Vol.2020-OS-150 No.13
2020/7/31

# 別経路 ACK を用いた MPTCP の公平性評価と手法の検討

安藤洸将[1]　阿部洋丈[2]　加藤和彦[3]

**概要**：インターネット全体の通信品質向上は大きな課題となっている.伝送制御で主に用いられている TCP（Transmission Control Protocol）通信は信頼性を保証した通信を実現する.TCP 通信のスループットを改善する為に提案されている技術の 1 つに MPTCP（Multi-Path TCP）があり,TCP を拡張して複数経路を単一コネクションとして扱えるようにしたものである.MPTCP は複数経路を用いる事で,TCP 通信と比較して大幅なスループット改善が見込まれる.しかし,選択した複数経路に特性差が大きく生じた場合,スループットの改善を充分に行う事が出来ない.
これを解決する為に,別経路 ACK を用いる MPTCP が提案されている.従来の MPTCP では ACK パケットがデータパケットと同一経路を用いて返送される仕組みになっているが,前述したような特性差が生じた経路選択の場合に RTT が大きくなってしまう問題がある.別経路 ACK を用いる MPTCP では ACK の経路選択を行う事で見かけ上の RTT を減少させ,スループット改善が行える事が分かっている.本研究では,この別経路 ACK を用いる MPTCP が実環境に置いて公平な通信が行えるか評価を行い,その改善手法について議論する.

**キーワード**：ネットワーク,MPTCP,スループット改善,公平性

## 1. はじめに

近年,インターネットに接続される機器の数は爆発的に増加しており,インターネット全体の通信品質向上は大きな課題となっている.ストリーミング配信等のコンテンツやその利用者も増加し,ネットワークの帯域やスループットを増加させる技術は必要不可欠である.

ソフトウェアの観点からこの問題を解決する為の技術の 1 つに MPTCP(Multi-Path TCP)がある[1].MPTCP は,インターネットで標準的に利用されている TCP 通信を拡張した技術で,複数の経路を 1 つのコネクションとして扱う事が可能である.MPTCP は実際にスループットを大幅に改善する事が可能であるが,選択した経路の特性に差が生じた場合にはスループット改善が見込めないという課題がある.

この問題を解決する為に,別経路 ACK を用いた MPTCP が提案されている.従来の MPTCP では確立したそれぞれの経路におけるデータパケットは,同じ経路を用いて返送される.特性差の大きい複数経路を選択した時,その特性に応じて ACK パケットの返送先を他の経路に変更する.これにより,RTT が大きかった経路のスループットが改善され,MPTCP が抱えていた問題を解決する事が可能となる[2].

しかし,この別経路 ACK を用いた MPTCP をインターネットのような共有ネットワークで利用する場合,既存の通信に悪影響を与えてしまう可能性が考えられる.ACK パケットが別経路を通じて返送された時,本来の経路は RTT が大きいにも関わらず,見かけ上の RTT が小さい為にデータパケットばかりが流れる事が予測される.この時,RTT が大きい経路を利用していた別の通信よりも MPTCP パケットが多く流れてしまい,帯域を占有してしまう可能性がある.

本研究では,別経路 ACK を用いた MPTCP がインターネットのような実環境において既存の通信と公平に経路を利用する事が可能か調査する.公平な利用が出来ていない場合はその原因を網羅的に調査し,改善手法について議論する.

## 2. 先行研究・既存技術

### 2.1 TCP(Transmission Control Protocol)

TCP(Transmission Control Protocol)はインターネットプロトコルスイートのトランスポート層に位置するプロトコルである.現在はインターネットにおいて標準的に利用されており,主に信頼性を重視したパケットの伝送制御を行う.

主要な機能の 1 つとして到達確認がある.これはデータパケットを受信したホストが送信元ホストへと確認応答パケット(ACK パケット)を返す機能である.TCP では ACK パケットを利用してデータパケットが到達した事を知らせる事により信頼性を向上している.

