Learned Internet Congestion Control for Short Video Uploading

Tianchi Huang
Tsinghua University
htc19@mails.tsinghua.edu.cn

Chao Zhou*
Kuaishou
zhouchao@kuaishou.com

Lianchen Jia
KLPC, Tsinghua University
jlc21@mails.tsinghua.edu.cn

Rui-Xiao Zhang
Tsinghua University
zhangrx17@mails.tsinghua.edu.cn

Lifeng Sun*
BNRist, Tsinghua University
sunlf@tsinghua.edu.cn

ABSTRACT

Short video uploading service has become increasingly important, as at least 30 million videos are uploaded per day. However, we find that existing congestion control (CC) algorithms, either heuristics or learning-based, are not applicable for video uploading–i.e., lacking in the design of the fundamental mechanism and being short of leveraging network modeling. We present DuGu, a novel learning-based CC algorithm designed by considering the unique proprieties of video uploading via the probing phase and internet networking via the control phase. During the probing phase, DuGu leverages the transmission gap of uploading short videos to actively detect the network metrics to better understand network dynamics. DuGu uses a neural network (NN) to avoid congestion during the control phase. Here, instead of using handcrafted reward functions, the NN is learned by imitating the expert policy given by the optimal solver, improving both performance and learning efficiency. To build this system, we construct an omniscient-like network emulator, implement an optimal solver and collect a large corpus of real-world network traces to learn expert strategies. Trace-driven and real-world A/B tests reveal that DuGu supports multi-objective and rivals or outperforms existing CC algorithms across all considered scenarios.

CCS CONCEPTS

• **Networks** → *Network control algorithms.*

KEYWORDS

Congestion control, Imitation learning, Video uploading.

ACM Reference Format:
Tianchi Huang, Chao Zhou, Lianchen Jia, Rui-Xiao Zhang, Lifeng Sun. 2022. Learned Internet Congestion Control for Short Video Uploading. In *Proceedings of the 30th ACM International Conference on Multimedia (MM '22), Oct. 10–14, 2022, Lisboa, Portugal.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3503161.3548436

1 INTRODUCTION

Recent years have seen a rapid increase in short video services, as users upload almost 30 million User Generated Content (UGC) videos to Kuaishou every day [41]. Different from existing transmission services, short video uploading contains several unique processes, such as pre-transcoding and segmentation-based transmission, bringing new challenges to the Internet congestion control (CC) algorithm. (§2.1)

Internet CC algorithms have already been proposed for three decades [43]. It dynamically adjusts the sending rate to avoid congestion events for better stream transmission on the Internet. Off-the-shelf heuristics leverage packet loss and latency [8, 38] as the signals and dynamically controls the sending rate or congestion window (cwnd) to achieve high throughput while avoiding congestion [10]. While the network environments are becoming complex, it's reasonable to embrace machine learning techniques to design effective CC algorithms [43]. Following this insight, several attempts have been made to adopt deep reinforcement learning (DRL) to generate policies from clean slates, achieving outstanding performances across various network conditions [3, 4, 16, 22, 48, 52].

Nevertheless, we empirically show that recent CC algorithms are not fully applicable to the short video uploading task. On the one hand, most learning-based CC algorithms heavily rely on the design of reward functions. While such carefully designed reward functions not only fail to guide the NN to learn the proper mechanism but also often lead to unstable behavior in terms of the learned policy. In other words, we have to learn policies from another perspective. On the other hand, prior CC algorithms fail to capture the fundamental characteristics of the video uploading task. Meanwhile, they also neglect the importance of domain knowledge, which estimates the optimal policy inaccurately. (§2.2)

In this work, we propose DuGu[1], a novel learning-based CC algorithm for short video uploading tasks. Different from previously proposed methods [3, 48], DuGu "mimics" both expert mechanisms and policies by two phases. First, at each decision time, DuGu adopts a NN to determine the congestion window size (cwnd) for the next period in the *control phase*, aiming to send the packets with high throughput and low latency. Second, DuGu follows the principle of heuristics [8, 11], as it actively detects networks if the segment has not been generated in the *probing phase*. DuGu drains the queue to catch the minimum RTT periodically. Such operation can not only adapt to the network changes but also provide *fresh* network metrics to help improve the accuracy of the neural network (NN) policy. Moreover, it's also a feasible way to enhance the fairness [11]. (§3.1)

To better execute the policy in the control phase, we propose an imitation-learning-based training framework to train DuGu's NN.

MM '22, October 10–14, 2022, Lisboa, Portugal

ACM ISBN 978-1-4503-9203-7/22/10.
https://doi.org/10.1145/3503161.3548436

[1] DuGu (*Nine Swords of Dugu*): a sword skill in Chinese wuxia novel *The Smiling, Proud Wanderer* [49]. DuGu doesn't follow any fixed sequence or pattern, which allows the swordsman to quickly identify the weaknesses in the moves executed by an opponent.

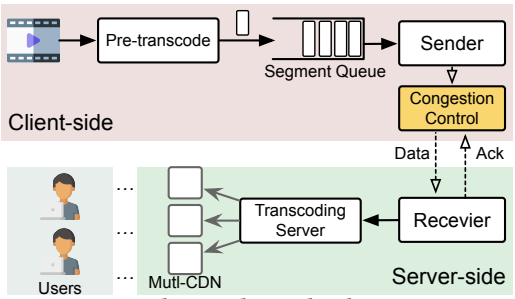
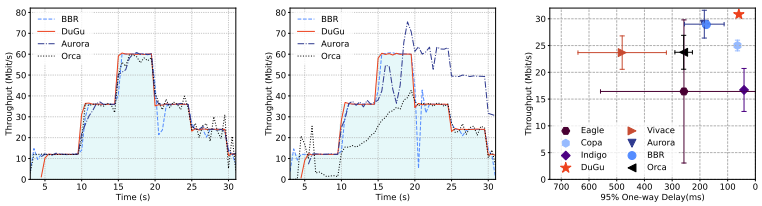


Figure 1: Short video uploading system.



(a) Best convergence behavior. (b) Worst convergence behavior. (c) Throughput-Delay

Figure 2: Examining learning-based algorithms over the same changeable networks. Error bars span ± one standard deviation from the average.

Imitation learning is an essential tool where an agent can expect to learn a policy with a slight regret w.r.t the expert [53]. Unlike recent model-free RL-based schemes [22, 52], DuGu is learned with the guidance of the *expert policy*, where the policy is represented as the Kleinrock's optimal operating point [25]. We design an omniscient-like network emulator and an optimal solver to obtain the point overall considered networks effectively, including time-varying networks. More specifically, given a network condition constructed by randomized network metrics, the learning agent adopts the omniscient-like network emulator to sample the trajectory w.r.t the NN policy. Meanwhile, it uses the optimal solver to instantly estimate the expert policy for each state and store the state-expert pair in the replay buffer. Consequently, the NN is faithfully optimized the NN via experience replay technologies. At the same time, inspired by recent multi-objective CC approaches [27, 28, 46, 51], we extend DuGu to support multi-objective requirements. (§3)

To make DuGu practical, we compare DuGu with twelve state-of-the-art CC algorithms and conduct a step-by-step evaluation process (§4): **i)** basic performance analysis indicates that DuGu achieves high throughput and low latency with different network conditions, such as varying stochastic loss rate, delay, buffer size, and bandwidth. **ii)** trace-driven emulation proves that DuGu improves the average power95 by 25.84%-30× compared with heuristics. Moreover, DuGu enhances the average throughput by 21.2%-131.59% and reduces the average 95-percentile delay of 1.3%-53.02% compared with learning-based algorithms. **iii)** in the "small scale" real-world experiments, DuGu obtains the highest Power95 [26] score among all baselines. **iv)** large-scale A/B testing illustrates the superiority of DuGu over the existing industry baseline. **v)** DuGu well supports multi-objective settings and demonstrates its fairness and friendliness with low computational overhead. In general, we summarize the contributions as follows:

- We first introduce short video uploading and highlight the short-coming of directly applying existing CC algorithms to that task. We then provide a feasible imitation learning-based solution to overcome such weaknesses. (§2)
- We propose DuGu, a novel learning-based that mimics both mechanism and expert policy. Meanwhile, we develop a training framework containing an omniscient-like network simulator to provide the expert policy precisely. (§3)
- We conduct comprehensive experiments, from basic performance analysis, trace-driven emulation to real-world A/B testing, to validate the performance of DuGu. Results demonstrate the superiority of DuGu across various experimental settings. (§4)

2 BACKGROUND AND MOTIVATION

2.1 Background and Related Work

Unlike live streaming uploading approaches [12, 23], the traditional short video uploading system is described in Fig 1, which consists of a client-side and a server-side. The users use a client-side App, often placed on a mobile (e.g., Kuaishou [1] and TikTok ([2]), to record a short video (i.e., a video that is 10-60 seconds in length [55]) and upload the video to their account. The recorded raw video is pre-transcoded at the high bitrate by the native encoder, where the additional operations contain dehazing, deraining, video enhancement, etc. Once the video has been encoded above a threshold (e.g., 1 MB), the encoded segment will be pushed into the segment queue and wait for the sender to transmit it to the server. The sender employs congestion control (CC) algorithms to adapt to various network conditions on the Internet according to the information acked by the receiver. The receiver emits the video to the transcoding server upon receiving all the segments. The server transcodes the video into several bitrate levels and pushes the encoded videos to multi-CDN. Users watch videos from the CDNs.

