情報処理学会論文誌 Vol.61 No.2 287–296 (Feb. 2020)

リカレントニューラルネットワークを用いた双方向通信ログに基づくTCP輻輳制御の推定方式

大関 尚紀[1] 山本 嶺[1] 大坐畠 智[1] 加藤 聰彦[1,a)]

受付日 2019年5月5日, 採録日 2019年9月11日

概要：近年，ネットワークの多様化にともない，様々なTCP輻輳制御方式が提案されている．TCP輻輳制御方式はインターネットにおけるトラヒックを特徴づけるため，どのような方式がどの程度使用されているかを解析することは，ネットワーク事業者が自身のネットワークを管理するうえで重要となる．このような解析では多数のTCPフローを自動的に処理することが必要となる．本論文では，機械学習を用いて輻輳制御方式を推定する方法を提案する．提案方法は，ネットワーク内において収集されたデータとACKセグメントの双方を含む通信ログを使用し，ログに含まれる個々のTCPフローに対して，輻輳ウィンドウの時系列を推定する．さらに，この時系列に対して，リカレントニューラルネットワークを用いた識別器を適用し，そのフローの輻輳制御方式を推定する．Linuxオペレーティングシステムに実装された10の輻輳制御方式に適用した結果，代表的な3つの方式は明確に区別でき，また10の方式はいくつかのグループに分類できるという結果を得た．

キーワード：TCP輻輳制御アルゴリズム，パッシブモニタリング，輻輳ウィンドウ

Estimating TCP Congestion Control Algorithms from Passively Collected Bidirectional Packet Log Using Recurrent Neural Network

Naoki Ohzeki[1] Ryo Yamamoto[1] Satoshi Ohzahata[1] Toshihiko Kato[1,a)]

Received: May 5, 2019, Accepted: September 11, 2019

Abstract: Recently, as various types of networks are introduced, a number of TCP congestion control algorithms have been introduced. Since the TCP congestion control algorithms affect traffic characteristic in the Internet, it is important for network operators to analyze which algorithms are used widely in their backbone networks. In such an analysis, a lot of TCP flows need to be handled and so the automatically processing is indispensable. Thin paper proposes a machine learning based method for estimating TCP congestion control algorithms. The proposed method uses passively collected packet traces including both data and ACK segments, and calculates time sequence of congestion window size for individual TCP flows contained in the traces. We use a recurrent neural network based classifier in the congestion control algorithm estimation. As the results of applying the proposed classifier for ten congestion control algorithms implemented in the Linux operating system, the major three algorithms were clearly classified from the packet traces, and ten algorithms could be categorized into several groups which have similar characteristics.

Keywords: TCP, congestion control, passive monitoring, congestion window, recurrent neural network

1. はじめに

広域高速ネットワークや無線ネットワークなどの様々なネットワークの導入にともない，多数のTCP輻輳制御アルゴリズムが提案されている [1]．TCPの輻輳制御が導入された当初は [2]，TCP Tahoe [3]，TCP Reno [3]，TCP NewReno [4] といった限られたアルゴリズムのみが使用されていた．これに対して近年，多様な輻輳制御アルゴリズムが新たに提案・導入されている．広帯域・高遅延のネットワー

[1] 電気通信大学
The University of Electro-Communications, Chofu, Tokyo 182–8585, Japan
[a)] kato@is.uec.ac.jp

クに対応することを目的として，HighSpeed TCP [5]，Scalable TCP [6]，BIC TCP [7]，CUBIC TCP [8]，Hamilton TCP [9] が使用されている．また TCP Westwood+ [10] は損失のある無線リンクでの利用を想定している．これらはパケット損失を輻輳の指標とする損失ベース方式である．一方，TCP Vegas [11] は RTT（Round-Trip Time）の増加を輻輳の指標とする遅延ベース方式を採用している．さらに，TCP Veno [12] は，輻輳制御はパケット損失を契機に起動するが，輻輳ウィンドウの増加方法は RTT により決定するというハイブリッド方式である．さらに 2016 年には Google が従来の輻輳制御アルゴリズムの課題を解決する TCP BBR [13] を提案した．

TCP はインターネットを流れるトラヒックの大部分で使用されており，輻輳制御は個々のフローの振舞いを決定づける．このためどのアルゴリズムがどの程度利用されているかを知ることはネットワーク事業者にとって重要となる．輻輳制御アルゴリズムの推定手法は大きく 2 つに分類される．1 つは受動的に収集された通信ログに基づき推定を行うパッシブ方式で，もう一方は，テスタが対象となるノードとの間で（通常手順ではない）テスト用のセグメント列を送信し，その結果から輻輳制御アルゴリズムを推定するアクティブ方式である．アクティブ方式の方が多数の輻輳制御アルゴリズムの推定が可能であるが，ネットワーク事業者が，自身の管理するネットワークを流れる TCP フローのアルゴリズムを推定するという作業には使用できない．一方，筆者らの提案以外のパッシブ方式は，対応可能なアルゴリズムの種類が少ないという課題がある．

筆者らはこれまでに，パッシブ方式を用い．現在使用されている多くの輻輳制御アルゴリズムを識別する方法を提案した [14], [15]．この提案では，パケットトレースにおいて，再送が行われていない範囲（シーケンス番号が増加している範囲）から，RTT ごとの輻輳ウィンドウサイズを推定し，ウィンドウサイズの値とその増加分の関係から輻輳制御アルゴリズムを識別するというアプローチを採用した．この方法により，双方向の通信ログから複数のアルゴリズムを推定可能であることを示した．

しかしながら筆者らの方法では，輻輳ウィンドウサイズとその増加分のグラフから人手による判断でアルゴリズムを識別する．このため多数の TCP フローを扱うことは困難である．そこで筆者らは，機械学習を用いて自動的に輻輳制御アルゴリズムを推定する方法を検討してきた [16], [17]．具体的には，データと ACK の双方を含む通信ログから，TCP フローごとに輻輳ウィンドウサイズ（congestion window size：cwnd）の時系列を推定し，リカレントニューラルネットワーク（recurrent neural network：RNN）を用いた識別器を用いて，その時系列からアルゴリズムを推定する．Linux オペレーティングシステムに実装されている代表的な 10 の輻輳制御アルゴリズムを対象として，その適用性を評価した．本論文では，筆者らの提案する方法の詳細と，輻輳制御アルゴリズムの推定結果について述べる．

本論文の構成は以下のとおりである．2 章では輻輳制御アルゴリズムの推定と，TCP に対する機械学習の適用に関する関連研究を示す．3 章で本論文での提案する方法を示し，4 章では提案方法による輻輳制御アルゴリズムの推定の結果を示す．最後に 5 章で結論を示す．

