IEEE 802.11n無線LAN上のTCP通信における Bufferbloat問題の解決方法に関する研究

野元 祐孝

電気通信大学 大学院情報システム学研究科

博士（工学）の学位申請論文

2018年3月

IEEE 802.11n無線LAN上のTCP通信における Bufferbloat問題の解決方法に関する研究

博士論文審査委員会

主査　加藤 聰彦　教授

委員　森田 啓義　教授

委員　吉永 努　教授

委員　藤井 威生　教授

委員　大坐畠 智　准教授

Abstract

Recently, high speed wireless LANs such as IEEE 802.11n are introduced widely. They use new high throughput data transfer mechanisms, such as frame aggregation and block ACK, which decrease the possibility of TCP segment losses and cause the bufferbloat problem where delay increases due to the buffered TCP segments.

This paper proposes a new method which resolves the bufferbloat problem over IEEE 802.11n wireless LANs by weakening their powerful retransmission capability intentionally when the data rate is low. Specifically, this paper provides three contributions. The first is the detailed performance evaluation on the bufferbloat problem invoked by TCP communication over IEEE 802.11n wireless LAN. The performance evaluation experiment was performed using an actual terminal and access point by changing the distance between them. The data rate, throughput, round-trip time, queue length, and TCP internals such as congestion window and retransmission count are examined in detail, and the results clarifies that, in the low data rate, no TCP segments are lost, and as a result the congestion window increases, which makes the internal sending queue piles up.

The second contribution is that this paper shows a method to implement the proposal in a data sending terminal. This is a straightforward implementation. The performance evaluation shows that the proposed method successfully decrease the buffering delay, similarly with the conventional scheme such as CoDel, which is an active queue management technique.

The third contribution is that this paper shows a method to implement the proposal in an access point, which works as a data receiver. Since different kinds of terminals are used in wireless LANs, it is difficult to install schemes avoiding the bufferbloat problem in all terminals. So, the third contribution proposes a way to allow access points to provide a similar functionality with the second contribution. The results of performance evaluation clarifies that this mechanism can suppress queueing delay in sending terminals without degrading TCP throughput.

要旨

近年，無線 LAN が企業や一般家庭において広く普及しており，無線 LAN の規格である IEEE 802.11 とその一連の規格が策定されている．特筆すべきは 802.11n のアグリゲーションと BlockACK である．802.11n では速度改善のための工夫が物理層だけでなく MAC 層にも関連して行われており，複数のフレームを 1 つにまとめることでオーバーヘッドを削減するアグリゲーションやそれらのフレームに一括して即時に ACK を行う HT-Immediate BlockACK の導入が行われている．この 802.11n 以降に採用されている仕組みはそれまでの 802.11 無線 LAN の送受信の方式から大きく変わっており，通信特性の向上に寄与している．

一方，LAN 上の通信では，TCP が支配的である．TCP は時々刻々と変化するネットワークの状況，例えば輻輳による帯域幅の変化やパケットロスに対応するために，再送や輻輳制御の仕組みを備えている．その場合のデータ送信速度は，輻輳ウィンドウを用いて制御される．TCP では輻輳ウィンドウを制御するための輻輳制御アルゴリズムが多数提案されているが，一般的な PC やスマートフォンなどの OS においては NewReno などのロスベースの輻輳制御アルゴリズムが利用される．

さらに TCP に関連して Bufferbloat という新たな問題が指摘されている．輻輳などの要因によってネットワーク上に存在するバッファに過度にパケットがキューイングされることで，パケットが長時間滞留し，それによって適切に動作しなくなり，スループットや遅延が悪化する問題である．Bufferbloat 問題を解決するための研究がいくつか行われている．Bufferbloat については無線 LAN においても発生することが指摘されているが，詳細に検討されていない．

IEEE 802.11n 無線 LAN においては，アグリゲーションや HT-Immediate BlockAck などの新しい仕組みによって，それまでの無線 LAN から通信特性が向上している．802.11n の新しい仕組みによって再送が強力になり TCP におけるセグメントロスが発生しにくくなった場合，TCP の Bufferbloat 問題が発生する懸念がある．

本研究では，802.11n 無線 LAN アクセスポイントを介して行われる TCP 通信を原因とする Bufferbloat による性能悪化を抑制することを目的とする．

本論文では，第一の研究として，実際に 802.11n 無線 LAN アクセスポイントを用いて

ネットワークを構成し，端末を移動させながらTCP通信を行う実験によって，データレートに応じてTCP性能がどのように変化するかを調査する，Bufferbloat問題の実験的検討を行う．その結果，TCPの実効速度についてはデータレート全域で大きな影響は見られないが，低データレート時の遅延性能が悪化していることを明らかにする．

続けて，802.11nを用いた通信におけるBufferbloat問題に対してのアプローチの検討を行い，既存のキュー長を指標としてパケットロスを行うREDなどのアクティブキュー管理方式は適切ではなく，別途指標を検討する必要があることを示す．指標としてCoDelの採用するキュー滞留時間の他，802.11n無線データレートの高低を用いることができる．本論文の第二の研究として，IEEE 802.11n無線LANにてデータレートに応じて再送回数を抑制する方法の提案を行う．802.11nを用いた通信におけるBufferbloatの要因として，その強力な再送方式にあるとし，最大再送回数を小さくすることによって再送を抑制することで意図的にセグメントロスを発生させ，ロスベースTCPの輻輳ウィンドウサイズの意図的な減少を狙う．さらに，端末側の無線LANデバイスドライバへの提案手法を適用し，評価した結果について述べる．802.11nで行われる通信のデータレートに応じて，低データレートであるほど最大再送回数を制限することによって，スループットに影響を与えることなく遅延性能の悪化を抑制できることを示す．また，再送回数の抑制をTCPのみに限定することで，パケットロスによる影響はその他のトランスポート層プロトコルに及ばないことを示す．

第三の研究として，第二の研究を発展させ，アクセスポイントの無線LANデバイスドライバへ適用した方式の詳細設計，評価実験，考察について述べる．アクセスポイント側で実装することによって端末側の変更を行うことなく，様々な端末に対応することができる．第二の研究の手法をアクセスポイント側実装に適応するため，CRCエラーのフレームヘッダ情報の活用による端末側の再送回数の推定や疑似的なリトライアウトの採用などの工夫を含んだ詳細設計を行い実装する．アクセスポイント側による手法が第二の研究と同様に遅延時間の悪化解消に有効であることを示す．さらに，既存のBufferbloat解決手法と本提案手法の重複適用時においても，悪影響を及ぼさないことを示す．

目 次

第 1 章　はじめに **1**

1.1　背景 1

1.2　本研究の問題意識とアプローチ 3

1.3　本論文の構成と内容 5

第 2 章　関連する技術と研究 **7**

2.1　IEEE 802.11 の概要 7

2.2　802.11 のマルチレートと再送 9

2.3　IEEE 802.11n およびフレームアグリゲーション・ブロック ACK の手順 . . 11

2.4　TCP と輻輳制御アルゴリズムの動作概要 20

2.4.1　TCP ヘッダの構造 20

2.4.2　スライディングウィンドウ 25

2.4.3　輻輳制御 26

2.4.4　TCP Tahoe 28

2.4.5　TCP Reno 30

2.4.6　TCP NewReno 31

2.5　TCP の課題と対応する実装 32

2.5.1　TCP Vegas 33

2.5.2　TCP Westwood+ 34

2.5.3　CUBIC TCP 35

2.6　Bufferbloat 問題の概要 36

2.7　Bufferbloat 問題に関連する研究 38

2.7.1　CoDel 39

2.7.2　TCP small queues 40

2.7.3 DRWA（Dynamic Receive Window Adjustment） 41
2.8 802.11n における Bufferbloat 問題 42
2.9 本研究の位置づけ 43

第 3 章 802.11n 無線 LAN 上の Bufferbloat 問題に関する実験的検討 45
3.1 実験条件 45
3.2 実験結果と考察 47
3.3 IEEE 802.11n の強力な再送 53
3.4 802.11n における Bufferbloat 問題の整理 54
3.5 まとめ 55

第 4 章 無線データレートに応じて再送機能を低下させる方法の提案 56
4.1 Bufferbloat 問題への既存アプローチの 802.11n 無線 LAN への適応性 56
4.2 方式提案のための基本検討 58
4.3 データレートに応じて再送機能を制御する手法の提案 60
4.4 提案方式の実装方法に関する検討 61
4.5 まとめ 63

第 5 章 端末側によるデータレートに応じて再送機能を制御する手法 64
5.1 詳細なアプローチ 64
5.2 最大再送回数の減少の影響範囲の限定 65
5.3 提案手法の無線 LAN ドライバへの実装の検討 66
5.4 最大再送回数の検討 68
5.4.1 最大再送回数を 2 回に制限した実験 68
5.4.2 データレートに応じた最大再送回数の決定方法 74
5.5 提案手法の端末側での実装方式 74
5.6 実験条件 75
5.7 実験結果 77
5.7.1 提案手法，CoDel，ネイティブの 802.11n の比較 77
5.7.2 Cubic，Westwood，Vegas との比較 85
5.8 まとめ 85

第 6 章　アクセスポイント側によるデータレートに応じて再送機能を制御する手法　89
6.1　詳細なアプローチ 89
6.2　シーケンスイメージ 90
6.3　疑似的なリトライアウト処理による影響 92
6.4　ドライバにおける実装 92
6.5　動作例 93
6.6　評価実験 95
6.6.1　実験条件 96
6.6.2　全体的な実験結果 97
6.6.3　個別の測定の評価 108
6.6.4　リトライアウトインデックスの影響 109
6.7　まとめ 111

第 7 章　結論　112
7.1　本研究のまとめ 112
7.2　本研究の貢献 115
7.2.1　802.11n 無線 LAN における Bufferbloat 問題の詳細な検討 115
7.2.2　802.11n 無線 LAN における Bufferbloat 問題の端末側による解決 . . 115
7.2.3　802.11n 無線 LAN における Bufferbloat 問題のアクセスポイント側による解決 116
7.3　今後の展望 116

表 目 次

2.1 Bufferbloat 問題に対するアプローチの実装箇所の一覧 44

3.1 端末の詳細仕様 46
3.2 使用可能なデータレート 46

4.1 802.11n における Bufferbloat 問題へのアプローチの比較 56

5.1 最大再送回数 74
5.2 端末の仕様 76
5.3 可能なデータレート 76

6.1 AP と端末の仕様 96
6.2 1 回の通信実験の詳細 97
6.3 提案方式ありの場合の TCP スループットの割合 102

7.1 802.11n 無線 LAN における Bufferbloat 問題への対応の比較 113

図 目 次

2.1 802.11n の PLCP フォーマット（Non-HT と HT-mixed） 12
2.2 1,500 バイトのデータに対するプリアンブルの割合 13
2.3 データフレームのフォーマット 14
2.4 802.11n（5Ghz 帯）のデータ送信手順の様子 14
2.5 A-MPDU のフォーマット 15
2.6 Block Ack の手順 17
2.7 スコアボード・コンテクスト制御の例 18
2.8 TCP ヘッダフォーマット 21
2.9 スライディングウィンドウ 25
2.10 Bufferbloat 問題のイメージ 37

3.1 ネットワーク構成の概要 46
3.2 機器の配置と移動の様子 47
3.3 0.5 秒単位の平均スループットの時間変化 48
3.4 無線データレートの時間変化 48
3.5 TCP 輻輳ウィンドウの時間変化 49
3.6 TCP セグメント毎の RTT 50
3.7 送信待ちキュー長の時間変化 51
3.8 データフレーム再送率の時間変化 54
3.9 Bufferbloat 問題発生のイメージ 55

4.1 802.11n における Bufferbloat 問題へのアプローチの各イメージ 57
4.2 データレートから試算したキューイング遅延 59
4.3 提案方式のイメージ 61
4.4 A-MPDU の送受信処理の分担 62

5.1 デバイスドライバの処理 67
5.2 無線データレートの時間変化（付加遅延なし） 69
5.3 スループットの時間変化（付加遅延なし） 69
5.4 往復遅延時間の時間変化（付加遅延なし） 70
5.5 送信待ちキュー長の時間変化（付加遅延なし） 70
5.6 TCP 輻輳ウィンドウ値の時間変化（付加遅延なし） 71
5.7 ネイティブのスループットの時間変化（付加遅延 100 ミリ秒） 72
5.8 ネイティブの往復遅延時間の時間変化（付加遅延 100 ミリ秒） 72
5.9 最大再送回数を制限した手法のスループットの時間変化（付加遅延 100 ミリ秒） 73
5.10 最大再送回数を制限した手法の往復遅延時間の時間変化（付加遅延 100 ミリ秒） 73
5.11 提案手法の最大再送回数 74
5.12 実験環境 75
5.13 追加遅延なしの場合の Cubic TCP の結果 79
5.14 追加遅延が 100ms の場合の Cubic TCP の結果 80
5.15 追加遅延が 100ms の場合の TCP Reno の結果 81
5.16 データレートが 216Mbps の場合の TCP Reno の個別の結果（100ms 追加遅延） 82
5.17 100Mbps より高いデータレート時の TCP 通信における平均スループットと平均パケットロス数 84
5.18 Cubic，Vegas，Westwood の比較結果（追加遅延なし） 86
5.19 Cubic，Vegas，Westwood の比較結果（追加遅延 100ms） 87

6.1 提案手法のシーケンス例 91
6.2 デバイスドライバの処理 94
6.3 提案方式による AP の処理の例 95
6.4 実験環境 96
6.5 平均データレートと平均誤り率の対応 98
6.6 平均データレートと Ping の平均 RTT の対応 98

6.7 平均データレートと平均送信キュー長の対応 99
6.8 平均データレートと平均 TCP スループットの対応 100
6.9 Ping RTT の時間変化 103
6.10 TCP スループットの時間変化 103
6.11 平均 MAC データレートが 12Mbps の場合の Ping 通信の RTT の累積分布 . 104
6.12 平均 MAC データレートが 12Mbps の場合の TCP 通信の輻輳ウィンドウの時間変化 105
6.13 平均 MAC データレートが 45Mbps の場合の Ping 通信の平均 RTT の累積分布 106
6.14 平均 MAC データレートが 45Mbps の場合の TCP 通信の輻輳ウィンドウの時間変化 107
6.15 リトライアウトインデックスを変更した場合の Ping の平均 RTT 109
6.16 リトライアウトインデックスを変更した場合の平均 TCP スループット . . 110

第1章　はじめに

1.1　背景

近年，無線 LAN が企業や一般家庭において広く普及している．有線 LAN とは異なり，端末ごとに物理回線の敷設の手間がない，自由なレイアウトが採用でき，再レイアウトに手間がかからないなどの特徴がある．こうした無線の特徴を生かし，大学であれば学内ネットワークの接続口として，企業においてはオフィス，会議室，イベントホールなどで活用されている．一般家庭においても，LINE などに代表されるスマートフォンでのインターネット電話や，キッチンなど有線 LAN を敷設しにくい場所での利用などに有効に利用されている．また公衆無線 LAN として飲食店や宿泊サービスなどでも活用されている．

一方，無線 LAN では無線の特性によって性能に影響を受けやすく，通信品質が大きく変化する．LAN における通信品質は，実効速度，遅延時間，安定性（フレーム損失率）などが挙げられる．実効速度はファイル転送時間などの短縮に，遅延時間は IP 電話やリモートログインなどの快適性に，安定性は IP 電話の音質に関連する．異なるベンダー同士の相互接続性を満たしながら，これらの品質要求に応えるため，無線 LAN の規格である IEEE 802.11[1] とその一連の規格が策定されている．

IEEE 802.11 では無線 LAN の基本となるインフラストラクチャモードやフレーム送信，再送などの仕組みを規定し，また複数のデータレートに対応するマルチレートについても定めている．その後の一連の規格では速度改善のための工夫が行われており，IEEE 802.11b では 11Mbps まで，802.11a や 11g では 54Mbps まで，802.11n では 600Mbps まで，そして 802.11ac では 7Gbps までのデータレートに対応するなど高速化が行われてきた．また IEEE 802.11e では遅延性能の要求に対応するため優先制御を規定している．

802.11 のマルチレートでは，アクセスポイント（AP）と端末などの装置同士が近接しており無線品質が良く伝送誤りが少ない場合は，より高速なデータレートを選択して広帯域に送信できるようにする．他方，通信装置同士に距離があるため無線品質が悪く伝送誤

りが多い場合は，より低速なデータレートを選択して伝送誤りを抑制して再送やフレームロスを減らし，効率的に通信できるようにしている．これによって有線 LAN に比べて，実効速度は大きく変化する．

これら 802.11 の工夫の中，特筆すべきは 802.11n のアグリゲーションと BlockACK である．802.11n では速度最善のための工夫が物理層だけでなく MAC 層にも関連して行われており，複数のフレームを 1 つにまとめることでオーバーヘッドを削減するアグリゲーションやそれらのフレームに一括して即時に ACK を行う HT-Immediate BlockACK の導入が行われている．この 802.11n 以降に採用されている仕組みはそれまでの 802.11 無線 LAN の送受信の方式から大きく変わっており，通信特性の向上に寄与している．

一方，LAN 上の通信では，TCP[2][3] が支配的である．TCP は時々刻々と変化するネットワークの状況，例えば輻輳による帯域幅の変化やパケットロスに対応するために，再送や輻輳制御の仕組みを備えている．この制御はエンドツーエンドのホスト間で行われ，そのデータ送信速度は輻輳ウィンドウを用いて制御される．TCP では輻輳ウィンドウを制御するための輻輳制御アルゴリズムが多数提案されているが，一般的な PC やスマートフォンなどの OS においては NewReno[4] や CUBIC[5] などのロスベースの輻輳制御アルゴリズムが利用される．ロスベースのアルゴリズムは，セグメントロスによって輻輳を検知する．経路に伝送誤りやハンドオーバによるセグメントロスが発生しうる無線環境が存在する場合，伝送誤りと輻輳との区別がつかず，輻輳と判断してウィンドウを縮小することから，実効速度が低下してしまう問題などが報告されている．こうした問題に対応するために，伝送誤りの多い無線を想定して設計された TCP Westwood[6] や，セグメントロスではなく往復遅延時間（RTT）を用いて輻輳を検知し制御する TCP Vegas[7] などの提案が行われている．

さらに TCP に関連して Bufferbloat[8] という新たな問題が指摘されている．輻輳などの要因によってネットワーク上に存在するバッファに過度にパケットがキューイングされることで，パケットが長時間滞留し，それによって適切に動作しなくなり，スループットや遅延が悪化する問題である．Bufferbloat 問題を解決するための研究がいくつか行われている．CoDel[9] はアクティブキュー管理の一種であり，送信キューにおけるパケットの滞在時間（キューイング遅延）からパケットの破棄を行う．パケットの破棄によって TCP の輻輳制御が働き，輻輳ウィンドウを縮小させる．かつての RED（Random Early

Detection）[10] などの AQM に比べてパラメータレスで制御できることが利点であるが，パケットロスによる無駄な送信を行うため回線を消費する．TCP small queues は TCP から送出されるセグメントをキューの状況に応じて TCP セッションごとに特定のサイズに制限する方法であり，送信によって空きができると送信を再開する TCP の実装である．パケット破棄を行わないため回線の消費は少ないが，TCP のエンドポイントとボトルネックキューが別の端末である場合は仕組み上，活用することはできない問題がある．この Bufferbloat については無線 LAN においても発生することが指摘されているが，詳細に検討されていない．

1.2 本研究の問題意識とアプローチ

IEEE 802.11n 無線 LAN においては，アグリゲーションや HT-Immediate BlockAck などの新しい仕組みによって，それまでの無線 LAN から通信特性が向上している．802.11n の新しい仕組みによって再送が強力になり TCP におけるセグメントロスが発生しにくくなった場合，TCP の Bufferbloat 問題が発生する懸念がある．ロスベースの TCP 輻輳制御アルゴリズムはセグメントロスの発生から輻輳の判断を行い，輻輳ウィンドウサイズを減少する．しかしながら，802.11n の無線電波の状況が悪化し低いデータレートが選択されるような状況においても，マルチレートやアグリゲーション，Block ACK などの仕組みによってセグメントロスが発生しにくくなっており，TCP は輻輳を検知できず，輻輳ウィンドウサイズの減少が行われない．そのため，TCP のセグメント送出速度は TCP 送信バッファに制限される最大速度まで拡大する．低いデータレートによって低下した無線 LAN の実効速度を TCP の送出速度が上回る場合，無線デバイスによって送信しきれなかった TCP セグメントは送信キューにつながれ送信まで待機する．送信キューにつながれて待機する時間（キューイング遅延）の最大は，フルにキューイングして低いデータレート（例えば 6Mbps）で送信される場合，数秒オーダーの長さとなる．

かつての無線環境の検討において TCP は無線の伝送誤りを原因とするセグメントロスを輻輳と区別できずにスループットの低下を招いていたが，興味深いことに，802.11n 無線 LAN の環境においてはその強力な再送機能によって輻輳を検知できずに遅延性能の悪化を引き起こすことになる．

Bufferbloat 問題が発生した場合，秒オーダーの遅延性能の悪化は，TCP のバルク転送を行う通信に対してはさほど影響を与えないが，送信キューを共有するインタラクティブ通信に影響を及ぼし，例えばリモートログイン時の操作の快適性が損なわれる．

本研究では，802.11n 無線 LAN アクセスポイントを介して行われる TCP 通信を原因とする性能悪化を抑制することを目的とする．LAN として求められる品質要求を考慮し，802.11n 無線 LAN のマルチレートが取り得るデータレートの全域において TCP スループット性能を損なうことなく，遅延悪化を抑制することを目指す．アクセスポイント介して行われる TCP 通信として，イントラネットの通信とインターネットを含む通信の双方を想定する．この遅延悪化の問題は 802.11n 無線 LAN アクセスポイントを介したダウンリンク方向の通信，アップリンク方向の通信の双方にて発生しうる．ダウンリンク方向の通信の場合はアクセスポイントのキュー内で，アップリンク方向の通信の場合は端末のキュー内でパケットが滞留することとなる．本研究では，特に端末からアクセスポイントへのアップリンク方向の TCP 通信を対象とする．

本論文では，まず第一の研究として，実際に 802.11n 無線 LAN アクセスポイントを用いてネットワークを構成し，端末を移動させながら TCP 通信を行う実験によって，データレートに応じて TCP 性能がどのように変化するかを検討する．その結果，TCP の実効速度についてはデータレート全域で大きな影響は見られないが，低データレート時の遅延性能が悪化していることを明らかにする．

この遅延悪化に対応可能な既存アプローチとして，TCP 輻輳制御アルゴリズムの変更，Bufferbloat 問題に対する研究で挙げた CoDel や TCP small queues などが挙げられる．

これに対して，本研究では，無線 LAN の通信手順を考慮したアプローチを採用する．具体的には，802.11n を用いた通信における Bufferbloat の要因として，その強力な再送方式を挙げる．この再送は通常，デバイスドライバによって定められる最大再送回数によって制限される．その最大再送回数を小さくすることによって再送を抑制することで，意図的にセグメントロスを発生させ，ロスベース TCP の輻輳ウィンドウサイズの意図的な減少を狙う．データレート情報の取得を試み，低データレートに限定して適用することで高データレート時の性能低下を回避する．さらに原因となる TCP 以外の通信の性能悪化を防ぐため，TCP 通信のみ最大再送回数の制限を行うこととする．

さらに，端末の無線 LAN デバイスドライバへ提案手法を適用し，評価した結果につい

て述べる．本研究ではアップリンク方向を対象とする．802.11n で行われる通信のデータレートに応じて，低データレートであるほど最大再送回数を制限することによって，スループットに影響を与えることなく遅延性能の悪化を抑制できることを検証する．また TCP Westwood や TCP Vegas などの既存の輻輳制御アルゴリズムとも性能の比較を行う．

しかしながら様々な端末の接続が考慮される 802.11n アクセスポイント環境下において個別の端末への適用は困難であり，第二の研究の設計も例外ではない．そこで第三の研究として，第二の研究を発展させ，アクセスポイントの無線 LAN デバイスドライバへ適用した方式の設計，評価実験，考察について述べる．アップリンク方向を対象とする．アクセスポイント側で実装することによって端末側の変更を行うことなく，様々な端末に対応することができる．第二の研究では，データレートに応じて送信端末による最大再送回数の制限を行うことを行ったが，例えば再送機能の弱化，再送回数の取得などの手法について，アクセスポイント側実装のために適応させる必要がある．再送機能の弱化においては，受信側において送信端末が送信しただろう再送回数を数え，最大再送回数以上の再送が行われた場合はリトライアウト（再送回数超過による中断）が行われたかのように振る舞い，以降 MPDU を正しく受信できたとしても，上位層への通知を行わず破棄する．再送回数の取得においては，フレームヘッダに含まれるシーケンス番号から推定を行い，その推定には CRC エラーとなったフレームを用いる．通常，CRC エラーとなったフレーム情報はデバイスドライバによって破棄されるが，この情報に含まれるシーケンス番号が正しいものとして，対応するシーケンス番号の再送回数を加算する．これらのアイディアをアクセスポイント側デバイスドライバに実装し，第二の研究と同様に遅延時間の悪化解消に有効であることを検証する．

1.3　本論文の構成と内容

本論文の構成は次のとおりである．2 章では，IEEE 802.11 無線 LAN，TCP，Bufferbloat の関連技術，関連研究について述べる．3 章では，802.11n を用いた無線 LAN 上において TCP 通信を行う場合に，端末と AP の距離を変えた場合に TCP 性能がどのように変化するのかを明らかにする．4 章では，3 章の結果を踏まえ，IEEE 802.11n 無線 LAN における Bufferbloat 問題をデータレートに応じて再送回数を抑制する方法の提案解決する方

式の提案を行う．5 章では端末側にて提案方式を設計・実装した提案の性能評価の結果を示す．6 章では AP 側にて設計・実装した提案の性能評価の結果を示す．最後に結論として，7 章にて，論文全体を総括し，本研究の意義や成果をまとめ，今後の課題について言及する．

第2章　関連する技術と研究

本章では，まず，802.11n について説明し，それまでの無線 LAN とは通信特性を大きく変化させる 802.11n の再送手順について示す．また，ネットワークの通信で広く利用される TCP の様々な機能について紹介し，TCP の輻輳制御アルゴリズムの既存研究について紹介する．次に TCP によって引き起こされる Bufferbloat 問題について，その仕組みと解決方法について述べる．

2.1　IEEE 802.11 の概要

IEEE 802.11 は IEEE により 1997 年に策定された無線 LAN の標準化された仕様であり，MAC 層と PHY 層を規定している．標準化仕様に準拠する製品であれば相互接続できることを目的としている．

802.11 が策定される以前，無線 LAN はベンダー毎に独自に存在しており，異なるベンダー同士の製品で相互に接続することはできなかった [11]．NCR によって設計された無線 LAN 製品である WaveLan は 1988 年に市場に投入され，WaveLan の設計は NCR 社によって IEEE 802 LAN / MAN Standards Committee に寄稿されている，この貢献が 802.11 無線 LAN ワーキンググループの創設につながっている [12].

IEEE 802.11 は有線 LAN の規格である Ethernet(IEEE 802.3)[13] と同様にフレーム単位で通信を行う．その基礎となるアクセス方式は DCF（Distributed Coordination Function）であり，CSMA/CA（Carrier Sense Multiple Access with Collision Avoidance）として知られている仕組みである．CSMA は Listen-Before-Talk（LBT）形式であり，端末がフレームを送信する際に，他の端末がすでに送信を開始しているために媒体が Busy になっていないかどうかを感知する．もし，媒体が Busy ではなく Idle である場合，送信が行われる．この点では 802.3 が採用している CSMA/CD（Carrier Sense Multiple Access with Collision Detection）と 802.11 が採用している CSMA/CA は同様である．CSMA/CD は，その対

象とする有線媒体において送信中に衝突を検知することが可能であり，衝突を検知した際には直ちに送信を停止し，ジャム信号を送信して，すべての受信者へ衝突が発生したことを知らせる．その後，ランダムなバックオフ時間を待つことで衝突を回避しようとする．一方，無線媒体では送信中に衝突を検知することが困難である．そのため，CSMA/CA では媒体が Busy であることを感知したときはランダムバックオフ時間を待ってから送信を行うことで，出来る限り衝突を避けようとする．

802.11 では，フレーム送信の間に特定の時間間隔を待つことを定めている．送信を行う端末は，この時間間隔の間，媒体が Idle であることを確認しなければならない．このフレーム間の時間間隔は IFS（InterFrame Space）と呼ばれる．時間間隔の長さに応じて SIFS や DIFS などが存在する．異なる長さの IFS が存在するのは優先度を決定するためであり，短い時間の IFS であるほど優先度は高い．より長い時間間隔の IFS は先に通信が行われた場合，媒体が Idle になるまで待ち，ランダムバックオフ時間を待つことになる．

SIFS（Short Interframe Space）は 802.11 規格における最小の IFS であり，最も優先度が高い IFS である．SIFS は，例えばデータフレームに対する ACK フレームの応答時に用いられる時間間隔である．データフレームを受信した端末は SIFS の期間内に，受信した PHY データをデコードして MAC に送り込む，SIFS 後の応答のために必要な MAC（ヘッダ）の処理，ACK など応答フレームなどの構築，受信から送信に切り替えて PHY に送信データを送り込むなどの処理を行う．このときキャリアセンスは行わない．DIFS（DCF Interframe Space）はデータフレームや管理フレームを送信するときに使われる．SIFS と 2 つのスロット時間で構成される．ランダムバックオフ時間は衝突を防止するための待ち時間であり，0 からコンテンションウィンドウまでの範囲からランダムにバックオフ回数を取り出し，メディアがアイドルである場合，バックオフ回数を引いていく．バックオフ回数の 1 回あたりはスロット時間に相当する．このバックオフ回数が 0 になるとフレームの送信を行える．もしバックオフが同期してしまいデータの衝突が発生した場合は，コンテンションウィンドウを 2 倍にして再度ランダムな値を取り出しバックオフを行う．このような動作を指数バックオフ制御と呼ぶ．コンテンションウィンドウは最大で 1023 まで広がる．802.11 端末が一か所に集中すると，このバックオフで衝突が発生しやすくなり効率が悪くなる．

2.2　802.11 のマルチレートと再送

802.11 は複数のデータレートによるデータ転送に対応しており，フレームを送信する毎に異なるデータレートで送信できる．このデータレートは送信端末と受信端末の双方が対応するレートのみが選択される．802.11 では 1Mbps と 2Mbps の 2 つのみの対応だが，拡張規格である IEEE 802.11b では 11Mbps まで，IEEE 802.11a および 11g では 54Mbps まで，IEEE 802.11n では 600Mbps まで，IEEE 802.11ac では約 7Gbps までの幅広いデータレートに対応する．高いデータレートであるほど，より多くのデータを単位時間あたりに送信することができるため，スループットは大きくなり，ファイル転送などが短時間で行えるようになる．

しかしながら，高いデータレートは遠くまで伝搬しない特性がある．高いデータレートほど必要とされる SN 比（信号雑音比，signal noise ratio）は大きくなる．端末同士が離れると受信信号強度（RSSI, Receive Signal Strength Indicator）は弱くなる．そのため十分な SN 比を満たすことができずに受信できなくなる．このことから，より遠くまで伝搬させるためには，より低いデータレートを選択する必要がある．また，SN 比を下げる要因は距離だけではない．外来波の干渉（電子レンジやコードレス電話機，気象レーダーなど）やフェージング（建物や人間など），他の無線 LAN 端末との競合などの要因も考えらえる．こうした要因，環境の時間的変化や端末の位置などの空間的変化に伴って，効率の良いデータレートは動的に変化する．

802.11 無線 LAN では，このような環境の変化に対して，動的レート切り替え（dynamic rate switching）を行うことができる．このレート切り替えは手動によってデータレートを固定する方法とレート適応アルゴリズムによって自動的に設定する方法の 2 つが存在する．前者のデータレートを固定する方法は用途が限られており，例えばスループットは小さくとも広範囲に通信を行えるようにしたい，との要件がある場合は有効である．後者のレート適応アルゴリズムによる設定は IEEE 802.11 において規定されておらず，無線 LAN デバイスのベンダー毎の独自の実装に任されている．レート適用アルゴリズムは一般的に受信信号強度やフレーム到達率を用いてデータレートの選択を行う．ほとんどの利用者は，後者のレート適応アルゴリズムによるレート切り替えを用いて，効率の良いデータレートを選択していると考えらえる．このように環境の変化に応じて，受信することができない無駄な送信を減らすことで，効率よく受信できるよう促す仕組みが IEEE 802.11

には規定されている．動的レート切り替えの結果，上位層からは実効スループットが変化しているように見える．Ethernet 有線 LAN の場合，リンク速度は変化することはなく，この点は無線 LAN の大きな特徴の 1 つである．

無線 LAN においては信号強度不足や外来波の影響によって，伝送誤りが発生する．この伝送誤りを許容できるかどうかは使用されるアプリケーションに左右される．例えば，VoIP やテレビ電話，映像伝送などの用途では伝送誤りはある程度まで許容される．それに対して Web ブラウジング，ファイル転送，メール，リモートログオンなどの用途のアプリケーションでは伝送誤りは想定されておらず許容されない．このようなアプリケーションに対しては後述する TCP などの上位層プロトコルによって，誤り訂正がなされる必要がある．しかしながら，TCP による誤り訂正はエンドツーエンド間の対応であり，伝送誤りの検知からデータの再転送を行うまで，終端間を往復する時間がかかる．また該当する無線 LAN 区間以外の回線の帯域を消費してしまう．さらに TCP は有線 LAN に最適化されて設計されており，伝送誤りによって輻輳制御が過度に働くことで輻輳ウィンドウが開かず大きくスループットが低下してしまう問題も存在する [14]．そのため，802.11 無線 LAN では単一の相手の通信（ユニキャスト）に限り，802.11 レベルで再送を行えるように規定されている．802.11 はデータフレームを受信すると FCS（Frame Check Sequence）を確認し，受信したデータにビット誤りが含まれていないかどうかを確認する．もし，受信したデータが正しく受信できていた場合，送信者に対して 802.11 ACK フレームを応答する．送信者は ACK フレームを受信すると通信に成功したと判断して，次のデータを送信する．ACK フレームを受信できなかった場合，同じデータフレームを再送する．MAC フレームのサイズ毎に最大再送回数が設定されており，その最大回数までの再送が行われる仕組みとなっている．最大再送回数に到達してもデータフレームの受信に失敗した場合，リトライアウトしたとして再送を諦める．その場合，伝送誤りは上位層に伝番する．802.11 再送は即時に行われるため，必要な送信バッファは高々 1 フレーム分である．送信バッファは ACK を受信すると解放される．802.11 再送によって上位層に対して伝送誤りを伝搬させずに，TCP スループットの低下を防ぎ，またエンドツーエンドの誤り訂正による遅延増加を抑制している．このような再送の仕組みを 802.11 は持っているが，前述したとおり各アプリケーションが要求する信頼性は異なる．例えば，VoIP など遅延性能にセンシティブな用途においては，再送によって遅延時間が増加し通話品質が低下して

しまうため，過度に誤り訂正が行われるよりも，早々に再送を諦めて素早く次のフレームが送信される方が望ましい．アクセスポイントや端末によってはフレームサイズによって再送回数を変更するなどの工夫を行っており，VoIP などが用いるショートフレームの場合，再送回数を小さく設定することも可能である．しかし，802.11 再送の仕組み上，フレームの順番を飛ばすことができないため，アクセスポイントにアソシエーションしている他の端末のロングフレーム（例えばデータフレーム）による再送回数で時間を消費してしまう．こうした事情から，802.11 再送の回数は工場出荷状態において各アプリケーションのトレードオフを考慮した値に設定され，ある程度の伝送誤りは上位層に伝達してしまうと考えられる．

2.3 IEEE 802.11n およびフレームアグリゲーション・ブロック ACK の手順

IEEE 802.11n は 2009 年に策定された 600Mbps までの最大速度に対応する無線 LAN 規格である [15]．複数のアンテナを用いる MIMO や複数のチャネル帯域を用いるチャネルボンディングなどの工夫によって物理的な伝送帯域を広げるとともに，フレームアグリゲーションやブロック ACK によってフレーム送信時のオーバーヘッドを削減することで実効速度の高速化を行っている．

802.11n は，802.11a/g の OFDM（Orthogonal Frequency Division Multiplexing）をベースにハイスループット化が行われている．HT OFDM（High Throughput OFDM）の HT を 802.11n 技術のプレフィックス, 例えば HT PHY（802.11n の物理層）や HT STA（802.11n に対応する端末）などとして用いられている．MIMO を用いた空間多重は 4 つまで対応し，また帯域幅はそれまでの 20MHz から 40MHz まで拡張できるようにすることで，高速化を実現している．AP ではない端末は SISO（Single Input and Single Output）による空間多重化されない帯域幅 20MHz への対応を，AP に対しては帯域幅 20MHz と 2 つの空間ストリームまでの対応を必須としており，それ以上の対応はオプションであり実装に任される．

PPDU のフォーマットは, Non-HT フォーマット, HT-mixed フォーマット, HT-greenfield フォーマットの 3 つが存在する．Non-HT フォーマットは従来の 11a/g のフォーマット

と同等のものである．HT-mixed フォーマットは 11a/g フォーマットと互換性のあるプリアンブルを持っており，これらのデバイスが混在しても解釈することができるフォーマットである．その後，HT-SIG などの 11n 対応端末が読み込むプリアンブルなどが続く．HT-greenfield フォーマットは 11a/g との互換性を排したものであり，11n 専用のプリアンブルとヘッダを採用することよりプリアンブル時間の短縮が可能なフォーマットである．Non-HT フォーマットと HT-mixed フォーマットの 2 つは実装は必須であり，HT-greenfield フォーマットの実装はオプションである．HT-greenfield を採用した場合，既存の 11a/g デバイスとの共存に問題が出る可能性があるため，一般的には HT-mixed フォーマットが利用される．加えて，802.11ac では HT-mixed と同様な VHT フォーマットが定義されるのみであり，HT-greenfield に相当する 11ac に限定されたプリアンブルを採用する形式は規定されていない．

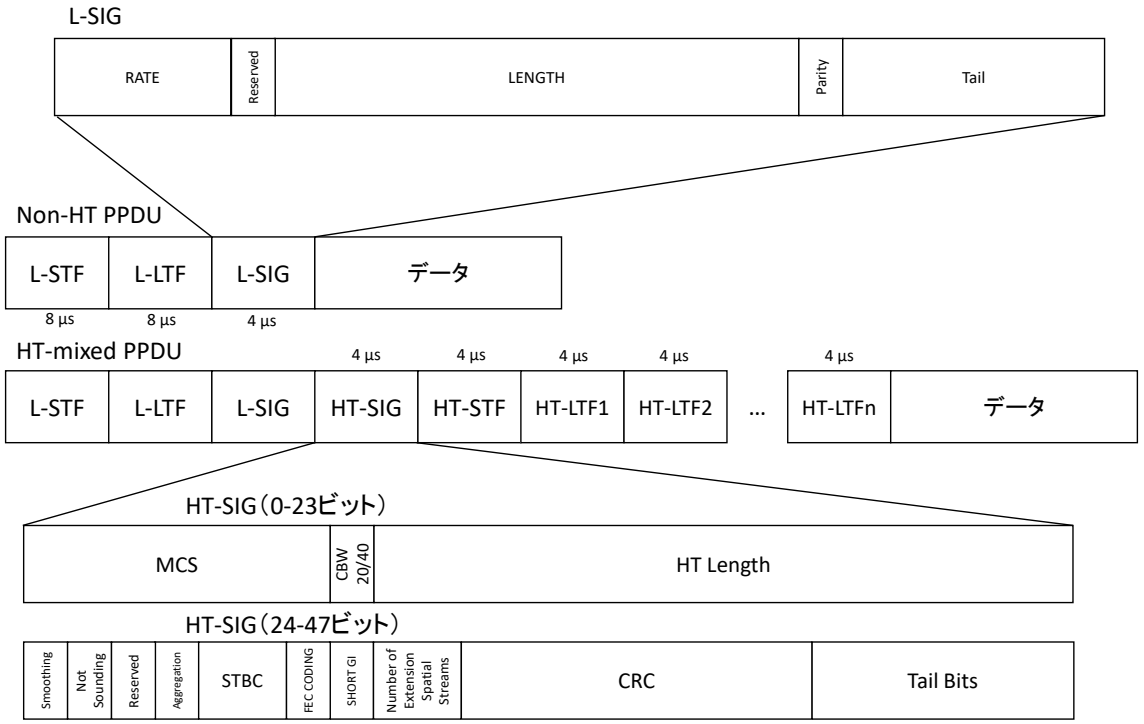


図 2.1: 802.11n の PLCP フォーマット（Non-HT と HT-mixed）

図 2.1 に 802.11n の PLCP フォーマットを示す．Non-HT である PPDU のフォーマットは L-STF（Legacy-Short Training Field），L-LTF（Legacy-Long Training Field），L-SIG（Legacy-Signal），Data フィールドの順に構成される．L-STF は自動ゲイン制御（AGC）の収束，選択ダイバーシチ，タイミング取得と周波数の粗調整に，L-LTF はチャンネル推定と周波数の微調整に用いられる．L-SIG には OFDM の変調モード（11/g のデータレート）

を示すRATEフィールドとPSDUのバイト数を示すLENGTHフィールドが含まれる．その後，PSDUが含まれるDataと続く．DataはRATEによって示されたデータレートで送信される．HT-mixedのPPDUフォーマットは，Non-HTと同様にL-STF，L-LTF，L-SIGから始まり，次にHT-SIG（High Throughput-Signal），HT-STF（High Throughput-Short Training Field），HT-LTF（High Throughput-Long Training Field）群，そしてDataと続く．HT-SIG以降は11a/gでは解釈できない情報であり，このHT-SIG以降のデータ送信完了までにかかる時間をL-SIGのLENGTHで指定することによって，既存の端末には壊れたデータを受信しているように見せかける．HT-SIGには，11nにて拡張された変調と符号化モードを示すMCS（Modulation and Coding Scheme）のインデックス，帯域幅20/40MHzの可否，PSDUの長さ（0から65,535の間），アグリゲーション（A-MPDU）かどうか，MIMOによる空間ストリームの数，HT-SIGそのもののCRCなどの情報が格納される．HT-STFはMIMOのAGC推定のため，HT-LTFは受信機におけるMIMOのチャンネル推定のために利用される．HT-LTFはストリーム数によって複数存在する．このようにデータ（PSDU）を高速に伝送するためにHT-SIGに11nにて拡張されたデータレートの情報を含んでいる．

