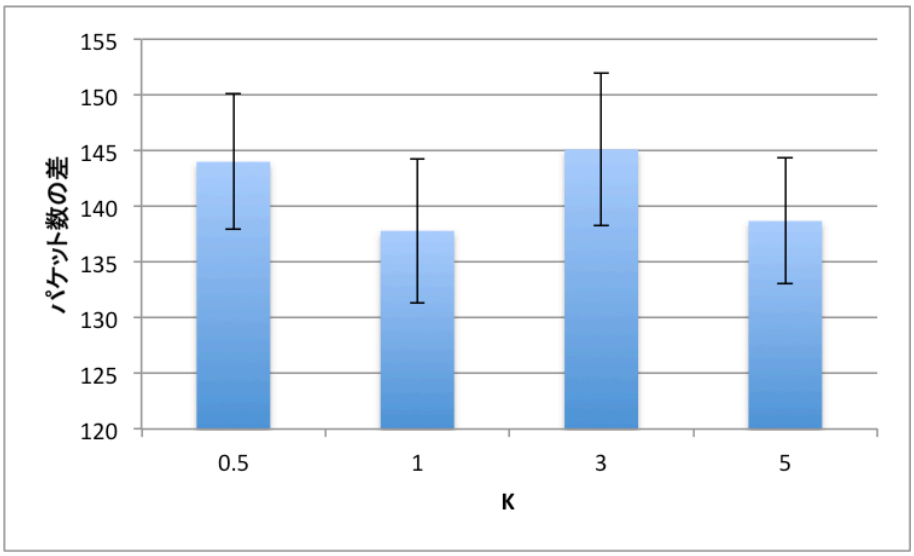

図 4.14. 提案手法 2 ホップ数 2 ($\alpha = 0.5, \beta = 0.2$)

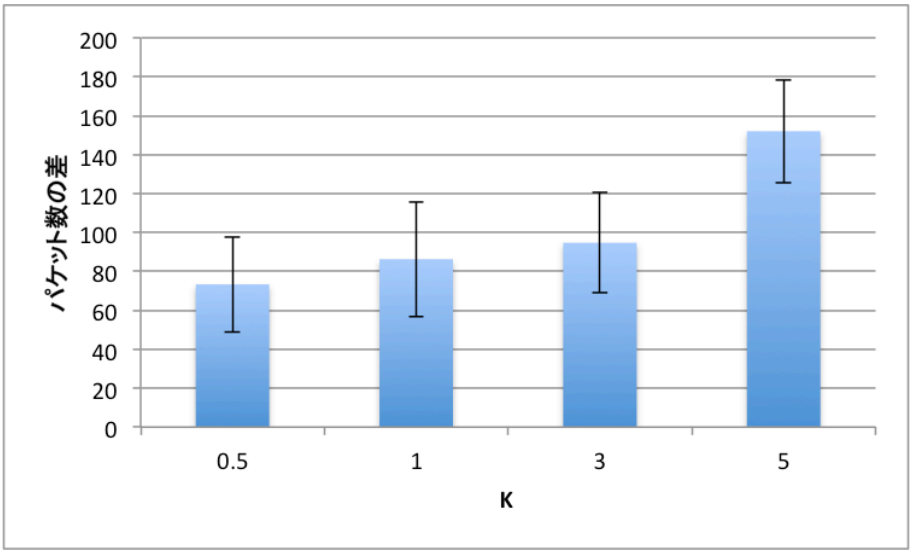

図 4.15. 提案手法 2 ホップ数 5 ($\alpha = 0.5, \beta = 0.01$)

(b) ホップ数 5 の場合

ホップ数が 5 の場合を見てみると、$K = 3, 5$ の場合に良い結果となっているものが多い。提案手法 1 の場合にも書いたようにホップ数が増えることにより RTT が大きくなり ackgap のぶれも大きくなる。ackgap のぶれが大きくなると条件 1 を満たしやすくなってしまうため、K の値が小さい場合には予期しない再セットアップ、再送が行われる可能性が高くなる。そのため、ホップ数が大きい場合には、K や β を大きくする必要がある。ホップ数 5 では、$\beta = 0.1$, $K = 3$ の時に最も良い結果となっている。

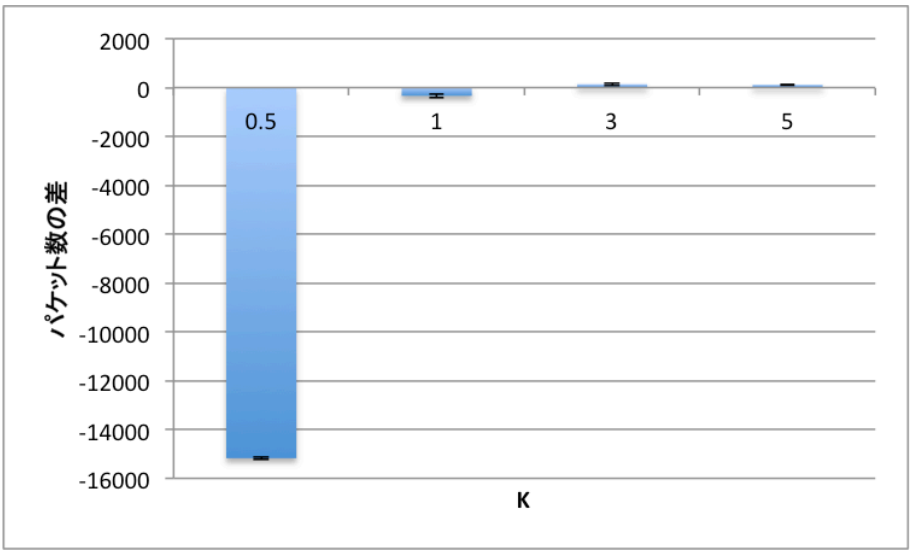

図 4.16. 提案手法 2 ホップ数 5 ($\alpha = 0.5, \beta = 0.05$)