### 2.2 MPTCP(Multi-Path TCP)

TCP の抱える問題の 1 つとして,UDP と比較した際のスループットの低さが挙げられる.MPTCP(Multi-Path TCP)は TCP を拡張して複数の経路を利用する事をサポートしたプロトコルであり,RFC6824 で仕様公開されている[3].TCP 通信は 1 つの IP アドレスを用いて 1 つのコネクションを確立する事が前提となっており,複数の経路を用いる場合には同数の NIC と IP アドレスを用意する必要がある.この場合,それぞれのコネクションは独立する事になり,データの取り扱いが難しくなる.MPTCP を用いると複数コネクションのデータが 1 つのコネクションのデータとして扱える為,スループットの改善や冗長化に加えてデータの取り扱いが容易になる.

MPTCP の実例を挙げると,Apple 社の iOS が MPTCP をサポートしている.Siri では冗長化を目的とした MPTCP の

1 筑波大学
a ando@osss.cs.tsukuba.ac.jp
b habe@cs.tsukuba.ac.jp
c kato@cs.tsukuba.ac.jp

利用が iOS7 からサポートされた.また,iOS11 からはデベロッパー向けに MPTCP の API が公開され,サードパーティーのアプリケーションで MPTCP を用いる事が可能となっている[4].

### 2.3 別経路 ACK を用いた MPTCP

上述した通り,MPTCP を用いた通信では複数の経路を用いてデータの送受信を行う.MPTCP は TCP と同様に通信開始前にコネクションの確立を行うが,経路選択の手法は TCP と同様である.回線の状況によっては,選択された複数の経路の特性に大きく差が出る可能性がある.例えば図 2 のように Subflow1 の経路は RTT(Round-Trip-Time)が小さいが,Subflow2 の経路は RTT が大きいような状況が考えられる.このような場合,Subflow1 に対して Subflow2 が効率良く利用されず,MPTCP によるスループット改善がうまく見込めなくなる.

そこで,別経路 ACK を用いた MPTCP(以下 HayACK)が提案されている.通常の MPTCP では,Subflow を通して送られてきたデータパケットは,その ACK パケットは同じ経路を通して返される.これは通常の TCP 通信のプロトコルに則ったルールである.HayACK ではこの ACK パケットが通る経路を選択出来る.図 2 の例では,Subflow2 から送られてきたデータ(MPTCP Data 2)に対する ACK パケット(MPTCP ACK 2)は Subflow1 を通して返されている.これは Subflow1 の RTT が低い事を利用して Subflow2 の通信効率を向上させる為である.これにより,Subflow2 は見かけ上の RTT が小さくなり,MPTCP 全体の利用効率が向上する.HayACK では常に経路毎の RTT を計測しており,常に RTT が小さい経路を優先的に利用して ACK を返す.

### 2.4 NS-3

NS-3 は研究や教育用途での使用を目的に開発されているオープンソースのネットワークシミュレータである[5].NS-3 ではシナリオスクリプトと呼ばれるシミュレーションルールを記述したコードに従ってシミュレーションが実行される.実験ネットワークのトポロジーからノードや経路の特性まで全てをシナリオスクリプトに記述し実行する.

NS-3 の機能の 1 つに DCE(Direct Code Execution)がある[6].本来 NS-3 でシミュレーションを行った場合,ノードのマシン上のプロトコルスタックは NS-3 で準備された実験用の物が用いられる.DCE の機能を用いるとユーザーが用意したカスタムのプロトコルスタックを用いてシミュレーションを実行する事が可能となる.

本研究では主に DCE を用いてシミュレーション上のノードに HayACK をインストールし,シミュレーション検証を行う.