The most critical module of the video uploading systems is the congestion control algorithm, a historical yet classical topic that has been proposed for about three decades [43]. Heuristics often perform well in some situations but may backfire in others. For example, BBR [10] increases the data in *flight* (data sent but not yet acknowledged) to detect the bottleneck size and adjust the sending rate w.r.t the estimated bandwidth-delay product (BDP). Nevertheless, such operations obtain high throughput but may cause high queueing delay or even congestion. Another delay-based CC algorithm Copa [8] sends packets conservatively towards the target latency, which sometimes fails to achieve high link utilization. To cope with these issues, researchers have proposed learning-based CC algorithms, which ushered in a renaissance with the increase of deep learning technologies [3, 4, 16, 22, 48] – these approaches understand the control policy from clean slates and generalize well across a wide range of network conditions [50]. One of the most successful methods is deep reinforcement learning (DRL) since its capabilities to learn from fresh raw data and without relying on handcraft engineering [3].

2.2 Motivation

Unfortunately, existing CC algorithms are only "partially" suitable for the video uploading task. To prove this, we conduct a simple experiment to verify the convergence behavior of several CC algorithms. The experiment is done by Pantheon [48], with the network

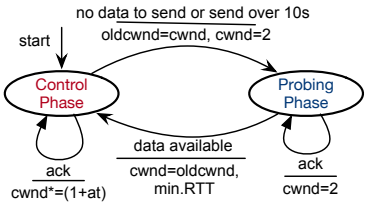


Figure 3: DuGu FSM

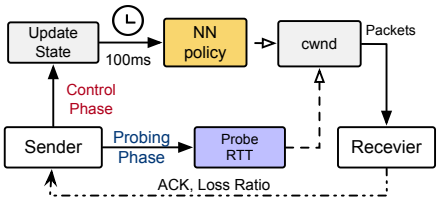


Figure 4: DuGu system overvew

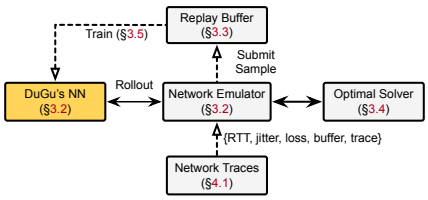


Figure 5: DuGu's training workflow

bandwidth changed every 5 seconds, minimum one-way delay as 30ms, stochastic loss as 0%, and 500 packets queue. We repeatedly test each algorithm five times. Results are reported as convergence behavior for each scheme and the relationship between throughput and 95% one-way delay in Figure 2. Here are some key findings.

Unstable behavior. Both bandwidth-delay product (BDP) and minimum RTT are changing over time in practice, while there's no efficient way to measure such contradicted network metrics synchronously. Thus, learning-based schemes often adopt specific reward functions to learn the specific rules or domain principles. For instance, Orca [3] integrates a bag of tricks into the reward function, aiming to implicitly enforce the agent to detect minimum RTT periodically like BBR [11]. Eagle [16] even enormously employs five reward functions to separate the CC policy into different BBR-like phases. Unfortunately, such a complex reward setting does obtain good performance, but it also brings out unstable behavior. There is a huge gap between the best and the worst convergence behavior of Aurora and Orca. By contrast, the typical scheme BBR is always on the right track since it leverages a simple yet effective scheme, namely RTT probing phase, to detect changeable networks [8, 10]. Hence, instead of utilizing "cumbersome" and handcrafted reward functions, *what about effectively learning the CC algorithm from the omniscient?*

Insufficient usage of domain knowledge. Moreover, recent CC schemes run in the transport layer fail to perceptualize the specific knowledge of the current task. While in this work, we find an opportunity to actively detect the network dynamics when the sender-side has no data to be sent. Moreover, recent work reveals that the Internet network model has stronger practical reference significance [8, 32, 48]. Based on this domain knowledge, Indigo [48] mimics the expert policy given by the Pantheon's emulator. However, Figure 2(c) shows that Indigo [48] does learn some correct policies from the domain knowledge while it still lacks solid performance. The key reason is that Indigo is only learned with very few types of network links, such as fixed bandwidth and stable latency environment. It's tough to enable Indigo's network emulator to support complex environments such as jitter links and time-varying networks [17]. To that end, *how to construct a unique mechanism for varying video uploading tasks? How to design a solution to estimate expert policy for all considered environments?*

In summary, existing CC schemes lack either sophisticated mechanism engineering or making full use of prior knowledge to infer the expert action. We, therefore, propose DuGu, a novel learning-based CC algorithm to solve the above challenges. In brief, DuGu is designed by considering the feature of the short video uploading task and mimics the expert actions for different kinds of networks. Figure 2 illustrates that DuGu performs stable, achieving high throughput with low latency.

3 DUGU MECHANISM

3.1 Big picture

In light of the characteristics of video uploading tasks, we propose DuGu, a novel learning-based CC algorithm. We model DuGu's control process as a finite state machine (FSM) as shown in Figure 3, which is mainly composed of the*control phase* and the *probing phase*. Here the states and transitions of FSM are explained as follows:

① *Control phase.* When the sender-side receives an ACK (acknowledgment) packet from the receiver side, FSM will be transited into the control phase. The sender analyzes useful network metrics from the ACK packet and updates the current state. At the same time, if the duration since the last decision time exceeds 100 milliseconds, the NN will take the current state as the input and estimates the proper action to adjust the congestion window size (cwnd).

② *Probing phase.* When there's no data to be transmitted (i.e., App. limits the transmission process), FSM will enter the probing phase. Technically, the sender stores the current cwnd and sets the cwnd to a minimum value (i.e., 2) for draining the network queue. At this time, it collects current RTT from the ACK packets and updates the minimum RTT into the current state. Once the segment has been encoded and ready to be sent (§2.2)), the sender will immediately recover the cwnd to the previously stored value. Moreover, inspired by the mechanism of recent heuristics [8, 11], FSM will be checked into the probing phase if the sender continually transmits data for 10 seconds as well. After finishing the probing phase, FSM enters the policy phase.

DuGu's system workflow is depicted in Figure 4. Having analyzed the primary mechanism of DuGu, we have to answer: *how to learn a good policy for the control phase?* Assuming all network parameters are observable, i.e., the network emulator is omniscient, the infinitive idea of finding the suitable policy is to compute the optimal cwnd for each iteration instantly. Thus, DuGu directly mimics the expert policy w.r.t the optimal action instead of maximizing the handcrafted reward function. To achieve this, we design a specific training flow for DuGu in Figure 5, where it contains a NN, an omniscient-like network emulator, and an optimal solver.

3.2 NN Overview

We explain the simple yet effective NN architecture as follows.

Inputs. At each decision time t, we pick several representative network information as a vector and take past k sequences as the state space s_t (see in Eq. 1).

$$s_t = \{r_t^{send}, r_t^{recv}, d_t^{min}, d_t^{queue}, l_t, c_t, dur_t, \omega\}. \quad (1)$$

Here r_{send} is the sending rate. r_{recv} denotes the EWMA of the receiving rate, which is calculated by the ACK (acknowledged) packets. d_{min} is the minimum observed latency (i.e., min. RTT).

d_{queue} is the estimated queuing delay w.r.t the current latency and d_{min}, i.e. $d_{queue} = d_t - d_t^{min}$. l is the packet loss rate. dur is the time interval since the last decision. All the above features are represented by vectors with past k sequences. ω is the multi-objective parameter. We employ the EWMA method to smooth sending rate, receiving rate, and queuing delay. We set the sequence number $k = 8$ to balance the trade-off between the performance and overhead.

Outputs. Upon observing state s_t, the agent outputs $a_t \in (-1, 1)$ to adjust the sending cwnd c_{t+1} for the next time slot $t+1$ via current cwnd c_t: $c_{t+1} = c_t(1 + a_t)$.

Architecture. We adopt 1D-CNN layers with 64 features and stride=1 to extract features for each metric and use fully connected layers with 64 neurons to further merge the features. The NN outputs a single scalar using *tanh* activation function. The choice of NN architectures will be discussed in §5.

