TCP Congestion Control Algorithm Estimation by Deep Recurrent Neural Network and Its Application to Web Servers on Internet

Takuya Sawada, Ryo Yamamoto, Satoshi Ohzahata, and Toshihiko Kato
Graduate School of Informatics and Engineering
University of Electro-Communications
Tokyo, Japan
e-mail: sawada@net.lab.uec.ac.jp, ryo-yamamoto@uec.ac.jp, ohzahata@uec.ac.jp, kato@net.lab.uec.ac.jp

Abstract — **Recently, as various types of networks are introduced, a number of Transmission Control Protocol (TCP) congestion control algorithms have been adopted. Since the TCP congestion control algorithms affect traffic characteristics 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 automatic processing is indispensable. This paper proposes a machine learning based method for estimating TCP congestion control algorithms. The proposed method uses a passively collected packet traces including both data and ACK segments, and calculates a time sequence of congestion window size for individual TCP flows contained in the traces. We use a classifier based on deep recurrent neural network in the congestion control algorithm estimation. As the results of applying the proposed classifier to ten congestion control algorithms, we obtained high accuracy of classification compared with our previous work using recurrent neural network with one hidden layer. This paper also describes the results of applying the classifier to popular web servers for checking the distribution of congestion control algorithms in the real world.**

Keywords — TCP; Congestion Control; Deep Recurrent Neural Network.

I. INTRODUCTION

This paper is an extension of our previous paper [1], which was presented at the IARIA conference EMERGING 2022.

Along with the introduction of various types of networks, such as a long-haul high speed network and a wireless mobile network, a number of TCP congestion control algorithms have been designed, implemented, and widely spread [2]. Since the congestion control was introduced [3], a few algorithms, such as TCP Tahoe [4], TCP Reno [4], and NewReno [5], have been used commonly for some decades. Recently, new algorithms have been introduced and deployed. For example, HighSpeed TCP [6], Scalable TCP [7], BIC TCP [8], CUBIC TCP [9], and Hamilton TCP [10] are designed for high speed and long delay networks. TCP Westwood+ [11] is designed for lossy wireless links. While those algorithms are based on packet losses, TCP Vegas [12] triggers congestion control against an increase of Round-Trip Time (RTT). TCP Veno [13] combines loss based and delay based approaches in such a way that congestion control is triggered by packet losses but the delay determines how to grow the congestion window (cwnd). In 2016, Google proposed a new algorithm called TCP BBR (Bottleneck Bandwidth and Round-trip propagation time) [14] to solve problems mentioned by conventional algorithms.

Since TCP traffic is a majority in the Internet traffic and the TCP congestion control algorithms characterize the behaviors of individual flows, the estimation of congestion control algorithms for TCP traffic is important for network operators. It can be used in various purposes such as the traffic trend estimation, the planning of Internet backbone networks, and the detection of malicious flows violating congestion control algorithms.

The approaches for congestion control algorithm estimation are categorized into the passive approach and the active approach. The former estimates algorithms from packet traces passively collected in the middle of networks. In the latter approach, a test system communicates with a target system with a specially designed test sequence. Although the active approach is capable to identify various congestion control algorithms proposed so far, it does not fit the algorithm estimation of real TCP flows. On the other hand, generally speaking, the detecting capability of passive approaches is relatively low comparing with the active approach.

Previously, we proposed a passive method that can estimate a number of congestion control algorithms [15][16]. In this proposal, we focused on the relationship between the estimated congestion window size and its increment. Their relationship is indicated as a graph and the congestion control algorithm is estimated based on the shape of the graph. Our proposal succeeded to identify eight congestion control algorithms implemented in the Linux operating system, including recently introduced ones.

However, the identification is performed manually by human inspectors, and so it is difficult to deal with a large number of TCP flows. So, we proposed a machine learning based classifier estimating the TCP congestion control algorithms using TCP packet traces in two steps [17][1]. In our first trial [17], we used a conventional Recurrent Neural Network (RNN) with one hidden layer. From a packet trace, we estimate the relationship of cwnd values and their increment with congestion control algorithm labels, and apply the results to a RNN classifier for training. Using the RNN classifier, we estimate the algorithms for other packet traces. We obtained a relatively good estimation result from the RNN classifier, but we could not classify similar algorithms, such as TCP Reno and Vegas. In our second trial [1], we proposed a revised version of machine learning classifier for automatic estimation of congestion control algorithms. Here, we adopted a Deep Recurrent Neural Network (DRNN) with multiple hidden layers. We also applied a hyper parameter tuning for the classifier. We picked up ten congestion control algorithms mentioned above and showed that our new approach can estimate those algorithms better than our first trial.

This evaluation of our approach was performed in a simple in-lab test network environment. Applying our approach to servers in the Internet will be effective to show its effectiveness and to investigate the trends of TCP congestion control algorithms in the real world. We estimated the congestion control algorithms of 20,000 web servers listed in Alexa Top Sites 1,000,000 offered by Alexa Traffic Rank [18]. This paper also shows the results of this estimation.

The rest of this paper is organized as follows. Section II gives some background information including the conventional studies on the congestion control estimation and the machine learning applied for the network areas. Section III describes the proposed method and Section IV gives the performance evaluation results. Section V discusses the results of the estimation of Internet web servers by the proposed method. In the end, Section VI concludes this paper.

II. BACKGROUNDS

A. Studies on TCP Congestion Control Algorithm Estimation

The proposals on the passive approach in the early stage [19-21] estimate the internal state and variables, such as cwnd and ssthresh (slow start threshold), in a TCP sender from bidirectional packet traces. They emulate the TCP sender's behavior from the estimated state/variables according to the predefined TCP state machine. But, they considered only TCP Tahoe, Reno and New Reno and did not handle any of recently introduced algorithms. Oshio et al. [22] proposed a method to discriminate one out of two different TCP congestion control algorithms randomly selected from fourteen algorithms implemented in the Linux operating system. This method keeps track of changes of cwnd from a packet trace and to extract several characteristics, such as the ratio of cwnd being incremented by one packet. Although this method targets all of the modern congestion control algorithms, they assumed that the discriminator knows two algorithms contained in the packet trace.

