OnRL: Improving Mobile Video Telephony via Online Reinforcement Learning

Huanhuan Zhang[+], Anfu Zhou[+], Jiamin Lu[+], Ruoxuan Ma[+], Yuhan Hu[+], Cong Li[+], Xinyu Zhang[◇], Huadong Ma[+], Xiaojiang Chen[†]

[+]Beijing Univ. of Posts and Telecom. {zhanghuanhuan, zhouanfu,mhd}@bupt.edu.cn
[◇] University of California San Diego xyzhang@ucsd.edu
[†] Taobao Inc. zhongsheng.cxj@taobao.com

ABSTRACT

Machine learning models, particularly reinforcement learning (RL), have demonstrated great potential in optimizing video streaming applications. However, the state-of-the-art solutions are limited to an *"offline learning"* paradigm, *i.e.*, the RL models are trained in simulators and then are operated in real networks. As a result, they inevitably suffer from the simulation-to-reality gap, showing far less satisfactory performance under real conditions compared with simulated environment. In this work, we close the gap by proposing OnRL, an online RL framework for real-time mobile video telephony. OnRL puts many individual RL agents directly into the video telephony system, which make video bitrate decisions in real-time and evolve their models over time. OnRL then aggregates these agents to form a high-level RL model that can help each individual to react to unseen network conditions. Moreover, OnRL incorporates novel mechanisms to handle the adverse impacts of inherent video traffic dynamics, and to eliminate risks of quality degradation caused by the RL model's exploration attempts. We implement OnRL on a mainstream operational video telephony system, Alibaba Taobao-live. In a month-long evaluation with 543 hours of video sessions from 151 real-world mobile users, OnRL outperforms the prior algorithms significantly, reducing video stalling rate by 14.22% while maintaining similar video quality.

CCS CONCEPTS

• **Networks** → **Transport protocols**; **Mobile networks**; • **Computing methodologies** → **Machine learning**.

KEYWORDS

Video Telephony; Online Learning; Reinforcement Learning

ACM Reference Format:

Huanhuan Zhang, Anfu Zhou, Jiamin Lu, Ruoxuan Ma, Yuhan Hu, Cong Li, Xinyu Zhang, Huadong Ma, Xiaojiang Chen. 2020. OnRL: Improving Mobile Telephony via Online Reinforcement Learning. In *The 26th Annual International Conference on Mobile Computing and Networking (MobiCom'20), September 21-25, 2020, London, United Kingdom.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3372224.3419186

MobiCom '20, September 21–25, 2020, London, United Kingdom

ACM ISBN 978-1-4503-7085-1/20/09…$15.00
https://doi.org/10.1145/3372224.3419186

1 INTRODUCTION

Real-time interactive video communication carries a dominant fraction of today's Internet traffic [5], largely due to many mainstream video telephony applications, such as Facetime, Skype, Google Hangouts, WeChat, *etc.* With the upgrade of the LTE-Advanced and 5G network infrastructures, new use cases are also rapidly emerging, such as live video/VR broadcasting [23, 31], cloud gaming [53, 58], tele-operation of surgery robots or vehicles [39, 59]. Such interactive video applications impose the toughest demand on the network in terms of bandwidth and latency. Although the telecom infrastructure strives to match the demands, it only promises best effort service, and has to rely on the application itself to adapt to the highly dynamic network conditions.

To maintain high quality of experience (QoE), traditional interactive video applications adopt rule based protocols, *e.g.*, congestion control at transport layer and video bitrate adaption on the application. However, the pre-programmed rules fall short of accommodating the highly heterogeneous modern Internet consisting of cellular/WiFi wireless links, cross-continent fiber links, intra-cloud data-center links, *etc.*, all with diverse bandwidth, latency, and buffer capacities. In recent years, data-driven approaches [25, 33, 42, 60, 69] have been explored to improve video QoE. For instance, Pensieve [42] employs reinforcement learning (RL) in video streaming systems to adapt the video bitrate, aiming to mitigate the risk of frame stalling while maximizing bandwidth utilization. Concerto [69] customizes imitation learning to harmonize the transport layer and video codec, so as to reduce stalling in mobile video telephony.

While showing potential, these solutions commonly adopt a *"learning offline, running online"* strategy. The learning models are trained in simulators or emulators, and then the trained models are directly deployed and tested in real applications. Unfortunately, such offline learning models lead to far less satisfactory performance in the real world, despite the high performance achieved in the simulator/emulator [40]. Moreover, they may exhibit even the opposite performance characteristics from those observed in simulations, as validated in [69]. The root cause lies in two interrelated factors: *(i)* It is very challenging to faithfully simulate the complicated real-world Internet dynamics [66]. Individual routers along a network path can bear various capabilities and states, *e.g.*, multiple-flow competition, packet drop strategy, load-induced quality fluctuation, let alone the complicated interaction among multiple protocol layers on end devices. *(ii)* In essence, the capacity of a data-driven algorithm is strictly bounded by its learning environment. Its experience acquired via trial-and-error in a simulator may become stale when coping with the real-world Internet.

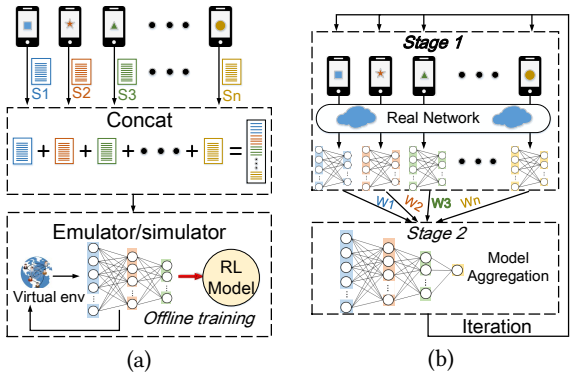


Figure 1: Learning architecture comparison: (a) offline learning (b) online learning.

In this work, we seek to close the simulation-to-reality gap [1, 20] by proposing OnRL, an *online reinforcement learning based adaptation framework for mobile video telephony*. OnRL runs and keeps training directly on an operational video telephony system (*i.e.*, Alibaba Taobao-Live [6, 7, 54]), so as to learn and respond to the real network conditions. OnRL is not merely a straightforward learning environment shift. Instead, it raises three unique design challenges.

(i) Learning from massive concurrent video telephony sessions. In conventional offline learning, one can collect and concatenate network traces from individual user, and feed them into a simulator/emulator to train an RL model [42, 69]. In this way, the RL model can explore diverse environments to enrich its experience, and converge to a universal model with swarm intelligence learned from all users. However, for online learning, a massive number of video telephony sessions run simultaneously, during which the learning algorithm needs to evolve with each session in real time. Therefore, the key challenge lies in how to transform from serial offline learning to parallel online learning while still harnessing the swarm intelligence. To meet the challenge, we propose a two-stage online learning architecture. *Firstly*, we design a new deep reinforcement learning model based on the state-of-the-art PPO algorithm [52], and associate one individualized RL model instance with each user. In this way, each user leads to a different RL model with its own learning experience. *Secondly*, we *aggregate* all users' experiences following a *federated learning* principle, so as to form a high-level model that can react to any network dynamics unseen by individuals. These two stages are coupled iteratively, so as to strike a balance between individualized experience and swarm intelligence.

(ii) Enforcing the RL algorithm under real-time video dynamics. A basic requirement for effective learning is that the learning algorithm's action should be faithfully executed. In our case, once the RL algorithm decides on a sending bitrate x, the video encoder should ideally produce the video stream at exactly the same rate. Though this is straightforward for offline simulation or VoD streaming, the video codec in real-time telephony cannot generate a perfect bitrate of x particularly in short time-scales [28]. Instead, the video codecs have their own control logic dependent on image scene dynamics, compression strategy and even device computing capacities, which results in inherent and substantial video bitrate fluctuation, and a gap between the RL's bitrate decision and the actual video traffic output rate (Sec. 3). To handle the problem, we adapt OnRL's learning algorithm by feeding the gap into the RL neural model. In this way, OnRL can learn the dynamics of the gap and then remedy the impacts by tuning its award operation, automatically. For instance, once it detects a large gap, OnRL will deem the previous RL action as contaminated, and reduce the action's impact by imposing a low weighting factor when computing the corresponding reward from the action.

(iii) Robust hybrid learning. In essence, an RL algorithm learns by following a trial-and-error principle, which risks disrupting the system when applied directly to online learning. Specifically, the algorithm may take incorrect exploitation actions (*e.g.*, selecting very high video bitrates under poor network conditions) that lead to catastrophic effects (*e.g.*, severe congestion and thus low QoE), especially during the early learning phase when the RL model is not well trained yet. To handle the problem, we design a *hybrid learning* mechanism, in which OnRL falls back to the legacy rule based video bitrate algorithm once the RL algorithm is regarded as operating abnormally (*e.g.*, continuously experiencing high loss rate and high latency), and switches back to RL otherwise. To manage such switching, we design an adaptive trend prediction algorithm to discriminate how well one control algorithm performs. Moreover, we augment the RL learning by mapping each such switching event as a penalty into the RL's reward function. In this way, the RL algorithm will learn that it should avoid invoking the legacy protocol as much as possible, and evolve to be a self-dependent and robust video bitrate adaptation algorithm.

We have implemented and deployed OnRL in Alibaba Taobao Live, a mainstream video telephony system[1]. We evaluate OnRL with 543 hours of Taobao Live sessions involving 151 beta users worldwide during one-month tests. The results confirm the advantages of online learning, which brings remarkable QoE improvement, *i.e.*, reducing video stalling time by 14.22%, while maintaining nearly equal video bitrate. Furthermore, our controlled experiments validate the individual design choices in OnRL and explain their effectiveness through micro-benchmarks.

Contributions: To our knowledge, OnRL represents the first online learning driven video telephony solution deployed in a large commercial system. OnRL resolves general problems that arise when RL meets real-time applications over the highly dynamic and heterogeneous mobile Internet. The specific contributions of OnRL include:

(i) We propose a two-stage iterative RL algorithm that enables online learning directly from massive concurrent video telephony sessions, instead of conventional simulation/emulation based offline learning (Sec. 2).

(ii) We design novel mechanisms to mitigate the inaccrate execution of actions when applying RL to real-time video applications (Sec. 3), and to boost the robustness of online learning (Sec. 4).

(iii) We implement and deploy OnRL on a mainstream commercial video telephony platform (Sec. 5) and validate its remarkable performance gain over the state-of-the-art solutions (Sec. 6).

[1]Taobao-live is part of the Taobao app–the 4th most popular mobile app in China with 666 million monthly active users [6, 7]. Over 1 million new users joined Taobao-Live during the single month of February 2020 [55].

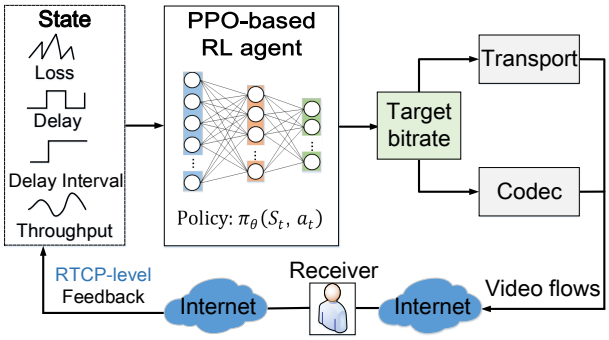


Figure 2: The workflow of OnRL individual learning.