3.3 Omniscient-like Network Emulator

Recall that we attempt to compute the oracle action $\pi^*(s_t)$ according to any state s_t, where the action is estimated by optimal cwnd – attaining Kleinrock's optimal operating point, at which the total data in-flight is equal to 1×BDP [25]. Finding such an optimal point is challenging in both fixed and time-varying networks since it requires obtaining *arbitrary network metrics at any time* at that instant. However, existing black-box emulators, such as Cellsim [45] and Mahimahi [32], are not applicable for this job because they cannot provide critical information in real-time, including queuing delay of each packet, the number of packets in the queue, the future bandwidth capacity, etc. We design an offline packet-level emulator, which can precisely *replay* the network environments w.r.t a saturated trace and a list of network parameters, such as packet loss, minimum one-way delay, jitter, and queue length (or buffer size).

The transmission process is composed of three parts: a loss module, a delay module, and a link module. The *loss module* is used to drop packets according to the given probability. The packet will be scholastically dropped if the randomized number is less than the given probability. The *delay module* emulates a link with a minimum one-way delay. The *link module* emulates a linked queue using packet-delivery traces. Unlike existing network emulators, all the metrics and variables in our proposed emulator are visible to the users.

3.4 Optimal Solver

Moreover, we design and build an optimal solver to compute the expert policy. The key idea of finding Kleinrock's optimal operating point is to obtain the actual link capacity for horizons of decision interval dur. Different from prior work [48] which can provide the expert policy for fixed networks only, We propose a two-phase process to precisely know the link capacity with any form of network traces, especially for time-varying networks. First, at decision time t, we free the time and store all the network status to o_t. We then apply a virtual process to *drain* all packets, including the packets in-flight and in the queue. Thus, we can obtain the time t' that the *first* sent packet is actually received by the receiver. Fasting forward to the time t', we then compute how many packets should be sent from t' to $t' + dur$, i.e., the "real" network link capacity. Finally, the network status is reset to o_t for executing the next step.

Moreover, motivated by recent multi-objective CC approaches [27, 28, 46, 51], we enable DuGu to support multi-objective requirements by integrating an importance weight ω, representing the target queuing delay. DuGu achieves high throughput but causes high queuing delay if we increase the ω. As suggested by prior work [45], we set the target queuing delay q_{tag} as 100 milliseconds if $\omega = 1.0$. In this work, we set $\omega = 0.5$ according to §5.

Putting them together, the target cwnd $\hat{c_t}$ is summarized calculated using Eq. 2, where p_x is packet count at time t. The solver outputs the expert action $\pi^*(s_t)$, representing the relative distance between the current cwnd and $\hat{c_t}$ (Eq. 3).

$$\hat{c_t} = (1 + \frac{\omega q_{tag}}{dur}) \frac{RTT_{min}}{dur} \int_{t'}^{t'+dur} p_x dx \tag{2}$$

$$\pi^*(s_t) = clip(\frac{\hat{c_t} - c_{t-1}}{c_{t-1}}, -1, 1) \tag{3}$$

3.5 Training Methodology

We now introduce the training algorithm of DuGu, which is basically implemented via imitation learning [33]. The imitation learning method allows the NN to explore environments and mimics the policy based on the expert policy. As listed in Eq. 4, the optimal policy $\hat{\pi}$ is optimized by minimizing the gap between the current policy and the expert policy according to the same state observed.

$$\hat{\pi} = arg\,min\, \mathbb{E}_{s \sim d_\pi} \left[l_t(\pi_t, \pi_t^*) \right]. \tag{4}$$

DuGu's training procedure is described as follows. First, we randomly pick network traces from the dataset with various network settings, including minimum latency, jitter, loss rate, and buffer size. Next, for each time slot t, the agent receives the state and takes action a_t w.r.t the policy given by the NN. In the meantime, the solver estimates expert policy $\hat{c_t}$ and computes the relative gap between action and expert. Here we adopt experience replay to restore a batch of state-expert sample pairs to train the DuGu's NN. Finally, we continually produce the process till the training ends.

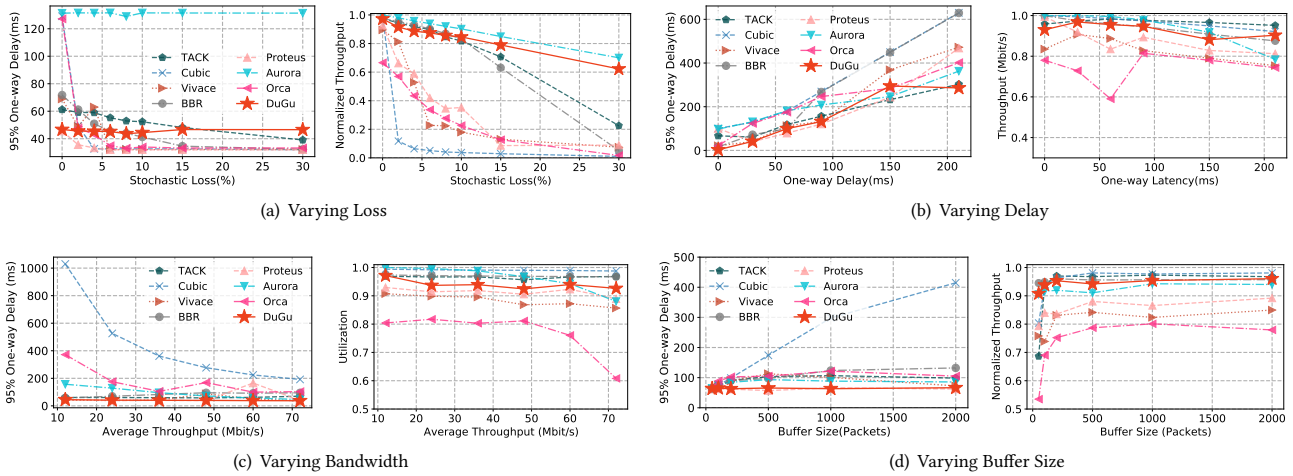
Experience replay. In light of the successes of recent off-policy RL methods, we apply a replay buffer to train the useful samples repeatedly to obtain better convergence behavior. During training, we store the state-expert strategy samples into the replay buffer and allow the agent to randomly picks the sample from the buffer. DuGu trains with diverse samples for each iteration, significantly improving sample efficiency.

Loss function. We use the squared-loss function between the output a_t returned by the NN and the expert label $\pi^*(s_t)$ (Eq. 5), since it enables better theoretical analysis (App. A).

$$l_t(\pi_t, \pi_t^*) = \ell_{\text{DuGu}} = \frac{1}{4} \left(a_t - \pi^*(s_t) \right)^2. \tag{5}$$

Learning efficiency. Using imitation learning methods can avoid not only redundant exploration but also make good use of the collected samples [21]. The training time of DuGu lasts less than an hour, much shorter than DRL-based methods (Figure 7).

Implementation. We build the network emulator with c++, and adopt Python and TensorFlow [40] to construct DuGu. We set learning rate $\alpha = 10^{-4}$ and use Adam [24] to optimize the model. Note that DuGu can be deployed solely online once it has been trained.



(a) Varying Loss (b) Varying Delay

(c) Varying Bandwidth (d) Varying Buffer Size

Figure 6: Examining learning-based CCAs in the same network setting, results are collected over the Pantheon environment.

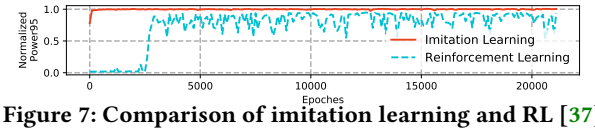


Figure 7: Comparison of imitation learning and RL [37].

Table 1: Range of env. during the training.

Traces	RTT	Jitter	Loss Rate	Buffer Size
0.1-300Mbps	0-800ms	0-400ms	0%-30%	0.1-3×BDP

4 EVALUATION

4.1 Experimental Setup

Training Setup. DuGu is effectively trained on the proposed network emulator. We collect about 2,000 real-world network traces in a totally of 60 hours, including: Orca [3], DeepCC [4] traces for wireless networks, FCC [34] network dataset for wired networks, and HSDPA [35] for cellular networks. Meanwhile, we adopt Saturator [45] to collect 50 network traces by ourselves, containing diverse network environments such as social libraries and cybercafes. We use 80% of the data for training and 20% of the data for validation. In addition, Table 1 lists the network parameters used during the training. It's worth noting that the parameter ranger is much larger than that of prior DRL-based approaches since imitation learning has better learning efficiency compared with DRL.

Testbed Setup. Upon training a network policy, we use Pantheon [48] to evaluate DuGu with both various offline emulation and real-world experiments. For the local emulation, we use Mahimahi [32].