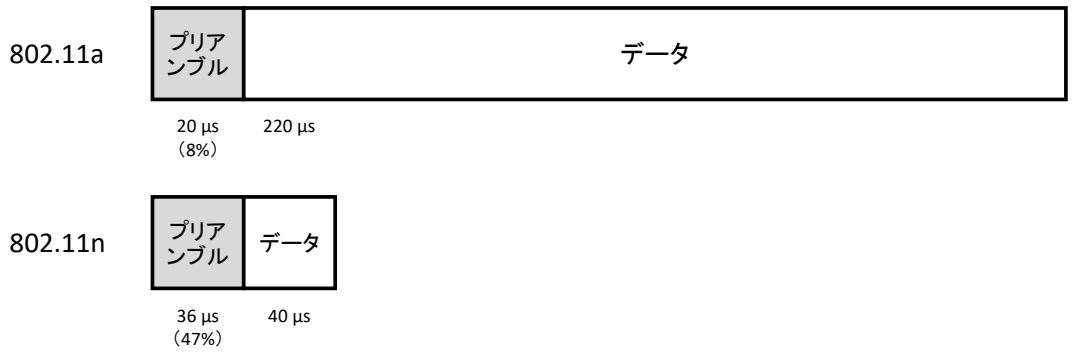


図 2.2: 1,500バイトのデータに対するプリアンブルの割合

このように物理層の改善によって高速なデータレートの使用が可能となるが，そのような高速なデータレートであってもMAC層のオーバーヘッドによって，パフォーマンスの向上は僅かである[16]．図2.2に1,500バイトのデータに対するプリアンブルの割合を802.11aと802.11nにて比較したものを示す．データレートが高速になればなるほどデータの送信時間は短縮され，例えば1,500バイトのデータを300Mbpsのデータレートで送信する場合，データの送信にかかる時間は約40マイクロ秒となる．HT-mixedのプリアンブルの送信にかかる時間は36マイクロ秒であり，約半分の時間を受信機の調整のため

に用いている．802.11a の場合，54Mbps による 1,500 バイトのデータ送信に約 220 マイクロ秒，プリアンブルに約 20 マイクロ秒かかる計算であり，プリアンブルの占める時間は 10%程であることから，802.11n はデータレートの高速化によってオーバーヘッドの割合が大きくなっている．

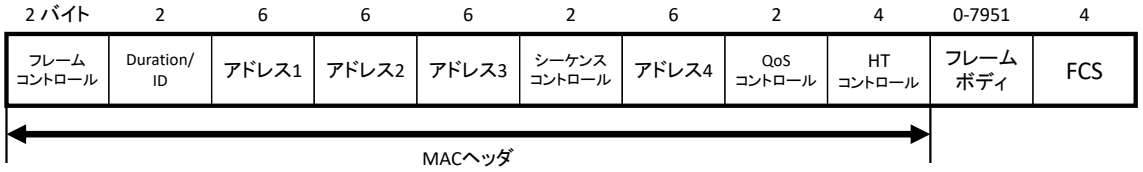


図 2.3: データフレームのフォーマット

図 2.3 に 802.11n のデータフレームのフォーマットを示す．アドレス 4，QoS コントロール，HT コントロールフィールドに関しては，特定のフレームコントロールが指定された場合のみ付与される．QoS コントロールは 802.11e にて，HT コントロールは 802.11n にて規定されている．HT コントロールフィールドは Reverse direction やビームフォーミングのキャリブレーションなどに用いられ，802.11n 形式のデータフレームでは必ず用いられる．QoS コントロール，HT コントロール以外のフィールドは 802.11 と同様である．よって，データフレームでは 802.11 に比べて 6 バイトほどの増分となるが，オーバーヘッド全体の中では微小であり無視できる．

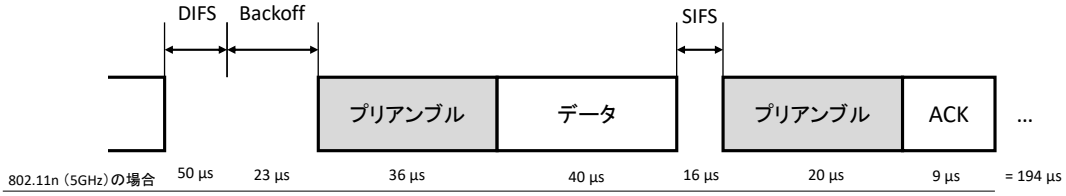


図 2.4: 802.11n（5Ghz 帯）のデータ送信手順の様子

図 2.4 に 802.11n（5GHz 帯）のデータ送信の様子を示し，データ送信の一連の手順におけるオーバーヘッドについて検討する．データフレームを送信後，正しく到着したかどうかを知るために受信端末が ACK フレームを送信する．フレーム送信に際して，データフレームと ACK フレームの間では SIFS として 16 マイクロ秒（5GHz の場合），データフレームを送信する直前には DIFS として 50 マイクロ秒間待つことが規定されている．またデータの送信前にはバックオフを行う．バックオフの平均時間は約 23 マイクロ秒である．ACK フレームは Non-HT プリアンブルで送信されると仮定して 20 マイクロ秒，ACK

フレームのデータは 14 バイトであり，12Mbps で伝送されるとすると，9 マイクロ秒である．したがって，一連の送信にかかる時間は 194 マイクロ秒となり，そのうちデータレート 300Mbps の送信データ割合は 21%である．1 フレームあたりに送るデータ時間に対してプリアンブル・IFS・ACK フレームのオーバーヘッドは大きく，このままデータ送信を行っても最大 60Mbps 程度に留まることになる．例えば HT-greenfield フォーマットを採用することでプリアンブルにかかる時間を 4 マイクロ秒の短縮が可能だが，その効果は限定的である．802.11n ではこれらのボトルネックを解消するためにフレームアグリゲーションと BlockACK を採用している．BlockACK は 802.11e にてオプションとして規定されたが，802.11n では必須としている．

802.11n のフレームアグリゲーションでは，MAC ヘッダ以降の MSDU を集約する A-MSDU と，PLCP ヘッダ以降の MPDU を集約する A-MPDU の 2 通りの方式を定義している．A-MSDU は既存の 2,312 バイトから 7,955 バイトまでペイロードを拡張する．しかしフレーム長に応じてフレームの誤り率が上昇し，また誤りが検出された場合は，A-MSDU を再送する必要がある．A-MPDU はペイロードが 65,535 バイトまで拡張される．A-MPDU では集約した MPDU 毎に誤り検出が行われ，誤り訂正の際には MPDU を個別に再送することが可能となっており，A-MSDU よりも効率が良い．2 つのアグリゲーションのうち，A-MSDU はオプション規定，A-MPDU は必須機能となっており，後者が広く利用されている．

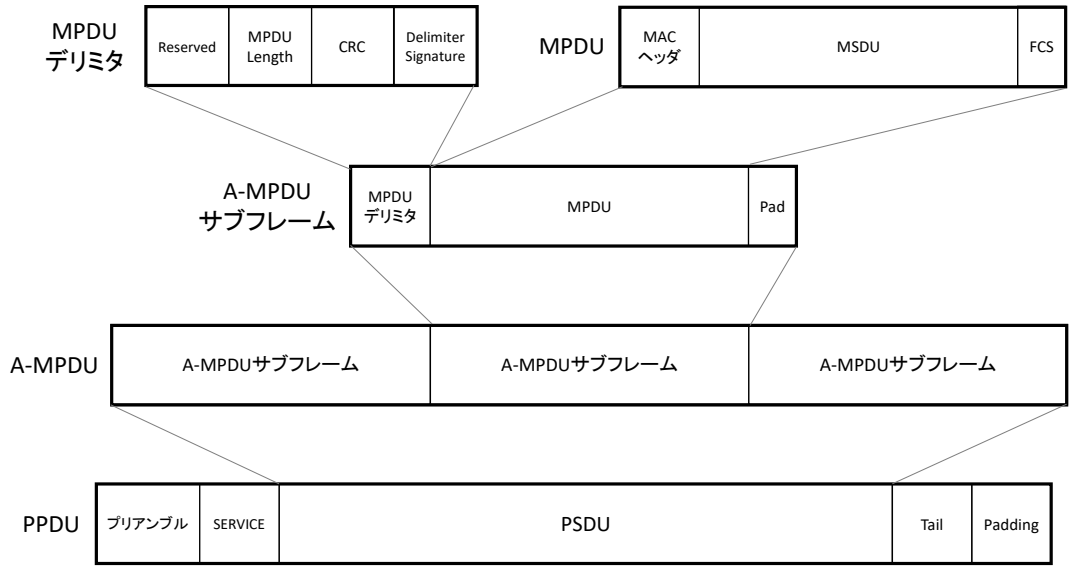


図 2.5: A-MPDU のフォーマット

図2.5に802.11nのA-MPDUのフォーマットを示す．PPDUのプリアンブル後にSERVICE，PSDU，Tail，Paddingと続き，PSDUにA-MPDUが格納される．A-MPDUはA-MPDUサブフレームの集合であり，A-MPDUサブフレームはMPDUデリミタ，MPDU本体，パディングによって構成される．MPDUデリミタには，MPDUの長さやデリミタ自身の誤りを検出するCRCなどが含まれる．またパディングは0から3バイトの長さで，A-MPDUサブフレームを4バイトの倍数になるように調整する．MPDUにはMACヘッダ，MSDU，FCSが含まれ，受信時にはMPDU毎にFCSによる誤り検出が行われる．なお，データフレーム以外の管理フレームや制御フレームは集約されない．

802.11nでは，802.11eのBlock Ackを拡張したものを規定している．Block Ackとはそれまでの ACK フレームに代わる確認応答の方式であり，複数の MPDU に対する確認応答を可能にする．802.11eにてImmediate Block AckとDelayed Block Ackの2つのタイプのBlock Ackが規定された．Immediate Block Ackは，Block Ack Requestフレームを受信した直後にBlock Ackの送信が行われる．この方式では受信したMPDUのデコードや誤り検出などの処理を短時間で行い応答する必要があるため実装の要求が高い．そこで実装の難易度を下げるため，Delayed Block Ackが規定されている．Delayed Block Ackでは，Block Ack Requestフレームに対してACKフレームで応答し，準備が整ってからBlock Ackを送信することができる．Delayed Block Ack方式の場合，既存の実装においてハードウェアの変更を最小限に抑えることができる．802.11nでは，HT-immediate Block AckとHT-delayed Block Ackの2つの応答確認方式を規定しており，これらは802.11eで規定されたBlock Ackの拡張である．802.11nはHT-immediate Block Ackへの対応は必須としている．

HT-immediate Block AckではACKポリシーがNormalの場合，A-MPDUを受信すると即時にBlock Ackフレームを送信するように規定されている．Block Ack Requestを送信する必要がなくなり，さらに効率が良くなる．A-MPDU全体を正しく受信できなかった場合はBlock Ackは応答されないため，タイムアウト後にA-MPDU全体を再送することになる．また従来のBlock Ackはシーケンス番号に加えてフラグメントもできるようになっておりビットマップには1,024ビット（128バイト）のフィールド長が必要であったが，802.11nではフラグメントを取り扱わない圧縮ビットマップとして64ビット（8バイト）のフィールド長にて記述することに対応している．圧縮ビットマップによって，Block

Ack の送信時間の削減とともに，必要なメモリ容量の削減など，実装の要件も引き下げている．

802.11n のアグリゲーションは複数個まとめるため，送信バッファは A-MPDU で多重化する分だけ必要になる．この多重化の最大数は決定されていないが，データ量は 65,535 バイトと規定されている．Block ACK で確認応答が行われたデータフレームは送信キューから解放され，確認応答が行われなかったデータフレームは再度，送信キューに入れられる．受信側では，Block Ack のビットマップを記述するための処理が行われており，スコアボード・コンテクスト制御と呼ぶ．スコアボード・コンテクスト制御では Block Ack ウィンドウを定めて，どの範囲のシーケンス番号の応答をビットマップにて行うのかを判断している．HT-immediate Block Ack では A-MPDU の受信後，即時（SIFS，例えば 16 マイクロ秒以内）に応答する必要があるため，この処理はハードウェアで対応する．また Block Ack 方式では，受信者は正しい順序で到着しなかったフレームを並び替えかえて正しい順序で上位層に伝えるための reoder バッファを持つ．

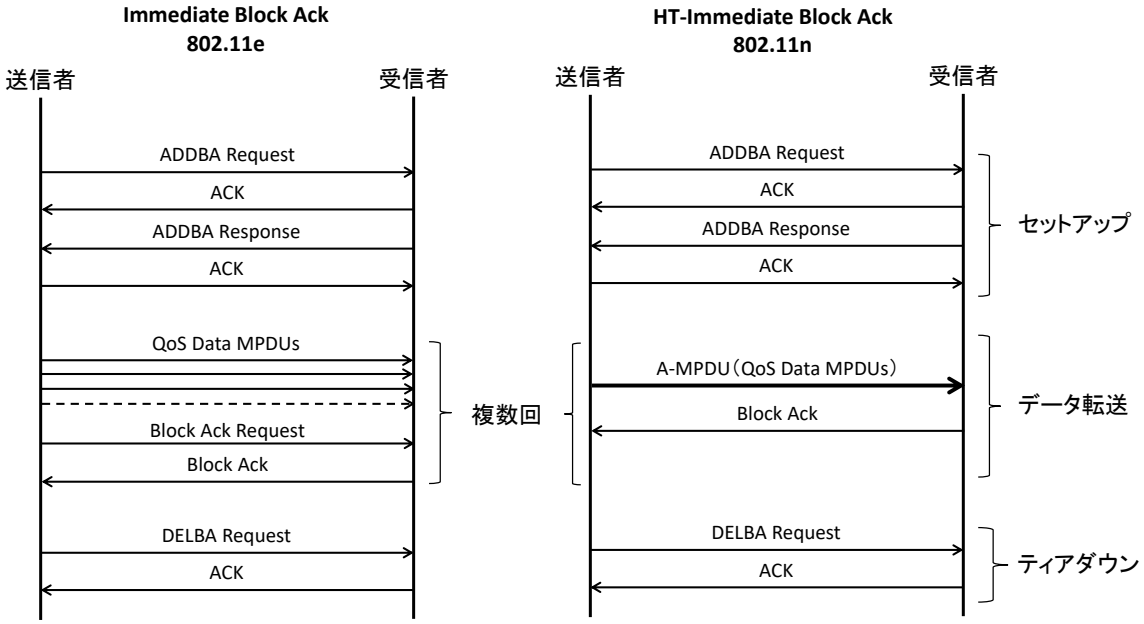


図 2.6: Block Ack の手順

802.11e の Immediate Block Ack と 802.11n の HT-Immediate Block Ack の手順について図 2.6 に示す．Block Ack セッションは管理フレームである ADDBA（add Block Acknowledgement）のリクエスト・レスポンスの交換によって始まる．ADDBA では TID（Traffic Identifier），Block Ack のパラメータ，タイムアウト，開始シーケンス番号などが

送られる．TID はセッションの ID が入る．Block Ack のパラメータでは，バッファサイズ，Block Ack ポリシー（Immediate・Delayed）などが含まれる．レスポンスでは，リクエストに対する可否が ResultCode によって応答される．リクエストによって要求されたバッファサイズによって Block Ack ウィンドウのサイズが決定され，受信者は要求された reorder のバッファサイズを確保することが出来る場合，肯定応答を返す．データ転送では Block Ack を用いる場合，QoS データフレームを用いる．Immediate Block Ack の場合，複数の QoS データ MPDU が送られ，Block Ack Request の要求に応じて即座に Block Ack が応答される．HT-Immediate Block Ack の場合，複数の QoS データは A-MPDU に集約されて 1 度に送られ，A-MPDU の受信が完了すると即座に Block Ack が応答される．Block Ack セッションの終了は，DELBA（Delete Block Acknowledgement）リクエストにて TID を伝え，ACK レスポンスを受け取ることで成立する．

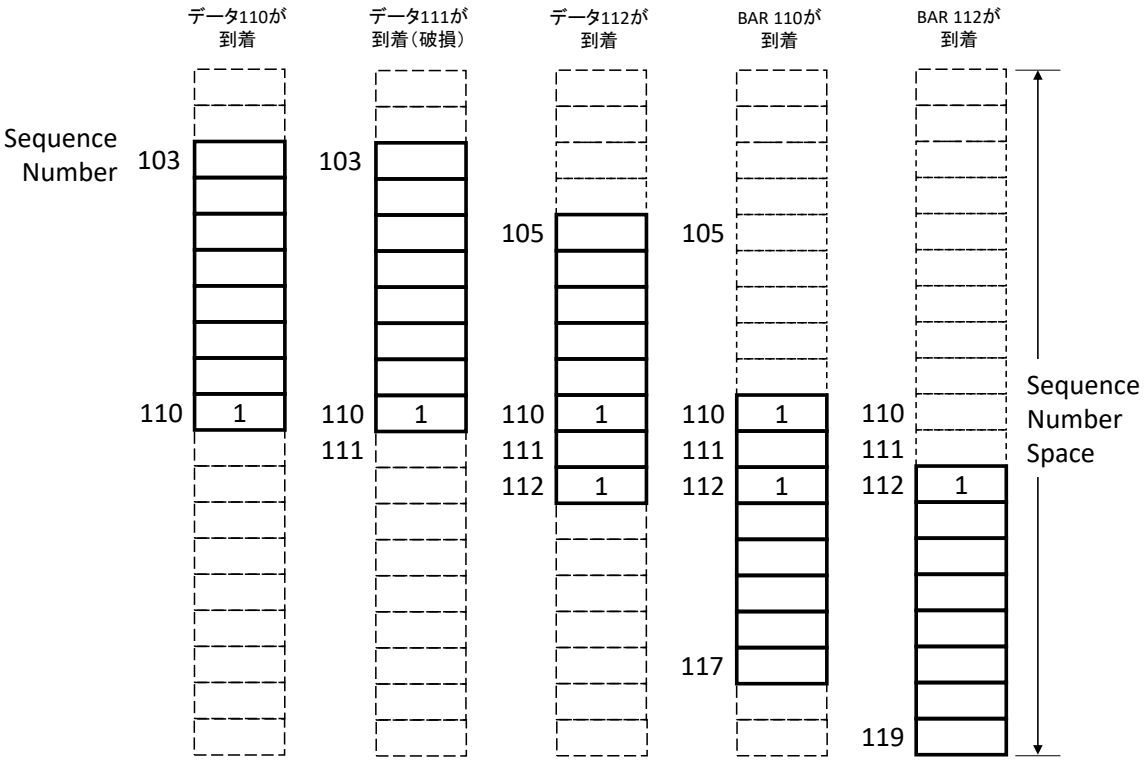


図 2.7: スコアボード・コンテクスト制御の例

図 2.7 に 802.11n のデータ受信時の処理の例としてスコアボード・コンテクスト制御を示す．縦に続く破線の枠はシーケンス番号空間を示しており，該当するシーケンス番号のデータを正しく受信したかどうかを管理している．正しく受信できた場合は 1 を表示する．実線の太線による枠は Block Ack のウィンドウを示しており，ここでは便宜上 8MSDU 分

のウィンドウサイズとする．まず，シーケンス番号 110 のデータフレームを受信した場合，もっとも新しいシーケンス番号であるので，110 を終端としたウィンドウが形成される．110 のデータは上位層（LLC）に渡される．次にシーケンス番号 111 が到着するも破損していた場合，何も行わない．次にシーケンス番号 112 を受信した場合，112 を終端としてウィンドウを移動し，103,104 はウィンドウの外となる．このとき，111 がウィンドウ内にあり受信できていないため，112 のデータは reorder バッファに残る．次に開始シーケンス番号 110 である Block Ack Request が到着した場合，Block Ack Request を起点としたウィンドウを形成する．次にシーケンス番号 111 が送信者側で期限切れとなり，新たに開始シーケンス番号 112 とした Block Ack Request が到着した場合，シーケンス番号 111 のデータはウィンドウ外となり，すでに受信した 112 のデータが上位層に引き渡される．

HT-immediate Block Ack では，A-MPDU や Block Ack Request の受信時に即時に Block Ack を応答しなければならない．SIFS 時間は 16 マイクロ秒であり，この時間内に処理する必要がある．そのためにはハードウェアにおけるメモリの取り扱いが重要になる．64MSDU を記述するスコアボードに必要なメモリサイズは 8 バイトであるが，AP の場合，このメモリは ADDBA によって確立したセッション分を用意する必要がある．多セッションに対応するためにはオンチップメモリが必要になりハードウェアが高価になる問題がある．そこで HT-immediate Block Ack では受信者に full-state と partial-state の 2 つの処理方針を定めている．前述のスコアボードを，full-state では静的メモリに割り当て，partial-state ではキャッシュメモリに割り当てる．full-state の場合，ADDBA によるセッション中，常にスコアボードはメモリ上に存在し続ける．partial-state の場合，もし，他の送信者からによるデータを受信した場合，キャッシュメモリはクリアされ，新しい受信者のスコアボードの状態が入る．例えば，Block Ack Request を受信したが，そのセッションに対するスコアボードを保持していない場合，Block Ack のビットマップはオール 0 で応答される．スコアボードの記録をホストなどに頼る方法も考えられるが，データのデコードの後，MAC を処理して適合するセッションのスコアボードを受け取り Block Ack を生成するには SIFS 時間では足りない．送信者は既に ACK を受けたシーケンス番号に関しては状態を更新しないというルールを定めることで，full-state と partial-state の両方に対応し，受信者がどちらの処理方針を用いていても良い．

このように，物理層レベルの高速化と，フレームアグリゲーションと HT-immediate

Block Ack による MAC 層レベルのオーバーヘッドの除去の両面によって，802.11n は高速化を達成している．

2.4 TCP と輻輳制御アルゴリズムの動作概要

TCP は，パケット交換ネットワークのエンドツーエンドのホスト間のデータ転送に対して信頼性を提供するトランスポート層プロトコルである．インターネットではパケットロスや重複などが発生し，また輻輳や障害の状況に応じて帯域幅，遅延，経路などが動的に変化する．そうしたネットワークの変化に対して TCP プロトコルは信頼性を提供する．TCP と同様のトランスポート層プロトコルに，UDP（User Datagram Protocol）[17] がある．UDP はアプリケーション間のパケット送受信以外は何も行わず，宛先にデータが到達したかどうかの確認も行われない．必要に応じてアプリケーションが再送や輻輳制御などを行うことになるため，UDP を採用した場合アプリケーションの通信の実装は膨らむ．一方，TCP を用いることでアプリケーションは輻輳や障害の状況などを考慮することなく通信することが可能になる．このような TCP の信頼性から，Web ブラウジング [18]，メール [19]，ファイル送受信 FTP[20] やリモートログイン [21][22] など，インターネットを経由するアプリケーションで支配的に利用されている．

2.4.1 TCP ヘッダの構造

TCP の初期仕様は Cerf と Dalal らによって RFC675[23] として発行され，1981 年には v4 に対応する RFC793[2] が発行されている．図 2.8 に TCP ヘッダフォーマットを示す．すべての TCP セグメントは 20 バイトまでの固定のヘッダを持ち，その次にオプション，データと続く．データを持たないセグメントは例えばコネクション確立や解放，もしくは確認応答の際に用いられる．個別のフィールドについて説明する．

送信元ポート，宛先ポート

ともに 16 ビット長のポート番号を含む．それぞれ送信元ホストの発ポート番号と宛先ホストの受ポート番号である．コネクションはプロトコル番号・送信元アドレス・宛先アドレス・送信元ポート番号・宛先ポート番号で識別され，これらの値を「5 タプ

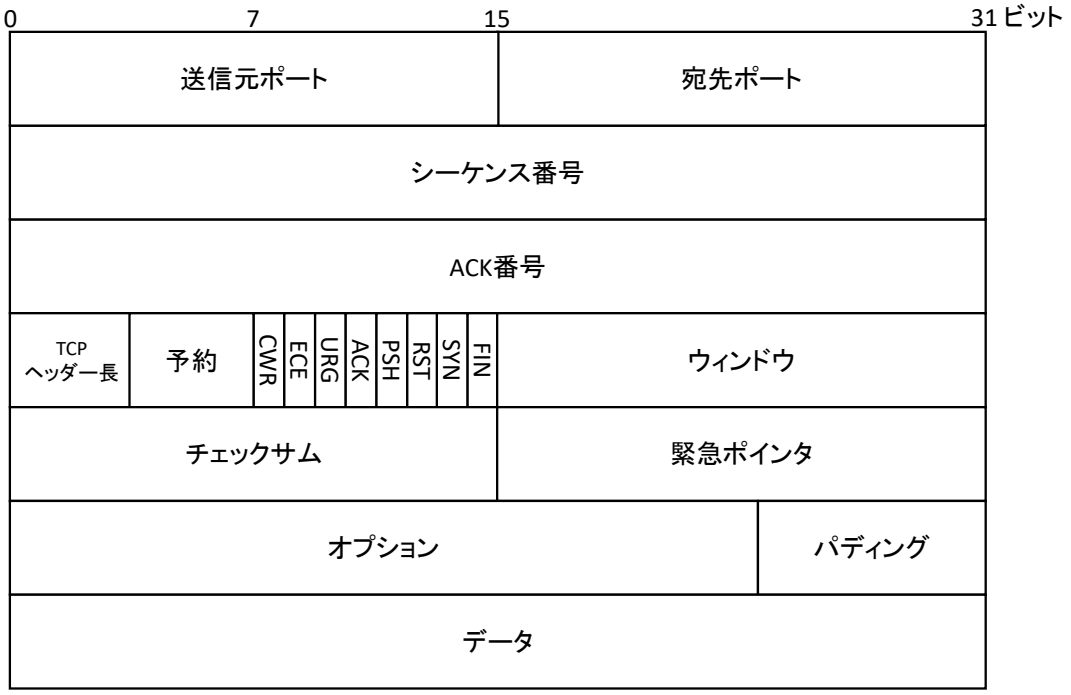


図 2.8: TCP ヘッダフォーマット

ル」と呼ぶ．ポート番号のうち, 0 から 1023 までの範囲はウェルノウン（well-known）ポート番号と呼ばれており, HTTP（Web サービス）や DNS（ドメインサービス）などの広く使用されるネットワークサービスで使用される．Unix 系の OS で well-known ポート番号を使用して発番号としてデータ送信したり，受番号としてサービス提供したりするためには通常，管理者権限を必要とする．

シーケンス番号

32 ビット長フィールドでデータの順番を識別する番号である．TCP で送信されるデータは，1 バイト単位で連続した番号を付与される．この番号を用いて同じデータを重複して受信していないか，もしくはデータの受信に抜けがないかなど，正しくデータを受信できているかどうかを判断する．32 ビット長であるため最大値である 4,294,967,295 に達すると 0 に戻る．コネクション確立時の初期シーケンス番号は，TCP 規定時は端末のクロックに応じて 1 だけ増加された数値を利用していたが，TCP シーケンス番号予測攻撃の手法 [24] が明らかになり，現在では初期値はランダム化されている．

ACK 番号

32 ビット長フィールドであり，確認応答のための番号である．確認応答は次に受信することを期待するバイトのシーケンス番号を指定する．TCP はこの ACK 番号を用いて，送信したデータを相手端末が正しく受信できたかどうかを判断する．新しいデータを送信したにも関わらず，ACK 番号が同じセグメントを何度も受信した場合，途中のデータセグメントが紛失した可能性が高い．この特徴を利用して，後に説明する TCP 実装である Tahoe では早期に再送を行う仕組みを採用している．また紛失検知の面では ACK 番号では限定されるため，より詳細に受信したシーケンス範囲の記述が行える SACK[25]（Selective Acknowledgement:RFC2018）と呼ばれる仕組みをオプションで採用している．

TCP ヘッダ長

4 ビット長フィールドであり，TCP ヘッダ長を 32 ビット単位で数えた値が入る．TCP ヘッダの長さはオプションフィールドが存在することから可変長であり，その終端を示すために必要である．ヘッダの終端からデータが始まるため，データの先頭部分を指し示すオフセットであり，TCP ヘッダをパースしてデータを取り出す際にアクセスする．TCP ヘッダの固定部分は 20 バイトなので，オプションフィールドに項目が存在しない場合 TCP ヘッダ長は最小で 5 である．それに対して 4 ビット長の最大値は 15 であり，TCP ヘッダ長としての最大値は 60 バイトである．

予約

4 ビット長のフィールドであり，未使用である．将来，TCP が拡張された際に利用するためのフィールドとなっている．TCP 策定当時（RFC793）は 6 ビット長のフィールドであったが，ECN[26]（Ecplicit Congestion Notification:RFC3168）の策定時に CWR と ECE フラグが新設され，2 ビットがそのフラグに使用されている．

制御ビット

1 ビットの制御用のフラグが 8 つ並ぶ．このビットが立つことによって，そのフラグが設定されているとみなす．

CWR，ECE CWR（Congestion Window Reduced）と ECE（ECN-Echo）は ECN による輻輳の通知に使用される．ECE はコネクション確立時に ECN に対応していることを示すためにも利用される．IP ヘッダの ECN フィールドで輻輳が

検知された場合，TCP が ACK を返す際に ECE フラグを設定して応答する．送信側の端末は ECE フラグを受信すると，自身が送信したデータセグメントが輻輳に関連していると考える．送信側はデータの送出速度を落とすと同時に CWR フラグを設定する．CWR フラグの設定された新しいデータのセグメントを受信すると，次の ACK では ECE フラグを非設定にする．

URG 緊急ポインタを使用する場合に設定される．

ACK ACK 時に設定される．0 の場合，ACK フラグは無効化される．

PSH データがプッシュされたことを示す．受信側にデータをバッファリングすることなくアプリケーションに通知するように促す．

RST コネクションを強制的にリセットする場合に用いる．もしくはコネクション確立要求を拒否するために使用される．例えば，TCP のサービスがポートにバインドされていない場合，そのポートへのコネクション確立要求は RST フラグを設定して返される．この応答の仕組みから，ホストの存在の有無やポートのバインドの有無を外部から確認して攻撃につなげる手法が存在する．

SYN コネクション確立時に設定される．TCP の通信開始時には必ず設定される．

FIN コネクションの解放時に設定される．

ウィンドウ

16 ビット長フィールドである．ウィンドウはフロー制御を実現するためのフィールドであり，受信者が受け入れることのできるバイト数，すなわち送信者が送信してもよいバイト数を通知する．このウィンドウは，ほかのウィンドウ（例えば輻輳ウィンドウ）との混在を防ぐため，広告ウィンドウと呼ぶ．例えば，この広告ウィンドウサイズが 0 だった場合，受信者は受信バッファに空きがないため，これ以上受信することはできないことを通知していることになる．受信バッファに空きができると，空きサイズを広告ウィンドウサイズに入れ，同じ ACK 番号で送信者に通知する．受信者の OS では広告ウィンドウのサイズはメモリ節約のため，固定長ではなくオートチューニングされることがあり，DRS（Dynamic Right-Sizing）[27] などの手法が用いられる．DRS が利用された OS の応答に入る広告ウィンドウサイズは通信開始時を上回って拡張されうる．

チェックサム

16 ビット長フィールドであり，TCP ヘッダ・データ・疑似ヘッダ（pseudo header）の 3 つでチェックサムを計算する．疑似ヘッダには，IP の送信元アドレス・宛先アドレス・プロトコル番号・パケット長（TCP ヘッダとデータの全長）が含まれる．TCP ヘッダとデータだけのチェックサムでは，異なる宛先への誤配してしまう場合への対応が行えないため，疑似ヘッダを含めて検証している．5 タプルのチェックサムが毎回行われることになる．チェックサムに誤りがある場合，そのセグメントは破棄される．

緊急ポインタ

16 ビット長のフィールドであり，緊急を要するデータの位置をシーケンス番号を基準にバイト数で指し示す．緊急ポインタは，相手のデータ送信を停止させたいとき，処理を切り上げたいときなどに用いられる．そのような処理を緊急ポインタなしに行う場合，別のコネクションを張る必要があるが，緊急ポインタを用いることで既存のコネクションまま要求を送ることができる．

オプション

オプションは TCP の固定ヘッダ部分 20 バイトに加えて，最大 40 バイト加えることができるヘッダ情報であり，可変長である．このオプションは当初はオプション終了・No-Op（No Operation:なにもしない）・MSS（Maximum Segment Size）の 3 つしか存在しなかった．RFC1323[28] による拡張にてウィンドウスケールやタイムスタンプなどのオプションが追加され，現在でもよく使われている．先に紹介した ACK を拡張する SACK もオプションである．

TCP ウィンドウスケールオプションは，ウィンドウフィールドを拡張するためのオプションである．オプションから 0 から 14 までの左シフトするスケールファクタを指定することによって，元々 16 ビット長であることから最大 64KB までのサイズしか値をとれなかったウィンドウ値を最大 1GB まで拡張できるようにする．大きなウィンドウ値ではなければスループット低下を招くような広帯域高遅延なネットワークの場合に有用である．

TCP タイムスタンプオプションは，TCP セグメントにタイムスタンプ情報を付与

することで，より正確な RTT 推定を可能にする．それまでの TCP 実装では，RTT の推定はデータセグメントに対応する ACK の受信によって得られた個々の RTT から平滑化された RTT を導出している．このような推定方法は再送やパケットロスによって有効ではなくなるため，タイムスタンプオプションが導入されている．送信者はセグメント送出時に自身のタイムスタンプをタイムスタンプオプションの TSval（TimeStamp Value）に格納する．またタイムスタンプオプションを含むセグメントの受信者は，応答時に TSval の値を TSecr（TimeStamp Echo Reply）に移し替えて返答する．このように応答され，受信した TSecr と自身が保持する現状のタイムスタンプと比較することによって，既存の手法より正確な RTT を推定することができる．TCP タイムスタンプオプションは送受信者双方が対応する必要があり，これを利用できる場合は RTT 推定は TCP タイムスタンプを用いたものに置き換えられ，利用できない場合は既存の RTT 推定方法が用いられる．

SACK は，ACK を拡張するオプションである [29]．オリジナルの TCP では，パケットの損失による抜けが発生した場合，ACK 番号から既に送ったデータのシーケンス番号まで確認応答を行うことができない．SACK は，この未確認のシーケンス範囲に対して，正常に受信できた範囲を応答できるようにする．これにより，送信者は損失したシーケンス番号のセグメントだけを再送するため，不要な再送を防ぐことができる．

2.4.2 スライディングウィンドウ

TCP の重要なアイディアはスライディングウィンドウである．スライディングウィンドウはフロー制御を必要とする信頼性の高い通信に用いられる．

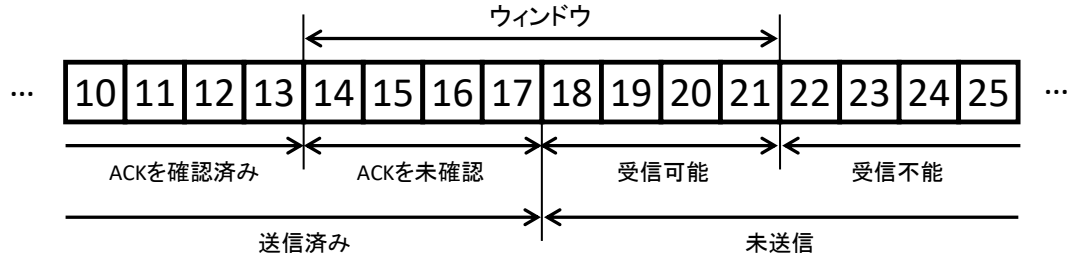


図 2.9: スライディングウィンドウ

図2.9にスライディングウィンドウのウィンドウ制御を示す．図中の枠には, TCPのシーケンス番号が記述されている．図は送信者の状態を示したものである．送信者は未確認のACK番号（SND.UNA）として14を，ウィンドウサイズ（SND.WND）として8を，次に送るシーケンス番号（SND.NXT）として18を記憶している状態である．SND.UNAより以前はACKを受信したシーケンス番号となっており，14,15と新しくACKを受信すると右にスライドする．途中で抜けがある場合は以降新しいACK番号が到着しても，抜けの地点で止まる．SND.WNDは受信者による広告ウィンドウないしは輻輳ウィンドウのどちらか小さい方を示している．SND.WNDによってネットワークに出力するセグメントを制御することができ，受信者が受信できるだけのデータを送信するフロー制御を行うことができる．SND.UNAは送信済みと未送信のシーケンス番号を切り分け, SND.UNA+SND.WND-SND.UNAによって，送信可能な残りサイズを導出することができる．

2.4.3 輻輳制御

輻輳は，ネットワークが処理できる以上のパケットが流入した際に発生する．ルータにおいて流入したパケットによる送信待ちキューの肥大化が始まり，やがて満杯になるとパケットを破棄する．当初，実装されたTCP（4.2BSDや4.3BSD）は輻輳に対して対処を行ってはいなかった．そうして1986年に深刻な輻輳崩壊（Congestion Collapse）に陥った[30][31].