2 PARALLEL ONLINE LEARNING: ARCHITECTURE AND ALGORITHM

2.1 Iterative Two-Stage Architecture

Existing machine learning driven video transport systems all adopt the "offline learning" approach [42, 60, 69] as illustrated in Fig. 1 (a). Such a system needs to first collect plethora of simulated [42, 60, 69] or real network traces [40, 65] from different users. Then these traces are serialized, *i.e.*, concatenated regardless of their inherent personal/temporal characteristics, and then replayed by a custom built simulator/emulator to train a neural network model (usually an RL model). Finally, the trained model is distributed to each user, who executes the learned bitrate adaptation strategy at run time.

However, such offline learning cannot deliver expected performance improvement due to the simulation-to-reality gap as reported in [40, 65, 66] and also corroborated in Sec. 6.2. Moreover, the serialized learning aggravates the gap: *(i)* The uniform trace serialization blurs the users' individual characteristics, *i.e.*, each user will get the same training experience, which loses the possibility of personalized optimization. *(ii)* Once the RL model is offline trained, it freezes and stops learning in the real-world. Therefore, it is unable to deal with new network conditions unseen before.

In contrast, OnRL adopts a fundamentally different design principle of learning at runtime. To realize the principle, we design *an iterative two-stage learning architecture* (Fig. 1(b)), which operates as follows, *(i) Stage-1: individualized learning*. Each user separately learns its own RL model at run time, based on a state-of-the-art PPO RL algorithm [52] with customized design for real-time video transport (Sec. 2.2). In this way, each user possesses an individualized model incorporating its personalized learning experiences. *(ii) Stage-2: Learning aggregation*. The individualized models are then fed to a backend server, and aggregated to generate a universal model. The aggregated model employs a weighted averaging over each model's neural model parameters following the principle of federated learning [44] (details in Sec. 2.3). From a high level, the aggregated model has experienced enough network variants from different users, and thus acquires the desired capability of coping with network dynamics.

The two stages operate iteratively. Specifically, the aggregated model will be distributed to all users, and each user, upon a new video telephony session, will start out with the latest aggregated model, and keep on learning from the new session. The procedure leads to a new individual model, which is fused at the back-end server for a new round of learning aggregation. The continuous iterations aim to strike a balance between each user's personalized experience and the swarm experience from all users. It is noteworthy that a mature aggregated model can be applied to a new user immediately, so as to eliminate the risk of low QoE due to random exploration at the bootstrapping phase.

We now proceed to elaborate on the individual learning algorithm and the learning aggregation, respectively.

2.2 Individualized Online Learning

We design a PPO-based RL algorithm for individualized online learning, as shown in Fig. 2. Specifically, the RL agent keeps observing the instantaneous network *state* S_t (*i.e.*, packet loss, delay, delay interval and throughput) at any time t, and deciding an *action* a_t (*i.e.*, a bitrate), which is expected to match currently available network bandwidth. Then the action a_t will be enforced on both the video codec and the transport layer protocol, which generates and consumes the video traffic at the rate a_t, respectively. The traffic, after going through the network path, will produce a new *state* S_{t+1}, which initiates a new round of RL *action*.

The mapping of $S_t \rightarrow a_t$, the key function of RL agent, is decided by the RL's control policy π_{S_t, a_t}, which is learned in the above process. Intuitively, if the RL agent produces a bitrate exceeding the available bandwidth, it will incur network congestion. The consequence will be reflected in the next state S_{t+1}, most likely with high packet loss or large delay. By observing the change from S_t to S_{t+1}, the RL agent can realize that it has made an inappropriate action a_t, so it needs to update the policy π to generate a more conservative bitrate when observing S_t or similar state next time.

Underlying the workflow above, we identify two requirements: *(i)* The RL agent should be very agile, *i.e.*, responding to network dynamics at video frame-level granularity, corresponding to a timescale of dozens of milliseconds. *(ii)* The RL agent should refrain from jumpy actions which hurt video perception quality and QoE [35]. To meet the requirements, OnRL first adopts a batch-level policy update running in the background at second level granularity, while executing agile response using the latest model at millisecond-level granularity. Secondly, OnRL guarantees smooth bitrate adaptation by utilizing a relatively stable policy adjustment mechanism. We now present OnRL's RL model and training methodology.

2.2.1 OnRL's PPO model. Fig. 3 depicts OnRL's RL model, including state, action, and the neural network architecture.

State. The state, also the input of the RL agent, at any time t, is denoted with $S_t = (\vec{l_t}, \vec{d_t}, \vec{i_t}, t_t)$, which represents the packet loss, delay, delay interval (*i.e.*, the difference between the arriving interval of two consecutive RTP packets at the receiver side and the corresponding departure interval at the sender side [22].), and the receiver-side throughput. These inputs can be easily derived at the sender side at a fine-grained timescale of around 50 ms, based on the periodic RTCP ACK message in WebRTC [3]– the transport/application layer protocol stack used in mainstream interactive video services.

Action. In each RTCP interval, OnRL maps S_t to a compact action space $\mathcal{A}: \{0.1Mbps, 0.2Mbps, \cdots, 2.5Mbps\}$, representing the target output bitrate of video codec. After the video traffic is transmitted through the network, OnRL will immediately transform to a new state S_{t+1}, and generate a new action a_{t+1}. Through such continuous iterations, OnRL will learn to cope with network dynamics.

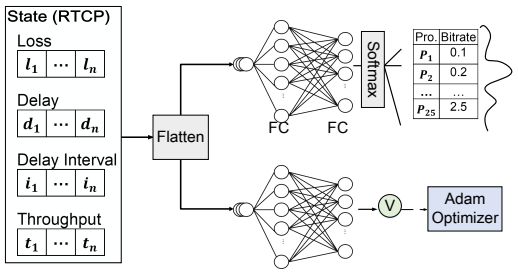


Figure 3: OnRL's RL model.

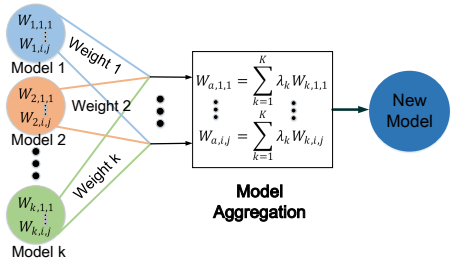


Figure 4: The workflow of learning aggregation.

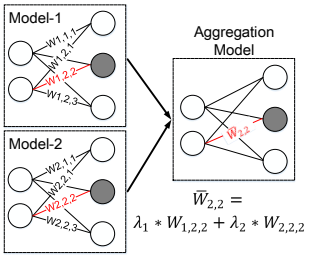


Figure 5: A showcase of the weighted models.

Reward. To ensure that OnRL can learn from past experience, each action is associated with a reward, which is the video QoE of previous time interval defined as follows,

$$r_t = \alpha \times \sum_{n=1}^{N} q_n - \beta \times \sum_{n=1}^{N} l_n - \eta \times \sum_{n=1}^{N} d_n - \varphi \times \sum_{n=1}^{N-1} \left| q_n - q_{n-1} \right| \quad (1)$$

where N represents the number of RTP packets in a state, q_n is the throughput measured at the receiver side which is directly related to the video quality, l_n and d_n are the packet loss rate and delay at the transport layer. The final term is used to enforce video smoothness by penalizing large bitrate fluctuations. These metrics are weighted by different impacting factors $\alpha, \beta, \eta, \varphi$, and then summed together. These hyper-parameters have significant impacts on training effectiveness. Clearly, the four metrics have different magnitudes. The value of q_n and the final item usually falls within 0.1~4.0 Mbps. l_n is commonly less than 10% in practical video telephony systems, while d_n ranges from dozens of milliseconds to hundreds of milliseconds depending on the end-to-end path length. To obtain a meaningful r_t, these metrics need to be normalized to a consistent range by adjusting values of the parameters $\alpha, \beta, \eta, \varphi$. In OnRL, the scope of the parameters is limited to 30~100, 50~100, 5~25, and 10~50, respectively. In the actual deployment of OnRL, they are empirically set to 50, 50, 10, 30, respectively.

Neural network (NN) architecture. OnRL uses a NN architecture to represent the policy π with a set of parameters θ. It adopts the simple fully-connected (FC) layer structures to distill implicit features hidden in different input elements. More specifically, OnRL first flattens the sequences $(\vec{l}_t, \vec{d}_t, \vec{i}_t, t_t)$, and then feeds them into two similar NNs, one is used for feature extraction and then outputs the bitrate action, and the other is served to judge the overall objectives like a critic. Each NN contains two FC layers, with 64 and 32 elements, respectively. We have empirically verified that flattening the input sequences performs better in terms of bandwidth prediction accuracy than feeding each state to a separate NN structure. In addition, each layer employs the tanh activation function [24], which shows more accurate bandwidth prediction capabilities than traditional relu or softmax activation. Besides FC, we have also tried CNN and long-short-term-memory (LSTM) for feature extraction, but their performance is not as good. Further analysis shows that the CNN architecture specializes in extracting image features that consist of complex spatial information, which does not exist in OnRL's state space. As for LSTM, it is more useful for reasoning the time-series data that takes into account long historic impact, but the performance of video telephony is more dependent on instantaneous network conditions.

2.2.2 Training Methodology. The objective of OnRL's training is to maximize the total cumulative rewards: $\mathbb{R}_\tau = \sum_{t'=t}^{T_\vartheta} \gamma^{t'-t} r_{t'}^{\vartheta}$, where γ servers as a discounting factor, usually customized as 0.99 or 0.9. The gradient policy updates for maximizing $\mathbb{R}_\tau$ can be formulated as follows:

$$\nabla \bar{\mathbb{R}}_\tau = \frac{1}{\Theta} \sum_{\vartheta=1}^{\Theta} \sum_{t=1}^{T_\vartheta} (\hat{A}(a_t^\vartheta, s_t^\vartheta)) \nabla log p_\theta(a_t^\vartheta, s_t^\vartheta) \quad (2)$$

where T_ϑ stands for the batch size for gradient policy updates, and Θ represents the number of batches. $\hat{A}(a_t^\vartheta, s_t^\vartheta) = \mathbb{R}_\tau - b$, (short as $\hat{A}$, $b \approx E[\mathbb{R}_\tau]$), represents the *advantage function* that expresses the difference in expected reward decided by practical action from state, compared with the averaged expected reward from policy π. Intuitively, the advantage function can show the extra benefit of a certain action over the average action. Recall that traditional policy gradient [48] may have large or little adjustment between contiguous policy, due to the uncertain learning rate value. This uncertainty can bring catastrophic QoE and difficult convergence in real-time video especially for the real-world learning of OnRL. To avoid this, we design two customized modules to maintain the online learning performance:

(i) Batch-level updates instead of instance-level (a single input) updates. As noted before, OnRL needs to respond to each input instance to adapt to the real-time bandwidth variation. Usually, the response also leads to an update of neural network parameters. However, such instance-level updates will largely increase the learning time and in turn slow down the response. We thus customize a *batch update policy*. In particular, the learning agent buffers the recent ⟨state, action, reward⟩ records. Only when the buffer is more than a batch-size, the agent will feed the buffer to the policy network to update the neural network parameters. In this way, the agent can execute fine-grained response in real time, while simultaneously running the online learning.