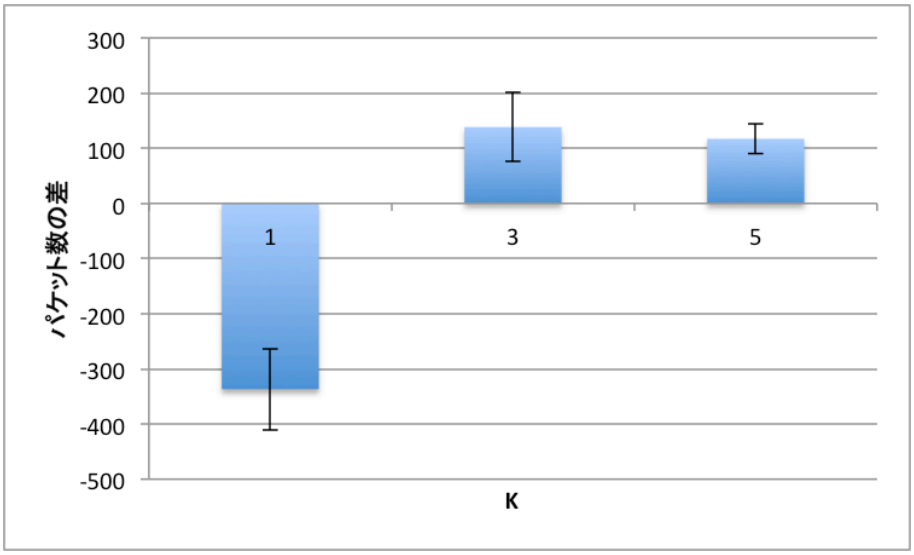

図 4.17. 提案手法 2 ホップ数 5 ($\alpha = 0.5, \beta = 0.05, K = 1, 3, 5$)

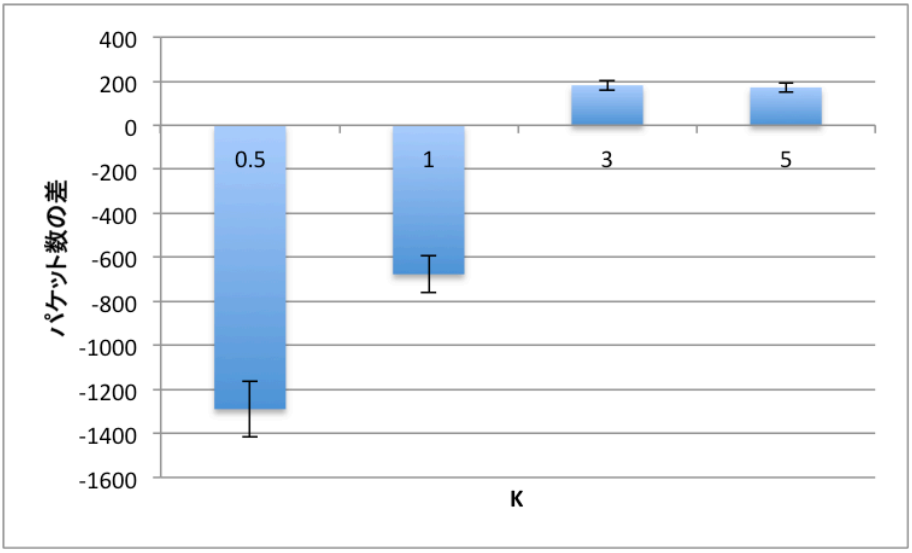

図 4.18. 提案手法 2 ホップ数 5 ($\alpha = 0.5, \beta = 0.1$)

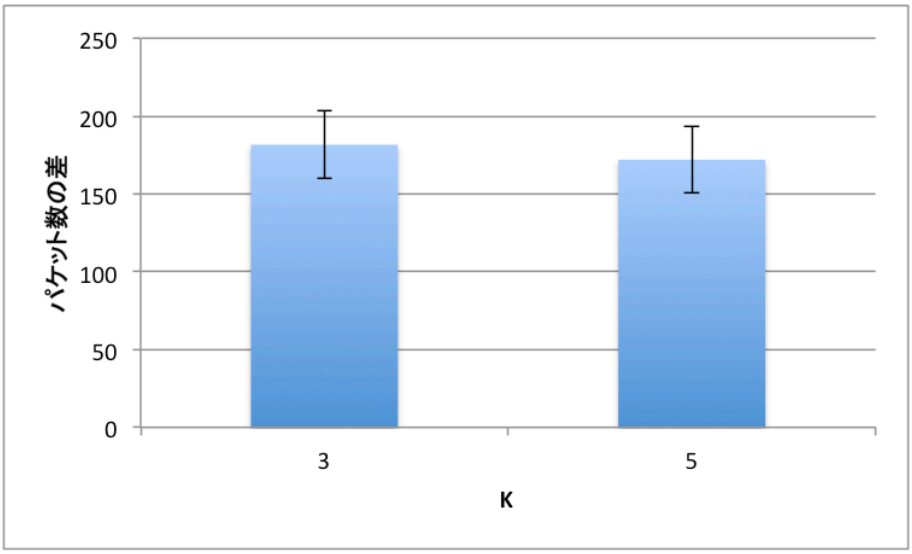

図 4.19. 提案手法 2 ホップ数 5 ($\alpha = 0.5, \beta = 0.1, K = 3, 5$)