輻輳に対処するためには，輻輳を検知し，輻輳に対応する必要がある．輻輳は主にルータの送信待ちキューで発生するので，検知はネットワーク層やトランスポート層の役割である．それに対して，対応を行うのはパケットを送信することで輻輳を発生させる原因となっているトランスポート層である．明示的な検知の方法としては，キューに輻輳が発生し始めるとIPパケットにECNフラグを設定して輻輳が発生していることを通知するECN（Explicit Congestion Notification）という仕組みがある．ECNに関しては2.4.1節にも記述している．もう1つの方法は暗黙的に検知する方法であり，ネットワークの挙動から輻輳の発生を判断する．主にパケットロスの発生有無によって判断される．トランスポート層による輻輳への対応は，ネットワークへのパケット流入を抑えるための速度制限である．

初期に輻輳制御が実装されたTCP Tahoeにおいて，輻輳検知の方法はパケットロスの

推定が用いられている．この輻輳検知の方針をロスベースと呼ぶ．ロスベースの利点は，明示的な通知なしに輻輳の発生を判断できる点である．その代わりにパケットロスが発生するまで検知できないので，輻輳が発生するまでネットワークに負荷を与え続けてしまう問題がある．そこで輻輳の初期の発生時点で検知を行うために RTT を用いるものも研究されており，RTT によって輻輳を判断する遅延ベースやパケットロスと RTT を組み合わせるハイブリッドベースなどが報告されている．

輻輳への TCP の対応としては速度制限となるが，輻輳の回避は最小限に抑えて，より高いスループット性能を得たいトレードオフが存在する．この速度の制御は輻輳ウィンドウと呼ばれる 1RTT の間に送信することのできるデータ量を上下することによって実現される．輻輳制御アルゴリズムは，輻輳を回避しつつも十分なスループットを得られるように，最適な輻輳ウィンドウのサイズに保とうとする．輻輳ウィンドウは，前節のスライディングウィンドウのウィンドウに適用され，受信者が通知する広告ウィンドウと送信者が保持する輻輳ウィンドウのどちらか小さい方が採用され，双方のウィンドウのサイズを超えないように配慮される．

TCP の初期の実装について説明する．1983 年，TCP の実験的な実装がなされた 4.2BSD が公開された．1986 年 6 月，TCP の安定的な実装を含む 4.3BSD が公開された．4.3BSD には輻輳ウィンドウの実装が含まれている．4.3BSD の TCP 実装の輻輳制御では，新しい ACK を受信時に輻輳ウィンドウを増加する．

$$cwnd \leftarrow 11 \times cwnd/10 \tag{2.1}$$

cwnd は輻輳ウィンドウサイズ（Congestion Window Size）である．この式によって，輻輳ウィンドウは増加し続け，やがて 65,535 バイトで制限される．なお，cwnd を下げる機構は含まれていない．こうした輻輳制御は十分ではなく，前述の輻輳崩壊を引き起こしている．有力な輻輳制御は後の 4.3BSD の Tahoe というバージョンで実装された．次節でより詳しく紹介する．

2.4.4 TCP Tahoe

1986年頃，ネットワークの成長と人気の高まりによって，激しい輻輳の問題が発生している [30]．この輻輳崩壊と呼ばれる障害によって，あるネットワーク経路のスループットは 32Kbps から 40bps へと 1,000 分の 1 まで低下したと報告されている．Jacobson らは，輻輳崩壊の原因について突き止め 4.3BSD の修正を行い [30] にて報告を行っている．

1988年6月，VAX プラットフォーム向けの 4.3BSD を Tahoe Power 6/32 プラットフォームへ移植した 4.3BSD Tahoe が公開された [32]．TCP Tahoe は 4.3BSD Tahoe に含まれていた TCP の実装の呼称である．4.3BSD から 4.3BSD-Tahoe への TCP 実装の修正作業は主に Karels が行っている．TCP Tahoe へ導入されたアルゴリズムは以下の 7 つである．

1. RTT の分散推定
2. 再送タイマの指数バックオフ
3. スロースタート
4. よりアグレッシブな ACK ポリシー
5. 輻輳時の動的なウィンドウサイズ調整（輻輳回避）
6. Karn のクランプされた再送バックオフ
7. 高速再転送

これらのアルゴリズムの中から，スロースタート・輻輳回避・高速再転送について説明する．

スロースタート

既存の TCP による輻輳は通信の開始時やパケットロスからの再開時にみられる．そこでスロースタートと呼ばれる，転送データを徐々に増加させるアルゴリズムを採用している．TCP セッション毎に cwnd（輻輳ウィンドウ）を管理し，通信開始時やパケットロスからの再開時は cwnd を 1MSS に設定する．新しいデータの ACK が到着する毎に cwnd を 1MSS 増加する．この手法では，ACK の到着時に cwnd を増加させる点が特徴である．

データを送信しACKを受信するため，データが到達するより前にACKを生成することはできない．ボトルネックを経由して到着したデータに応じてACKが応答されることになる．このACKによるクロッキングをセルフクロッキングと呼ぶ．セルフクロッキングシステムは帯域幅や遅延の変化に自動的に順応できる．

輻輳回避

ネットワークの輻輳を引き起こすのは簡単だが回復するのは困難であり，ウィンドウサイズを，小さく一定に変化させていくことが最善の方針であるとしている．輻輳回避の状態では，加算的に増加（Addaptive Increase）させる．実際にはACKを受信する毎に1/cwndのMSSを増加させていく．1RTTの間にcwnd/MSS個のデータセグメントが送られ，その個数分のACKを受信するため，1RTT毎にcwndを1MSS分増加させることになる．この方針を輻輳回避と呼ぶ．

スロースタートの挙動を制限して輻輳回避に遷移するために，ssthresh（スロースタートしきい値，Slow Start Threshold）を定める．タイムアウト時に現在のcwndの半分の値がssthreshに格納される．そのssthreshをcwndの値が超えない場合はスロースタートによる増加を，超える場合は輻輳回避による増加を行う．

一方，パケット損失によってタイムアウトが発生し輻輳が検知された場合には，輻輳状態を解消するためcwndを倍数減少（Multiple Decrease），具体的には1/2に減少させる．輻輳が継続する場合，指数的に減少することになる．

高速再転送

パケットロスの原因は，伝送中に破損しているか，輻輳によって破棄されているかのどちらかである．実際は伝送中に破損する確率は1%と小さく，ほとんどのパケットロスは輻輳に起因する．それまでのTCPはセグメントが損失した場合，回復を行うためには再送タイムアウトを待つ必要があった．セグメントが消失すると，その消失した先のシーケンスの確認応答を行うことができないため，ACK番号はロスする直前のシーケンス番号に固定される．そのため送信側ではウィンドウが満たされ，新しいデータセグメントを送信することができなくなる．再送タイムアウトが時間切れになり，再送が行われるが，この再送までにかかる時間がボトルネックとなっていた．

このような課題に対して，セグメントの消失をより早く送信側が検知するための手法としてTCP Tahoeは高速再転送（Fast Retransmit）を採用した．データセグメントの到着によってACKが行われるが，セグメントの消失があった場合は前述のとおりACK番号が固定され，重複して送信側に応答される．この重複したACKシーケンス番号を持つACKのことを重複ACKと呼ぶ．重複ACKを3つ受信すると，その直後のシーケンス番号のセグメントを消失したと判断し，TCPの再送タイムアウトが時間切れになる前に再送を行う．高速再転送が行われると，再送タイムアウトと同様にスロースタートしきい値を輻輳ウィンドウの半分の値に設定し，輻輳ウィンドウは1セグメント分に制限される．このため，セグメント消失が発生した場合は常にスロースタートから再開することになる．

2.4.5 TCP Reno

TCP Renoは1990年の4.3BSD Renoのリリース時に含まれていたTCP実装である．TCP Tahoeは再送タイムアウトの待ち時間を解消することで性能を向上させたが，再送時はスロースタート状態から始まるため，スループットが低下する問題があった．このスロースタートを行うのは過度にセグメントを送信してしまうことによってネットワークの輻輳をひどくしてしまう懸念からであり，特にタイムアウト再送の場合，送信セグメントが消失しているのかどうかが送信側からは判断できないため，輻輳ウィンドウを最小に設定することで対応している．しかしながら，重複ACKが受信される場合はデータセグメントが到着していることが前提となっており，そのデータセグメントはネットワークから消失しており，それ以上ネットワークの輻輳に影響を与えない．また同時に重複ACKがされる毎に1つずつデータセグメントが到着したと考えることができる．

TCP Renoでは，スロースタートを行わずに再送を行う高速リカバリ（Fast Recovery）を採用した．高速リカバリでは3つ目の重複ACKを受信すると高速リカバリ状態に入り，ssthreshを輻輳ウィンドウの半分に設定する．消失したセグメントを再送するが，高速再転送と異なり輻輳ウィンドウをssthresh + 3*MSSに設定する．そして，追加で重複ACKを受信するごとにCWNDをMSSずつ増加させていく．この時点で輻輳ウィンドウに空きがあるのであれば，セグメントを送信する．消失したシーケンスのデータをACKするセグメントが到着したら，高速リカバリ状態を解除し，輻輳ウィンドウをssthreshに設定する．この高速リカバリによって，輻輳ウィンドウは帯域幅遅延積に近い値で通信し続け

られるようになり，TCP として広く利用されるようになった．

2.4.6 TCP NewReno

TCP NewReno[33] は 2004 年に RFC 3782 にて規定されたアルゴリズムである．TCP Reno は 2 セグメント以上の消失があった場合，タイムアウト再送を待つ必要がある．そのため，セグメントの消失が多いネットワークでは性能が良くない問題があった．この問題を解決するために高速リカバリの手順を改善したものが NewReno である．

改善された高速リカバリの手順は以下のとおりである．“recover” という変数を用意する．重複 ACK を 3 つ受信した時点で，ACK フィールドの値と “recover” を比較して，より大きければ高速再転送を行う．小さい場合は何も行わない．高速再転送を行う場合，ssthresh はフライトサイズ（送信済みで ACK を受信していないデータサイズ）の半分か 2 * MSS のより大きいほうを設定する．その際に “recover” 変数に送信済みのシーケンス番号の最も大きい値を入れる．重複 ACK を受信する毎に MSS 分の輻輳ウィンドウを増加する．もし，輻輳ウィンドウと広告ウィンドウに空きができた場合，新しいセグメントを送信する．新しいデータに対して ACK を受信した場合，“recover” のシーケンスを含むかどうかによって挙動を変える．もし，“recover” を含んでいる場合，送信したすべてのセグメントの ACK を受信できたことになるので，TCP Reno と同じく輻輳ウィンドウを ssthresh に設定する．もし，“recover” を含まない場合，部分 ACK であることがわかる．この場合，最初の未 ACK のシーケンスのセグメントを再送する．このとき ACK された輻輳ウィンドウ分は増加する．この処理が “recover” を含む ACK が到着するまで続けられ，すべてのセグメントに ACK できた場合は輻輳ウィンドウを ssthresh に設定する．この一連の処理によって，タイムアウト再送を待つことなく，複数のセグメントの回復を行うことができるようになった．

現在，Microsoft Windows のクライアント OS では NewReno がデフォルトで使用されている．

2.5 TCP の課題と対応する実装

TCP にて様々な課題が報告され，その解決方法が報告されている．TCP の課題を一通り解説し，それに対応する輻輳制御アルゴリズムについて説明する．

パケットロスの抑制

先に説明した TCP NewReno などのロスベースの輻輳制御では，途中経路のバッファ溢れが発生してパケットロスを引き起こすまで輻輳を検知しない．そのため，過度に再送が行われスループット性能が低下する．

無線環境など伝送誤りが多いネットワークへの対応 ロスベースの輻輳制御では，パケットロスを輻輳の検知に用いる．これは伝送誤りによるパケットロスも輻輳として判断してしまうことを意味しており，本来，送信速度を下げるべきではない場面で下げてしまいスループット性能が低下する．このスループットの低下を防ぐため，無線リンクレベルでの再送が行われるが，それでも有線 LAN よりも高い確率で伝送誤りが発生する．

広帯域・高遅延のネットワークへの対応 TCP NewReno などの AIMD 方式では，輻輳制御状態に入ると，輻輳ウィンドウを 1RTT 毎に 1MSS 増加させていく．RTT によって輻輳ウィンドウの増加速度が左右され，高遅延のネットワークでは増加スピードは遅くなる．十分な送信速度に達するまで時間がかかり，スループット性能が低下する．このような広帯域・高遅延ネットワークのことを LFN（Long Fat Network）もしくは LFP（Long Fat Pipe）と呼ぶこともある．

過度なバッファリングへの対応 ロスベースの輻輳制御ではバッファ溢れを起こすまで送出速度を高める．そのため, 送信バッファから送信されるデータの速度が低速の場合，キューイング遅延が膨大になり，キューを共有する通信に影響を与える．これまでの課題は TCP のスループット性能の低下の影響を及ぼすものだったが，本課題は他の通信に遅延性能の悪化の影響を与えるものとなる．次節にて本課題を Bufferbloat 問題として詳細に解説する．

2.5.1 TCP Vegas

TCP Vegas は，RTT の増減によって輻輳を検出する遅延ベースの輻輳制御アルゴリズムである．ロスベースのアルゴリズムでは輻輳が発生した後にパケットロスが発生することを検出するが，Vegas では RTT によって輻輳の初期段階で検出することができる．

まず，Vegas では輻輳していない状態のセグメントの RTT を BaseRTT として定義する．この BaseRTT は全てのセグメントの RTT の中で最小のものを用いる．この接続が輻輳を起こしていないと仮定し，WindowSize を現状の輻輳ウィンドウの値とすると，期待されるスループットは以下の式となる．

$$Expected = WindowSize/BaseRTT \tag{2.2}$$

次に実際のスループットを計算する．セグメントが送信され，その ACK が受信できるまでの時間を測定し，かつ，その間に送信されたデータのバイト数を記録することによって計算する．

$$Actual = WindowSize/SampleRTT \tag{2.3}$$

これら期待されるスループットと実際のスループットの差分を d と定義し，d は 0 か正の値をとる．

$$d = (Expected - Actual) \times SampleRTT \tag{2.4}$$

この d と 2 つの閾値 α と β の関係によって，以下のように輻輳ウィンドウサイズの増減 w は決定される．

$$w \leftarrow \begin{cases} increase & d < \alpha \\ decrease & \beta < d \\ unchange & \alpha < d < \beta \end{cases} \tag{2.5}$$

この α と β はネットワーク上のバッファに存在する余剰のパケットの目標数を示しており，α はその最小数を，β はその最大数を示している．なお，Linux において α，β の初期値は 2, 4 となっている．

2.5.2 TCP Westwood+

TCP Westwood は，エンドツーエンドの帯域幅推定をベースとする輻輳制御アルゴリズムである．ACK を受信することでウィンドウを広げていくロスベースの特徴を持っているが，タイムアウト再送や高速再転送時に推定された帯域幅を ssthresh に設定することによって，過度に cwnd が縮小することを防ぐ．Westwood の帯域幅推定は，送信したデータセグメントに対しての応答である ACK セグメントが到着するまでの往復遅延時間（RTT）を用いる．この往復遅延時間によってデータセグメントのサイズを除算することによって，単位時間あたりに送信できる帯域幅が推定できる．

ある ACK を受信した時間を t_k，その 1 つ前に ACK を受信した時間を t_{k-1} として，t_k の ACK に対応するデータ量を d_k とする場合，帯域幅 b_k の推定は以下の式によって計算される．

$$b_k = \frac{d_k}{t_k - t_{k-1}} \tag{2.6}$$

重複 ACK を受信したときやタイムアウト再送時は $ssthresh$ は以下の式によって計算される．推定された帯域幅を BWE，TCP の最大セグメントサイズを MSS，その経路で計測された最小の RTT を RTT_{min} とする．

$$ssthresh = BWE \times RTT_{min}/MSS \tag{2.7}$$

ボトルネックリンクのバッファが空となるのは $RTT_{min} \times BWE$ となるときであるとし，輻輳回避状態に入るときにはバッファを一旦，空にするように動作する．タイムアウト再送の場合はスロースタート状態となり，その経路のボトルネックのバッファにセグメントが滞留し始める寸前まで指数的に増加させる．例えば Reno の場合，$ssthresh = cwnd/2$ と設定するが，伝送誤りの多い無線経路を含む場合，輻輳と判断され過度に輻輳ウィンドウを縮小させることがあったが，Westwood では帯域幅推定によって必要以上に縮小させることがなくなった．

このような特性を持っているが，セグメントロスがあるまで輻輳ウィンドウを縮小することはなく，その点では他のロスベースと同様である．

2.5.3 CUBIC TCP

CUBIC TCP は高速ネットワークに最適化されたロスベースの輻輳制御アルゴリズムである．CUBIC TCP は BIC TCP を拡張したものであり，TCP フレンドリネスと RTT フェアネスの改良を行っている．

BIC TCP は NewReno などのように輻輳回避状態において 1RTT に 1MSS 増加する手法をとらず Binary search increase と呼ばれる特殊な手法を採用している．この手法では，高速リカバリ直前のウィンドウサイズを W_{max}，直後のウィンドウサイズを W_{min} として，輻輳ウィンドウはその中間値を設定する．セグメント消失がなければ W_{min} は現状の輻輳ウィンドウの値に設定され，次回の計算時に使用される．前回，セグメント消失が発生した値の近くで長く留まろうとすることで，セグメント消失の頻度を減らし，性能を落とさずに通信することを目的としている．

CUBIC TCP では，BIC TCP の手法は低遅延ネットワークや低帯域のネットワークに対してアグレッシブであり，また複数のウィンドウ増加関数を持つために解析が複雑になるとして，単純な計算式で制御できるアルゴリズムを提案している．この計算式で求まる輻輳ウィンドウの挙動は BIC TCP と似ているが，経過時間を用いている点が大きな違いである．

輻輳ウィンドウサイズ（$cwnd$）は以下の計算によって求まる．

$$cwnd = C(t - K)^3 + W_{max} \tag{2.8}$$

$$K = \sqrt[3]{W_{max}\beta/C} \tag{2.9}$$

C はスケーリングファクター，t は最後のウィンドウ減少（セグメント消失）からの経過時間，W_{max} は最後のウィンドウ減少直前の輻輳ウィンドウサイズである．β は定数倍の減少率である．輻輳ウィンドウは 3 次関数にて求まり，K は 3 次関数の変曲点までにかかる時間を示している．C には 0.4 が，β には 0.2 がよく使用される．

CUBIC TCP は Linux 2.6.19 より NewReno と置き換わり，デフォルトの輻輳制御となっている．

2.6 Bufferbloat 問題の概要

Bufferbloat は古くて新しい問題である。パケット交換ネットワークの蓄積型の特徴，および TCP の輻輳崩壊への対応が問題をはらんでいる．

回線交換ネットワークは回線を占有する。そのため帯域は保証されていても、空き回線の不足によって接続不能となることがある。また通信の有無に関わらず回線を占有してしまう。パケット交換ネットワークは通信の単位をパケットという単位に分割し、交換機はひとまずバッファに蓄積してから送信する。同じ宛先に送るパケットが存在する場合は後続のパケットをバッファに待機させ、先行するパケットの送信完了を待ってから送信を行うことで対応できる。この工夫によって複数の端末で伝送路を共有できるようにし，回線交換ネットワークに比べて高い利用効率を実現している。また異なる接続速度のネットワーク間の相互接続も容易に可能である。パケット交換ネットワークは広がり、インターネットとして多くの人が参加できるネットワークとなる。

他方、パケット交換は蓄積して送信する性質上、輻輳時には遅延が発生する。より激しく輻輳が発生すれば輻輳崩壊を引き起こす。前節で紹介した 1986 年の輻輳崩壊は Jacobson らによる TCP への輻輳制御の実装の貢献によって解消された。この時に採用されたのがロスベースの輻輳制御である。Jacobson らの TCP 実装を含む BSD はインターネットを取り扱うための参照実装として Microsoft の Windows や Apple の MacOS など多くの OS により参照されており，結果として NewReno のようなロスベースの輻輳制御アルゴリズムが PC やスマートフォンにおいても支配的に利用されるようになっている。

パケット交換ネットワークで考慮すべき遅延の種類は 4 つ存在する。ヘッダを処理するプロセス遅延、送信までバッファで待機するキューイング遅延、データを符号化するシリアル化遅延、データの物理的な伝達にかかる伝搬遅延。遅延はこの数十年の発展によって短縮されている。プロセス遅延は CPU の処理能力の向上によって、キューイング遅延やシリアル化遅延は回線の広帯域化によって、伝搬遅延はルーティングプロトコルによる経路の最適化や CDN などのように地理的に近い場所にデータを置くことによって、それぞれ短縮している。近年、輻輳の問題が再度発生しており、先に説明した 4 つの遅延の中でキューイング遅延の増加が現象として認識されている。この問題は Bufferbloat と呼ばれており、2012 年に報告 [8] され、議論されている。

図 2.10 に Bufferbloat 問題のイメージを示す．図の中央に，ボトルネックリンクの送信

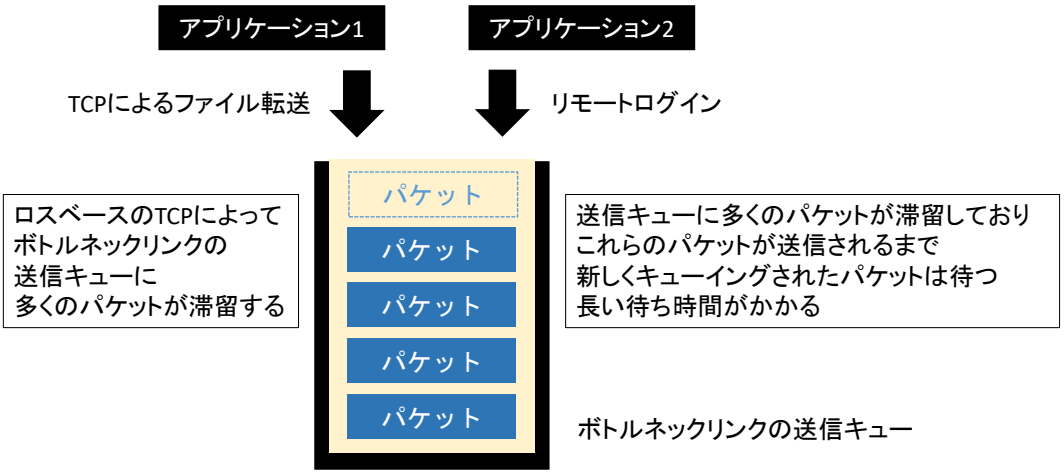


図 2.10: Bufferbloat 問題のイメージ

キューがある．このボトルネックリンクを経由するトラフィックが 2 つのアプリケーションから与えられる．片方は TCP によるファイル転送であり大量の TCP セグメントが送信される．このトラフィックがボトルネックリンクを TCP セグメントで溢れさせる．もう一方のトラフィックはリモートログインである．リモートログインのトラフィックは，ファイル転送のトラフィックと送信キューを共有する．そうした場合，既に送信キューはファイル転送の TCP セグメントが多く滞留しているので，それらのセグメントが送信されるまで待機する．これが Bufferbloat 問題のポイントである．すなわち，リモートログインなどの遅延性能が強く影響を与えるアプリケーションでは，他の大量データ転送のアプリケーションにより送信キューにおける遅延が増大し快適性を損なってしまう．

キューイング遅延は、パケットが到達した時点の送信速度と既にキューに待機しているデータ量（パケット数）の積算によって計算できる。輻輳がひどくなればキュー待機パケットの数は増加するが、以前はバッファのサイズが小さく，そのサイズに待ち行列は制限されるため問題とはなっていない。近年、このバッファのサイズが肥大化してきており，それに従ってキューイング遅延が増加する現象が発生している。

バッファサイズが肥大化する背景には、広帯域高遅延なネットワークへの対応が考えられる。このネットワークの帯域幅遅延積は大きく、同様に TCP では大きなバッファを必要とする。バッファの容量が十分ではない場合、バースト的なトラフィックが到達することによってパケットロスが発生してしまい、TCP の輻輳制御アルゴリズムによっては、十分な実効速度が得られない場合がある。ネットワークを利用する顧客の間では実効スループットはネットワーク品質における分かりやすい指標であり，スループットの低下

はしばしば問題とされやすく，インターネット・サービス・プロバイダ（ISP）としては避けたい話題である。またメモリは安価になっており、同じ価格で大容量のバッファを積み込めるようになっている。このような事情からバッファサイズは肥大化している。このような大きなバッファサイズを持ったモデムが広帯域ではなく狭帯域ネットワークで用いられ、バッファサイズの調整がなされない場合に Bufferbloat 問題が発生する。

さらに前述した TCP のロスベースの輻輳制御アルゴリズムも Bufferbloat を発生させる要因の一つである。ロスベースのアルゴリズムは、輻輳が発生するまで送信速度の制限を行わず、バッファをあふれさせるまで送信し続ける。これによってバッファサイズの最大までキューイング遅延は広がることになる．例えば TCP Vegas などの遅延ベースのプロトコルであれば，遅延の増加によって輻輳の発生を検知して送信速度の制限を行うため，ロスベースの輻輳制御ほどひどい遅延は発生しないと考えられる．

Bufferbloat 問題への対応は，有線ネットワークと無線ネットワークによって容易さが異なる．有線ネットワークの場合、対処は比較的容易である。ネットワークの帯域に合わせてバッファサイズの調整を行えばよい。有線ネットワークの場合、帯域幅の変動は小さく安定していることが多い。狭帯域であればバッファサイズを小さく設定することで実効速度を犠牲にせずに遅延時間を抑制することができる。

無線ネットワークにおける Bufferbloat への対処は有線に比べてより困難である。無線はその性質上、帯域のダイナミックレンジは大きく、環境に応じて帯域が変化するからである。例えば LTE は実効速度では数百 kbps 程度から数百 Mbps まで、無線 LAN は数 Mbps から数百 Mbps までの広大なダイナミックレンジがある。この可動範囲の中、無線ネットワークは無線状況に応じてデータレートを使い分ける。このように帯域幅を固定であると仮定して適切なバッファサイズを求めることはできず、考えなくバッファサイズを小さく設定すれば逆に実効速度が低下してしまう。

2.7 Bufferbloat 問題に関連する研究

Bufferbloat の問題は前節のとおり，過度なバッファリングにある．TCP のロスベースの輻輳制御アルゴリズムが輻輳ウィンドウを増加させ，ネットワーク経路上のバッファをあふれさせることにある．このバッファあふれを防ぐための研究が行われている．

この研究の方向性は以下に分類される．

1. バッファサイズを適切に設定する

2. パケットロスを意図的に発生させる

3. TCP の送信を抑制する

4. 受信ウィンドウサイズを意図的に小さくする

これらの研究では，ネットワーク環境に合わせた手動設定を必要とせずに自動で調整されること，加えてスループットを犠牲にすることなく遅延性能の低下を抑制することが求められている．バッファサイズの固定的な設定は可変帯域幅を持つ回線には不向きであり，可変帯域幅に対応する 2. 以下の手法について紹介する．

2.7.1 CoDel

CoDel はアクティブキュー管理機構（AQM）の一種である．それまで，AQM として RED や ECN（Expilicit Congestion Notification）などが議論されてきたが，RED などのアルゴリズムは手動による調整を必要としていたが，CoDel は手動による設定をすることなく動作することができるアルゴリズムとしている．

CoDel では，送信キューにおけるパケットの滞在時間を監視し，その他のキューの指標は用いない．パケット滞在時間は，送信キューにエンキューされたときにタイムスタンプを保存し，デキューされたときに算出される．

インターバル時間の最小のパケット滞留時間を算出する．インターバル時間の最後のパケットがデキューされた時点において，その間の最小パケット滞留時間が目標時間（5 ミリ秒）を超える場合，次の 1 パケットは破棄され，インターバル時間はより短い値に設定される．もし，デキューされたパケットが最小パケット滞留時間が目標時間を下回った場合や MTU（Maximum Transmission Unit）よりも小さいサイズであった場合は破棄は行われなず，次のパケットがデキューされるときにインターバル時間は初期値にリセットされる．目標時間とインターバル時間はそれぞれ 5 ミリ秒と 100 ミリ秒が選択されている．インターバル時間 100 ミリの設定は，RTT10 ミリ秒から 1 秒の範囲で良好に機能するとされている．この目標時間は 5 ミリ秒より小さい値では特定の状況によってリンクの

使用率が落ちてしまい，5 ミリ秒より大きい値では利用率の改善はほとんどなかったとしている．100 ミリ秒のインターバル時間については，RTT と関連しており，10 ミリ秒から 1 秒までの RTT の範囲で良好に機能するとしている．

$$drop\_next = t + \frac{interval}{\sqrt{count}} \tag{2.10}$$

次のインターバル時間の決定を行う式を示す．本式は，最小パケット滞留時間が目標を超え，破棄が行われた際に利用される．$interval$ は設定されたインターバル時間（100 ミリ秒），$count$ には（CoDel によって破棄が行われた回数+1）が入る．これによって，破棄が行われるほど，次のインターバル時間が短くなる．本式は，TCP スループットに対して線形変化を得るためのドロップ率の既知の関係 [34] を用いており，ドロップ数の平方根に反比例するように設計されている．

RED と CoDel をシミュレーションで比較した結果, 10Mbps 以上の場合, RED に比べて CoDel の方が高い利用率を示しており，RED は過度にパケット破棄を行っている可能性があることを指摘している．また無線 LAN のように動的に帯域幅が変化するネットワークにおけるシミュレーションも行っており，その結果においても CoDel は RED と同等の遅延時間でありながら，高いリンク利用率を示しており，リンクの帯域幅変化にすばやく対応できるとしている．

2.7.2 TCP small queues

TCP small queues は TCP から送出されるセグメントをキューイング状況に応じて制限する方式であり，Linux にて TCP で実装されている．通常，TCP 送信ソケットバッファに格納されたアプリケーションのデータは，輻輳ウィンドウや受信側が広告したウィンドウで規定されるサイズまで TCP セグメントとして下位層に出力される．その TCP から出力され，下位層に滞留しているデータサイズを計算し，指定量（デフォルトでは 128KB）以上の場合, TCP は新たにデータセグメントを生成して下位層に渡すことなく送信ソケットバッファ上でキューイングを続ける．下位層にて送信が行われ，滞留しているパケットサイズが減少して指定量までに空きができた場合，送信ソケットバッファ上にある未送信データを下位層に渡す．

具体的には，TCP で作成されたセグメントの構造体の確保と解放の処理をフックする．構造体の確保時には滞留しているデータサイズに加算を行い，解放時にはデータサイズの減算を行い，TCP 送信キューの起動を予約する．この構造体はデバイスで送出されるまで紐づけられたまま保持され，送信が終わったり破棄された時点で解放される．パケットスケジューラやネットワークインタフェースのデバイスドライバまで，その構造体が送信されるまでに対応するすべてのバッファを対象とする．そのため，パケットスケジューラとデバイスドライバにおいて滞留するデータサイズが指定量を超えた場合，送信ソケットバッファに未送信データが蓄積し，アプリケーションから TCP へのデータ転送が停止する．これによって，TCP セグメントを破棄することなく送信端末内の Bufferbloat 問題を抑えることができる．

しかしながら，TCP small queues は設計上，ボトルネックが同一の端末でなければ，その効果を発揮することができない．これは，ボトルネックが次のホップの端末であった場合，その端末に送信完了した時点でＴＣＰ構造体は解放されており，パケットが滞留していることを知ることができないことを意味する．例えば，無線 LAN の AP を介した低速なデータレートによるダウンリンク方向の通信の場合，AP にてパケットが滞留するが対応を行うことはできない．

2.7.3 DRWA（Dynamic Receive Window Adjustment）

DRWA[35] は受信者の通知する受信ウィンドウを用いて TCP の送信速度を制限する方式である．2.4.1 節のウィンドウにて紹介した通り，TCP では受信者が受け入れられるウィンドウサイズを送信者に通知することでフロー制御を実現しており，その動的なサイズの決定に DRS（Dynamic Right-Sizing）が用いられている．DRS はまず RTT の推定を行い，次にその推定した RTT 期間に受信したデータ量から送信者の輻輳ウィンドウサイズを推定し，その輻輳ウィンドウサイズを制限しないように受信ウィンドウサイズを決定する．これによって受信ウィンドウサイズによってスループットが抑制することはないが，受信ウィンドウサイズは一方的に拡大されるのみである．

DRWA のアイディアは，この DRS を Bufferbloat 問題に適用するため，推定した RTT を拡大だけではなく縮小も含めた双方向に対応する点にある．DRWA は DRS と同様の方法で RTT の推定を行い，その推定 RTT 間に受信したデータ量の平滑化を行い，送信者の

持つ輻輳ウィンドウサイズの推定を行う．そして以下の式で受信ウィンドウ（$rwnd$）を決定する．

$$rwnd \leftarrow \lambda \times \frac{RTT_{min}}{RTT_{est}} \times cwnd_{est} \tag{2.11}$$

λはチューニングパラメータ（1 より大きく設定する），RTT_{min} は最小の RTT，RTT_{est} は推定された RTT，$cwnd_{est}$ は推定された輻輳ウィンドウである．推定 RTT が大きくなる場合，TCP は過度にデータを送信しているとして，受信ウィンドウサイズによって徐々に送信量を制限していく．推定 RTT は $RTT_{min} \times \lambda$ の値に留まろうとする．[35] では 3G/4G セルラーネットワークに適用され，有効に機能したとされている．

本方式は，TCP の受信機構に組み込む実装であり，受信者側が対応する必要がある．

2.8 802.11n における Bufferbloat 問題

本章では，IEEE 802.11n による無線 LAN の速度改善，TCP の性能改善，そして古くて新しい問題である Bufferbloat 問題について議論してきた．Bufferbloat 問題は過度なバッファサイズやボトルネックリンク，そして TCP の採用によって発生する現象であり，この問題は 802.11n を用いて構成されたネットワークにおいても発生する可能性は高い．802.11n は多くの実装では，2.4GHz 帯では 1Mbps から 300Mbps まで，5GHz 帯では 6Mbps から 300Mbps まで通信のデータレートが変化するダイナミックレンジを持つ．ネットワークは大きなバッファを持たなければ TCP スループットの低下を招いてしまうため，最大 300Mbps に見合う十分なサイズのバッファを用意している．そして，無線状況が悪化した場合，例えば 5GHz 帯では 6Mbps まで低下し，それによってバッファに滞留したパケットは Bufferbloat を引き起こすだろうと予想される．

さらに 802.11n で導入された HT-Immediate Block Ack も状況を悪化させる要因である．それまでの無線 LAN の再送は，データ送信直後に行われるためバースト的なノイズに弱い特徴があり，さらにフレームの順序通りに行う必要があるため VoIP などの遅延性能にセンシティブなアプリケーションを考慮した再送回数に設定される必要があった．HT-Immediate Block Ack によって，送信者は送信に失敗したデータフレームをある期間保持して他のデータフレームの送信後に再度再送が行うことができる強力な再送となっている．これによって伝送誤りは上位層へ伝わることは減少し，既存の無線 LAN に比べ

て TCP は輻輳ウィンドウをパケットロスによって落とさなくなるものと考えられる．さらに，他のフレームを先行させることができるようになったため遅延性能を要求するアプリケーションにそれほど影響を与えず，Immediate Block Ack による再送の回数そのものも以前に比べて大きく設定することが可能になっている．

2.9 本研究の位置づけ

本研究では，IEEE 802.11n 無線 LAN 上のアップリンク TCP 通信における Bufferbloat 問題に関して詳細に検討を行い，スループット性能を低下させることなく遅延性能の悪化を抑制することを目的とする．

一般的なインターネットの利用者は，Web ブラウジング，メール，ファイル転送などの作業を行うが，これらの通信に用いられるプロトコルは TCP である．先に述べた通り，TCP を用いることでアプリケーションソフトウェアは個々に輻輳制御を持つ必要がなく，OS の共通のネットワークスタックの制御に任せることになる．OS の TCP の輻輳制御アルゴリズムは現時点においてもロスベース方式が多く利用されている．そのため，途中経路で輻輳に起因するパケットロスが発生するまで，送信速度を拡大し続け，バッファをあふれさせる．前節で述べた通り，802.11n 無線 LAN ネットワークにおいても Bufferbloat 問題は発生しうると考えられる．本研究では，こうした無線 LAN を介したイントラネット利用とインターネット利用の双方を考慮し，ネットワークに疑似的に遅延を追加した場合の実験検討を行い，高遅延環境における性能の評価も行っていく．

有線 LAN やセルラーネットワークにおける Bufferbloat 問題に関しては，すでに [8] や [35] などにおいて議論が行われている．しかしながら無線 LAN における Bufferbloat 問題の発生は [8] にて示唆されているがその議論は十分ではなく，802.11n 無線 LAN における詳細な検討はなされていない．本研究ではまず，3 章にて 802.11n 無線 LAN ネットワークを構成し，そのネットワークにて実際に Bufferbloat 問題が発生するかどうか，どのような性能低下を及ぼすのか，802.11n 特有の事項について詳細に検討を行う．

続けて 802.11n における Bufferbloat 問題の解決方式について検討を行い，新手法の提案を行う．802.11n における Bufferbloat 問題の解決には，CoDel や TCP small queues などの先行研究の手法の適用が考えられるが，本研究では，802.11n 特有の情報を用いるア

プローチを採用する．具体的にはIEEE 802.11n無線LANで導入された強力なフレーム再送の機能をあえて抑制することで意図的にセグメントロスを発生させる方式を用いる．さらにこの方式を送信者である無線LAN端末に実装する方法と，受信者であるアクセスポイントに実装する方法を提案する．表2.1にて，Bufferbloat問題に対応する各アプローチの実装箇所を比較する．

表 2.1: Bufferbloat問題に対するアプローチの実装箇所の一覧

アプローチ	実装箇所	
CoDel	ボトルネックリンクのキュー（AQM）	
TCP small queues	ボトルネックリンクを持つ送信者のTCP	
TCP Vegas	送信者のTCP	
DWRA	受信者のTCP	
提案方式	送信者のMAC	受信者のMAC

第3章　802.11n無線LAN上のBufferbloat問題に関する実験的検討

第 2 章において，IEEE 802.11n のスループット向上のための工夫，Bufferbloat 問題について議論した.802.11n では高速化のための工夫によって，その特性はそれまでの無線 LAN とは異なっている．例えば，802.11n によってデータレートの取りうるダイナミックレンジは現存する実装では最大 300Mbps まで広がる．回線が広帯域化した場合，TCP の輻輳制御アルゴリズムによってはウィンドウが枯渇してしまいスループットが低下してしまう可能性が考えらえる．もしくは通信品質の低下によってデータレートが低下した場合，TCP のウィンドウが過剰であることによって，過度なトラフィックがネットワークに送出されることによって輻輳やパケットロスを引き起こしてしまう．さらに 802.11n から導入されたアグリゲーション（A-MPDU）や HT-Immediate Block Ack などの工夫によって，2.8 項で説明した通り，Bufferbloat が発生することが予想される．そこで 802.11n ネットワークにおける TCP 通信の特性を調査するために，802.11n の AP を経由するネットワーク環境を構築し，端末と AP の距離を変えて，TCP によるデータ転送の実験的検討を行った．本章では，その結果を示す．

3.1　実験条件

ネットワーク構成の概要を図 3.1 に示す．5GHz 帯を用いた IEEE 802.11n を使用する端末と AP を接続し，AP とサーバは 1Gbps のイーサネットで接続する．AP はブリッジとして動作している．端末の詳細な仕様を表 3.1 に示す．また，AP はバッファロー社の市販製品（型番 WZR-HP-AG300H）を用いている．この AP の最大通信速度は 300Mbps

である．TCP の輻輳制御には CUBIC TCP を用いる．

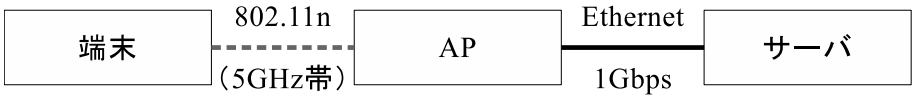


図 3.1: ネットワーク構成の概要

表 3.1: 端末の詳細仕様

Linux カーネル	2.6.38（self build）
Linux ディストリビューション	Ubuntu 12.04 LTS server
TCP 輻輳制御	CUBIC
メーカ名／型番	Lenovo ThinkPad X61
無線 LAN カード	NEC Aterm WL300NC
無線 LAN ドライバ	ath9k[36]

端末の無線 LAN デバイスドライバ ath9k で AP に対して使用できるレートの一覧を表 3.2 に示す．端末の無線 LAN デバイスと AP は，2 ストリームまでの空間多重化と 2 つのチャンネルを束ねるチャネルボンディング（HT40）に対応しており最大で 300Mbps で送受信することができる．ath9k ではその中から表 3.2 に示される 12 段階のデータレートの使用に限定している．108Mbps 以上は 2 ストリームで，未満は 1 ストリームで送信する．300Mbps のみガードインターバルは 400ns であり，それ以外は 800ns を使用する．6.5Mbps のみはチャネルボンディングなしの 20MHz で，それ以外は 40MHz のバンド幅で送信する．

表 3.2: 使用可能なデータレート

6.5	13.5	27.0	40.5	54	81
108	162	216	243	270	300

（単位 Mbps）

本実験では，ネットワーク測定ツール iperf（2.0.5）を用いて端末からサーバへ TCP で 120 秒間のデータ転送を行う．転送と並列して端末上で，tcpdump によるパケットキャプチャ，tcpprobe モジュール [37] による TCP コネクション情報（輻輳ウィンドウの値など）の取得を行う．また，より正確に無線フレーム送受信の様子を調べるため，無線 LAN ドライバに対してデータフレーム送受信のログを出力するように改変を行い，フレーム毎の

データレート，送信待ちキューの長さ，再送の状況などの情報を取得する．さらに，無線 LAN インタフェースのキューの長さの取得を行う．

実験機器の配置および移動の様子を図 3.2 に示す．本実験は木造 2 階建て家屋にて行い，AP は 2 階に設置する．実験中に，AP から近い地点と遠い地点との間で，移動を行う．近い地点とは，AP から 1.2m 程度の近接した場所，遠い地点とは，10m 程度離れた，階段を挟んだ 1 階の奥の部屋の中である．実験中は次のように移動を行う．端末は，遠い地点で通信を開始し，そのまま 30 秒間静止する．続いて，15 秒程度かけて近い地点へ移動し，そのまま 30 秒間，静止する．さらに，近い地点から遠い地点へ，15 秒程度で移動し，遠い地点で 30 秒間，静止する．

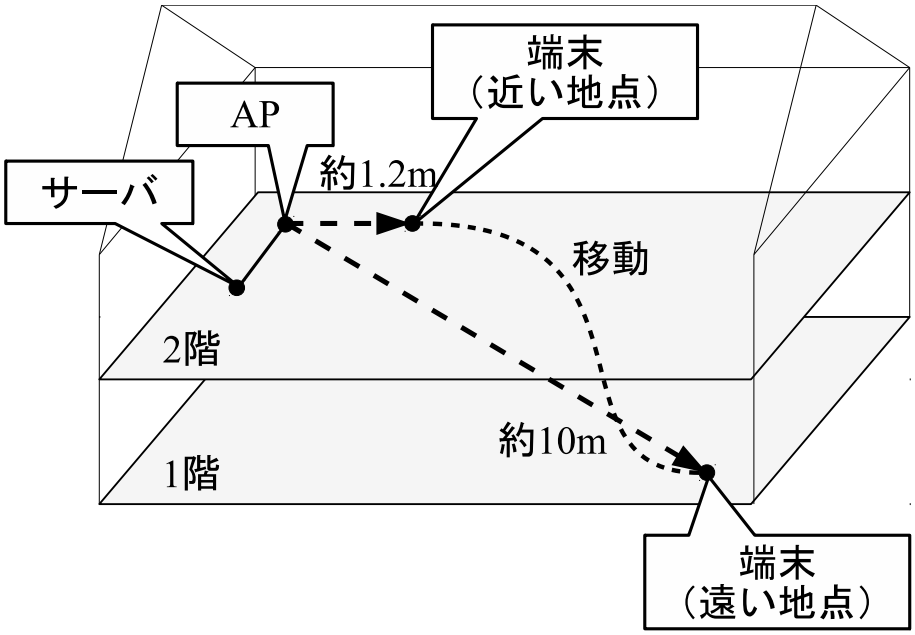


図 3.2: 機器の配置と移動の様子

3.2 実験結果と考察

以上のような条件の実験を 50 回程度行った．結果はいずれも同様の傾向を示した．実験の 1 つについて，その結果を詳細に述べる．

図 3.3 に，送信側でキャプチャした TCP セグメントのシーケンス番号から計算した 0.5 秒ごとのスループットの時間変化を示す．最初の AP から遠い地点では 3〜5Mbps 程度，移動中は徐々に上昇し，AP に近い地点では 80〜100Mbps 程度のスループットとなってい

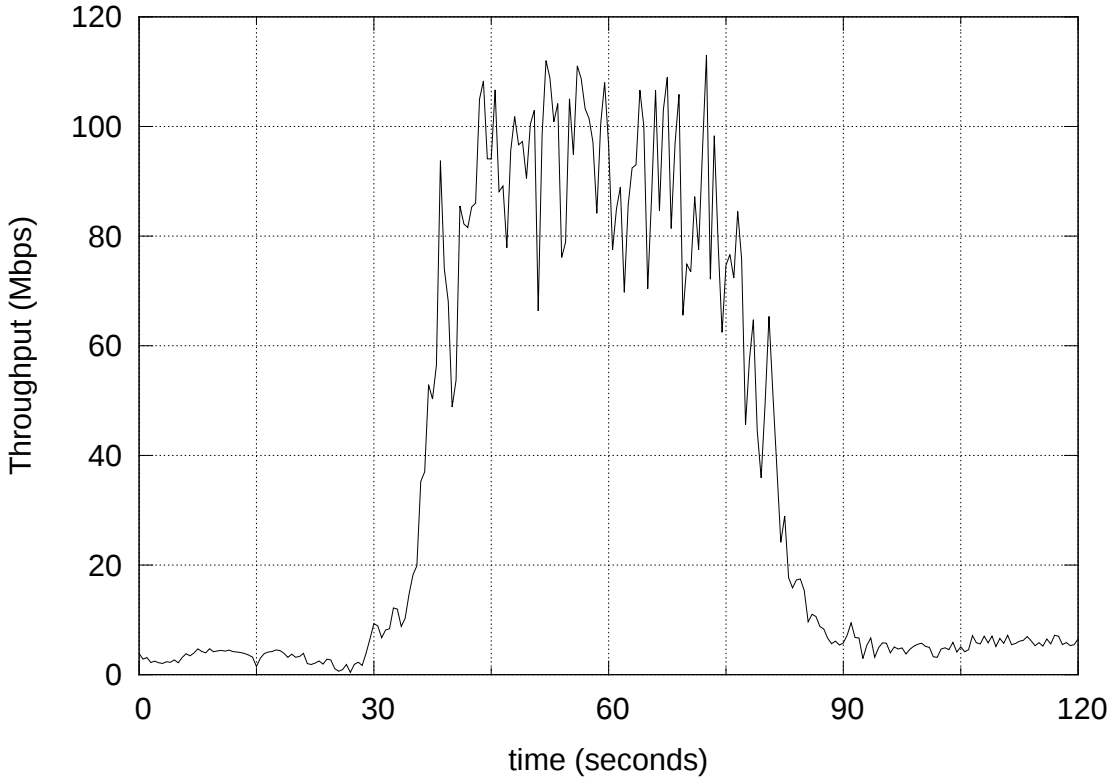


図 3.3: 0.5 秒単位の平均スループットの時間変化

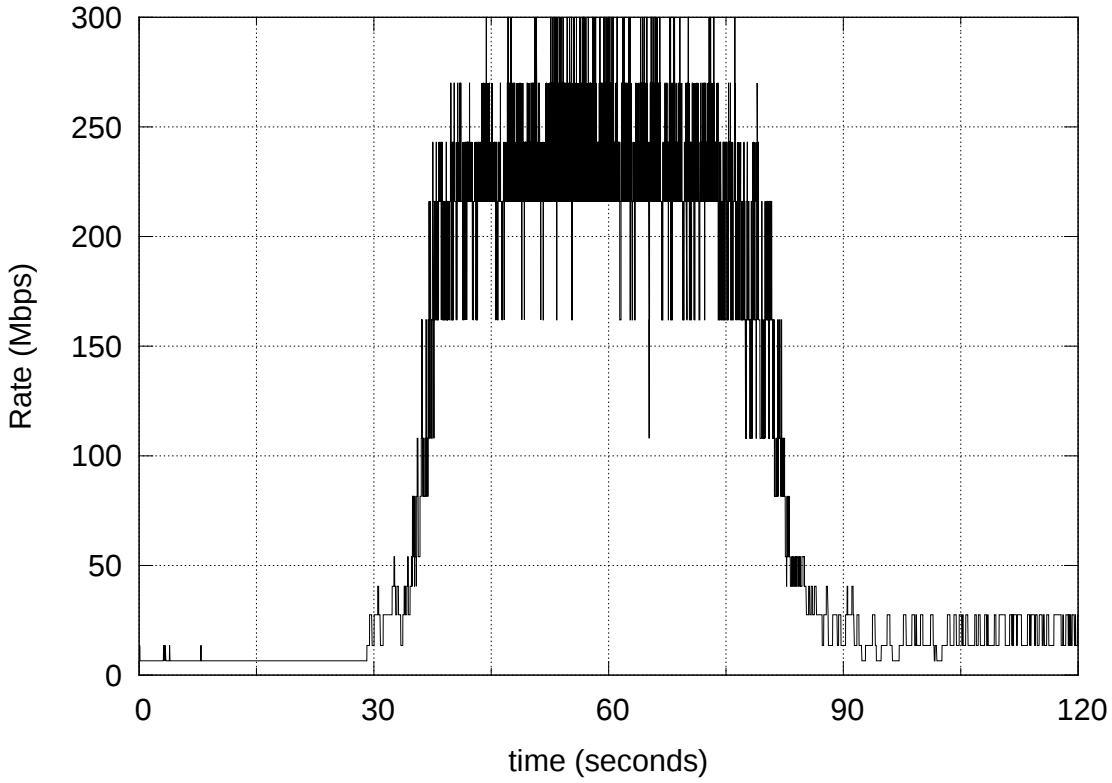


図 3.4: 無線データレートの時間変化

る．再度，移動中には徐々に下降し，遠い地点でのスループットは 4〜6Mbps 程度となった．二度の遠い地点でのスループットの違いは，設置場所の微妙な違いによって無線環境が変化したためと考えられる．図 3.4 に，無線 LAN ドライバから収集した選択されたデータレートの時間変化を示す．遠い地点では 6.5Mbps などの低いレートが，近い地点では 216Mbps 以上の高いデータレートが選択されており，得られるスループットに大きな影響を与えている．