(ii) Smoothing-level updates. OnRL utilizes the loss function about gradient updates as follows,

$$L(\theta) = \hat{\mathbb{E}}[min(\frac{p_\theta(s,a)}{p_{\theta_{old}}(s,a)}\hat{A}, clip(\frac{p_\theta(s,a)}{p_{\theta_{old}}(s,a)}, 1-\varepsilon, 1+\varepsilon)\hat{A})] \quad (3)$$

where we bound the difference between a new policy and an old one, by taking into account their probability ratio $\frac{p_\theta(s,a)}{p_{\theta_{old}}(s,a)}$. Specifically, we clip the ratio and thus remove the incentive from of the loss function $L(\theta)$, when $\frac{p_\theta(s,a)}{p_{\theta_{old}}(s,a)}$ is beyond the interval [1-ε, 1+ε] (here ε is a hyper-parameter set to 0.2). In this way, OnRL will update a learning policy smoothly, and in consequence avoids too jumpy bitrate decisions, so as to enhance bitrate smoothness for better QoE.

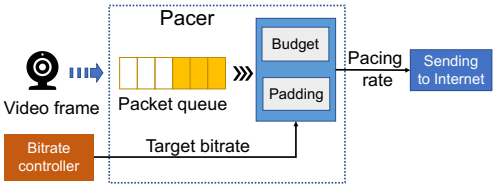


Figure 6: The pacing mechanism.

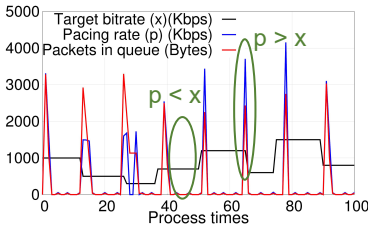


Figure 7: Action deviation.

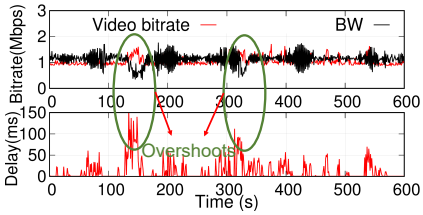


Figure 8: RL's bitrate overshoots.

Neural network implementation parameters. We use TensorFlow [13] to implement the training workflow and TFlearn [14] to construct the NN architecture. OnRL takes the fine-grained exploration in RTCP-level, and the batch size for batch training is 32 in our design. It is noteworthy that we have tried the common practice of batch size adjustment, *e.g.*, 16, 32 and 64, and did not observe substantial change on the QoE-related metric. We employ Adam optimizer [37] to update the stochastic gradient descent.

2.3 Learning Aggregation

The learning aggregation is inspired by two key observations from our analysis of the Taobao-Live traces: *(i)* Different users exhibit different levels of network dynamics. For instance, some users typically initialize video sessions at home (*e.g.*, make-up tutors or in-store shopping guides), with stable WiFi connections. Correspondingly, the PPO learning algorithm tends to generate stable video bitrate decisions. In contrast, other client users (*e.g.*, outdoor travelers) usually undergo fluctuant cellular network conditions, due to mobility or handoff. Accordingly, the learning algorithm will learn to change its video bitrates frequently to match the instantaneous network variations. *(ii)* While a user usually has relatively consistent network conditions, she may change the daily routines, thus encountering new network dynamics. In this case, an RL model purely trained from her own previous network conditions will react inappropriately.

To strike a balance between individual and swarm experience, we propose a *weighted model aggregation* method. Recent works [29, 44] have shown that, averaging the parameters of the same neuron across many NN models can achieve similar effect of aggregating these models' experience. We thus utilize the strong feature representation ability of NN parameters to realize the model aggregation, as shown in Fig. 4.

Specifically, suppose we have a total number of K users, and each user k's neural model can be denoted with a matrix $W_{k,i,j}$, where each element of $W_{k,i,j}$ is the parameter of a neuron in the neural model (i represents the i-th layer of the NN, and j is the j_{th} neural cell in each layer). Note that the dimensions of all matrices $W_{k,i,j}(k \in [1,K])$ are the same, since the NN architecture across all users is the same. To fuse the individual models, we perform a weighted averaging operation following the principle of federated learning [44]:

$$\overline{W} = \sum_{k=1}^{K} \sum_{i=1}^{I} \sum_{j=1}^{J} \lambda_k W_{k,i,j} \tag{4}$$

where $\overline{W}$ represents the aggregated model, and λ_k is the weight of user k's model. We provide an example in Fig. 5 to illustrate the model averaging process. In particular, the weights $\overline{W}_{2,2}$ of the dark neural cell is computed from the corresponding neuron position of two individual models: model-1 and model-2.

We have two ways to determine the value of each λ_k:

(i) Average aggregation. For a newly joined user (*i.e.*, a user without any prior experience), we generate and apply a model with average experience from all users, *i.e.*, we let $\lambda_k = \frac{1}{K}$ for $k \in [1,K]$.

(ii) Prioritized aggregation. For other users, we prioritize a user's own weight thus it can achieve optimal performance under its typical network conditions, while reacting appropriately when confronting unusual conditions. Specifically, for each user k, we let $\lambda_k = p$ ($p \in [0,1]$), and $\lambda_m = \frac{1-p}{K-1}, \forall m \neq k$. In Sec. 6, we experimentally evaluate the impact of weight parameter of p.

3 ENFORCING RL ACTIONS IN REAL-TIME VIDEO TELEPHONY

Understanding action deviation. In a practical video telephony system such as WebRTC, a single video frame often needs to instantly burst through the network. To avoid transient congestion, a *pacer* mechanism is introduced to evenly distribute the video data across continuous time slices. A simplified pacing workflow is illustrated in Fig. 6. When a video frame is produced, it is fed into a pacer queue. The pacer splits the video frame into groups of packets and progressively injects them into the network path every Δ interval (usually 5ms). The pacer has its own control logic (*e.g.*, budget and padding) to send these packets based on the instantaneous target bitrate (denoted as x), which is generated by congestion control algorithms (*e.g.*, GCC in WebRTC). Surprisingly, by analyzing the behavior of the default WebRTC pacer used in Taobao-Live, we find that the pacer's actual sending rate still deviates wildly from the target x. Our analysis reveals two reasons that inevitably cause the gaps.

(i) The pacer occasionally needs to increase the sending bitrate to accelerate its buffer flushing. The pacer queue has its own control logic which depends on image scene dynamics, compression strategy and even end devices' computing capacities. On the other hand, the video traffic is quite bursty, *e.g.*, a key-frame may comprise excessive data when the video scene changes quickly. In such a case, if the pacing rate strictly executes x, it will cause long latency in the pacer queue, which may lead to a detrimental freezing effect for real-time video. Hence, the pacing rate will be recalculated by multiplying a factor (typically around 2.5 in WebRTC) to accelerate the buffer clearance. This means that *RL's action (x) cannot be strictly executed by the interactive video system, which compromises its effectiveness and may eventually mislead its policy update.*

(ii) The pacer sometimes has insufficient video data to meet the target bitrate x, *e.g.*, given relatively static video scenes. Whereas the padding logic inside the pacer can artificially add some dummy data packets, we find that such a mechanism seldom works for real-time video. The root cause is that the dummy packets bring more

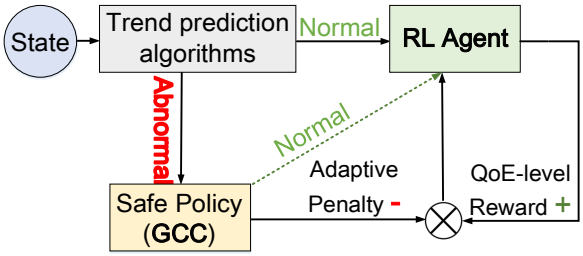


Figure 9: Hybrid learning mechanism of OnRL.

traffic and worsen the bandwidth cost. Besides, the padding data tend to increase the latency of actual video packets and hence lower the QoE. As a result, the operational video telephony services like Taobao-Live are usually reluctant to use padding, and the target bitrate of being commanded by RL model cannot be accurately enforced.

We showcase the action deviation by logging and plotting a segment of the target bitrate, pacing rate, and queue size in the pacing buffer sampled from a video session, in Fig. 7. We can find that, when the pacer queue accumulates many packets (*e.g.*, larger than 1000 bytes whenever key-frames appear), the pacer will execute a sending bitrate p higher than the target bitrate x to accelerate the buffer flushing. In contrast, when the queue is almost empty, the actual sending rates usually drop to zero due to the absence of padding.

Learning to tolerate the action deviation. To overcome the aforementioned problem, we further customize OnRL's RL model to implicitly learn the gap between the target bitrate and actual video traffic. We realize this by simply feeding the gap into the RL neural model. Specially, we define the gap as $g_t = x_t - p_t$, where x_t stands for the target rate of OnRL's RL action at time t, and p_t is the pacing rate. Then the state/input in Sec. 2.2 is updated to $S'_t = (\vec{l_t}, \vec{d_t}, \vec{i_t}, t_t, g_t)$. In this way, OnRL can learn the dynamics of the gap and then remedy the impacts by tuning its reward operation automatically. Once it detects a large gap (*e.g.*, $|g_t| > 0.5$ Mbps), OnRL will deem the previous RL action as corrupted, and reduce the action's impact by imposing a low weighting factor (default 0.5 in our design) in the cumulative reward (Sec. 2.2.2). Sec. 6.4 verifies the effectiveness of this mechanism compared with the gap-oblivious RL model.

4 ROBUST HYBRID LEARNING

In this section, we describe the hybrid learning policy to ensure the reliability of RL when it is trained online subject to real network dynamics. Typical RL models learn in a trial-and-error manner, whereas at run time, any erroneous action may lead to severe QoE degradation for the interactive video. We provide a showcase in Fig. 8. The video bitrate, packet delay, and ground truth network bandwidth are sampled from a controlled experiment on a local testbed (the testbed implementation will be introduced in Sec. 5). Within a period of 10 minutes, there exist 8 instances where the RL-generated bitrate overshoots the available bandwidth, causing latency spikes. Ideally, the RL algorithm should learn from these mistakes. But unlike simulation based training, accumulating experience through failures is unacceptable for real-time video. This is likely to happen particularly at the beginning of a video session, where the RL model may need to keep exploiting and failing, so as to materialize its policy.

On the other hand, traditional rule-based video transport control algorithms, such as GCC [22], tend to make conservative decisions. So a straightforward solution that ensures the reliability of online learning is to combine the rule-based algorithm and the RL algorithm, *i.e.*, switching to a conservative algorithm once RL's decision becomes too aggressive. However, this compromises RL's learning capacity, *i.e.*, the RL's learning experience is interrupted and it becomes oblivious of the network state changes during its absence period. As a result, RL cannot react appropriately in the future. Therefore, the key question is: *How to suppress the RL algorithm's catastrophic exploration actions while preserving its learning ability at the same time?*

To meet this challenge, we design a *hybrid leaning mechanism* that integrates the RL and conservative algorithm in a coupled close-loop. Intuitively, we do not merely regard the rule-based algorithm (*i.e.*, GCC in our design) as a protective backup, but treat it as a "teacher" for the RL algorithm and provide useful feedback to prevent inappropriate actions. We believe that the hybrid complexity is critical for ensuring robustness. In particular, since OnRL runs in an operational video telephony system, any erroneous action caused by trial-and-error policies may lead to catastrophic degradation of the system performance at run time.