The active approaches, on the other hand, are able to identify more TCP congestion control algorithms including those introduced recently. Yang et al. [23] proposed a tool called CAAI (TCP Congestion Avoidance Algorithm Identification), which could identify recent TCP congestion control algorithms at first. It makes a web server send 512 data segments under the controlled network environment, and observes the number of data segments contiguously transmitted. From those results, it estimates the window growth function and the decrease parameter to determine the congestion control algorithm. Mishra et al. [24] proposed another active approach based tool called Gordon. It makes a web server send several data segments and causes a packet loss intentionally. By analyzing the retransmission and the following congestion avoidance sequence, it obtains cwnd values. It then estimates congestion control algorithms based on the shape of cwnd time sequence graph, the increase of cwnd, and the back-off at packet loss. It could estimate the recent congestion control algorithms and was applied to the 20,000 web servers listed in the Alexa Top Sites.

Our previous proposals [15][16] estimated cwnd in RTT intervals from bidirectional packet traces, in the similar way with the other methods. Different from other methods, we focused on the incrementing situation of estimated cwnd values. From the definition of individual congestion control algorithms, the graph of cwnd increments vs. cwnd has their characteristic forms. For example, in the case of TCP Reno, the cwnd increment is always one segment. In the case of CUBIC TCP, the graph of cwnd increment follows a $\sqrt[3]{cwnd^2}$ curve. In this way, we proposed a way to discriminate eight congestion control algorithms in the Linux operating system.

B. Studies on Application of Machine Learning to TCP

Recently, several papers focus on applying the machine learning to TCP analysis. Edalat et al. [25] proposed a method to estimate RTT using the fixed-share approach from measured RTT samples. Mirza et al. [26] estimated the future throughput of TCP flow using the support vector regression from measured available bandwidth, queueing delay, and packet loss rate. Chung et al. [27] proposed a machine learning based multipath TCP scheduler based on the radio strength in wireless LAN level, wireless LAN data rate, TCP throughput, and RTT with access point, by the random decision forests.

III. PROPOSED METHOD

A. Estimation of cwnd values at RTT interval

In the passive approach, packet traces are collected at some monitoring point inside a network. So, the time associated with a packet is not the exact time when the node focused sends/receives the packet. Our scheme adopts the following approach to estimate cwnd values at RTT intervals using the TCP time stamp option.

- Pick up an ACK segment in a packet trace. Denote this ACK segment by *ACK1*.
- Search for the data segment whose TSecr (time stamp echo reply) is equal to TSval (time stamp value) of *ACK1*. Denote this data segment by *Data1*.
- Search for the ACK segment that acknowledges *Data1* for the first time. Denote this ACK segment by *ACK2*. Denote the ACK segment prior to *ACK2* by *ACK1'*.
- Search for the data segment whose TSecr is equal to TSval of *ACK2*. Denote this data segment by *Data2*.

From this result, we estimate a cwnd value at the timing of receiving *ACK1* as in (1).

$$cwnd = \left\lfloor \frac{seq\ in\ Data2 - ack\ in\ ACK1'}{MSS} \right\rfloor \text{ (segments)} \qquad (1)$$

Here, *seq* means the sequence number, *ack* means the acknowledgment number of TCP header, and *MSS* is the maximum segment size (MSS). $\lfloor a \rfloor$ is the truncation of a.

Figure 1 shows an example of cwnd estimation. In this figure, MSS is 1024 byte. Data segments are indicated by

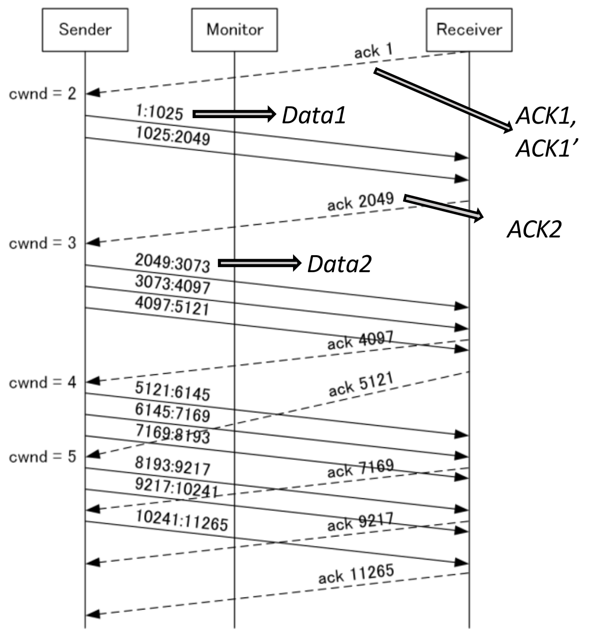


Figure 1. Example of cwnd estimation.

solid lines with "sequence number : sequence number + MSS." ACK segments are indicated by dash lines with acknowledgment number. When "ack 1" is picked up, data segment "1:1024" is focused on as *Data1* above. ACK segment "ack 2049" responding the data segment corresponds to *ACK2*. The ACK segment before this ACK segment (*ACK1'* above) is "ack 1" again. *Data2* in this case is "2049:3073." So, the estimated cwnd is (2049 – 1)/1024 = 2. Similarly, for the following two RTT intervals, the estimated RTT values are (5121 – 2049)/1024 = 3 and (10241 –5121) /1024= 5.

B. Selection and Normalization of Input Data to Classifier

When a packet is lost and retransmitted, cwnd is decreased. In order to focus on the cwnd handling in the congestion avoidance phase, we select a time sequence of cwnd between packet losses. We look for a part of packet trace where the sequence number in the TCP header keeps increasing. We call this duration without any packet losses *non-loss duration*. We use the time variation of estimated cwnd values during one non-loss duration as an input to the classifier. However, the length of non-loss duration differs for each duration, and the range of cwnd values in a non-loss duration also differs from one to another. So, we select and normalize the time scale and the cwnd value scale for one non-loss duration.