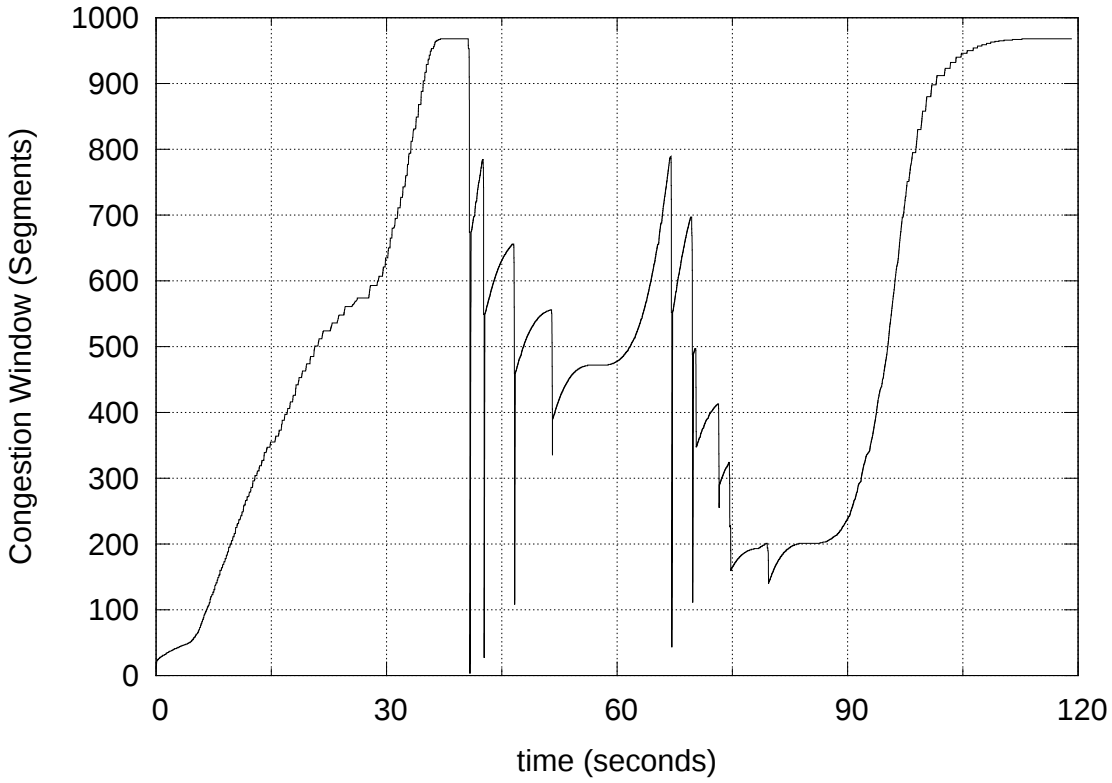


図 3.5: TCP 輻輳ウィンドウの時間変化

図 3.5 に tcpprobe により取得した TCP の輻輳ウィンドウサイズを示す．ウィンドウサイズはセグメント数で表している．今回のネットワークの帯域幅遅延積は，仮に最小 RTT が 1 ミリ秒，最大の帯域幅を 300Mbps とした場合，25 セグメント分（37.5kB 程度）である．図 3.5 で示されている輻輳ウィンドウの値は全体を通して帯域幅遅延積をはるかに超える値となっており，このことから，端末の移動を行っても輻輳ウィンドウに起因するスループット低下はなかったと考えられる．しかし，高遅延のネットワークでは必要とされる帯域幅遅延積はより大きいので，ウィンドウの欠乏によるスループットの低下は起こりうると考えられ，さらなる検証が必要である．

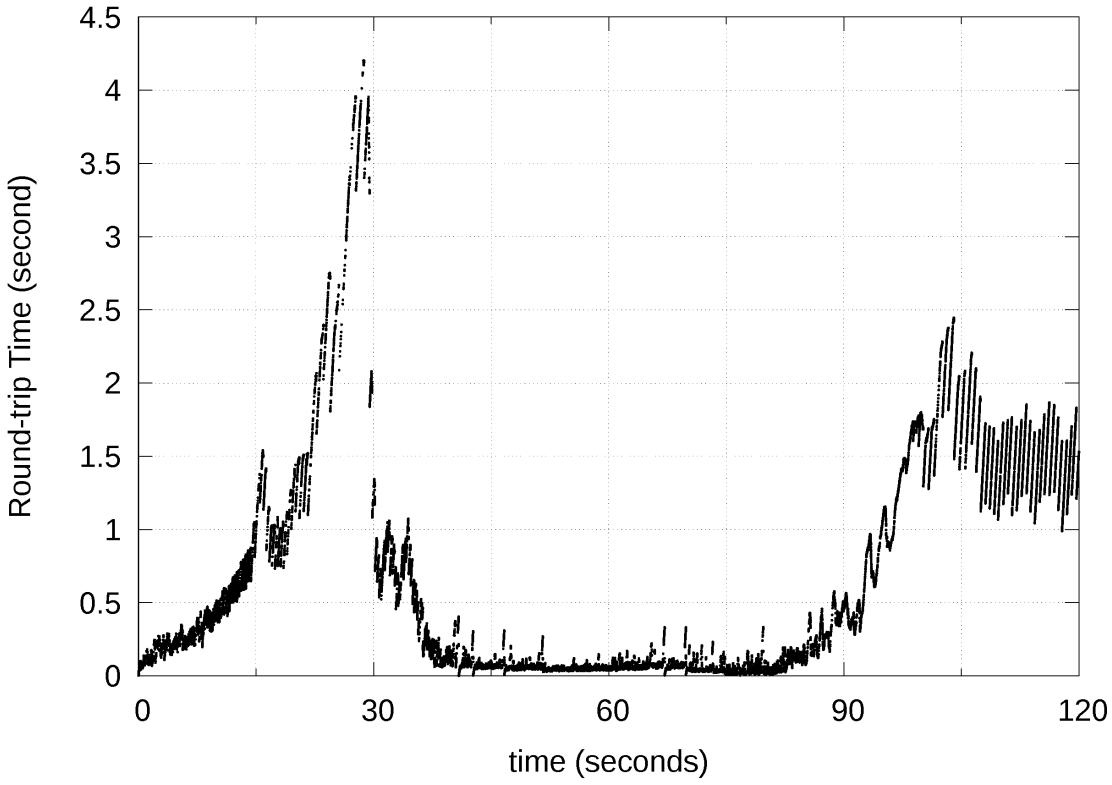


図 3.6: TCP セグメント毎の RTT

図 3.6 は，セグメント毎の RTT の時間変化を表している．なおこの値は，TCP セグメントに付与されている TCP タイムスタンプオプションから導出したものであり，送信側端末で，ACK セグメントを受信するごとに求めている．遠い地点にて RTT が大きく上昇しており，最初の遠い地点では 4 秒，二度目では 1.5 秒程度の RTT が見られている．30 秒付近の移動時に急激に RTT が下がり，近い地点では RTT が増加することなく安定している．遠い地点では，過度に大きい RTT を記録していることから，802.11n においても Bufferbloat 問題が発生しているものと考えられる．

図 3.7 に，データが送信される前に待つキューの長さを示す．送信待ちのためには，2 種類のキューが存在する．Linux カーネルにおいて，IP 層で送信されたパケットが無線 LAN ドライバに渡される前にバッファリングされるキュー（最大 1000 パケット）と，無線 LAN ドライバ内の送信待ちキュー（最大 124 パケット）である．図は 2 つのキューに入っているパケット数を合計した数を示している．前者のキュー長の取得は，独自のプログラムをユーザ空間で動作させ，短間隔のポーリングでキュー長を問い合わせることにより行った．後者のキュー長の取得は，無線 LAN ドライバを改修し，無線 LAN コントロー

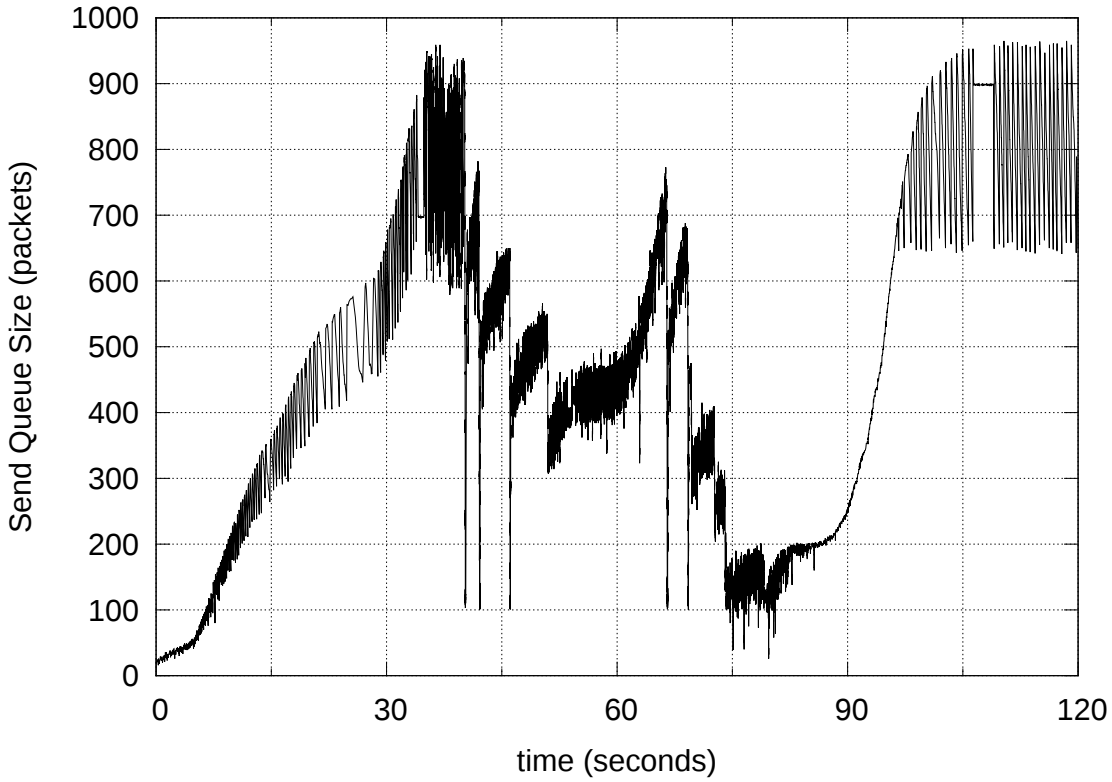


図 3.7: 送信待ちキュー長の時間変化

ラに対して送信コマンドを実行する時点でログに出力することで行っている．図 3.7 と図 3.5 の 2 つのグラフから，以下がいえる．

図 3.7 に示したキュー長は，値が頻繁に上下しているが，その上限の値は図 3.5 の輻輳ウィンドウの値に対応している．これは，輻輳ウィンドウ分のセグメントが，そのまま送信待ちキューにつながれるタイミングが存在することを意味する．従って，輻輳ウィンドウサイズがキューイングされるパケット数に大きな影響を与えていることが分かる．

その一方で，キュー長は頻繁に振動している．キューに格納された TCP のデータセグメントがキューから取り出され無線 LAN に送信されることでキュー長が減少する．これらのデータセグメントをサーバが受信し，対応する TCP の ACK セグメントを送信．その ACK セグメントを送信端末の TCP が受信することで，TCP のウィンドウが開き，次のデータセグメントが送信され，キュー長が増加する．これらの一連の挙動を繰り返すことで，キュー長の振動が発生している．

ここで注意すべきなのは，キュー長も，キューにおける振動の様子も，端末の位置（AP から遠い地点と近い地点），従ってレートの大小に関係なく同様の振る舞いをとることで

ある．特に，レートの高い近い地点においても，キュー長が 0 になることはない，従って，キュー長を観測しただけでは，遅延の大小は推定できないといえる．

送信待ちキューの長さはその時点の RTT と関連する．例えば，図 3.7 の 100 秒以降（後半の遠い地点）では，キュー長が 800 パケットを中心にして振動している．図 3.3 より，その場合の TCP スループットは 6Mbps 程度であることがわかる．従って，RTT は $1500[Bytes/pkt] \times 800[pkt]/6[Mbps]$ から求められ，1.5 秒となる．図 3.6 に示された RTT の実測値は，この区間で 1.5 秒付近を中心に上下しており，対応がとれていることが分かる．

今回実験で使用した CUBIC は，セグメント損失に限り輻輳ウィンドウを減少させる．図 3.5 では輻輳ウィンドウの減少が全体で 9 回であることが分かる．言い換えれば，通信全体で 9 個程度のセグメント損失しか発生していない．さらに低いレートが用いられた遠い位置においてはセグメント損失は発生していない．これは 802.11n による再送機能が有効に働いた結果である．

このような結果から，次のことが分かる．まず，端末と AP の距離が近い地点では，無線 LAN デバイスドライバは 216Mbps や 243Mbps などの高いデータレートを選択し，TCP 輻輳ウィンドウが十分に大きくなっていることから，高速な TCP 通信が可能である．一方，端末と AP の距離が遠い地点においては，6.5Mbps や 13.5Mbps などの低いデータレートに低下し，それに伴って TCP スループットも低下している．そして，RTT の顕著な増加があるが，これは Bufferbloat 問題が発生していることを示す．120 秒間を通してセグメント損失の数はそれほど多くはなく，特に距離が遠い地点ではその傾向は顕著である．その影響を受けて CUBIC では輻輳ウィンドウが増加する．TCP の輻輳ウィンドウの大きさに対応したパケット数が，送信キューにつながれる．これは無線 LAN のデータレートが高い場合も低い場合も同様である．しかし，データレートが高い場合はキューのパケットが高速に転送されるため，RTT は小さい．他方，データレートが低い場合は，キューにつながれたパケットを転送するために時間がかかり，RTT が増加している．

3.3 IEEE 802.11n の強力な再送

前節で示した通り，データレートの高低にかかわらず，IEEE 802.11n 無線 LAN の再送が有効に機能している．2.3 節で説明した通り，802.11n ではその高速化のためにフレームアグリゲーション（A-MPDU）と HT-immediate Block Ack の 2 つの工夫を行っている．しかしながら，従来通りの 802.11 再送も行っている．無線 LAN ドライバ ath9k では，IEEE 802.11n の再送手順をハードウェア再送とソフトウェア再送に分けて実現している [36]．ハードウェア再送は，データ送信側が Block Ack フレーム自身の受信に失敗した場合に，無線 LAN コントローラにて即時に行われる再送であり，従来通りの 802.11 再送である．ソフトウェア再送は，受信された Block Ack の Bitmap フィールドによって，未受信が通知されたフレームを，無線 LAN ドライバが再度，送信要求することで行われる再送であり，HT-imediate BlockAck によって導入された手順である．ハードウェア再送，ソフトウェア再送ともに，最大の回数が制限されており，Linux カーネル 2.6.38 時点の ath9k では，ハードウェア再送は 19 回，ソフトウェア再送は 10 回が制限回数となっている．この回数を超えた場合には再送を諦め，対応するデータフレームを破棄する．

3.2 節で示した実験結果のデータフレームの再送率（再送したデータフレーム数／送信したデータフレーム数を 0.5 秒ごとに計算した値）を図 3.8 に示す．破線がソフトウェア再送のみを，実線がハードウェア再送とソフトウェア再送を含む再送全体の割合である．

この図から，遠い地点においても近い地点においてもデータフレームの再送は一定量存在することがわかる．遠い地点の方が若干高く，40 から 50 %程度が再送である．また近い地点においても，データフレームの転送の内，30 %程度は再送となる．なお，遠い地点の方が再送率の変動が大きい．

遠い地点ではハードウェア再送が多くを占めており，逆に近い地点ではソフトウェア再送が多くを占めている．これは，遠い地点でレートが低くなった場合は，データフレームのプリアンブルや Block Ack が破損するなどして，アグリゲートしたデータフレーム全体を再送する必要が多かったことを示している．また少なからずソフトウェア再送が行われており，これは従来の無線 LAN であればデータフレーム全体の再送となっていたところを選択的な再送に留めている．一方，近い地点では，Block Ack による選択的な再送が効率よく働いているといえる．

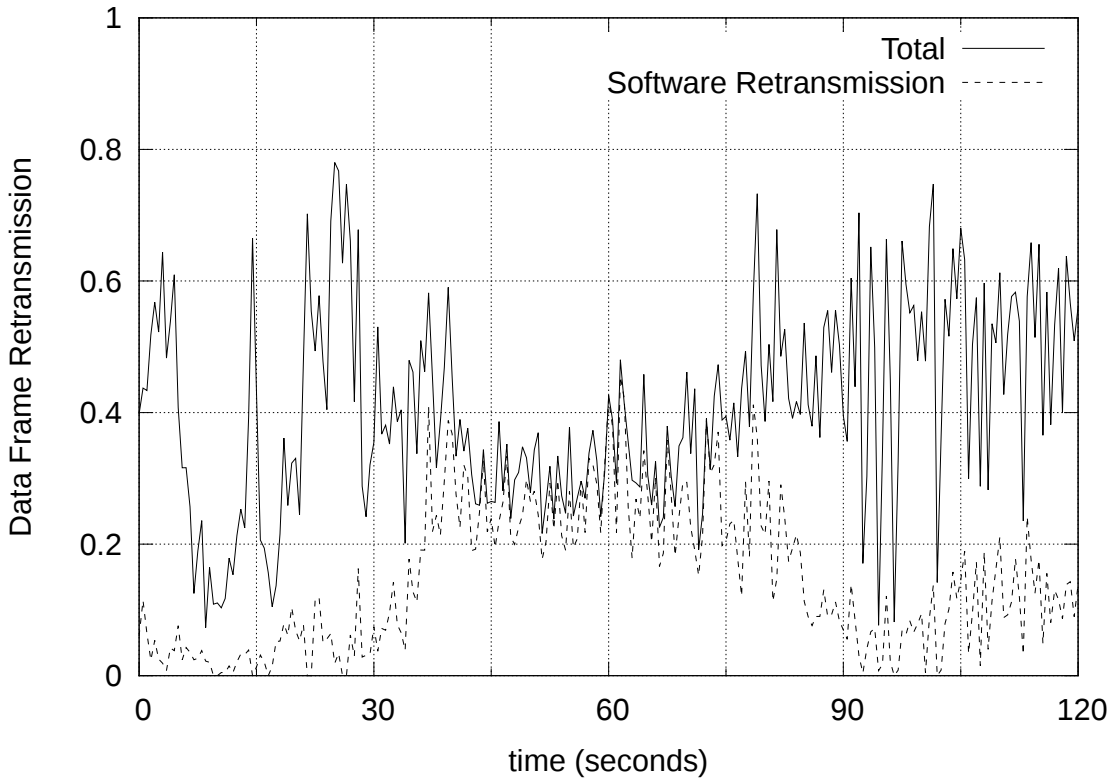


図 3.8: データフレーム再送率の時間変化

3.4 802.11n における Bufferbloat 問題の整理

3.1,3.2 節で得られた知見を基に，802.11n における Bufferbloat 問題がどのように発生するのかについて図 3.9 を使って改めて説明する．3.1 節の実験と同じように，端末とアクセスポイントが無線 LAN で接続され，アクセスポイントとサーバが Ethernet で接続されている．アクセスポイントはブリッジとして動作しており，無線 LAN インタフェースに到達したパケットは，Ethernet 側のインタフェースに転送される．端末からアクセスポイントを介してサーバに TCP 通信を行う．端末側の TCP から送信されたセグメントは IP パケットに格納されたのち，対応するインタフェースのキューにつながれる．キューからパケットが取り出され，無線 LAN ドライバによって送信される．このとき，端末とアクセスポイント間の無線通信品質が悪化すると高いデータレートによる送信にてフレームロスが多くなる．より効率的に送信を行うために，無線 LAN ドライバのレート適応アルゴリズムは低いデータレートを選択する．このデータレートの選択によって送信速度は低下するため，端末のキューには多くの送信待ちパケットがつながれる．一方，TCP がロスベースの TCP 輻輳制御アルゴリズムを採用している場合，パケットロスが発生す

るまでウィンドウを縮小しない．そのため，低データレートによって減少した帯域幅遅延積よりも過度に大きいTCPウィンドウサイズとなる．この広がったウィンドウサイズによって送信されたセグメントはキューにつながれる．このとき，802.11n無線LANでは再送が強力に動作することによって，誤りが上位層に伝搬することはなく，前述のTCPウィンドウは縮小されることなく広がっている．

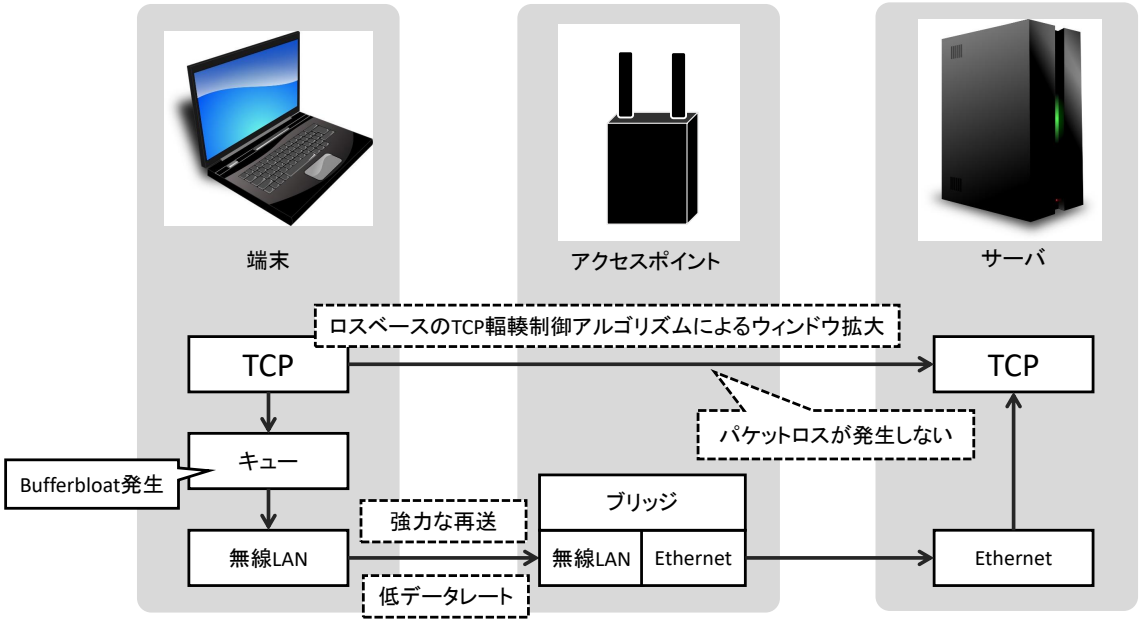


図 3.9: Bufferbloat問題発生のイメージ

3.5 まとめ

本章では，802.11nネットワークにおけるTCP通信の特性を調査するために，802.11nのAPを経由するネットワーク環境を構築し，端末とAPの距離を変えて，TCPによるデータ転送の実験的検討を行った．移動によるTCPスループットの低下は見らなかったが，Bufferbloat問題が発生している様子が見られた．その原因として802.11nの強力な再送があるとし，802.11nにおけるBufferbloat問題の整理を行った．さらに検討を行い，データレートが低い場合にBufferbloat問題が発生している可能性が高いことを示した．

第4章　無線データレートに応じて再送機能を低下させる方法の提案

本章では，802.11n 無線 LAN のデータレートに着目して，TCP 通信における Bufferbloat 問題に対してスループット性能の低下を引き起こすことなく遅延性能の悪化を抑制する方法を提案する．提案方法は，無線 LAN デバイスから取得可能であるデータレート情報を用いて意図的にパケットロスを発生させることによって送信者の TCP レートの送信速度を調整する仕組みであり，データレート情報に応じて再送回数の限界を制御することによってパケットロスを発生しやすくする．本提案方式は端末側，アクセスポイント側の双方に実装可能であり提案手法の詳細および効果については第 5,6 章にてそれぞれ示す．

4.1　Bufferbloat 問題への既存アプローチの 802.11n 無線 LAN への適応性

3 章にて明らかとなった 802.11n における Bufferbloat 問題に対して，既存のアプローチの適応性を検討する．3 章で整理したが，Bufferbloat 問題の原因はおおまかに大きいキューサイズである点，TCP が経路上のキューを溢れさせるまで送信速度を下げない点にある．これらの要因に対応する解決方法を分類して表 4.1 にて示す．さらに図 4.1 にて，その解決方法の適用のイメージを補足する．以下に各手法に対して検討する．

表 4.1: 802.11n における Bufferbloat 問題へのアプローチの比較

Bufferbloat問題に対するアプローチ	具体例	対応を要する箇所			検知する指標	備考
		送信キュー	TCP送信者	TCP受信者		
①送信キュー長の最大値を制限		○				×性能悪化
TCPの送信速度を抑制						
②TCPセグメントの送信を制限	TCP small queues	○	○		データ滞留量	△適用不能なケースあり
③広告ウィンドウサイズを抑制	DRWA			○	RTT	×サーバの変更は非現実的
輻輳ウィンドウの拡大を抑制						
④輻輳制御アルゴリズムの変更	Vegasなど		○		RTT	×性能悪化の懸念
⑤意図的なパケットロスを発生	REDなどのAQM	○			キュー長など	△性能悪化の懸念

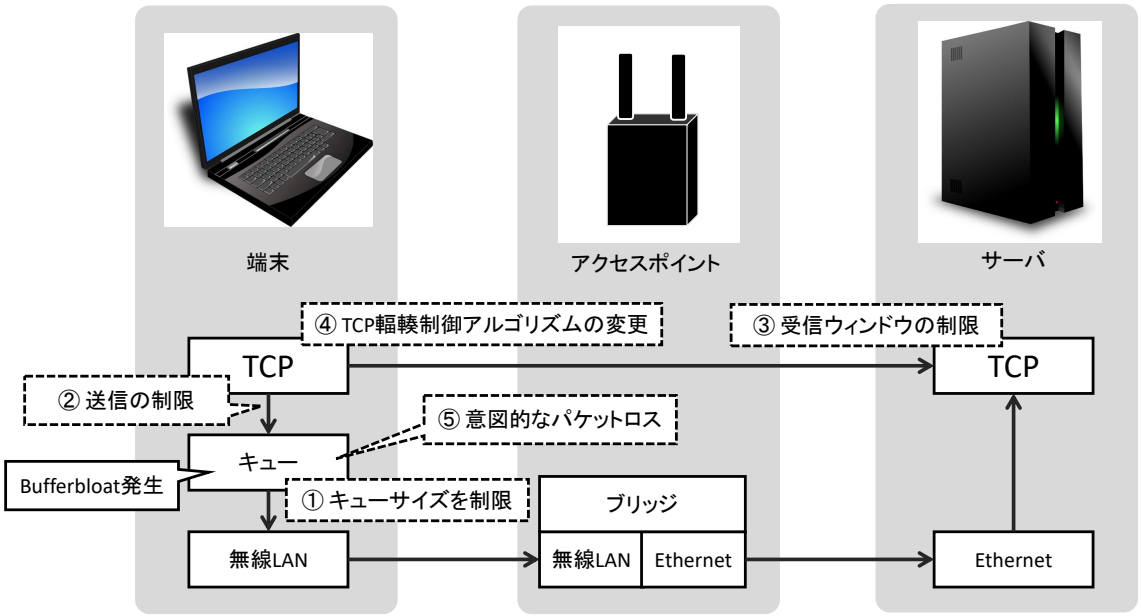


図 4.1: 802.11n における Bufferbloat 問題へのアプローチの各イメージ

① **キューサイズを制限する** Bufferbloat 問題における遅延悪化の原因はキューイング遅延である．キューに多くのパケットがつながれ，その待ち時間が問題となっているため，キューそのもののサイズを小さくすることでこれを解消することができる．しかしながら，802.11n などの高速無線 LAN は，高速な通信が可能である．さらにアクセスポイントからサーバへの経路が高遅延のネットワークである場合，より大きな帯域幅遅延積が必要となる．TCP のバースト性のある通信に対応するため，キューは帯域幅遅延積を満たさなければバッファ溢れによるパケットロス，それに伴う TCP 輻輳ウィンドウの縮小を引き置きしてしまい，結果としてスループット低下となると考えらえる．

② **TCP 送信を抑制する** 生成できる TCP のデータ量を制限し，制限以上はデータセグメントを送信しない方法であり，キューにつながれるパケット数を制限することができる．2.7.2 節に説明した TCP small queues が代表例である．不要なデータ送信を行わない方法であり，パケットロスを引き起こさない．しかし，この生成されたデータ量の測定は自身の端末のインタフェースキューおよびドライバ内のキューに待機しているセグメントに対して行われるため，他の端末で発生したキューイング遅延の解消は難しい．

③ **TCP 受信側の受信ウィンドウを制限する** TCP フロー制御を利用して，受信側の広告

する受信ウィンドウサイズを小さく設定することで受信ウィンドウを制限することで，送信側 TCP の送信ウィンドウを制限することができる．TCP のウィンドウが広がらないため，キューにつながれるパケット数を制限することができる．ただし，スループット低下の可能性があるため，DRWA では RTT による動的な受信ウィンドウサイズの決定の工夫を行っている．しかしながら，TCP を受信するサーバ側での対応が必要である点が問題である．具体的には，サーバでは多数の要求を同時に処理する必要がある点と，端末が無線 LAN を使用しているかどうかを検知することができない点である．

④ **TCP の輻輳制御アルゴリズムを遅延ベースのものに変更する** ロスベースの輻輳アルゴリズムはセグメント損失が発生するまで，輻輳ウィンドウを増加しつづける．遅延に応じてウィンドウを変更する遅延ベースの輻輳制御を採用すれば，輻輳ウィンドウの過度な増加を防ぐことができる．しかしながら，現状ではロスベースの輻輳制御アルゴリズムが支配的であり，変更は容易ではない．さらに，バッファを溢れさせるまで送信速度の上昇・維持を続けるロスベースのアルゴリズムに対して，往復遅延時間に過敏に対応するため過度に輻輳ウィンドウサイズの制限を行ってしまい，スループット性能の悪化が懸念される．

⑤ **意図的なパケットロスを発生させる** キューにパケットが滞留する場合に意図的にパケットロスを発生させる方法である．2.7.1 節に示した CoDel が代表例である．パケットロスによって TCP 輻輳ウィンドウが縮小することを利用し，TCP の送信速度を下げることでキューイング遅延の増加を抑制する．反面，パケットロスによる輻輳ウィンドウの縮小はスループット低下を招く懸念がある．

4.2 方式提案のための基本検討

前述のように，802.11n 無線 LAN では，通信環境により無線データレートが大きく変動する．それに伴い遅延の悪化の状況も変化する．以下にこの無線データレートによる遅延悪化に関して検討する．

Bufferbloat 問題はロスベースの TCP がキューをあふれさせるまでウィンドウを拡大し続けることから発生する．このキューがパケットであふれている状態のキューイング遅延

$Delay_{queueing}$ の計算式を示す．

$$Delay_{queueing} \leftarrow MaxQueueLength/DataRate \tag{4.1}$$

MaxQueueLength はキューの最大長，DataRate は伝送速度を示している．無線 LAN における無線データレートと実質のデータ転送速度は異なるが，簡単のためにデータレートを用いる．式から得られたデータレートとキューイング遅延の対応を図 4.2 に示す．キューの最大長は現実のシステムに即して，インタフェースキューの長さが 1000 パケット，ドライバ内のバッファが 128 フレームとした 1,500[bytes/pkt] を 1,128 パケットとする．両対数上で直線となっており，式から明らかなとおり，キューイング遅延とデータレートは反比例の関係にある．データレートが 100Mbps のとき 100 ミリ秒程度の遅延であるが，10Mbps のときは 1 秒，1Mbps の場合は 10 秒とサービス品質を大きく悪化しうる遅延時間となっている．

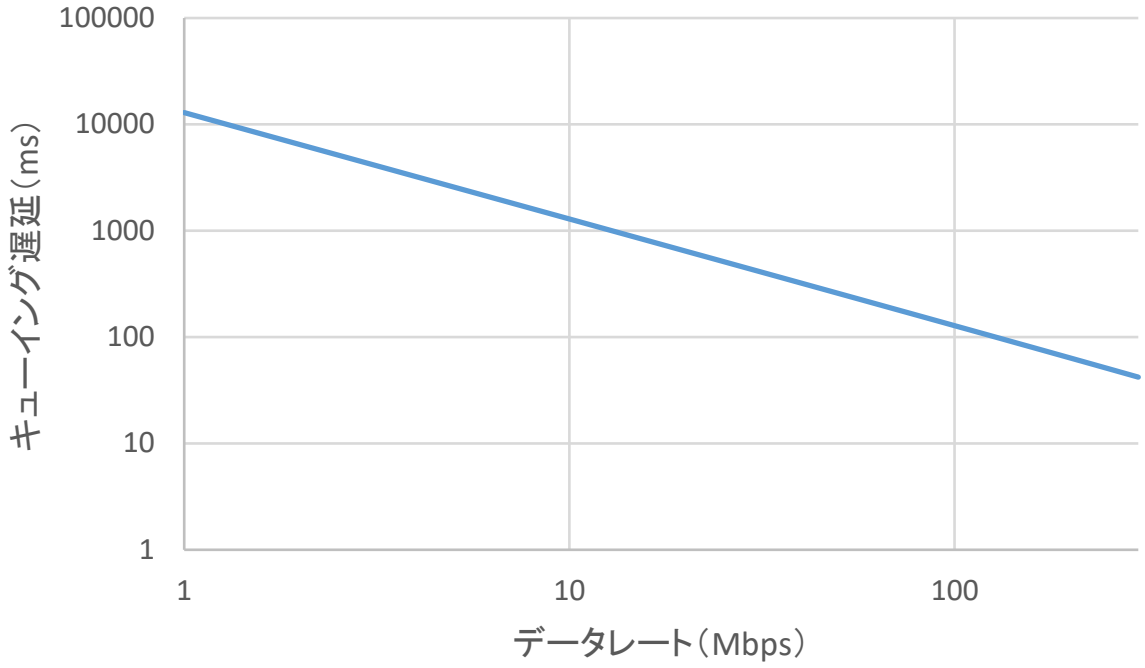


図 4.2: データレートから試算したキューイング遅延

このように，データレートによって遅延性能の悪化の度合いが異なり，データレートがより低いほど悪化する．

一方，3.2 節で得られた図 3.7 の結果では，キュー長の大小と遅延時間は関連しておらず，キュー長から遅延悪化の推定は行えないことを示している．さらに，高い無線データ

レート時においてもキュー長は伸びており，キュー長によるパケットロスを採用した場合，かえってパケットロスによるスループット低下を招く懸念があることを示している．これはリンク速度が動的に変化する無線 LAN の特徴によるものである．

このため，無線 LAN における Bufferbloat 問題を解決するためには，無線データレートの値に対応する方法が必要であると考えられる．

4.3 データレートに応じて再送機能を制御する手法の提案

以上の検討から，802.11n 無線 LAN を対象とする端末からアクセスポイントへのアップロード TCP 通信においてインタフェースキューで発生するバッファリング遅延を，無線データレートに応じて 802.11n 再送機能を調整することで解消する方法を提案する．

本提案方式のアイディアは 802.11n の Bufferbloat 問題を 802.11n のデータレートと再送機能の変更で対応する点にある．3 章の実験にてパケットロスが発生しにくい状況であるとし，その原因に 802.11n の強力な再送機能があることを示した．そこで，802.11n の再送機能に着目して Bufferbloat 問題を解決する方法を検討する．具体的には 802.11n にて新たに導入された HT-immediate Block ACK の再送最大回数を変更することで再送機能を弱化し，あえてパケットロスを発生させる．

さらに，データレートに応じて，再送回数を調整する．4.2 節で述べた通り，低いデータレートであるほど遅延悪化が顕著である．そのため，低いデータレートであるほど再送機能を弱化する必要がある．しかし，反面，高遅延ネットワークに接続している場合，高いデータレートでは TCP ウィンドウが拡大できなければスループット低下を招いてしまうため，再送機能の弱化は控える．すなわち低データレートの場合は再送回数を小さくすることで遅延性能の低下を抑制し，高データレートになるにしたがって再送回数を通常に戻していくことによって，TCP のスループット性能の低下を防ぐ．

このアイディアの実装に際して，4.4 節に示した通り，デバイスドライバにて送受信データレートの取得が行え，かつ再送機能の制御が行えるため，無線 LAN デバイスドライバの変更による対応が十分に可能である．無線 LAN チップの変更は必要ないため，現実的な手法である．

本提案手法は 802.11n に特化することによって端末のインタフェースキューの遅延問題

に対して，端末側とアクセスポイント側の双方で対応可能である．図 4.3 に提案手法のイメージを示す．アクセスポイントに無線 LAN で接続している端末からサーバへの TCP 通信において，端末のキューで Bufferbloat 問題が発生する．このキューに紐づく無線 LAN リンクに対して提案手法は再送機能の弱化を行う．

提案手法を実現する直接的な方法は，送信側である端末側において再送機能を弱化するというものである（図中の①）．本論文では，このアプローチに対する詳細な実装方法を示すとともにその性能評価を述べる．一方，このアプローチではすべての端末において，その実装が必要となる．しかし，様々な種類の端末が使用される現状を考慮すると，個々の端末への適用は困難であると考えられる．そこで第二のアプローチとして，提案手法を受信側であるアクセスポイントに実現する方法を考案する（図中の②）．本論文では，第二のアプローチに関しても詳細な実装方法と性能評価を示す．

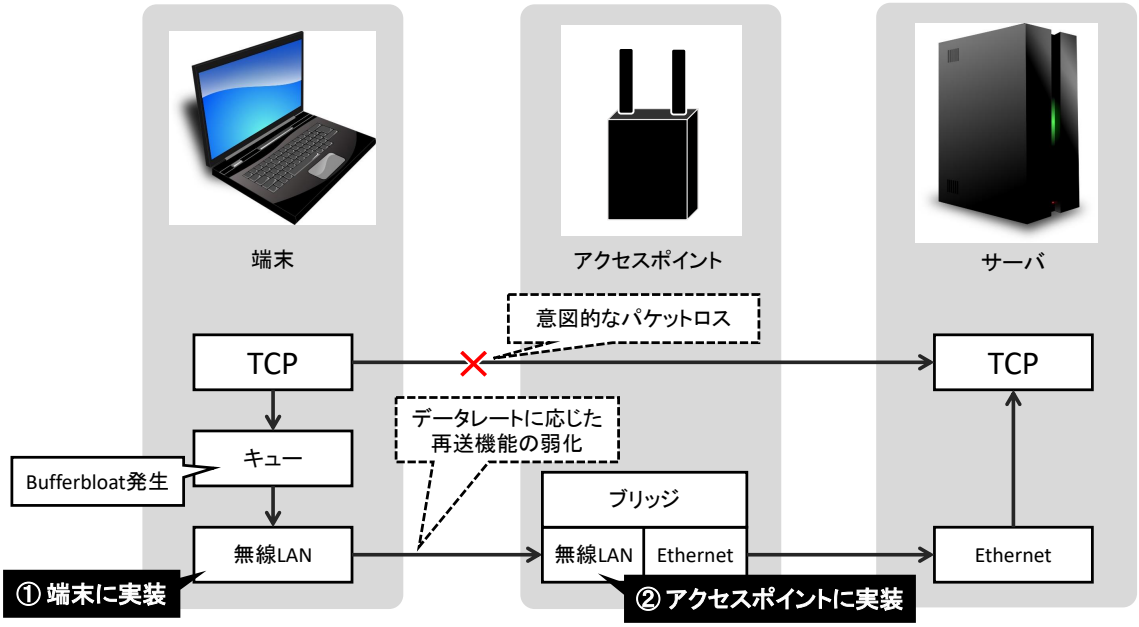


図 4.3: 提案方式のイメージ

4.4 提案方式の実装方法に関する検討

一般的に 802.11n における A-MPDU の送受信処理は，データ送信側とデータ受信側の無線 LAN チップとデバイスドライバによって実現される．それぞれの行う処理は図 4.4 のようになる．無線 LAN チップでは，物理層の電波の送受信や即時の応答が求められる