### 2.5 公平性の評価

インターネットのように複数のコネクションが 1 つの回線を共有して利用するネットワークの場合,単一のコネクション

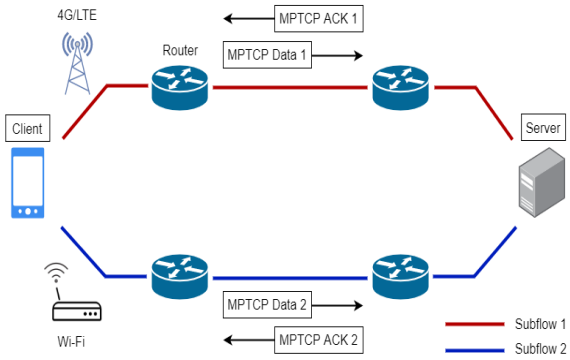



図 1 MPTCP を用いた通信の動作例

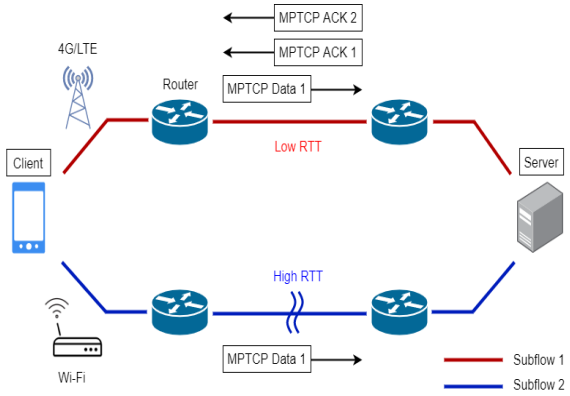



図 2 HayACK を用いた通信の動作例

が帯域を優先してしまう等の公平でない通信が行われてしまうとネットワーク全体の品質を落としてしまう事に繋がる.その為,経路を利用する全てのコネクションが公平に通信を行えているか評価する場合がある.これを公平性(fairness)の評価という.現在のインターネットでは主に TCP を用いた通信が利用されており,公平性の評価を行うケースは少ない.しかし,新規のプロトコルがインターネットで利用されるケースを想定する場合は公平性を保証する必要がある.

## 3. 調査

### 3.1 公平性の評価手法

上述の公平性を評価する為には複数の手法が既に提案されている.本調査ではこれら提案手法の中から Jain らによる Fairness Index[7]の手法を用いて公平性の評価を行う.Jain らによる Fairness Index は以下の式(1)を用いて導出される.

$$F(x_1, x_2, .., x_n) = \frac{(\sum_{i=1}^{n} x_i)^2}{n \cdot \sum_{i=1}^{n} x_i^2} \qquad (1)$$

ここで,n はコネクションの総数,x_i は各コネクションのスループットを表す.F はスループットを測定した経路の公平性の評価値であり,0.5 から 1.0 の範囲で表される.f=1.0 の時に全てのコネクションのスループットが等しい,すなわち公平であると言える.

### 3.2 シミュレーションシナリオ

本調査では前述の NS-3 によるシミュレーションを行う.本調査では HayACK がインターネット環境下において公平な通信を行えるか調査する為,HayACK と TCP が混在するネットワークを用いてシミュレーションを行う事が望ましいと考えられる.実験に使用するネットワークの構成図を図 3 に示す.

まず初めに,ノード同士の接続関係について記述する.今回の調査では,図のような MPTCP の基本的なトポロジーに,TCP 通信を行うノードを加えたようなトポロジーを用いる.HayACK の Subflow1 と TCP フローが重なるように TCP の通信を行う.このフローが重なるリンクを Link1 とする.もう一方の,HayACK のみが利用するリンクを Link2 とする.

次に,各ノードのインタフェースについて記述する.TCP 通信を行うノードの Client A および Server A には IP アドレスを 1 つずつ付与する.HayACK による通信を行うノードの Client B および Server B にはインタフェースを 2 つずつ付与し,それぞれ IP アドレスを割り当てる.