The algorithm for selecting and normalizing input to classifier is given in Figure 2. In this algorithm, the input E is as time sequence of cwnd values estimated from one packet trace. The input *InputLength* is a number of samples in one input to the classifier. In this paper, we used 128 as *InputLength*. This is because we think that the cwnd vs time curve can be drawn by 128 points. In the beginning, the time sequence of cwnd is divided at packet losses, and the divided sequences are stored in a two dimensional array *S*. Next, the

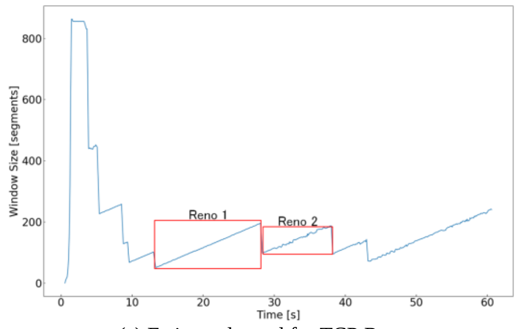
```
Algorithm 1
1. function Normalize (E, InputLength)
2.      S <- DivideAtLoss(E)
3.      Delete(S[0])
4.      S <- SortBySequenceLength(S)
5.      for t = 0 to Len(S) − 1 do
6.           S <- MinMaxNormalization(S)
7.      end for
8.      I <- Array(Len(S))
9.      for t = 0 to Len(S) − 1 do
10. 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

Figure 2. Selection/normalization algorithm.

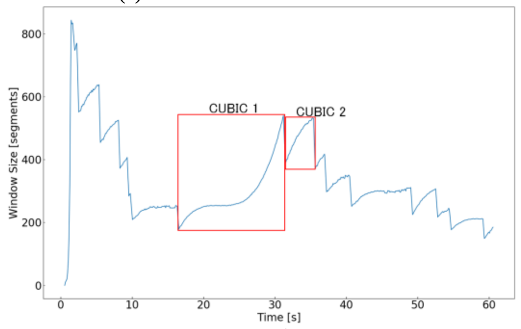
first sequence *S*[0] is removed, because we focus only on the congestion avoidance phase. Then *S* is reordered according to the length of cwnd sequence. Then the cwnd values for one sequence *S*[t] are normalized between 0 and 1. The normalization is performed in the following way.

Let $w_{max}[t] = \max(S[t][u])$
for $u = 0 \cdots \mathrm{Len}(S[t]) - 1$, and
$$w_{min}[t] = \min(S[t][u])$$
for $u = 0 \cdots \mathrm{Len}(S[t]) - 1$.
Each cwnd value in S[t] is normalized by
$$S[t][u] \leftarrow \frac{S[t][u] - w_{min}[t]}{w_{max}[t] - w_{min}[t]}.$$

After that, the cwnd values are resampled into the number of *InputLength* (128 in this paper). This is done by the loop



(a) Estimated cwnd for TCP Reno



(b) Estimated cwnd for CUBIC TCP

Figure 3. Examples of cwnd estimation.

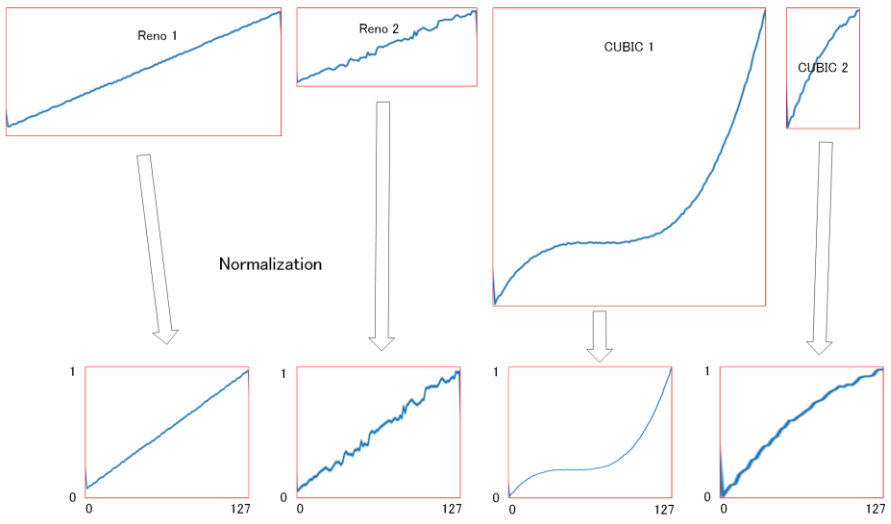


Figure 4. Examples of normalization.

between step 11 and step 15. As a result, a cwnd sequence in *S*[t] is converted to an array *I*[t] with 128 elements. By this algorithm, all of the time sequences of cwnd values are the arrays with 128 elements whose value is between 0 and 1.

Figure 3 shows some examples of cwnd estimation. Figure 3 (a) and (b) show the estimated cwnd time sequences for TCP Reno and CUBIC TCP, respectively. We focus on the non-loss durations as described above. Reno 1, Reno 2, CUBIC 1, and CUBIC 2 in the figure are examples. The size of these sequences differ from each other, both for the time scale and the scale of cwnd. Therefore, it is necessary to normalize these sequences.

Figure 4 shows the results of the normalization for the examples shown in Figure 3. Different scale of cwnd time sequences are transformed into a canonical form with 128 samples in the range of 0 through 1.

C. *DRNN Based Classifier for Congestion Control Algotithm Estimation*

We used DRNN for constructing the classifier, which has three hidden layers and whose output layer defines the TCP congestion control algorithms. Among the RNN technologies, we pick up the long short-term memory mechanism [28], which was proposed to handle a relatively long time sequence of data. The input is a normalized time sequence of cwnd as described above, with using labels of congestion control algorithms represented by one-hot vector.

In our previous work, we selected the hyper parameters given in Table I. In the work presented in this paper, we select the hyper parameter ranges shown in Table II. The input length is the same as that of the previous work. We use three hidden layers and the number of neurons are as specified in the table. As for the optimizer, the learning rate and the weight decay, we propose the alternatives shown in the table. We perform the hyper parameter tuning based on the target area in this table.

In the training of the classifier, we use the mini-batch method, which selects a specified number of inputs randomly

TABLE I. HYPER PARAMETERS OF CLASSIFIER IN OUR PREVIOS WORK.

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

TABLE II. HYPER PARAMETER RANGES IN THIS WORK.

Parameter	Value
Input Length	128
Hidden Layers	3
Hidden Neurons	1st./2nd: 512, 3rd: 256
Optimizer	Adam or MomentumSGD
Learning Rate	$[10^{-5}, 10^{-1}]$
Weight Decay	$[10^{-10}, 10^{-3}]$