Fig. 9 illustrates the workflow of hybrid learning scheme, with the following key elements. *(i)* Different from the pure RL learning (Sec. 2.2), we introduce a safety condition detector to evaluate whether the RL algorithm functions properly, by examining the current input state. *(ii)* If so, the input state goes to the RL agent, who will be in charge of bitrate adaptation following the process in Sec. 2.2. *(iii)* Otherwise, GCC will take over. Remarkably, there will be a penalty whenever GCC is invoked, which is incorporated into the RL's reward function. In this way, OnRL can still learn implicitly even when GCC is running. *(iv)* OnRL will roll back to RL once the detector decides that the possibility of RL causing QoE damage is low. Note that the hybrid switches will mostly happen in the early stage of online learning. After the penalty feedback becomes effective beyond the early stage, the OnRL agent will learn to converge to appropriate actions and switch to GCC as less as possible. We now proceed to detail the design of the safety condition detector and penalty feedback in the hybrid learning framework.

Safety condition detector. Due to the elusive network conditions, it is non-trivial to detect whether the learning agent has made an inaccurate action. To enable early detection while suppressing false positives, we design a delay-based filter to predict video QoE damage, which is inspired by the delay based congestion control mechanism in [22, 61].

The basic idea is to detect whether recently received latency sequence is showing a rising trend. If so, the filter will command OnRL to switch from RL to GCC. Specifically, we employ the inter-packet interval $\Delta d(t_i)$ sequences to estimate the queuing build-up trend on the network path, defined as follows:

$$\Delta d(t_i) = (a_i - a_{i-1}) - (s_i - s_{i-1}) \tag{5}$$

where s_i and a_i denote the sending and receiving timestamp of the i-th RTP packet. At time T, the possibility of QoE degradation in the near future can be expressed by the historical sequences: $\{\Delta d(t_0), \Delta d(t_1), ..., \Delta d(t_T)\}$. We define the *possibility metric* by an

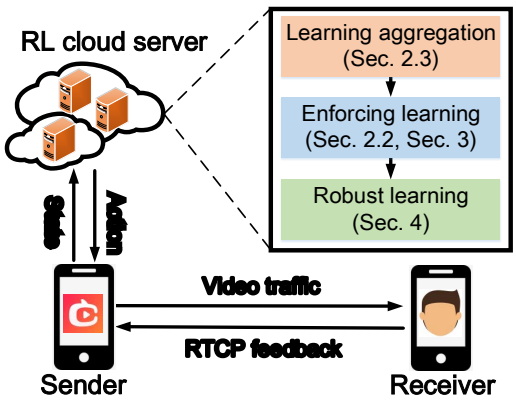


Figure 10: System implementation of OnRL.

exponential weighted average value $D(t_i)$, as follows,

$$D(t_i) = \sum_{i=1}^{T} 2^{-i} \times \Delta d(t_{T-i}) \tag{6}$$

where the closer time a delay condition, the more important weight it has in $D(t_i)$. Note that the filter computes $D(t)$ using a sliding window along t, and once $D(t_i)$ exceeds a threshold $\gamma(t)$, the filter predicts a high risk of QoE degradation, and then switches to GCC. Since GCC is latency-sensitive, its control policy will immediately decrease current sending bitrate, thereby mitigating the risky situation. On the other hand, once the condition returns to be safe (*i.e.*, no trend of latency growth), the RL policy will take control.

One challenge here is to determine the threshold $\gamma(t)$. We note that a constant threshold cannot handle inherent network heterogeneity and dynamics. So we design an adaptive threshold $\gamma(t_i)$ which evolves with the latency sequences:

$$\gamma(t_i) = \gamma(t_{i-1}) + k_\gamma \times (|D(t_i)| - \gamma(t_{i-1})) \tag{7}$$

Eq. (7) is a function of the adaptive threshold $\gamma(t_i)$, which evolves with the sliding window of latency. Specifically, k_γ is a constant to determine the ratio of $\gamma(t_i)$'s increase/decrease. Intuitively, once the exponential weighted average value $D(t_i)$ becomes larger within last threshold, the dynamic threshold $\gamma(t_i)$ tends to be decreased, and consequently OnRL has a larger possibility to evoke GCC.

Switching penalty for RL. We leverage the switch between RL and GCC as a feedback to augment the RL model. Each switching event is configured as an extra penalty into the agent's reward function. In this way, it will learn to act appropriately and thus switch to GCC as less as possible in the future. Specifically, we design an adaptive penalty parameter η' to replace the default η in the standard RL award in Eq. (1):

$$\eta' = \eta \times 2^{\epsilon}, subject\ to\ \epsilon = latency/10 \tag{8}$$

Under the η' regulation, the larger latency that the RL action triggers, the larger penalty will get in the overall process. We will validate the design of the entire hybrid learning scheme in Sec. 6.

5 IMPLEMENTATION

Implementation on Taobao-Live. We implement OnRL based on the operational Taobao-Live video telephony system and release it as a beta app to users. Taobao-Live builds on WebRTC [3], a real-time video communication framework with built-in support of video codec and transport-layer protocol (*i.e.*, GCC). WebRTC allows flexible re-implementation of the video control algorithms, and has been used in the state-of-the-art transport and video application studies, such as BBR [21], Salsify [28] and Concerto [69], *etc.*

Our OnRL implementation essentially replaces the existing bitrate control module in Taobao-Live. Ideally, OnRL's components should be implemented inside the Taobao-Live app. However, due to the lack of API support for training RL neural networks on mobile devices [57], we implement OnRL via a cloud-assisted framework, as shown in Fig. 10. Besides the pair of video telephony's sender and receiver, we introduce an RL server, on which we implement the three design components of OnRL (*i.e.*, the two-stage iterative learning, coordination learning and robust learning) based on Tensorflow [13]. During each telephony session, the sender maintains a connection and exchanges information with the RL server. It collects RTCP feedbacks (*i.e.*, packet loss, delay, throughput, *etc.*) from the receiver and sends them to the RL server as the input of OnRL. Then OnRL processes the input and returns an action (*i.e.*, the target video bitrate) to the sender, which is then executed by Taobao-Live. Meanwhile, the OnRL module on the server periodically updates its control policy to realize online learning. We found that the information exchange latency from the sender to the server is only about 10 ms in most cases, which has negligible impacts on video telephony as will be validated in Sec. 6.3.

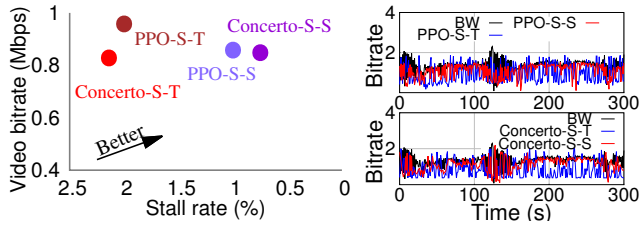
To handle some extreme events (*e.g.*, possible connection failure between the sender and RL server, or when OnRL consistently leads to low QoE), we also implement a fallback mechanism on the sender, *i.e.*, the sender automatically downgrades to the default WebRTC controller. In addition, we deploy a separate back-end server to perform learning aggregation at the frequency of once per day (typically in early morning with the least user activity).

RL server. Each RL server is a COTS PC equipped with 32 kernels, 2.5-GHz CPU and 65-GB memory, and runs in RedHat Linux 4.8.5-11. We adopt the Tensorflow version 1.15.2 to host OnRL's neural network, and each server can accommodate at least 50 concurrent users, due to the efficient neural network structure of OnRL with a small size of *32KB* and runtime memory requirement of 0.5 GB. Currently, we have deployed 5 RL servers for our beta testing and OnRL's high *scalability* is guaranteed using the Alibaba cloud management system Apsara [4], which has the capacity of efficient resource scheduling and load balancing across millions of servers [8].

Simulator and local testbed. Besides the operational app, we have also developed a trace-driven simulator similar to the one used in existing work [69]. We also follow [69] to build a local video telephony testbed, comprised of a client and server, whose end-to-end network bandwidth follows the traces of which we collect from Taobao-Live sessions and is enforced by Linux tc [11]. Unlike the operational app, these platforms allow for controlled experiments with known bandwidth ground-truth, which facilitates deeper diagnosis of the results.

6 EVALUATION

In this section, we evaluate OnRL from three aspects: *(i)* We validate the necessity of online learning by comparing the performance of models under different algorithms and operating environments including simulation, testbed and real networks (Sec. 6.2). *(ii)* We deploy and evaluate OnRL in the operational Taobao-Live with a real-world user base (Sec. 6.3). *(iii)* We evaluate each design module



(a) QoE metrics. (b) A showcase.

Figure 11: Performance of simulation-trained models under simulator and testbed, respectively.

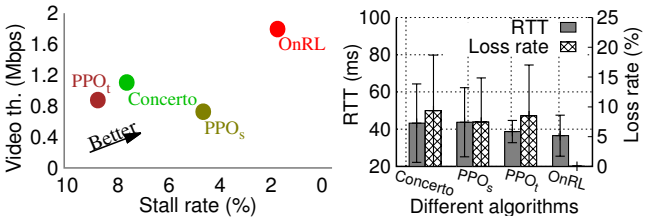


Figure 12: Performance comparison of different models running over Taobao-Live.

inside OnRL separately to gain an in-depth microscopic understanding of online learning (Sec. 6.4).

6.1 Methodology

Evaluation Metrics. We employ both application-layer and transport-layer metrics for a comprehensive evaluation, including: *(i)* Video throughput and Peak Signal to Noise Ratio (PSRN) that characterize the perceived quality of the video frames [32]. Note that our experiments are performed in both controlled testbed and the operational Taobao-Live system. Whereas PSNR can be easily measured on the testbed, it is unavailable in the Taobao-Live system. As a remedy, we use the receiver-side video throughput as a metric to reflect the video quality. *(ii)* Stall rate to measure the fluency of the video telephony session. In the Taobao-Live system, a stalling happens whenever the packet RTT exceeds 300ms [2]. Here we remark that the stall rate indicates how frequent video freezing occurs, which is different from stalling duration. In the simulator/testbed in-lab environments, the RTT hardly rises to 300ms, so a stalling event is considered to occur when the instantaneous video frame rate drops below 12 fps, as suggested by DevOps engineers of Taobao-Live. *(iii)* We also examine the transport layer metrics including packet loss rate and RTT during video telephony, which help to understand OnRL's behaviors in-depth and explain the application-layer performance.

