End-to-End TCP Congestion Control as a Classification Problem

Guanglu Sun, *Member, IEEE*, Chuan Li, Yu Ma, Shaobo Li, and Jing Qiu

***Abstract*—The traditional rule-based congestion control algorithms cannot set congestion window size flexibly, resulting in the inadaptation of the dynamic networks. This article presents a method to model end-to-end TCP congestion control problem using classification techniques. The network status parameters as the input and the type of network status as the output are defined through the analysis of some existing congestion control algorithms. NewReno, CUBIC, and Compound are used as feedback to produce the training data of the XGBoost classifier. The experimental results show that the classifier effectively shapes the strategies of three outstanding congestion control algorithms and almost achieves the same throughput, delay, and fairness. The proposed method makes the congestion control algorithm able to learn from data produced by network.**

***Index Terms*—Classification algorithms, congestion control, machine learning, network status, XGBoost.**

I. Introduction

NETWORK congestion diminishes quality of service, which occurs when the aggregate requirement for a resource exceeds the obtainable capacity of the resource. Typical effects include the long delays in data delivery, wasted resources due to lost or dropped packets. The goal of end-to-end TCP congestion control is to dynamically adapt the resource usage of a flow to the network's available resources. As the Internet continues to grow rapidly, end-to-end TCP congestion control has become an important research topic [1].

Typically, the design of congestion control algorithms is based on the prior knowledge and experience about the existing network structure. However, in practice, the network environment is continuously changing with network developing. When the congestion control algorithm cannot adapt to the changing environment, new algorithms that can dynamically adapt to the changing environment are needed to ensure the performance of network [2]. For example, the TCP variants [3] and [4] are designed to satisfy the requirements of the data center networks, and [5] and [6] are designed for mobile cellular networks. Moreover, the difference between diverse types of networks becomes more obvious because of the development of novel network technologies and devices. More factors should be considered when designing a new congestion control algorithm, which has increased the difficulty and complexity in designing. Different congestion control algorithms perform differently in the same situation, which results in the fairness problem in network and makes the application of algorithms a difficult task. It can be seen that the future direction of congestion control is intelligent modeling methods [7].

After analyzing three well-known congestion control algorithms such as NewReno [8], CUBIC [9], and Compound [10], we find that the core of the algorithms is the correspondence between network status and the adjustment of congestion window (cwnd). Although the form of the correspondence differs in different algorithms, the correspondence is usually defined by human experts using prior knowledge. So, it can be neither generated automatically nor easily adjusted online.

To address the above problems, this article proposes a method to model network congestion control as a classification problem and develops a machine learning based congestion control algorithm called ML-CC, which makes the automatic generation and adjustment of congestion control algorithm possible. In ML-CC, a feature vector extracted from network status is utilized as the input of the classifier, and the output of the classifier is a label for each network status. Then, the corresponding cwnd size is adjusted according to the label outputted by the classifier.

In order to verify the proposed method in this article, we tested whether ML-CC can trace the behavior of existing congestion control algorithms. A classifier in ML-CC was trained with the data generated by simulation of the existing congestion control algorithms including NewReno, CUBIC, and Compound. The performance of ML-CC was then evaluated by comparing the cwnd size, throughput, delay, and fairness between ML-CC and the three congestion control algorithms. The results show that the performance of ML-CC and the chosen three algorithms is virtually identical. The classifier in ML-CC effectively learns the strategies used by existing congestion control algorithms.

The contributions of this article are as follows:

Manuscript received 10 March 2022; revised 10 April 2022; accepted 18 April 2022. Date of publication 20 May 2022; date of current version 3 March 2023. This work was supported in part by the Fundamental Research Foundation for Universities of Heilongjiang Province under Grant JMRH2018XM04, in part by the Natural Science Foundation of Heilongjiang Province under Grant LH2021F032, in part by the National Natural Science Foundation of China under Grant 60903083. Associate Editor: Yun Lin. *(Corresponding author: Guanglu Sun.)*

Guanglu Sun, Chuan Li, Yu Ma, Shaobo Li, and Jing Qiu are with the School of Computer Science and Technology, Harbin University of Science and Technology, Harbin 150080, China, and also with the Research Center of Information Security and Intelligent Technology, Harbin University of Science and Technology, Harbin 150080, China (e-mail: sunguanglu@hrbust.edu.cn; 1334677114@qq.com; misakaikato@outlook.com; sizhonglsb@163.com; topmint@gmail.com).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TR.2022.3172335.

Digital Object Identifier 10.1109/TR.2022.3172335

1) We model the congestion control problem as a classification problem, and find a data-driven approach for congestion control.
2) We develop a machine learning based congestion control algorithm using XGBoost, which can be trained using data produced by network.
3) We propose a schema to describe the network status with more parameters and to simplify the form of congestion control strategies, which is convenient to be developed and deployed.

The rest of this article is organized as followings. In Section II, we show related work of congestion control problems and the limitations of rule-based TCP variants. We propose the modeling method of ML-CC in Section III. In Section IV-A, we introduce the XGBoost classifier used in ML-CC, and describe the generation of training set. We present experimental results in Section V. Finally Section VI concludes this article.

II. Related Work

Additive-increase multiplicative-decrease (AIMD) based congestion control algorithms, such as TCP Tahoe [11], TCP Reno [12], and TCP NewReno, depend on a series of mechanisms including slow-start, fast retransmit, and congestion avoidance. In slow start, the cwnd size is increased by one each time a TCP acknowledgment (ACK) is received until it reaches the slow start threshold. After sender receives three duplicate ACKs or retransmission time out occurs, sender simply halves the cwnd size. Before receiving a new ACK, the sender sends a new packet when an ACK is received. In congestion avoidance, the sender increases the cwnd size by at most one packet per round-trip time.

Although the AIMD-based congestion control algorithms are successfully applied in low bandwidth-delay product (BDP) network, they are noneffective in discovering available bandwidth and recovering from packet loss in high BDP networks. Because the BDP of high-speed networks is very large, the cwnd size should be maintained at a high range to fully utilize the network bandwidth.

To address the problem above, many congestion control algorithms have been presented for high-speed network. Floyd [13] proposed Highspeed TCP (HSTCP) in 2003, which improves the additive factor and multiplicative factor of AIMD process to accomplish relatively high increasing rate and low reduction rate of congestion window. It thus maintains a larger congestion window. In 2004, Binary Increase Congestion Control (BIC) [14] was proposed to find the optimal cwnd size by binary search algorithm, which is fundamentally different from AIMD-based algorithm. In 2008, to improve BIC behavior, Ha *et al.* proposed CUBIC to simplify the binary search procedure and the max probe procedure of BIC by replacing the window growth process of BIC with a cubic function. The CUBIC function only depends on the time since the last congestion event, so it can achieve better RTT fairness and stability. Furthermore, Compound proposed by Kun *et al.* applies packet loss and delay as inputs to the cwnd control mechanism. Because of the high efficiency, Compound and CUBIC have been widely used in Windows and Linux, respectively.

In order to generate congestion control algorithms for different network environments automatically, Winstein *et al.* proposed Remy [15] in 2013. Remy takes the prior assumptions about the target network and a predefined objective function as input, and returns a congestion control algorithm called RemyCC as output. The main idea of Remy is to find the best congestion control strategies for the target network by adjusting and evaluating the strategies iteratively. In each iteration, the adjustment of the strategies is according to empirical parameters, the evaluation of the strategies is performed through the evaluation of the simulation result using the objective function.

However, congestion control strategies in RemyCC is induced in the form of a rule set, which exposes a lack of ability of generalization and update for new network environment. Moreover, the performance of RemyCC is mainly determined by the quality of the empirical parameters, which is hard to be guaranteed in practice.