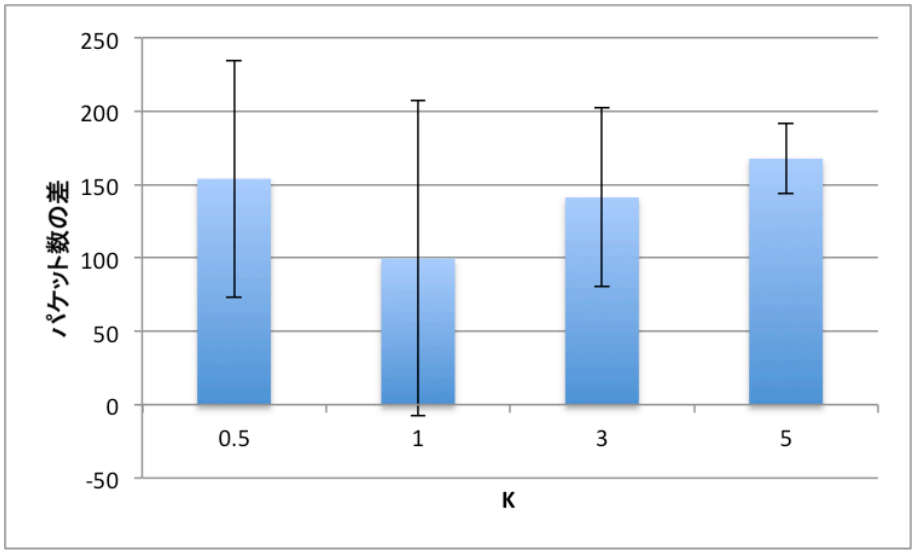

図 4.20. 提案手法 2 ホップ数 5 ($\alpha = 0.5, \beta = 0.2$)

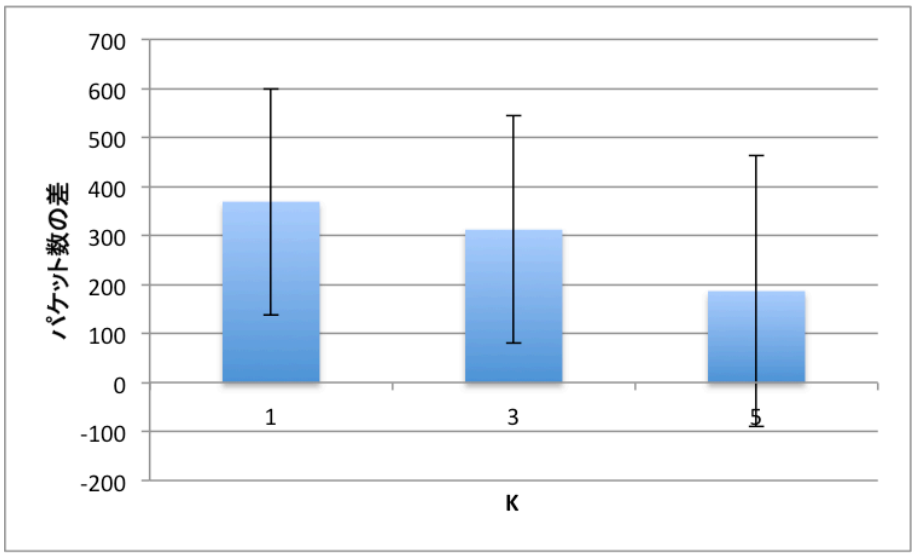

図 4.21. 提案手法 2 ホップ数 15 ($\alpha = 0.5, \beta = 0.01$)

(c) ホップ数 15 の場合

ホップ数が 15 の場合には、ホップ数が 5 の場合の結果を踏まえ $\beta = 0.01, 0.1$, $K = 1, 3, 5$ の場合に対して実験を行った。提案手法 1 の場合と同様にホップ数が大きくなるにしたがって、既存手法より多く送受信できたパケット数の、全体のパケット数に対する割合は大きくなっていることが分かる。ホップ数が大きくなるにしたがって K の値も大きくする必要があると考えられるが、ホップ数を 15 まで大きくしても $\beta = 0.1$ の場合には $K = 3$ もあれば十分であることが確認できた。

4.4.3 パラメータ調整のまとめ

この実験では、リンクが切断した際の復帰までの時間が結果を左右している。RTO × α 単一のタイミングのものでは、α を一定以上小さくしてしまうと通常の通信状態の場合にも再送が行われてしまい性能が低下してしまう。そのためルートが途切れたような特殊な状況を判別する必要があり、その時点までの ackgap を用いた値を用いて ackgap が大きく外れた値となった場合に再セットアップ、再送の条件を満たすような方式とした。これにより、その特殊な状況に対してだけ更に早いタイミングで復帰させることができている。ホップ数が大きくなるにつれ、ackgap を用いた条件についても誤作動することが多くなってしまうので、K の値を大きくする必要があることが分かった。提案手法 1 のパラメータについては、ホップ数が 5 程度の場合には $\alpha = 0.3$ の時、ホップ数が 15 の場合には $\alpha = 0.3$ から $\alpha = 0.6$ の時に良い結果となった。提案手法 2 のパラメータについては $\beta = 0.01, 0.05$ では小さく、$\beta = 0.1$ の時に最も良い結果となった。ホップ数が大きい場合でも問題が生じないために $K = 3$ 程度にすべきである。これらの実験より、ホップ数が最大 15 程度の環境の場合には、提案手法 1 では $\alpha = 0.3$, 提案手法 2 では $\alpha = 0.5, \beta = 0.1, K = 3$ のパラメータを用いた場合が最も汎用性が高いと考えられる。