ようなハードウェアに依存した処理が行われる．一方，デバイスドライバではデータの送り込みなどのハードウェアの制御やMPDUの選定や再送判断などの時間の余裕のある処理がソフトウェアによって行われる．

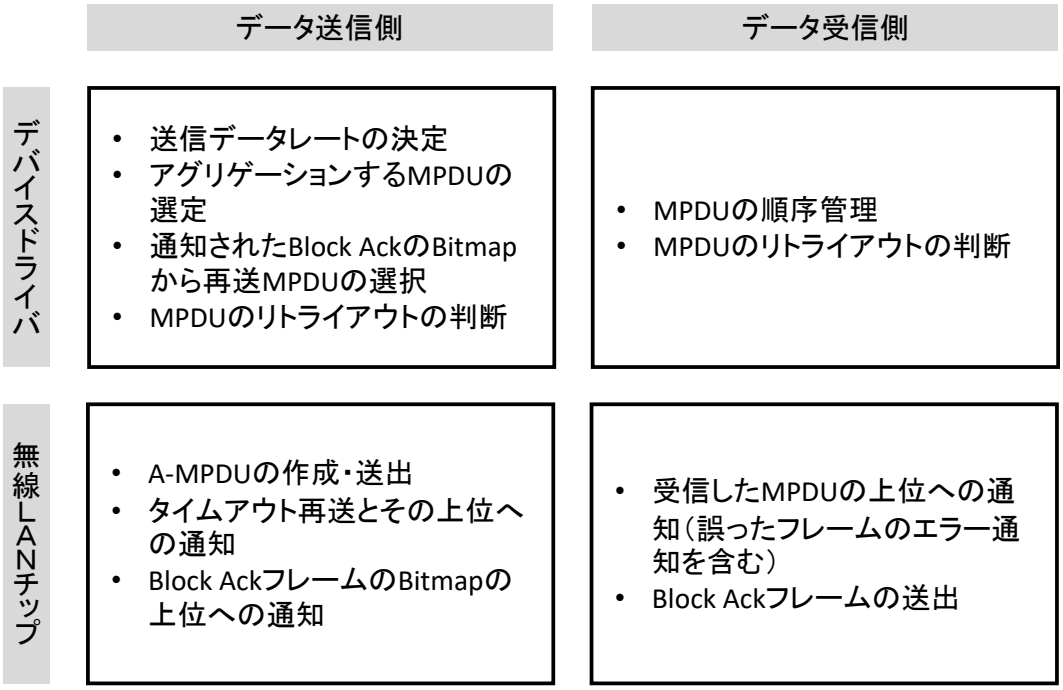


図 4.4: A-MPDU の送受信処理の分担

具体的には以下のように行う．データ送信側では，デバイスドライバにおいて，上位層からのデータ送信要求をリングバッファに格納し，その中からアグリゲーションを行うMPDUを選択する．その中には，Block AckフレームのBitmapから判断された再送フレーム分のMPDUも含まれる．また，MPDUごとに再送回数を記録しており，一定回数の再送が行われた場合にリトライアウトを判断する．また，この送信成否を次回の送信データレートの決定に用いている．これに対して，無線LANチップでは，実際にA-MPDUを作成して送出する処理，送信したがACKが返ってこない際のA-MPDU全体のタイムアウト再送の処理，受信したBlock AckフレームのBitmapのデバイスドライバへの通知などの処理を行う．

一方，データ受信側では，無線LANチップにおいて，受信したMPDUのデバイスドライバへの通知，Block Ackフレームの送信などを行う．なお，MPDUの通知機能では，CRCエラーとなったMPDUに関する情報も知らされる．それに対して，受信側のデバイスドライバでは，MPDUの順序管理やリトライアウトの判断を行う．受信側でのリトラ

イアウトは，送信側の Block Ack Request によって判断される．さらにデバイスドライバでは，受信した MPDU の送信データレート，受信信号強度などの情報も取得しており利用可能である．

このように，デバイスドライバと無線 LAN チップによって，処理の分担が行われている．無線 LAN チップ側の処理の変更は困難であるが，デバイスドライバ側の処理はソフトウェアで構成されているため，現実的に変更が可能である．このため，本論文では提案方式を端末およびアクセスポイントのデバイスドライバに実装するという方針を採用する．

4.5 まとめ

本章では，まず Bufferbloat 問題への既存アプローチの無線 LAN への適応性の検討を行った．そして，無線 LAN に特化した方式提案のための基本検討を行い，無線データレートが低いほど遅延性能が悪化することを示した．また 3 章の実験結果から無線データレートに関わらずキュー長が伸びていることから，パケットロスによるスループット低下の懸念があることを示した．これらの検討から 802.11n 無線 LAN のデータレートに着目して，TCP 通信における Bufferbloat 問題に対してスループット性能の低下を引き起こすことなく遅延性能の悪化を抑制する方法を提案した．本提案方式の実現方法は端末側とアクセスポイント側の 2 つのアプローチがある．さらに，提案方式の実装方法に関する検討を行い，デバイスドライバにて現実的に実現可能であることを述べた．

第5章　端末側によるデータレートに応じて再送機能を制御する手法

本章では，3 章にて提案した 802.11n にてデータレートに応じて再送機能を制御する手法を端末側に実装して評価を行い，遅延悪化抑制の効果があることを確認する．3 章にて，802.11n 無線 LAN における低データレート時の通信において，TCP によるバルクデータ通信が引き起こす遅延の悪化について述べた．この問題を解決すべく，4 章にて無線 LAN の再送回数を制限することによって意図的にパケットロスを生じ，TCP の輻輳ウィンドウの減少を狙う手法のアプローチを示した．この提案手法を，まず端末のデバイスドライバで実装を行い，その性能を評価する．

5.1　詳細なアプローチ

本提案方式は，802.11n の Block Ack に基づく再送の，最大再送回数を少なくすることによって，再送機能の弱化を行う．この意図的に少なくした最大再送回数は宛先ごとに管理する．送受信のたびに MPDU 毎の再送回数を取得し，再送回数が最大再送回数を超えているかどうかの確認を行う．本方式は，端末側およびアクセスポイント側のデバイスドライバにて実装が可能であるが，実装箇所によって詳細な設計は異なる．端末側による手法の実現のためのデータレートや MPDU 毎の再送回数などの情報は，もともとの 802.11n 仕様に基づく振る舞いで使用されており，これらの情報を積極的に用いることとする．

以下に，提案方式の端末側の詳細な実装方法を示す．

再送機能の低下　既存の Block Ack 時の処理時，再送判断に意図的に少なくした最大再送回数を用いるようにすることで，この値を超える場合には MPDU 再送のための再キューイングが行われずに破棄される．MPDU が TCP セグメントを含み，かつ再送回数が最大再送回数を超えた場合，その MPDU は紛失したものとして処理する．

TCP セグメントを含まない場合は，デバイスドライバの既定動作を行う．

データレートの取得 A-MPDU 作成時，使用されるデータレートを，指数移動平均により平滑化したものを使用する．その際の平滑化指数は 0.25 とする．

再送回数の取得 MPDU 毎の再送回数は，ドライバ上で Block Ack に基づく再送のために保持しているため，この値を利用する．

5.2 最大再送回数の減少の影響範囲の限定

本提案では最大再送回数について，データレートに応じてより少なくした値を適用することでパケットロスを意図的に誘発する．この制御によって，本来の無線 LAN において発生しなかった悪影響が生じることはないか検討する．

最大再送回数を既定の回数よりも小さくすることによる影響は，早期に再送を諦めることによってリトライアウトしやすくなる点にある．リトライアウトしやすい，という性質は特に上位層に伝送誤りとして伝わる．上位層が TCP である場合は，ロスベースであれば輻輳ウィンドウを縮小し再送を行う．UDP である場合は，再送を行わないため，回復が行われるかどうかはアプリケーションの実装に左右されるが，一般的には損失として扱われる．そのため，最大再送回数を少なくする方法は損失は UDP を用いるアプリケーションの品質を悪化させる懸念がある．

そこで，本提案方式では，最大再送回数の制御時に，フレームのペイロード部分の解析を行い，本制御の適用は TCP に限定する工夫を行っている．さらに，データフレーム以外の管理フレームなどに対する再送も規定の回数を用いる．これらの工夫によって，UDP などの他のトランスポート層プロトコルに対しても，既存と同様の通信品質を提供できる．具体的には，MPDU のパースを行い，IP が含まれる場合は IP ヘッダのパースをさらに行い，プロトコル番号が TCP となっているかどうかを見て提案手法の適用可否を判断する．物理層エラーの場，MPDU に IP ヘッダが含まれない場合，IP ヘッダを含むがプロトコル番号が TCP ではない場合は提案手法は適用せず，既定の動作を行うものとする．

5.3 提案手法の無線LANドライバへの実装の検討

本提案手法を Atheros 社製の IEEE 802.11n 無線 LAN チップセットに対応する Linux OS 向けの無線 LAN デバイスドライバである ath9k に実装する．ath9k は自由に変更できるオープンソースソフトウェアであり，端末・AP・モニターなどの機能性が存在する．

図 5.1 に無線 LAN デバイスドライバ（ath9k）のデータフレーム送信処理とソフトウェア再送処理において提案方式を実装した場合の処理フローを示す．図のうち，斜線部分が提案方式で追加した処理である．処理フローの始まりは，OS からの送信要求とデバイスドライバからの割り込みに分かれている．まず，OS からの送信要求を受けると，デバイスドライバは送信に利用するデータレートの選択を行う．提案方式では，決定したデータレートと，これまでのデータレートの平均値から，新たな移動平均を導出する．続けて，フレームアグリゲーションを行わない場合は，無線 LAN チップに対して送信コマンドを発行する．フレームアグリゲーションを行う場合は，アグリゲーションのためのキューにフレームをつなぐ．なお，送信待ちキューとアグリゲーションキューの全体が前述のドライバ内キューに相当する．

送信可の割り込みに関しては，AMPDU 送信のモジュールを呼び出す．アグリゲーションはデータレートに応じて集約数が決定される．

送信完了割り込みでは，対応する送信フレームをキューから削除し，送信フレームがアグリゲートされていない場合は，バッファを解放して送信ステータスを設定する．アグリーゲートされていた場合は，個々のフレームに対して次の処理を行う．

- 正常に受信された場合は BlockAck のウィンドウを更新し，バッファの解放と送信結果の設定を行う．
- 受信されていない場合は，TCP セグメントの場合はデータレートの平均値から最大再送回数を決定し，それ以外はデフォルト値の最大再送回数（今回の実装では 10）とする．
- その後，再送回数が最大回数より小さい場合，フレームを再送 AMPDU バッファに入れ，そうではない場合はバッファを解放し受信結果を設定する．

再送 AMPDU バッファが空ではない場合は，それをアグリゲーションキューの先頭に挿

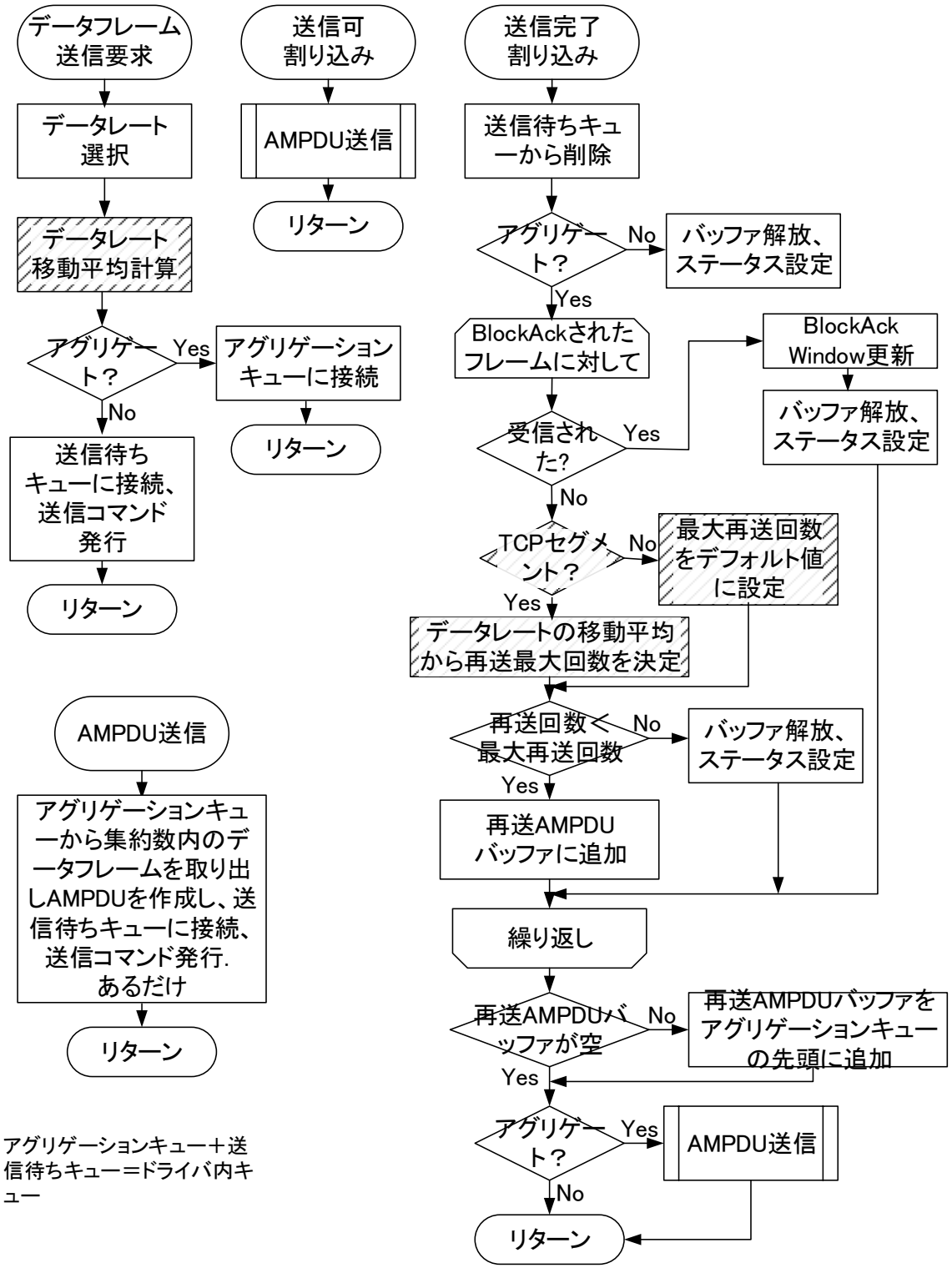


図 5.1: デバイスドライバの処理

入し, AMPDU 送信モジュールを呼び出す．AMPDU 送信モジュールでは, 実際に APMDU を作成し送信待ちキューにつなぎ，送信コマンドを発行する．

5.4 最大再送回数の検討

5.4.1 最大再送回数を 2 回に制限した実験

まず，最大再送回数の制限が Bufferbloat 問題による遅延悪化の抑制に効果を示すのかどうかを確認を行うため，第 3 章の実験環境において，以下のような手法を用いる．

1. 使用する無線データレートが 80Mbps よりも小さい場合にソフトウェア思想の再送制限回数を 2 回に設定し，データレートが 80Mbps 以上の場合は，そのままの値である 10 回のままとする

2. 判断に用いる無線データレートの値は，送信時に用いたデータレートを指数移動平均により平滑化したものを使用する．その際の平滑化指数は 0.25 とする．

本実験では，インターネットアクセスを行った場合の評価を行うため，遅延器をアクセスポイントとサーバの間に追加する．この遅延器はブリッジで構成され，インタフェースの送信キューにて Linux カーネルの netem[38] を適用することで実現している．実験場所や移動時間の割り当ては第 3 章と同様である．

まず，図 5.2 から図 5.6 までに付加遅延なしの場合の結果を示す．図 5.2 のデータレートは第 3 章の実験結果と比較して同様の結果となっている．図 5.3 のスループットは，図 3.3 と同様な結果を示しており，性能悪化は見られない．図 5.4 の RTT では，図 3.6 のような遅延悪化が抑制されている．後半にて大きい遅延悪化がみられるが，図 5.5 のキュー長では後半部分で大きく上昇はしていないため，キューイングによる遅延ではなく無線リトライアウトによる TCP 再送の影響によるものだと考えられる．図 5.6 の TCP 輻輳ウィンドウは，遠い地点では低く，近い地点では高くなっており，結果として，遠い地点では送信待ちキューを抑えることによって，遅延悪化を防げることを示している．

次に，100 ミリ秒の遅延を付加した場合の結果を示す．図 5.7 と図 5.8 に，ネイティブの手法のスループットと RTT を，図 5.9 と図 5.10 に最大再送回数を 2 回に制限した場合のスループットと RTT を示す．図 5.9 のスループットは，図 5.7 と大きく変わらず，スルー

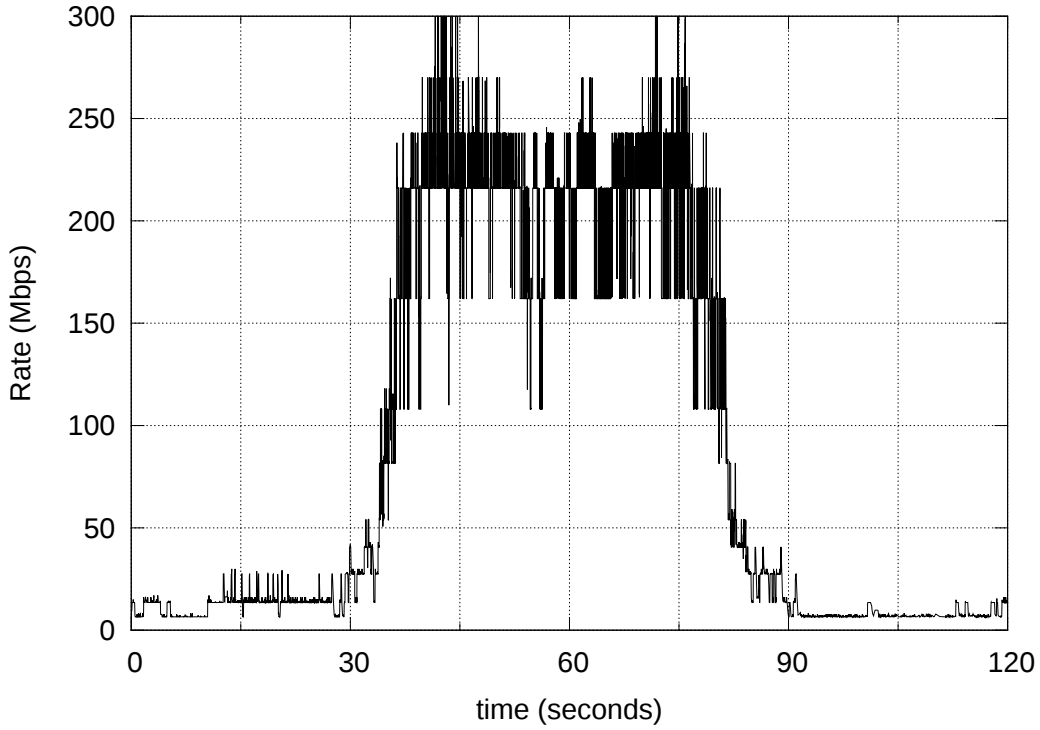


図 5.2: 無線データレートの時間変化（付加遅延なし）

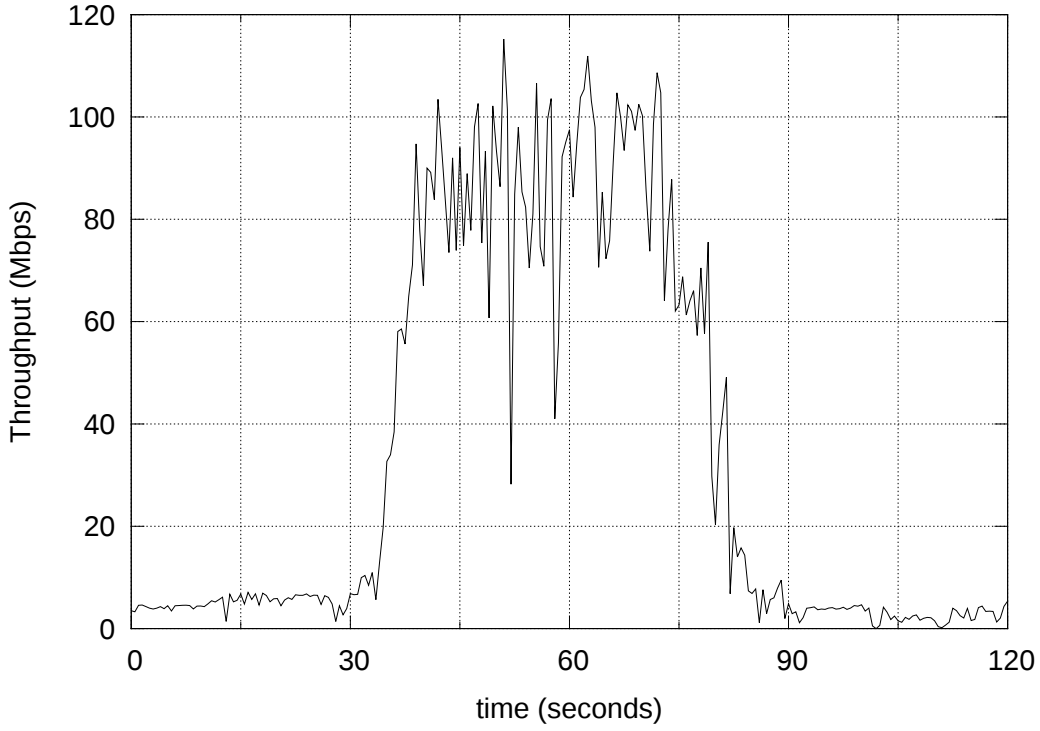


図 5.3: スループットの時間変化（付加遅延なし）

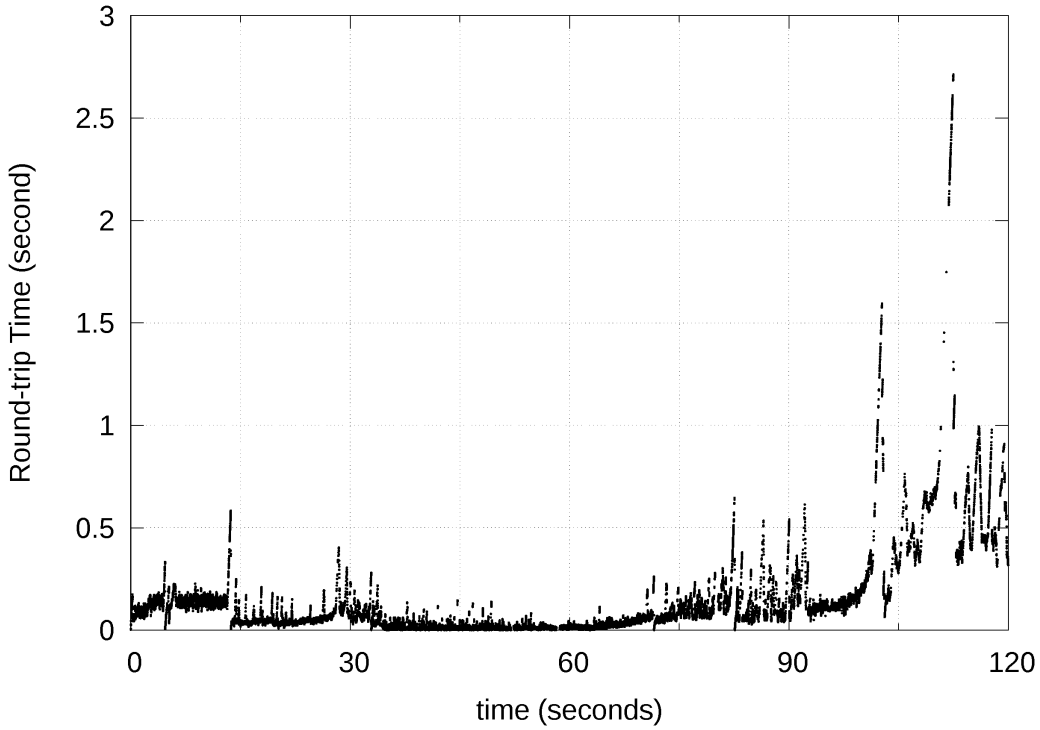


図 5.4: 往復遅延時間の時間変化（付加遅延なし）

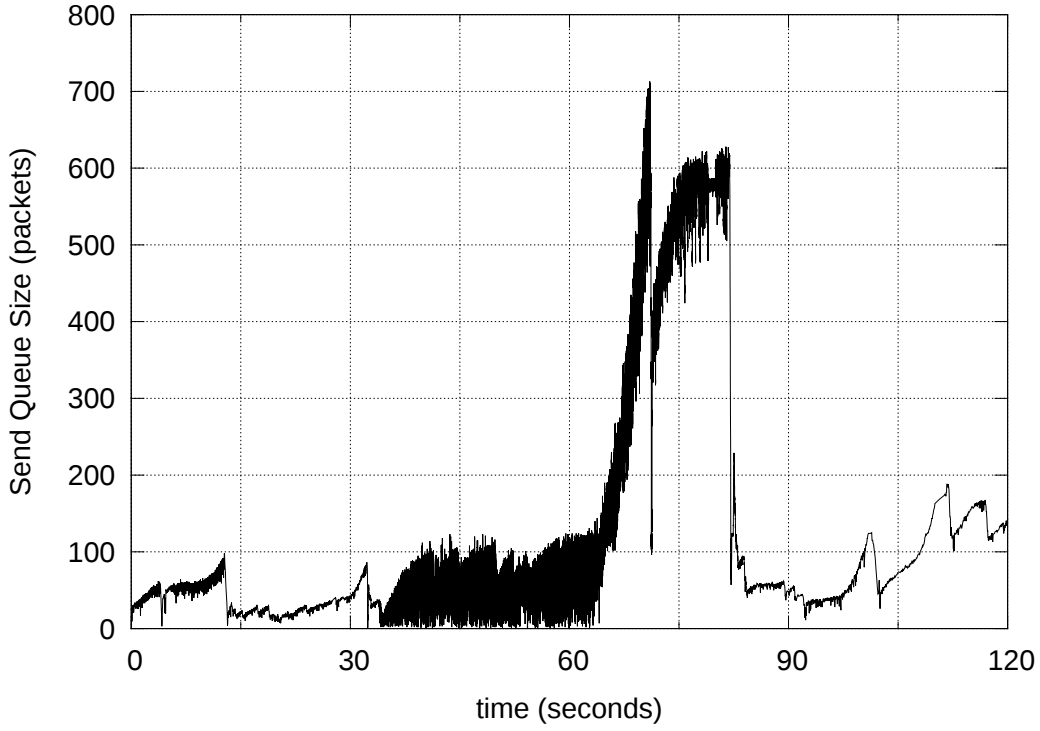


図 5.5: 送信待ちキュー長の時間変化（付加遅延なし）

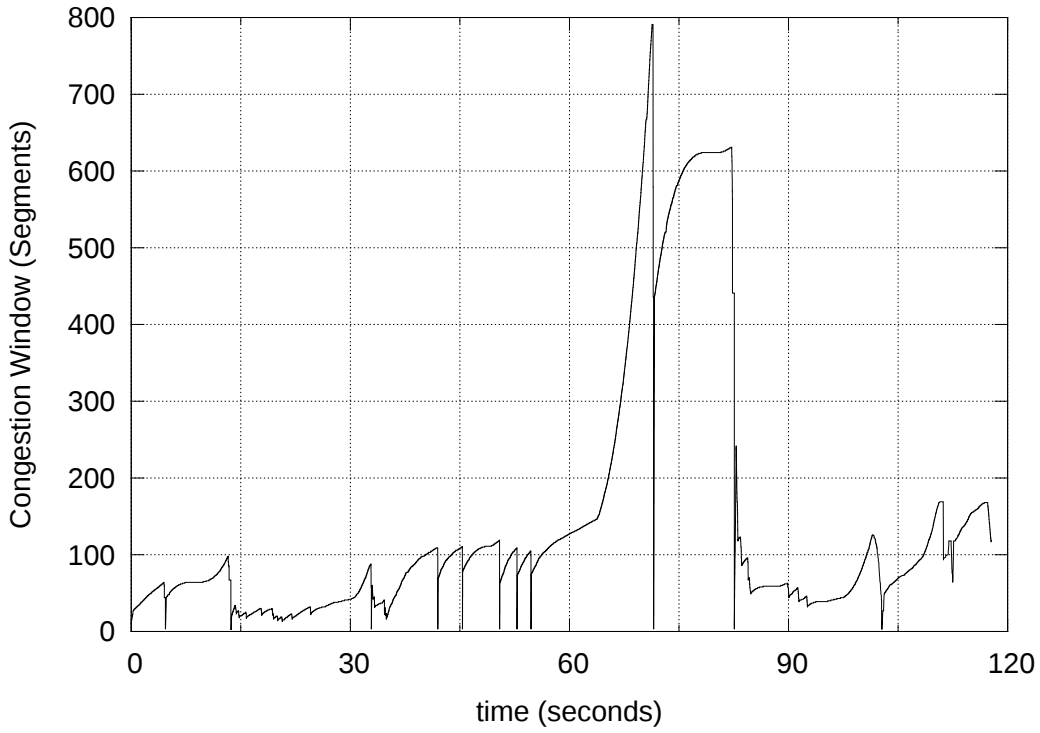


図 5.6: TCP 輻輳ウィンドウ値の時間変化（付加遅延なし）

プット性能の悪化は見られない．図 5.10 の RTT は，付加遅延により 100 ミリ秒の底上げが行われているが，図 5.8 と比較すると，追加遅延なしの場合と同様に，低い値に抑えられている．これらの結果から，100 ミリ秒の往復遅延が存在するネットワーク環境においても，最大再送回数の制限は有効である．

以上のことから，6.5Mbps および 13.5Mbps のような低い無線データレートにおいて，最大再送回数を 2 回に設定することは有効であり，遅延悪化の抑制に効果があるといえる．

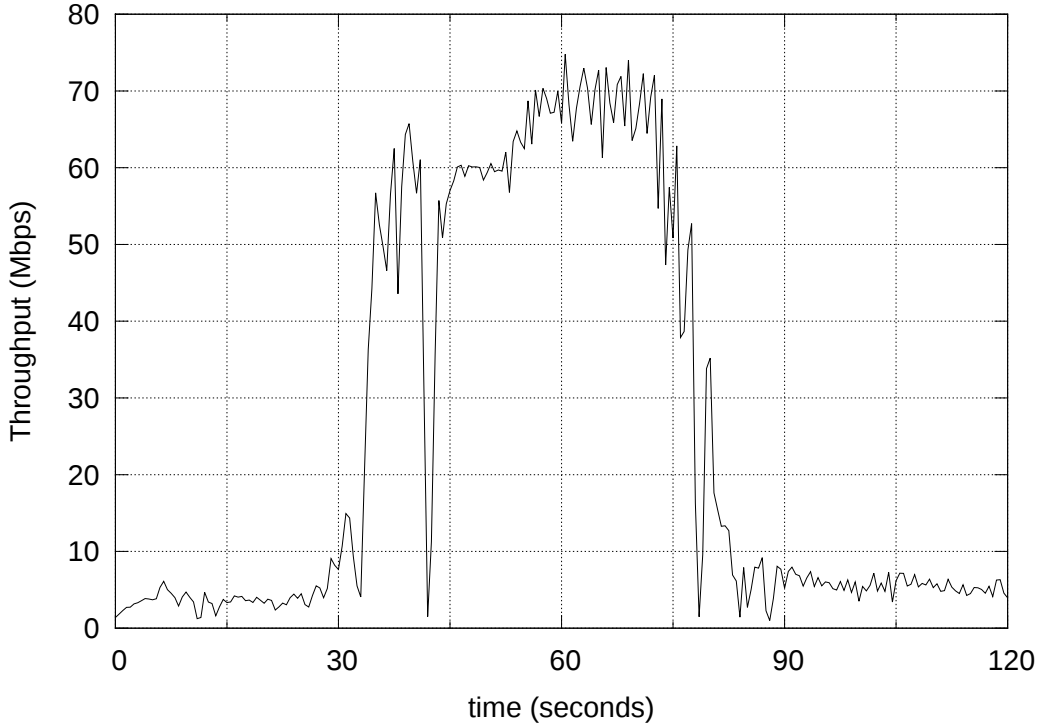


図 5.7: ネイティブのスループットの時間変化（付加遅延 100 ミリ秒）

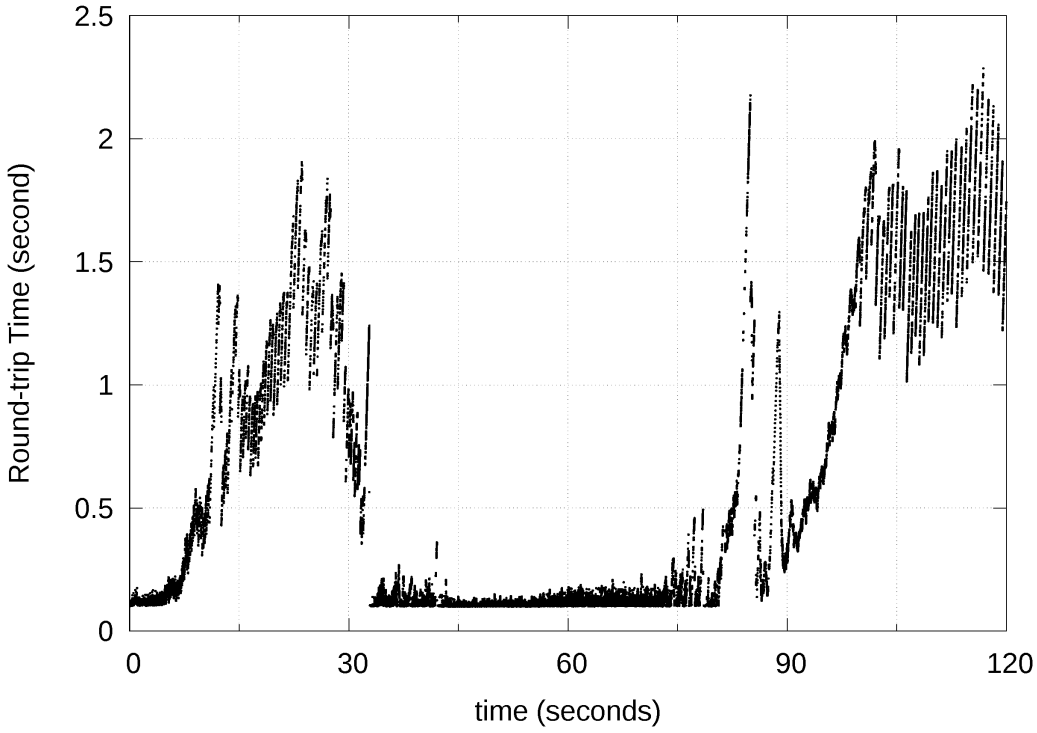


図 5.8: ネイティブの往復遅延時間の時間変化（付加遅延 100 ミリ秒）

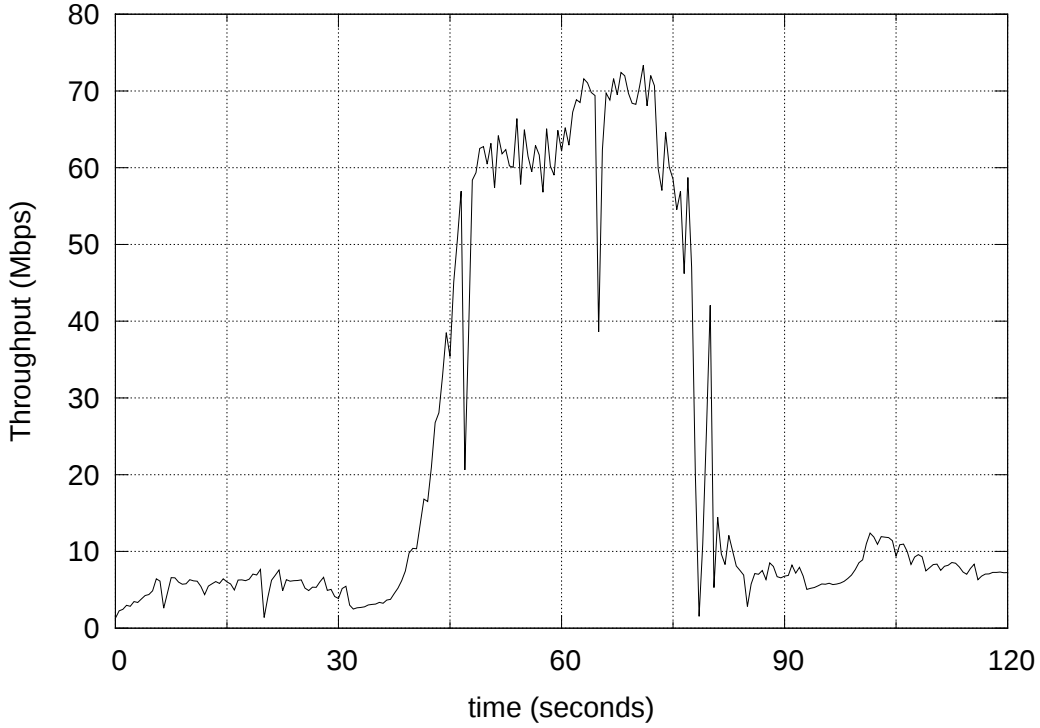


図 5.9: 最大再送回数を制限した手法のスループットの時間変化（付加遅延 100 ミリ秒）

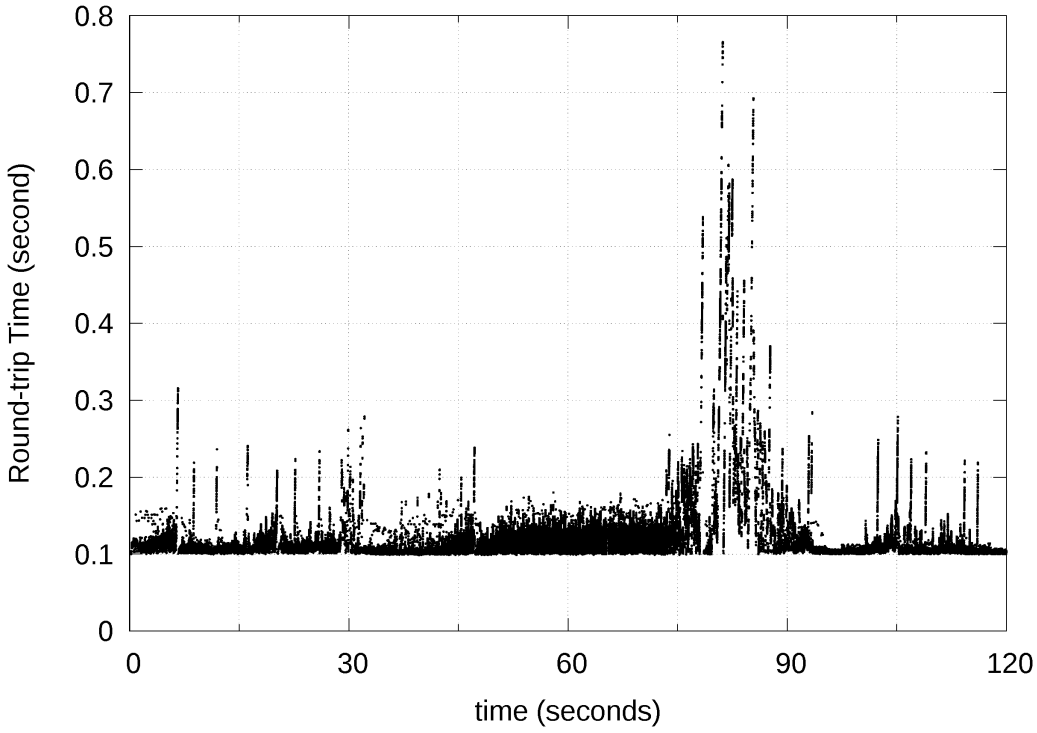


図 5.10: 最大再送回数を制限した手法の往復遅延時間の時間変化（付加遅延 100 ミリ秒）

5.4.2　データレートに応じた最大再送回数の決定方法

前項の実験においては，近い地点と遠い地点の 2 箇所に注目して実験を行っている．しかしながら，その間の範囲に関しては明確な結果を得ていないため，補間の必要がある．

データレートに応じて設定される最大再送回数は，遅延悪化の抑制とスループット低下のトレードオフを決定する．前項の事前実験にて，データレートが 6.5Mbps および 13.5Mbps の場合，最大再送回数として 2 回が適切であることが分かっている．一方，100Mbps 以上においては既定の最大再送回数である 10 回で遅延悪化は見られない．10Mbps と 100Mbps の間に関しては，図 5.11 のようにログスケールの MAC データレートに対して 2 と 10 の間を補間する直線（linear）を想定する．その直線を参考にして，図の実線のような階段状の値（step）を採用する．

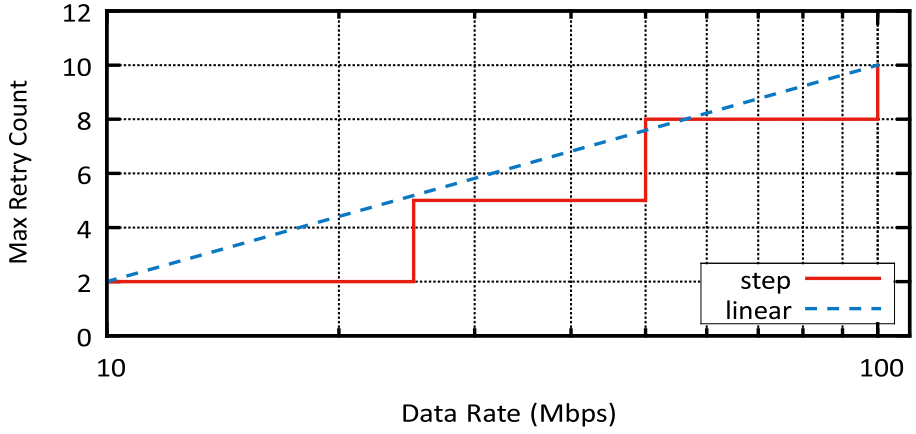


図 5.11: 提案手法の最大再送回数

改めて，最大再送回数を表 5.1 に示す．

表 5.1: 最大再送回数

平滑化データレート [Mbps]	～25	25～50	50～100	100～
最大再送回数	2	5	8	適用外

5.5　提案手法の端末側での実装方式

以上の検討を踏まえ，端末側において提案手法を実装する方法は以下の通りである．

1. 提案手法は ath9k 無線 LAN デバイスドライバに実装する．

2. Block Ack の受信時に無線データレートに応じた最大再送回数を設定する．このとき，無線フレームの内容が TCP セグメントかどうかを確認し，TCP セグメントではない場合は提案手法は適用しない．

3. 判断に用いる無線データレートの値は，送信時に用いたデータレートを指数移動平均により平滑化したものを使用する．その際の平滑化指数は 0.25 とする．

4. 無線データレートに応じた最大再送回数は図 5.1 を採用する．

5.6 実験条件

実験に用いたネットワークを図 5.12 に示す．端末および AP は，IEEE 802.11n に準拠した 5GHz 帯の無線 LAN を用いる．AP とサーバはブリッジを経由するギガビットイーサネットを通じて接続される．ブリッジは，インターネット回線の通信を模擬して遅延を追加するために使用する．