最後に,各ノードのアプリケーションについて記述する.Client A および Server A には NS-3 標準のネットワークプロトコルスタックをインストールする.Client B および Server B には HayACK を組み込んだネットワークプロトコルスタックをインストールする.中継ルータには適切なルーティングテーブルを割り当てる.評価を行う為に必要なスループットの測定を行う方法として iperf[8]を用いる.iperf はクライアント・サーバ間の通信性能を測定する為のツールで,今回は各コネクションのスループット測定の為に用いる.Server A および Server B には iperf サーバを構築し,シミュレーション開始と同時に Client A－Server A,Client B－Server B 間で iperf によりスループットを測定する.1 回のシミュレーション時間は 60[s]とする.ただし開始後 5[s]まではノードのセットアップ時間になる為,パケットの送信は行われない.シミュレーションを行うマシン環境を表 1 に示す.また,評価に用いる MPTCP プロトコルスタックを動作させる際のカーネルパラメータを表 2 に示す.

### 3.3 公平性の測定

作成したシミュレーションシナリオを用いて公平性(Fairness)の測定を行う.シミュレーションを行う上で Link1 の RTT を変化させる.HayACK は RTT の値をベースに ACK パケットの経路を選択している為,公平性の問題を観測するには有効であると考えられる.Link2 は TCP 側の通信に影響を及ぼす可能性が低いと考え,今回は RTT や帯域は変化させずに測定を行う.また今回は,シミュレーション全体(60[s])を平均してデータ収集する事よりもシミュレーションの各時点における Fairness やスループットの変化に着目する.これは,シミュレーション全体の平均からデータを収集してしまうと,問題が頻発する時点を発見しづらい為である.公平性の測定には 3.1 で示した Jain らによる Fairness Index の導出

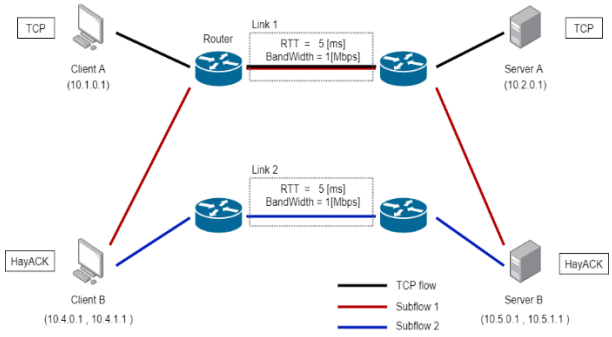



図 3 シミュレーションネットワーク

Table 1. 調査環境

| | |
|---|---|
| ホスト OS バージョン | Ubuntu 14.0.6 LTS |
| NS-3 バージョン | ns-3.26 |
| MPTCP スタックバージョン | MPTCP_v0.89 |

Table 2. カーネルパラメータ

| パラメータ | 値 |
|---|---|
| net.mptcp.mptcp_enabled(TCP) | 0 |
| net.mptcp.mptcp_enabled(HayACK) | 1 |
| net.mptcp.mptcp_scheduler | Default |
| net.core.wmem_max | 212992 |
| net.core.rmem_max | 212992 |
| net.ipv4.tcp_rmem | 4096 87380 87380 |
| net.ipv4.tcp_wmem | 4096 16384 65536 |
| net.ipv4.tcp_congestion_control | Reno[9] |

式を用いる.Link1 を通過する TCP と HayACK のデータパケット量からスループットを算出し,Fairness Index を算出する.Table1 の環境及び Table2 のパラメータを用いてシミュレーションを実行する.今回は Link 1 の RTT が 5 [ms] ,100 [ms] ,250 [ms] ,350 [ms] の場合のそれぞれにおいて測定を行った.Link1 上での TCP データパケット量の変化,HayACK データパケット量の変化,これらから導出した Fairness のグラフを Fig 4 に示す.

### 3.4 公平性測定の考察

RTT=5[ms]のように RTT が十分に小さく,HayACK が選択した 2 つの Subflow が共に同じ特性である場合,パケット送信を連続して続けても Fairness は高い値を示している事が分かる.シミュレーション時間が 20[s],52[s]付近で見られる瞬発的な Fairness の低下は,TCP の輻輳制御方式で選択した TCP Reno による制御によるものであると考えられ,その後は再度 Fairness が高い値を示す様子が窺える.