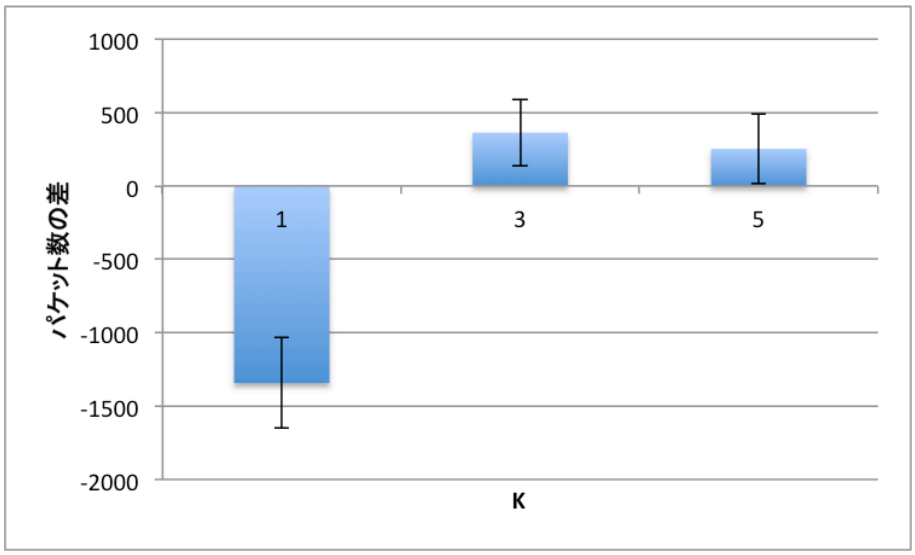

図 4.22. 提案手法 2 ホップ数 15 ($\alpha = 0.5, \beta = 0.1$)

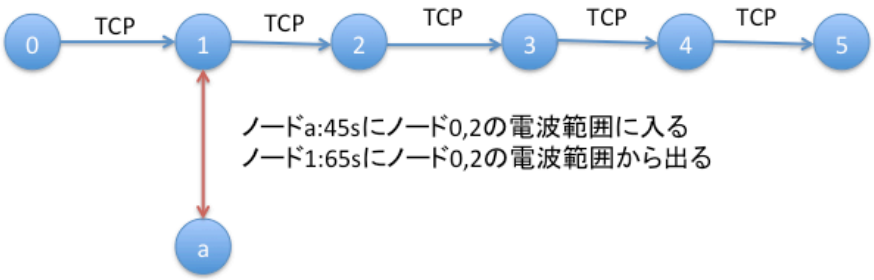

図 4.23. トポロジー　ノード 1 が動いた場合

4.5　提案手法の比較評価

4.5.1　ノードの移動に対する比較評価

ホップ数 5 の場合に対して、それぞれのノードが移動した場合の影響を観測する。以下のグラフで 5 つのうち左側 2 つのものは提案手法 1, 右側 3 つのものは提案手法 2 である。提案手法 2 では $\alpha = 0.5$, $\beta = 0.1$ としている。ノード 1 が動いた場合のトポロジー例は図 4.23 である。

移動ノードが 1,2,3,4 いずれの場合にも大きく変化は見られず、同じような傾向となった。この結果より、ノードの移動によりリンクが途切れる位置と提案手法の効果にはあまり関連がないと考えられる。