2. 関連研究

2.1 TCP 輻輳制御アルゴリズムの推定に関する研究 [15]

パッシブ方式を用いた初期の研究 [18], [19], [20] では，通信ログの情報から，TCP の送信側の内部状態や内部変数（cwnd やスロースタート閾値（ssthresh）など）を推定し，事前に用意した TCP 状態マシンに従って，送信側 TCP の振舞いをエミュレートする．これらの研究では Tahoe，Reno，NewReno などの限られた輻輳制御アルゴリズムのみを対象としている．文献 [21], [22] では，Linux に実装されている 13 の輻輳制御アルゴリズムの識別を対象としているが，13 のアルゴリズムのうち任意の 2 つのみを対象として，そのどちらが実装されているかを区別することを目的としている．推定にあたっては，RTT ごとの cwnd を推定し，その値の増加率，1 セグメント分だけ増加した回数の割合などの特徴量を求め，その特徴量をクラスタ分析するという手法を用いている．

一方，アクティブ方式の研究例 [23] では，テスタが Web サーバに対して 512 のデータセグメントを，制御された環境下で送信させ，連続に転送されるデータセグメントの数を観測する．その結果から cwnd の増加と再送時の cwnd の減少の様子を調べ，サーバが 13 の輻輳制御アルゴリズムのいずれを実装しているかを判定する．

筆者らの過去の研究 [14], [15] では，双方向の通信ログから再送が行われていない区間を対象として，RTT と RTT ごとの cwnd の値を推定する．得られた cwnd の時系列から，cwnd と cwnd の増加分の対応をグラフ化し，グラフの形状から使用されている輻輳制御アルゴリズムを推定する．たとえば，TCP Reno の場合は，cwnd の増加分は cwnd の値によらず 1 パケットとなる．また CUBIC TCP では増加分は cwnd の値に対して $\sqrt[3]{cwnd^2}$ のカーブを描く．このようにして，Linux オペレーティングシステムに実装された 8 つのアルゴリズムを推定可能であることを示した．

2.2 機械学習の TCP への適用の研究

近年，ネットワーク分野に対して機械学習を適用する研究がさかんに行われている．例として，自己組織化ネットワークの管理への適用 [24]，侵入検知への適用 [25]，通信ログからのモバイルアプリケーションの推定 [26] などがあげられる．

TCP の分野でも多くの研究が報告されている．文献 [27]

は fixed-share と呼ばれる機械学習手法と移動平均を組み合わせて，測定された RTT サンプルから正確な RTT を推定する方法を提案している．また文献 [28] は，測定された利用可能帯域，キュー長，パケット損失率などから，サポートベクタ回帰により TCP フローの次のタイミングでのスループットを推定する方法を提案している．さらに文献 [29] では，無線 LAN の電波強度，データレート，アクセスポイントとの間の RTT などに用いて，マルチパス TCP におけるサブフローを選択する機械学習ベースのスケジューラを提案している．

最近，TCP の振舞いの解析に機械学習を活用する研究が報告されている [30], [31]．文献 [30] では CUBIC TCP，TCP Reno，BIC TCP を対象として，機械学習を用いて，通信ログの情報から cwnd の時間変化を推定する方法を提案している．また文献 [31] も [30] と同様に，有線および無線ネットワーク上の TCP 通信の通信ログから，RNN を用いて cwnd や RTT などの TCP 通信の振舞いの時間変化を推定する方法を提案している．

これに対して本論文では，通信ログの情報から機械学習を用いて輻輳制御アルゴリズムを推定する方法を提案している．筆者らの知る限りでは，このような研究報告はこれまでに行われていないと考えている．

3. 提案方法

3.1 RTT ごとの輻輳ウィンドウサイズの推定

提案方法では，筆者らのこれまでの提案 [15] と同様に，通信ログから RTT ごとに cwnd の推定を行う．パケットをキャプチャするモニタがネットワークの途中にあることを考慮して，以下のような方法を用いる．ここでは TCP タイムスタンプオプションが使用されていることを前提とする．

- 再送データに対する ACK セグメントなど，通信ログにおける特定の ACK セグメントに着目する．これを $ACK1$ と呼ぶ．
- 着目した ACK セグメント $ACK1$ の TSval（timestamp value）の値を TSecr（timestamp echo reply）に持つデータセグメントを探す．これを $DATA1$ と呼ぶ．
- データセグメント $DATA1$ を最初に確認応答する ACK セグメントを探す．これを $ACK2$ とする．さらに $ACK2$ の 1 つ前の ACK セグメントを $ACK1'$ と呼ぶ．
- ACK セグメント $ACK2$ の TSval の値を TSecr に持つデータセグメントを探す．これを $DATA2$ と呼ぶ．

これにより，$ACK1$ を受信した時点での cwnd を以下の式により推定する．

$$cwnd = \left\lfloor \frac{seq\ in\ DATA2 - ack\ in\ ACK1'}{MSS} \right\rfloor \quad \text{(segments)} \tag{1}$$

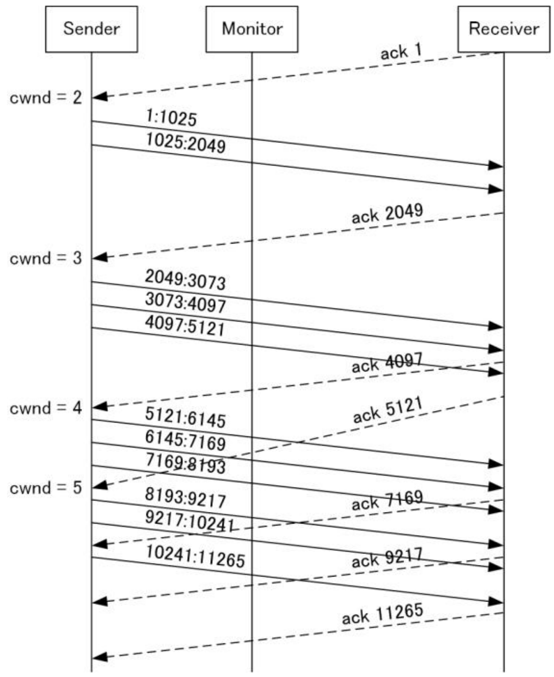


図 1 RTT ごとの cwnd の推定の例

Fig. 1 Example of estimation of cwnd associated with one RTT.