RTT=100[ms]以降では,HayACK のスループットが常に TCP のスループットを上回る結果となった.振動はあるものの RTT=5[ms]の時のようなスループットの収束は発生せず,

結果として Fairness が低い値を示した.スループットの値で比較すると最大約 4 倍の差がある時点もあることから,公平な通信が行えていない事が窺える.また,それぞれの RTT において Link2 上のパケットキャプチャから同様の測定を行った所,安定したスループット値が継続的に出ていた.この事から,Fairness 低下の主原因は Subflow1 と TCP flow に依るものであると考えられる.

### 3.5 パケットロスの調査

3.3 節では TCP と HayACK が混在するネットワークにおけるそれぞれのスループット及び Fairness の変化を測定した.RTT が大きい時に HayACK が帯域を多く使用するという結果に対して,TCP 通信でパケットロスのような問題が発生しているのではないかと考え,エラーパケットの発生状況について調査を行った.

3.3 節で得られたシミュレーション結果のデータを用いる.Link1 のパケットキャプチャのデータからエラーパケットの分布について調査する.データの比較の為,HayACK の代わりに通常の MPTCP を用いて 3.3 節と同様の測定を行った結果についても用意する.RTT についても 3.3 節と同様の 4 時点のデータを用いる

TCP-HayACK と TCP-MPTCP の組み合わせのシミュレーション結果の RTT4 時点それぞれにおいて以下のエラーパケットの分布を調査する.

- Duplicate ACK
- Fast Retransmission
- Retransmission
- Spurious Retransmission

本調査では,上記のパケットは MPTCP(HayACK)のパケットのみを図示する.エラーパケットの分布を表すグラフを Fig 5 に示す.

### 3.6 パケットロス調査の考察

TCP-MPTCP の組み合わせのネットワークを用いた場合,Duplicate ACK および Retransmission が頻発する.出現回数は多いが,周期的に発生している事が分かる.これは TCP Reno のタイムアウト周期によるものであると考えられ,輻輳ウインドウの制御が適切に行えている事が分かる.

対して,TCP-HayACK の組み合わせのネットワークを用いた場合,HayACK 側に発生するエラーパケットは少ない.これは HayACK がうまく輻輳ウインドウ制御が行われておらず,

そこで,TCP-MPTCP と TCP-HayACK の両者において,シミュレーション時間(60[s])の間のウインドウサイズの変化について調査する.RTT は 350[ms]の時のデータを用い,受信ウインドウと出力バイトをグラフにしたものを Fig 6 に示す.Fig 6 の(a)および(b)はそれぞれ,TCP-MPTCP 環境の Link1 における TCP フローと MPTCP フローの受信ウインドウサイズ変化/出力バイト数変化のグラフである.(c)および(d)はそれぞれ,MPTCP-HayACK 環境の Link1 における MPTCP フ

