輻輳制御の競合制御によるLDoS攻撃の緩和性能の改善

藤本 陽人[1,a)] 久末 瑠紅[1] 稲村 浩[2] 石田 繁巳[2]

概要：トランスポートプロトコルの脆弱性を悪用するDoS攻撃の1つにLDoS（Low-rate DoS）攻撃がある．LDoS攻撃は他のDoS攻撃と比較して攻撃トラフィックの送信量が少なく，既存の検知システムでの検知は困難である．代表的なLDoS攻撃であるShrew DoS攻撃はTCPのRTOによる再送タイミングの一意性を悪用するため，再送タイマ管理アルゴリズムを変更し再送タイミングの一意性を排除することで攻撃効果を緩和する事が可能である．しかしながら，再送タイマ管理アルゴリズムの変更によってTCPのタイマ値の最小値である1秒を下回り，Fast retransmissionとRTOの同時発生，すなわち輻輳制御の競合が確認された．Fast retransmissionはRTOによる再送と比較して再送に必要なコストが低いため，競合を制御することでTCPの性能低下を軽減しLDoS攻撃の緩和性能を改善できる可能性がある．本稿では，輻輳制御の競合を制御してTCPの性能低下を軽減し，LDoS攻撃の緩和性能の改善可能性を示す．

1. はじめに

インターネットを利用したサービスを対象とした攻撃にDoS（Denial of Service）攻撃がある．DoS攻撃は，攻撃対象に攻撃トラフィックを送信することでネットワークの輻輳を発生させ，サービスの品質を低下させる．代表的なDoS攻撃であるFlooding DoS攻撃は，攻撃トラフィックを継続的に送信して輻輳を発生させる．Flooding DoS攻撃は攻撃効果が非常に高いものの，攻撃時にトラフィックを大量に送信するため攻撃の検知が容易である．

一方で，攻撃トラフィックの送信量が少なく検知が難しいDoS攻撃も存在する．その1つがLDoS（Low-rate DoS） 攻撃である．LDoS攻撃は短時間の攻撃トラフィックを周期的に送信するため，Flooding DoS攻撃と比較して攻撃トラフィックの通信量が少ない [1].

LDoS攻撃は，トランスポートプロトコルの脆弱性を悪用して攻撃を行う．代表的なLDoS攻撃に，TCPのRTO（Retransmission Timeout）の一意性を悪用するShrew DoS攻撃がある [2].

著者らは先行研究 [3] にて，QUICの再送タイマ管理アルゴリズムを参考にしてTCPの再送タイマ管理アルゴリズムに変更を加えることで再送タイマ値が一意に決定することを抑制し，TCPに対するShrew DoS攻撃への緩和効果を付与できることを検証した．TCPの再送タイマ管理アルゴリズムとは異なり定数による支配項が存在しないことから，クロック粒度を除くパラメータがすべて通信状況を反映し動的であるため，再送タイマが一様には決定しないと考え，攻撃に対する緩和効果が期待できると考えた．結果，再送タイマ管理アルゴリズムを変更することでセグメントの再送タイミングと攻撃トラフィックの送信タイミングの連続的な同期を抑制することが可能となり，通信品質の改善に貢献できた．

しかしながら，提案した再送タイムアウトのタイマ計算アルゴリズムでは設定されるタイマの値が従来より短くなることを考慮できておらず，Fast retransmissionとRTOの同時発生，すなわち再送とそれに伴う輻輳制御の競合が発生した．これがにより輻輳ウィンドウの不必要な縮退が発生し，損失リカバリに大きなコストが発生することでTCPの性能に悪影響の発生が判明した．したがって，このTCPの性能低下を軽減することで，先行研究 [3] で示したLDoS攻撃の緩和性能を改善できる可能性がある．

本研究では，輻輳制御の競合を制御し，LDoS攻撃緩和効果の改善を目的とする．本研究のコントリビューションは次の2点である：

- 輻輳制御のメカニズムを分析し，輻輳制御の競合を制御する手法を提案した．
- 輻輳制御の競合制御を行うことで，先行研究 [3] と比較してLDoS攻撃下での正規化スループットの改善効果を示した．

本稿の構成は次のとおりである．2章にて背景と目的を示した．3章では，関連研究を示し，RTOを悪用する

[1] 公立はこだて未来大学大学院システム情報科学研究科
Grad. Sch. Systems Information Science, Future Univ. Hakodate
[2] 公立はこだて未来大学システム情報科学部
Sch. Systems Information Science, Future Univ. Hakodate
[a)] g2124040@fun.ac.jp

表 1　記号の説明

記号	パラメータ
t[秒]	時間
R[Mbps]	トラフィックレート
C[Mbps]	ボトルネックリンク帯域幅
R_A[Mbps]	攻撃トラフィックレート
L [秒]	攻撃パルス幅
T [秒]	攻撃トラフィック送信周期

LDoS 攻撃のメカニズム，短いタイマが設定されることによる TCP への影響，LinuxTCP に備わっている輻輳制御の競合制御システムについて述べる．4 章では，輻輳制御の競合制御を行う提案手法の原理を述べる．5 章では，提案手法の LDoS 攻撃緩和性能について評価し，その結果について議論する．最後に，6 章にてまとめとする．

本稿のグラフで使用する記号は**表 1** に示す．

2. 関連研究

本章では，Shrew DoS 攻撃の攻撃方法，短い再送タイマ値が設定されることによる TCP の伝送性能への影響，TCP タイムスタンプオプションを用いた輻輳ウィンドウサイズの調整方法について述べる．