from the prepared training data. The mini-batch size will be determined for individual training. The training will be continued until the result of the loss function becomes smaller than the learning rate.

IV. Experimental Results

A. *Experimental Setup*

Figure 5 shows the experimental configuration for collecting time sequence of cwnd values. A data sender, a data receiver, and a bridge are connected via 100 Mbps Ethernet links. In the bridge, 50 msec delay for each direction is inserted. As a result, the RTT value between the sender and the receiver is 100 msec. In order to generate packet losses that will invoke the congestion control algorithm, packet losses are inserted randomly at the bridge. The average packet loss ratio is 0.01%. The data transfer is performed by use of iperf3 [29], executed in both the sender

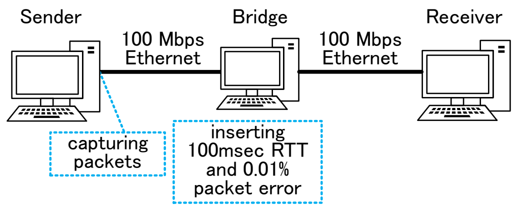


Figure 5. Experiment configuration.

and the receiver. The packet traces are collected by use of tcpdump at the sender's Ethernet interface. We use the Python 3 dpkt module [30] for the packet trace analysis. We changed the congestion control algorithm at the sender by use of the sysctl command provided by the Linux operating system.

The targeted congestion control algorithms are TCP Reno, HighSpeed TCP, BIC TCP, CUBIC TCP, Scalable TCP, Hamilton TCP, TCP Westwood+, TCP Vegas, TCP Veno, and BBR. We collected more than 1,500 samples for individual algorithms, and prepared 1,000 samples as training data, 250 samples as verifying data, and 250 samples as test data.

B. *Results of Congestion Control Algorithm Estimation*

First, we re-evaluated the performance of our previous approach. The result is shown in Figure 6. The total accuracy for ten congestion control algorithms was 42.8%, which is rather worse than the result described in our previous paper [17]. This means that our previous classifier will depend largely on the prepared training data.

So, we applied the same training data and verifying data for a DRNN based classifier with three hidden layers and selected optimal values for the hyper parameters mentioned in Table II. We tried to look for optimal values 100 times by the mini-batch method using 256 as the mini-batch size.

TABLE III. TUNED UP HYPER PARAMETER VALUES.

Parameter	Value
Optimizer	Adam
Learning Rate	0.0015967736
Weight Decay	2.967486×10^{-8}

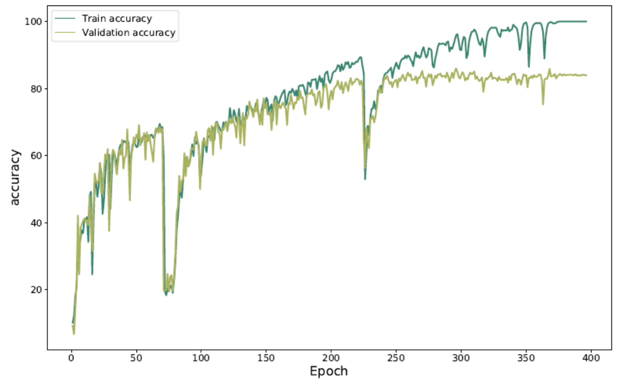


Figure 7. Learning curve for ten congestion control algorithms.

Table III shows the values of hyper parameters obtained by this tuning.

Figure 7 shows the learning curve for ten congestion control algorithms using the DRNN based classifier with the selected hyper parameter values. The horizontal axis of this figure indicates the epoch, which is the number of training and verifying trials. The vertical axis indicates the accuracy for the training process and the verifying process. The blue line is the accuracy for the training process and the green line is for the verification process. This result shows that the classifier learns the model for estimating congestion control algorithms. Figure 8 shows the confusion matrix for this experiment. By comparing Figures 6 and 8, we can conclude that the DRNN based classifier estimates the congestion control algorithms much better than our previous classifier.

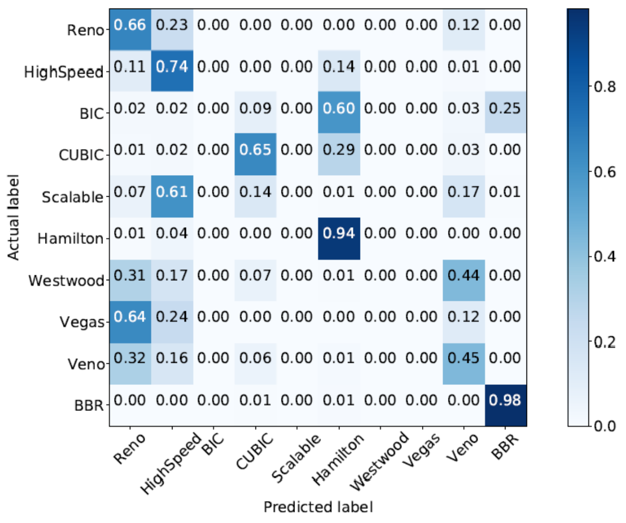


Figure 6. Confusion matrix for previous approach.

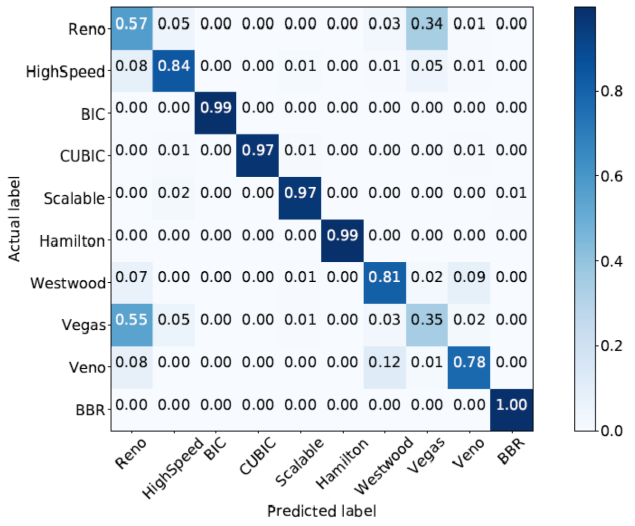


Figure 8. Confusion matrix for this approach.

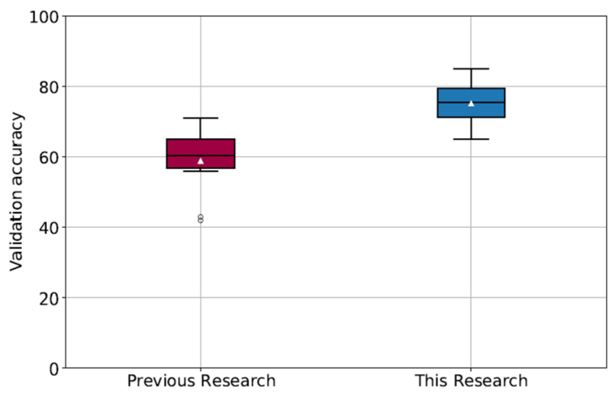


Figure 9. Generalization accuracy of previous work and this work.

The total accuracy was 82.9%, which is higher than that of our previous work. The only problem is that it still confuses TCP Reno and TCP Vegas. Further studies are required.

As the last analysis, we evaluated the generalization accuracy for our previous work and this work using the 10-fold cross-validation. We divided the training data into ten folds, and selected one fold for the validation and used the rest folds for the training. Figure 9 shows the result. The vertical axis is the validation accuracy. In our previous classifier, the validation accuracy sometimes drops to 40%, although it goes up 70%. On the other hand, our new classifier provides 70% through 80% accuracy stably.

V. Congestion Control Algorithm Estimation of Web Servers on Internet

A. Steps