For better performance and adaptability, the idea of self-adaption is integrated into congestion control. In 2015, Mo *et al.* proposed performance-oriented congestion control (PCC). PCC [16] controls the sending rate of packets directly, and tries to find the optimal sending rate by testing different sending rates in different time intervals. The performance of different sending rates is measured by the utility function, which is calculated according to the ACKs received during the current time interval. In 2016, Cardwell *et al.* proposed TCP BBR [17], whose main idea is maintaining the cwnd size at the optimal operating point to achieve the maximum bandwidth without increasing the queue size. The optimal operating point is calculated according to the maximum bandwidth and minimum delay which is measured independently. Furthermore, in 2018, Li *et al.* proposed QTCP [18], which integrates Q-learning framework and TCP, so that the sender can gradually learn the optimal congestion control strategy online without hard coding.

PCC, BBR, and QTCP learn and adapt to the network environment for optimal performance. It can be seen that the development direction of congestion control is moving toward intelligence and automation. This article argues that, an intelligent congestion control algorithm should meet the following requirements: 1) Can be automatically generated according to the prior knowledge (like Remy) and 2) can be adjusted online in response to changes in the network environment (like PCC, BBR, and QTCP). Therefore, we propose a method to model network congestion control problem using classification technique, then provide a data-driven approach for congestion control.

III. Modeling the Congestion Control Problem

A. Analysis of End-to-End TCP Congestion Control Algorithms

In order to model the network congestion control problem, we first analyze the process of end-to-end TCP congestion control. Without adding extra traffic to the network, the network congestion can only be inferred from the ACKs received. Thus,

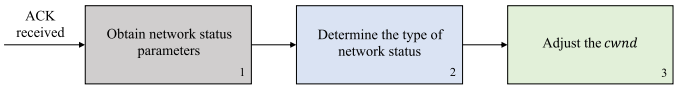


Fig. 1. Congestion control process.

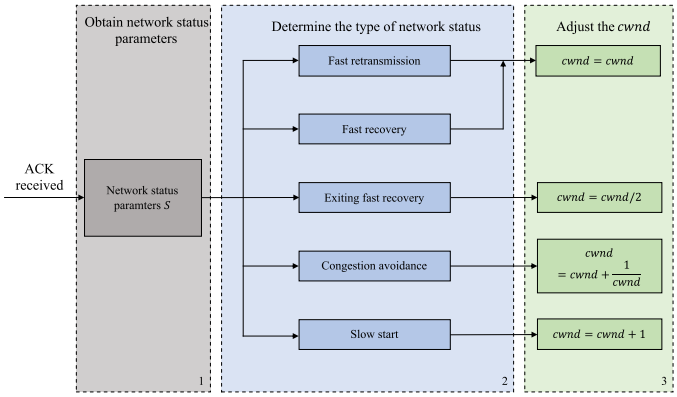


Fig. 2. Modular structure of NewReno.

TABLE I
NETWORK STATUS PARAMETERS USED IN NEWRENO

Parameter	Definition	Calculation
$dupacks$	The number of duplicate ACKs received	Increase when duplicate ACK is received, set to 0 otherwise.
$cwnd$	The current congestion window size	Update when the current congestion window size changes
$ssthresh$	Slow start threshold	Set to the half of the cwnd when congestion occurs
$recover$	Fast recovery parameter	Set to *max_send_seq* when congestion occurs
max_ack_seq	The highest acknowledged packet sequence number	Update according to the sequence number of ACK received
max_send_seq	The highest outstanding unacknowledged packet sequence number	Update according to the sequence number of packet sent

the process of congestion control is described as follows. First, after an ACK is received from the receiver, the sender obtains the parameters that represent the network status based on the ACK, then uses these parameters to determine how to adjust the cwnd. The process is shown in Fig. 1.

In Fig. 1, the function of the first module is obtaining the network status parameters. The second module is determining the type of network status according to the built correspondence between network status parameters and the types of network status. The third module is adjusting the cwnd according to the type of network status that results from the second module.

We analyze NewReno, CUBIC, Compound, and RemyCC to characterize them based on the modules in Fig. 1. The modular structure of NewReno is shown in Fig. 2.

In Fig. 2, the network status parameters are represented by S. The definition of these parameters and how they are calculated

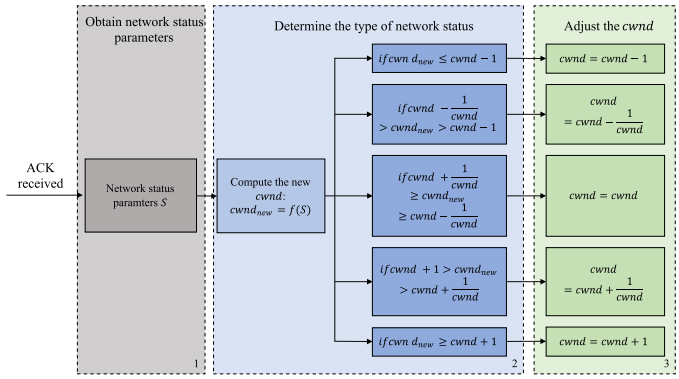


Fig. 3. Modular structure of algorithm using the cwnd function.

are shown in Table I. NewReno uses these parameters to differentiate the type of network status on the basis of its rule set. The rule set is as follows. When the cwnd size is less than $ssthresh$ or congestion never occurs, the sender is in slow start and increases the cwnd size by 1 MSS for every received ACK. When $dupacks$ is equal to 3, the sender enters fast retransmit/fast recovery. It sets the variable $recover$ to the value of max_send_seq and does not change the cwnd size, until max_ack_seq is larger than $recover$. Then, the sender halves the cwnd size. When the cwnd size is greater than $ssthresh$ and $dupacks$ is less than 3, the sender is in congestion avoidance and the cwnd size is increased by 1 MSS for each RTT.

Through the above analysis, we can treat NewReno as an algorithm that evaluates the network status parameters according to the predefined rule set to determine the type of network status, and to adjust the cwnd accordingly.

Unlike NewReno, which uses the rule set, CUBIC uses a cwnd function to compute the cwnd size directly. It also can be characterized as the modules in Fig. 1. The cwnd computation function $f(S)$ takes network status parameters S as input and produces a new cwnd size as an output. The S includes the cwnd size when the last packet loss is detected and the interval since the last packet loss. The modular structure is shown in Fig. 3.

In Fig. 3, new_cwnd is the output of the cwnd function and $cwnd$ is the value of current cwnd size. By dividing the difference between new_cwnd and the $cwnd$, we can simply classify the network status into five types.

The synergy of Compound is implemented by adding a new scalable delay-based component in the congestion avoidance mechanism of Reno. Compound updates the cwnd size by adding a delay-based cwnd ($dwnd$) to the original cwnd. The delay-based component uses one of three different functions according to the network status to compute $dwnd$. So, Compound can also be considered as a cwnd function, whose input includes the network status parameters used by Reno, the parameter derived from Vegas *diff*, the current $dwnd$, and the current cwnd size. It also can be characterized using the same way as CUBIC.

RemyCC also uses rule sets to represent its congestion control strategies, but its difference with NewReno is that the rule sets are generated automatically. The network status parameters used in RemyCC include an exponentially weighted moving average (EWMA) of the interarrival time between new ACKs

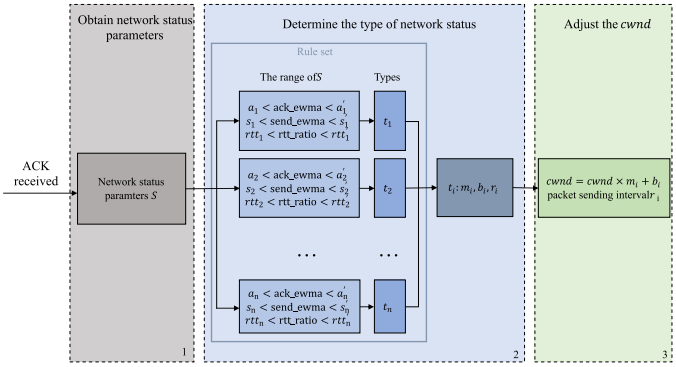


Fig. 4. Modular structure of RemyCC.