2.1 RTO を悪用する LDoS 攻撃

LDoS 攻撃の 1 つである Shrew DoS 攻撃は，TCP で使用される再送制御の RTO 再送のタイミングを外部から予測できるという点を悪用し，攻撃トラフィックの送信タイミングを設定する [2]．**図 1** に Shrew DoS 攻撃の攻撃方法を示す．RTO はセグメントに対する ACK がタイマで指定した時間以内に受信できなかった場合に発生し，損失したセグメントの再送を行う．このとき，再送タイマ値の再設定を行う．再設定時に使用されるタイマのアルゴリズムは，RFC6298 により次式にて定義されている：

$$RTO_n = 2 \times RTO_{n-1} \tag{1}$$

最初に使用されるタイマ値は，RFC6298 により次式にて定義されている [4]：

$$RTO = \max(minRTO, SRTT + \max(G, 4 \times RTTVAR)) \tag{2}$$

ここで，$SRTT$ は平滑化された RTT，G はクロック粒度，$RTTVAR$ は RTT の分散を表す．$minRTO$ は，RFC6298 で 1 秒が推奨されている．$SRTT + \max(G, 4 \times RTTVAR)$ で導出される値はほとんどの場合 1 秒を下回るため，RTO 再送に使用される再送タイマは 1 秒となる．

タイマの値は指数関数で増加し最初に設定されるタイマの値は多くの場合 1 秒であるため，Shrew DoS 攻撃は攻撃トラフィックの送信タイミングを 1 秒に設定することで RTO 再送が行われたセグメントの連続的な損失が実現

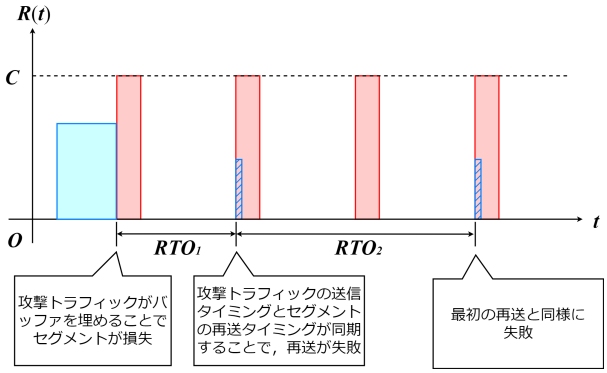


図 1　Shrew DoS 攻撃の攻撃方法

する．すなわち，外部から予測しやすい定数 $minRTO$ を設定することが LDoS 攻撃の成功要因を高めている．このことから，再送タイマ管理アルゴリズムから定数である $minRTO$ を排除することで LDoS 攻撃への耐性を付与できると考えられる．

2.2 PTO 処理による再送制御

著者らは先行研究 [3] にて，QUIC が TCP と通信方式が類似している点に着目し，TCP の再送タイマ管理アルゴリズムを式 (3) を参考にして変更を施すことで，$minRTO$ を取り除くことによって再送タイミングの一意性を排除し，LDoS 攻撃を緩和可能であることを示した．QUIC には再送タイマを用いた再送制御が備わっており，PTO 再送と呼ばれている．PTO 再送で使用されるタイマのアルゴリズムは，RFC9000 より次式にて示される [5]：

$$PTO = SRTT + \max(G, 4 \times RTTVAR) + max\_ack\_delay \tag{3}$$

TCP の RTO に式 (3) のアルゴリズムを適用したところ，再送タイマ値の大幅な上昇を抑え，RTT が 10 ミリ秒の環境ではセグメント 1 つあたりに対して最大 2 回，RTT が 100 ミリ秒の環境では最大 1 回の再送でセグメントの送信に成功した．すなわち，提案手法によって攻撃トラフィックの送信タイミングとセグメントの再送タイミングの同期を外すことができたといえる．

しかしながら，先行研究 [3] で示した手法を用いた場合，再送タイマ値が 1 秒を下回るため TCP の伝送性能へ悪影響を与える可能性が考えられる．RFC に準拠した TCP および先行研究 [3] での提案手法における，攻撃を行わなかった場合の TCP のタイマ値の変化を**図 2** に，攻撃時のタイマ値の変化を**図 3** に表す．先行研究 [3] で設定した再送タイマ管理アルゴリズムは $minRTO$ を排除したうえで実験は外乱が存在しないシミュレータで行われているため，RTT が小さくなり 1 秒を下回る RTO の値になったと考えられる．多くの外乱トラフィックが流れ輻輳レベルが高い環境では，$minRTO$ が小さいほど RTO による再送が行われたセグメントの数が増加する可能性がある．文献 [6] で

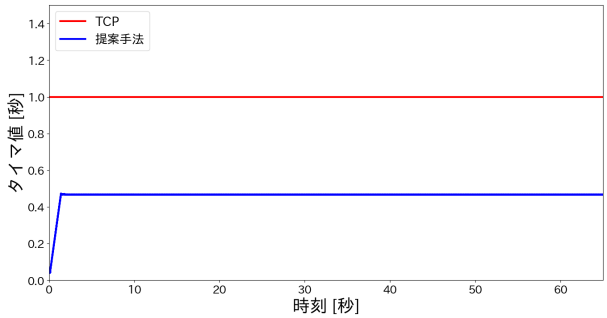


図 2　攻撃を行わなかった場合のタイマ値の変化 [3]