The next work is to apply our DRNN based classifier for estimating congestion control algorithms used by web servers on the Internet. Basically, we will use the classifier described in the previous sections. That is, the classifier is trained in the in-lab test network and applied to the estimation using data obtained from outside web servers.

For this purpose, we decided to take the following steps. The first step is to redesign our classifier to fit the estimation for real servers. Again, it should be mentioned that the training and testing of the redesigned classifier are performed using the in-lab network. This is because we do not know the congestion control algorithms of web servers on the Internet. The second step is to estimate the algorithms used by real web servers using our redesigned classifier.

B. Redesign of Classifier

As mentioned in Section II, Mishra et al. tried to estimate congestion control algorithms used by popular web servers given in the Alexa Top Site list based on an active approach. The measurements were made between July and October in 2019. This measurement found that thirteen algorithms are adopted by the target web servers. They include TCP Hybra [31], YeAH TCP [32], and TCP Illinois [33] besides ten algorithms mentioned in the previous section. It should be mentioned that CTCP (Compound TCP) [34], which was said to be adopted in Windows OS, is handled as TCP Illinois. So,

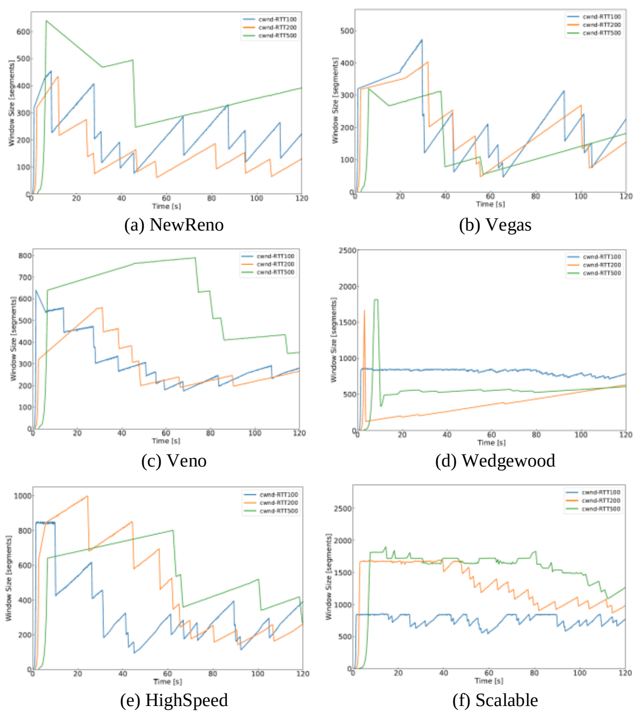


Figure 10. Estimated cwnd vs. time for different RTT values.

we needed to extend our classifier so as to include the added three congestion control mechanisms.

Another redesign point is RTT. The experiment shown in the previous section used 100 msec RTT inserted at the bridge. However, the real web server accesses take various values of RTT. So, for training our classifier in various RTT situations, we adopted 100 msec, 200 msec, and 500 msec as RTT values. Specifically, the delay of the half of individual RTT values are inserted for each direction at the bridge. The average packet loss rate inserted at the bridge is 0.01% for all cases. It should be mentioned that these values are constant ones throughout one experiment run.

Figure 10 shows some examples of estimated cwnd values. The blue, orange, and green lines indicate cwnd values for 100 msec, 200 msec, and 500 msec, respectively. By introducing different RTT values, we could obtain different behaviors of cwnd. Especially, this helped to discriminate NewReno, Vegas, Veno, Westwood, and HighSpeed, for which the classifier in the previous section was suffered from erroneous estimation.

We collected more than 2,400 samples for individual congestion control algorithms. We prepared 2,000 samples as training data and 400 samples as test data. Table IV shows the selected hyper parameter values for the redesigned classifier. LAMB (Layer-wise Adaptive Moments optimizer for Bach training) [35], adopted as the optimizer, is a training method for deep neural networks with large batch size.

Figure 11 shows the learning curve of the redesigned DRNN classifier for thirteen congestion control algorithms. The horizontal axis is the epoch and the vertical axis is the accuracy, similarly to Figure 7. This result shows that the redesigned classifier also learns the model for estimating the

TABLE IV. TUNED UP HYPER PARAMETER VALUES OF REDSIGNED CLASSIFIER.

Parameter	Value
Optimizer	LAMB
Learning Rate	0.01866181607680214
Weight Decay	$3.094144144895362 \times 10^{-7}$

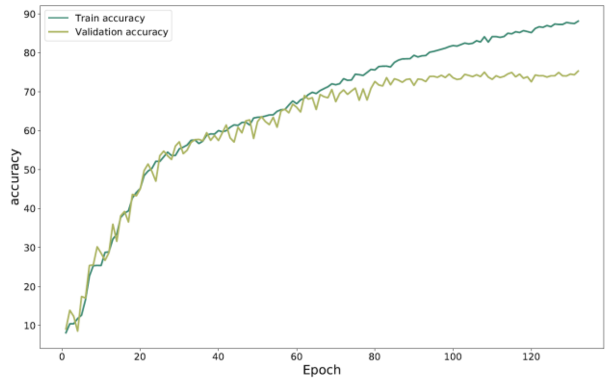


Figure 11. Learning curve for thirteen congestion control algorithms.

targeted algorithms. More specifically, the accuracies for training and validation become different from 80 epochs, and so the trained model will keep the status around that area.

Figure 12 shows the confusion matrix of the redesigned classifier. The total accuracy is 73.2 %, which is a little lower than the result of Figure 8, but it will be high enough to estimate a server's congestion control algorithm from a passively collected packet trace. The misestimation between Reno and Vegas decreased compared with the result of Figure 8. This is a result of introducing different RTT values. The confusion among Scalable, Illinois, and YeAH, the latter two are of which introduced newly, is one reason of decreasing the total accuracy. We used this classifier to categorize the congestion control algorithms used by actual web servers over the Internet.