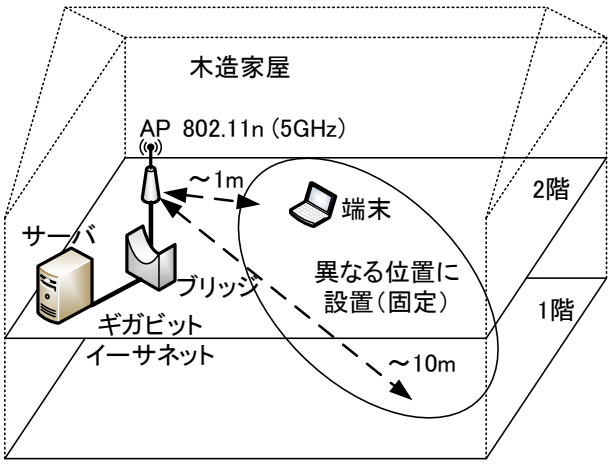


図 5.12: 実験環境

実験は木造二階建て家屋で行う．サーバ，AP，ブリッジは 2 階に設置する．端末は 1 階と 2 階の様々な場所に設置する．端末と AP との間の距離は，もっとも近い位置で約 1.2m であり，最も遠い位置で約 10m である．1 回の試行では，端末は固定して，60 秒間，端末からサーバに向けてデータを送信する．データの送受信には iperf を用いる．

端末の仕様を表 5.2 に示す．AP は Buffalo 社の型番 WZR-HP-AG300H をオリジナルのソフトウェアのまま使用する．この AP は最大 300Mbps のマルチレートをサポートしている．実験では，表 5.3 に示す 12 レベルのデータレートのすべてを使用した．

表 5.2: 端末の仕様

機種	Lenovo ThinkPad X61
Kernel	Linux 3.2.38
Linux	Ubuntu 12.04 LTS Server
無線 LAN MAC	IEEE 802.11n （5GHz）
TCP 輻輳制御	Cubic TCP
その他の TCP の設定	ECN，RED などの機能は使用せず， 送受信ソケットバッファは OS のデフォルト設定を利用
NIC	NEC Aterm WL300NC
Driver	ath9k[36]

表 5.3: 可能なデータレート

6.5	13.5	27.0	40.5	54	81
108	162	216	243	270	300

実験では，提案方式，CoDel，およびネイティブ 802.11n の性能を評価した．実験の詳細な条件は以下のとおりである．

- 提案方式は ath9k デバイスドライバに実装．
- CoDel は Linux 3.5 から実装されており，その CoDel を Linux 3.2.28 に移植して使用した．CoDel のパラメータはデフォルトの 5 ミリ秒を用いた．
- TCP バージョンは，従来の方式として TCP Reno を採用し，Linux ではデフォルトとして Cubic TCP を採用した．
- 実験では，遅延なし・ありの 2 つのケースを評価した．遅延は前述のブリッジにて挿入される．追加した遅延は，100 ミリ秒の往復遅延（片方向で 50 ミリ秒）が使用される．挿入は Linux で netem を用いて行った．
- 60 秒間の通信で以下のデータの取得を行った．
  - tcpdump を使用して端末のパケットトレース

- tcpprobe を使用して，端末での輻輳ウィンドウサイズなどの TCP 情報
- 無線 LAN デバイスドライバからデータレートなどの転送情報

これらのデータから，個々の TCP 通信のデータレートや RTT，スループットおよび cwnd の平均などの計算を行った．

- 端末 AP 間の距離と性能の関連付けは適切ではなく，その理由は距離は本研究環境でのみ意味があるパラメータだからである．一方，1 つの位置で使用されるデータレートは安定している．したがって，端末の位置を指定するパラメータとしては，TCP 通信中の平均のデータレートを使用する．結果の図のマッピングは平均データレートを横軸として用いてマッピングする．

5.7 実験結果

5.7.1 提案手法，CoDel，ネイティブの 802.11n の比較

図 5.13 に CUBIC TCP を使用して，追加遅延を付加していない場合の結果を示す．この図では，(a)，(b)，(c) の縦軸は平均スループット，平均 RTT，平均 cwnd を，横軸は平均データレートを示している．図 5.13(a) から本提案，CoDel およびネイティブの 802.11n にて同様の TCP スループットを得ているといえる．

しかし，図 5.13(b) では，ネイティブの平均 RTT が大きく，平均データレートが 30Mbps より低い場合は 100 ミリ秒となっている．CoDel の平均 RTT は平均データレートの全域にてネイティブの平均 RTT より小さい．ネイティブおよび CoDel の平均 RTT は，両対数スケールの平均データレートと線形関係にある．一方，本提案は異なる特徴を示している．平均データレートは 80Mbps を超えている区間では，提案方式では平均 RTT はネイティブと似ている．しかし，平均データレートが 80Mbps を下回る場合，本提案の平均 RTT はネイティブよりも小さくなり，CoDel のものよりも小さくなる．

図 5.13(c) に RTT の結果を示す．ネイティブは，平均データレートの全域にて，平均 cwnd が 700〜900 セグメントと大きくなっている．この大きな cwnd は過度なキューイングを発生させる．CoDel の場合，平均 cwnd は平均データレートの全ての範囲で低くなっている．これは，キューにつながれたパケットを破棄することで実現されている．逆に，

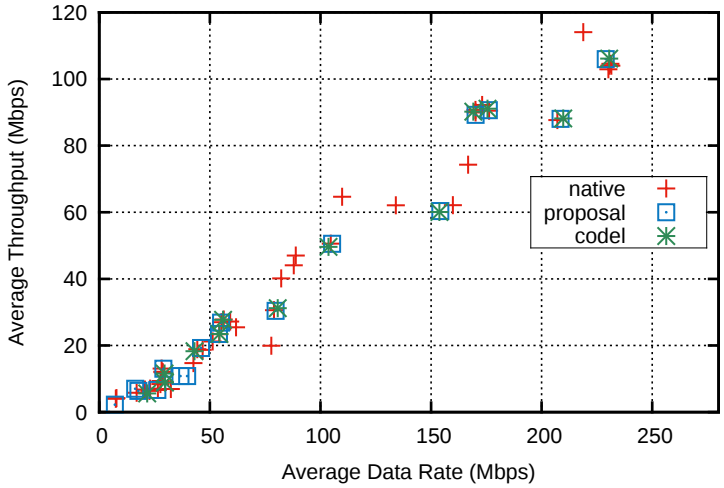
提案手法では，平均データレートが 100Mbps 以上の場合，平均 cwnd はネイティブのものと似ている．平均データレートが 100Mbps よりも小さい場合，平均 cwnd も小さくなり，40Mbps 未満では CoDel よりも小さくなる．低データレートでの MAC レベルの再送制限を抑制する提案方式は，TCP 通信に適していると考えられる．

TCP Reno を用いた場合も，追加遅延を付加していない場合と同様の結果となった．

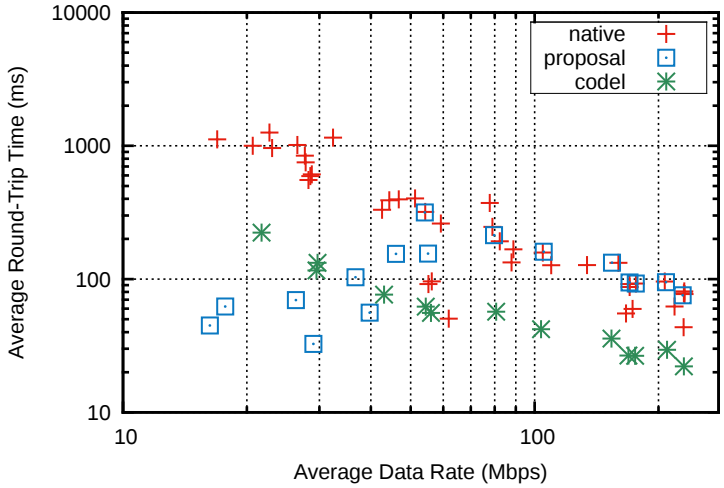
図 5.14 に，Cubic TCP を使用して，ブリッジに 100 ミリ秒の遅延を追加した場合の結果を示す．平均データレートが 100Mbps より小さい場合，平均 RTT は，ネイティブは大きな値となっており，CoDel はネイティブよりは小さくなっている．他方，提案手法は平均データレートが 80Mbps より大きい場合，ネイティブと同様の平均 RTT を有している．しかし，平均データレートが 80Mbps より小さい場合，提案手法の平均 RTT はネイティブのものよりも小さくなり，CoDel も小さくなっている．これは図 5.13 の場合と同様である．

図 5.14(c) は 5.13 とは異なる結果が得られている．遅延を追加した場合の CoDel の平均 cwnd は，遅延を追加していない場合よりも大きい．平均データレートが 100Mbpps 以上のとき，CoDel の平均 cwnd はネイティブや提案手法に似ている．これは同様にスループットをもたらしている．

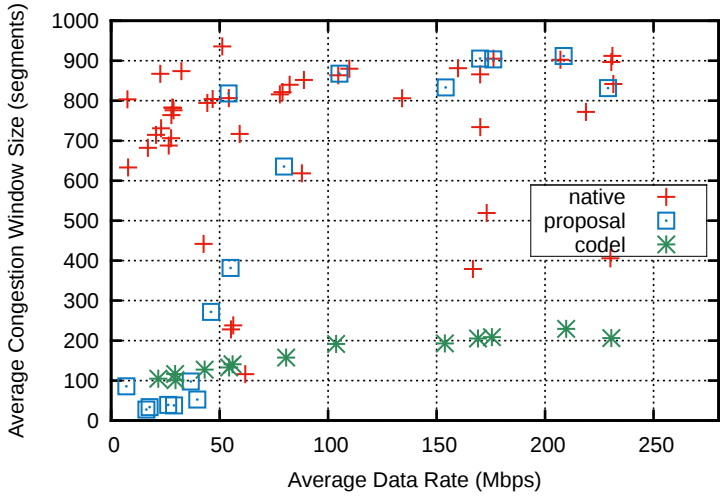
図 5.15 は TCP Reno を使用し，かつ 100 ミリ秒の追加遅延を付与した場合の結果を示している．図 5.15(b) は平均 RTT が図 5.14(b) と同様であることを示している．しかし，図 5.15(a) から，CoDel の平均スループットは平均データレート 100Mbps 以上の範囲で，他の方式よりも低くなっている．図 5.15(c) は，100Mbps 以上の平均データレートの範囲で，Codel の平均 cwnd が提案方式やネイティブよりも小さいことを示している．これがスループットが低い理由だと考えらえる．この状況を明らかにするために，図 5.16 に平均データレートが 216Mbps の場合の，スループットと cwnd の時間変化を示す．図 5.16(a) から 15 秒で CoDel のスループットが低下していることがわかる．図 5.16(b) では，そのタイミングでパケットロスが発生して TCP スロースタートが発生し，その後はゆっくりと cwnd が増加していることがわかる．この結果は，CoDel がパケットを不必要に廃棄し，適度に輻輳ウィンドウが増加しているのにも関わらず，スループットを抑制してしまう可能性があることを示している．



(a) 平均スループット 対 平均データレート

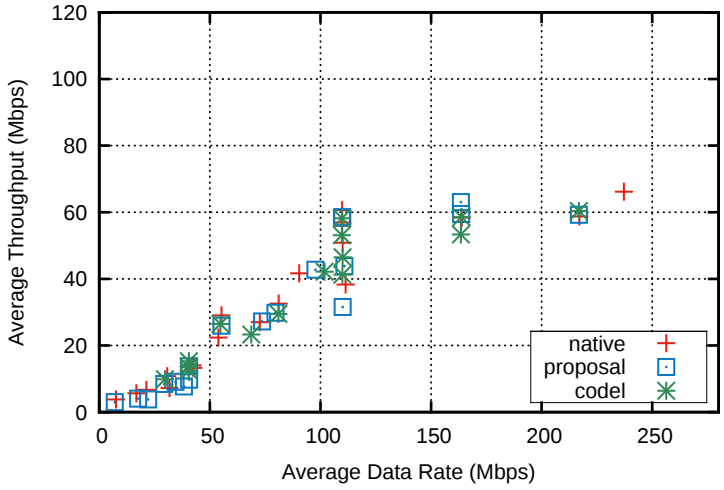


(b) 平均 RTT 対 平均データレート

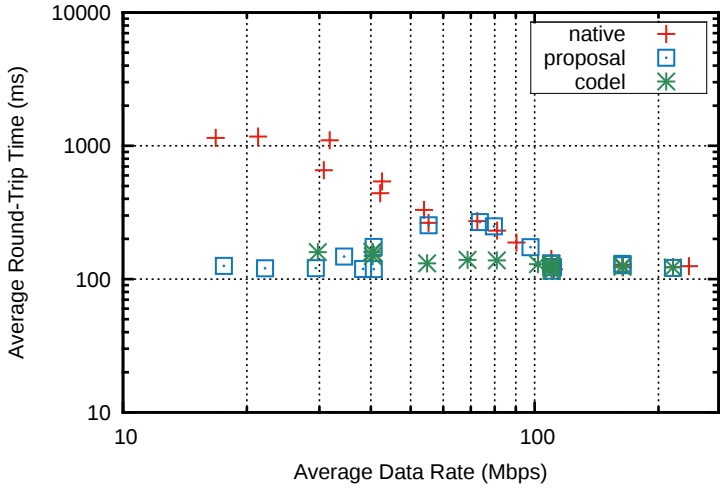


(c) 平均輻輳ウィンドウサイズ 対 平均データレート

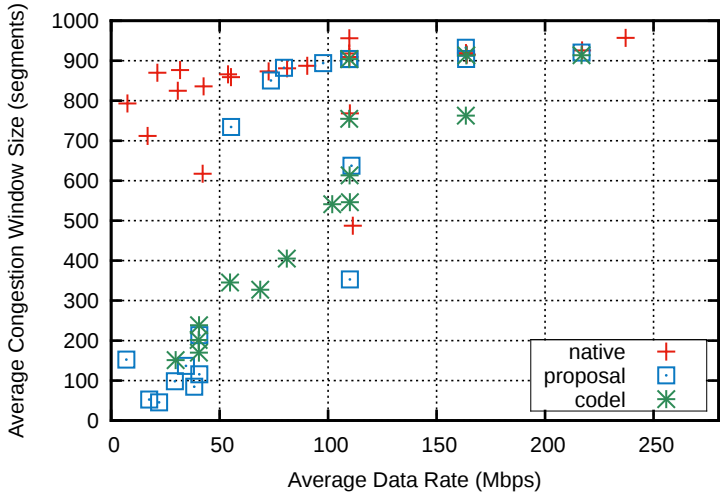
図 5.13: 追加遅延なしの場合の Cubic TCP の結果



(a) 平均スループット 対 平均データレート

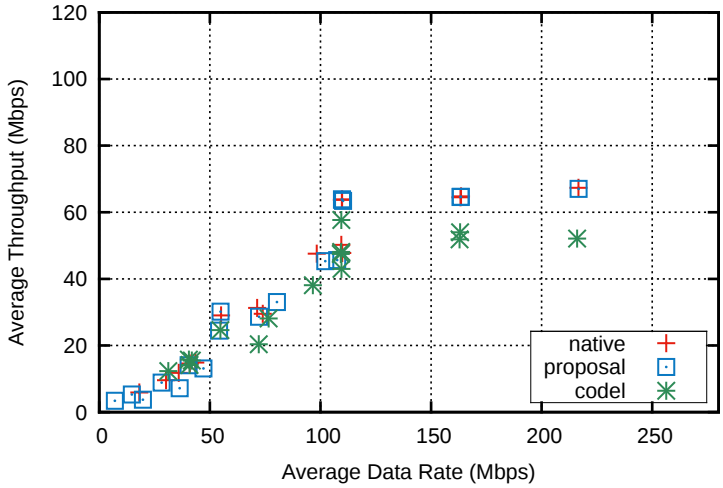


(b) 平均 RTT 対 平均データレート

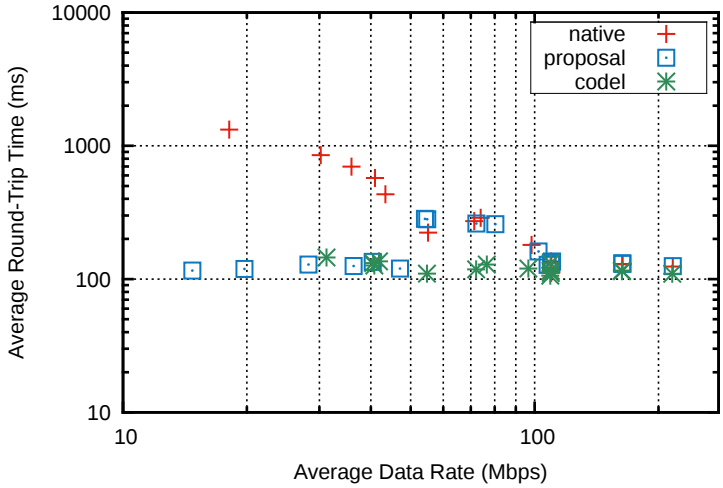


(c) 平均輻輳ウィンドウサイズ 対 平均データレート

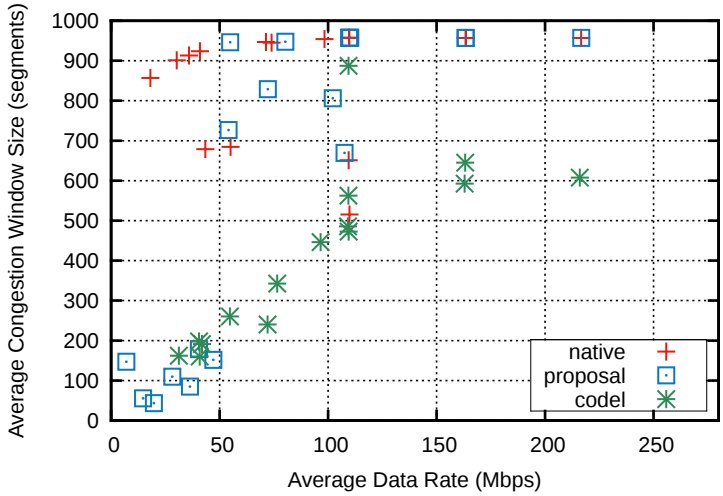
図 5.14: 追加遅延が 100ms の場合の Cubic TCP の結果



(a) 平均スループット 対 平均データレート

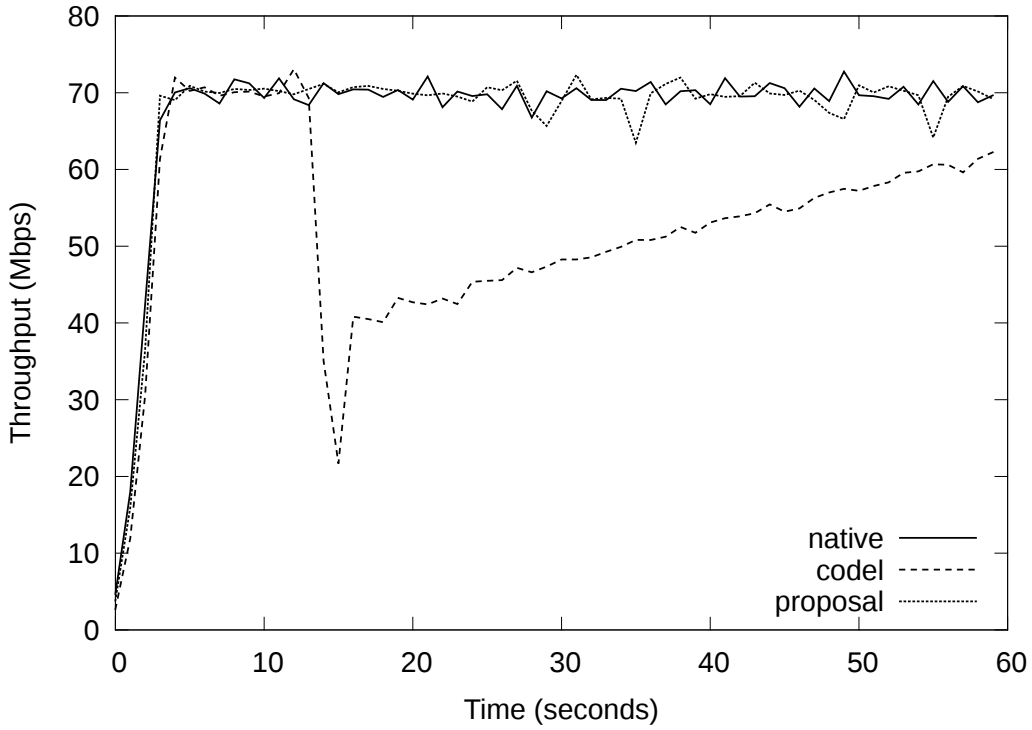


(b) 平均 RTT 対 平均データレート

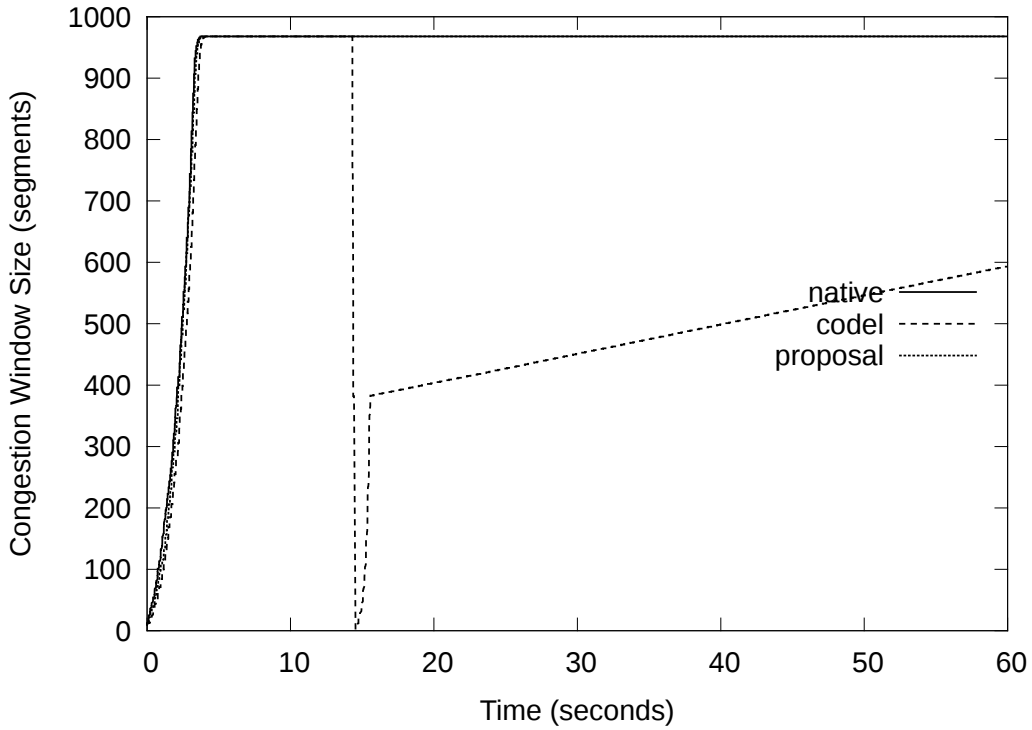


(c) 平均輻輳ウィンドウサイズ 対 平均データレート

図 5.15: 追加遅延が 100ms の場合の TCP Reno の結果



(a) スループットの時間変化



(b) 輻輳ウィンドウの時間変化

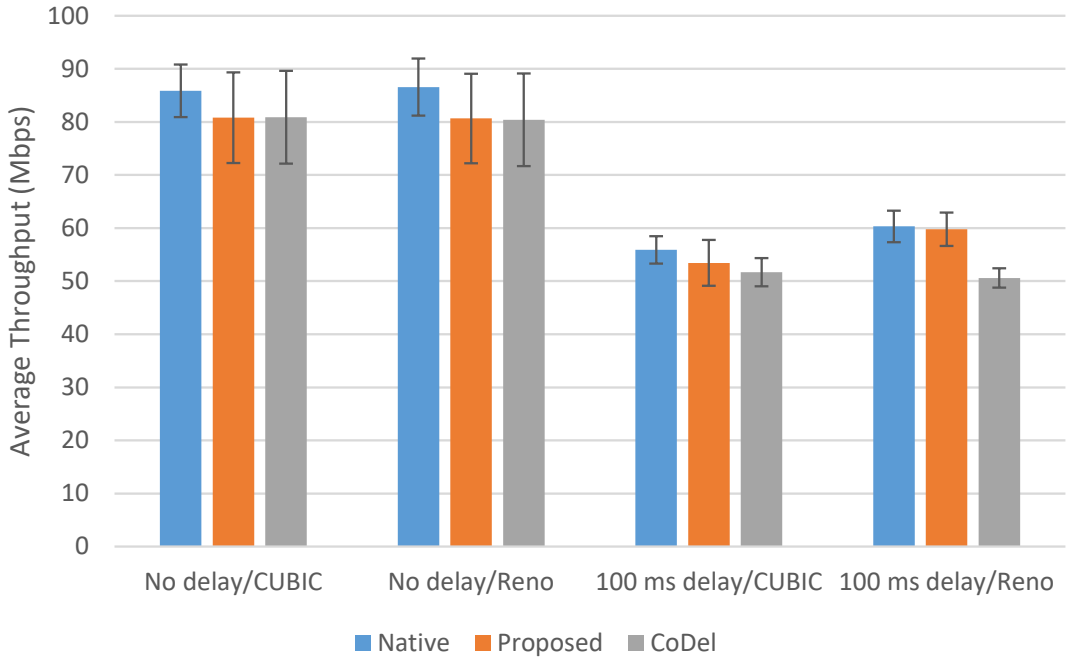
図 5.16: データレートが 216Mbps の場合の TCP Reno の個別の結果（100ms 追加遅延）

高い無線データレートの範囲におけるネイティブ 802.11n，提案方式，CoDel の性能比較を行うため，スループットとパケットロスの数を比較する．図 5.17 は，この実験の 100Mbps より高い平均無線データレートで試行された個々の TCP 通信の平均スループットとパケット損失数の平均と標準偏差を示している．図は追加遅延なし・100 ミリ秒で測定した Cubic TCP と TCP Reno の結果を示している．

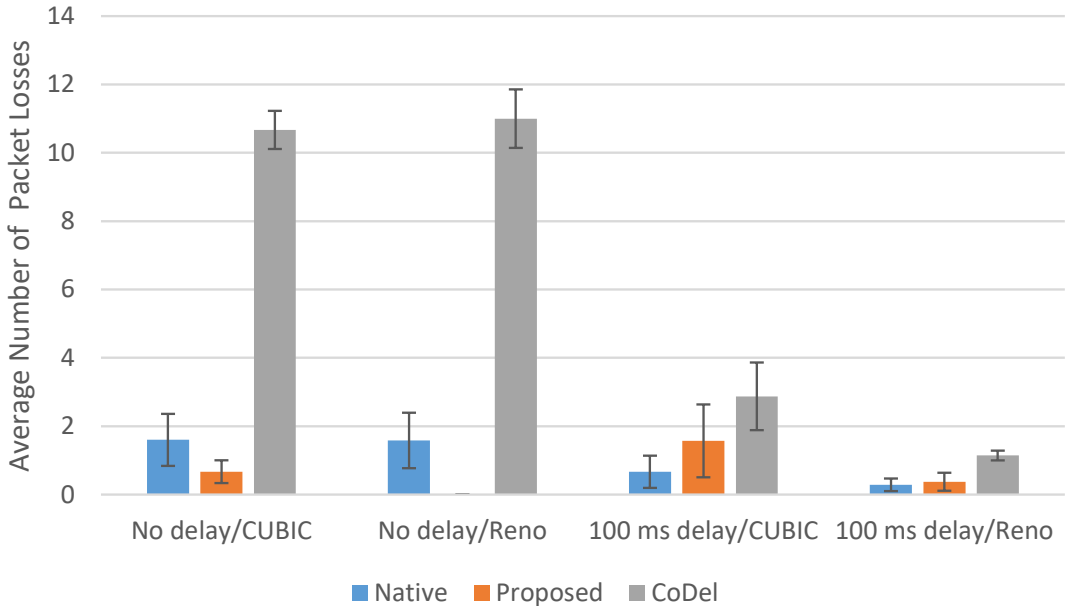
図 5.17(a) と 5.17(b) から以下のことが分かる．

- 追加遅延が導入されない場合，CoDel 方式の TCP 接続ではパケットロス数が，ネイティブの 802.11 や提案方式に比べてはるかに大きくなる．
- しかしながら，スループットはネイティブの 802.11n，提案方式，CoDel 方式において，ほぼ同じであり，CUBIC TCP と TCP Reno にて同様の傾向を示している．これは，遅延が小さい場合，損失の数と輻輳制御の起動回数がスループットに影響しないことを意味してる．
- 100 ミリ秒の遅延が挿入されると，ネイティブの 802.11n・提案方式と CoDel 方式の平均スループットが異なることが分かる．特に TCP Reno の場合，CoDel 方式ではそれ以外のネイティブの 802.11n，提案方式に比べて悪化している．
- スループットが悪化した理由は TCP 通信中のパケット損失の数である．CoDel 方式ではネイティブ 802.11n と提案方式以上のパケット損失数となっている．

これらの結果から，データレートが高い場合に CoDel 方式ではパケットを多く破棄する傾向があり，TCP Reno のように輻輳ウィンドウが穏やかに拡大される TCP バージョンの場合，スループットが低くなってしまうことを示している．



(a) 平均スループット



(b) 平均パケットロス数

図 5.17: 100Mbps より高いデータレート時の TCP 通信における平均スループットと平均パケットロス数

5.7.2 Cubic，Westwood，Vegasとの比較

図5.18にCubic TCP，TCP Vegas，TCP Westwoodがネイティブの802.11nと共に使用された場合の結果を示す．追加遅延は存在しない．図5.18(a)は平均遅延を示しているが，Westwoodの遅延は大きくBufferbloat問題を引き起こしている．Vegasは，どのデータレートに対してもRTTが10msを上回ることはなく，RTTの抑制に効果を上げている．

図5.18(b)では平均輻輳ウィンドウを示す．Westwoodはデータレート全域においてCubicを上回る平均値であるのに対して，Vegasは非常に低い値である．図5.18(c)では平均スループットを示す．WestwoodはCubicと変わらない結果となっているが，他方Vegasは，全域において非常に低いスループットを記録している．Vegasの遅延ベースによる輻輳制御が遅延の変動に敏感であり，その結果，図5.18(b)にみられた通り輻輳ウィンドウが開かれずに，スループット低下を引き起こしてしまったものと考えられる．

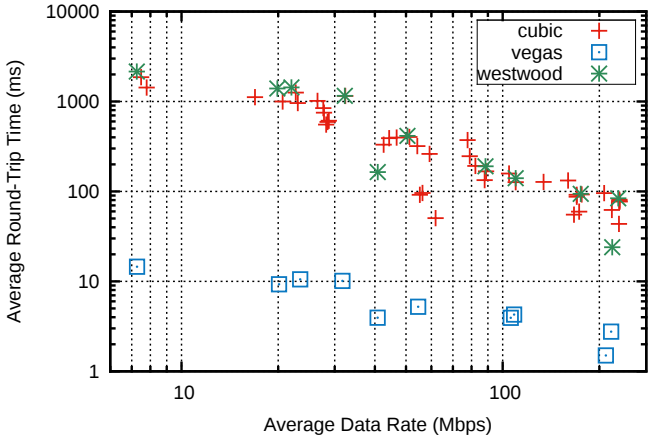
図5.19に，ブリッジに疑似遅延100msを追加して実験を行った結果を示す．図5.19(a)，図5.19(b)，図5.19(c)はそれぞれデータレートに応じた平均RTT，平均輻輳ウィンドウ，平均スループットの値となっている．Westwoodは，先ほどの遅延なしの状態とほぼ同様の傾向を示している．Vegasは先ほどの結果と少々の違いがある．図5.19(a)にて同様にデータレート全域にてRTTの抑制している様子が見られるが，図5.19(b)において著しく低かった輻輳ウィンドウは広がりを見せており，図5.19(c)においても他のCubic，Westwoodとのスループットの差は狭まっているが，依然としてスループット低下がみられる状態である．

以上の結果をまとめる．

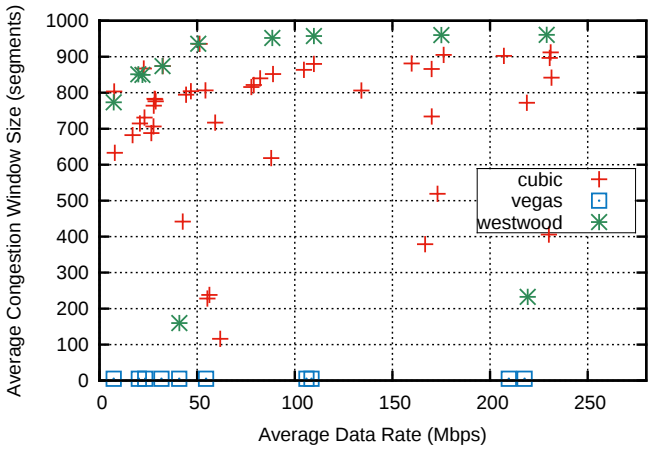
- TCP WestwoodはCubic TCPと同様の傾向を示しており，Bufferbloat問題に対して効果はない．
- TCP Vegasはデータレート全域にてRTTを抑制するが，同時にスループットの悪化がみられ，その影響は追加遅延が存在しないときに顕著である．

5.8 まとめ

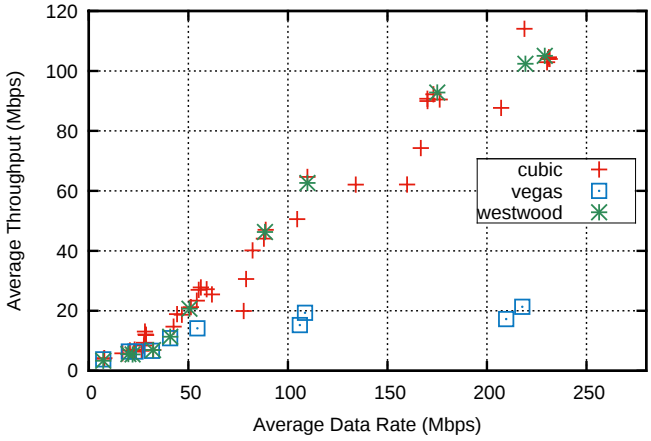
本章では端末側による提案手法の詳細な設計を行った．さらに評価を行い，提案手法



(a) 平均 RTT 対 平均データレート

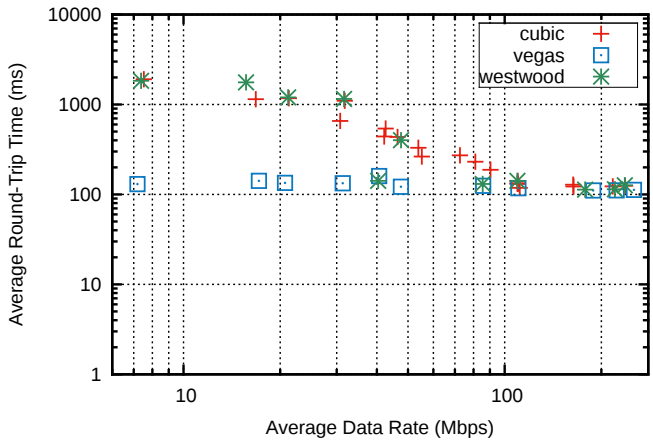


(b) 平均輻輳ウィンドウ 対 平均データレート

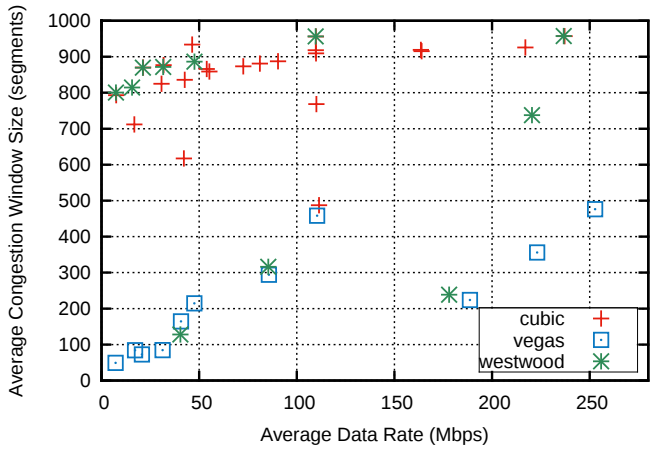


(c) 平均スループット 対 平均データレート

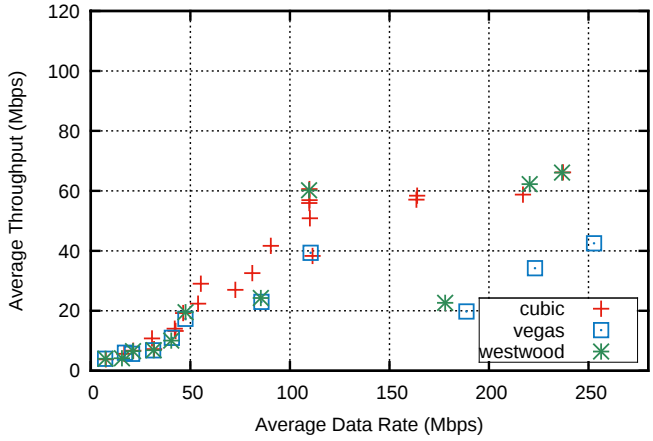
図 5.18: Cubic，Vegas，Westwood の比較結果（追加遅延なし）



(a) 平均 RTT 対 平均データレート



(b) 平均輻輳ウィンドウ 対 平均データレート



(c) 平均スループット 対 平均データレート

図 5.19: Cubic，Vegas，Westwood の比較結果（追加遅延 100ms）

がネイティブや CoDel と比較して同様のスループットを得ながらも，遅延悪化を抑制していることを示した．また，CoDel 方式においては，高い無線データレートの場合はパケット損失が多く，TCP Reno のように輻輳ウィンドウが穏やかに拡大される輻輳制御アルゴリズムではスループットが低くなることを示した．さらに，TCP Westwood や TCP Vegas についても実験を行い，TCP Westwood は Bufferbloat 問題に対して効果はないこと，TCP Vegas は無線データレート全域にて RTT を抑制するがスループットの悪化がみられることを示した．

第6章　アクセスポイント側によるデータレートに応じて再送機能を制御する手法

本章では，提案をアクセスポイントのデバイスドライバを変更して実現する方式について述べる．前章にて述べたように，端末で実現する手法について有効であることが分かった．しかし，端末にて実現する手法では，個々の異なるOS上のデバイスドライバへの対応や導入の手間があり，実運用の面で問題となる．これに対して，本章のアクセスポイントでの対応方式では，アクセスポイント上のデバイスドライバにて提案手法と同様の効果を実現することによって，アクセスポイントに所属する端末すべてのアップロード方向のBufferbloatに対応することができる．アクセスポイントのデバイスドライバにて実装を行い，性能評価を行った結果について述べる．

6.1　詳細なアプローチ

前章の端末側のアプローチでは、再送機能の低下などの提案方式について，オリジナルのデバイスドライバが持つ再送機能（BlockAckのbitmapによるソフトウェア再送）を変更することで実現している．一方，アクセスポイント側による提案方式の実現においては，受信側の処理となるため，送信側が主体となる再送機能を流用することは難しい．よって，送信側の再送機能を受信側から疑似的に低下させるための工夫が必要である．

以下に，提案方式のアクセスポイント側の詳細な実装方法を示す．

再送機能の低下 アクセスポイント側でBlock Ackによる再送を制限する場合，リトライアウトしたMPDUをBlock AckのBitmapで受信したと意図的に設定することが単純である．しかしながら，HT-immediate Block Ackによる応答はSIFS時間による

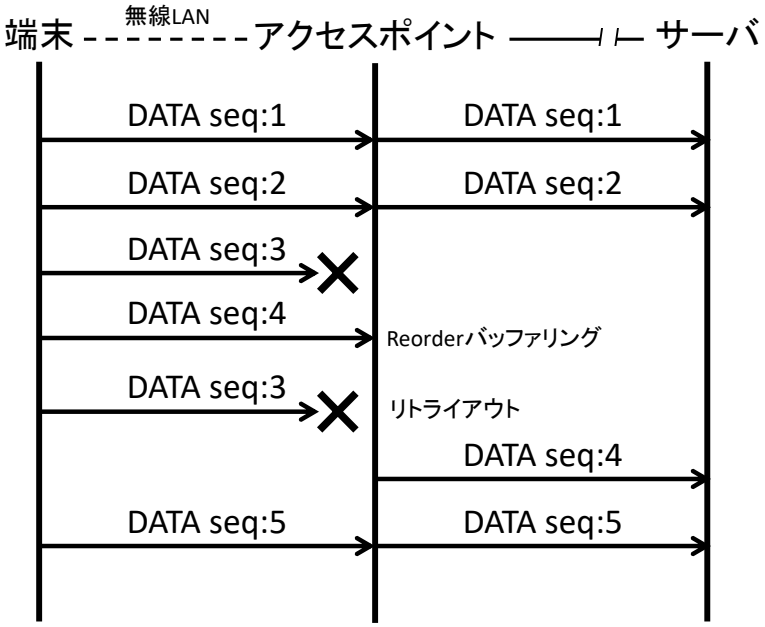
即時応答であり，一般的には無線 LAN チップ上で処理されるため，変更が容易ではない．そこで，Block Ack の仕様は変更しない．最大再送回数以上の回数の再送が送信された場合，あたかもリトライアウトしたかのように振る舞う．この疑似的な再送回数のことをリトライアウトインデックスと呼ぶ．送信側では受信側でリトライアウトしたことを知らないため，既定の最大再送回数まで再送を行う．このとき MPDU の受信に成功したとしても，リトライアウトしたとみなし，上位層への通知は行わず破棄する．ただし，受信したフレームに TCP セグメントが含まれない場合は，リトライアウト処理は行わない．