CC Baselines. We take twelve state-of-the-art CC algorithms published in very recent years as the baselines. **1) BBRv2 (BBR) [11]:** a model-based approach that detects maximum bandwidth and minimum RTT periodically; **2) Cubic [18]:** a loss-based approach, which utilizes additive increase/multiplicative decrease (AIMD) scheme to avoid congestion events; **3) Copa [8]:** a delay-based heuristic, which computes the target sending rate by the estimated minimum delay; **4) TACK [26]:** an innovative model-based mechanism that gives a full protocol design with minimized ACK frequency required on the transport layer; **5) Ledbat [38]:** a classic delay-based scheme that is friendly to TCP protocols. **6) Aurora [22]:** the first DRL-based CC approach; **7) Orca [3]:** a learning-assisted scheme that utilizes NN to help improve Cubic; **8) Eagle [16]:** a learning-based approach that uses reinforcement learning to learn the BBR's fundamental mechanism; **9) Indigo [48]:** a learning-based CC algorithm via behavior cloning, where the algorithm is trained with limited network conditions on Mahimahi. Furthermore, we also use three online-learning approaches as the baselines, including **10) PCC Allegro (Allegro) [14]**, **11) PCC Vivace (Vivace) [15]**, and **12) PCC Proteus (Proteus) [15]**.

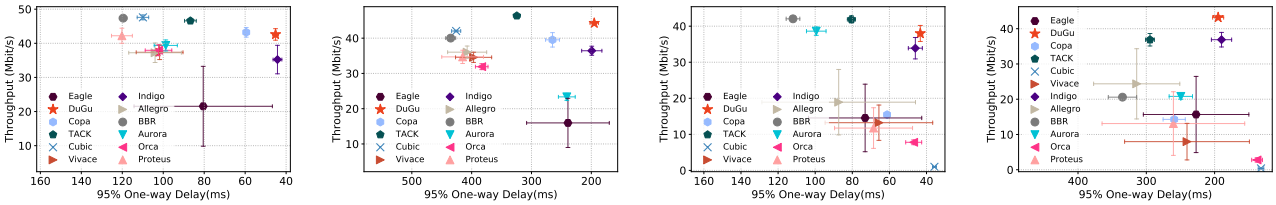
4.2 Basic Performance Analysis

We validate DuGu against the above CC algorithms with basic network settings, which sweep the values of minimum latency, stochastic loss rate, buffer size, and bandwidth bottlenecks. Each scheme is repeatedly tested five times.

Stochastic Loss. We fix network bandwidth as 12Mbps and let the stochastic loss range from 0% to 30%, which has already covered most network situations. As expected, Cubic, the loss-based method, fails to overcome the network condition if the stochastic loss is more prominent than 2%. Orca is incrementally implemented based on Cubic. Thus it illustrates the same observation. Moreover, model-based approach such as BBR and TACK performs well if the loss is lower than 10%, but the throughput degrades heavily if the loss is above 10%. Both Vivace and Proteus cannot handle all loss rates for the learning-based schemes. Aurora obtains the best link utilization while it also performs with high latency. In contrast, DuGu can provide high throughput and low latency under all stochastic loss rates. We report the results in Figure 6(a).

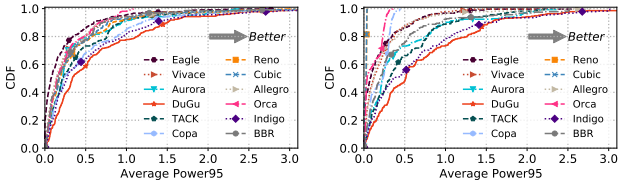
Delay. We then set the network as previous settings but adjusted the minimum one-way delay from 0ms to 200ms. Unsurprisingly, DuGu rivals or outperforms recent heuristics, with the reduction of 95-percentile of one-way delay of 50%-60% compared with Cubic and BBR in a similar throughput. What's more, DuGu surpasses the state-of-the-art learning-based approach Orca on both average throughput and delay, yielding the high stability of DuGu.

Buffer Size. Next, we fixed the network bandwidth as 48Mbps, 30 ms minimum delay, 0% loss, and adjusted the buffer size from 10 to 2000 packets. DuGu keeps its performance above the link utilization



(a) Min.RTT=60ms, loss=0% (b) Min.RTT=240ms, loss=0% (c) Min.RTT=60ms, loss=3% (d) Min.RTT=240ms, loss=3%

Figure 8: Comparing DuGu with existing CC algorithms across various network traces. Results are demonstrated with the comparison of several underlying metrics.



(a) DuGu vs. CC, loss=0% (b) DuGu vs. CC, loss=3%

Figure 9: Comparing DuGu with existing CC algorithms across various network traces (CDF of Power95).

of 0.9 across all scenarios as maintaining the 95-percentile one-way delay of around 50ms. It isn't easy since none of the other CC algorithms also complete this task. For example, TACK, BBR, and Cubic result in good link utilization but high one-way delay. Proteus achieves a suitable one-way delay but fails to reach high link utilization. Moreover, as much as Aurora is well behaved on this task, DuGu also improves the average throughput by 2% and it heavily decreases the average 95-percentile one-way delay by 21% compared with Aurora.

Bandwidth. Finally, we investigate how DuGu works in different network bandwidths. Here we set the minimum one-way delay as 30ms, loss as 0%, and buffer size as 1000 packets. The bandwidth is swept from 12 - 72Mbps. We report the average link capacity and 95-percentile one-way delay of each CC scheme. DuGu stands for the top-3 scheme by considering link utilization and one-way delay. Furthermore, the performance of learning-based approaches degrades when the network bandwidth is over 50Mbps. We reason that learning-based schemes, except Orca, adopt a network dataset with a limited parameter range for training, which lacks sufficient generalization abilities.

4.3 Trace-driven Emulation

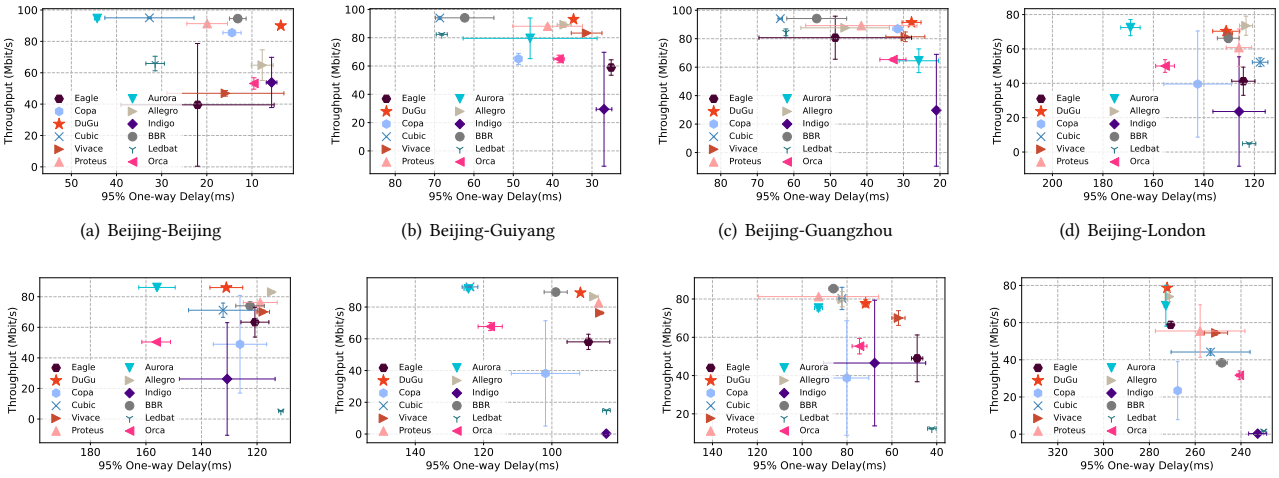
In this part, we use cellular network environments (i.e. time-varying networks) with different network settings (i.e., {60ms,240ms} minimum RTT, {0%, 3%} stochastic loss) and 1 × BDP buffer). We use a version of Kleinrock's power metric [26, 48], namely Power95 ($\frac{throughput}{95-percentile\ delay}$) to measure the performance of the baselines for video uploading. Results are shown in Figure 8, where the top right region of the figures is the desired operation region for any CC algorithm: gaining high throughput while achieving low delay, and error bars show the average standard deviation of an algorithm performed in the same environment.

DuGu vs. Heuristics. Comparing DuGu with recent heuristic-based approaches highlights one of our key motivations: an excellent CC algorithm is inseparable from the design principle of its mechanism. Figure 9(a) depicts the results with the stochastic loss as 0%. DuGu outperforms others, which can achieve 25.84%-95.28% better Power95 compared to heuristics (i.e, Copa: 25.84%, TACK: 50.94%, Reno: 72.2%, Cubic: 78.15%, BBR: 95.28%). Moreover, Figure 9(b) illustrates the comparison results with 3% stochastic loss. We observe that the performance of loss-based approaches like Cubic degrades heavily. Other model-based approaches (e.g., BBR and TACK) perform well in network emulations with 30ms RTT, while they fail to provide acceptable link utilization in the dynamic link emulation with 120ms RTT. The key reason lies in the rationality of the mechanism. DuGu mimics expert principle for the probing phase which can achieve up to 30× compared with Cubic. As shown in Figure 8(a) (min. RTT=60ms) and Figure 8(b) (min. RTT=240ms), comparing DuGu with the best heuristic-based method, TACK, we see that DuGu rivals TACK in terms of the average throughput but it heavily reduces 38%-41% 95-percentile one-way-delay.