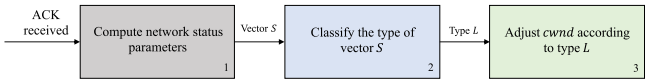


Fig. 5. Architecture of ML-CC.

received (ack_ewma), an EWMA of the time between TCP sender timestamps reflected in those ACKs ($send\_ewma$) and the ratio between the most recent RTT and the minimum RTT seen during the current connection (rtt_ratio). RemyCC adjusts the sending rate by multiplying the cwnd size by m ($m >= 0$), increasing the cwnd size by b (b could be negative) and setting a lower bound $r > 0$ (milliseconds) on the time between successive sends. Identically, RemyCC can also be characterized as shown in Fig. 4.

When an ACK is received, RemyCC calculates the network status parameters S={ack_ewma, $send\_ewma$, rtt_ratio}, and determines the type of network status t_i according to the generated rule set. Then, RemyCC uses the parameters m_i, b_i, r_i corresponding to t_i to adjust the cwnd.

B. Modeling the End-to-End TCP Congestion Control Using a Classifier

The modeling method proposed in this article is not confined to design of the rule-based congestion control strategies. Data-driven approach is introduced to generate congestion control algorithms automatically, which changes the problem of congestion control into the classification problem. The analysis of existing congestion control algorithms suggests that the significant part of the algorithm is classifying the type of network status in order to find the optimal adjustment of the cwnd. And the existing knowledge used by network status classification is in the form of rule sets or functions. Similarly, a classifier can be used to classify the network status. The input is a vector S composed by network status parameters and the output is the label of S, namely the type of network status. Then, the sender adjusts the cwnd according to the labels outputted by the classifier.

According to this idea, we developed a congestion control algorithm ML-CC, which treats the congestion control problem as a classification problem. The architecture of ML-CC is shown in Fig. 5.

TABLE II
NETWORK STATUS PARAMETERS USED IN ML-CC

No	Parameter	Definition
1	ack_ewma	The same as the parameters used in RemyCC
2	$send\_ewma$	
3	rtt_ratio	
4	ack	The interarrival time between new ACKs received
5	$send$	The time between TCP sender timestamps reflected in new ACKs received
6	rtt	Current RTT
7	*rtt_diff*	The difference between the current RTT and the previous RTT
8	$cubic\_t$	The interval since the last packet loss
9	$last\_lost\_cwnd$	The cwnd size when the last packet loss is detected
10	$dupacks$	The number of duplicate ACKs received
11	ack_aver	The average of ack
12	ack_ratio	The ratio between ack and the minimum ack
13	tsa_tla	The ratio between the average of last 20 ack and the average of last 100 ack
14	ack_stable	Set ack_stable to ack when $tsa\_tla < 0.75$
15	$cwnd$	The current cwnd size
16	*diff*	The *diff* used in Vegas
17	ack_counts	The number of ACKs received in current RTT

TABLE III
LABEL OUTPUTTED BY THE CLASSIFIER

Label	Adjustment
l_1	$cwnd = cwnd + 1/cwnd$
l_2	$cwnd = cwnd + 1$
l_3	$cwnd = cwnd$
l_4	$cwnd = cwnd - 1$
l_5	$cwnd = cwnd - 1/cwnd$

The quality of network status parameters will influence the performance of the classifier. To better quantify the network status, we choose many network status parameters from existing congestion control algorithms to compose the vector S, which is taken as the input of the classifier. The network status parameters used are shown in Table II.

In Table II, the parameters ack_ewma, $send\_ewma$ and rtt_ratio are from RemyCC, $cubic\_t$ and $last\_lost\_cwnd$ are from CUBIC, *diff* is from Vegas. The parameters rtt and *rtt_diff* are used to estimate the buffer usage. The parameters ack_aver, ack_ratio, ack_stable, ack_counts, and tsa_tla are related to the rate of the ACKs received by the sender, which represents the throughput of the sender.

The output of the classifier is one of the labels shown in Table III. Each label represents a type of network status and corresponds to a type of cwnd adjustment.

Similar to AIMD-based algorithms, the type l_1 is used to increase the cwnd size linearly, l_2 is used to increase the cwnd size exponentially and l_4 is used to decrease the cwnd size multiplicatively. The type l_5 is chosen from Vegas to decrease the cwnd linearly and l_3 is used to keep a constant cwnd size.

When an ACK is received, the sender extracts the network status parameters shown in Table II from received ACK. These parameters together form a 17-dimension vector, which is taken as the input of the classifier H. The output of H is the label L of S. Each label corresponds to a specific type of cwnd adjustment. In this way, the congestion control algorithm is made able to

learn from data with a classifier. Next, we will introduce the classifier used in this article, and the process of generating the data for training a classifier.

IV. Training of the Classifier

A. The Classifier: XGBoost

In machine learning, classification technique is used to build a classifier for identifying to which of a set of categories a new instance belongs, on the basis of a set of training data containing instances whose category is known [19]. The classification algorithms have been applied in many fields, such as text identification [20], image recognition [21], traffic classification [22], etc. Because this article does not focus on the selection or optimization of classification algorithms, XGBoost [23] is chosen as our classification algorithm because of its regularization and resampling to resolve the problem of robustness and data bias. XGBoost is one of Gradient boosting tree algorithms for regression and classification problems, which produces a prediction model in the form of an ensemble of weak prediction models.

In practice, a given dataset with n samples D can be represented as $D=\{< x_1, y_1 >, < x_2, y_2 >, \ldots, < x_n, y_n >\}$ where x_i is an instance (d-dimensional feature vector) characterizing sample i and y_i is the corresponding known category label. XGBoost uses K additive functions to predict the output $\hat{y}_i$ shown as follows:

$$\hat{y}_i = \phi(\mathbf{x}) = \sum_{k=1}^{K} f_k(\mathbf{x_i}), f_k \in \mathcal{F} \tag{1}$$

where $\mathcal{F}$ is the space of regression trees (also known as CART [24]), and f_k denotes a single tree. Each f_k corresponds to an independent tree structure and leaf weights. For a given sample, we will use the decision rules in the trees to classify it into the leaves and calculate the final prediction by summing up the weights in the corresponding leaves.

The objective function of XGBoost includes loss function and regularization. The objective function is defined shown as follows:

$$\mathcal{L}(\phi) = \sum_i l(\hat{y}_i, y_i) + \sum_k \Omega(f_k) \tag{2}$$

where l is a differentiable convex loss function that measures the difference between the prediction $\hat{y}_i$ and the target y_i. $\sum_k \Omega(f_k)$ is the regularization term, which calculates the model complexity. It helps to smooth the final learnt weights to avoid overfitting. The complexity of a single tree is calculated as shown as follows:

$$\Omega(f) = \gamma T + \frac{1}{2}\lambda \|w\|^2 \tag{3}$$

where T is the number of leaves in the tree, $\|w\|^2$ denotes the square of the L2-norm of the weights in the tree, γ and λ are the coefficients.

B. Training Set Generation

The classifier can learn rules and behaviors based on positive or negative feedback of congestion from an actual network.