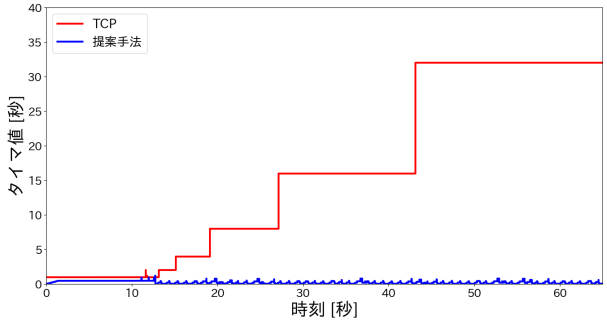


図 3　攻撃時のタイマ値の変化 [3]

は，*minRTO* を 1 秒に設定したときと比較し，*minRTO* が 750 ミリ秒，500 ミリ秒のときは約 1.3 倍，250 ミリ秒のときは約 1.8 倍まで不要な再送セグメントの数が増加し，TCP の性能に悪影響を及ぼすことを示している．

2.3　Linux TCP での輻輳ウィンドウサイズの調整

Linux TCP には，RFC と異なる仕様の機能がいくつか備わっている [7]．その 1 つとして，Linux TCP の *minRTO* は 200 ミリ秒に設定されており，IETF によって標準化されたものと比較すると大幅に小さい値となっている．このことから，2.2 で述べた通り TCP の性能に影響を及ぼす可能性がある．

この例外に対し，Linux TCP は輻輳ウィンドウサイズの調整を行うことで TCP の性能低下を抑制している．Linux TCP では不必要な再送を TCP タイムスタンプオプションによって判別することが可能であり，TCP 受信者は確認応答をトリガーしたセグメントのタイムスタンプを送信者にエコーし，TCP 送信者が ACK が元のセグメントによってトリガーされたのか，再送信されたセグメントによってトリガーされたのかを判断できるようにする．不必要な再送信が発見された場合，送信者側は RTO が発生する前の輻輳ウィンドウサイズに戻す．しかしながら，このアルゴリズムを使用するためにはタイムスタンプオプションを使用していることが前提であり，TCP タイムスタンプオプションが使用されていない TCP では適用できない．

2.4　既存研究の課題

既存研究では，再送タイマ値の縮小によって TCP の伝送性能の低下が発生することを明らかにしているが，RTO を除く TCP の機能に対する影響についての検証は行われていない．

Linux で使用されている TCP では，タイムスタンプを用いることで輻輳ウィンドウサイズの低下を抑制して伝送性能低下の軽減を可能であるが，タイムスタンプオプションを使用していることが前提である．

著者らの調査した範囲では，TCP のオプション機能を使用せずに輻輳ウィンドウサイズの調整を行い，TCP の伝送性能を改善することで LDoS 攻撃の緩和を行う研究はこれまでのところ報告されていない．本研究では，TCP 伝送性能低下の原因を特定し，TCP のオプション機能を使用せずに輻輳制御の競合を制御する手法を提案する．

3.　輻輳制御の競合による再送コストの増加

本章では，先行研究 [3] の実装による TCP の伝送性能低下の原因を特定し，悪影響について説明する．

3.1　損失リカバリと伝送性能の関係性

再送タイマ値が短縮することによって，Fast retransmission に対する ACK が返ってきていない状況で RTO による再送が発生する，すなわち Fast retransmission と RTO による再送がほぼ同時に発生するケースが見られた．Fast retransmission はセグメントの再送を行う機能であり，Duplicate ACK を一定回数受信することで該当セグメントの再送を行う．Fast retransmission は，送信したセグメントがすべて損失した場合ではなく，送信したセグメントの一部が損失した場合に発生することが多い．**図 4** は従来の TCP と先行研究 [3] のタイムシーケンスグラフを示しており，先行研究 [3] の方では Fast retransmission と RTO がほぼ同時に発生していることがわかる．この現象は TCP の正常な動作であり，RFC6675 では Fast recovery が長期化することによってタイマが期限切れを起こす可能性があると述べられている [8]．

通信の信頼性の観点からすれば，どちらかの再送が成功していれば通信は成立するので通信が不可能な状況に陥る可能性は低い．しかしながら，RTO による再送が行われることによって通信の効率が大きく低下する可能性がある．

表 2 は，RTO と Fast retransmission での 1 セグメントを再送するのに必要なコストを示す．輻輳ウィンドウサイズとは送信者が設定する値であり，輻輳ウィンドウサイズを用いて送信者が送信するセグメントの量を制限する．*ssth*（スロースタート閾値）は前回使用された輻輳ウィンドウサイズの半分の値，*mss* は最小セグメントサイズを表す．このことから，Fast retransmission よりも RTO のほうが損失リカバリに多くの時間を必要とし，輻輳ウィンド