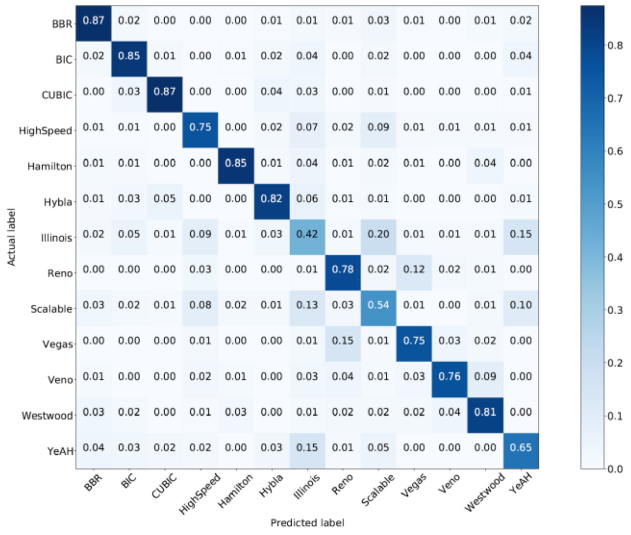


Figure 12. Confusion matrix for thirteen congestion control algorithms.

C. Results of Estimation for Real Web Servers

We applied our redesigned classifier to the web servers listed in the Alexa Top Site list. We examined the list in December 2020, and it should be noted that this service was terminated at May 2022. We selected 20,000 popular web servers in the list, and communicated with each server for 30 seconds through three minutes to obtain cwnd time sequences. Then, we applied the results to our classifier to estimate the congestion control algorithm of the individual server.

Figure 13 shows the distribution of estimated congestion control algorithms for 20,000 servers. TCP BBR and CUBIC TCP are popular algorithms, which occupy 29.9 % and 40.7 %, respectively. TCP Illinois and YeAH TCP follow them. As described above, it is possible that Illinois includes CTCP. On the other hand, NewReno, which used to be dominant, is not used any more.

Figure 14 shows the distribution of estimated algorithms for top 100 servers. In this case, TCP BBR occupies 41.0 % in the first place. CUBIC TCP is in the second place and occupies 26.0 %. The third is TCP Illinois. The reason that BBR and CUBIC change the place in top 100 servers will be that a lot of video distribution sites and Google related sites are included in top 100 servers. This means that the large portion of Internet TCP traffic will include BRR and CUBIC.

Several studies conducted the TCP congestion control algorithm classification in the past. Padhye et al. [36] proposed an active method called TBIT (TCP Behavior Inference Tool), in which a client test tool communicates with a web server with dropping data segments intentionally. It was applied to several web servers during 2000 and 2001. Medina et al. [37] measured the TCP variants by use of TBIT in 2004. Yang et al. applied their tool, CAAI, to several web servers to determine their TCP variants in 2011 [23]. Most recently, Mishra et al. applied their tool, Gordon, to the Alexa Top 20,000 servers as mentioned above [24].

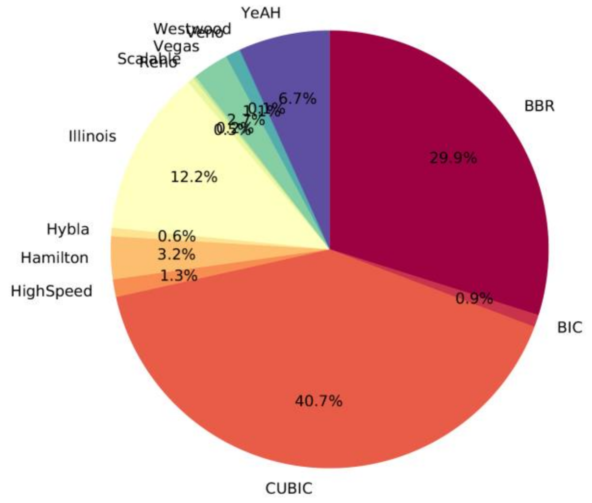


Figure 13. Distribution of estimated congestion control algorithms for AlexaTop-20000.

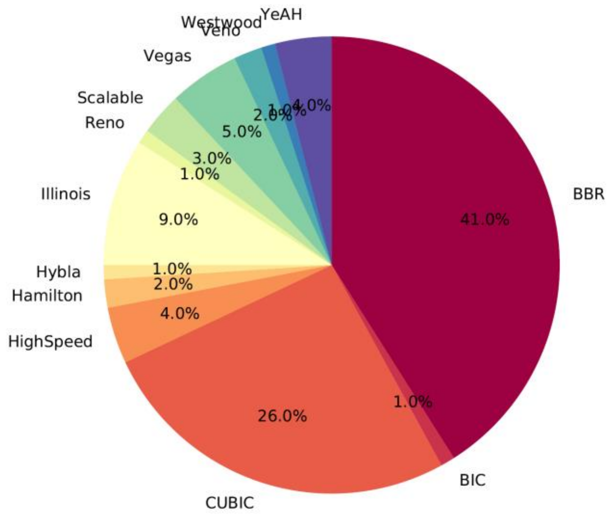


Figure 14. Distribution of estimated congestion control algorithms for AlexaTop-100.

Table V shows the results of classifying TCP variants by the previous studies and by us. This table is a modified version of the table given in [24] and our results are added to the modified one. Each entry contains a TCP variant and its rate/number. Variants are categorized into loss-based, delay-based, and rate-based ones. "Unknown etc." indicates the case that the method could not estimate the algorithm.

In the time frame of 2000s, only Reno-like TCP variations are identified. In the beginning of 2000s, NewReno was more popular than Reno and Tahoe [36]. In the middle of 2000s, the variants other than Reno types were increasing, that is, the unknown variants occupied 67% [37].

At the beginning of 2010s, a variety of TCP congestion control algorithms were introduced [23]. The Reno-type (AIMD) variants decreased, and instead, CUBIC was the most popular variant, and BIC, HighSpeed, and CTCP were adopted by some servers.

At the end of 2010s, CUBIC and BBR were top two variants [24]. Besides them, a few web servers were using Illinois including CTCP, YeAH, Hamilton, and Vegas/Veno. Our experiment used the measurement results performed in December 2020, and so the used web servers may be different from those used in the Gordon measurement, and the same server may change its algorithm. So, the results of ours and the Gordon's are a little different from each other. However, the trends of both results are similar. The most popular algorithms are CUBIC and BBR, and Illinois and YeAH follow. A certain amount of servers keep using the delay-based algorithms such as Vegas and Veno.

VI. CONCLUSIONS

This paper is an extended version of our conference paper [1] presented in IARIA EMERGING 2022. This paper provides two contributions.