Algorithm 1: Generation of the Training Set.

input : Network structure, Bottleneck bandwidth and delay, Chosen existing congestion control algorithm
output: Training set D
Simulate the network according to the input;
while *the simulation is not over* **do**
  if *ACK is received by senders* **then**
    $old\_cwnd \leftarrow$ current cwnd size;
    $S \leftarrow$ the network status parameters vector;
    $new\_cwnd \leftarrow$ cwnd size computed by the chosen existing congestion control algorithm;
    if $new\_cwnd > old\_cwnd + 1$ **then**
      $lable = l_2$;
    else if $new\_cwnd > old\_cwnd + \frac{1}{old\_cwnd}$ **then**
      $lable = l_1$;
    else if $new\_cwnd < old\_cwnd - 1$ **then**
      $lable = l_4$;
    else if $new\_cwnd < old\_cwnd - \frac{1}{old\_cwnd}$ **then**
      $lable = l_5$;
    else
      $lable = l_3$;
    Set current cwnd size to new_cwnd;
    Append the sample composed by the vector S and $label$ to training set D;

If there is no practical feedback, existing congestion control algorithms are utilized as feedback to produce the training data of the classifier in ML-CC, so that ML-CC can be verified to approximate the behavior and performance of the existing algorithms. First, an existing congestion control algorithm is implemented at the sender. When an ACK is received, the sender extracts the parameters listed in Table II and maps the cwnd adjustment to the types listed in Table III. The process of training set generation is shown in Algorithm 1.

As the output of the algorithm, the training set D is taken as the input of XGBoost algorithm to train the classifier H. The default parameters are used when training with XGBoost algorithm. In ML-CC, when an ACK is received, the sender extracts the same network status parameters S listed in Table II and uses the classifier H to predict the label L of S, then adjusts cwnd according to L.

V. Experiments

A. Experiment Setup and Metrics

We use the ns-2 [25] network simulator for all the experiments. The setup of the experiment is as follows: the network topology is shown in Fig. 6, including a dumbbell shaped topology with two routers and four pairs of sender and receiver. The bottleneck bandwidth varies from 10 to 1000 Mbps and its delay varies from 10 to 100 ms.

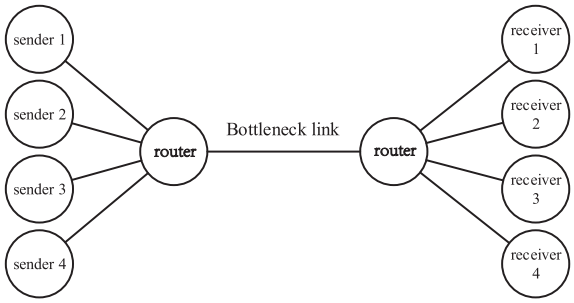


Fig. 6. Network topology used in experiments.

The effectiveness of the proposed method is verified by determining if well-behaved congestion control strategies in the network can be effectively learned by ML-CC. Because there is no gold standard data for training and testing, we choose NewReno, CUBIC, and Compound to generate the data in the experiment network (the generated data will be used for training and fivefold cross-validation). The performance and behavior between ML-CC and three algorithms are compared to see if ML-CC performed as required.

The experimental results include three parts: the preliminary verification of proposed method, the evaluation of classifier performance and the comparison of the performance between ML-CC and existing algorithms. We use the following metrics to evaluate the performance of the classifier [26]:

1) Accuracy: Ratio of correctly classified instances.
2) F1-score: The harmonic average of the precision (the number of correctly classified positive examples divided by the number of examples labeled by the system as positive) and recall (the number of correctly classified positive examples divided by the number of positive examples in the data), where an F1-score reaches its best value at 1 (perfect precision and recall) and worst at 0.

We also use the cwnd behaviors, throughput, delay, and fairness as metrics to compare the congestion control performance [27]. The fairness is measured by Jain's fairness index [28].

B. Preliminary Verification of the Proposed Method

The experiment is designed to validate the effectiveness of ML-CC by comparing the cwnd change of ML-CC and NewReno. We simulate ML-CC and NewReno in various network environment, respectively. As shown in the network topology in Fig. 6, there are four pairs of sender and receiver in each simulation experiment. As shown in Fig. 7, the simulation results of the same network environment are drawn together. The cwnd change of four ML-CC senders are plotted in solid lines and the cwnd change of four NewReno senders are plotted in dotted lines.

In most results, ML-CC and NewReno can get almost the same results, that is, the cwnd change of ML-CC and NewReno are identical. In some cases, the cwnd change of ML-CC and NewReno are different, but the changing trends are consistent.

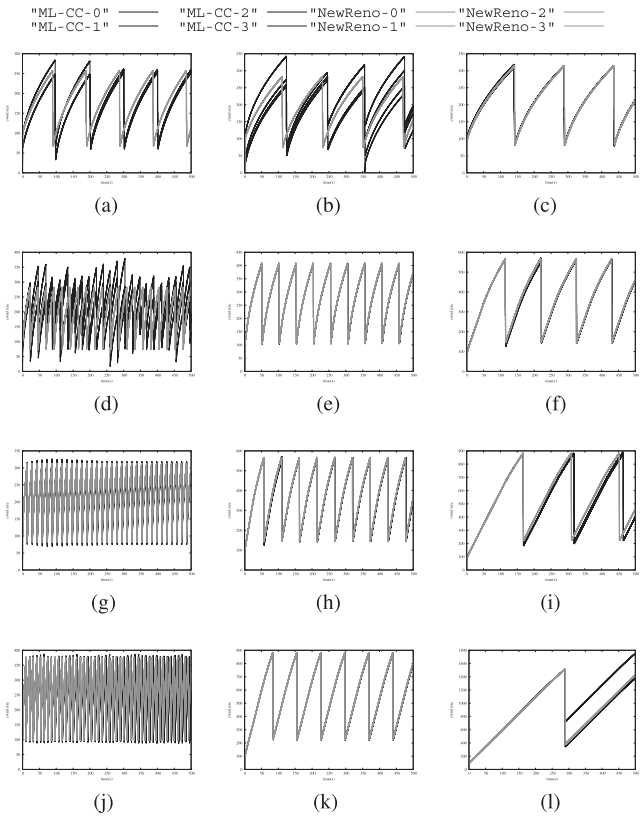


Fig. 7. cwnd change of ML-CC and NewReno in different bandwidth and delay. (a) 10Mb-10ms. (a) 10Mb-10ms. (c) 10Mb-100ms. (d) 50Mb-10ms. (e) 50Mb-50ms. (f) 50Mb-100ms. (g) 100Mb-10ms. (h) 100Mb-50ms. (i) 100Mb-100ms. (j) 200Mb-10ms. (k) 200Mb-50ms. (l) 200Mb-100ms.

For example, there are some mismatches on the *X*-axis in 10 Mb–10 ms; the ranges of cwnd change are not exactly same in 10 Mb–50 ms; the cwnd change of ML-CC is more dramatically in 50 Mb–10 ms. From the results, ML-CC can effectively represent the congestion control mechanisms in NewReno, like slow start and congestion avoidance. The experimental results show the modeling method proposed in this article is practicable.

C. Performance of Classifier

We got 36 sets of experimental results to illustrate the performance of classifier. Table IV shows the performance of classifier obtained from training set generated by Compound algorithm. Table V shows the performance of classifier obtained from training set generated by CUBIC algorithm. The last column of each table represents the size of the training set used in the simulation experiments. All the results are based on fivefold cross validation.

As can be seen from Tables IV and V, most F1-scores are higher than 0.9, even 0.99. When the bottleneck delay is 10 ms, the bandwidth is 500 or 1000 Mb, the F1-score of l_4 is unsatisfactory. This is because the l_4 samples only account for 0.01 to 0.3% of the training set, which has little impact on the performance of classifier.