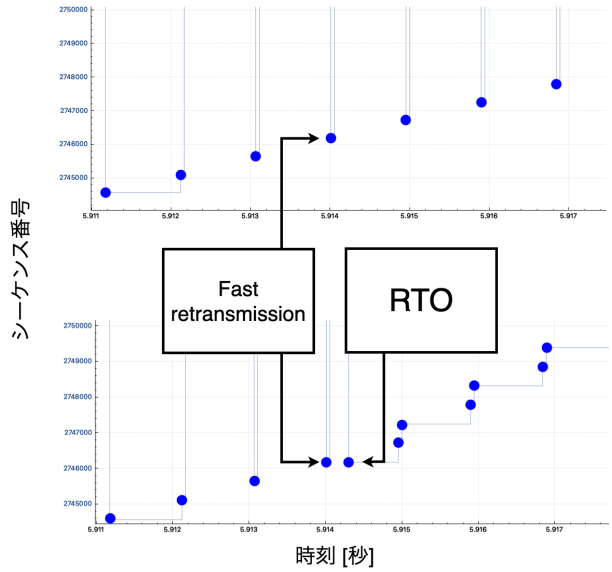


図 4 従来の TCP（上）と先行研究 [3]（下）のタイムシーケンスグラフ [3]

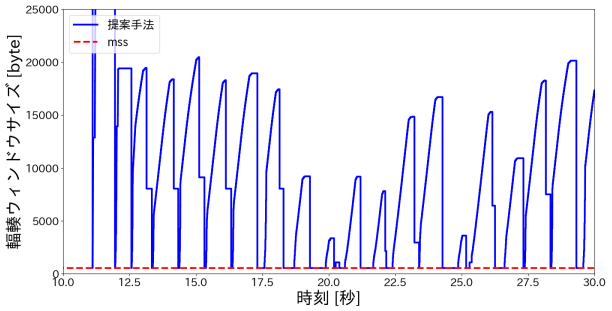


図 5 先行研究の輻輳ウィンドウサイズの変化 [3]

ウサイズを大幅に減少させてデータの転送可能量が制限されることから，再送コストが高いといえる．

図 5 は，先行研究 [3] の輻輳ウィンドウサイズの変化を示す．このことから，先行研究 [3] では輻輳ウィンドウサイズが高頻度で最小セグメントサイズまで低下していることがわかる．

これらのことから，先行研究 [3] では，輻輳制御の競合が発生したときに RTO の再送が優先されることで損失リカバリにより多くのコストがかかり，TCP の伝送性能を下げている可能性が高いと考えられる．

3.2 輻輳制御の競合の発生数

Fast retransmission が発生した場合，それと同時に輻輳制御の機能の 1 つである Fast recovery が実行される性質を利用し，先行研究 [3] を実装した TCP に LDoS 攻撃を行った際に，どのくらいの RTO 処理が Fast recovery の実行中に発生するのかについて調査を行った．**表 3** は，先行研究 [3] の手法を実装した TCP に対して LDoS 攻撃を行い，そこで発生した RTO の回数とその中で Fast recovery の実行中に発生した RTO の回数を表す．設定されたネットワークの遅延が低い環境ほど，Fsat recovery の実行中に発生する RTO の回数が多いことがわかる．ネットワークの遅延が低いほど設定されるタイマの値が小さくなることが原因の 1 つとして考えられる．このことから，輻輳制御の競合を制御することで特にネットワーク遅延の低い環境においては高い伝送性能の改善が期待できる．

表 2 1 セグメントの再送コスト

再送	リカバリに必要な時間	輻輳ウィンドウサイズ
FR	$1RTT$	$ssth$
RTO	$RTO + 1RTT$	mss

表 3 Fast recovery 中に RTO が発生した回数

ネットワーク遅延（ミリ秒）	RTO の発生回数	Fast recovery 中の RTO 発生回数
10	49	31
20	75	21
30	88	13
40	79	4
50	68	2

4. 輻輳制御の競合制御による LDoS 攻撃緩和手法

再送タイマ管理アルゴリズムに変更を加えることで LDoS 攻撃の緩和を実現できるが，$minRTO$ を排除することによって再送タイマ値が従来より短縮され，RTO が発生しやすくなり，TCP の伝送性能が低下する可能性が高い．

この課題を解決するために，本研究では輻輳制御の競合を制御し，再送コストの低い Fast retransmission による損失リカバリを優先させることで輻輳ウィンドウサイズの低下を軽減し，TCP の伝送性能の低下を防ぐ手法を提案する．

本章では，輻輳制御の機能の実行を確認することで Fast retransmission の発生を確認して競合制御を行う手法について述べる．

4.1 提案手法の流れ

アルゴリズム 1 に，提案手法の動作の流れを示す．RTO 発生時に Fast recovery が実行されている場合，輻輳ウィンドウサイズを低下させずに RTO を再設定する．Fast recovery が行われていない場合は，RFC に準拠した TCP と同様の動作を行う．

本研究では，先行研究 [3] が実装された TCP に対して提案手法を追加で実装することで，RTO による再送を遅延させることでスループットの低下を軽減し，LDoS 攻撃緩和性能を改善する．

4.2 Fast retransmission の発生確認

RTO が発生したタイミングで Fast retransmission の実行の有無を確認する必要がある．**図 6** は，Fast retrans-

アルゴリズム 1: 提案手法のアルゴリズム

```
Data: Used function of congestion control
Result: Control timing of retransmission
Expire RTO timer;
if Working fast recovery then
    Set RTO timer after RTO;
else
    RTO = 2 * RTO;
    Set RTO timer after RTO;
    cwnd = mss;
    Stop fast recovery;
    Retransmit segment;
```

mission と輻輳制御の関係を示す．Fast retransmission が発生した場合，輻輳制御は Fast recovery を行う．このことから，RTO が発生したときに輻輳制御が Fast recovery を行っている場合，Fast retransmission がまだ完了していない可能性が考えられる．