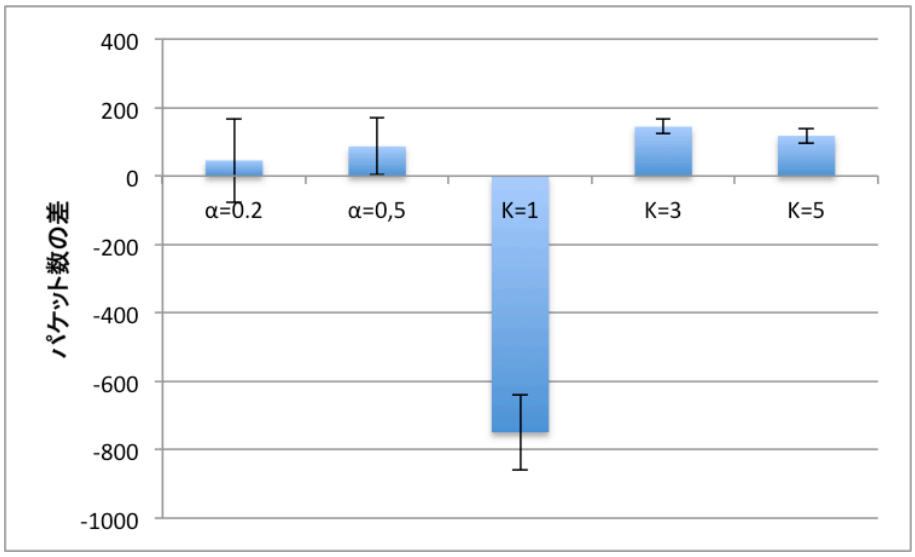

図 4.24. ノード 1 が動いた場合

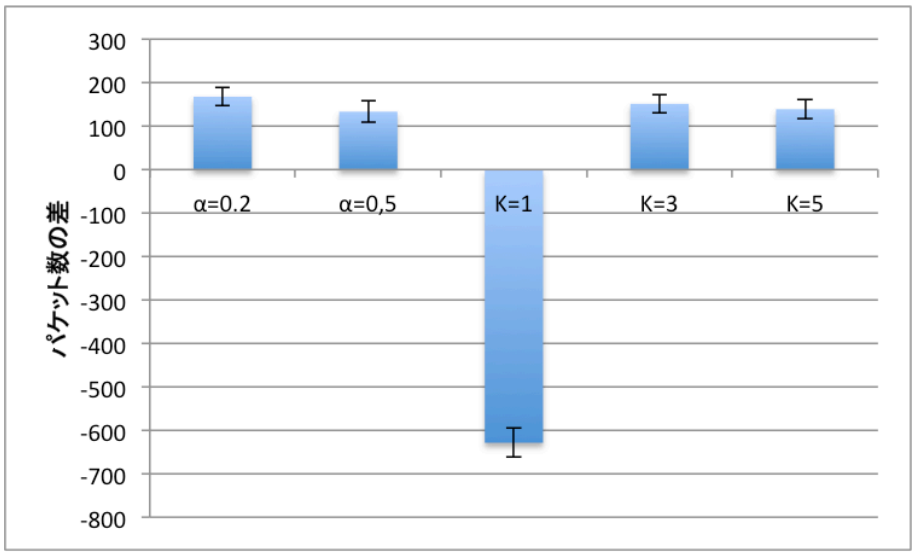

図 4.25. ノード 2 が動いた場合

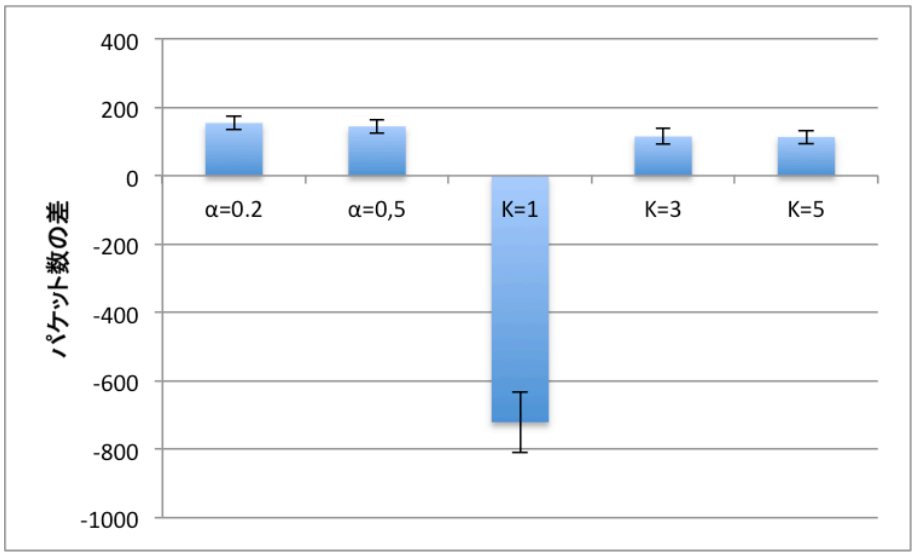

図 4.26. ノード 3 が動いた場合

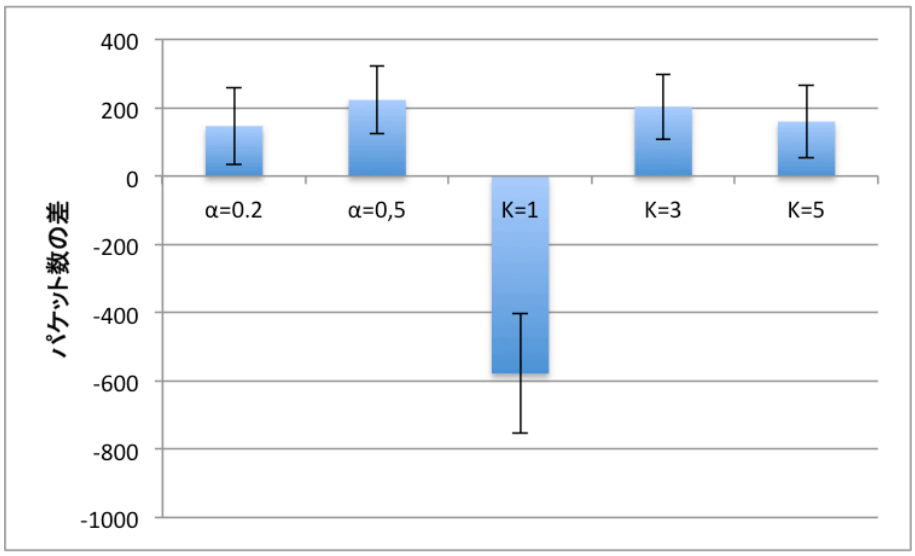

図 4.27. ノード 4 が動いた場合

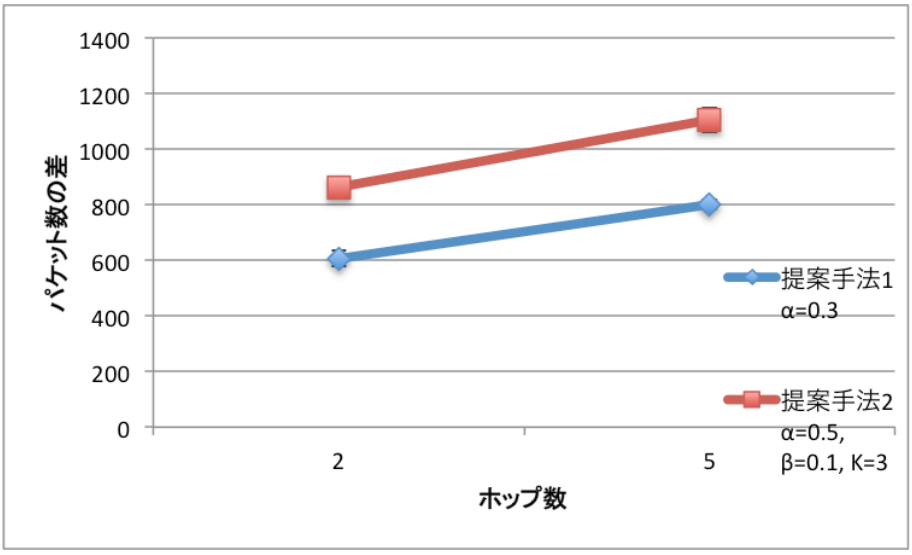

図 4.28. RTO の最小値が大きい場合の比較評価

4.5.2 RTO の最小値が大きい場合の比較評価

これまでは RTO の最小値を 0.2s と設定して実験を行った。その理由は現在の実装では RTO の最小値を 0.2s と設定しているものが多いからである。しかし、RFC2988[15] では RTO の最小値として 1s が推奨されていたことから、RTO の最小値を 1s としている実装も考えられる。そこで、RTO の最小値が異なる場合に提案手法の性能評価を行った。RTO の最小値を 1s とし、トポロジーとして図 4.4 を用いて、ホップ数が 2,5 の場合の実験を行った。結果を図 4.28 に示す。

RTO の最小値を 0.2s と設定した実験と比較すると、ホップ数が 2 の場合にはパケット数の差が約 5 倍となっている。ホップ数が 5 の場合には 5 倍とは少しずれているが誤差の範囲と考えられる。つまり、RTO の最小値が大きくなればなるほど、提案手法の有用性は相対的に向上する。この結果から、既存手法でグッドプットが低下したのは、タイムアウトによる再送が生じるまで不必要に待機していた時間と丁度対応することが分かる。

4.5.3 ホップ数に対する比較評価

パラメータ調整を行った提案手法について、ホップ数に変化に対する結果を図 4.29 に示す。トポロジーは 4.4 とした。