ここで，seq と ack は TCP ヘッダ中のシーケンス番号と応答確認番号を示す．MSS はその TCP コネクションで使用される最大セグメントサイズ（maximum segment size：MSS）である．また記号 $\lfloor x \rfloor$ は x の整数部分を示す．

具体的な例を**図 1** に示す．ここでは説明を容易にするためにコネクション確立直後のスロースタートフェーズを用いる．MSS は 1024 バイトとし，データセグメントは実線，ACK セグメントは点線で示す．データセグメントには，ヘッダ中のシーケンス番号とそのセグメントの最後のバイトに付与された番号 + 1 の値を示す．また ACK セグメントには応答確認番号を示す．

モニタで収集された通信ログから ack 1 に着目する．次にその ACK セグメントの TSval を TSecr に持つデータセグメント 1:1025 に着目し，これを ACK とする ack 2049 を $ACK2$ として識別する．その TSval を TSecr に持つデータセグメント 2049:3073 を $DATA2$ と識別する．ack 2049 の前の ACK セグメント（$ACK1'$）は ack 1 であるため，1:1025 のデータセグメントから始まる RTT の間の cwnd は，$(2049 - 1)/1024 = 2$ から 2 セグメントと判断する．同様に，次の 2 回の RTT に対しては，ack 2049 に着目し，$ACK2$ を ack 4097，$ACK1'$ を ack 2049，$DATA2$ を 5121:6145 とすることにより $(5121 - 2049)/1024 = 3$ が得られる．次の RTT に対しては，ack 4097 を $ACK1$ と見なし，同様に $(10241 - 5121)/1024 = 5$ が求められる．

3.2 識別機への入力の選択と正規化

パケット損失が発生し再送されると，cwnd は減少する．

提案方法では輻輳回避フェーズにおける cwnd の増加に着目するために，パケット損失間の cwnd の時系列を求める．このために，特定の TCP フローについて，TCP ヘッダのシーケンス番号が増加し続ける部分を通信ログから探す．この期間をロスレス区間と呼ぶ．提案方法では，識別器の入力として，1 つのロスレス区間の推定された cwnd の時系列を用いることとする．

しかし，ロスレス区間の長さはそれぞれ異なり，また cwnd の値の範囲もそれぞれで異なる．そこで，提案方法では，ロスレス区間の長い cwnd 時系列を優先的に入力として選択し，さらに選択した cwnd 時系列に対して，時間軸と cwnd 軸の双方で正規化することとした．

識別機への入力の選択と正規化のアルゴリズムを**図 2** に示す．アルゴリズムへの入力の E は通信ログから得られた特定の TCP フローの RTT ごとの cwnd の配列である．このアルゴリズムでは，後述する実験（RTT がほぼ一定）を想定しており，RTT の実際の値は考慮せず，単なる cwnd の配列を使用している．またもう 1 つの入力 $InputLength$ は識別器への 1 つの入力のサンプル数であり，提案方法では 128 を使用した．

最初に cwnd の配列をパケット損失で分割し，その結果を二次元配列 S に格納する．次にスロースタートフェーズを省くために最初の cwnd 配列 $S[0]$ を除く．さらに，S の各要素をそれぞれの cwnd 配列の長さの降順に並べ替える，

続いて S の各要素 $S[t]$ に対して，配列の値が 0 から 1 の間になるように正規化する．正規化は次のように行われる．

cwnd の最大値を求める．

$w_{max}[t] = \max(S[t][u])$

ただし $u = 0 \cdots \mathrm{Len}(S[t]) - 1$

cwnd の最小値を求める．

$w_{min}[t] = \min(S[t][u])$

ただし $u = 0 \cdots \mathrm{Len}(S[t]) - 1$

$S[t]$ の中のそれぞれの cwnd の値に対して，以下のように正規化する．

$$S[t][u] \leftarrow \frac{S[t][u] - w_{min}[t]}{w_{max}[t] - w_{min}[t]}.$$

次に，cwnd の値を $InputLength$（提案方法では 128）の個数に再サンプリングする．この処理はアルゴリズムのステップ 11 から 15 のループで行われる．この結果，$S[t]$ の cwnd の配列は，128 の要素を持つ配列 $I[t]$ に変換される．このアルゴリズムにより，すべての cwnd の配列は，値が 0 と 1 の間の 128 個の要素を有する配列に変換される．

3.3 RNN に基づく識別器の採用

提案方法では，TCP の輻輳制御アルゴリズムを出力とする識別器を構築するために RNN を採用した．ニューラルネットワークは，様々なデータの分類・回帰・推定のために広く使用されている．特に RNN は時間的な順序のある振舞いを扱うために使用される．例としては，映像・音声の認識や手書き文字の識別などがあげられる．提案方法においても cwnd の値の時系列を扱うため，RNN が適切であると判断した．さらに RNN の中でも長期的な依存関係を学習可能な LSTM（long short-term memory）[30] を用いることとした．識別器への入力は上記のような正規化された cwnd の時系列とし，輻輳制御アルゴリズムを one-hot ベクトルで表したものをラベルとする．

```
Algorithm 1
1. function Normalize (E, InputLength)
2.         S <- DivideAtLoss(E)
3.         Delete(S[0])
4.         S <- SortBySequenceLength(S)
5.         for t = 0 to Len(S) - 1 do
6.                 S <- MinMaxNormalization(S)
7.         end for
8.         I <- Array(Len(S))
9.         for t = 0 to Len(S) - 1 do
10.                I[t] <- Array(InputLength)
11.                for u = 0 to InputLength - 1 do
12.                        SurjectiveMap <- InputLength/Len(S[t])
13.                        Index <- Trunc(u / SurjectiveMap)
14.                        I[t][u] <- S[t][Index]
15.                end for
16.        end for
17.        return I
18. end function
```

図 2 識別器への入力の選択と正規化のアルゴリズム
Fig. 2 Algorithm to select and normalize inputs to classifier.

表 1 識別器のハイパーパラメータ
Table 1 Hyper parameters of classifier.

Parameter	Value
Input Length	128
Hidden Layers	1
Hidden Neurons	512
Optimizer	Adam
Learning Rate	2×10^{-4}

RNN のハイパーパラメータを**表 1** のように定める．このうち，隠れ層の数が 512 と，入力データ長（128）に比べて大きく設定されている．これは，隠れ層に冗長な表現力を持たせ，学習過程でニューラルネットワーク自身にその要否を判断させるようにするためである．

また識別器の学習においては，ミニバッチ学習を用いることとした．用意した学習データからミニバッチサイズの数だけランダムに入力を選び識別器に与える．学習は損失関数の値が収束したと判断できるまで行った．

4. 実験結果

4.1 実験構成

図 3 に cwnd の時系列データを収集するための実験構

情報処理学会論文誌　Vol.61　No.2　287–296 (Feb. 2020)

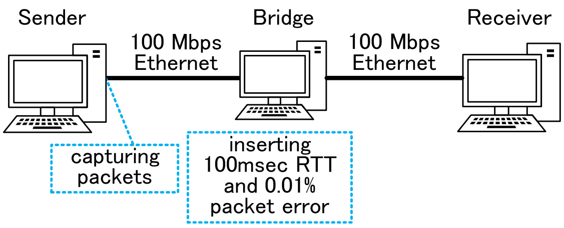


図 3　実験構成
Fig. 3　Experiment configuration.