そこで提案手法では，RTO が発生したときに Fast recovery が同時に行われている場合，Fast retransmission が未完了であると想定して従来の RTO とは異なる動作を行うように仕様を変更する．

4.3 RTO による再送の遅延

RTO 時に Fast recovery が行われている場合 Fast retransmission が行われている可能性が高いため，RTO による再送を行う必要はないが，Fast retransmission が行われたセグメントが損失する可能性は十分に考えられるので，新たな RTO の設定が必要である．**図 7** は，Fast retransmission が完了するまでに必要な時間を示す．このことから，Fast retransmission で送信されたセグメントは，再送してから ACK を確認するまでおおよそ $1RTT$ 分の時間が必要である事がわかる．

そこで提案手法では，RTO が発生したタイミングに Fast recovery が実行されている場合，Fast retransmission の完了を待機するために再送タイマを $1RTO$ 遅延する．

5. 実験と評価

5.1 評価方法

先行研究と比較した本研究の優位性を明らかにするために，本研究では 2 つの評価を行う．1 つは，攻撃中のスループットから正規化スループットを求め，改善前と改善後の正規化スループットを比較して改善率を導出して比較を行う．正規化スループットは式 (4) で求める：

$$nomalized\_throughput = \frac{actual\_throughput}{maximum\_throughput} \quad (4)$$

この評価を行うことで，指定時間あたりに送信できたデータ量を比較し，LDoS 攻撃を受けている状態でも提案手法がデータの送信を継続性を示す．

もう 1 つは，輻輳制御の状態遷移して提案手法を実装することによる輻輳制御の競合をどれくらい制御できたかについて評価を行う．ns-3 の実装 [9] を参考に，輻輳制御の状態を以下のように定義する．

- *OPEN*: 輻輳制御が Slow start または Congestion avoidance 状態．損失が発生していないことを示す．
- *DISODER*: *OPEN* 状態で Duplicate ACK を 2 回連続で受信した状態．輻輳が発生していることを示す．
- *RECOVERY*: *OPEN* 状態で Duplicate ACK を 3 回連続で受信し Fast retransmission が実行されてから，新規 ACK を受信するまでの状態．輻輳が発生しており，NewReno においては Fast recovery が行われている状態を示す．
- *LOSS*: RTO 発生から新規 ACK を受信するまでの状態．深刻な輻輳状態であることを示す．

図 8 は，輻輳制御の状態を用いたの状態遷移図を示す．本研究で着目するべきなのは，*RECOVERY* からどの輻輳制御状態に遷移するかである．提案手法によって輻輳制御の競合制御が成功した場合，Fast recovery の実行中に RTO の期限切れが発生する可能性が低下するため，先行

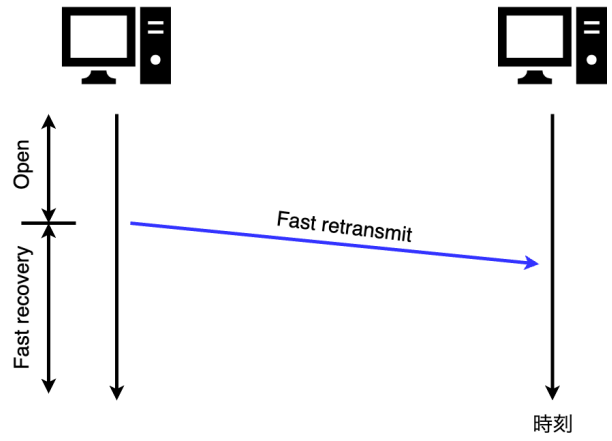


図 6 Fast retransmission と Fast recovery の関係

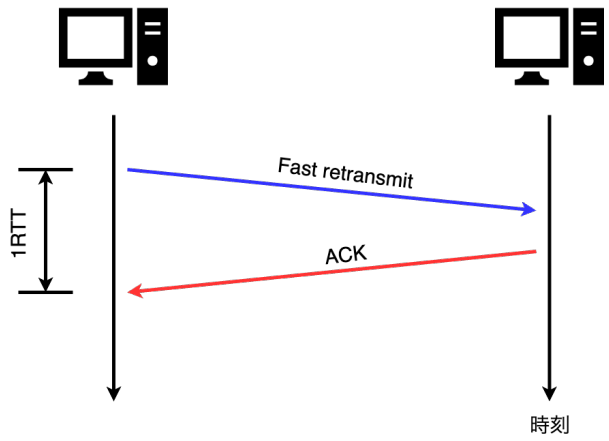


図 7 Fast retransmission に必要な時間

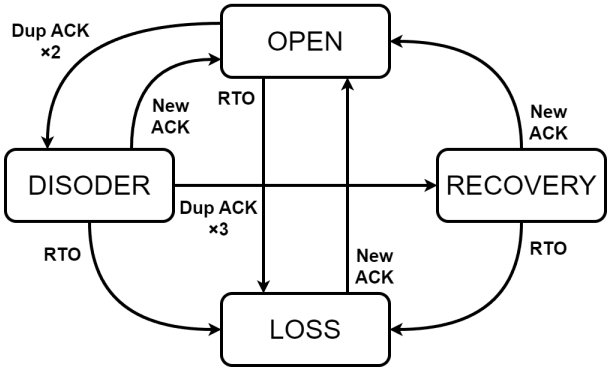


図 8　輻輳制御の状態遷移図 [9]