ホップ数が小さい場合には提案手法 1,2 共に同じ程度の結果となった。これは、提案手法 2 において条件 2 のみを満たす場合が多かった影響であると考えられる。つまり、ホップ数が小さい場合に提案手法 2 でより性能を上げるには動的なパラメータ制御などを用いて、ホップ数が小さい場合のみ K の値を小さくする方式が必要となる。ホップ数が 10 以上の場合には提案手法 2 の結果が勝っている。分散も大きいため比較はしにくいが、この結果の平均値ではホップ数が 10 以上の場合に提案手法 2 の方が提案手法 1 より 60〜110%程度性能が向上している。

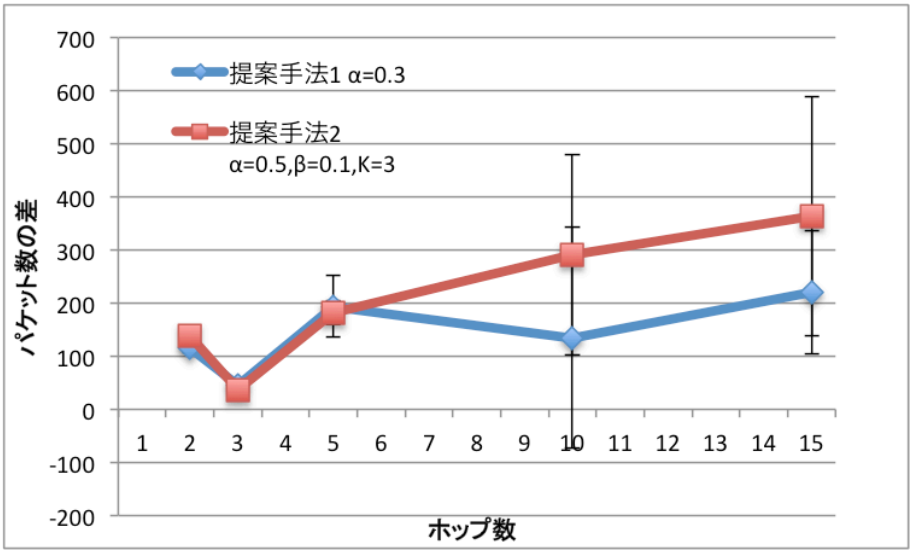

図 4.29. ホップ数に対する比較評価

また、ホップ数が 15 の場合についてあるシードに関するシミュレーションを行い、ノードの移動によりリンク切断が生じてからソースノードが再送を始めるまでの時間を計測した。その結果は、提案手法 1 では 60.1ms, 提案手法 2 では 96.6ms, 既存手法では 197ms だった。この結果は提案手法 1 では $RTO \quad \times \alpha(\alpha = 0.3)$ で早期の再送を行い、提案手法 2 では条件 1 は満たさなかったものの $RTO \times \alpha(\alpha = 0.5)$ で早期の再送を行い、既存手法では RTO となったと考えられる。提案手法 2 で $\alpha = 0.3$ としていた場合には提案手法 1 とほとんど同じ結果となることが期待される。この結果より、ホップ数が 15 の場合にも早期の再送は十分に成功していると考えられる。ホップ数が 15 の場合のグッドプットは約 2.0Mbps なので、100ms 早く再送を行うことができれば約 167 パケット多く送信できることになる。提案手法 $1(\alpha = 0.3)$ の場合には約 220 パケット、提案手法 $2(\alpha = 0.5, \beta = 0.1)$ で条件 1 を満たした場合には約 300 パケット、条件 2 を満たした場合には約 170 パケット多く送信できるということである。つまり、理想的に最速のタイミングでルートの再セットアップをした場合には約 340 パケット多く送信できると考えられる。つまり、提案手法 2 ではルートが一度切断することに、その影響でパケットがロスする量を約 87%近く削減することができる。提案手法 1 では約 56%の削減となる。