成を示す．送信側，受信側，ブリッジが 100 Mbps イーサネットで接続される．各ノードのオペレーティングシステムとして Ubuntu18.04 を用いた．ブリッジにおいて片方向それぞれ 50 ミリ秒の遅延を挿入した．その結果，TCP フローの RTT は 100 ミリ秒となる．また輻輳制御を起動するために，ブリッジにおいてランダムなパケット損失を挿入した．パケット損失率は 0.01%である．データ転送は iperf3 [31] を送信側と受信側で動作させることにより行う．通信ログは，送信側のイーサネットインタフェースにおいて，tcpdump を用いて収集した．また通信ログの解析のために，Python 3 dpkt ライブラリ [32] を用いた．送信側において，sysctl コマンドにより輻輳制御アルゴリズムを変更した．

4.2　輻輳ウィンドウサイズの推定と正規化

実験においては，TCP Reno，HighSpeed TCP，Scalable TCP，BIC TCP，CUBIC TCP，Hamilton TCP，TCP Westwood+，TCP Vegas，TCP Veno，TCP BBR の 10 種類の輻輳制御アルゴリズムを使用した．**図 4**，**図 5**，**図 6**，**図 7**，**図 8**，**図 9**，**図 10**，**図 11**，**図 12**，**図 13** はそれぞれのアルゴリズムを使用した場合の，通信ログから推定された cwnd の時系列の例である．これらの例ではいずれも iperf3 によるデータ転送を 60 秒行っており cwnd は 3.1 節で述べた方法によりセグメント単位で求められている．

提案方法はロスレス区間における cwnd の時系列を識別器への入力とすることを基本としている．図 4 と図 8 において *Reno*1，*Reno*2，*CUBIC*1，*CUBIC*2 で示した部分がその例である．しかしこれらが示すように，各時系列は時間軸および cwnd の値の範囲が異なっている．このため 3.2 節のアルゴリズムにより正規化することとしたわけである．**図 14** はこれらの cwnd 時系列の正規化の概要を図示している．その結果異なるスケールの cwnd 時系列が，値が 0 から 1 の 128 サンプルの正規形に変換されている．

4.3　輻輳ウィンドウサイズの推定結果

4.3.1　識別器の実装概要

TCP 輻輳制御アルゴリズムを推定するための識別器を，Google から提供されている TensorFlow ライブラリ [33] を用いて実装した．**表 2** に識別器の実行環境を示す．Ten-

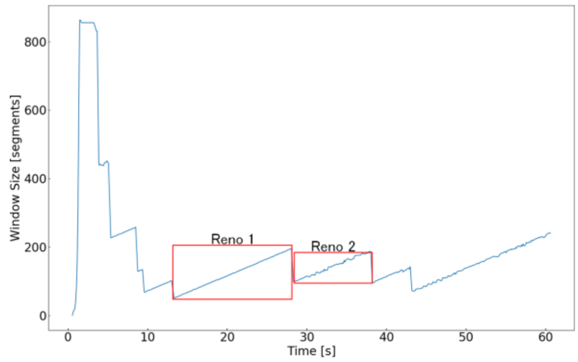


図 4　TCP Reno の cwnd 推定結果例
Fig. 4　Estimated cwnd for TCP Reno.

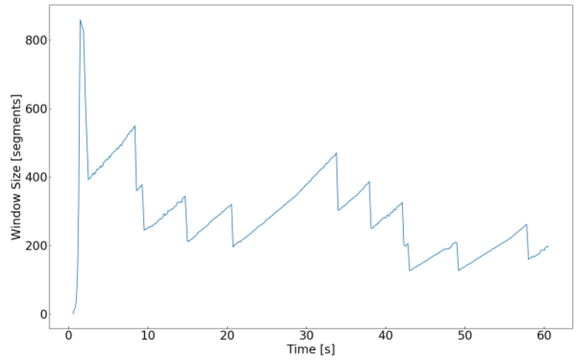


図 5　HighSpeed TCP の cwnd 推定結果例
Fig. 5　Estimated cwnd for HighSpeed TCP.

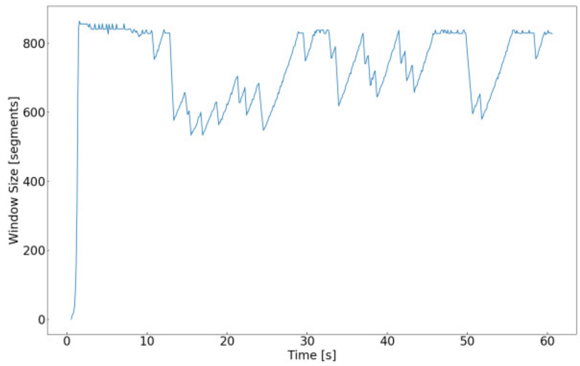


図 6　Scalable TCP の cwnd 推定結果例
Fig. 6　Estimated cwnd for Scalable TCP.

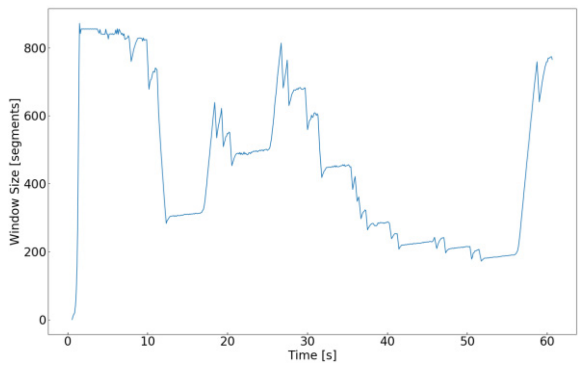


図 7　BIC TCP の cwnd 推定結果例
Fig. 7　Estimated cwnd for BIC TCP.

情報処理学会論文誌　Vol.61　No.2　287–296 (Feb. 2020)

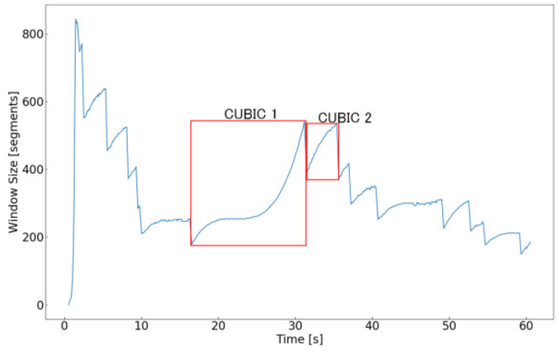


図 8　CUBIC TCP の cwnd 推定結果例
Fig. 8　Estimated cwnd for CUBIC TCP.

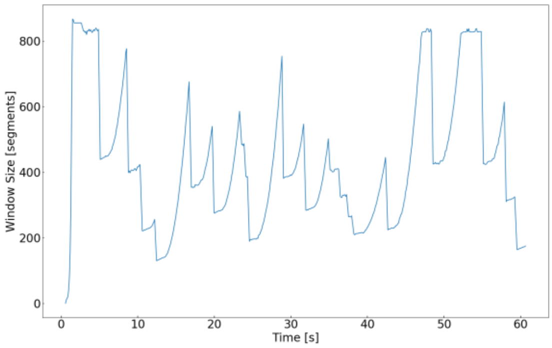


図 9　Hamilton TCP の cwnd 推定結果例
Fig. 9　Estimated cwnd for Hamilton TCP.