研究と比較して *RECOVERY* 状態の後に *OPEN* 状態に遷移する確率が高くなる．

そこで，先行研究と提案手法の実験結果から，輻輳制御の状態遷移および輻輳ウィンドウサイズの変化を用いて，競合制御によって輻輳制御の *LOSS* 状態への遷移をどのくらい抑制できたのかを明らかにする．

5.2　実験環境

本研究では，ネットワークシミュレータ（ns-3）上で**図 9** に示すトポロジを用いて実験を行う．送信者は，受信者に対して 1GB のデータをバルク転送する．2 つのルータ間の帯域幅を 10 Mbps, その他のリンクの帯域幅を 100 Mbps に設定し，ルータ間がボトルネックリンクになるように設定する．輻輳制御は，TCP で使用されている代表的な Loss-based 輻輳制御の 1 つである NewReno を採用する [10]．先行研究 [3] では設定されたネットワーク遅延によって結果が異なったため，本研究でも同様に異なるネットワーク遅延（10, 20, 30, 40, 50 ミリ秒）が設定された環境を用いて評価を行う．ただし，ネットワーク遅延の計測には SYN パケットの RTT を用いるものとする．バッファサイズは 500 KByte 固定とする．

攻撃パラメータを**図 10** に示す．攻撃レート R は，ボトルネックリンクの帯域を埋めることができるように 30 Mbps に設定する．攻撃の周期 T は 1 秒とし，攻撃パルス幅 L は 300 ミリ秒とする．

シミュレーション時間は，65 秒に設定する．最初の 10 秒間は，TCP の通信を安定させるために攻撃を行わない．用意した通信環境でそれぞれ 1 回ずつシミュレーションを行い，スループットを測定する．

5.3　実験方法

従来の TCP を実装した環境，先行研究 [3] の再送タイマ管理アルゴリズムのみを実装した環境，先行研究 [3] の再送タイマ管理アルゴリズムと提案手法を実装した通信環境の 2 つ対して 3 つの攻撃パルス幅についてそれぞれ 1 回ずつシミュレーションを行い，送信されたデータ量，輻輳

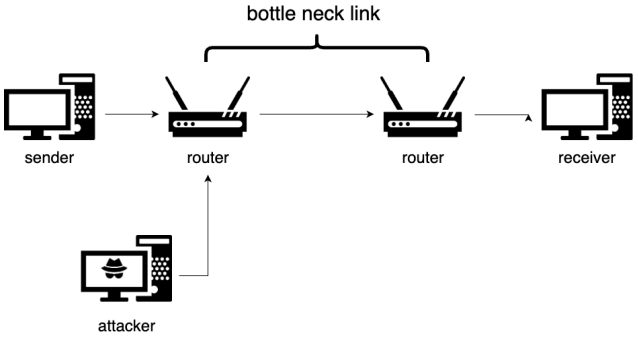


図 9　実験で使用するネットワーク

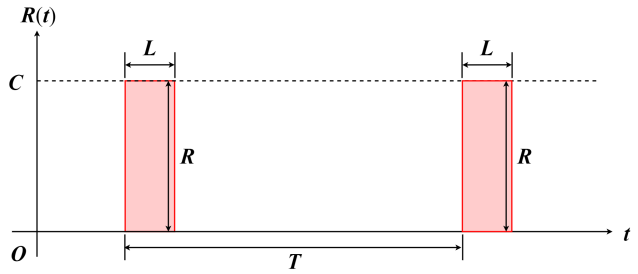


図 10　LDoS 攻撃のパラメータ

ウィンドウサイズ，再送タイマ値を取得する．

5.4　正規化スループットを用いた伝送性能の比較

図 11 は，実験で取得した正規化スループットを示す．従来の TCP は LDoS 攻撃によってスループットが大幅に減少しており，その値はほぼ 0 になっていることがわかる．一方で，再送タイマ管理アルゴリズムに変更を加えた先行研究 [3] を実装した TCP では，従来の TCP と比較して正規化スループットの改善が見られ，最大で約 0.35 まで改善することができた．そこからさらに変更を加えた提案手法は [3] よりも高い改善効果が確認でき，その値は約 0.45 まで改善された．

いずれのネットワーク遅延を設定した環境でも従来の TCP 及び先行研究 [3] よりも優位性は見られた．しかしながら，先行研究 [3] に見られた高いネットワーク遅延が設定された環境ほど緩和効果が低下する傾向は提案手法にも同様に見られた．

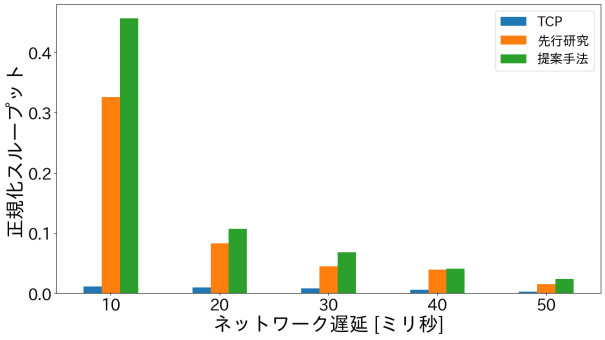
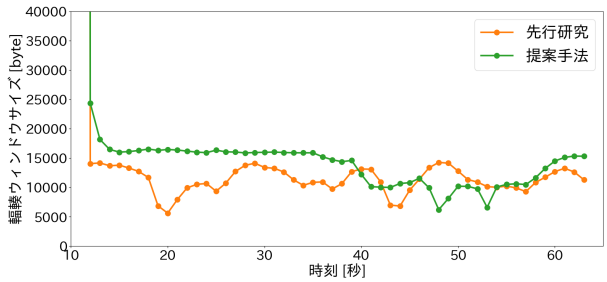
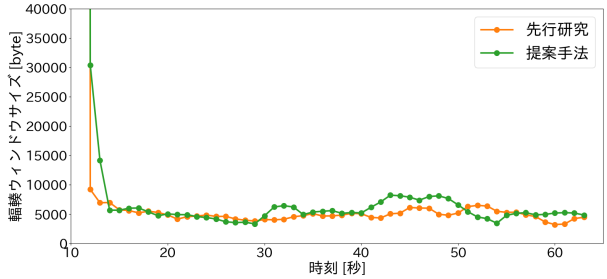