As can be seen from the Figs. 8 and 9, most accuracy of classifier are greater than 90% in CUBIC experiments, even

TABLE IV
F1-SCORE OF CLASSIFIER OBTAINED FROM TRAINING DATA GENERATED BY COMPOUND ALGORITHM

Bottleneck Bandwidth-Delay	l_1	l_2	l_3	l_4	l_5	Sample size
10Mb-10ms	1.00	0.98	1.00	0.99	1.00	156411
10Mb-50ms	0.99	0.95	1.00	0.99	1.00	156257
10Mb-100ms	0.99	0.89	1.00	0.99	1.00	155927
50Mb-10ms	1.00	0.97	1.00	1.00	1.00	781122
50Mb-50ms	1.00	0.99	1.00	1.00	1.00	778840
50Mb-100ms	0.99	0.87	1.00	0.16	0.99	756282
100Mb-10ms	1.00	0.99	1.00	0.99	1.00	1562241
100Mb-50ms	0.99	0.89	1.00	0.25	0.99	1537104
100Mb-100ms	1.00	0.52	1.00	0.09	0.99	1391321
200Mb-10ms	1.00	0.94	1.00	0.99	1.00	3123988
200Mb-50ms	1.00	0.89	1.00	0.12	0.99	2919179
200Mb-100ms	1.00	0.94	1.00	0.02	1.00	2348992
500Mb-10ms	0.99	0.98	1.00	0.33	1.00	7779865
500Mb-50ms	0.98	0.93	1.00	0.02	0.96	6358985
500Mb-100ms	0.98	0.93	1.00	0.11	0.97	10671251
1000Mb-10ms	0.97	0.91	1.00	0.02	0.96	15656718
1000Mb-50ms	1.00	1.00	1.00	0.21	1.00	15349821
1000Mb-100ms	0.98	0.95	0.99	0.32	0.96	13656498

TABLE V
F1-SCORE OF CLASSIFIER OBTAINED FROM TRAINING DATA GENERATED BY CUBIC ALGORITHM

Bottleneck Bandwidth-Delay	l_1	l_2	l_3	l_4	l_5	Sample size
10Mb-10ms	0.85	0.94	0.98	0.87	0.96	149837
10Mb-50ms	0.98	0.96	0.99	0.89	0.99	162645
10Mb-100ms	0.82	0.94	0.98	0.86	0.97	158635
50Mb-10ms	0.91	0.93	0.96	0.87	0.97	631163
50Mb-50ms	0.99	0.95	0.98	0.90	0.99	893278
50Mb-100ms	0.90	0.94	0.99	0.81	0.98	873507
100Mb-10ms	1.00	0.94	0.95	0.94	1.00	1972731
100Mb-50ms	0.99	0.94	0.98	0.92	0.99	1523099
100Mb-100ms	0.95	0.94	0.99	0.89	0.97	1300788
200Mb-10ms	0.98	0.95	0.92	0.94	1.00	3124146
200Mb-50ms	0.98	0.93	0.97	0.94	0.99	2749755
200Mb-100ms	1.00	0.95	0.99	0.92	1.00	2208996
500Mb-10ms	0.99	0.98	0.90	0.93	0.99	8330265
500Mb-50ms	1.00	0.92	0.98	0.92	1.00	6669525
500Mb-100ms	0.91	0.94	0.99	0.89	0.99	10998059
1000Mb-10ms	0.77	0.88	0.84	0.72	0.91	15164682
1000Mb-50ms	0.91	0.87	0.97	0.88	0.90	15150884
1000Mb-100ms	0.96	0.92	1.00	0.82	0.93	12299580

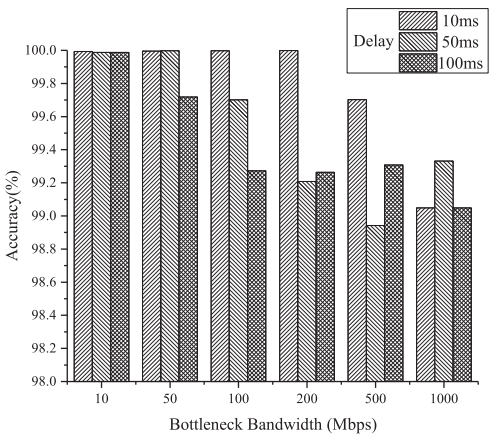


Fig. 8. Accuracy of the classifier obtained from training data generated by Compound in different bandwidth and delay.

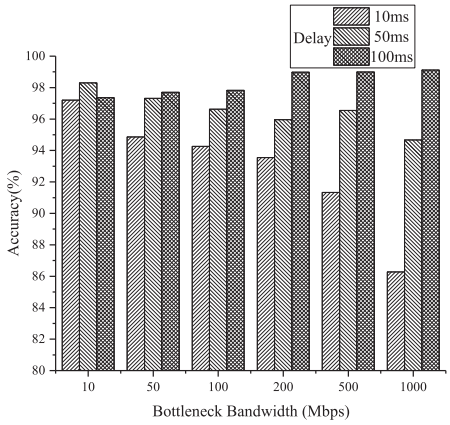


Fig. 9. Accuracy of the classifier obtained from training data generated by CUBIC in different bandwidth and delay.

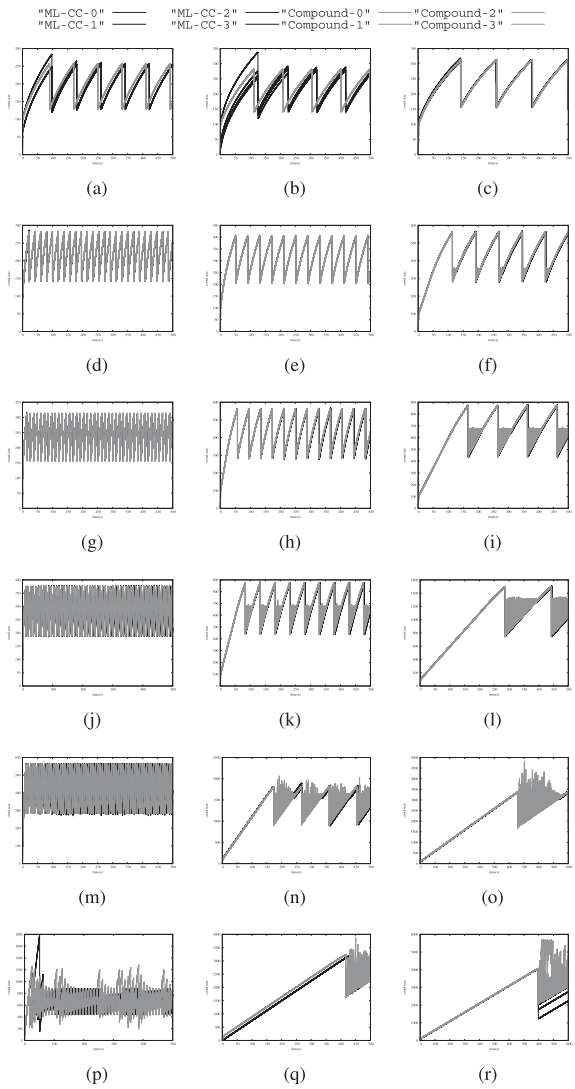


Fig. 10. cwnd change of ML-CC and Compound in different bandwidth and delay. (a) 10Mb-10ms. (b) 10Mb-50ms. (c) 10Mb-100ms. (d) 50Mb-10ms. (e) 50Mb-50ms. (f) 50Mb-100ms. (g) 100Mb-10ms. (h) 100Mb-50ms. (i) 100Mb-100ms. (j) 200Mb-10ms. (k) 200Mb-50ms. (l) 200Mb-100ms. (m) 500Mb-10ms. (n) 500Mb-50ms. (o) 500Mb-100ms. (p) 1000Mb-10ms. (q) 1000Mb-50ms. (r) 1000Mb-100ms.