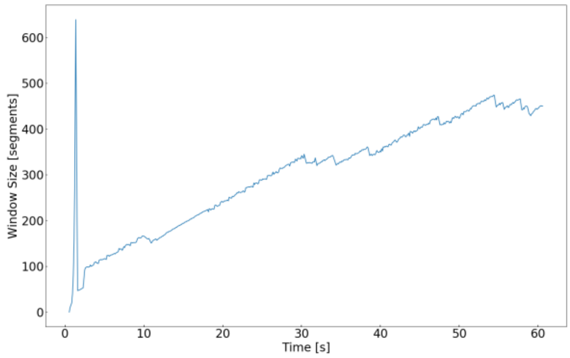


図 10　TCP Westwood+ の cwnd 推定結果例
Fig. 10　Estimated cwnd for TCP Westwood+.

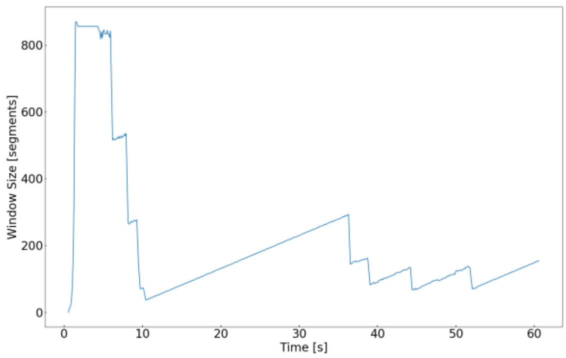


図 11　TCP Vegas の cwnd 推定結果例
Fig. 11　Estimated cwnd for TCP Vegas.

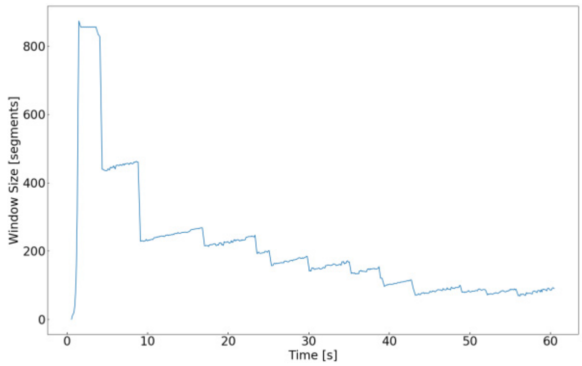


図 12　TCP Veno の cwnd 推定結果例
Fig. 12　Estimated cwnd for TCP Veno.

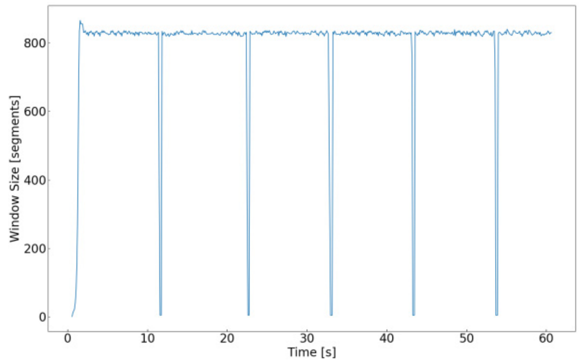


図 13　TCP BBR の cwnd 推定結果例
Fig. 13　Estimated cwnd for TCP BBR.

sorFlow において使用したプログラム言語は Python 3 で，実行環境は Google Colaboratory [34] を用いた．実行時には，Google クラウドプラットフォーム上の GPU を使用した．

通信ログの取得，正規化，識別器の学習用データとテスト用の取得，識別器の学習とテストによる輻輳制御アルゴリズムの推定について，具体的な手順として次のようなステップを用いた．

- 各輻輳制御アルゴリズムに対して，学習用データとテスト用データを得るために，iPerf3 を用いた 10 分間のデータ転送を 10 回ずつ，計 20 回行った．1 回ずつの結果に対して，3.2 節のアルゴリズム 1 を用いて，通信の開始時点以外のロスレス区間の正規化された cwnd 時系列を求める．これはサンプル数の多い順に並べられている．これらの結果から，学習用およびテスト用に，サンプル数の多いものから 400 ずつの cwnd 時系列を選択し，識別器への入力とする．この処理は，輻輳制御アルゴリズムごとに行う．**図 15** では，これらの結果を最上部の長方形で図示している．
- 実験では，得られた入力を基づいて，学習とテストを繰り返して行った．学習については，1 回のトライアルに対して，ミニバッチサイズ分の入力を 400 の cwnd 時系列からランダムに選び識別器に与え，識別機の出

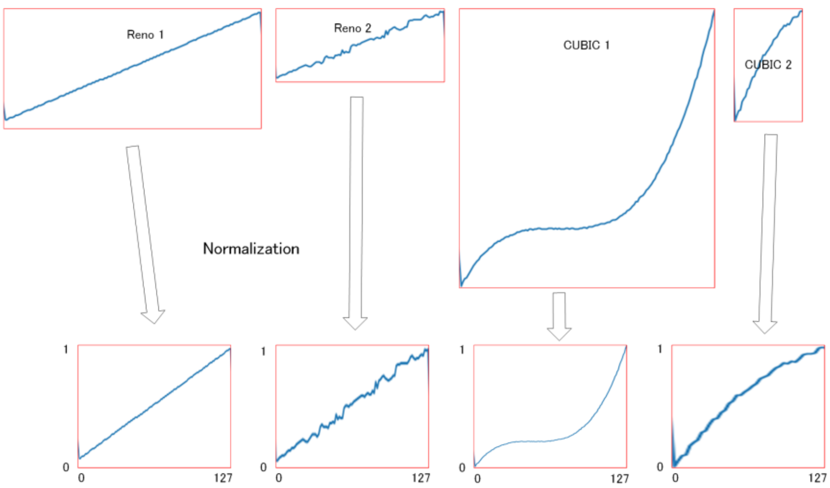


図 14　正規化の概要
Fig. 14　Illustration of normalization.

表 2　識別器の実行環境
Table 2　Environment for machine learning.

Item	Value
Neural Network Library	TensorFlow
Execution Environment	Google Colaboratory
Runtime Type	Python 3
Hardware Accelerator	GPU
Number of Training Data	400
Number of Test Data	400

力と入力のラベルとが一致するかを調べた．これを対象となるすべての輻輳制御アルゴリズムに対して行い，アルゴリズムごとの識別精度と，全体の識別精度を求めた．

- 1 回の学習トライアルが終わると，テストトライアルを実行する．テストトライアルでは，輻輳制御アルゴリズムごとに 400 の入力をすべて用いる．その結果からテスト用データに対する推定の識別精度（アルゴリズム別と全体）を計算した．
- これらの実験を繰り返して行い，トライアルの回数（エポック数）に対する識別精度の変化を求め，学習過程を評価した．