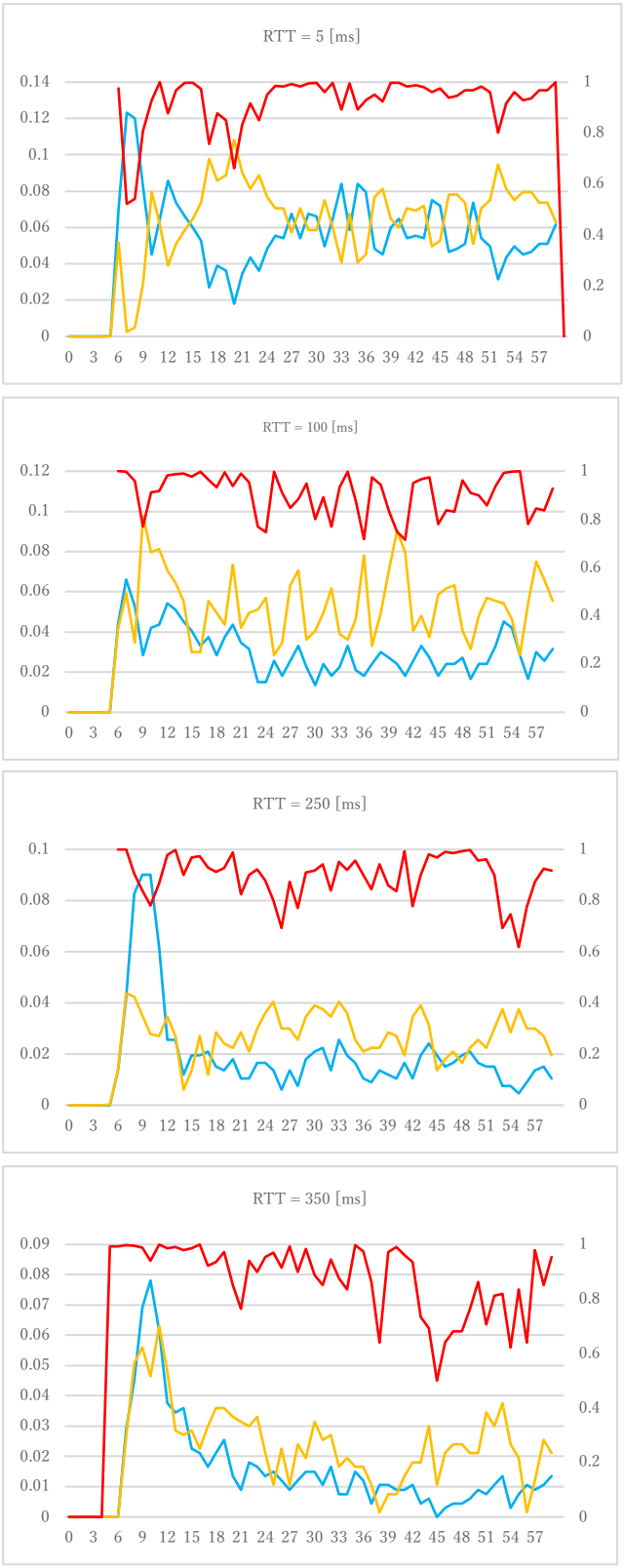



図 4 Fairness 及びスループットのシミュレーション結果
（ 右縦軸：Fairness [-] 左縦軸：Throughput [Mbps] ）
（ 横軸： Time [s] ）

ローと HayACK フローの受信ウインドウサイズ変化/出力バイト数変化のグラフである.(a)(b)(c)においては TCP Reno による輻輳制御が有効に働いている様子が観測される.しかし,(d)においてはシミュレーション初期にウインドウサイズが上限に達した後は一度もタイムアウトが発生せず,出力バイト数のみ上昇している.これは HayACK のフローが適切に輻輳制御を行えていない事と言える.

### 3.7 議論

今回行ったシミュレーション検証の結果を通して,HayACK が MPTCP と比較して帯域を占有する可能性が高い事が判明した.その理由として考えられる一番の理由はウインドウサイズの制御が適切に調整されていない点であり,この問題を解決する為には HayACK とその輻輳制御の組み合わせの適合に着目して調査を進めていく必要がある.今回用いた TCP 輻輳制御方式の 1 つである Reno はパケットロス率が高い通信を行う際にアルゴリズムレベルの不具合を抱えており,パケット破棄時にパケット送信を停止してしまう事がある.これを改善した TCP New Reno[10]を用いたシミュレーション検証について行いたいと考えている.

また,今回のシミュレーションではインターネットのような実環境を想定したネットワークモデルを用いたが,HayACK の複数フロー全てが TCP と共有している状態は十分想定される為,ネットワーク構成やシミュレーションシナリオを変更しながら追加検証を行っていきたい.

## 4. 結論・今後の課題

本研究では先行研究である別経路 ACK を用いた MPTCP の中で行われた公平性調査を更に網羅的に行った.先行研究中ではシミュレーション全体を通した Fairness の測定を行う事で公平性の高さを示していたが,本研究ではシミュレーションの各時点における Fairness を測定する手法を取る事により,改めて TCP 通信と HayACK の Fairness の低下の問題を取り上げた.また,スループットと Fairness の変化及びパケットロスの調査により,HayACK の輻輳制御が有効でない事の問題を提示した.