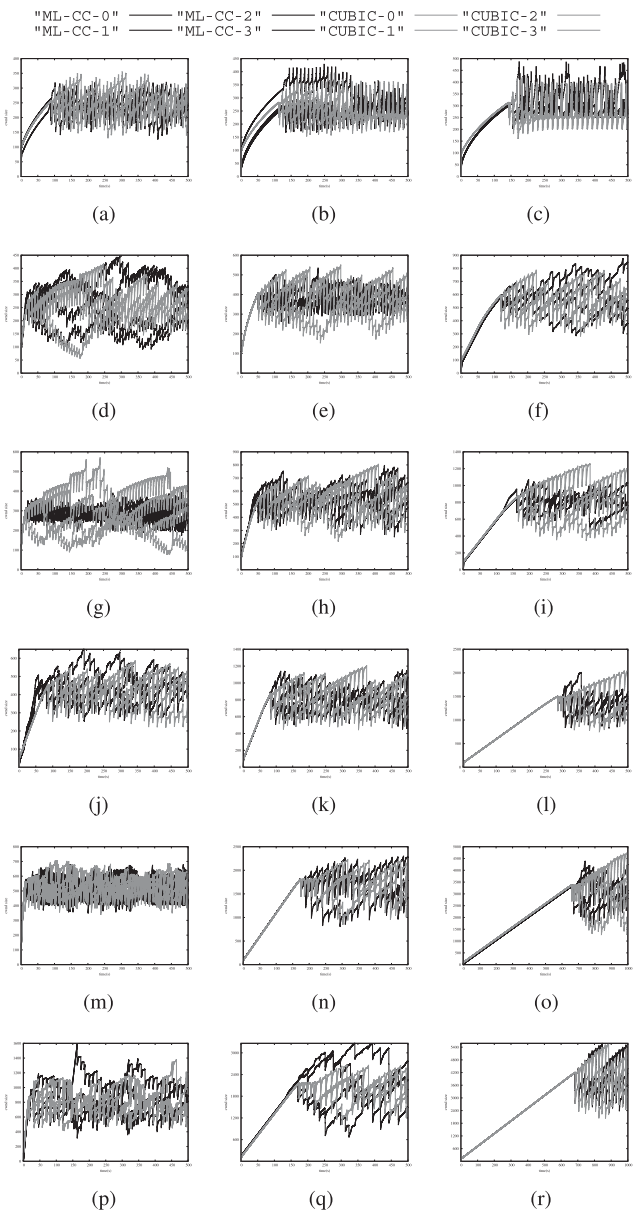


Fig. 11. cwnd change of ML-CC and CUBIC in different bandwidth and delay. (a) 10Mb-10ms. (b) 10Mb-50ms. (c) 10Mb-100ms. (d) 50Mb-10ms. (e) 50Mb-50ms. (f) 50Mb-100ms. (g) 100Mb-10ms. (h) 100Mb-50ms. (i) 100Mb-100ms. (j) 200Mb-10ms. (k) 200Mb-50ms. (l) 200Mb-100ms. (m) 500Mb-10ms. (n) 500Mb-50ms. (o) 500Mb-100ms. (p) 1000Mb-10ms. (q) 1000Mb-50ms. (r) 1000Mb-100ms.

99% in Compound experiments. It shows the features obtained from network status are able to distinguish five types of network status. XGBoost performs well on the training set generally.

D. Comparison of Cwnd Behavior Between ML-CC and Compound/CUBIC

The purpose of the experiment design is to compare the behavior of the cwnd over time between ML-CC and Compound/CUBIC. The experiment is divided into two groups: group 1 in which we compare ML-CC and Compound; group 2 in which we compare ML-CC and CUBIC. We simulate ML-CC and Compound/CUBIC in various network environment, respectively. As shown in Fig. 10, the cwnd change of four ML-CC senders are plotted in solid lines and the cwnd change of four Compound senders are plotted in dotted lines. As shown in Fig. 11, the cwnd change of four ML-CC senders are plotted in solid lines and the cwnd change of four CUBIC senders are plotted in dotted lines.

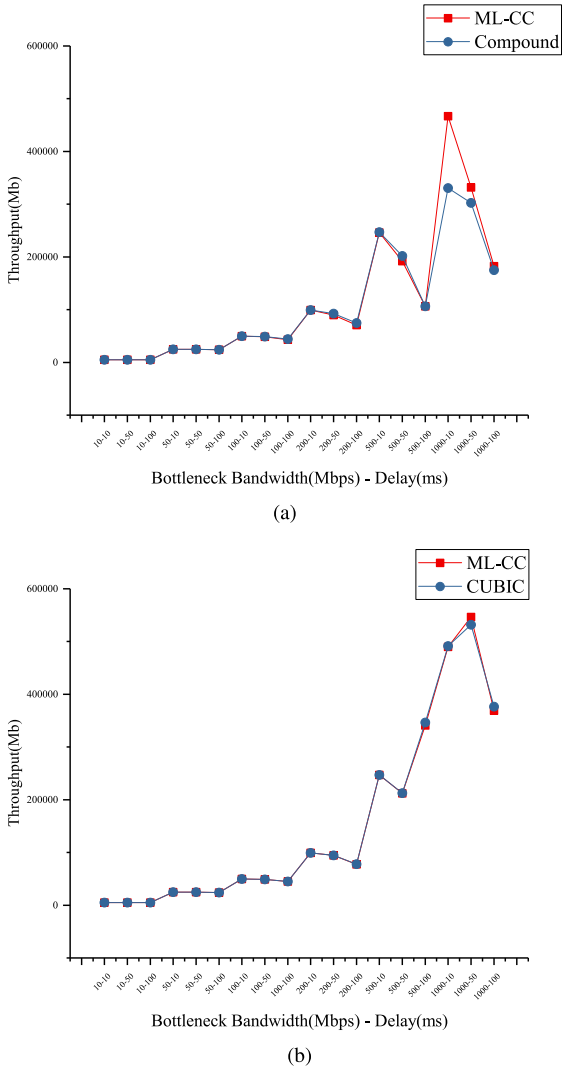
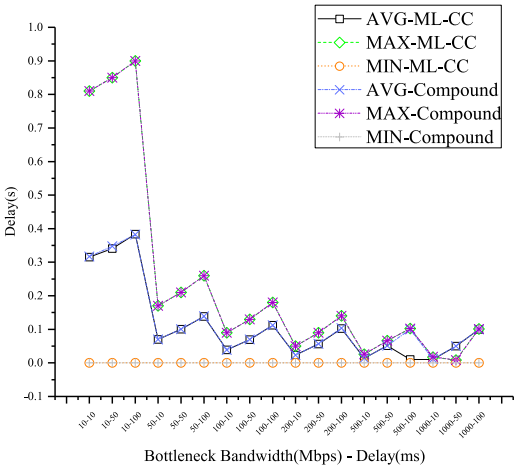


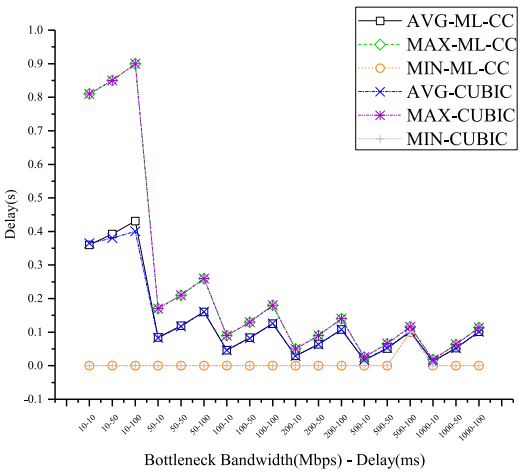
Fig. 12. Comparison of throughput. (a) ML-CC VS Compound. (b) ML-CC VS CUBIC.