Baseline methods. We compare OnRL against the following algorithms which represent the state-of-the-art in video telephony: *(i)* Concerto [69], which utilizes deep imitation learning [50] to optimize video QoE. As a semi-supervised algorithm, Concerto is trained in a simulator with known ground-truth network bandwidth. The trained model is then deployed and tested in real applications. In this work, we run Concerto using its original model [69] which is trained over 1 million real-world video telephony sessions unless otherwise stated. *(ii)* WebRTC [3], which is the most popular rule-based video telephony framework. WebRTC has been incorporated into Google Chrome [9] and mainstream real-time video apps, including Google Hangout, Facebook messenger, Amazon Chime and Taobao-Live.

6.2 The Need for Online Learning

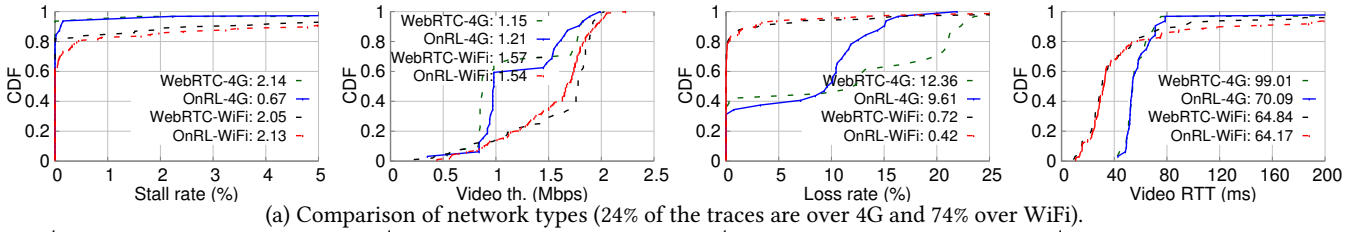
The gap between simulation and testbed. In order to validate the limitations of "offline learning", we first run a micro-benchmark to compare the performance of simulation-trained models, *i.e.*, Concerto and the basic PPO model described in Sec. 2.2. To ensure fairness, we use the same 3-hour trace in two different testing environments: the simulator and local testbed. The resulting video bitrates and stall rates are presented in Fig. 11(a), which confirms that Concerto and PPO work well when the testing is done in the same environment as the training (*i.e.*, the cases marked as Concerto-S-S and PPO-S-S). However, the performance drops significantly after the testing environment is changed to be the testbed (*i.e.*, Concerto-S-T and PPO-S-T). Specifically, their stall rates increase by 64.9% and 51.2% respectively, with minor video bitrate deviation (between 0.02 Mbps and 0.1 Mbps). These results clearly validate the necessity of online learning. To understand the performance drop, we plot a 300-second segment of the models' actions (*i.e.*, video bitrates) against the ground-truth bandwidth in Fig. 11(b). We observe that both Concerto and PPO in simulation environment can follow the bandwidth very closely despite high dynamics, with average deviations of only 17.4% and 13.6%, respectively. In contrast, the deviation increases to 30.3% and 37.12%, when coping with the same traces in the testbed environment. *To summarize, once the deployment environment differs from the training environment, the offline RL models' experience becomes stale, leading to unsatisfactory performance.*

The gap between simulation/testbed and real-world network conditions. We further examine the limitations of "offline learning" under real-world network dynamics. In addition to OnRL, we intentionally integrate three offline-trained models into the Taobao-Live system, including the Concerto and PPO models trained in simulator (referred to as PPO_s), and the PPO model trained in testbed (referred to as PPO_t). We then run the models on a randomly selected Taobao-Live app user. Note that the offline-trained models all showed satisfactory video QoE in their own training environment as validated above. We plot the average video throughput, stall rate, packet rtt and loss rate in Fig. 12, over a random set of video sessions lasting 10 hours in total. We find that OnRL achieves the most compelling video QoE. In particular, it outperforms the most competent offline-trained scheme Concerto by 31.9% in terms of video throughput, and leads to a remarkable 78.3% reduction in stall rate, which again corroborates the benefits of online learning mechanism. On the other hand, PPO_t and PPO_s exhibit large QoE gaps on both metrics. Accordingly, OnRL also shows the smallest packet delays and loss rates, which confirms its ability to cope with network dynamics. *The result further validates the necessity of online learning for optimizing real-time interactive video applications under real network conditions.*

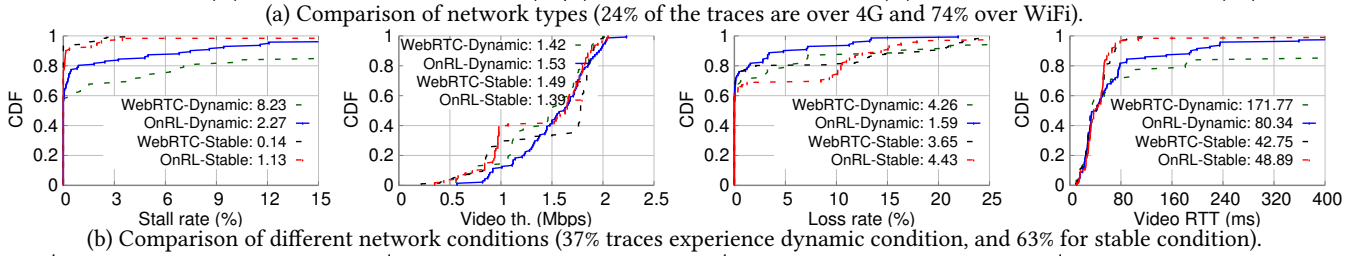
In particular, we find that PPO_t, which is equipped with the same pacing mechanism with OnRL in Taobao-Live, performs worse (*e.g.*, 6.88% lower stall rate and 0.75 Mbps lower throughput) than OnRL, which demonstrates that the online learning, instead of the pacing mechanism, is the reason for performance improvement.

6.3 System-level evaluation in the wild

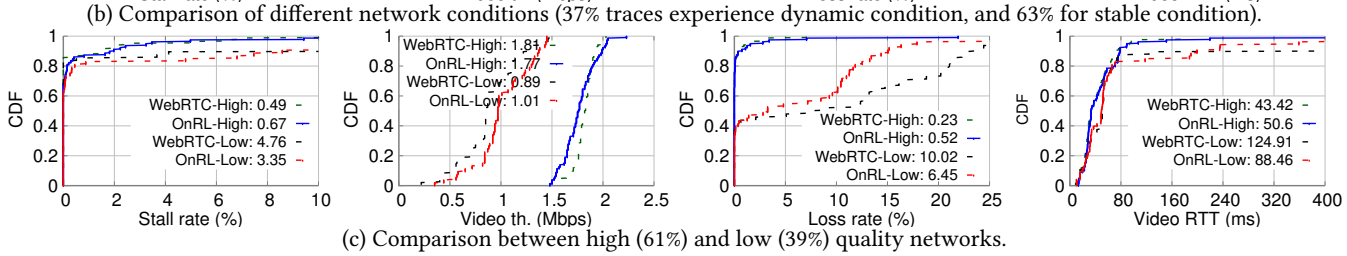
We distribute a beta version of Taobao-Live equipped with OnRL



(a) Comparison of network types (24% of the traces are over 4G and 74% over WiFi).



(b) Comparison of different network conditions (37% traces experience dynamic condition, and 63% for stable condition).



(c) Comparison between high (61%) and low (39%) quality networks.

Figure 13: OnRL's performance while running in Taobao-Live: a breakdown analysis.

Table 1: Average statistics from Taobao-Live.

Metrics	Stall rate (%)	Video th. (Mbps)	Pkt loss (%)	Rtt (ms)
WebRTC	2.04	**1.48**	3.79	73.05
OnRL	**1.75**	1.46	**2.90**	**65.8**

to real-world users for a large-scale system-level evaluation. In particular, we have recruited 151 users from 30 cities (3 abroad, 27 domestic), and collected 543 hours of video telephony sessions from them. In each session trace, we log the network/application performance metrics (Sec. 6.1) at second-level granularity, which form a dataset of 4.55 GB in total. Currently, OnRL only supports iOS, and the end user devices range from iPhone 7 to iPhone 11 Pro. It is infeasible to implement all the ML based algorithms directly on the operational Taobao-Live app, so we mainly compare OnRL against the default WebRTC algorithm for an in-depth analysis. For a fair comparison, we command each session to use the default WebRTC for the first 10 minutes, and then switch to OnRL afterwards. In this way, we can roughly regard the two algorithms undergoing nearly the same network conditions.

Overall performance. Table 1 summarizes the QoE metrics averaged over all sessions. We observe that, *(i)* OnRL achieves remarkable improvement in terms of transport-layer metrics: 23.5% less packet loss, and 9.92% RTT reduction. *(ii)* The transport-layer advantages transform into application-layer performance gains, *i.e.*, reducing stall rate by 14.22% while maintaining almost the same video throughput. It is noteworthy that the stalling rate here differs from the stalling time metric in Concerto [69] and the user base also differs (151 vs. 6). Thus, the real-world evaluation results are not directly comparable (a direct comparison under the same conditions has been presented in Sec. 6.2). Furthermore, the performance gain may seem modest compared with RL-based VoD systems driven by a simulator/emulator [42], primarily because the operational video telephony system involves real-world users with more heterogeneous network conditions. In addition, for the operational Taobao-Live system, even a small improvement is significant, considering that it can benefit millions of sessions each day. Below we provide a more in-depth analysis of the results.

Performance breakdown. We group the users according to various session attributes, and replot the evaluation results in Fig. 13. Here the number following each legend label stands for the average value of the performance metric.

(i) 4G vs. WiFi. OnRL significantly reduces stall rate by 68.7% and slightly increases the video throughput over 4G, while has very close performance (difference within 3%) over WiFi networks. Accordingly, the reductions in terms of packet loss and RTT are more remarkable over 4G networks. For Taobao-Live use cases, most WiFi users stay indoor with relatively stationary setup, in comparison to 4G which tends to serve more dynamic outdoor scenarios. Therefore, the experiment implies that *OnRL's advantages become more prominent when handling dynamic network conditions.*

(ii) Dynamic vs. stable networks. We thus divide all sessions into stable/dynamic categories according to the variance of video throughput (separated by a threshold set to be the mean std. of 0.197 Mbps). We find that, OnRL improves QoE significantly over dynamics networks, while is not as good (sometimes slightly worse) over stable networks. For instance, the average video RTT is decreased from 171.77ms to 80.34ms for dynamic networks, but is increased from 42.75ms to 48.89ms for stable networks. The other metrics have the similar trend. Still, we conjecture that the main reason behind lies in OnRL's abilities to explore and closely track the available bandwidth under network dynamics. If the network bandwidth is stable, the exploring mechanism may cause overshoots, which result in relatively lower video quality.

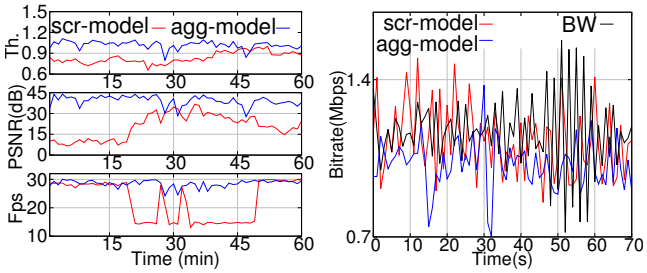
(iii) High vs. low network quality. We further categorize the video sessions based on the empirically observed network condition. If one session's throughput is larger than the mean of all the sessions,

Table 2: The effect of handling RL action deviation.