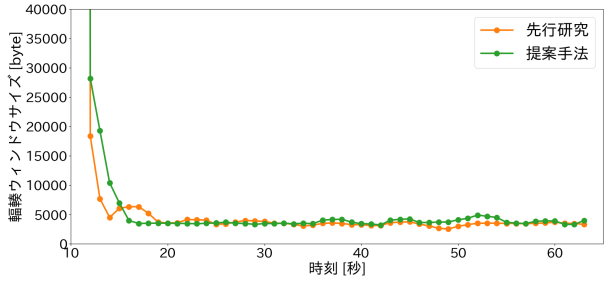
図 11　正規化スループット



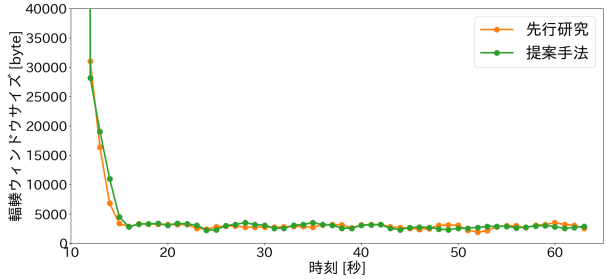
ネットワーク遅延：10ミリ秒



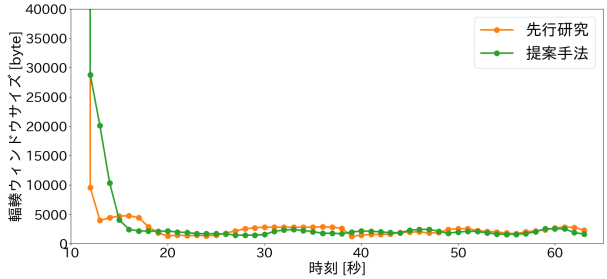
ネットワーク遅延：20ミリ秒



ネットワーク遅延：30ミリ秒



ネットワーク遅延：40ミリ秒



ネットワーク遅延：50ミリ秒

図 12　輻輳ウィンドウサイズの移動平均

図 12 は，攻撃中の輻輳ウィンドウサイズの移動平均を示す．ネットワーク遅延が低い環境ではある程度の移動平均の差が確認できることから，輻輳ウィンドウサイズの低下を軽減することでスループットが改善されたといえる．

一方で，ネットワーク遅延が高い環境では明確な輻輳ウィンドウサイズの差が見られないことから，輻輳ウィンドウサイズの低下をあまり軽減できなかったことから，スループットも大きく改善しなかったと考えられる．

5.5　輻輳制御の状態遷移を用いた競合解消の評価

表 4 は，輻輳制御が *RECOVERY* 状態から *OPEN* 状態に遷移した割合を表す．スループットの改善効果が一番高いネットワーク遅延が 10 ミリ秒に設定された環境に着目すると，先行研究 [3] を実装した TCP では輻輳制御の競合が多く発生しており，*RECOVERY* 状態の次に遷移した状態のうち，*OPEN* 状態に遷移したのはわずか 10.3%だけであった．

一方で，提案手法を実装した TCP では *RECOVERY* 状態の次に遷移した状態のうち *OPEN* 状態に遷移したのは全体の 93.3%であった．

図 13 は，ネットワーク遅延が 10 ミリ秒に設定された環境の輻輳ウィンドウサイズの変化を示す．赤枠の部分が輻輳制御の競合が発生した部分である．先行研究 [3] を実装した TCP では *RECOVERY* 状態に移行することで輻輳ウィンドウサイズが半減し，その直後に RTO が発生することで輻輳制御が *LOSS* 状態に遷移することで輻輳ウィンドウサイズが最小サイズに変化し，Slow strat で輻輳ウィンドウサイズが増加していることがわかる．

一方で，提案手法は輻輳制御の競合制御を行うことによって *RECOVERY* 状態に遷移して輻輳ウィンドウサイズが半減した後更に低下することはなく，輻輳制御が *OPEN* 状態に遷移して Congestion avoidance による輻輳ウィンドウサイズの増加が確認できる．

表 5 は，輻輳制御が *RECOVERY* 状態から *LOSS* 状態に遷移した割合を表す．先程と同様にネットワーク遅延が 10 ミリ秒に設定された環境に着目すると，*RECOVERY* 状態の次に遷移した状態のうち，*LOSS* 状態に遷移したのは 89.7%であり，多くのセグメントが Fast retransmission に対する ACK の受信を待機している間に RTO が発生したといえる．

一方で，提案手法を実装した TCP では *RECOVERY* 状態の次に遷移した状態のうち *LOSS* 状態に遷移したのはわずか 6.7%であり，競合制御を行うことによって不必要な RTO の発生を防ぐことができたと言える．