4.3.2　主要 3 種の輻輳制御アルゴリズムに対する推定結果

第 1 の実験として，TCP Reno，CUBIC TCP，TCP BBR に着目し，この 3 つの間で輻輳制御アルゴリズムの推定を行った．この 3 つのアルゴリズムを対象としたのは以下の理由による．TCP Reno は TCP に輻輳制御機能が導入された当初より，広く用いられているアルゴリズムである．CUBIC TCP は Linux，macOS X Yosemite，Windows 10 などの主要オペレーティングシステムにおいてデフォルトに設定されているアルゴリズムである．TCP BBR は，Google によって新たに提案されたアルゴリズムであり，広く注目されている．

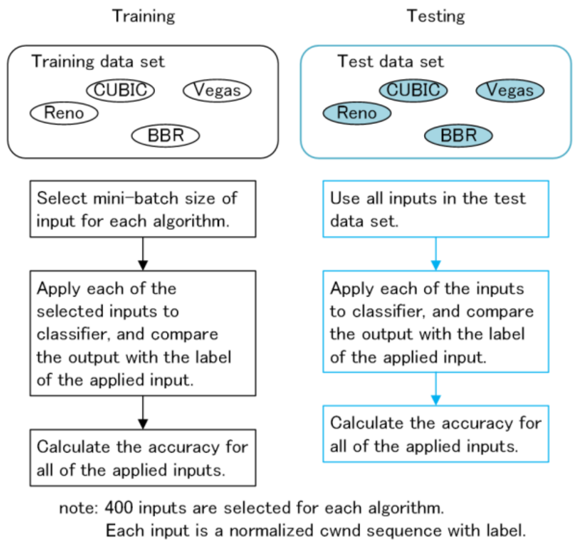


図 15　実験手順の概要
Fig. 15　Overview of experiment.

この実験においては，ミニバッチサイズとして 128 を使用した．**図 16** に 3 つの輻輳制御アルゴリズムの全体に対する学習曲線を示す．図の横軸は 500 までのエポック数であり，縦軸は学習トライアルとテストトライアルにおける 3 つのアルゴリズム全体に対する識別精度である．青線が学習トライアル，赤線がテストトライアルを示す．図 16 の結果から，エポック数が増加するにつれて，トライアル回数 220 以降において，学習・テストトライアルともに識別精度が 1 に収束している．この結果は識別器において過学習が発生していないことを示す．

図 17 に 3 つのアルゴリズムの識別に対する混同行列を示す．表の行は各アルゴリズム（Reno，CUBIC，BBR）の真の値のラベルを示す．列は推定されたラベルに対応する．表の値は列で示すラベルに対するアルゴリズムごとの

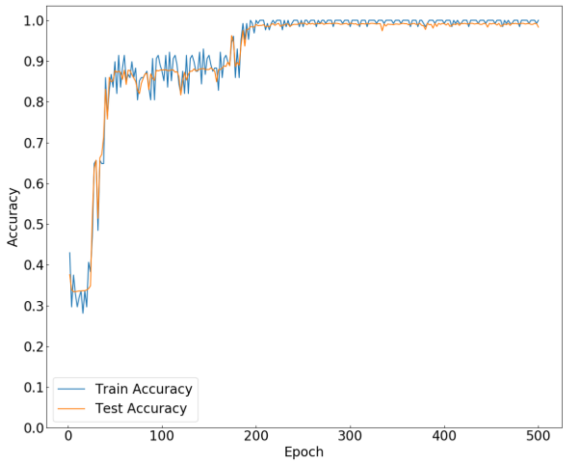


図 16　主要 3 アルゴリズムに対する学習過程
Fig. 16　Learning curve for three algorithms.

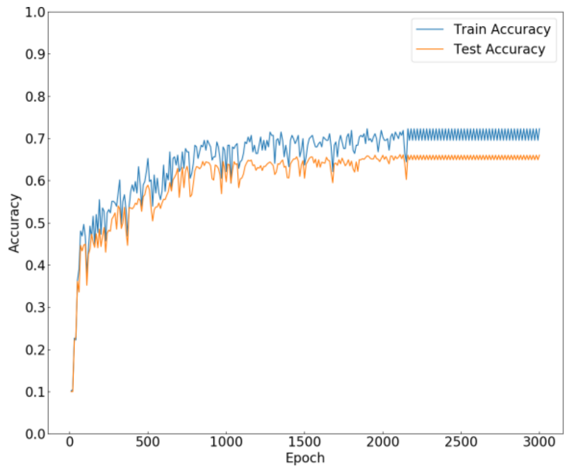


図 18　10 のアルゴリズムに対する学習過程
Fig. 18　Learning curve for ten algorithms.

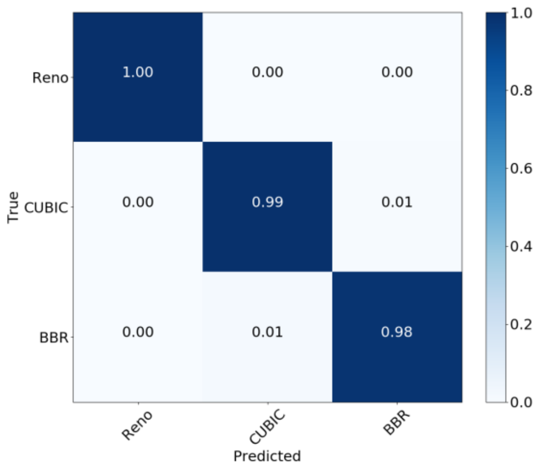


図 17　主要 3 アルゴリズムに対する混同行列
Fig. 17　Confusion matrix for three algorithms.

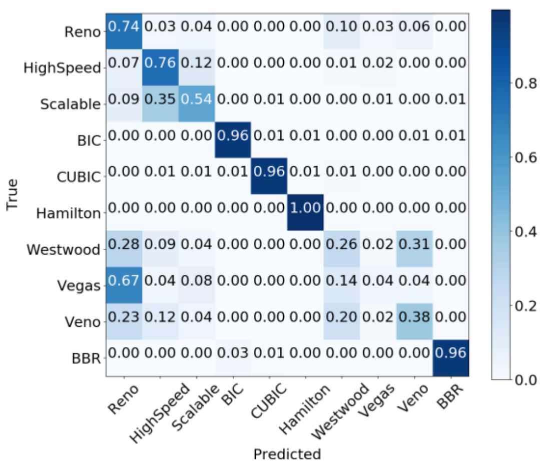


図 19　10 のアルゴリズムに対する混同行列
Fig. 19　Confusion matrix for ten algorithms.