データレートの取得 A-MPDU の受信ごとに，使用されているデータレートを，指数移動平均により平滑化したものを使用する．その際の平滑化指数は 0.25 とする．

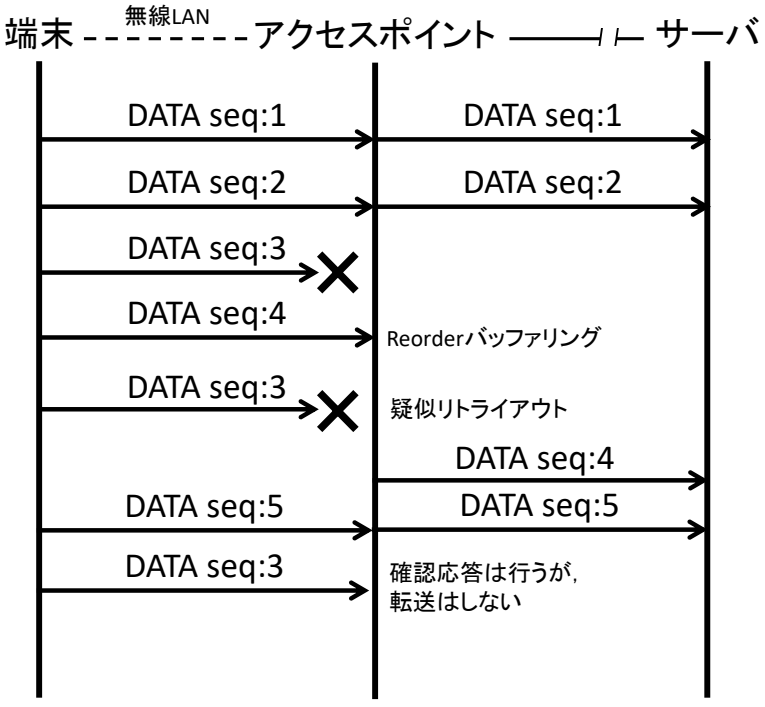
再送回数の取得 端末側で送信されたフレームの再送回数はフレームに付与されないため，フレームヘッダに含まれるシーケンス番号から推定する必要がある．推定には，受信した CRC エラーの MPDU の通知により行う．MPDU の受信時に通常，CRC エラーとなったフレームは破棄されるが，このフレームに含まれるシーケンス番号が正しいとして，対応するシーケンス番号の MPDU の再送回数を加算する．なお，この場合の推定された回数には，Block Ack フレームによる選択的再送と，A-MPDU 全体のタイムアウト再送の両方を含むことになる．

6.2 シーケンスイメージ

端末側による手法とアクセスポイント側による手法の差異を説明するため，図 6.1 に提案手法の通信シーケンスのイメージを示す．端末とアクセスポイントは無線 LAN で接続されている．アクセスポイントはブリッジとして機能し，イントラネットもしくは WAN 等を経由してサーバに接続している．最大再送回数（アクセスポイント側実装の場合はリトライアウトインデックス）は 1 に設定されているとする．端末より送信された無線 LAN データフレームにはシーケンス番号が付与されている．データフレーム 1,2 は誤りなく到達しているが，データフレーム 3 は 2 度の伝送誤りが発生しており，2 度目の伝送誤り時に最大再送回数を超えるためリトライアウトする．このとき，端末側に実装した場合，送信側端末はデータフレーム 3 を再送することなく次のデータを送信する．アクセスポイン



(a) 端末側実装方式



(b) アクセスポイント側実装方式

図 6.1: 提案手法のシーケンス例

ト側に実装した場合は，アクセスポイントで疑似的なリトライアウト処理を行い，以後，正しくデータフレーム 3 が到達したとしても確認応答に留める．アクセスポイントによる当該データの転送は行われない．確認応答が行われることで端末はデータフレーム 3 が到達したと判断し，さらなるデータフレーム 3 の再送は行われない．

このように，端末側ではリトライアウト，アクセスポイント側では疑似的なリトライアウトを用いて再送機能の低下を実現する．アクセスポイント側ではリトライアウトを受信側で知り得る情報を用いて送信側に知らせずに疑似的に行うため，送信側は疑似リトライアウトの以降も不要な再送を繰り返す可能性があり，これによって実効スループットの低下を及ぼす懸念がある．

6.3 疑似的なリトライアウト処理による影響

本章の提案方式では，アクセスポイント側による疑似的なリトライアウトを実現するために，受信側による送信側の再送回数の推定を行う．推定には CRC エラーのフレームヘッダ情報を用いる．CRC エラーであるためヘッダ情報が常に正しいとは限らない．そのためヘッダ情報中のシーケンス番号が誤ってしまう可能性があり，影響を受け得る，

ビットエラーによってフレームのシーケンス番号が誤った場合，まず CRC によって誤りとして検出され，誤った元フレームのシーケンス番号の再送回数の推定数が 1 回減少する．そして誤った先のシーケンス番号が Block Ack のウィンドウ枠内に入っている場合，別のフレームの再送回数が 1 回増加する．仮に，送信側の特定のフレームにて誤り続けた場合においても，送信側でリトライアウトするだけであり，その点においては通常のデータ転送と変わりはなく，通常の再送で誤りは回復される．このように，フレーム番号の誤りがある場合においても，通常と同等の通信は保証される．

6.4 ドライバにおける実装

図 6.2 に無線 LAN デバイスドライバにおける受信時の処理を示す．提案手法にて追加された処理は斜線部分である．無線 LAN ドライバでは，ハードウェアチップからの割り込み時に，ハードウェアから受信したフレームの情報を読み込みを試行する．受信データが存在している場合はそのまま処理を続け，存在しない場合はリターンする．データが存

在している場合の処理では，通常，物理エラーや CRC エラー，復号エラーなどの誤りが生じている場合は破棄され，誤りが存在しない場合は上位層へ通知される．提案手法では，物理エラーが生じていないフレームに対して処理を行い，その判定に応じて破棄するか否かを決定する．

6.5 動作例

図 6.3 に提案方式による AP における処理の例を示す．図に示された表の第一行は受信した A-MPDU を，第二行は各 A-MPDU に含まれる MPDU のシーケンス番号をそれぞれ示す．また MPDU の番号の上の斜線は CRC エラーであったことを意味する．例えば，A-MPDU(1) は 5 つの MPDU を含み，その内シーケンス番号 2 と 4 の MPDU が CRC エラーであったことを示している．

表のセルは，第二行で示された MPDU を受信した場合の各 MPDU（表の第一列の番号に対応）の処理の様子を示している．「√ 」は受信したデータを上位に通知することを意味し，「保留」と「無視」は受信したデータをバッファリングまたは無視することをそれぞれ示す．またセル中の数字はそのシーケンス番号の MPDU の受信回数を示す．これは推定再送回数＋１を意味する．なお，この例ではリトライアウトインデックスを 2 と仮定する．

上述のように，最初の A-MPDU(1) は 5 つの MPDU を含み，その内 2 と 4 が誤っている．このため，1 のデータは上位に通知し，3 と 5 はバッファリングする．2 と 4 に対しては受信回数を 1 とする．次に，A-MPDU(2) に対して，同様に，6 の受信回数を 1 とし，7 と 8 をバッファリングする．

次に，A-MPDU(3) は A-MPDU 全体が誤った場合である．この場合も誤った MPDU のシーケンス番号はデバイスドライバに通知されることは実験的に確認済みである．そこで，2，4，9，10 の MPDU に対して受信回数を設定する．次に A-MPDU(4) で前の A-MPDU がタイムアウト再送される．ここで 2 の MPDU が正しく受信されるため，2 とバッファリングされていた 3 を上位に通知する．また 4 と 10 の受信回数を増加させ，9 を保留とする．ここで，MPDU4 に対しては，受信回数が 3 となり，リトライアウトインデックス分だけ再送されたことになる．このため MPDU4 は紛失した処理を行う．このとき MPDU5

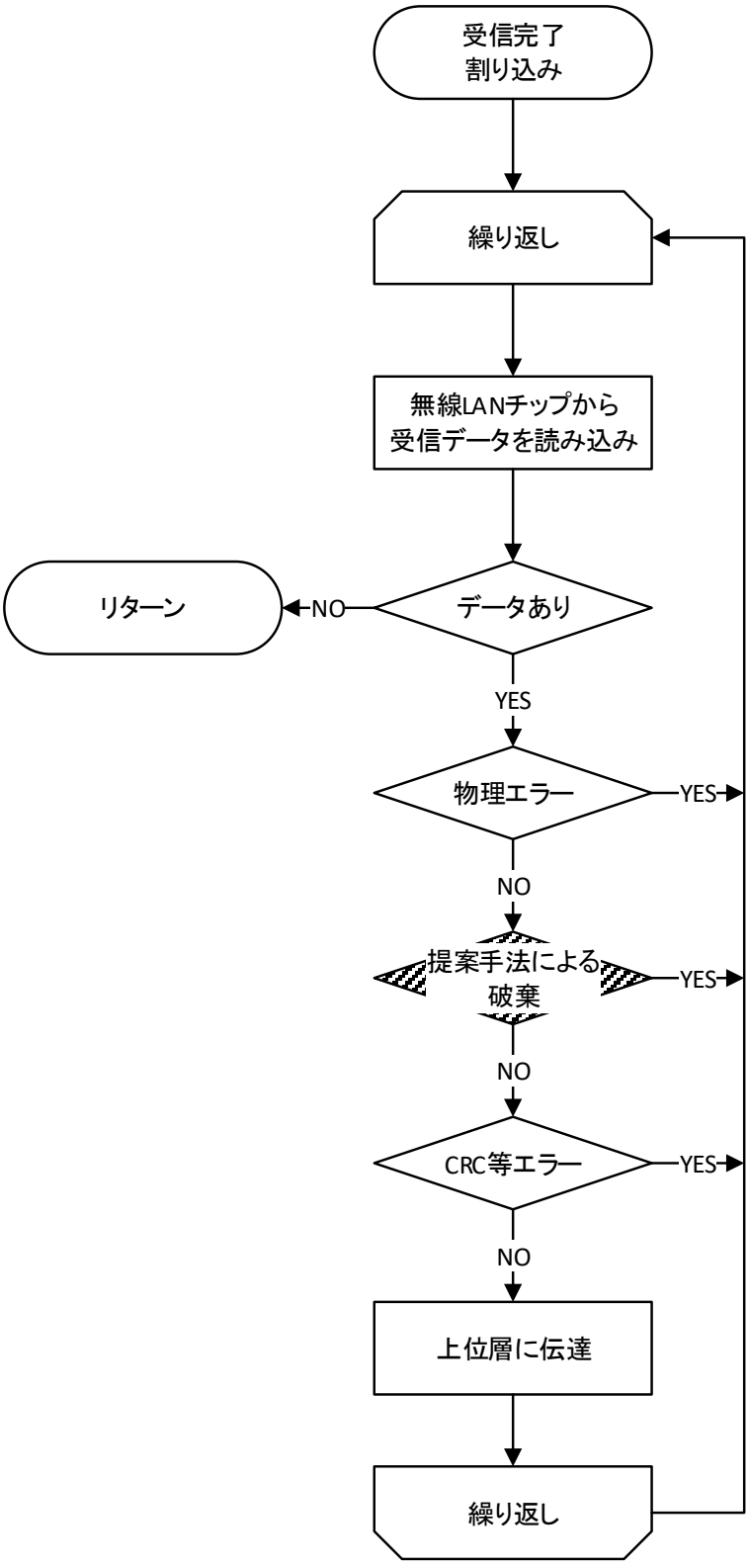


図 6.2: デバイスドライバの処理

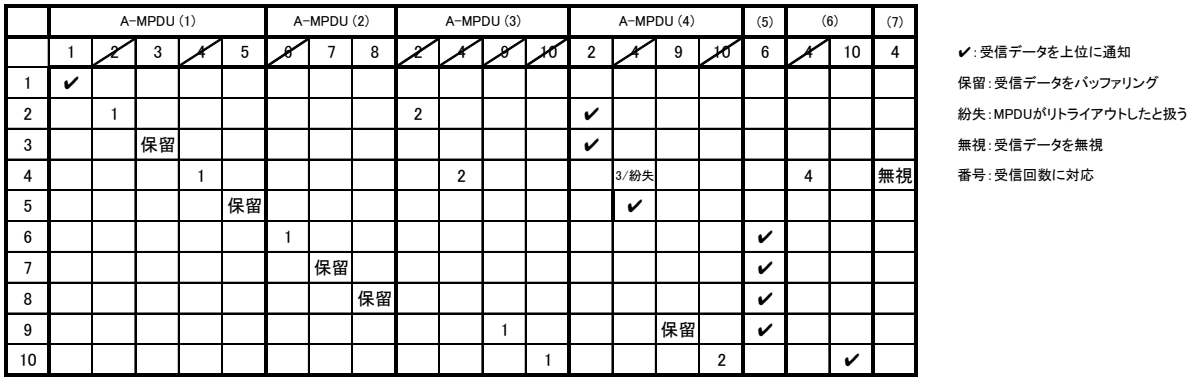


図 6.3: 提案方式による AP の処理の例

のデータがバッファリングされているため，そのデータが上位に通知される．続いて，6 の MPDU が単独で受信され，6 およびそれに続く 7 から 9 の MPDU のデータを上位に通知する．さらに，A-MPDU(6) において，MPDU4 が誤って，MPDU10 が正しく受信される．MPDU4 は紛失処理が行われているため単に受信回数を増加させる．また MPDU10 は上位に通知する．最後に，A-MPDU(7) において MPDU4 が正しく受信されるが，これに対しては，無線 LAN チップで Block ACK を返答するものの，デバイスドライバではそのデータを無視する．

このようにして，送信側の最大再送回数よりも小さいリトライアウトインデックスを受信側で設定し，早めに MPDU の紛失処理を行う．これにより，本来の 802.11n 手順にはない MPDU の紛失が発生し，TCP において輻輳制御が起動され，無線 LAN への送信負荷の軽減につながることが期待される．

6.6 評価実験

提案方式の有効性を示すために，Bufferbloat 問題が発生する環境下において TCP 通信と Ping（ICMP Echo Request/Reply）通信の実験を行い，TCP 通信のスループットと Ping 通信の RTT を評価した．具体的には，TCP 通信と送信キューを共有する Ping 通信の RTT が抑制されて Bufferbloat 問題が解決しているかどうか，また提案方式によるパケット損失によって TCP 通信のスループット性能が悪化していないかどうかについて確認を行う．併せて，すでに Bufferbloat 問題への対策を導入している端末の場合，提案方式を AP に導入した場合に性能悪化がないかどうかも確認する．

6.6.1 実験条件

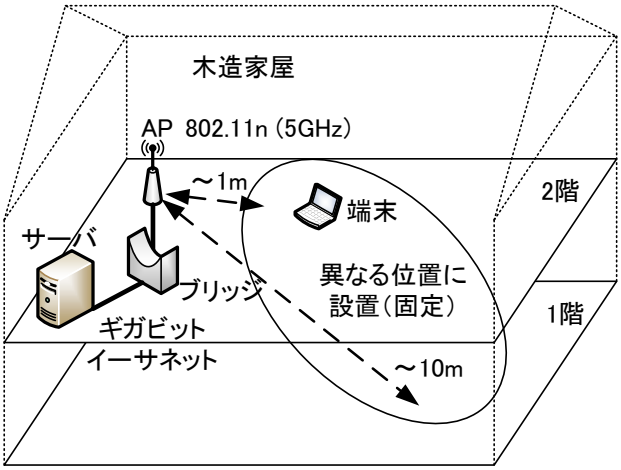


図 6.4: 実験環境

図 6.4 の環境で性能評価実験を行った．建物は木造家屋 2 階建てであり，サーバと AP は 2 階に設置した．1 台の端末は様々な地点に設置し，その場所で端末から AP に接続されたサーバに対して TCP 通信と Ping 通信を並行して行う．

実験に用いた機器の仕様を表 6.1 に示す．端末としては，Linux，Windows 端末，Mac OS 端末の 3 種類を利用した．また AP には Atheros 社製チップを搭載した無線 LAN PC カードを装備した Lenovo 社のパソコンを用いた．AP には hostapd ソフトウェアをインストールしている．

表 6.1: AP と端末の仕様

	AP	Linux 端末	Windows 端末	Mac OS 端末
機種	Lenovo ThinkPad X61		Apple MacBook Air (Mid 2012)	
Kernel	Linux 3.2.38	Linux 3.13.0	Windows 10	Mac OS 10.11
無線 LAN MAC	IEEE 802.11n （5GHz）			
TCP 輻輳制御	-	Cubic TCP	OS のデフォルト設定	
その他の TCP の設定	-	ECN，RED などの機能は使用せず， 送受信ソケットバッファは OS のデフォルト設定を利用		
AP ソフトウェア	hostapd 0.7.3	-		
NIC	NEC Aterm WL300NC		Mac 内蔵 （Broadcom）	
Driver	ath9k		OS 標準ドライバ	

なお，Linux 端末に対しては，Bufferbloat 問題の対策を施していない端末，TCP small queues を実装した端末，CoDel を適用した端末の 3 種類のバージョンを用意した．TCP

small queues を適用した端末は Linux 3.13 をそのまま利用する．Bufferbloat 問題の対策を施していない端末は Linux 3.13 の TCP small queues の制限を大きな値（10,000 パケット）にしたものを用いる．CoDel を適用した端末は，Linux 3.13 で TCP small queues の制限を大きくした上で CoDel を実装したものを利用する．なお，使用した Linux では，後述の図 6.12(a) に示すように，輻輳ウィンドウは 1,000 パケット付近に上限が設けられている．このため，TCP small queues の制限を 10,000 パケットに設定することにより，その機能を使用しないようにできる．また Linux 端末の場合は，TCP 通信における輻輳ウィンドウの値や，送信キューに接続されたパケット数（送信キュー長）など，詳細な情報が入手可能である．

表 6.2 に評価のための通信実験の詳細を示す．この表に示すように，端末の種類および位置に対して，それぞれ 60 秒間の通信を行っている．

表 6.2: 1 回の通信実験の詳細

通信時間	TCP トラヒック	Ping トラヒック
60 秒	iperf を 60 秒間使用．転送データ量は TCP スループットに依存．例えば，6Mbps, 60Mbps のスループットの場合は，それぞれ 45M バイト，450M バイト．	1 秒間に 1 回 ICMP エコー要求を送信．データは 64 バイト．1 回の通信実験で 60 回の Ping を実行．

6.6.2 全体的な実験結果

本節では全体的な実験結果，すなわち端末の種類と位置を変えた 60 秒間の通信実験における各性能の平均値について示す．ここで，端末の位置の指標として，その場所での 60 秒間の通信において転送されたすべての A-MPDU の MAC データレートの平均値を使用することとする．

まず，予備的な検討結果として，図 6.5 に提案方式を AP に導入していない場合の，端末とアクセスポイントの間の MAC レベルの平均誤り率を示す．この値は，Linux 端末から TCP 通信を行った場合の，転送された全 MPDU 数（MAC レベルの再送も含む）と AP

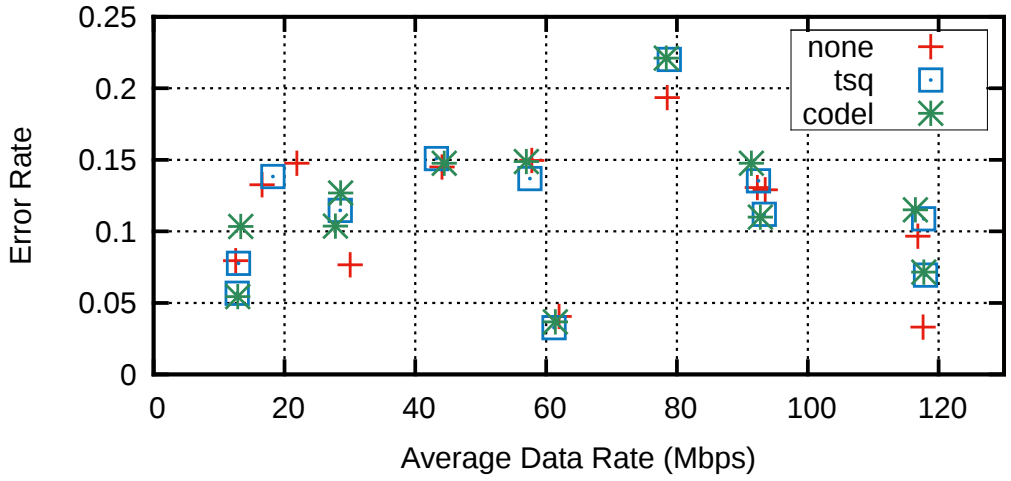
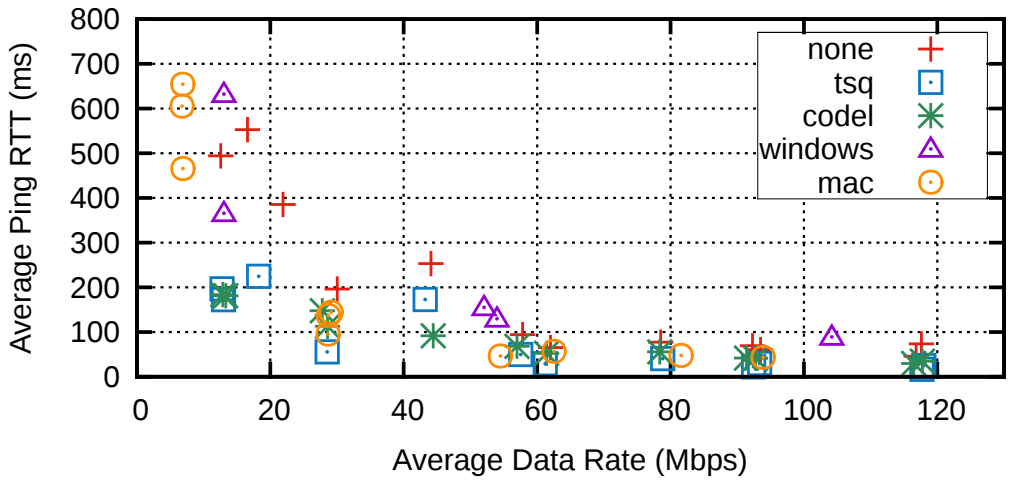
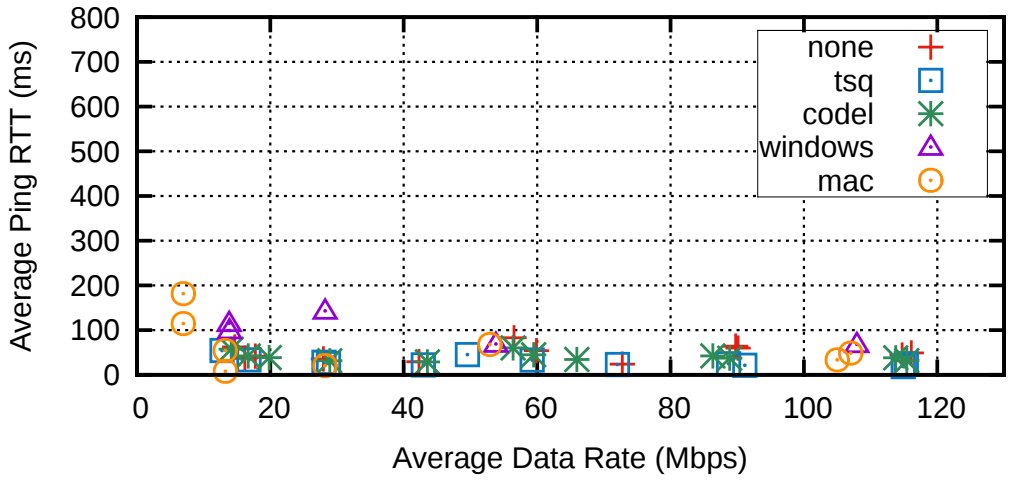


図 6.5: 平均データレートと平均誤り率の対応

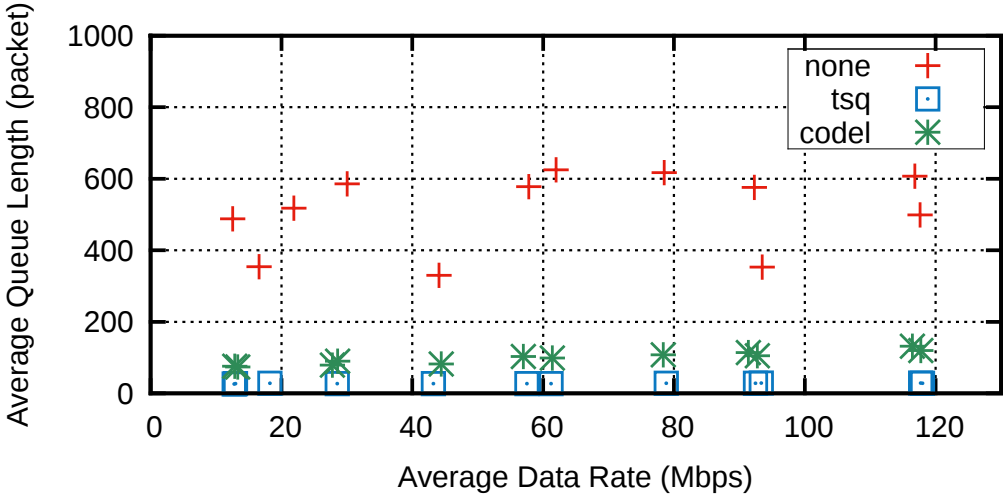


(a) 提案方式なし

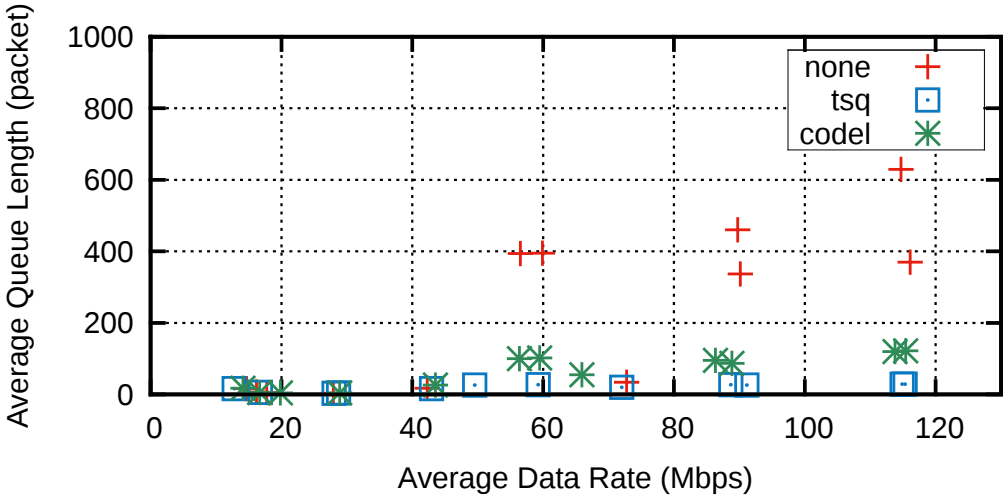


(b) 提案方式あり

図 6.6: 平均データレートと Ping の平均 RTT の対応

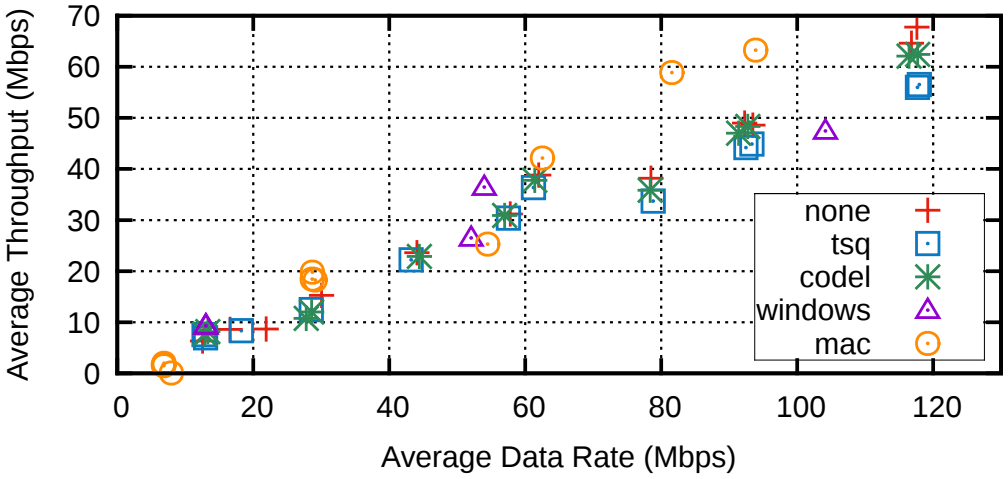


(a) 提案方式なし

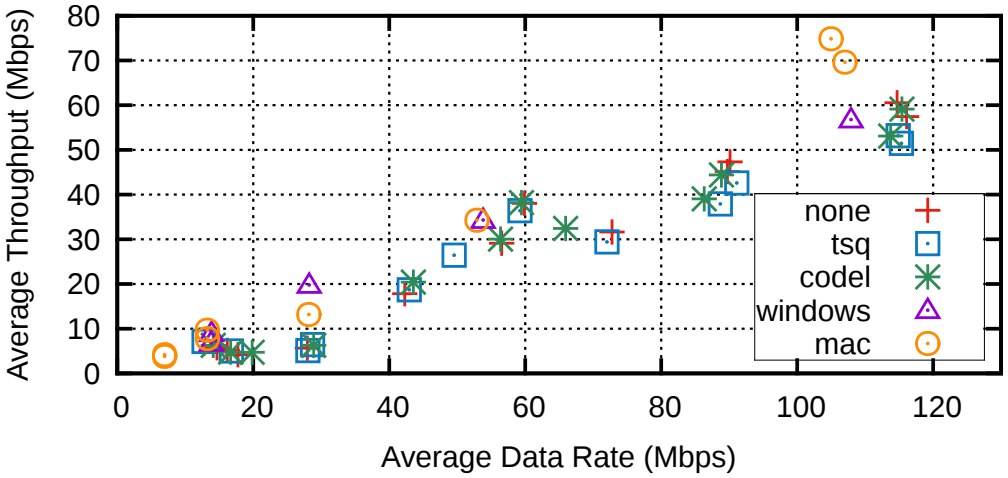


(b) 提案方式あり

図 6.7: 平均データレートと平均送信キュー長の対応



(a) 提案方式なし



(b) 提案方式あり

図 6.8: 平均データレートと平均 TCP スループットの対応

により受信確認されたMPDU数から算出したものである．図6.5は，Linux端末で3種類のバージョンを用いた場合の，測定における平均MACデータレートと，平均誤り率の対応を示している．図で，none，tsq，codelはそれぞれBufferbloat問題の対策を適用していない端末，TCP small queuesを適用した端末，CoDelを適用した端末の結果を示す．

この結果から，MACレベルの平均誤り率は0.05から0.2程度の範囲で分散していること，同じような位置（同程度の平均MACデータレート）ではLinux端末のバージョンにはあまり依存しないことなどがわかる．

次に，図6.6から6.8に具体的な測定結果を示す．各図の横軸は図6.5と同様に60秒間の測定において端末から送信されたA-MPDUのMACデータレートの平均値である．図6.6は平均MACデータレートとPing通信の平均RTTの対応，図6.7は端末の送信キューの平均長との対応，図6.8はTCP通信の平均スループットとの対応をそれぞれ示す．各図で(a)が提案方式を用いていない場合，(b)が提案方式をAPに実装した場合である．図の1つ1つのプロットは，1回の測定値に対応する．また，none，tsq，codelは図5と同様であり，windowsとmacはそれぞれWindows端末とMac OS端末の結果を示す．なお，図6.7の送信キューの平均長についてはLinux端末のみに対して測定している．

図6.6(a)では，提案方式を用いていない場合において，Bufferbloat問題に対応していないLinux 端末のPing平均RTTが，平均MACデータレートが40Mbps程度より小さい範囲で200ミリ秒を超えている．特に20Mbps以下では500ミリ秒より大きくなっている．図には示していないが，TCP通信の平均RTT（データセグメントと対応するACKセグメントの時間間隔の平均値）も測定しており，Ping通信の平均RTTの約2倍となっていることを確認している．このため，Echo Requestパケットの送信間隔によっては平均RTTが悪化することも考えられる．この結果から，端末からAP方向への通信において，Bufferbloat問題が発生していると判断できる．また，Windows端末やMac OS端末においても，平均MACデータレートが20Mbps以下で，Ping平均RTTが350ミリ秒から700ミリ秒程度に増大している．このため，Windows 10やMac OS 10.11のデフォルト設定のTCPにおいてもBufferbloat問題が発生すると考えられる．これに対して，同図のtsqおよびcodelは，平均RTTを下げており，Bufferbloat問題を抑制していることがわかる．図6.6 (b)では，提案方式により，Linux端末の3種類と，WindowsおよびMac OSの各端末において，50Mbps以下の範囲で平均RTTが抑えられている．特にBufferbloat

問題に対応していない場合の Linux 端末および Windows と Mac OS の各端末おいて改善が顕著である．これによって，提案方式を AP に実装することにより Bufferbloat 問題に対して有効に対応可能であるといえる．

図 6.7 (a) の平均送信キュー長の結果より，Bufferbloat 問題に対応していない Linux 端末は 500 から 600 パケット程度が，端末の送信キューにバッファリングされていることが分かる．これに対して，TCP small queues の Linux 端末と CoDel の Linux 端末における滞留パケット数は 150 パケット以下に抑えられており，送信キューにて滞留するパケット数が減少することによってキューイング遅延を小さくして Bufferbloat 問題を抑制することができていると考えられる．図 6.7 (b) において，平均 MAC データレートが 50Mbps 以下の範囲では，いずれの種類の端末に対しても，キュー長が 50 パケット以下に抑えられており，その結果，平均 RTT が抑えられたといえる．50Mbps を超える範囲では，Bufferbloat 問題に対応していない端末において，送信キュー長の減少があまり見られないが，RTT の増加は見られないため問題はないと考えられる．

図 6.8 (a) は，TCP 通信の平均スループットが端末の種類にはあまり依存していないことを示す．また図 6.8 (b) と図 6.8 (a) を比較すると，提案方式を導入した場合に，3 種類の Linux 端末および Windows 端末と Mac OS 端末における平均スループットは，ほぼ同様の結果を示している，

提案方式の有無における TCP 通信の平均スループットを数値的に比較する．図 6.8(a) と (b) の結果は，同じ平均 MAC データレートに対して得られているわけではないので，各端末の結果を直線で近似しその傾きの違いをもって評価することとしたい．提案方式なしの傾きに対する提案方式ありの傾きの割合で測定したスループットの割合を表 6.3 に示す．なおこの結果は測定回数の多い Linux 端末の 3 種類について示している．この結果からも提案方式を導入することによる TCP スループットの低下はほぼないといえる．

表 6.3: 提案方式ありの場合の TCP スループットの割合

	none	tsq	codel
スループット割合	97.6%	102.4%	98.3%

本節の最後に，各測定における測定値の時間的変化について言及したい．各測定は TCP 通信と Ping 通信を開始してから 60 秒間継続させたもので，TCP コネクション確立の動作なども含まれている．そこで，例として，平均 MAC データレートが約 13.5Mbps と約

110Mbps の場合における Ping 通信の RTT と TCP スループットの時間変化を図 6.9 と 6.10 にそれぞれ示す．両図では提案方式がない場合とある場合（without / with proposal）の双方を示している．これらから，Ping 通信 RTT も TCP スループットも測定の開始直後から平均値に近づいていることがわかる．このため，図 6.6 から 6.8 で得られた 60 秒間の平均値は，十分安定した値であると考えられる．

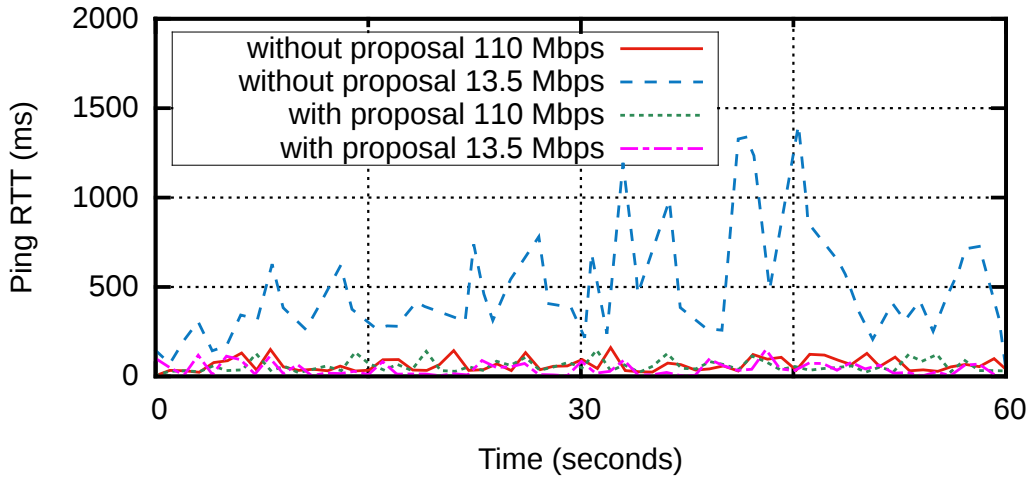


図 6.9: Ping RTT の時間変化

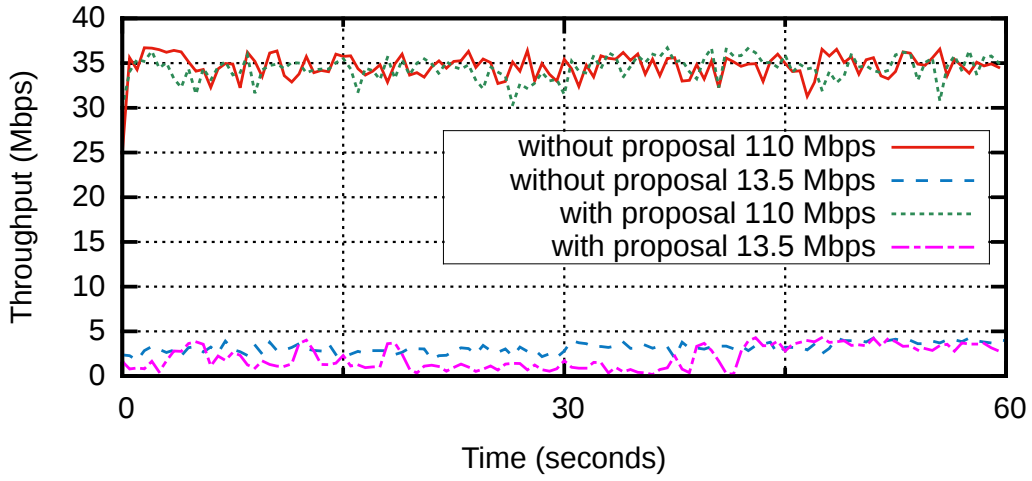
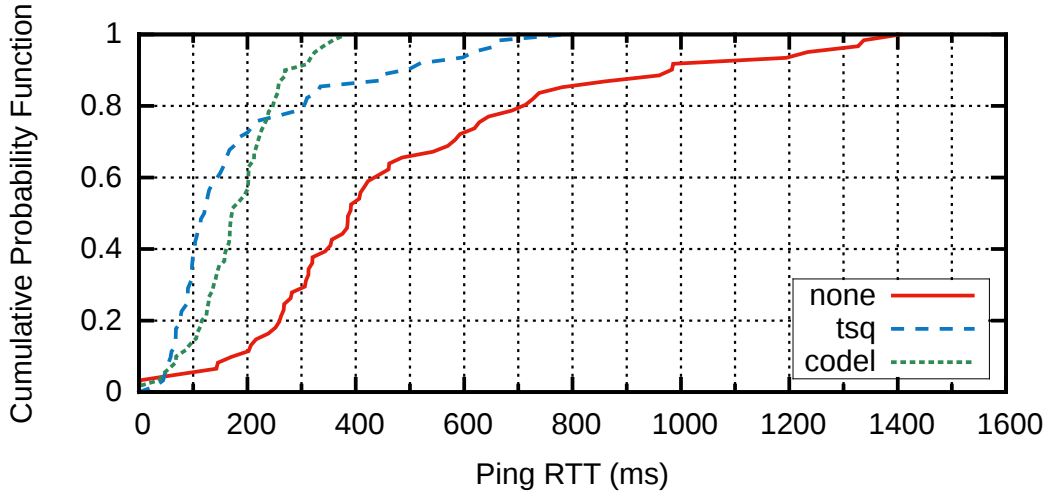
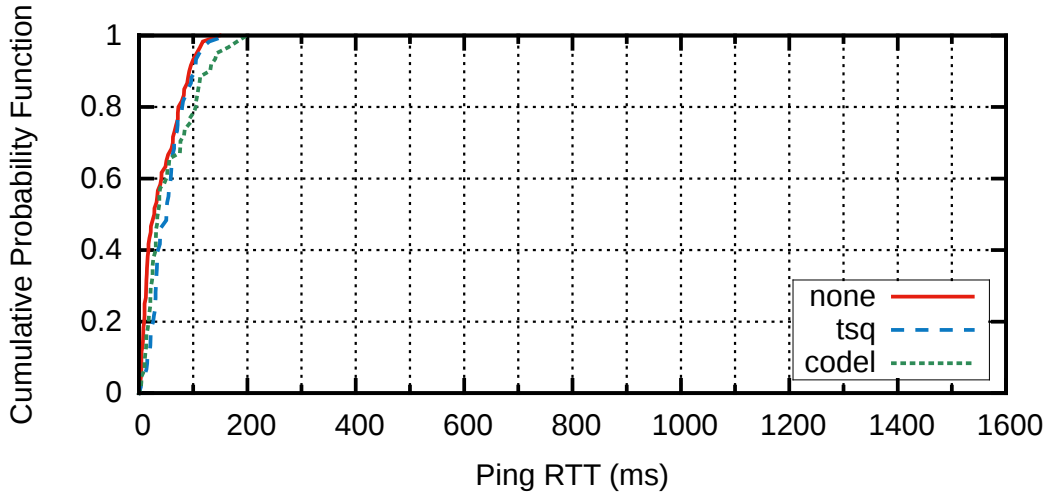