DuGu vs. Learning-based CC Algorithms. Moreover, we discuss the comparison results of DuGu and existing learning-based schemes. First, Figure 9(a) illustrates that DuGu consistently stands for the best scheme among all candidates, which improves average power95 by 13.59% (Indigo) to 1.7× (Eagle). Meanwhile, we also observe a similar rank with 3% stochastic loss. Figure 9(b) demonstrates that DuGu outperforms others with up to 5.9× better power95 metric. In addition, as demonstrated in Figure 8, DuGu consistently achieves its high performance with low one-way delay, outperforming recent learning-based baselines with up to 1.31× better throughput and 53.02% latency reduction.

▹ **DuGu vs. Indigo.** Indigo also leverages the imitation learning method to learn the policy from scratch. The critical difference between Indigo and DuGu is the way to obtain the expert policy. Figure 8 shows that DuGu betters Indigo on average throughput of 21.22% and 95-percentile delay of 1.3% with 0% packet loss environment, as it also improves average throughput by 13.86% and decreases delay of 0.12% with 3% loss. In other words, the advantages of the mechanism enable DuGu to increase at least 13% throughput with a similar delay. Moreover, compared with Indigo, DuGu shows outstanding behavior in terms of robustness, especially when min. RTT reaches 240ms. Here please recall that, "robustness" is a critical factor in determining whether the CC algorithm can be successfully deployed in the real-world.

▹ **DuGu vs. PCC family.** Here we see that three PCC algorithms have almost the same behavior: they achieve high throughput but high latency with no loss network conditions and perform low throughput with 3% loss. Proteus performs slightly better than the



(a) Beijing-Beijing (b) Beijing-Guiyang (c) Beijing-Guangzhou (d) Beijing-London

(e) Beijing-Frankfurt (f) Beijing-Sydney (g) Beijing-Tokyo (h) Beijing-Johannesburg

Figure 10: We conduct a real-world evaluation on the real Internet, which is composed of *eight* servers.

other two algorithms. We reason that the online learning methods rely on a formal utility function (or reward function), while the existing weighted reward function hardly maps the actual requirement for all networks [19]. DuGu doesn't have such issues since it's optimized by the expert policy rather than maximizing rewards.

▹ **DuGu vs. Orca.** Orca's two-level mechanism enables the NN policy to control Cubic. However, Orca is still constrained by the *harmful* mechanism of Cubic. Figure 8(c) and Figure 8(d) show that, as much as Orca has tried its best to overcome the hostile policy acted by Cubic, we also observe a heavy drop in the overall throughput compared to that in the zero-loss network. Hence, we must recall that the fundamental mechanism is a non-trivial cornerstone for implementing an excellent learning-based CC algorithm. DuGu performs well in all considered network conditions.

4.4 Real-world Evaluation

To better understand how DuGu works in the real world, we comprehensively evaluate the performance on the real Internet by utilizing eight servers located around the world, which include: Beijing, Guangzhou, Guiyang, and Tokyo from Asia; Sydney from Australia; Frankfurt and London from Europe; Johannesburg from Africa. All the experiments are completed on the Pantheon platform [48]. We, located in Beijing, send a flow using the CC scheme to test the video uploading task for each client-server link. The scheme is tested three times and picked by random order. The entire testing process lasts over 4 hours. We evaluate the performance for more than 15 days, from July 12 to July 28, 2021, and totally collected more than 500GB of data, which is almost two times of Orca's [3].

We categorize the servers into two network types, i.e., intra-continental (in China) and inter-continental scenarios (the other five servers). The results of intra-continental scenarios are shown in Figure 10(a) to Figure 10(c), in which the error bar indicates the standard deviation of an algorithm running in the same network. We have three observations here. First, DuGu achieves high and stable performance, which rivals BBR on average throughput while decreasing the average 95-percentile one-way delay up to 2×. Especially in the Beijing-Beijing link 10(a), DuGu's performance even fulfills the requirement of cloud gaming [5]. Apart from DuGu, none of the other algorithms can transmit such a high throughput (90.03Mbps) with low latency (3.60ms). It's an interesting observation that motivates us to formally verify DuGu in future work [7]. Besides, with the increase in minimum delay, we can see that learning-based approaches, including Aurora, Eagle, and Indigo, perform with unstable behavior. In contrast, DuGu reaches the right top of the figure in all considered scenarios, demonstrating the superiority against recent schemes.

Next, from the results on the inter-continental link, we see that the results of Ledbat [38] demonstrate the critical difference between intra-continental and inter-continental scenarios: the large min. RTT makes it difficult for the delay-based algorithm to obtain high link utilization w.r.t the estimated queuing delay (default 100ms). Eagle performs much better in the inter-continental link than in the intra-continental connection on both efficiency and stability, ranking top-5 competitive among all candidates. We reason that Eagle learns the policy w.r.t the BBR's mechanism, suitable for the inter-continental link. Orca follows the fundamental principle of Cubic, resulting in poor performances. By contrast, DuGu mimics BBR's mechanism, achieving high throughput and low delay. Further, the PCC family also maintains their outstanding abilities. In particular, both Allegro and Vivace show extraordinary stability in Figure 10(e) (Frankfurt), Figure 10(f) (Sydney) and Figure 10(g) (Tokyo). We confirm that the inter-continental link is often constructed by a large buffer queue [3], which can provide an accurate delay signal. The signal enhances the effectiveness of online learning. The only exception is Proteus since it changes its CC policy periodically. We can also see that DuGu rivals or outperforms the PCC family, especially in the Beijing-Johannesburg link. It's challenging because South Africa is over 8,000 miles away from China, leading to a pretty *huge* RTT (almost 500ms).

Finally, we observe that Copa performs well in intra-continental networks but is vividly fair and unstable in inter scenarios. One of the reliable reasons is Copa's mechanism can hardly estimate the min. RTT in the interlink, while DuGu leverages meticulous probing phase that accurately detects whether the network condition has been changed.

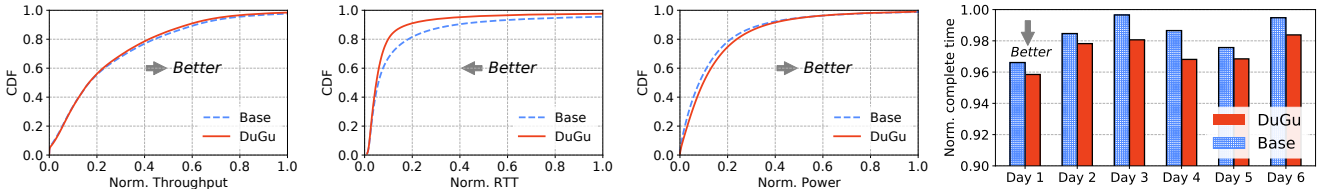


Figure 11: Performance comparison with production CC policy. Results are tested on the Kuaishou's short video uploading task.

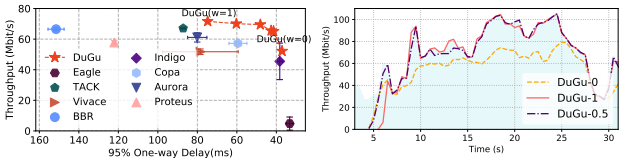


Figure 12: DuGu with different weights.

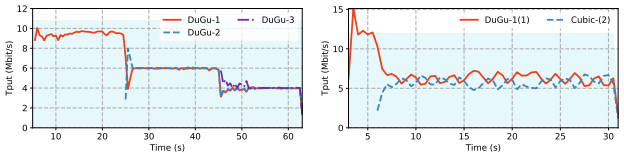


Figure 13: DuGu's fairness and friendliness.

Table 2: DuGu with different NN architectures.

DuGu	1D-CNN (32)	1D-CNN (64)	GRU (128)	GRU (256)
Inference (ms)	0.38	0.66	1.38	2.27

4.5 A/B Testing

Finally, we test DuGu for the short video uploading service in the Kuaishou App [1], a popular commercial short video platform, and present results from two A/B tests with millions of real upload videos each day in China, over four weeks from December 2021 to January 2022. DuGu's primary process is implemented by c++ and deployed as a part of KTP (i.e., Kuaishou Transport Protocol). The DuGu's NN policy is executed by YCNN (i.e., Kuaishou's deep learning inference engine), written in c++ as well. During the test, we compare DuGu with Kuaishou's base CC algorithm – an extremely tuned heuristic with several rounds of A/B testing, spanned almost three years.

We have tried many typical DNNs, such as Fully-Connected (FC), GRU [13], or even Transformer-based architectures [42]. Unfortunately, FC-based DuGu lacks performance and Transformer-based DuGu consumes too much computational time (i.e., almost 4ms). A/B tests illustrate that GRU and 1D-CNN based DuGu perform similarly but their average inference time is different.