識別精度である．TCP Reno は識別精度 1.0 で推定できた．また CUBIC TCP は 0.99 の精度で推定され，TCP BBR に 0.01 の割合で誤識別された．TCP BBR は 0.98 の精度で推定され，0.01 の割合で CUBIC TCP に誤識別された．これらの結果から，RNN に基づく識別器は TCP Reno, CUBIC TCP，TCP BBR の 3 つの輻輳制御アルゴリズムの推定に対して，有効に動作すると判断できる．

4.3.3　10 種類の輻輳制御アルゴリズムに対する推定結果

第 2 の実験として，4.2 節に示した 10 の輻輳制御アルゴリズムを対象として，その推定を行った．この実験ではミニバッチサイズとして 256 を用いた．

図 **18** に 10 の輻輳制御アルゴリズム全体の推定における学習曲線を示す．図 16 と同様に横軸はエポック数，縦軸は識別精度である．図の青線が学習トライアル，赤線がテストトライアルにおける識別精度である．この場合は 3,000 までのエポック数を示している．10 の輻輳制御アルゴリズム全体に対しては，トライアル回数が 2,200 を超えると，学習トライアルにおいて 0.7，テストトライアルに対して 0.65 の識別精度を得ている．本論文では，できるだけ高い精度での輻輳制御アルゴリズムの推定を目指しているため，この結果は高いとはいえない．

図 **19** にこの実験における混同行列を示す．10 の輻輳制御アルゴリズムのうち，BIC TCP, CUBIC TCP, Hamilton TCP，TCP BBR については有効に識別することができている．一方，HighSpeed TCP と Scalable TCP は混同して推測されてしまっている．この理由は，採用した実験環境においては，両者の振舞いが同様であったためと考えられる．

また，TCP Reno, TCP Westwood+, TCP Vegas, TCP Veno については，互いに混同して識別されている．これは，これらのアルゴリズムが TCP Reno と同様な AIMD（Additive Increase Multiplicative Decrease）に従っているためであると考えられる．TCP Westwood+ は cwnd を減少させる動作以外は TCP Reno と同様である．提案方法では輻輳回避フェーズの cwnd の増加方法のみに着目しているため，TCP Reno と区別することが困難であったと考えられる．また TCP Vegas と TCP Veno については，RTT

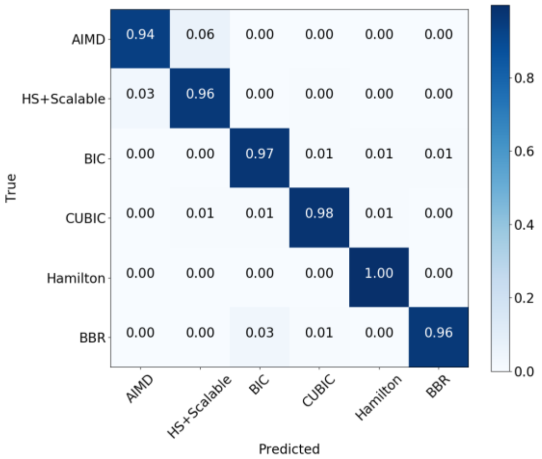


図 20　グループ化したアルゴリズムに対する混同行列

Fig. 20　Confusion matrix for grouped algorithms.

の増加が生じた場合に TCP Reno と異なる動作をする．しかし，4.1 節に述べた実験環境では，人為的に発生させたパケット損失により輻輳を模擬しているため，RTT の増加が生ずる状況は発生しにくい．このため Reno との区別が困難であったと考えられる．

そこで，類似した輻輳制御アルゴリズムについては同一のカテゴリとして推定するという方針の下で再度評価を行った．その結果を**図 20** に示す．具体的には，TCP Reno，TCP Westwood+，TCP Vegas，TCP Veno を AIMD というカテゴリとし，HighSpeed TCP と Scalable TCP を 1 つのカテゴリとした．この結果を見ると，互いを正しく識別できていることが分かる．

5.　おわりに

本論文では，リカレントニューラルネットワークを用いて TCP 輻輳制御アルゴリズムを推定する方法の提案とその結果について述べた．データセグメントと ACK セグメントの双方を含む双方向の通信ログから，パケット損失が生じていないロスレス区間に対して RTT ごとの輻輳ウィンドウサイズを推定し，それを正規化したものを RNN ベースの識別器への入力とした．TCP Reno，CUBIC TCP，TCP BBR の 3 つのアルゴリズムを対象とした推定では，互いを区別することが可能であった．また Linux オペレーティングシステムで実装されている 10 の輻輳制御アルゴリズムに対しては，類似したグループに分類することにより，グループごとの推定が可能であった．

本論文の実験では，学習用およびテスト用のデータを収集するにあたり，人為的なパケット損失を用いて輻輳を模擬していた．しかし，より現実的なデータを収集するためには，実際の輻輳により遅延が増加した状況などを対象とする必要がある．さらに実際のインターネットを介した TCP フローに対する輻輳制御アルゴリズムの推定を行う必要もある．これらを今後の課題としたい．

参考文献

[1] Afanasyev, A. et al.: Host-to-Host Congestion Control for TCP, *IEEE Commun. Surveys & Tutorials*, Vol.12, No.3, pp.304–342 (2010).

[2] Javobson, V.: Congestion Avoidance and Control, *ACM SIGCOMM Comp. Commun. Review*, Vol.18, No.4, pp.314–329 (1988).

[3] Stevens, W.R.: TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms, IETF RFC 2001 (1997).

[4] Floyd, S., Henderson, T. and Gurtov, A.: The NewReno Modification to TCP's Fast Recovery Algorithm, IETF RFC 3728 (2004).

[5] Floyd, S.: HighSpeed TCP for Large Congestion Windows, IETF RFC 3649 (2003).

[6] Kelly, T.: Scalable TCP: improving performance in high-speed wide area networks, *ACM Comp. Commun. Review*, Vol.32, No.2, pp.83–91 (2003).

[7] Xu, L., Harfoush, K. and Rhee, I.: Binary increase congestion control (BIC) for fast long-distance networks, *Proc. IEEE INFOCOM 2004*, Vol.4, pp.2514–2524 (2004).

[8] Ha, S., Rhee, I. and Xu, L.: CUBIC: A New TCP-Friendly High-Speed TCP Variant, *ACM SIGOPS Operating Systems Review*, Vol.42, No.5, pp.64–74 (2008).

[9] Leith, D. and Shorten, R.: H-TCP: TCP for high-speed and long distance networks, *Proc. Protocols for Fast Long Distance Networks* (*PFLDnet*), pp.1–16 (Mar. 2004).

[10] Grieco, L. and Mascolo, S.: Performance evaluation and comparison of Westwood+, New Reno, and Vegas TCP congestion control, *ACM Comp. Commun. Review*, Vol.34, No.2, pp.25–38 (2004).

[11] Brakmo, L. and Perterson, L.: TCP Vegas: End to End Congestion Avoidance on a Global Internet, *IEEE J. Selected Areas in Commun.*, Vol.13, No.8, pp.1465–1480 (1995).

[12] Fu, C. and Liew, S.: TCP Veno: TCP Enhancement for Transmission Over Wireless Access Networks, *IEEE J. Selected Areas in Commun.*, Vol.21, No.2, pp.216–228 (2003).

[13] Cardwell, N. et al.: BBR: Congestion-Based Congestion Control, *ACM Queue*, Vol.14 No.5, pp.20–53 (2016).

[14] Kato, T. et al.: Comparing TCP Congestion Control Algorithms Based on Passively Collected Packet Traces, *Proc. 10th Int. Conf. Systems and Networks Communications* (*ICSNC 2015*), pp.135–141 (Nov. 2015).