本研究では TCP/HayACK の輻輳制御方式として TCP Reno を用いたが,今後の方針としてまずはあらゆる制御方式の組み合わせで正しく動作するパターンが存在するか等の具体的な調査を行う必要があると考えている.そして,正しく動作しない輻輳制御方式との違いについて調査し,対策が必要となれば HayACK の新たな制御方式を検討していく必要がある.そして,最終的に対策を行う事で HayACK の実環境での有用性を高めていく.

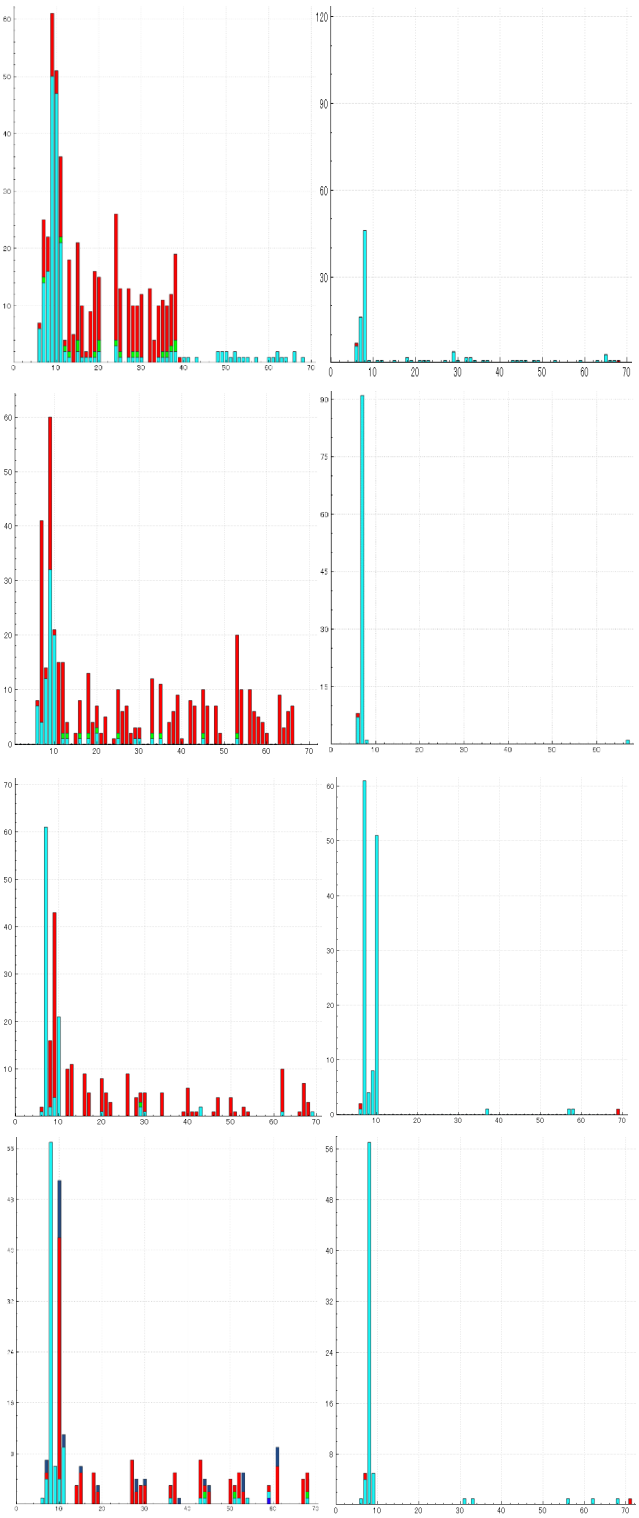
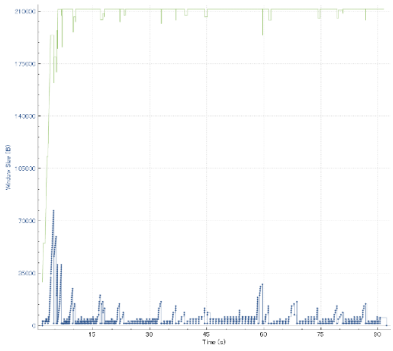