From the comparison between ML-CC and Compound, the lines are almost overlapping with few deviations. It means that the change of cwnd are almost identical. When bottleneck delays are above 50 ms, the cwnd change of Compound have some jitter in most case, but that does not occur in ML-CC.

From the comparison between ML-CC and CUBIC, ML-CC can adjust the cwnd in the shape of a cubic function. The fairness of ML-CC is not good because CUBIC lacks the ability of fairness [29]. For example, the change of cwnd between four senders are quite different in 10 Mb–50 ms and 50 Mb–10 ms.

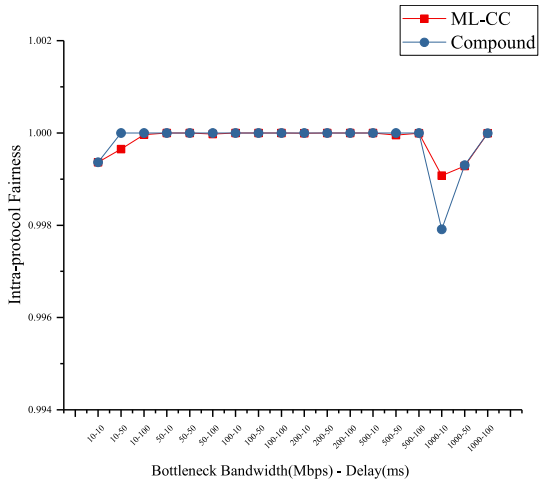


(a)

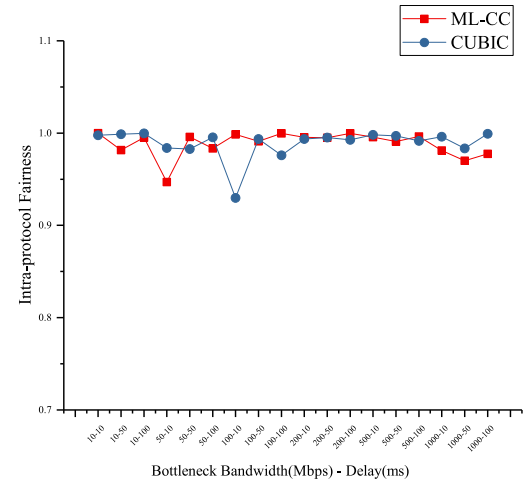


(b)

Fig. 13. Comparison of delay. (a) ML-CC VS Compound. (b) ML-CC VS CUBIC.



(a)



(b)

Fig. 14. Comparison of intraprotocol fairness. (a) Comparison of intraprotocol fairness. (b) ML-CC VS CUBIC.

To sum up, ML-CC traces the behaviors of Compound/CUBIC unambiguously, so the classifier can learn the characteristic of the Compound/CUBIC very well.

E. Performance Comparison

The purpose of the experiment design is to compare the performance of ML-CC and Compound/CUBIC. In the experiment, we compare the throughput, delay, intraprotocol fairness, and TCP fairness between ML-CC and Compound/CUBIC. The simulation is the same as Section 5.4, except of the simulation for the TCP fairness.

We compare the TCP fairness between ML-CC and Compound/CUBIC by measuring the TCP fairness of two groups of simulation: In group 1, ML-CC is deployed on the first two senders and NewReno is deployed on the other senders; in group 2 Compound/CUBIC is deployed the first two senders and NewReno on the other senders.

Fig. 12 shows the throughput comparison between ML-CC and Compound/CUBIC. Fig. 13 shows the delay comparison between ML-CC and Compound/CUBIC. The AVG represents the average value of delay, the MAX represents the max value of delay, the MIN represents the min value of delay. Fig. 14 shows the intraprotocol fairness comparison between ML-CC and Compound/CUBIC. Fig. 15 shows the TCP fairness comparison between ML-CC and Compound/CUBIC.

In Figs. 12–15, the horizontal axis represents the bandwidth and delay, the vertical axis represents the metric. As can be seen from the experimental results, the classification-based algorithm ML-CC and Compound/CUBIC can achieve the same performance in throughput, delay, intraprotocol fairness, and TCP fairness. So, the congestion control strategies can be defined as a classifier instead of rule set and cwnd function.

There is a large range of curve shaken in ML-CC, because the five types of actions are more refined than the adjustment of the traditional algorithm. When the traditional algorithm encounters a half or a multiple for the congestion window, it usually takes multiple factors to multiply the original congestion window to get the actual congestion window.

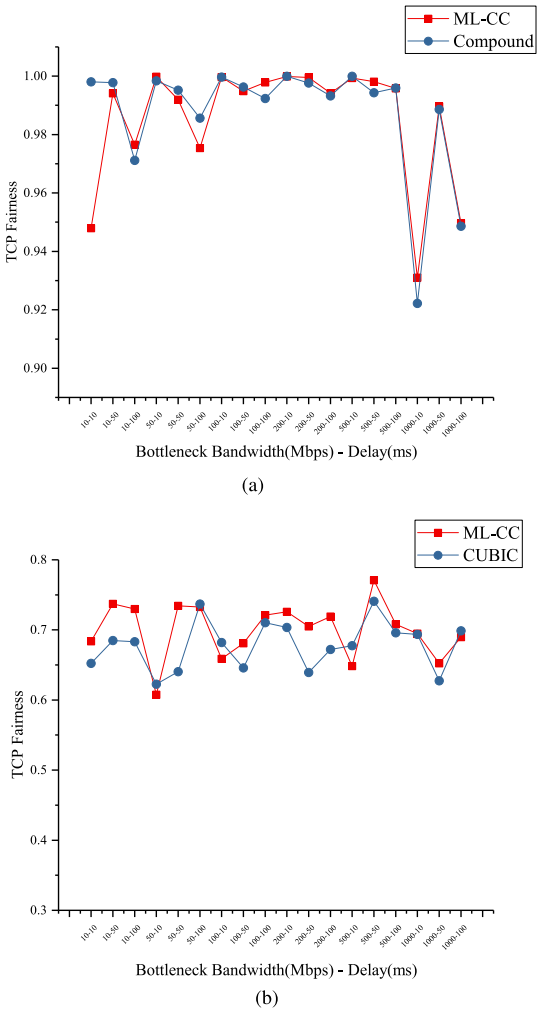


Fig. 15. Comparison of TCP fairness. (a) Comparison of intraprotocol fairness. (b) ML-CC VS CUBIC.

However, in ML-CC, this case is composed of multiple $cwnd = cwnd - 1$ or $cwnd = cwnd + 1$. Under our finer granularity condition, the curve of the obtained congestion window is more sensitive to the network congestion control environment, that is, it produces shaken. Compared with the final result and the general upward trend, ML-CC algorithm and traditional congestion control algorithm are almost the same. These shaken do not affect the overall efficiency of ML-CC algorithm.

VI. Conclusion

In this article, through the analysis of some existing congestion control algorithms, we propose a method to model network congestion control problem as a classification problem and develop ML-CC, which is trained by a data-driven approach. In ML-CC, the classifier is utilized to represent the congestion control strategies used to the adjustment of the congestion window. In the experiments, NewReno, CUBIC, or Compound is used to generate the network data for training and testing. With the generated data, ML-CC achieves a comparable throughput, delay, and fairness in a learning module. It demonstrates that the classifier can be used to learn rules and behaviors based on the feedback of congestion from network.