図 6.10: TCP スループットの時間変化

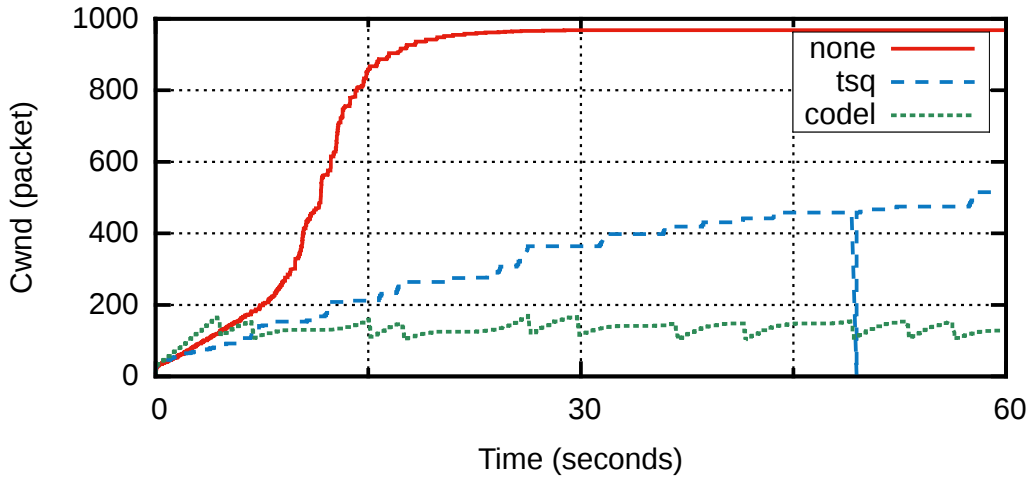


(a) 提案方式なし

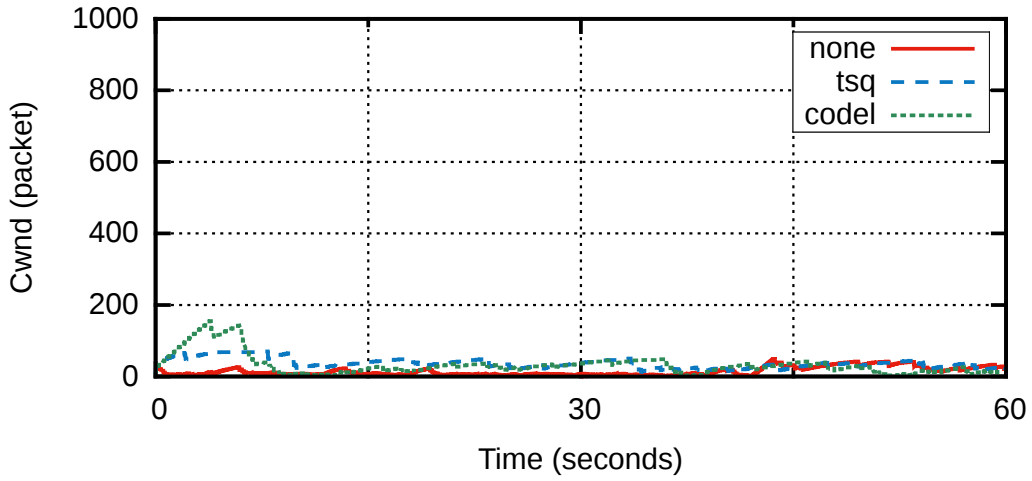


(b) 提案方式あり

図 6.11: 平均 MAC データレートが 12Mbps の場合の Ping 通信の RTT の累積分布

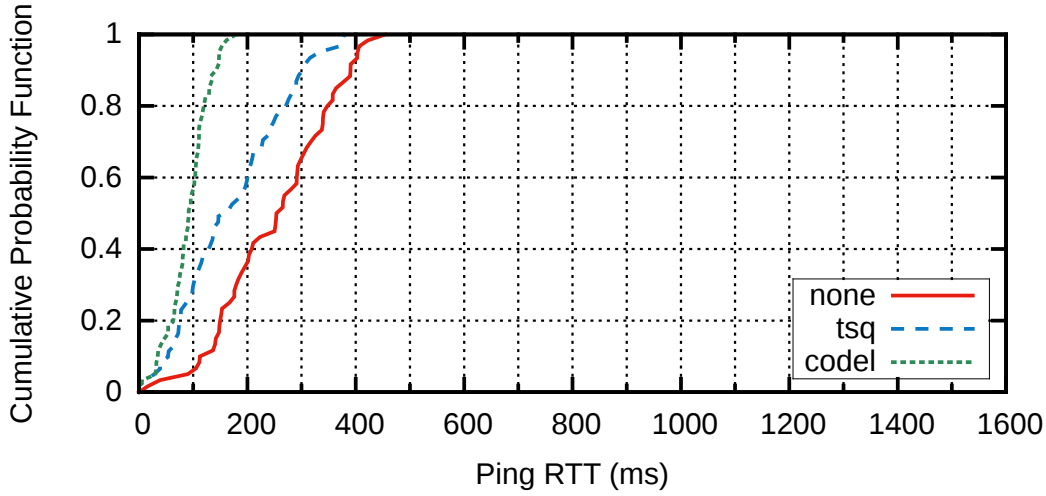


(a) 提案方式なし

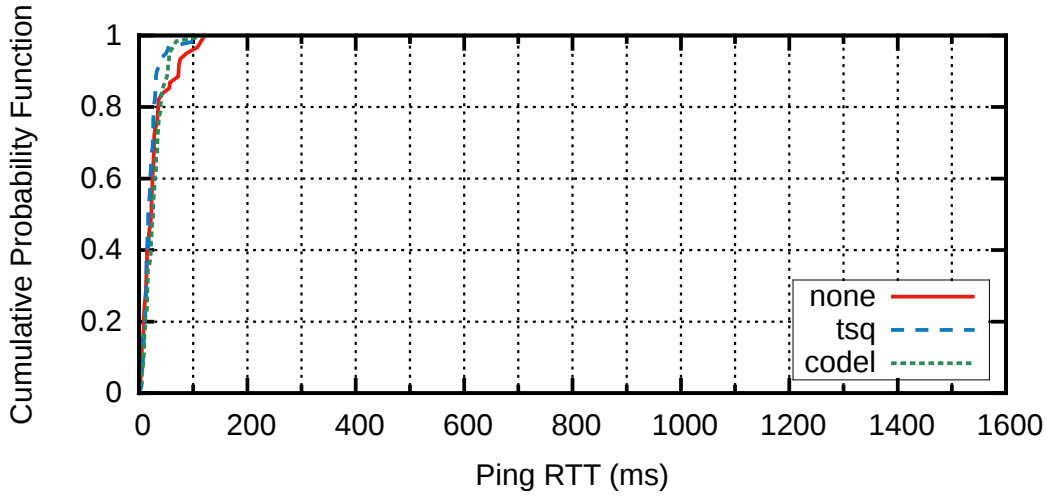


(b) 提案方式あり

図 6.12: 平均 MAC データレートが 12Mbps の場合の TCP 通信の輻輳ウィンドウの時間変化

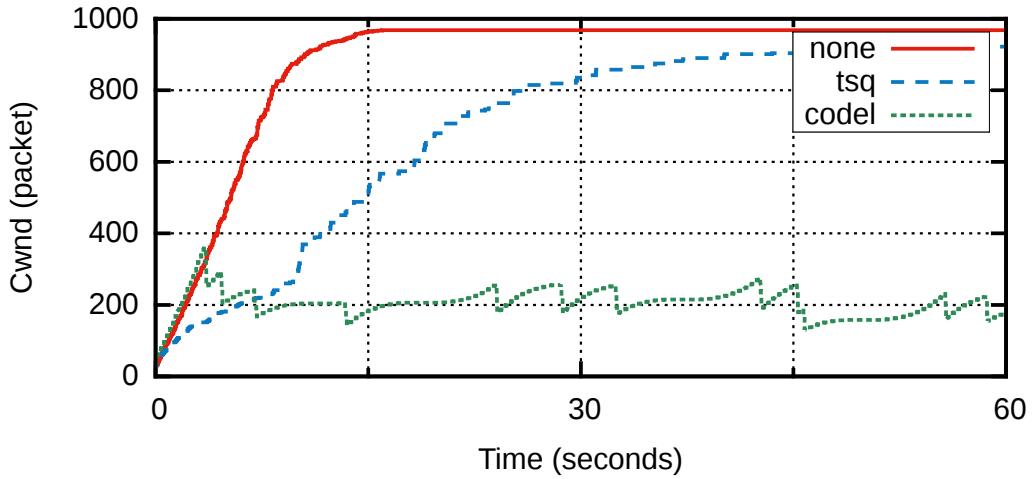


(a) 提案方式なし

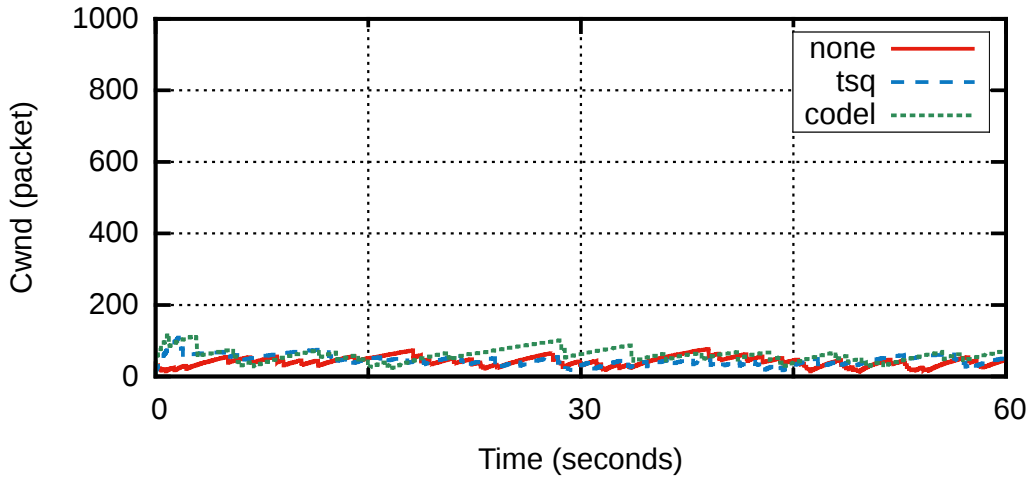


(b) 提案方式あり

図 6.13: 平均 MAC データレートが 45Mbps の場合の Ping 通信の平均 RTT の累積分布



(a) 提案方式なし



(b) 提案方式あり

図 6.14: 平均 MAC データレートが 45Mbps の場合の TCP 通信の輻輳ウィンドウの時間変化

6.6.3 個別の測定の評価

次に本節では，6.6.2 項で示した 60 秒間の個別の測定の詳細について，Linux 端末の通信を対象として述べる．図 6.11 に平均 MAC データレートが約 12Mbps だった場合の Ping 通信での個々の RTT の累積分布を示す．(a) が提案方式を用いていない場合，(b) が提案方式を AP に実装した場合である．提案方式を用いない場合，この測定では，Bufferbloat 問題に対応していない端末，TCP small queues を適用した端末，CoDel を適用した端末で，Ping RTT の平均値がそれぞれ約 500 ミリ秒，約 200 ミリ秒，約 200 ミリ秒であった．その詳細を見ると，Bufferbloat 問題に対応していない端末では 0 ミリ秒から 1400 ミリ秒まで分布していた．TCP small queues と CoDel の場合でも，それぞれ最大値が 800 ミリ秒と 400 ミリ秒であった．提案方式を AP に実装しない場合は，いずれも Ping 通信の RTT が平均値の 2 から 4 倍の値まで広がっていた．これに対して，提案方式を AP に実装した場合は，いずれの場合も Ping RTT の平均値が 100 ミリ秒以下で，分布の最大値も 200 ミリ秒よりも小さい．このためユーザレベルの応答性の悪化を解消できることが期待できる．

これらの結果は，図 6.12 に示した TCP 輻輳ウィンドウ（congestion window: cwnd）の時間変化から説明できる．なおこの図の結果は，端末において tcpprobe [12] を用いて測定したものである．提案方式を導入していない場合（図 6.12 (a)），Bufferbloat 問題に対応していない端末と TCP small queues を適用した端末では，パケットロスが発生せず輻輳ウィンドウが上昇し続ける．このため，TCP からの送信のレートが時間とともに増加し，それが内部の送信キューに蓄積され，Ping 通信の RTT 増加につながったと考えられる．CoDel を用いた端末ではそのアルゴリズムによってパケットロスが発生し，3 種類の内では輻輳ウィンドウが小さく抑えられた．これに対して提案方式を導入した場合は，図 6.12 (b) のように 3 種類ともに輻輳ウィンドウは低いままとなっている．

平均 MAC データレートが約 45Mbps だった場合の Ping 通信の RTT の累積分布を図 6.13 に示す．提案方式を使用しない場合（図 6.13 (a)），Bufferbloat 問題に対応していない端末，TCP small queues を適用した端末，CoDel を適用した端末で，Ping RTT の平均値がそれぞれ約 250 ミリ秒，約 180 ミリ秒，約 100 ミリ秒であった．分布をみるとその最大値が，Bufferbloat 問題に対応していない端末と TCP small queues を適用した端末では，双方とも 400 ミリ秒まで，CoDel を用いた端末では 200 ミリ秒程度までそれぞれ広がっている．この例でも最大の RTT は，平均値の 2 倍程度となっていることがわかる．これ

に対して，図 6.13 (b) に示すように，提案方式を AP に導入する場合は，いずれの場合も Ping RTT の平均値および最大値を低く抑えることが可能となっている．また図 6.14 に平均 MAC データレートが 45Mbps の場合の TCP 輻輳ウィンドウの時間変化を示す．これは 12Mbps の場合と同様な結果を得ている．

これらの結果から，提案方式は輻輳ウィンドウを低く保つことで Bufferbloat 問題に対処することができているといえる．

6.6.4　リトライアウトインデックスの影響

リトライアウトインデックスの値は第 5 章の表 5.1 のように定めた．5.4.1 項にて 6.5Mbps や 13.5Mbps の MAC データレートにおいては制限値を 2 回とするのが適当なことを実験的に確認し，一方，MAC データレートが 100Mbps においては最大再送回数として 10 を用いている．この補間については，図 5.11 のように階段状の値を採用した．

しかし，図の直線補間とそれに基づく階段状の値の設定は，理論的な背景があるわけではない．そこで，リトライアウトインデックスを表 5.1 の値から変更した場合の性能への影響を評価した．

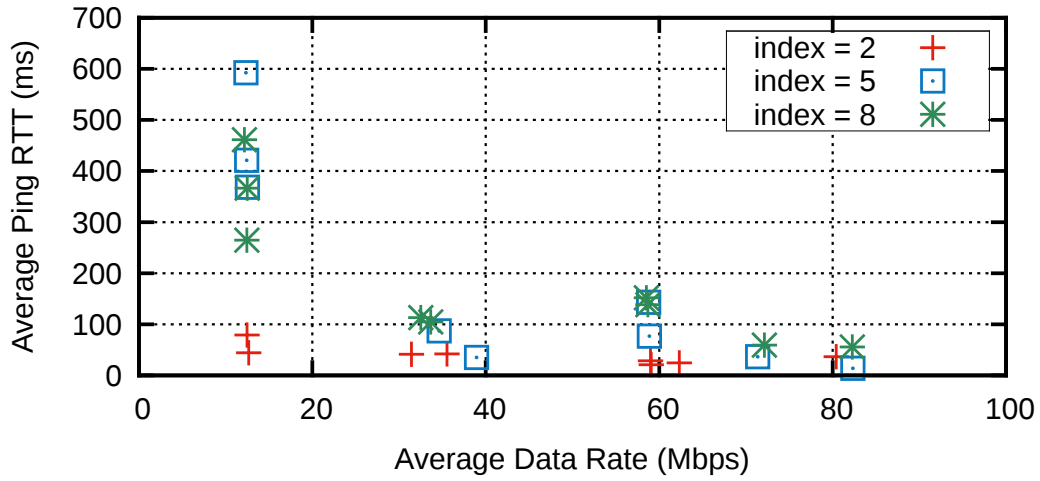


図 6.15: リトライアウトインデックスを変更した場合の Ping の平均 RTT

具体的な評価方法は次のとおりである．6.6.1 項で示した実験環境において，AP に提案方式を導入し，リトライアウトインデックスを 2，5，8 の値で固定する．Bufferbloat に対応していない Linux 端末を異なる場所に設置し通信実験を行い，Ping 通信の RTT と TCP

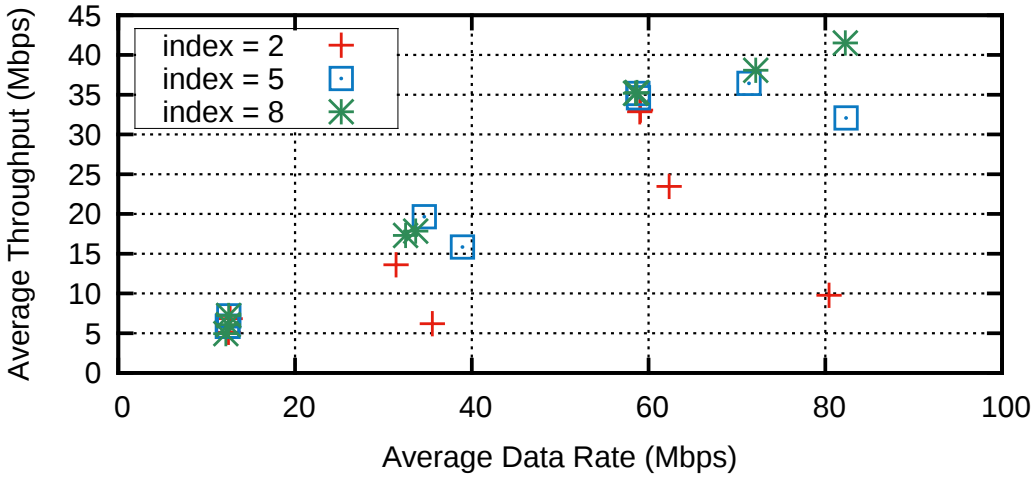


図 6.16: リトライアウトインデックスを変更した場合の平均 TCP スループット

スループットを測定する．

図 6.15 と 6.16 に測定結果を示す．図の表記は 6.6.2 節に示したものと同様である．図 6.15 では，平均 MAC データレートが 20Mbps 以下かつリトライアウトインデックスが 5 と 8 の場合，300 ミリ秒から 600 ミリ秒と Ping 通信の RTT が増加している．30Mbps から 60Mbps の平均 MAC データレートでは，リトライアウトインデックスが 8 の場合の Ping RTT が他に比べて若干大きくなっている．

一方，図 6.16 では，30Mbps 以上の平均 MAC データレートの場合において，リトライアウトインデックスが 2 の場合スループットが小さい．顕著な例では，リトライアウトインデックスが 2 かつ平均データレート 80Mbps の場合，10Mbps という著しく低いスループットを記録している．また平均 MAC データレートが 80Mbps を超えると，リトライアウトインデックスが 5 の場合も若干低いスループットを記録している．

以上の結果から，100Mbps までの MAC データレートの範囲において，表 5.1 に示したように階段状にリトライアウトインデックスを決定するのが有効であると考えられる．固定的に値を設定する場合よりも，高いデータレートではスループットを落とさず，かつ低いデータレートにおいて遅延を減少させることができる．

6.7　まとめ

本章では，アクセスポイント側による再送機能の制御のための設計を行い，評価を行った．その結果，端末側による手法と同様の効果を得られることを示した．また，Linux 以外の Windows や Mac OS などについても実験を行い Bufferbloat 問題が発生していることを確認し，提案手法を用いることで，OS 実装を変更することなしに遅延悪化を抑制できることを示した．さらに，リトライアウトインデックスの影響についても検討を行い，提案手法に適用した設定が有効であることを示した．

第7章　結論

7.1　本研究のまとめ

本論文では，IEEE 802.11n 無線 LAN 上の TCP 通信における Bufferbloat 問題の解決方法に関する研究として，無線データレートに応じて再送機能を低下させる方法を提案し，スループット性能の低下を引き起こすことなく遅延性能の悪化を抑制できることを示した．

第 1 章では，研究の背景として無線 LAN の一般への普及，アプローチと本論文の構成について示した．

第 2 章では，まず IEEE 802.11 における様々な工夫について述べ，802.11n のフレームアグリゲーションや Block ACK の必要性について説明した．次に TCP と輻輳制御アルゴリズムについて説明をし，様々な課題に対応するために輻輳制御アルゴリズムが提案されていること，輻輳崩壊に対して導入されたロスベースの輻輳制御アルゴリズムが広く採用されていることについて述べた．さらに Bufferbloat 問題が近年問題となっていることを挙げ，関連する研究を説明し，802.11n において Bufferbloat 問題が発生する可能性について述べた．

第 3 章では，802.11n 無線 LAN ネットワークにおける TCP 通信の特性を調査するため，実機による 802.11n の AP を経由するネットワーク環境を構築し，TCP によるデータ転送の実験的検討を行った．この検討によって，第 2 章で説明した Bufferbloat 問題が発生していることを，さらにその原因として 802.11n の強力な再送があることを示した．さらに，802.11n における Bufferbloat 問題の整理を行った．

第 4 章では，Bufferbloat 問題への既存アプローチの無線 LAN への適応性の検討を行った．次に無線 LAN に特化した方式提案のための基本検討の中で，無線データレートが低いほど遅延性能が悪化すること，キュー長でパケットロスを制御する場合にはスループット低下の懸念があることを示した．これらの検討から，無線データレートに応じて再送機

能の制御を行う方式を提案した．この提案方式の実現方法には端末側とアクセスポイント側の 2 つのアプローチがあるとし，さらに提案方式の実装方法に関する検討にてデバイスドライバにて現実的に実現可能であることを述べた．

第 5 章では，端末側によるデータレートに応じて再送機能を制御する方法を示し，その効果を評価した．無線データレートに応じた最大再送回数の検討を行い，提案手法に適用した．既存の手法と比較して，提案手法を用いることでスループット性能の低下を引き起こすことなく遅延性能の悪化を抑制できることを示した．さらに Westwood や Vegas などの輻輳制御アルゴリズムについても比較を行い，Westwood は従来のアルゴリズムと同様に効果はなく，Vegas は遅延悪化を抑制する効果は見られたが，一方でスループット性能の悪化がみられた．

第 6 章では，端末側の実装を変更することなく，アクセスポイント側によるデータレートに応じて再送機能を制御する方法を示し，評価した．結果，第 5 章と同様の効果を得られることを示した．また，Linux 端末に加えて，Windows 端末や Mac OS 端末についても実験を行い，提案手法を用いない場合は Bufferbloat 問題が発生すること，提案方式を用いることで遅延悪化を抑制することができることを示した．さらに，リトライアウトインデックスの妥当性について，固定のリトライアウトインデックスの値での実験を行い，第 5 章で決定したように，階段状のリトライアウトインデックスを採用するのが有効であることを示した．

本研究では，802.11n 無線 LAN における Bufferbloat 問題を解決するための方式を，第 5 章では端末側による技術として，第 6 章ではアクセスポイント側による技術として提案した．これらの提案方式は，同じ問題に対して，それぞれ異なる実現方法によって対応しており，適用したい環境や状況に合わせて選択できる．

表 7.1: 802.11n 無線 LAN における Bufferbloat 問題への対応の比較

手法		Westwood	Vegas	TCP Small Queues	CoDel	DRWA	提案手法（端末側）	提案手法（AP側）
実装の範囲	端末	TCP送信側（端末）				TCP受信側（サーバ）	端末	AP
	レイヤ	TCP	TCP	TCP	送信キュー	TCP	デバイスドライバ	デバイスドライバ
送信キューとTCPが別の端末		○	○	×	○	○	○	○
サーバへの実装が不要		○	○	○	○	×	○	○
端末の修正が不要		×	×	×	×	○	×	○
APの修正が不要		○	○	○	○	○	○	×
Cubic	追加遅延なし	×	△	○	○	−	○	○
	追加遅延100ミリ秒	×	△	−	○	−	○	−
Reno	追加遅延なし	×	△	○	○	−	○	○
	追加遅延100ミリ秒	×	△	−	△	−	○	−

第 4 章の 802.11n 無線 LAN の Bufferbloat 問題に対する検討において，様々なアプローチ手法があることを示した．表 7.1 に，第 5 章と第 6 章の結果を取り込み，それぞれの対

応の結果の比較について示す．表中の-に関しては未評価であることを示している．

まず，TCP 輻輳制御アルゴリズムの変更に関して議論する．無線に対応する Westwood と遅延ベースである Vegas の 2 つのアルゴリズムの検証を行った．Vegas への変更には一定の効果がみられたが，Westwood への変更には効果が見られなかった．Westwood は無線に対応するアルゴリズムであるとはいえ，速度低下を抑制するための制御であるため，効果がなかったものである．一方，Vegas は往復遅延時間に対応して輻輳ウィンドウの増減を行うため，Bufferbloat 問題に対して遅延抑制の効果がみられたが，逆に過敏に反応するためスループットの低下を招くことが分かった．

TCP small queues による Bufferbloat 対応は，TCP 送信者と無線 LAN 送信者が同一でなければならない，という条件付きではあるが，遅延抑制に効果があることが分かった．また，送信キューによるパケット滞留時間をパケットロスの指標とする CoDel についても，Bufferbloat 問題に効果的に働くことが分かった．一方，追加遅延が存在する場合，過度にパケットロスを発生させてしまい，高い無線データレートかつ TCP Reno の場合についてスループットの低下がみられた．DRWA に関しては，今回は評価を行っていないため，議論を見送る．

提案手法の端末側による実現方法では，端末の無線 LAN ドライバの実装を変更することで実現可能であり，Bufferbloat 問題に対して遅延抑制の効果があることが分かった．さらに，高い無線データレートにおける通信時においてもスループットの悪化は見られなかった．これは，高無線データレート時は最大再送回数を多くなるように決定しているためである．アクセスポイントの変更は許されないが，端末の変更は可能である場合に効果的である．

提案手法のアクセスポイント側による実現方法では，アクセスポイントの無線 LAN ドライバの実装を変更することで実現可能であり，そのアクセスポイントに所属するすべての端末に対して Bufferbloat 問題の対応を行うことが可能である．また，端末側で既存の Bufferbloat 対策（TCP small queues や CoDel）が行われていても，スループットの悪化を引き起こす結果は見られていない．このアクセスポイント側の手法は，端末の変更が許可されていないがアクセスポイントの変更が許可されている場合，または端末の数や種類が多く端末側の変更が容易ではない場合に効果的である．

本提案手法は 802.11n 無線 LAN における Bufferbloat 問題を中心に検討を行った．802.11n

の MAC 層の工夫である Block ACK とフレームアグリゲーションによる強力な再送によってパケットロスが発生しにくいことから，Bufferbloat 問題が発生したと考えられる．802.11n 以降の無線 LAN 規格である IEEE 802.11ac などにおいても，これら MAC 層の工夫は引き続き行われるためパケットロスは発生しにくく，802.11n と同様に Bufferbloat 問題は発生しうると考えられる．802.11ac には MU-MIMO など新しい工夫が導入されているが，1 対 1 の通信において問題の構図は変わらず，また，その解決方法も変わらないため，本提案手法は適用可能だと考えらえる．

7.2 本研究の貢献

本研究における貢献をまとめる．

7.2.1 802.11n 無線 LAN における Bufferbloat 問題の詳細な検討

今まで，Bufferbloat 問題について無線 LAN においても発生することは知られていたが，802.11n 無線 LAN において詳細に検討した研究は行われていなかった．本論文では，実際に 802.11n 無線 LAN アクセスポイントを経由するネットワーク環境を用いて，Bufferbloat 問題が発生していること，802.11n の強力な再送がその要因であることを示した．また，キュー長と遅延時間の間に関連はないことを示した．

7.2.2 802.11n 無線 LAN における Bufferbloat 問題の端末側による解決

802.11n 無線 LAN に特化した Bufferbloat 問題の解決方法として，データレートに応じて最大再送回数を制御する方法を提案して評価を行い，効果があることを示した．さらに CoDel 方式に対しても評価を行い，高い無線データレートの場合において，緩やかに輻輳ウィンドウを拡大していく TCP 輻輳制御の場合，過度なパケットロスが引き起こされることによってスループットが低下してしまうことを示した．

7.2.3　802.11n 無線 LAN における Bufferbloat 問題のアクセスポイント側による解決

前項の端末実装が変更不能である場合や多種多様な端末に対応するため，アクセスポイント側にて前節の解決方法を実現する方法を提案した．アクセスポイント側にて，再送回数の推定と疑似的にリトライアウトを行うことによって，端末側による方法と同様の効果がある Bufferbloat 解決法を提案した．再送回数の推定には，本来捨てられる CRC エラーのフレームヘッダ情報を使用するアイディアを示した．提案方式に対して評価を行い，効果があることを確認した．さらに，既存の Bufferbloat 対策方法が適用されている場合においても，提案手法の重複適用によるスループット性能の悪化がないことを示した．

7.3　今後の展望

本論文では，802.11n 無線 LAN における Bufferbloat 問題に対しての解決方法の提案を行った．通信の高速化によってスループット性能を十分に活用するために要求される送信キューのサイズは増加することになる．キューのサイズの増加によって Bufferbloat 問題により引き起こされる遅延悪化は深刻化するため，今後，この問題はより顕著化すると考えられる．本問題を前提とした，TCP 輻輳制御アルゴリズム，アクティブキュー管理，無線 LAN ドライバなどの実装が必要である．

本論文では，平均データレートを用いることで実験環境の再現性を示した．今回は既製品と Linux PC による 2 つの無線 LAN アクセスポイントの無線データレート適応アルゴリズムにおいて，本提案方式が有効であることを示した．他の適応アルゴリズムの場合においても，本提案方式が有効であるかどうかは，より詳細な検討が必要であり，今後の課題である．

今回の Bufferbloat 問題において，パケットロスによるアプローチでは，Bufferbloat 問題が発生していることの検知のための方法として，無線データレートを採用し，低いデータレートである場合は問題が生じている可能性が高いため，パケットロスが発生しやすい制御を行った．無線データレートは，有線 LAN にも適用できるように一般化すれば，そのネットワークインタフェースの持つ送信キューの実効スループットである．ボトルネックリンクの受信側において，有線 LAN，無線 LAN を問わず実効スループットは取得可能

である．問題発生検知の 1 つの手法として今後の課題である．

さらに, 無線 LAN においてはパフォーマンスアノーマリによる端末の送信速度が低下してしまう問題も存在しており研究が行われている．このような速度低下によって Bufferbloat 問題が引き起こされる可能性があるため，より詳細な検討が必要である．

第 5 章では，端末側が自然に持つ情報をもとに Bufferbloat 問題の発生の検知を行い，また再送回数の制限を行いことで意図的なパケットロスを発生させた．第 6 章では，端末側の挙動の推定，さらにはリトライアウトの疑似的な実現を行うという新たな視点を加えることで，受信側，すなわち AP による Bufferbloat 問題への対応を実現した．この新たな視点は，無線 LAN に特化しない Bufferbloat 対策にも有益であると考えている．

例えば，CoDel は，かつては送信キュー長を指標としていた RED に対して，送信キューにおける滞留時間を指標とすることによって，問題となるキューイングに対してのみパケットロスを引き起こすように改善されている．しかしながら，CoDel は AQM の発展という考え方によって，その実装箇所は送信キューのみに限定されており，自身のキューにて Bufferbloat 問題が発生した場合しか対応できない．これはすべての送信キューにて CoDel を実装する必要があることを示しており，CoDel は経路上の他の端末にて発生している Bufferbloat 問題に関しては解決することができないという課題を抱えている．

端末の実装の変更なしに経路上の端末にて発生する Bufferbloat 問題への対応を検討する場合，送信キューにおける滞留時間による Bufferbloat の検知という CoDel のアイディアと，端末側の挙動の推定を行うという本論文の提案のアイディアを融合させることで新たな制御方式の可能性を模索できる．経路全体における遅延時間の増大を，例えば IP ヘッダや TCP ヘッダの情報から推定を行い，最小遅延時間から現状の遅延時間が大きく肥大化した場合にパケットロスを引き起こすような仕組みが考えられる．途中経路の送信キューにおける滞留時間を，途中の経路にて，パッシブに検知することができれば，送信キューに限定せずに実装が可能である．

さらに, 本論文においては TCP の輻輳制御アルゴリズムはロスベースのものが支配的であるとして議論を進めたが，他の輻輳制御アルゴリズムへの考慮も考えられる．Bufferbloat 問題が問題たる原因の一つは TCP がロスベースの輻輳制御アルゴリズムを採用している点にある．ロスベースのアルゴリズムはバッファを溢れさせるまで送信速度を緩めない．一方，遅延ベースなどの場合は往復遅延時間を指標として送信速度を決定しており，

Bufferbloat の原因とはならない．遅延ベースのようなトラヒックに対してパケットロスを引き起こすのは問題であるため，峻別すべきである．TCP の輻輳制御アルゴリズムを推定する研究が行われている．そのような研究の成果を用いて，ロスベースと遅延ベースのアルゴリズムの分類を行い，ロスベースの TCP に対してのみパケットロスによる Bufferbloat 対策が行われるように限定できるようにすることも，課題である．

謝辞

本論文をまとめるにあたり，直接のご指導を頂きました電気通信大学大学院情報理工学研究科情報学専攻の加藤聰彦教授に深く感謝いたします．また，本論文の副査をご担当頂きました電気通信大学大学院の森田啓義教授，吉永努教授，藤井威生教授，大坐畠智准教授には，深く感謝申し上げます．

さらに，旧所属である電気通信大学大学院 情報システム学研究科　情報ネットワークシステム学専攻の助教から同大学院の情報理工学研究科情報・ネットワーク工学専攻に異動された策力木格准教授，また現任である山本嶺助教には研究環境の整備やご助言をいただき感謝申し上げます．

本研究を進めるにあたり，お名前を挙げることはできませんが，議論に参加して頂いた加藤研究室，大坐畠研究室の皆様に感謝に申し上げます．

最後に本論文の執筆にあたり，気にかけて頂いた友人，理解を頂いた家族に感謝申し上げます．

参考文献

[1] IEEE 802.11 Standard. *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*. IEEE, 2012.

[2] J. Postel. *Transmission Control Protocol*. RFC793. Sept. 1981.

[3] W. Richard Stevens. *TCP/IP Illustrated (Vol. 1): The Protocols*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1993.

[4] S. Floyd and T. Henderson. *The NewReno Modification to TCP's Fast Recovery Algorithm*. RFC2582. Apr. 1999.

[5] Sangtae Ha, Injong Rhee, and Lisong Xu. "CUBIC: A New TCP-friendly High-speed TCP Variant". In: *SIGOPS Oper. Syst. Rev.* 42.5 (July 2008), pp. 64–74.

[6] Saverio Mascolo et al. "TCP Westwood: Bandwidth Estimation for Enhanced Transport over Wireless Links". In: *Proceedings of the 7th Annual International Conference on Mobile Computing and Networking*. MobiCom '01. Rome, Italy: ACM, 2001, pp. 287–297.

[7] Lawrence S. Brakmo, Sean W. O'Malley, and Larry L. Peterson. "TCP Vegas: New Techniques for Congestion Detection and Avoidance". In: *Proceedings of the Conference on Communications Architectures, Protocols and Applications*. SIGCOMM '94. London, United Kingdom: ACM, 1994, pp. 24–35.

[8] Jim Gettys and Kathleen Nichols. "Bufferbloat: Dark Buffers in the Internet". In: *Queue* 9.11 (Nov. 2011), 40:40–40:54.

[9] Kathleen Nichols and Van Jacobson. "Controlling Queue Delay". In: *Commun. ACM* 55.7 (July 2012), pp. 42–50.

[10] Sally Floyd and Van Jacobson. “Random Early Detection Gateways for Congestion Avoidance”. In: *IEEE/ACM Trans. Netw.* 1.4 (Aug. 1993), pp. 397–413.

[11] モバイルワーク温故知新：無線 *LAN* に託された壮大な夢と未来 - *ITmedia* エンタープライズ. `http://www.itmedia.co.jp/enterprise/articles/1208/21/news004.html`.

[12] Wolter Lemstra, Vic Hayes, and John Groenewegen. *The Innovation Journey of Wi-Fi: The Road To Global Success*. New York, NY, USA: Cambridge University Press, 2010.

[13] IEEE 802.3 Standard. *802.3-2015 IEEE Standard for Ethernet*. IEEE, 2015.

[14] G. Xylomenos et al. “TCP performance issues over wireless links”. In: *IEEE Communications Magazine* 39.4 (Apr. 2001), pp. 52–58.

[15] Matthew Gast. *802.11n: A Survival Guide*. O’Reilly Media, Inc., 2012.

[16] Eldad Perahia and Robert Stacey. *Next Generation Wireless LANs: 802.11n and 802.11ac*. 2nd. New York, NY, USA: Cambridge University Press, 2013.

[17] J. Postel. *User Datagram Protocol*. RFC 768 (Standard). Internet Engineering Task Force, Aug. 1980.

[18] R. Fielding et al. *Hypertext Transfer Protocol – HTTP/1.1*. RFC 2616 (Draft Standard). Updated by RFCs 2817, 5785, 6266. Internet Engineering Task Force, June 1999.

[19] J. Klensin. *Simple Mail Transfer Protocol*. RFC 5321 (Draft Standard). Internet Engineering Task Force, Oct. 2008.

[20] J. Postel and J. Reynolds. *File Transfer Protocol*. RFC 959 (Standard). Updated by RFCs 2228, 2640, 2773, 3659, 5797. Internet Engineering Task Force, Oct. 1985.

[21] J. Postel and J.K. Reynolds. *Telnet Protocol Specification*. RFC 854 (Standard). Updated by RFC 5198. Internet Engineering Task Force, May 1983.

[22] T. Ylonen and C. Lonvick. *The Secure Shell (SSH) Connection Protocol*. RFC 4254 (Proposed Standard). Internet Engineering Task Force, Jan. 2006.

[23] V. Cerf, Y. Dalal, and C. Sunshine. *Specification of Internet Transmission Control Program*. RFC0675. Dec. 1974.

[24] S. Bellovin. *Defending Against Sequence Number Attacks*. RFC1948. May 1996.

[25] M. Mathis et al. *TCP Selective Acknowledgment Options*. RFC2018. Oct. 1996.

[26] K. Ramakrishnan, S. Floyd, and D. Black. *The Addition of Explicit Congestion Notification (ECN) to IP*. RFC3168. Sept. 2001.

[27] Mike Fisk and Wu-chun Feng. "Dynamic right-sizing in TCP". In: *http://lib-www.lanl.gov/la-pubs/00796247.pdf* (), p. 2.

[28] V. Jacobson, R. Braden, and D. Borman. *TCP Extensions for High Performance*. RFC1323. May 1992.

[29] S. Floyd et al. *An Extension to the Selective Acknowledgement (SACK) Option for TCP*. RFC 2883 (Proposed Standard). Internet Engineering Task Force, July 2000.

[30] V. Jacobson. "Congestion Avoidance and Control". In: *Symposium Proceedings on Communications Architectures and Protocols*. SIGCOMM '88. Stanford, California, USA: ACM, 1988, pp. 314–329.

[31] J. Nagle. *Congestion Control in IP/TCP Internetworks*. RFC 896. Internet Engineering Task Force, Jan. 1984.

[32] Keith Bostic. *4.3BSD-tahoe release*. `https://groups.google.com/d/msg/comp.sys.tahoe/50ManvdM1-s/FQqaB\_0ex-YJ`.

[33] S. Floyd, T. Henderson, and A. Gurtov. *The NewReno Modification to TCP's Fast Recovery Algorithm*. RFC3782. Apr. 2004.

[34] Matthew Mathis et al. "The Macroscopic Behavior of the TCP Congestion Avoidance Algorithm". In: *SIGCOMM Comput. Commun. Rev.* 27.3 (July 1997), pp. 67–82.

[35] Haiqing Jiang et al. "Tackling bufferbloat in 3G/4G networks". In: *Proceedings of the 2012 ACM conference on Internet measurement conference*. ACM. 2012, pp. 329–342.

[36] *ath9k Linux Wireless*. http://wireless.kernel.org/en/users/Drivers/ath9k.

[37] *tcpprobe*. http://www.linuxfoundation.org/collaborate/workgroups/networking/.

[38] Stephen Hemminger et al. "Network emulation with NetEm". In: *Linux conf au*. 2005, pp. 18–23.

関連論文の印刷公表の方法及び時期

1. 全著者名

   野元 祐孝, 策力木格, 大坐畠 智, 加藤 聰彦

   論文題名

   IEEE 802.11n 無線 LAN のアップロード TCP 通信における Bufferbloat 問題に対するアクセスポイントでの対応方式

   印刷公表の方法及び時期

   情報処理学会論文誌, Vol.58, No.2, pp.427-438 (2017.2)

2. 全著者名

   Masataka Nomoto, Toshihiko Kato, Celimuge Wu, Satoshi Ohzahata

   論文題名

   Resolving Bufferbloat Problem in 802.11n WLAN by Weakening MAC Loss Recovery for TCP Stream

   印刷公表の方法及び時期

   Proc. of the IASTED International Conference Parallel and Distributed Computing and Networks 2014 (PDCN 2014), pp.293-300, (2014.2)

3. 全著者名

   Masataka Nomoto, Celimuge Wu, Satoshi Ohzahata, Toshihiko Kato

   論文題名

   Resolving Bufferbloat in TCP Communication over IEEE 802.11n WLAN by Reducing MAC Retransmission Limit at Low Data Rate

   印刷公表の方法及び時期

   Proc. of the Sixteenth International Conference on Networks (ICN 2017), pp.69-74 (2017.4)

4. 全著者名

   Masataka Nomoto, Celimuge Wu, Satoshi Ohzahata, Toshihiko Kato

   論文題名

   A Solution for Bufferbloat Problem in Upload TCP Communication over IEEE 802.11n WLAN Only by Modifying Access Point

   印刷公表の方法及び時期

   Proc. of the Sixth International Conference on Communications, Computation, Networks and Technologies (INNOV 2017), pp.8-13 (2017.10)
   Best Paper Award

著者略歴

野元 祐孝（のもと まさたか）

平成１８年	３月	電気通信大学電気通信学部卒業
平成１８年	４月	電気通信大学大学院情報システム学研究科
		情報ネットワークシステム学専攻博士前期課程入学
平成２０年	３月	同上修了
平成２０年	４月	電気通信大学大学院情報システム学研究科
		情報ネットワークシステム学専攻博士後期課程入学
平成２８年	３月	同上を単位取得のうえ退学
平成３０年	３月	同上にて博士（工学）取得