[15] 加藤聡彦ほか：輻輳ウィンドウとその増加分に着目したパッシブな TCP 輻輳制御アルゴリズムの推定方法，情報処理学会論文誌，Vol.60, No.2, pp.479–490 (2019).

[16] 大関尚紀，山本　嶺，大坐畠智，加藤聡彦：リカレントニューラルネットワークを用いた TCP 輻輳制御アルゴリズムの推定，信学技報，Vol.118, No.465, pp.385–390 (2019).

[17] Ohzeki, N., Yamamoto, R., Ohzahata, S. and Kato, T.: Estimating TCP Congestion Control Algorithms from Passively Collected Packet Traces Using Recurrent Neural Network, *Proc. 16th Int. Joint Conf. e-Business and Telecommunications* (*ICETE 2019*), Vol.1: DCNET, ICE-B, OPTICS, SIGMAP and WINSYS, pp.33–42 (2019).

[18] Paxson, V.: Automated Packet Trace Analysis of TCP Implementations, *ACM Comp. Commun. Review*, Vol.27, No.4, pp.167–179 (1997).

[19] Kato, T. et al.: Design of Protocol Monitor Emulating

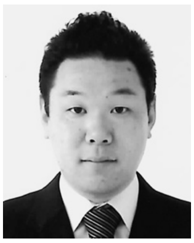
Behaviors of TCP/IP Protocols, *Proc. 10th Int. Workshop on Testing of Communicating Systems* (*IWTCS '97*), pp.416–431 (Sep. 1997).
[20] Jaiswel, S. et al.: Inferring TCP Connection Characteristics Through Passive Measurements, *Proc. INFOCOM 2004*, pp.1582–1592 (Mar. 2004).
[21] 大塩純平，阿多信吾，岡　育生：クラスター分析に基づく各種 TCP バージョンの識別手法，信学技報，Vol.108, No.457, NS2008-179, pp.205–210 (2009).
[22] Oshio, J., Ata, S. and Oka, I.: Identification of Different TCP Versions Based on Cluster Analysis, *Proc. 18th Int. Conf. Computer Communications and Networks* (*ICCN 2009*), pp.1–6 (Aug. 2009).
[23] Yang, P. et al.: TCP Congestion Avoidance Algorithm Identification, *Proc. 2011 Int. Conf. Distributed Computing Systems* (*ICDCS '11*), pp.310–321 (Jun. 2011).
[24] Klaine, P., Imran, M., Onireti, O. and Souza, R.: A Survey of Machine Learning Techniques Applied to Self-Organizing Cellular Networks, *IEEE Commun. Surveys & Tutorials*, Vol.19, No.4, pp.2392–2431 (2017).
[25] Buczak A. and Guven, E.: A Survey of Data Mining and Machine Learning Methods for Cyber Security Intrusion Detection, *IEEE Commun. Surveys & Tutorials*, Vol.18, No.2, pp.1153–1176 (2016).
[26] Nakao, A. and Du, P.: Toward In-Network Deep Machine Learning for Identifying Mobile Applications and Enabling Application Specific Network Slicing, *IEICE Trans. Commun.*, Vol.E101-B, No.7, pp.1536–1543 (2018).
[27] Edalat, Y., Ahn, J. and Obraczka, K.: Smart Experts for Network State Estimation, *IEEE Trans. Network and Service Management*, Vol.13, No.3, pp.622–635 (2016).
[28] Mirza, M., Sommers, J., Barford, P. and Zhu, X.: A Machine Learning Approach to TCP Throughput Prediction, *IEEE/ATM Trans. Networking*, Vol.18, No.4, pp.1026–1039 (2010).
[29] Chung, J., Han, D., Kim, J. and Kim, C.: Machine Learning based Path Management for Mobile Devices over MPTCP, *Proc. 2017 IEEE Int. Conf. Big Data and Smart Computing* (*BigComp 2017*), pp.206–209 (Feb. 2017).
[30] Hagos, D., Engelstad, P., Yazidi, A. and Kure, O.: A Machine Learning Approach to TCP State Monitoring from Passive Measurements, *Proc. 2018 Wireless Days*, pp.164–171 (Apr. 2018).
[31] Yamamoto, A. et al.: Prediction of Traffic Congestion on Wired and Wireless Networks Using RNN, *Proc. 13th Int. Conf. Ubiquitous Information Management and Communication* (*IMCOM 2019*), pp.315–328 (May 2019).
[32] Hochreiter, S. and Schimidhuber, J.: Long short-term memory, *Neural Computation*, Vol.9, No.8, pp.1735–1780 (1997).
[33] iPerf3: iPerf—The ultimate speed test tool for TCP, UDP and SCTP, available from ⟨https://iperf.fr/⟩ (2019).
[34] Dpkt: dpkt, available from ⟨https://dpkt.readthedocs.io/en/latest/⟩ (2019).
[35] TensorFlow: TensorFlow, available from ⟨https://www.tensorflow.org/⟩ (2019).
[36] Google Colab: Google Colaboratory, available from ⟨https://colab.researach.google.com/⟩ (2019).

大関 尚紀

平成 29 年芝浦工業大学工学部卒業，平成 31 年電気通信大学大学院情報理工学研究科博士前期課程修了．同年（株）ドワンゴ入社．電気通信大学大学院在学中に，機械学習を用いたパケットトレースからの TCP 輻輳制御アルゴリズムの推定手法について研究を進める．

山本 嶺

平成 19 年芝浦工業大学システム工学部電子情報システム工学科卒業，平成 21 年同大学大学院電気電子情報工学専攻修士課程修了．平成 25 年早稲田大学大学院博士課程修了．平成 22～26 年同大学国際情報通信研究科助手を経て，現在，電気通信大学大学院情報理工学研究科准教授，博士（国際情報通信学）．主に無線アドホック・センサネットワークに関する研究に従事．平成 26 年から早稲田大学国際情報通信研究センター招聘研究員．平成 22 年電子情報通信学会学術奨励賞受賞．



大坐畠 智　（正会員）

平成 15 年筑波大学大学院博士課程工学研究科電子・情報工学専攻修了，同年東京農工大学工学部情報コミュニケーション工学科助手，平成 19 年同大学助教，平成 21 年電気通信大学大学院情報システム学研究科准教授．ネットワーク性能評価の研究に従事．平成 17 年度電子情報通信学会学術奨励賞受賞，IEEE，ACM，IEICE 各会員．博士（工学）．

加藤 聰彦　（正会員）

昭和 53 年東京大学工学部電気工学科卒業．昭和 58 年同大学大学院博士課程修了．同年国際電信電話（現，KDDI）（株）入社．平成 14 年まで同社研究所および（株）KDDI 研究所において，OSI，ATM，インターネットなどの研究に従事．平成 14 年より，電気通信大学大学院情報システム学研究科助教授．現在，同大学教授．インターネットやアドホックネットワーク等の通信プロトコルの研究に従事．工学博士．