In future work, we will pay more attention to adjust the congestion control model based on the online feedback and propose an online learning module.

References

[1] S. H. Low, F. Paganini, and J. C. Doyle, "Internet congestion control," *IEEE Control Syst.*, vol. 22, no. 1, pp. 28–43, Feb. 2002.
[2] U. Ahmad, M. A. B. Ngadi, and I. F. B. Isnin, "Survey on end to end congestion control techniques in different network scenarios," *J. Theor. Appl. Inf. Technol.*, vol. 77, no. 3, pp. 1–18, 2015.
[3] M. Alizadeh *et al.*, "Data center TCP (DCTCP)," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 41, no. 4, pp. 63–74, 2011.
[4] H. Wu, Z. Feng, C. Guo, and Y. Zhang, "ICTCP: Incast congestion control for TCP in data-center networks," *IEEE/ACM Trans. Netw. (ToN)*, vol. 21, no. 2, pp. 345–358, Apr. 2013.
[5] K. Brown and S. Singh, "M-TCP: TCP for mobile cellular networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 27, no. 5, pp. 19–43, 1997.
[6] Y. Zaki, T. Pötsch, J. Chen, L. Subramanian, and C. Görg, "Adaptive congestion control for unpredictable cellular networks," in *Proc. ACM Conf. Special Int. Group Data Commun.*, vol. 45, no. 4, 2015, pp. 509–522.
[7] M. Schapira and K. Winstein, "Congestion-control throwdown," in *Proc. 16th ACM Workshop Hot Topics Netw.*, 2017, pp. 122–128.
[8] S. Floyd, T. Henderson, and A. Gurtov, "The newreno modification to TCP's fast recovery algorithm," Tech. Rep., 2004.
[9] S. Ha, I. Rhee, and L. Xu, "Cubic: A new tcp-friendly high-speed tcp variant," *ACM SIGOPS Operating Syst. Rev.*, vol. 42, no. 5, pp. 64–74, 2008.
[10] K. Tan, J. Song, Q. Zhang, and M. Sridharan, "A compound TCP approach for high-speed and long distance networks," in *Proc. IEEE INFOCOM Conf. Comput. Commun.*, 2006, pp. 1–12.
[11] V. Jacobson, "Congestion avoidance and control," in *ACM SIGCOMM Comput. Commun. Rev.*, vol. 18, no. 4, 1988, pp. 314–329.
[12] J. Padhye, V. Firoiu, D. F. Towsley, and J. F. Kurose, "Modeling TCP reno performance: A simple model and its empirical validation," *IEEE/ACM Trans. Netw.*, vol. 8, no. 2, pp. 133–145, Apr. 2000.
[13] S. Floyd, "Highspeed tcp for large congestion windows," Tech. Rep., 2003.
[14] L. Xu, K. Harfoush, and I. Rhee, "Binary increase congestion control (BIC) for fast long-distance networks," in *Proc. INFOCOM 23rd AnnualJoint Conf. IEEE Comput. Commun. Societies*, vol. 4, 2004, pp. 2514–2524.
[15] K. Winstein and H. Balakrishnan, "TCP ex machina: Computer-generated congestion control," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 4, 2013, pp. 123–134.
[16] M. Dong, Q. Li, D. Zarchy, P. B. Godfrey, and M. Schapira, "PCC: Re-architecting congestion control for consistent high performance," in *Proc. 12th USENIX Symp. Netw. Syst. Des. Implementation*, vol. 1, no. 2.3, 2015, pp. 395–408.
[17] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson, "BBR: Congestion-based congestion control," *Queue*, vol. 14, no. 5, 2016, Art. no. 50.
[18] W. Li, F. Zhou, K. R. Chowdhury, and W. M. Meleis, "QTCP: Adaptive congestion control with reinforcement learning," *IEEE Trans. Netw. Ence Eng.*, vol. 6, no. 3, pp. 445–458, Jul.-Sep. 2019.
[19] S. B. Kotsiantis, I. Zaharakis, and P. Pintelas, "Supervised machine learning: A review of classification techniques," *Emerg. Artif. Intell. Appl. Comput. Eng.*, vol. 160, pp. 3–24, 2007.
[20] T. Joachims, "Transductive inference for text classification using support vector machines," in *Proc. Int. Conf. Mach. Learn.*, vol. 99, 1999, pp. 200–209.
[21] A. Bosch, A. Zisserman, and X. Munoz, "Image classification using random forests and ferns," in *Proc. IEEE 11th Int. Conf. Comput. Vis.*, 2007, pp. 1–8.
[22] A. Dainotti, A. Pescape, and K. C. Claffy, "Issues and future directions in traffic classification," *IEEE Netw.*, vol. 26, no. 1, 2012.
[23] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2016, pp. 785–794.

[24] L. Breiman, *Classification and regression trees*. London, U.K.: Routledge, 2017.
[25] T. Issariyakul and E. Hossain, "Introduction to Network Simulator 2 (NS2)," in *Introduction to Network Simulator NS2*. Springer, 2012, pp. 21–40.
[26] R. Caruana and A. Niculescu-Mizil, "An empirical comparison of supervised learning algorithms," in *Proc. 23rd Int. Conf. Mach. Learn.*, 2006, pp. 161–168.
[27] C. Sergiou, P. Antoniou, and V. Vassiliou, "A comprehensive survey of congestion control protocols in wireless sensor networks," *IEEE Commun. Surv. Tut.*, vol. 16, no. 4, pp. 1839–1859, Oct.–Dec. 2014.
[28] R. Jain, A. Durresi, and G. Babic, "Throughput fairness index: An explanation," *ATM Forum contribution*, vol. 99, no. 45, pp. 1–13, 1999.
[29] D. J. Leith, R. N. Shorten, and G. McCullagh, "Experimental evaluation of cubic-TCP," in *Proc. Protocols Fast Long Distance Networks*, 2008.

Guanglu Sun (Member, IEEE) received the B.S. degree in computer science and technology, the M.S.degree in pattern recognition and intelligent system, and the Ph.D. degree in computer application technology from the Harbin Institute of Technology, Harbin, China, in 2001, 2003, and 2008, respectively.

He was a Visiting Scholar with Northwestern University, Evanston, IL, USA, from 2014 to 2015. He is currently a Professor and the Director of the Center of Information Security and Intelligent Technology with Harbin University of Science and Technology, Harbin, China. His current research interests include computer networks and security, machine learning, and intelligent information processing.

Chuan Li received the B.S. degree in internet of things engineering from the Qufu Normal University, Shandong, China, in 2017, and the M.S. degree in computer science and technology from the Harbin University of Science and Technology, Harbin, China, in 2020.

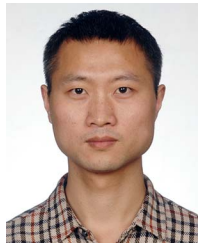
His current interests include machine learning and computer networks.

Yu Ma received the B.S. degree in computer science and technology, in 2018, from the Harbin University of Science and Technology, Harbin, China, where he is currently working toward the master's degree in computer science and technology.

His current interests include computer networks, reinforcement learning, and deep learning.

Shaobo Li received the M.S. degree in computer science and technology from the Harbin University of Science and Technology, Harbin, China, in 2018.

His current interests include machine learning and computer networks.



Jing Qiu received the B.S. degree in civil engineering from Harbin Institute of Technology, Harbin, China, in 2005 and the M.S. degree in computer science and technology from Harbin University of Science and Technology, Harbin, China, in 2009, and the Ph.D. degree in computer science and technology from Harbin Institute of Technology, in 2015.

He was with Harbin University of Science and Technology, from 2017 to 2021.