Metrics	Video th. (Mbps)	Stall rate (%)	Loss rate (%)	PSNR (dB)
Ignoring deviation	0.81	2.56	5.11	35.89
Considering deviation	0.79	1.14	4.58	34.21

Table 3: The effect of robust learning.

Metrics	Video th. (Mbps)	Stall rate (%)	Loss rate (%)	PSNR (dB)
RL-only	0.96	2.30	5.05	38.71
Robust-RL	0.93	0.99	1.84	42.28



(a) QoE metrics. (b) A showcase of bitrate.

Figure 14: Video performance with and without learning aggregation.

we regard the corresponding network quality as high, and low otherwise. We find that OnRL performs significantly better in low-quality network conditions.

6.4 Detailed Analysis of OnRL Pipeline

We now study the impact of each design module inside OnRL separately. Here we conduct controlled experiments on our local testbed since: *(i)* The operational Taobao-Live does not allow testing of separate modules on the end users. *(ii)* We need ground-truth bandwidth for an in-depth understanding of each module's behaviors, which is not available when running Taobao-Live in real networks.

The gain from learning aggregation. We first demonstrate the gain of exploiting prior learning experience, by comparing the performance of an aggregated model (*agg-model*) against the one that starts from scratch (*scr-model*). More specifically, we train 8 separate models from randomly selected Taobao-Live session traces, each lasting longer than 2 hours. Then we aggregate them through averaging as specified in Sec. 2.3. Fig. 14(a) plots the video throughput, PSNR and fps, when running agg-model and scr-model over the same set of new traces lasting 1 hour. We find that, agg-model outperforms scr-model on all important QoE-related metrics: throughput, PSNR, and fps gains are 18.2%, 41.1%, and 47.6%, respectively. Moreover, we can observe that the lower PSNR (< 15dB) under scr-model indicating extremely low quality picture happens at the beginning phase, which demonstrates the cost of starting without any experience.

To further understand above results, we analyze the two models' actions (*i.e.*, the video sending bitrates) in Fig. 14(b). We see that *scr-model* usually sends more traffic than the available bandwidth, which incurs congestion and thus low QoE. Moreover, the actions of *scr-model* exhibit 47.62% higher variance than that of the agg-model, indicating it struggles to explore the available bandwidth at the cost of overshooting/underuse.

The impact of the number of models involved in aggregation. We train 15 different models using randomly 10 hour traces and select 0 (*i.e.*, starting from scratch), 5, 10, and 15 out of them for aggregation. We examine the performance of the resulting 4 aggregated models under a same 1-hour test trace, as depicted in Fig. 15. We observe that: *(i)* Compared with the case with 0 aggregation, the aggregation of 5, 10, 15 models reduce the stall rate by 77.1%, 80.7%, 83.7%, respectively. *(ii)* While model aggregation also improves video throughput, the gain is relatively lower, *i.e.*, 10.8%, 21.27% and 21.3%, respectively. Overall, *the more models involved in aggregation, the more swarm intelligence the aggregated model will have, and the better it can cope with the complex network dynamics.*

The effect of prioritized aggregation. We aggregate 8 random individual models (Each model is trained for 1-hour) in 3 ways: averaged-model, a single model without averaging (named model-1), and prioritized-model (assigning a controlled weight of $p = \lambda_k$ to model-1, $(1-\lambda_k)/7$ to the remaining models, and we vary the range of λ_k from 0.1 to 0.9.). Here we intentionally choose 3 traces from model-1's user, over which the eleven models run. We make an average over 3 traces to mitigate occasionality, and depict the results in Fig. 16. We observe that, *(i)* Even the simplest learning aggregation (averaged-model, stage-2 of OnRL) has 44.9%, 7.3% gains on stall rate and throughput than model-1 (stage-1 of individualized learning). The result indicates that OnRL's learning aggregation stage indeed helps to improve QoE after leveraging swarm intelligence. *(ii)* While the throughput is relatively similar (the largest difference occurs under the two settings of $p = 0.3$ and $p = 0.7$, with a gap of 0.13 Mbps), the prioritized-models have much lower stall rate than model-1 and averaged-model. In particular, $p = 0.4$ shows the lowest stall of 0.27%, while that of model-1 and averaged-model is 4.09% and 2.25%, respectively. *(iii)* Different weighted value of prioritized-models has certain impacts on the QoE metrics. Specially, we find that, the benefits of boundary weights (*e.g.*, $p = 0.1, 0.2, 0.7, 0.9$) are not as good as the middle values (*e.g.*, $p = 0.3, 0.5$). These findings indicate that *for an existing user who already has a trained model, it is better to balance its individual experience with others', by using medium aggregation weights (i.e., close to 0.5), so as to achieve the optimal QoE.*

Effect of handling RL action deviation. We compare two models with and without handling the action deviation, by running them under the same randomly selected network trace lasting 2 hours, respectively. From the result in Table 2, we find that by taking the deviation as an extra input to the RL, the stall rate decreases significantly by 55.4%, loss rate degradation by 10.3%, with slight sacrifice on throughput (0.02Mbps), and PSNR (1.68dB). We thus conclude that *the performance of online training benefits from learning the action deviation explicitly, and the model can therefore achieve high performance even though its action is not strictly executed.*

Effect of robust hybrid learning. We now run OnRL with and without the robust hybrid learning mechanism, labeled as RL-only and Robust-RL, respectively. Table 3 summarizes the performance when the two schemes run on the same 1-hour trace. We observe that *Robust-RL can significantly improve the robustness*: the stall rate and loss rate are reduced by 56.9% and 63.5%, respectively. From the video definition's perspective, robust-RL maintains nearly the same throughput, while improving PSNR by 3.57 dB. To understand the effect of Robust-RL more intuitively, Fig. 17 showcases a 150s

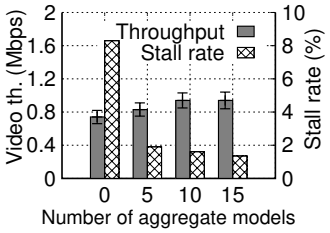


Figure 15: QoE vs. number of models in aggregation.

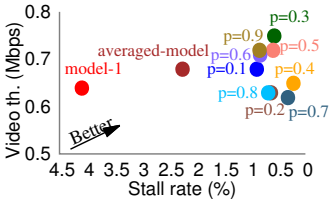


Figure 16: Effect of prioritized aggregation.

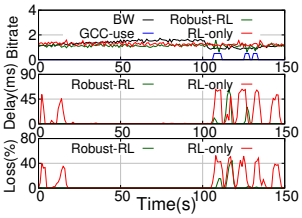


Figure 17: A showcase of Robust-RL's effect.

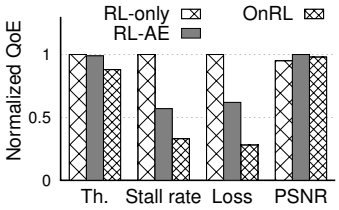


Figure 18: Contribution of separate modules in OnRL.

segment of the experimental traces. We find that during 25 to 100 seconds, the loss rate and delay are relatively low, which shows that the video playback is smooth and the three methods all perform relatively conservatively. Then, we can observe that when RL performs poorly (*e.g.*, during 100s–150s), the loss and delay performance degrades sharply. Robust-RL manages to promptly switch to the conservative GCC algorithm, so as to avoid QoE damage.

In addition, within the 1-hour testing period, we found that Robust-RL evokes GCC for 17 times, and GCC lasts only 28 seconds in total. In particular, at around 10 minutes, Robust-RL has a more frequent switching of 7 times, with GCC lasting 13 seconds in total and taking up around half of the GCC time across the whole session. The results imply that hybrid learning mostly happens in the early stage of online learning. Afterwards, the OnRL agent will learn to converge to appropriate actions and rarely evoke GCC.

Contribution of separate modules in OnRL. We further run a progressive experiment to demonstrate the benefit of OnRL's three modules: *(a)* Pure online RL-only, *(b)* Online RL with action enforcement (RL-AE), and *(c)* Three modules putting together (OnRL). We examine the performance of above three methods under the same 1-hour test trace, and plot the normalized results in Fig. 18. We observe that: *(i)* OnRL's action enforcement and robust learning mechanism have significant impacts on stall rate and loss rate. In particular, the action enforcement mechanism (RL-AE) decreases the stall rate by 42.8%, and the robust learning mechanism further decreases the stall rate by 66.6%. Similarly, the loss rate is reduced by 38.1% and 72.0%, respectively by the two mechanisms. The reason lies in that OnRL's RL-AE is beneficial due to learning the action deviation explicitly, and robust learning can further decrease stalls by mitigating RL's trial-and-error impacts. *(ii)* OnRL's action enforcement and robust learning modules lead to slightly lower throughput and higher PSNR than RL-only, which indicates that they reduce stall rate and packet loss without harm on other performance metrics.

7 RELATED WORK

Low-latency video transport. Real-time Internet video applications impose the toughest requirements on data transport protocols [5, 34]. Although the transport layer has been studied for decades, new network characteristics and applications are constantly emerging that motivate new designs. In contrast to the packet loss metrics commonly used in traditional congestion control protocols [30, 36], many emerging systems target low latency and have explored customized congestion indicators, including a mix of delay and loss [22, 61, 64, 68] as well low-layer KPIs of cellular networks [62, 63]. While these methods can better balance the tradeoff between delay and throughput, they rely on a set of hand-crafted controlling rules, and fall short of handling the increasing heterogeneity and dynamics especially for mobile networks [60].

Starting from Remy [60], machine learning models and particularly RL are utilized to generate more adaptive controlling rules automatically beyond the hand-crafted ones. Remy [60] explores a Markov model (*i.e.*, a kind of tabular RL) to optimize congestion control algorithms. A recent system, Concerto [69], designs an imitation learning (IL, a type of RL) based algorithm to better coordinate the codec and transport layers of video telephony. While making remarkable progress, these models are all trained offline on simulators or emulators. Due to the simulation-to-reality gap, as reported in [20] and corroborated by our experiments (Sec. 6), these approaches often deliver marginal performance gain when tested at large-scale under real network conditions [40].

Learning-based transport for Video-on-Demand (VoD). Learning-based optimization has also been used in VoD services. To name a few, Pensieve [42] adopts the A3C RL algorithm [45] to predict the optimal VoD bitrate that fits the instantaneous network condition. Indigo [66] proposes an IL algorithm to improve VoD QoE. The most recent works [17, 40, 65] move forward to adapt and evaluate ML-based video streaming algorithms in real-world environment. In particular, ABRL [40] makes customized designs based on Pensieve algorithm to accommodate the unique challenges when running it on Facebook VoD systems. QFLow [17] introduces a learning based network reconfiguration framework to achieve high QoE for YouTube streaming. Puffer [65] builds and operates a VoD open platform, and trains a supervised learning model "in situ", *i.e.*, using the traces collected from the real deployment.