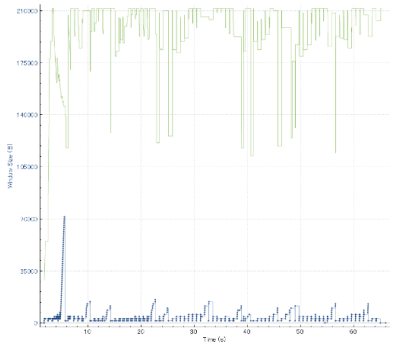
図 5 エラーパケット分布グラフ
（ 横軸 : シミュレーション時間[s] ）
（ 縦軸 : パケット出現回数[-] ）
: Duplicate ACK
: Fast Retransmission
: Retransmission
: Spurious Retransmission

## 参考文献

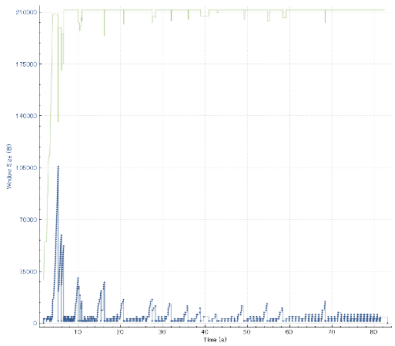
[1] mptcp , https://www.multipath-tcp.org/.
[2] Y. Morikoshi, H. Abe and K. Kato, “HayACK: Exploiting Characteristically Diverse Paths to Achieve Quick ACKing in MPTCP," 2017 IEEE International Conference on Cloud Computing Technology and Science (CloudCom), Hong Kong, 2017, pp. 383-390.,” 2017. “科学技術情報流通技術基準 参照文献の書き方 (SIST 02)”. http://jipsti.jst.go.jp/sist/pdf/SIST02-2007.pdf, (参照 2018-12-02).
[3] TCP Extensions for Multipath Operation with Multiple Addresses, https://tools.ietf.org/html/rfc6824
[4] Improving Network Reliability Using Multipath TCP, https://developer.apple.com/documentation/foundation/urlsessionconfiguration/improving_network_reliability_using_multipath_tcp
[5] ns-3, https://www.nsnam.org/
[6] Direct Code Execution, https://www.nsnam.org/docs/dce/release/1.9/manual/singlehtml/index.html
[7] Throughput Fairness Index: An Explaination,Raj Jain, ArjanDurresi, Gojko Babic, https://www.cse.wustl.edu/~jain/atmf/ftp/af_fair.pdf
[8] iPerf - The ultimate speed test tool for TCP, UDP and SCTP, https://iperf.fr/
[9] TCP Congestion Control, https://tools.ietf.org/html/rfc5681
[10] The NewReno Modification to TCP’s Fast Recovery Algorithm, https://tools.ietf.org/html/rfc6582



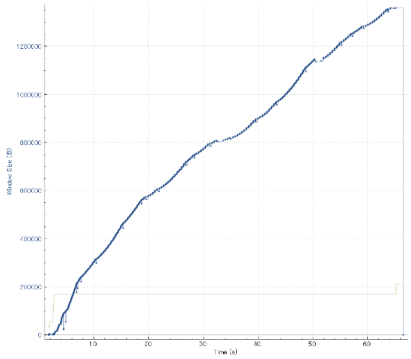
(a) TCP-MPTCP の TCP フロー



(b) TCP-MPTCP の MPTCP フロー



(c) TCP-HayACK の TCP フロー



(d) TCP-HayACK の HayACK フロー

図 6 各環境におけるウインドウスケーリング
( RTT = 350 [ms] )
：出力バイト数 [Packet]
：ウインドウサイズ [Packet]