The first, which is the contribution provided by the conference paper, is that we showed a result of TCP congestion control algorithm estimation using a Deep Recurrent Neural Network (DRNN) based classifier. From packet traces including both data segments and ACK segments, we derived a time sequence of cwnd values at RTT intervals without any packet retransmissions. By ordering the time sequences and normalizing in the time dimension and the cwnd value dimension, we obtained the input for the

TABLE V. CLASSIFICATIONS OF TCP VARIANTS IN SEVERAL STUDIES

	2001 [36]		2004 [37]		2011 [23]		2019 [24]		2020 (our results)	
Loss-based	NewReno	35% (1,571)	NewReno	25% (21,266)	AIMD	12.46% (623)	NewReno	0.80% (160)	NewReno	0.49% (98)
	Reno	21% (945)	Reno	5% (4,115)			Reno	-		
	Tahoe	26% (1,211)	Tahoe	3% (2,164)			Tahoe	-		
					CUBIC	22.30% (1,115)	CUBIC	30.70% (6,139)	CUBIC	40.69% (8,139)
					BIC	10.62% (531)	BIC	0.90% (181)	BIC	0.90% (181)
					HighSpeed	7.38% (369)	HighSpeed	in NewReno	HighSpeed	1.30% (260)
					Scalable	1.38% (69)	Scalable	0.20% (39)	Scalable	0.20% (39)
									Hybra	0.62% (124)
Delay-based					Vegas	1.16% (58)	Vegas	2.82% (564)	Vegas	2.65% (531)
					Westwood	2.08% (104)	Westwood	0% (0)	Westwood	0.05% (10)
					Illinois	0.56% (28)	Illinois	5.74% (1,148)	Illinois	12.23% (2,445)
	-	-	-	-	Veno	0.90% (45)	Veno	in Vegas	Veno	1.07% (214)
					YeAH	1.44% (72)	YeAH	5.81% (1,162)	YeAH	6.71% (1,342)
					Hamilton	0.36% (18)	Hamilton	2.28% (560)	Hamilton	3.15% (630)
					CTCP	6.68% (334)	CTCP	in Illinois		
Rate-based							BBR	17.75% (3,550)	BBR	29.84% (5,967)
	-	-	-	-	-	-	BBR G1.1	0.84% (167)		
							Akamai CC	5.51% (1,103)		
Unknown etc.		18% (822)		67% (56,479)		32.84% (1,642)		26.14% (5,227)		-
Total		100% (4,550)		100% (84,394)		100% (5,000)		100% (20,000)		100% (20,000)

note: BBR G1.1 indicates the Google dialect of BBR, and Akamai CC is a rate-based congestion control used by the Akamai content delivery network.

DRNN classifier. As the results of applying the proposed classifier for ten congestion control algorithms implemented in the Linux operating system, we showed that the DRNN based classifier can estimate ten algorithms effectively, with a problem that TCP Reno and TCP Vegas are difficult to discriminate. This result is much better than our previous classifier that used a simple recurrent neural network.

The second is an original contribution newly provided in this paper. We applied our DRNN based classifier to the estimation of congestion control algorithms used by 20,000 frequently accessed web servers identified by the Alexa Top Sites list. For this purpose, we redesigned our classifier so as to handle TCP Hybra, YeAH TCP, and TCP Illinois variants, and to include the situations with different RTT values. We confirmed that the redesigned classifier estimates thirteen variants with the accuracy of 73.2 %. Then we applied our classifier to 20,000 web servers listed in Alexa Top Sites. The results were that the top two variants were CUBIC and BBR, and that Illinois (including CTCP) and YeAH followed them. The results have the similar trends with the study conducted by Mishra et al. in 2019 [24], and this indicates that our estimation will be reasonable.

REFERENCES

[1] T. Sawada, R. Yamamoto, S. Ohzahata, and T. Kato, “Estimation of TCP Congestion Control Algorithms by Deep Recurrent Neural Network,” Proc. IARIA EMERGING 2022, pp. 19-24, 2022.

[2] A. Afanasyev, N. Tilley, P. Reiher, and L. Kleinrock, “Host-to-Host Congestion Control for TCP,” IEEE Commun. Surveys & Tutorials, vol. 12, no. 3, pp. 304-342, 2010.

[3] V. Jacobson, “Congestion Avoidance and Control,” ACM SIGCOMM Comp. Commun. Review, vol. 18, no. 4, pp. 314-329, 1988.

[4] W. R. Stevens, “TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algotithms,” IETF RFC 2001, 1997.

[5] S. Floyd, T. Henderson, and A. Gurtov, “The NewReno Modification to TCP’s Fast Recovery Algorithm,” IETF RFC 3728, 2004.

[6] S. Floyd, “HighSpeed TCP for Large Congestion Windows,” IETF RFC 3649, 2003

[7] T. Kelly, “Scalable TCP: Improving Performance in High-speed Wide Area Networks,” ACM SIGCOMM Comp. Commun. Review, vol. 33, no. 2, pp. 83-91, 2003.

[8] L. Xu, K. Harfoush, and I. Rhee, “Binary increase congestion control (BIC) for fast long-distance networks,” Proc. IEEE INFOCOM 2004, vol. 4, pp. 2514-2524, 2004.

[9] S. Ha, I. Rhee, and L. Xu, “CUBIC: A New TCP-Friendly High-Speed TCP Variant,” ACM SIGOPS Operating Systems Review, vol. 42, no. 5, pp. 64-74, 2008.