Compared with these systems, OnRL differs in two major aspects: *(i)* Essentially, the ML models in these works are still trained offline (though using real-world traces), and thus detour unique challenges arising from online training, *e.g.*, parallel learning and robust learning. *(ii)* They focus on VoD applications, which differ from the interactive video telephony that imposes much harsher requirements atop today's best-effort Internet service [69]. In essence, VoD and video telephony applications are quite different for two reasons: *(i)* The VoD clients commonly maintain a playback buffer of dozens of seconds [15], so they are insensitive to short term (*e.g.*, sub-second level) network dynamics. In contrast, video telephony is more sensitive to instantaneous network traffic variation, which is hard to reproduce in simulation. *(ii)* Most information of a VoD session is known in advance,*e.g.*, the size of video chunks and buffers. In contrast, the video telephony content is always generated and consumed instantaneously. The algorithms need to responsively react to video dynamics under very tough low-latency intervals.

Online learning. Classical online learning advocates training with data streams on the fly [51]. The salient feature is that it keeps refining its prediction model constantly in the presence of a new

environment [47], in contrast to using a stationary post-trained model. Much research effort has been made on theoretical aspects of online learning in the AI community, ranging from the earliest online gradient update [18, 19, 27, 56] to the recent meta-learning [49]. Online learning has rarely been explored for practical network optimization. PCC [25] and PCC-Vivace [26] recently propose congestion control protocols sharing an online learning flavor. They leverage probing packets to continuously optimize a network utility objective. Park [41] proposes an open platform to facilitate experimenting with online RL for solving computer system problems, beyond RL's conventional application domains (*e.g.*, gaming [46]). In this work, OnRL adapts online learning to address the unique challenges in real-time video telephony.

In OnRL, we also propose a hybrid learning mechanism, which has shown its capability of preventing the catastrophic effect caused by RL's inherent but risky trial-and-error exploration. The hybrid learning component is inspired by the concept of safe learning, which has been adapted in other RL-based domains. For example, Shielding [16] detours RL's certain actions when a safety condition is triggered. A more recent work [43] proposes a guaranteed-safe policy to dynamically solve load balancing problem. To our knowledge, OnRL is the first framework to customize hybrid learning to improve the robustness of real-time video telephony.

Federated Learning is a distributed machine learning architecture, in which data is stored on distributed local devices instead of central servers [44]. Generally, federated learning aims to aggregate multiple users' experiences while preserving privacy, and also to reduce communication overhead [38, 67]. The learning aggregation component of OnRL is inspired by the concept of federated learning. But OnRL goes much further to solve practical problems in optimizing video telephony, including designing the specific RL neural network suitable with video telephony, enforcing RL's action despite video traffic dynamics and enhancing video robustness by designing the hybrid learning mechanism.

8 DISCUSSION

On-device online learning. Our current deployment of OnRL adopts a cloud-assisted architecture. The key design modules are located in remote cloud servers instead of the mobile devices, to accommodate the lack of mobile platform for RL training. Note that the RL cloud-server is unlikely to cause any privacy issue, as it only collects transport layer performance statistics, *i.e.*, loss rate, RTT, without any personal information. In addition, it is also noteworthy that the mainstream Tensorflow-Lite [12] and Core-ML [10] only allow executing pre-trained RL models, *i.e.*, they support the inference phase but not the training phase. The only RL-training-support platform in a recent work [57], as far as we know, is developed in Java and hard to be integrated with Taobao-Live. However, as neural processing units (NPUs) rapidly become available on mobile devices, neural network training on mobile devices will be feasible. On-device training will eliminate the overhead and cost of deploying RL servers, which shall facilitate the wide use of online learning algorithms like OnRL.

Adaptive learning aggregation. In OnRL, we have verified the significance of learning aggregation, by designing the weighted linear aggregation algorithms. Besides, we have tried more fine-grained aggregation methods, *i.e.*, aggregating a separate model for a group of users belonging to a certain network type (*e.g.*, WiFi or 4G) or ISP, but observed little gain. The results hint that the available bandwidth dynamics (particularly the short-term bandwidth variation that is important for OnRL) of an Internet path may not have strong correlation to its network type or ISP. On the other hand, we conjecture that other fine-grained aggregation methods using content-related attributes, such as grouping users by telephony scenarios (indoor, outdoor walking, or driving), may further improve video QoE. These approaches involve non-trivial design (*e.g.*, addressing user privacy), and is left for the next-phase of OnRL.

Closer integration of RL and rule-based algorithms. The evaluation in Sec. 6.3 demonstrates that RL exhibits significant advantages in handling network dynamics, but sometimes underperforms rule-based protocols under stable network conditions. It is possible to integrate OnRL and the rule-based algorithms according to the level of network dynamics, just as done in the hybrid learning scheme. We leave this for future work.

OnRL's scalability. We note that OnRL does not aim to improve reinforcement learning in general, but focuses on addressing the system-level challenges when adapting learning algorithms to optimize video telephony transport. The novelty of OnRL lies in the two-stage learning framework (including the neural network architecture and training methodology), the RL action enforcement and robust hybrid learning mechanisms, as well as in implementing, deploying and evaluating online RL on a mainstream operational video telephony system. We believe these customized mechanisms of OnRL may benefit future work on optimizing video transport and even other computer system design, such as resource scheduling and load balancing in practical networks. To extend the usage of OnRL to other applications, one can reuse the architecture and mechanisms, but need to re-train RL models using the applications' own data.

9 CONCLUSION

In this work, we designed an online reinforcement learning based real-time video telephony system named OnRL. We solve three unique challenges: learning from concurrent telephony sessions; enforcing RL actions despite of inherent video traffic variations; avoiding the QoE damage from reckless exploitation of RL. The real-world evaluation on a mainstream operational video telephony system demonstrates that OnRL outperforms the state-of-the-art solutions. We believe OnRL hints on a new direction that embraces online learning into more video communication applications, such as VoD, 360 panoramic video, virtual reality, *etc.*

ACKNOWLEDGMENTS

We appreciate the insightful feedback from the anonymous reviewers and our shepherd who helped improve this work. Anfu Zhou and Huadong Ma are the corresponding authors. This project was supported by the Innovation Research Group Project of NSFC (61921003), NSFC (61772084, 61720106007, 61832010), the 111 Project (B18008), the Fundamental Research Funds for the Central Universities (2019XD-A13) and Alibaba Innovation Research Program.

REFERENCES

[1] 2017. Closing the Simulation-to-Reality Gap for Deep Robotic Learning. https://ai.googleblog.com/2017/10/closing-simulation-to-reality-gap-for.html.

[2] 2017. Video Quality of Service (QOS) Tutorial . https://www.cisco.com/c/en/us/support/docs/quality-of-service-qos/qos-video/212134-Video-Quality-of-Service-QOS-Tutorial.html.

[3] 2018. WebRTC Homepage. https://webrtc.org/.

[4] 2019. Alibaba Cloud Overview. https://www.assistanz.com/alibaba-cloud-overview/.

[5] 2019. Cisco Visual Networking Index: Forecast and Trends. https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white-paper-c11-741490.html?dtid=osscdc000283.

[6] 2019. Taobao statistics, market share, trends, insights . https://www.chinainternetwatch.com/tag/taobao/.

[7] 2020. 30 amazing taobao statistics and facts (2020). By the numbers. https://expandedramblings.com/index.php/taobao-statistics/.

[8] 2020. Alibaba cloud, Server Load Balancer. https://www.alibabacloud.com/help/doc-detail/27544.htm?spm=a2c63.p38356.b99.5.4bc42299yeUcWm.

[9] 2020. Chromium in webrtc. https://chromium.googlesource.com/external/webrtc/.

[10] 2020. Core ML framework. https://developer.apple.com/documentation/coreml.

[11] 2020. Linux Traffic Control. https://events.static.linuxfound.org/sites/events/files/slides/Linux_traffic_control.pdf.

[12] 2020. Tensorflow Lite . https://www.tensorflow.org/lite.

[13] 2020. Tensorflow source code. https://github.com/tensorflow/tensorflow/tree/master/tensorflow/tools..

[14] 2020. TFLearn: Deep learning library featuring a higher-level API for TensorFlow. http://tflearn.org/.

[15] Saamer Akhshabi, Sethumadhavan Narayanaswamy, Ali C. Begen, and Constantine Dovrolis. 2012. An experimental evaluation of rate-adaptive video players over HTTP. *Signal Process. Image Commun.* 27, 4 (2012), 271–287. https://doi.org/10.1016/j.image.2011.10.003

[16] Mohammed Alshiekh, Roderick Bloem, Rüdiger Ehlers, Bettina Könighofer, Scott Niekum, and Ufuk Topcu. 2018. Safe Reinforcement Learning via Shielding. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018.* 2669–2678. https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/17211

[17] Rajarshi Bhattacharyya, Archana Bura, Desik Rengarajan, Mason Rumuly, Srinivas Shakkottai, Dileep M. Kalathil, Ricky K. P. Mok, and Amogh Dhamdhere. 2019. QFlow: A Reinforcement Learning Approach to High QoE Video Streaming over Wireless Networks. In *Proceedings of the Twentieth ACM International Symposium on Mobile Ad Hoc Networking and Computing, Mobihoc 2019, Catania, Italy, July 2-5, 2019.* 251–260. https://doi.org/10.1145/3323679.3326523

[18] Léon Bottou and Yann LeCun. 2003. Large Scale Online Learning. In *Advances in Neural Information Processing Systems 16 [Neural Information Processing Systems, NIPS 2003, December 8-13, 2003, Vancouver and Whistler, British Columbia, Canada].* 217–224.

[19] Haitham Bou-Ammar, Eric Eaton, Paul Ruvolo, and Matthew E. Taylor. 2014. Online Multi-Task Learning for Policy Gradient Methods. In *Proceedings of the 31th International Conference on Machine Learning, ICML 2014, Beijing, China, 21-26 June 2014.* 1206–1214.

[20] Konstantinos Bousmalis, Alex Irpan, Paul Wohlhart, Yunfei Bai, Matthew Kelcey, Mrinal Kalakrishnan, Laura Downs, Julian Ibarz, Peter Pastor, Kurt Konolige, Sergey Levine, and Vincent Vanhoucke. 2018. Using Simulation and Domain Adaptation to Improve Efficiency of Deep Robotic Grasping. In *2018 IEEE International Conference on Robotics and Automation, ICRA 2018, Brisbane, Australia, May 21-25, 2018.* 4243–4250.

[21] Neal Cardwell, Yuchung Cheng, C. Stephen Gunn, Soheil Hassas Yeganeh, and Van Jacobson. 2016. BBR: Congestion-Based Congestion Control. *ACM Queue* 14, 5 (2016), 20–53.

[22] Gaetano Carlucci, Luca De Cicco, Stefan Holmer, and Saverio Mascolo. 2017. Congestion Control for Web Real-Time Communication. *IEEE/ACM Trans. Netw.* 25, 5 (2017), 2629–2642.

[23] Jason Clements, Teodros Gessesse, Darshan Sedani, and Jerry Klein. 2016. Live video broadcasting mobile application for social sharing. US Patent App. 14/821,519.

[24] Yann Le Cun, Ido Kanter, and Sara A. Solla. [n.d.]. Eigenvalues of covariance matrices: Application to neural-network learning. *Physical Review Letters* 66, 18 ([n. d.]), 2396–2399.