6.　おわりに

本研究では，輻輳制御の競合が発生した際に Fast retransmission を優先して行うことで TCP の性能低下を軽減し，LDoS 攻撃の緩和効果を改善できることを示した．具体的には，先行研究 [3] と比較して *RECOVERY* 状態

表 4　*RECOVERY* 状態から *OPEN* 状態に遷移した回数と割合（%）

ネットワーク遅延（ミリ秒）	先行研究 [3]	提案手法
10	3 (10.3)	28 (93.3)
20	0 (0)	5 (71.4)
30	2 (66.7)	4 (66.7)
40	3 (60)	3 (75)
50	4 (66.7)	7 (87.5)

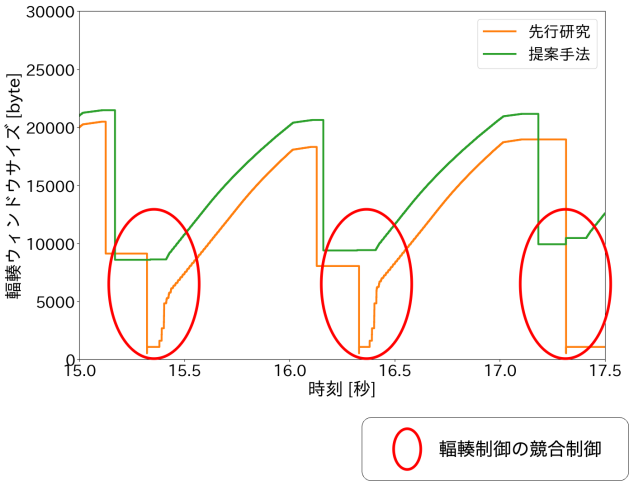


図 13　*RECOVERY* 状態から *OPEN* 状態に遷移した際の輻輳ウィンドウサイズの変化

表 5　*RECOVERY* 状態から *LOSS* 状態に遷移した回数と割合（%）

ネットワーク遅延（ミリ秒）	先行研究 [3]	提案手法
10	26 (89.7)	2 (6.7)
20	2 (100)	2 (28.6)
30	2 (66.7)	2 (33.3)
40	3 (60)	1 (25)
50	4 (66.7)	1 (12.5)

から *LOSS* 状態への遷移を 89.7%から 6.7%にまで減少し，スループットが約 1.3 倍まで上昇した．今後は，以下の 3 つの課題を解決する必要がある．

1 つ目は，ネットワーク遅延が高い環境での緩和効果の改善である．本研究では先行研究 [3] と同様に，設定されたネットワーク遅延が高い環境では緩和効果があまり大きくはならないよいう結果になり，依然として緩和効果は低い．

2 つ目は，輻輳制御のメカニズムを変更したことによる TCP への影響についての調査である．本研究の提案手法では輻輳制御に変更を加えているので，攻撃を受けていない通常の通信に対して影響を与える可能性が考えられる．

3 つ目は，実機環境での再現である．本研究の実験はネットワークシミュレータで行っているため，今回行った実装を実機環境で再現する方法について調査が必要である．

謝辞　本稿で示した研究の一部は，科研費 (JP21K11847) の助成で行われた．

参考文献

[1] Zhijun, W., Wenjing, L., Liang, L. and Meng, Y.: Low-rate DoS attacks, detection, defense, and challenges: A survey, *IEEE Access*, Vol. 8, pp. 43920–43943 (2020).

[2] Kuzmanovic, A. and Knightly, E. W.: Low-rate TCP-targeted denial of service attacks: the shrew vs. the mice and elephants, *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, pp. 75–86 (2003).

[3] 藤本陽人，久末瑠紅，稲村　浩，石田繁巳：RTO 計算変更による LDoS 攻撃耐性のシミュレーション評価，情報処理学会第 86 回全国大会講演論文集，pp. 3:129–130 (2024).

[4] Sargent, M., Chu, J., Paxson, D. V. and Allman, M.: Computing TCP's Retransmission Timer, RFC 6298 (2011).

[5] Iyengar, J. and Thomson, M.: QUIC: A UDP-Based Multiplexed and Secure Transport, RFC 9000 (2021).

[6] Loukili, A., Wijesinha, A. L., Karne, R. K. and Tsetse, A. K.: TCP's Retransmission Timer and the Minimum RTO, *2012 21st International Conference on Computer Communications and Networks (ICCCN)*, IEEE, pp. 1–5 (2012).

[7] Sarolahti, P. and Kuznetsov, A.: Congestion Control in Linux TCP, *2002 USENIX Annual Technical Conference (USENIX ATC 02)*, Monterey, CA, USENIX Association, (online), available from ⟨https://www.usenix.org/conference/2002-usenix-annual-technical-conference/congestion-control-linux-tcp⟩ (2002).

[8] Blanton, E., Allman, M., Wang, L., Järvinen, I., Kojo, M. and Nishida, Y.: A Conservative Loss Recovery Algorithm Based on Selective Acknowledgment (SACK) for TCP, RFC 6675 (2012).

[9] nsnam: ns-3 Network Simulator (2024). `https://www.nsnam.org/` (2024-5-12 アクセス).

[10] Gurtov, A., Henderson, T., Floyd, S. and Nishida, Y.: The NewReno Modification to TCP's Fast Recovery Algorithm, RFC 6582 (2012).