Figure 11 shows detailed results of 1D-CNN model. There are three key takeaways from these results. First, DuGu achieves similar performance in terms of average throughput against the base algorithm while heavily reducing the average RTT by 24.57%. We reason that DuGu faithfully controls the cwnd as 1×BDP, resulting in high performance with low buffer utilization. Second, DuGu improves average Power by 3% compared with the base algorithm due to its advantages in the low RTT metrics. Finally, we show a detailed comparison of DuGu and the baseline on average completion time for six consecutive days. As expected, DuGu performs stably better than the base group by 0.66%-1.91% in several successive days of experiments, with the average improvement of 1.35%–it's indeed a significant improvement. On the one hand, DuGu is directly deployed on KTP, and apparently, it does not adapt to KTP's FEC and ARQ methods [9]. We believe DuGu would perform better if we considered all factors. On the other hand, the baseline algorithm has been updated for two years, almost representing the upper bound of heuristics. Meanwhile, the result is comparable to the magnitude of the total benefit reported by some academic work that used real-world experiments [6, 29, 47].

5 DISCUSSION

Multi-objectives. We validate DuGu with different weights over the high-variance bandwidth collected from the real world. DuGu is configured with multiple weights ranging from 0 to 1 (i.e., DuGu-ω). We plot the throughput dynamic in Figure 12. DuGu-0 (i.e., DuGu with $\omega = 0$) performs conservative to avoid congestion events and cannot achieve high link utilization. By contrast, DuGu-1 performs aggressively, almost fully utilizing the bottleneck. We see DuGu-0.5 reaches the best of both worlds.

Fairness. DuGu follows the principle of BBR [11] and Copa [8], as these two heuristics that have already proved their fairness. To validate fairness, we initiate three flows in an emulated network. Each flow starts in a 10-second-interval. Figure 13 shows the throughput of the three flows performed by DuGu. We can see that DuGu with the same default weight achieves a fair share. Furthermore, DuGu also achieves TCP friendliness when $w = 1$.

Overhead. During A/B testing, one of the critical metrics is the computational overhead of executing DuGu. We validate DuGu with two NN architectures and two-parameter settings. In Table 2, results are computed as *average inference time* over *all users in the group*. Almost all configurations can be successfully performed on the user side – most mobiles are sufficient to execute such light-weighted NNs. While considering the real-time requirement of the server-side, we believe that distilling policies as decision tree models should be a practical approach [54].

6 CONCLUSION

We presented DuGu, a congestion control algorithm for short video uploading. By leveraging domain principle, DuGu mimics both the mechanism and strategy from the expert. The DuGu's mechanism leveraged the probing phase to detect the network change periodically and adopted a NN for adjusting the cwnd for achieving high throughput and low latency, We constructed several essential tools and modules to effectively train DuGu via imitating learning. We made comprehensive experiments to prove the advantages of DuGu across a wide range of network scenarios.

Acknowledgements. We thank the ACM MM reviewers for the valuable feedback. Besides App. F, we also thank Tianchi's wife, Yuyan Chen, aka. Comyco Chen, for her great support again – It's really a long journey, from theory to practice. This work was supported by NSFC under Grant 61936011, Beijing Key Lab of Networked Multimedia, and Kuaishou-Tsinghua Joint Project.

REFERENCES

[1] 2022. Kuaishou. https://kuaishou.com.

[2] 2022. Tiktok Technology. https://tiktok.com.

[3] Soheil Abbasloo, Chen-Yu Yen, and H Jonathan Chao. 2020. Classic meets modern: A pragmatic learning-based congestion control for the Internet. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*. 632–647.

[4] Soheil Abbasloo, Chen-Yu Yen, and H Jonathan Chao. 2020. Wanna Make Your TCP Scheme Great for Cellular Networks? Let Machines Do It for You! *IEEE Journal on Selected Areas in Communications* 39, 1 (2020), 265–279.

[5] Ahmad Alhilal, Tristan Braud, Bo Han, and Pan Hui. 2022. Nebula: Reliable Low-latency Video Transmission for Mobile Cloud Gaming. *arXiv preprint arXiv:2201.07738* (2022).

[6] Abdullah Alomar, Pouya Hamadanian, Arash Nasr-Esfahany, Anish Agarwal, Mohammad Alizadeh, and Devavrat Shah. 2022. CausalSim: Toward a Causal Data-Driven Simulator for Network Protocols. *arXiv preprint arXiv:2201.01811* (2022).

[7] Venkat Arun, Mina Tahmasbi Arashloo, Ahmed Saeed, Mohammad Alizadeh, and Hari Balakrishnan. 2021. Toward Formally Verifying Congestion Control Behavior. *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication* (2021).

[8] Venkat Arun and Hari Balakrishnan. 2018. Copa: Practical delay-based congestion control for the internet. In *15th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 18)*. 329–342.

[9] Chadi Barakat, Eitan Altman, and Walid Dabbous. 2000. On TCP performance in a heterogeneous network: a survey. *IEEE Communications Magazine* 38, 1 (2000), 40–46.

[10] Neal Cardwell, Yuchung Cheng, C Stephen Gunn, Soheil Hassas Yeganeh, and Van Jacobson. 2016. BBR: Congestion-Based Congestion Control: Measuring bottleneck bandwidth and round-trip propagation time. *Queue* 14, 5 (2016), 20–53.

[11] Neal Cardwell, Yuchung Cheng, S Hassas Yeganeh, Ian Swett, Victor Vasiliev, Priyaranjan Jha, Yousuk Seung, Matt Mathis, and Van Jacobson. 2019. BBRv2: A model-based congestion control. In *Presentation in ICCRG at IETF 104th meeting*.

[12] Ying Chen, Qing Li, Aoyang Zhang, Longhao Zou, Yong Jiang, Zhimin Xu, Junlin Li, and Zhenhui Yuan. 2021. Higher quality live streaming under lower uplink bandwidth: an approach of super-resolution based video coding. In *Proceedings of the 31st ACM Workshop on Network and Operating Systems Support for Digital Audio and Video*. 74–81.

[13] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555* (2014).

[14] Mo Dong, Qingxi Li, Doron Zarchy, P Brighten Godfrey, and Michael Schapira. 2015. {PCC}: Re-architecting congestion control for consistent high performance. In *12th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 15)*. 395–408.

[15] Mo Dong, Tong Meng, Doron Zarchy, Engin Arslan, Yossi Gilad, Brighten Godfrey, and Michael Schapira. 2018. {PCC} vivace: Online-learning congestion control. In *15th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 18)*. 343–356.

[16] Salma Emara, Baochun Li, and Yanjiao Chen. 2020. Eagle: Refining congestion control by learning from the experts. In *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*. IEEE, 676–685.

[17] Prateesh Goyal, Mohammad Alizadeh, and Thomas E Anderson. 2022. Optimal Congestion Control for Time-varying Wireless Links. *arXiv preprint arXiv:2202.04321* (2022).

[18] Sangtae Ha, Injong Rhee, and Lisong Xu. 2008. CUBIC: a new TCP-friendly high-speed TCP variant. *ACM SIGOPS operating systems review* 42, 5 (2008), 64–74.

[19] Tianchi Huang, Ruixiao Zhang, and Lifeng Sun. 2021. Zwei: A Self-play Reinforcement Learning Framework for Video Transmission Services. *IEEE Transactions on Multimedia* (2021).

[20] Tianchi Huang, Chao Zhou, Xin Yao, Rui-Xiao Zhang, Chenglei Wu, Bing Yu, and Lifeng Sun. 2020. Quality-aware neural adaptive video streaming with lifelong imitation learning. *IEEE Journal on Selected Areas in Communications* 38, 10 (2020), 2324–2342.

[21] Tianchi Huang, Chao Zhou, Rui-Xiao Zhang, Chenglei Wu, Xin Yao, and Lifeng Sun. 2019. Comyco: Quality-aware adaptive video streaming via imitation learning. In *ACM Multimedia 2019*.

[22] Nathan Jay, Noga Rotman, Brighten Godfrey, Michael Schapira, and Aviv Tamar. 2019. A deep reinforcement learning perspective on internet congestion control. In *International Conference on Machine Learning*. PMLR, 3050–3059.

[23] Jaehong Kim, Youngmok Jung, Hyunho Yeo, Juncheol Ye, and Dongsu Han. 2020. Neural-enhanced live streaming: Improving live video ingest via online learning. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*. 107–125.

[24] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[25] Leonard Kleinrock. 1979. Power and deterministic rules of thumb for probabilistic problems in computer communications. In *Proceedings of the International Conference on Communications*, Vol. 43. 1–43.

[26] Tong Li, Kai Zheng, Ke Xu, Rahul Arvind Jadhav, Tao Xiong, Keith Winstein, and Kun Tan. 2020. Tack: Improving wireless transport performance by taming acknowledgments. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*. 15–30.