[10] D. Leith and R. Shorten, “H-TCP: TCP for high-speed and long distance networks,” Proc. Int. Workshop on PFLDnet, pp. 1-16, 2004.

[11] L. Grieco and S. Mascolo, “Performance evaluation and comparison of Westwood+, New Reno, and Vegas TCP congestion control,” ACM Computer Communication Review, vol. 34, no. 2, pp. 25-38, 2004.

[12] L. Brakmo and L. Perterson, “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.

[13] C. Fu and S. Liew, “TCP Veno: TCP Enhancement for Transmission Over Wireless Access Networks,” IEEE J. Sel. Areas in Commun., vol. 21, no. 2, pp. 216-228, 2003.

[14] N. Cardwell, Y. Cheng, C. S. Gumm, S. H. Yeganeh, and V. Jacobson, “BBR: Congestion-Based Congestion Control,” ACM Queue vol. 14 no. 5, pp. 20-53, 2016.

[15] T. Kato, A. Oda, S. Ayukawa, C. Wu, and S. Ohzahata, “Inferring TCP Congestion Control Algorithms by Correlating Congestion Window Sizes and their Differences,” Proc. IARIA ICSNC 2014, pp.42-47, 2014.

[16] T. Kato, A. Oda, C. Wu, and S. Ohzahata, “Comparing TCP Congestion Control Algorithms Based on Passively Collected Packet Traces,” Proc. IARIA ICSNC 2015, pp. 145-151, 2015.

[17] N. Ohzeki, R. Yamamoto, S. Ohzahata, and T. Kato, “Estimating TCP Congestion Control Algorithms from Passively Collected Packet Traces using Recurrent Neural Network,” Proc. ICETE DCNET 2019, pp. 33-42, 2019.

[18] “Alexa Top Sites 1M,” http://s3.amazonaws.com/alexa-static/top-1m.csv.zip. (Accessed on 12/03/2020).

[19] V. Paxson, “Automated Packet Trace Analysis of TCP Implementations,” ACM Comp. Commun. Review, vol. 27, no. 4, pp.167-179, 1997.

[20] T. Kato, T. Ogishi, A. Idoue, and K. Suzuki, “Design of Protocol Monitor Emulating Behaviors of TCP/IP Protocols,” Proc. IWTCS ’97, pp. 416-431, 1997.

[21] S. Jaiswel, G. Iannaccone, C. Diot, J. Kurose, and D. Towsley, “Inferring TCP Connection Characteristics Through Passive Measurements,” Proc. INFOCOM 2004, pp. 1582-1592, 2004.

[22] J. Oshio, S. Ata, and I. Oka, “Identification of Different TCP Versions Based on Cluster Analysis,” Proc. ICCCN 2009, pp. 1-6, 2009.

[23] P. Yang, W. Luo, L. Xu, J. Deogun, and Y. Lu, “TCP Congestion Avoidance Algorithm Identification,” In Proc. ICDCS ’11, pp. 310-321, 2011.

[24] A. Mishra, et al., “The Great Internet TCP Congestion Control Census,” Proc. ACM Meas. Anal. Comput. Syst., vol. 3, no. 3, article 45, pp. 1-24, 2019.

[25] Y. Edalat, J. Ahn, and K. Obraczka, “Smart Experts for Network State Estimation,” IEEE Trans. Network and Service Management, vol. 13, no. 3, pp. 622-635, 2016.

[26] M. Mirza, J. Sommers, P. Barford, and X. Zhu, “A Machine Learning Approach to TCP Throughput Prediction,” IEEE/ATM Trans. Networking, vol. 18, no. 4, pp. 1026-1039, 2010.

[27] J. Chung, D. Han, J. Kim, and C. Kim, “Machine Learning based Path Management for Mobile Devices over MPTCP,” Proc. 2017 IEEE International Conference on Big Data and Smart Computing (BigComp 2017), pp. 206-209, 2017.

[28] S. Hochreiter and J. Schimidhuber, “Long short-term memory,” Neural Computation, vol. 9, no. 8, pp. 1735-1780, 1997.

[29] iPerf3, “iPerf - The ultimate speed test tool for TCP, UDP and SCTP,” https://iperf.fr/.

[30] dpkt, “dpkt,” https://pkt.readthedocs.io/en/latest/.

[31] C. Caini and R. Firrincieli, “TCP Hybla: a TCP enhancement for heterogeneous networks,” Int. J. Satell. Commun. Network., vol. 22, no. 5, pp. 547‑566, 2004.

[32] A. Baiocchi, A.P. Castellani, and F. Vacirca, “YeAH-TCP: Yet Another Highspeed TCP,” Proc. PFLDnet, vol.7, pp. 37‑42, 2007.

[33] S. Liu, T. Basar, and R. Srikant, “TCP-Illinois: A loss-and delay-based congestion control algorithm for high-speed networks,” Performance Evaluation, vol. 65, no. 6-7, pp. 417-440, 2008.

[34] K. Tan, J. Song, Q. Zhang, and M. Sridharen, “A Compound TCP Approach for High-speed and Long Distance Networks,” Proc. IEEE INFOCOM 2006, pp. 1-12, 2006.

[35] Y. You, et al., “Large Batch Optimization for Deep Learning: Training BERT in 76 minutes,” Proc. ICLR 2020, pp. 1-37, 2020.

[36] J. Padhye and S. Floyd, “On Inferring TCP Behavior,” SIGCOMM Comput. Commun. Rev., vol. 31, no. 4, pp. 287-298, 2001.

[37] A. Medina, M. Allman, and S. Floyd, “Measuring the Evolution of Transport Protocols in the Internet,” SIGCOMM Comput. Commun. Rev., vol. 35, no. 2, pp. 37-52, 2005.