第5章　結論

5.1　まとめ

アドホックネットワーク上の TCP 通信は、ノードの動きによるリンクの途切れと、マルチホップ無線通信によるパケット衝突により大きく性能が低下してしまう。これは通信が可能であっても RTO まで再送が行われず無駄な待ち時間を費やしてしまうことと、RTO により不要な輻輳ウィンドウサイズの低下の影響によるものである。そこで本研究では、RTO を引き起こす可能性の高い状況で早期のルート再セットアップ、パケット再送を行うことにより、その後の RTO をなるべく防ぐための手法を提案した。提案手法では、RTO と RTT の関連性を考え RTO の定数倍のタイミングで再セットアップ、再送を行う方式と、正常な ACK の受信の間隔で RTO のタイマーがリセットされることを利用して ACK の受信間隔の変化に程度を用いたタイミングで再セットアップ、再送を行う方式を検討した。

本論文では、ns-2 のソースコードを元に修正して提案手法を実装し、シミュレーションにより提案手法の有効性を示した。実際に輻輳ウィンドウサイズの不要な低下を抑え、グッドプットが向上していることを確認した。また、ホップ数が大きく隠れ端末問題やさらし端末問題による MAC 層の衝突が頻繁に生じる場合には提案手法はうまく働かなかった。そこで、輻輳ウィンドウサイズの最大値を設定し、直線上のトポロジーに対する実験により、提案手法のパラメータの影響を検証し、パラメータの調整を行った。パラメータ調整を行った提案手法について、実験により以下のことが分かった。

- 移動ノードによるリンク切断が生じる位置は提案手法の有効性にあまり影響及ぼさない
- RTO の最小値が大きい実装環境においては、RTO の最小値の増加に対応して提案手法の有効性が高まる
- ホップ数が大きい環境においては、提案手法 2 の方が提案手法 1 より 60〜110%程度性能が向上した
- ホップ数が 15 の場合のあるシミュレーションでは提案手法 2 によりルート切断の影響でパケットがロスする量を最大約 87%, 提案手法 1 では約 56%削減できた

5.2　今後の課題

1. 複数フローへの適用

TCP フローが複数の場合や UDP 通信が存在する場合に適切なパラメータを調べる必

要がある。特に UDP 通信が存在する場合には提案手法の通信による UDP 通信への悪影響を考慮してパラメータを設定しなければならない。

2. 動的なパラメータ制御

今回、ホップ数の変化に対して適切なパラメータに違いが見られる部分があった。ホップ数やその他のパラメータを用いることによって動的にパラメータの制御を行うことにより性能を向上させられると考えられる。

3. アプリケーションや通信環境への最適化

本研究では、アドホックネットワーク上で TCP を用いる場合全般に対して用いることができる手法を考えたが、特定のアプリケーションや通信環境において最適なパラメータを求め、適応させることにより更なる性能の向上が期待できる。

4.MAC 層の衝突への対処

本研究では、TCP と AODV を考慮した提案手法を用いたが、MAC 層の衝突の影響が大きいことは分かっている。MAC 層の衝突への直接的な対処を方式に組み込むことも考えられる。

謝辞

本研究を進めるにあたって多くの方々にお世話になりました。

指導教員である若原恭教授には毎週打ち合わせの機会をいただき、研究や発表に関して数多くのご指導を賜りました。また、中山雅哉准教授、小川剛史准教授、関谷勇司准教授、宮本大輔助教、妙中雄三助教には、本研究に対して様々な視点からアドバイスをいただきました。

2 年間励ましあいながら苦楽を共にした同期の方々をはじめとして、研究室の皆様には大変お世話になりました。

本当に多くの方々の支えられこれまでの研究生活を送ることができました。この場をお借りして厚くお礼申し上げます。

最後に、これまでの研究生活を支えてくださった両親に心より感謝いたします。本当にありがとうございました。

参考文献

[1] Umut Akyol, Matthew Andrews, Piyush Gupta, John Hobby, Iraj Saniee, and Alexander Stolyar. Joint scheduling and congestion control in mobile ad-hoc networks. In *INFOCOM 2008. The 27th Conference on Computer Communications. IEEE*, pp. 619–627. IEEE, 2008.

[2] Beizhong Chen, Ivan Marsic, Huai-Rong Shao, and Ray Miller. Improved delayed ack for tcp over multi-hop wireless networks. In *Wireless Communications and Networking Conference, 2009. WCNC 2009. IEEE*, pp. 1–5. IEEE, 2009.

[3] Lijun Chen, Steven H Low, Mung Chiang, and John C Doyle. Cross-layer congestion control, routing and scheduling design in ad hoc wireless networks. 2006.

[4] Rung-Shiang Cheng and Hui-Tang Lin. A cross-layer design for tcp end-to-end performance improvement in multi-hop wireless networks. *Computer Communications*, Vol. 31, No. 14, pp. 3145–3152, 2008.