[25] 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)*. USENIX Association, Oakland, CA, 395–408. https://www.usenix.org/conference/nsdi15/technical-sessions/presentation/dong

[26] 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 2018, Renton, WA, USA, April 9-11, 2018.* 343–356.

[27] John C. Duchi, Elad Hazan, and Yoram Singer. 2011. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *J. Mach. Learn. Res.* 12 (2011), 2121–2159.

[28] Sadjad Fouladi, John Emmons, Emre Orbay, Catherine Wu, Riad S. Wahby, and Keith Winstein. 2018. Salsify: Low-Latency Network Video through Tighter Integration between a Video Codec and a Transport Protocol. In *15th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2018, Renton, WA, USA, April 9-11, 2018.* 267–282.

[29] Ian J. Goodfellow and Oriol Vinyals. 2015. Qualitatively characterizing neural network optimization problems. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings.* http://arxiv.org/abs/1412.6544

[30] Sangtae Ha, Injong Rhee, and Lisong Xu. 2008. CUBIC: a new TCP-friendly high-speed TCP variant. *Operating Systems Review* 42, 5 (2008), 64–74.

[31] Marie Hopkins. 2017. Live sports virtual reality broadcasts: Copyright and other protections. *Duke L. & Tech. Rev.* 16 (2017), 141.

[32] Quan Huynh-Thu and Mohammed Ghanbari. 2012. The accuracy of PSNR in predicting video quality for different video scenes and frame rates. *Telecommunication Systems* 49, 1 (2012), 35–48. https://doi.org/10.1007/s11235-010-9351-x

[33] Nathan Jay, Noga H. Rotman, Brighten Godfrey, Michael Schapira, and Aviv Tamar. 2019. A Deep Reinforcement Learning Perspective on Internet Congestion Control. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA.* 3050–3059.

[34] Junchen Jiang, Rajdeep Das, Ganesh Ananthanarayanan, Philip A. Chou, Venkat N. Padmanabhan, Vyas Sekar, Esbjorn Dominique, Marcin Goliszewski, Dalibor Kukoleca, Renat Vafin, and Hui Zhang. 2016. Via: Improving Internet Telephony Call Quality Using Predictive Relay Selection. In *Proceedings of the ACM SIGCOMM 2016 Conference, Florianopolis, Brazil, August 22-26, 2016.* 286–299.

[35] Junchen Jiang, Vyas Sekar, and Hui Zhang. 2012. Improving Fairness, Efficiency, and Stability in HTTP-Based Adaptive Video Streaming with FESTIVE. In *Proc. of International Conference on Emerging Networking Experiments and Technologies (CoNEXT).*

[36] Yueqiu Jiang, Junkun Zhang, and Qixue Guan. 2014. Improvement of TCP Reno Congestion Control Protocol. *Sensors & Transducers* 163, 1 (2014), 308.

[37] Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings.* http://arxiv.org/abs/1412.6980

[38] Yujun Lin, Song Han, Huizi Mao, Yu Wang, and Bill Dally. 2018. Deep Gradient Compression: Reducing the Communication Bandwidth for Distributed Training. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings.*

[39] Amanda Lee Kelly Lockwood, Ravi Gogna, Gary Linscott, Timothy Caldwell, Marin Kobilarov, Paul Orecchio, Dan Xie, Ashutosh Gajanan Rege, and Jesse Sol Levinson. 2019. Interactions between vehicle and teleoperations system. US Patent App. 15/644,310.

[40] Hongzi Mao, Shannon Chen, Drew Dimmery, Shaun Singh, Drew Blaisdell, Yuandong Tian, Mohammad Alizadeh, and Eytan Bakshy. 2019. Real-world Video Adaptation with Reinforcement Learning. In *Proceedings of the 2019 Reinforcement Learning for Real Life Workshop.*

[41] Hongzi Mao, Parimarjan Negi, Akshay Narayan, Hanrui Wang, Jiacheng Yang, Haonan Wang, Ryan Marcus, Ravichandra Addanki, Mehrdad Khani Shirkoohi, Songtao He, Vikram Nathan, Frank Cangialosi, Shaileshh Bojja Venkatakrishnan, Wei-Hung Weng, Song Han, Tim Kraska, and Mohammad Alizadeh. 2019. Park: An Open Platform for Learning-Augmented Computer Systems. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, 8-14 December 2019, Vancouver, BC, Canada.* 2490–2502. http://papers.nips.cc/paper/8519-park-an-open-platform-for-learning-augmented-computer-systems

[42] Hongzi Mao, Ravi Netravali, and Mohammad Alizadeh. 2017. Neural Adaptive Video Streaming with Pensieve. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication, SIGCOMM 2017, Los Angeles, CA, USA, August 21-25, 2017.* 197–210.

[43] Hongzi Mao, Malte Schwarzkopf, Hao He, and Mohammad Alizadeh. 2019. Towards Safe Online Reinforcement Learning in Computer Systems. In *33rd Conference on Neural Information Processing Systems (NeurIPS 2019).*

[44] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. 2017. Communication-Efficient Learning of Deep Networks from Decentralized Data. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics, AISTATS 2017, 20-22 April 2017, Fort Lauderdale, FL, USA.* 1273–1282. http://proceedings.mlr.press/v54/mcmahan17a.html

[45] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. 2016. Asynchronous Methods for Deep Reinforcement Learning. In *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016.* 1928–1937.

[46] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis

Antonoglou, Daan Wierstra, and Martin A. Riedmiller. 2013. Playing Atari with Deep Reinforcement Learning. *CoRR* abs/1312.5602 (2013). arXiv:1312.5602 http://arxiv.org/abs/1312.5602

[47] Anusha Nagabandi, Chelsea Finn, and Sergey Levine. 2019. Deep Online Learning Via Meta-Learning: Continual Adaptation for Model-Based RL. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*.

[48] Matteo Pirotta, Marcello Restelli, and Luca Bascetta. 2013. Adaptive Step-Size for Policy Gradient Methods. In *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States*. 1394–1402. http://papers.nips.cc/paper/5186-adaptive-step-size-for-policy-gradient-methods

[49] Sachin Ravi and Hugo Larochelle. 2017. Optimization as a Model for Few-Shot Learning. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*.

[50] Stéphane Ross, Geoffrey J. 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, AISTATS 2011, Fort Lauderdale, USA, April 11-13, 2011*. 627–635.

[51] Doyen Sahoo, Quang Pham, Jing Lu, and Steven C. H. Hoi. 2018. Online Deep Learning: Learning Deep Neural Networks on the Fly. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden*. 2660–2666.

[52] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal Policy Optimization Algorithms. *CoRR* abs/1707.06347 (2017).

[53] Ivan Slivar, Mirko Suznjevic, and Lea Skorin-Kapov. 2018. Game Categorization for Deriving QoE-Driven Video Encoding Configuration Strategies for Cloud Gaming. *TOMM* 14, 3s (2018), 56:1–56:24.

[54] Taobao. 2020. Taobao Live APP. https://apps.apple.com/cn/app/E6B798E5AE9DE79BB4E692AD/id1448831879/.

[55] Taobao. 2020. Taobao-live monthly report, 2020 February. https://mp.weixin.qq.com/s/wNMfYAYcTI5T3O10x6hA2w/.

[56] Kyriakos G. Vamvoudakis and Frank L. Lewis. 2009. Online actor critic algorithm to solve the continuous-time infinite horizon optimal control problem. In *International Joint Conference on Neural Networks, IJCNN 2009, Atlanta, Georgia, USA, 14-19 June 2009*. 3180–3187. https://doi.org/10.1109/IJCNN.2009.5178586

[57] Cong Wang, Yanru Xiao, Xing Gao, Li Li, and Jun Wang. 2019. Close the Gap between Deep Learning and Mobile Intelligence by Incorporating Training in the Loop. In *Proceedings of the 27th ACM International Conference on Multimedia, MM 2019, Nice, France, October 21-25, 2019*. 1419–1427. https://doi.org/10.1145/3343031.3350904

[58] Haiyang Wang, Tong Li, Ryan Shea, Xiaoqiang Ma, Feng Wang, Jiangchuan Liu, and Ke Xu. 2018. Toward Cloud-Based Distributed Interactive Applications: Measurement, Modeling, and Analysis. *IEEE/ACM Trans. Netw.* 26, 1 (2018), 3–16.

[59] Ziheng Wang, Isabella Reed, and Ann Majewicz Fey. 2018. Toward Intuitive Teleoperation in Surgery: Human-Centric Evaluation of Teleoperation Algorithms for Robotic Needle Steering. In *2018 IEEE International Conference on Robotics and Automation, ICRA 2018, Brisbane, Australia, May 21-25, 2018*. 1–8.

[60] Keith Winstein and Hari Balakrishnan. 2013. TCP ex machina: computer-generated congestion control. In *ACM SIGCOMM 2013 Conference, SIGCOMM'13, Hong Kong, China, August 12-16, 2013*. 123–134.

[61] Keith Winstein, Anirudh Sivaraman, and Hari Balakrishnan. 2013. Stochastic Forecasts Achieve High Throughput and Low Delay over Cellular Networks. In *Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation (nsdi'13)*.

[62] Xiufeng Xie, Xinyu Zhang, Swarun Kumar, and Li Erran Li. 2015. PiStream: Physical Layer Informed Adaptive Video Streaming over LTE. In *Proceedings of the ACM Annual International Conference on Mobile Computing and Networking (MobiCom)*.

[63] Xiufeng Xie, Xinyu Zhang, and Shilin Zhu. 2017. Accelerating Mobile Web Loading Using Cellular Link Information. In *Proceedings of the Annual International Conference on Mobile Systems, Applications, and Services (MobiSys)*.

[64] Qiang Xu, Sanjeev Mehrotra, Zhuoqing Mao, and Jin Li. 2013. PROTEUS: Network Performance Forecast for Real-time, Interactive Mobile Applications. In *Proceeding of the 11th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys '13)*. 347–360.

[65] 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 2020, Santa Clara, CA, USA, February 25-27, 2020*. 495–511. https://www.usenix.org/conference/nsdi20/presentation/yan

[66] 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 2018, Boston, MA, USA, July 11-13, 2018*. 731–743.

[67] Xin Yao, Tianchi Huang, Chenglei Wu, Rui-Xiao Zhang, and Lifeng Sun. 2019. Federated Learning with Additional Mechanisms on Clients to Reduce Communication Costs. *CoRR* abs/1908.05891 (2019).

[68] Yasir Zaki, Thomas Pötsch, Jay Chen, Lakshminarayanan Subramanian, and Carmelita Görg. 2015. Adaptive Congestion Control for Unpredictable Cellular Networks. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication (SIGCOMM '15)*. ACM, New York, NY, USA, 509–522. https://doi.org/10.1145/2785956.2787498

[69] Anfu Zhou, Huanhuan Zhang, Guangyuan Su, Leilei Wu, Ruoxuan Ma, Zhen Meng, Xinyu Zhang, Xiufeng Xie, Huadong Ma, and Xiaojiang Chen. 2019. Learning to Coordinate Video Codec with Transport Protocol for Mobile Video Telephony. In *The 25th Annual International Conference on Mobile Computing and Networking, MobiCom 2019, Los Cabos, Mexico, October 21-25, 2019*. 29:1–29:16.