[27] Xu Li, Feilong Tang, Jiacheng Liu, Laurence T Yang, Luoyi Fu, and Long Chen. 2021. {AUTO}: Adaptive Congestion Control Based on Multi-Objective Reinforcement Learning for the Satellite-Ground Integrated Network. In *2021 {USENIX} Annual Technical Conference ({USENIX} ATC 21)*. 611–624.

[28] Yiqing Ma, Han Tian, Xudong Liao, Junxue Zhang, Weiyan Wang, Kai Chen, and Xin Jin. 2021. Multi-Objective Congestion Control. *arXiv preprint arXiv:2107.01427* (2021).

[29] Hongzi Mao, Shannon Chen, Drew Dimmery, Shaun Singh, Drew Blaisdell, Yuandong Tian, Mohammad Alizadeh, and Eytan Bakshy. 2020. Real-world video adaptation with reinforcement learning. *arXiv preprint arXiv:2008.12858* (2020).

[30] Tong Meng, Neta Rozen Schiff, P Brighten Godfrey, and Michael Schapira. 2020. PCC Proteus: scavenger transport and beyond. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*. 615–631.

[31] Zili Meng, Yaning Guo, Yixin Shen, Jing Chen, Chao Zhou, Minhu Wang, Jia Zhang, Mingwei Xu, Chen Sun, and Hongxin Hu. 2021. Practically deploying heavyweight adaptive bitrate algorithms with teacher-student learning. *IEEE/ACM Transactions on Networking* 29, 2 (2021), 723–736.

[32] Ravi Netravali, Anirudh Sivaraman, Somak Das, Ameesh Goyal, Keith Winstein, James Mickens, and Hari Balakrishnan. 2015. Mahimahi: Accurate record-and-replay for {HTTP}. In *2015 {USENIX} Annual Technical Conference (ATC 15)*. 417–429.

[33] Takayuki Osa, Joni Pajarinen, Gerhard Neumann, J Andrew Bagnell, Pieter Abbeel, and Jan Peters. 2018. An algorithmic perspective on imitation learning. *arXiv preprint arXiv:1811.06711* (2018).

[34] Measuring Fixed Broadband Report. 2016. Raw Data Measuring Broadband America 2016. https://www.fcc.gov/reports-research/reports/measuring-broadband-america/raw-data-measuring-broadband-america-2016. [Online; accessed 19-July-2016].

[35] Haakon Riiser, Paul Vigmostad, Carsten Griwodz, and Pål Halvorsen. 2013. Commute path bandwidth traces from 3G networks: analysis and applications. In *Proceedings of the 4th ACM Multimedia Systems Conference*. ACM, 114–118.

[36] Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. 2011. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings, 627–635.

[37] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347* (2017).

[38] Sea Shalunov, Greg Hazel, Janardhan Iyengar, Mirja Kuehlewind, et al. 2012. Low extra delay background transport (LEDBAT). In *RFC 6817*.

[39] Anirudh Sivaraman, Keith Winstein, Pratiksha Thaker, and Hari Balakrishnan. 2014. An experimental study of the learnability of congestion control. *ACM SIGCOMM Computer Communication Review* 44, 4 (2014), 479–490.

[40] Daniel Smilkov, Nikhil Thorat, Yannick Assogba, Ann Yuan, Nick Kreeger, Ping Yu, Kangyi Zhang, Shanqing Cai, Eric Nielsen, David Soergel, et al. 2019. Tensorflow.js: Machine learning for the web and beyond. *arXiv preprint arXiv:1901.05350* (2019).

[41] Kuaishou Technology. 2022. ANNOUNCEMENT OF THE RESULTS FOR THE YEAR ENDED DECEMBER 31, 2021. https://ir.kuaishou.com/system/files-encrypted/nasdaq_kms/assets/2022/03/29/0-05-35/E_887559_KUAISHOU-W_0325_1941_ESS.pdf.

[42] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in neural information processing systems*. 5998–6008.

[43] Wenting Wei, Huaxi Gu, and Baochun Li. 2021. Congestion Control: A Renaissance with Machine Learning. *IEEE Network* (2021).

[44] Keith Winstein and Hari Balakrishnan. 2013. Tcp ex machina: Computer-generated congestion control. *ACM SIGCOMM Computer Communication Review* 43, 4 (2013), 123–134.

[45] Keith Winstein, Anirudh Sivaraman, and Hari Balakrishnan. 2013. Stochastic forecasts achieve high throughput and low delay over cellular networks. In *10th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 13)*. 459–471.

[46] Zhenchang Xia, Yanjiao Chen, Libing Wu, Yu-Cheng Chou, Zhicong Zheng, Haoyang Li, and Baochun Li. 2021. A Multi-objective Reinforcement Learning

Perspective on Internet Congestion Control. In *IWQOS2021*. 0–0.

[47] Francis Y Yan, Hudson Ayers, Chenzhi Zhu, Sadjad Fouladi, James Hong, Keyi Zhang, Philip Levis, and Keith Winstein. 2020. Learning in situ: a randomized experiment in video streaming. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*. 495–511.

[48] Francis Y Yan, Jestin Ma, Greg D Hill, Deepti Raghavan, Riad S Wahby, Philip Levis, and Keith Winstein. 2018. Pantheon: the training ground for Internet congestion-control research. In *2018 {USENIX} Annual Technical Conference (USENIX ATC 18)*. 731–743.

[49] Jin Yong. 1967. The Smiling, Proud Wanderer. https://en.wikipedia.org/wiki/The_Smiling,_Proud_Wanderer.

[50] Lei Zhang, Yong Cui, Mowei Wang, Zhenjie Yang, and Yong Jiang. 2019. Machine learning for internet congestion control: Techniques and challenges. *IEEE Internet Computing* 23, 5 (2019), 59–64.

[51] Lei Zhang, Yong Cui, Mowei Wang, Kewei Zhu, Yibo Zhu, and Yong Jiang. 2021. DeepCC: Bridging the Gap Between Congestion Control and Applications via Multi-Objective Optimization. *arXiv preprint arXiv:2107.08617* (2021).

[52] Lei Zhang, Kewei Zhu, Junchen Pan, Hang Shi, Yong Jiang, and Yong Cui. 2020. Reinforcement Learning Based Congestion Control in a Real Environment. In *2020 29th International Conference on Computer Communications and Networks (ICCCN)*. IEEE, 1–9.

[53] Boyuan Zheng, Sunny Verma, Jianlong Zhou, Ivor Tsang, and Fang Chen. 2021. Imitation Learning: Progress, Taxonomies and Opportunities. *arXiv preprint arXiv:2106.12177* (2021).

[54] Zhiren Zhong, Wei Wang, Yiyang Shao, Zhenyu Li, Heng Pan, Hongtao Guan, Gareth Tyson, Gaogang Xie, and Kai Zheng. 2022. Muses: Enabling Lightweight Learning-Based Congestion Control for Mobile Devices. In *IEEE INFOCOM 2022-IEEE Conference on Computer Communications*. IEEE, 1–10.

[55] Chao Zhou, Shucheng Zhong, Yufeng Geng, and Bing Yu. 2018. A Statistical-based Rate Adaptation Approach for Short Video Service. In *2018 IEEE Visual Communications and Image Processing (VCIP)*. IEEE, 1–4.

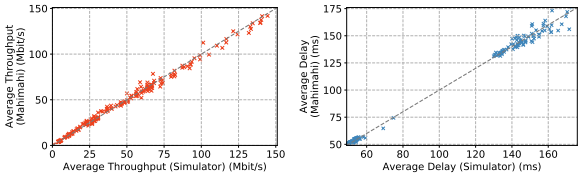
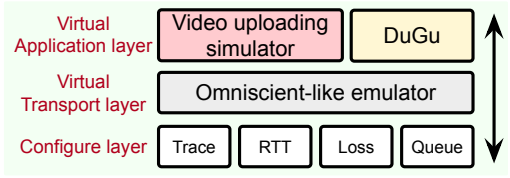
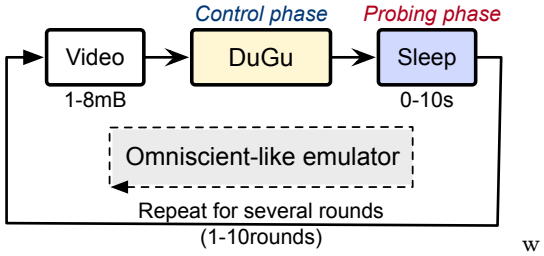


Figure 15: Results comparison between our proposed network emulator and real network emulator Mahimahi [32]. We show the close relationship between these two schemes.



(a) Structure of our simulators



(b) Short video uploading simulator overview

Figure 14: Details of short video uploading simulator.