[5] Jakob Eriksson, Hari Balakrishnan, and Samuel Madden. Cabernet: vehicular content delivery using wifi. In *Proceedings of the 14th ACM international conference on Mobile computing and networking*, pp. 199–210. ACM, 2008.

[6] Sally Floyd, Tom Henderson, and Andrei Gurtov. The newreno modification to tcp ' s fast recovery algorithm. Technical report, RFC 2582, April, 1999.

[7] Zhenghua Fu, Petros Zerfos, Haiyun Luo, Songwu Lu, Lixia Zhang, and Mario Gerla. The impact of multihop wireless channel on tcp throughput and loss. In *INFOCOM 2003. Twenty-second annual joint conference of the IEEE Computer and Communications. IEEE Societies*, Vol. 3, pp. 1744–1753. IEEE, 2003.

[8] Arsahd Hussain, M Saad Akbar, and Mubashir A Cheema. A simple cross-layer approach to reduce duplicate acknowledgements for tcp over wlan. In *Networking and Communications Conference, 2008. INCC 2008. IEEE International*, pp. 63–66. IEEE, 2008.

[9] Van Jacobson. Congestion avoidance and control. In *ACM SIGCOMM Computer Communication Review*, Vol. 18, pp. 314–329. ACM, 1988.

[10] Van Jacobson, Robert Braden, and David Borman. Tcp extensions for high performance. 1992.

[11] Wonsoo Kim, Hyrum K Wright, and Scott M Nettles. Improving the performance of multi-hop wireless networks using frame aggregation and broadcast for tcp acks. In *Proceedings of the 2008 ACM CoNEXT Conference*, p. 27. ACM, 2008.

[12] Ilias Leontiadis, Paolo Costa, and Cecilia Mascolo. Extending access point connectivity through opportunistic routing in vehicular networks. In *INFOCOM, 2010 Proceedings IEEE*, pp. 1–5. IEEE, 2010.

[13] Mahesh K Marina and Samir R Das. Ad hoc on-demand multipath distance vector routing. *ACM SIGMOBILE Mobile Computing and Communications Review*, Vol. 6, No. 3, pp. 92–93, 2002.

[14] Steven McCanne, Sally Floyd, Kevin Fall, Kannan Varadhan, et al. Network simulator ns-2, 1997.

[15] Vern Paxson and Mark Allman. Computing tcp 's retransmission timer. Technical report, RFc 2988, November, 2000.

[16] Charles Perkins, E Belding-Royer, Samir Das, et al. Rfc 3561-ad hoc on-demand distance vector (aodv) routing. *Internet RFCs*, pp. 1–38, 2003.

[17] Biplab Sikdar, Shivkumar Kalyanaraman, and Kenneth S Vastola. Analytic models for the latency and steady-state throughput of tcp tahoe, reno, and sack. *Networking, IEEE/ACM Transactions on*, Vol. 11, No. 6, pp. 959–971, 2003.

[18] Prasanthi Sreekumari, Sang-Hwa Chung, and Won-Suk Kim. A timestamp based detection of fast retransmission loss for improving the performance of tcp newreno over wireless networks. In *Wireless and Mobile Computing, Networking and Communications (WiMob), 2011 IEEE 7th International Conference on*, pp. 60–67. IEEE, 2011.

[19] W Richard Stevens, 橘康雄, 井上尚司, 詳解 TCP, IP Vol. プロトコル, ピアソン・エデュケーション, 2000. 1.

[20] Karthikeyan Sundaresan, Vaidyanathan Anantharaman, Hung-Yun Hsieh, and AR Sivakumar. Atp: A reliable transport protocol for ad hoc networks. *Mobile Computing, IEEE Transactions on*, Vol. 4, No. 6, pp. 588–603, 2005.

[21] Xuyang Wang and Dmitri Perkins. Cross-layer hop-by-hop congestion control in mobile ad hoc networks. In *Wireless Communications and Networking Conference, 2008. WCNC 2008. IEEE*, pp. 2456–2461. IEEE, 2008.

[22] Ajit Warrier, Sankararaman Janakiraman, Sangtae Ha, and Injong Rhee. Diffq: Practical differential backlog congestion control for wireless networks. In *INFOCOM 2009, IEEE*, pp. 262–270. IEEE, 2009.

[23] Zhenqiang Ye, Srikanth V Krishnamurthy, and Satish K Tripathi. Effects of multipath routing on tcp performance in ad hoc networks. In *Global Telecommunications Conference, 2004. GLOBECOM'04. IEEE*, Vol. 6, pp. 4125–4131. IEEE, 2004.

[24] Yujun Zhang, Guiling Wang, Qi Hu, Zhongcheng Li, and Jie Tian. Design and performance study of a topology-hiding multipath routing protocol for mobile ad hoc networks. In *INFOCOM, 2012 Proceedings IEEE*, pp. 10–18. IEEE, 2012.

[25] André Zúquete and Carlos Frade. A new location layer for the tcp/ip protocol stack. *ACM SIGCOMM Computer Communication Review*, Vol. 42, No. 2, pp. 16–27, 2012.

発表文献

[1] 小原知也, 若原恭. 早期再送によるアドホックネットワーク上の TCP 転送のグッドプット改善. 電子情報通信学会 2014 年総合大会, B-6-158, Mar. 2014.(発表予定)

[2] 小原知也, 若原恭. アドホックネットワークにおける TCP 通信のグッドプットを改善する早期再送. モバイルネットワークとアプリケーション研究会 (MoNA), Mar. 2014.(発表予定)