Appendices

A PROOF

Suppose we sample m trajectories with π_i at iteration i. Let ϵ_N be the average loss of the best policy on w.r.t the collected samples and N be the total training iteration. We have the following upper bound of the ℓ_{DuGu}:

Theorem A.1. *For* DuGu, *if* $N = O(T^2 log(1/\sigma))$ *and* $m = O(1)$ *with the probability at least* $1-\sigma$, *there exists a policy* $\hat{\pi}$ *s.t.* $\mathbb{E}[\ell_{\text{DuGu}}] \leq \epsilon_N + O(1/T)$, *in which* T *is the number of samples for trajectories.*

Proof. Let $Q_t^{\pi'}(s,\pi)$ denote the t-step cost of performing π in state s and then following policy π':

$$Q_t^{\pi'}(s,\pi) = \frac{1}{4}\left((\pi^*(s)-a)^2 + \sum_{\psi=2}^{t}(\pi^*(s_\psi)-\pi'(s_\psi))^2\right) \quad (6)$$

Consider ℓ_{DuGu} is the 0-1 loss, $t \in 1,2,\ldots,T$, we have the following equation for all action a:

$$Q_{T-t+1}^{\pi^*}(s,a) - Q_{T-t+1}^{\pi^*}(s,\pi^*) = \frac{1}{4}(\pi^*(s)-a)^2 \leq 1. \quad (7)$$

From this end, all conditions are satisfied and we can follow a similar proof to [20, 31, 36]. ∎

B SHORT VIDEO UPLOADING SIMULATOR

We now introduce our simulator for the short video uploading task. First, we show the relationship between the video uploading simulator and our proposed omniscient-like emulator in Figure 14(a)–both the simulator and DuGu are run as applications on the emulator. Next, the video uploading simulator can be summarized in the following processes. i) we generate a video chunk with the size of 1 to 8 *MegaBytes*; ii) the video chunk is transferred by our proposed NN-based CC algorithm DuGu; iii) then the transmission process will be slept for a while, basically 0 to 10 seconds. In the meantime, DuGu is changed to the probing phase and detects the network metrics (i.e. min. RTT); iv) finally, DuGu will be changed to the control phase once the video has been generated (i.e., sleep ends).

Here please recall that, we use Pantheon [48] and Mahimahi [32] as the "real transport tools" in the evaluation, as the omniscient-like emulator is only used for training. We formally verify the accurateness of the proposed offline emulator in Figure 15. The graphs show the close correlation of strategy between our emulator and Mahimahi [32] for both metrics.

C TRAINING ALGORITHM

Alg. 1 illustrates the detailed training algorithm of DuGu. First, we randomly pick network traces from the dataset and set various network settings, such as minimum latency, loss rate, and buffer size, as the network environment. Then the agent establishes a virtual session with a virtual sender and receiver. Next, for each time slot t, the agent, running in the given environment, receive a state and takes action a_t w.r.t the policy given by the NN. Once the sender receives the ACK packet, it updates the state and calculates the sending rate and the receive rate iteratively. At the same time, the solver estimates the expert policy and computes the relative gap between the action and the expert. We store the state and the expert policy as the label in the replay buffer. Moreover, we restore a batch of samples from the replay buffer and use samples to update the NN via ℓ_{DuGu}. Finally, we continually produce the process by performing the estimated action a_t till the agent receives the next state. In addition, We modify DuGu's training in the single-agent as training in multi-agents. By default, DuGu adopts 40 forward propagation agents and one central agent.

D WIFI AND 4G EXPERIMENTS

D.1 Public WiFi and Cellular Networks.

Before online A/B testing, we also establish two experiments for WiFi and Cellular networks. Specifically, we first apply all the baselines on a laptop (Ubuntu 20.04, 16G RAM) and treat the laptop as the sender. Next, we connect a public WiFi and evaluate the performance of each scheme. Finally, we use a "4G USB stick adapter" to enable a cellular network and test all the schemes over the unique servers from eight different destinations. Figure 16 reports the comparison results of CC schemes over public WiFi and cellular networks. Note that we normalize the average throughput and delay since each link has different capacities. We can observe that DuGu performs well, always top-3 of the existing schemes. Furthermore, we also see that DuGu can only achieve 60% throughput compared with BBR. It could be another challenge that we aim to solve it in the future.

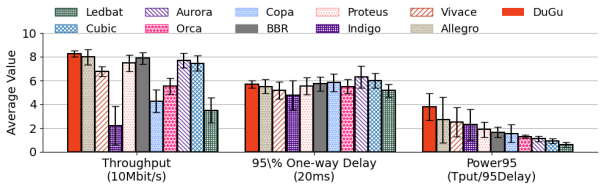
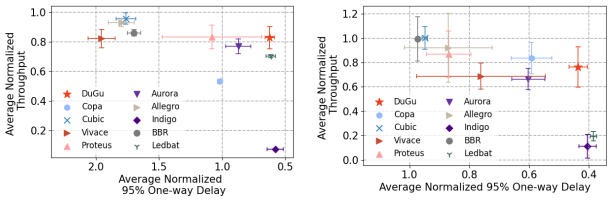


Figure 17: Summarizing all underlying metrics of DuGu and existing schemes.

Algorithm 1 DuGu Overall Training Procedure

Require: NN model θ, `Solver`.

procedure DuGu Training
  Initialize θ.
  Experience Replay $B = \{\}$.
  repeat
    Env ← randomized *trace* and *network settings*.
    t←0
    while time slot **do**
      Get current state s_t from the Env.
      Get a_t according to policy $\pi(s_t;\theta)$.
      # Imitation Learning Here.
      Get expert action $\pi^*(s_t) = \texttt{Solver}(s_t)$.
      Store samples: $B \leftarrow B \bigcup \{s_t, \pi^*(s_t)\}$.
      Sample a batch $\hat{B} \in B$.
      Train θ with $\hat{B}$ using ℓ_{DuGu}
      Perform a_t and receive the new state s_t.
      if done **then** ▷ End of the session.
        break
      t←t+1
  until Converged



(a) Public WiFi (b) Cellular Network

Figure 16: Examining CC algorithms in the public WiFi and Cellular network environment.

D.2 Key Takeaway

In general, we summarize all the experiment results from July 1 to July 28. Figure 16 shows the overall performance for each protocol, in which the results are computed as the average throughput, 95-percentile one-way delay, and power95 metric. The error bar represents the standard deviation for each run. DuGu reaches the highest power95 score among all the schemes. Still, results of Indigo show that a policy that achieves low throughput and extra-low latency can also score a high power95 score. In other words, gaining a higher power95 metric doesn't mean obtaining a better policy. It might demystifies that only optimized strategies towards higher power-based objective function [3, 8, 39, 44] may not be on the right track. Instead, we recommend taking multiple metrics into account, such as throughput, one-way delay, etc.

E RELATED WORK

Congestion control is one of the most fundamental and challenging problems in computer network with more than thirty years. Cubic [18] is a loss-based scheme which a modify version of the linear window growth function. BBR [11] obtains minimum RTT and the maximum bandwidth iteratively, and adjust the pacing rate w.r.t the estimated BDP. Copa [8] is a delay-based approach which uses one-way delay to evaluate the target sending rate. However, heuristics require careful tuning as it is difficult to well-behaved over all considered scenarios. Moreover, with the rapid increase of deep learning technology, the congestion control problem has ushered in a renaissance. Remy [3] is an offline optimization framework for CC and it can surpass many heuristic algorithms under specific networks, however it can't adapt well when meets new network condition. Indigo [48] employs imitation learning to get optimal expert policy, while its expert policy is not so precise. Muses [54] focuses on proposing light-weighted CC algorithms while pay less attention to specific tasks. Moreover, deep reinforcement learning has shown promising performance in complex real-world tasks. For example, Aurora [22] directly adopts the DRL method towards a linear-based reward function. Eagle [16] imitates the mechanism of BBR to learn the phase via DRL. Orca [3] provides another direction to integrate classic and modern using DRL and heuristics, but it's still hard to solve the origin limitation of the heuristics. PCC-Allegro [14] calculates the value of utility function to update the sending rate. PCC-Vivace [15] apply the gradient-based no-regret online optimization to speed up the convergence process. PCC-Proteus [30] adapts to different application scenarios by using a variety of utility functions. However, such *smart* mechanisms are really hard to be learned by reward functions, as the state space is partially observable. Although online-learning methods have made good progress, they are still struggle with performing in low convergence rate. Meanwhile, none of the existing methods well supports the short video uploading task.

F EPILOGUE

It has been almost two years when we finished the first version of DuGu on the subway. We would like to thank every colleagues from Kuaishou who help us improve the work in the last two years, including Bing Yu, Liang Guo, Yangchao Zhao, Yixuan Ban, Dan Yang, Xiaoyi Zhang, Chengyuan Zheng, Yufeng Geng, Shucheng Zhong, Yusong Yang, etc.

While back to the task itself, we believe that it will get more promotion in the MM field. Cross-layer optimization is one of the future directions. For instance, the pre-transcoding setting can be determined w.r.t the network capacity and the users' device ability; the rate control method can leverage I-frame interval information to make the post-processing, placed on the server, more effective. Before finishing such promising approaches, we have to analyze its task characteristics in advance and provide a feasible transport mechanism to solve the fundamental problem. Our work is a constructive step in the short video uploading field.