Loki: Improving Long Tail Performance of Learning-Based Real-Time Video Adaptation by Fusing Rule-Based Models

Huanhuan Zhang[+], Anfu Zhou[+], Yuhan Hu[+], Chaoyue Li[+], Guangping Wang[+], Xinyu Zhang[◇], Huadong Ma[+], Leilei Wu, Aiyun Chen, Changhui Wu

[+]Beijing Univ. of Posts and Telecom. {zhanghuanhuan, zhouanfu,mhd}@bupt.edu.cn

[◇] University of California San Diego xyzhang@ucsd.edu

ABSTRACT

Maximizing the quality of experience (QoE) for real-time video is a long-standing challenge. Traditional video transport protocols, represented by a few deterministic rules, can hardly adapt to the heterogeneous and highly dynamic modern Internet. Emerging learning-based algorithms have demonstrated potential to meet the challenge. However, our measurement study reveals an alarming long tail performance issue: these algorithms tend to be bottlenecked by occasional catastrophic events due to the built-in exploration mechanisms. In this work, we propose Loki, which improves the robustness of learning-based model by coherently integrating it with a rule-based algorithm. To enable integration at feature level, we first reverse-engineer the rule-based algorithm into an equivalent "black-box" neural network. Then, we design a dual-attention feature fusion mechanism to fuse it with a reinforcement learning model. We train Loki in a commercial real-time video system through online learning, and evaluate it over 101 million video sessions, in comparison to state-of-the-art rule-based and learning-based solutions. The results show that Loki improves not only the average but also the tail performance substantially (26.30% to 44.24% reduction of stall rate and 1.76% to 2.17% increase in video throughput at 95-percentile).

CCS CONCEPTS

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

KEYWORDS

Real-time video; Hybrid learning; Large-scale deployment

ACM Reference Format:

Huanhuan Zhang, Anfu Zhou, Yuhan Hu, Chaoyue Li, Guangping Wang, Xinyu Zhang, Huadong Ma, Leilei Wu, Aiyun Chen, Changhui Wu. 2022. Loki: Improving Long Tail Performance of Learning-Based Real-Time Video Adaptation by Fusing Rule-Based Models. In *The 27th Annual International Conference on Mobile Computing and Networking (ACM MobiCom '21), January 31-February 4, 2022, New Orleans, LA, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3447993.3483259

ACM MobiCom '21, January 31-February 4, 2022, New Orleans, LA, USA

ACM ISBN 978-1-4503-8342-4/22/01…$15.00
https://doi.org/10.1145/3447993.3483259

1 INTRODUCTION

Real-time interactive video is becoming increasingly prevalent. From online education, remote work, to trade shows [11] and even social gatherings [10], real-time video is reshaping our lifestyle, especially amid the COVID-19 pandemic [24]. Looking into the near future, new real-time video applications are emerging owing to the 5G mobile broadband deployment. Examples include 3D volumetric video interaction [39, 45], robotic teleoperation [12, 42] and so on. Market reports predict that real-time video will account for 17% of the total Internet video traffic [8], and generate up to $217.3 billion revenue by 2025 worldwide [15].

Despite decades of development, guaranteeing quality of experience (QoE) for real-time video remains an open challenge. On one hand, real-time video is both latency sensitive and bandwidth hungry (*e.g.*, around 5 Mbps for 1080p and 25 Mbps for 4K video [4]). The lifecycle of a video frame, from video capture, encoding at the sender side, transmission over the Internet, to the decoding and rendering at the receiver side, should be accomplished in dozens of milliseconds [3]. On the other hand, the modern Internet is becoming highly heterogeneous, including fiber, WiFi, LTE, as well as the emerging 5G mobile networks, with very diverse bandwidth and latency characteristics. Classical general-purpose data transport protocols (*e.g.*, GCC [27], BBR [26], Proteus [64]) struggle in simultaneously optimizing the video bitrate and latency. They are plagued with overly conservative bandwidth estimation [27, 61], inability to track the instantaneous network dynamics [26], and lack of agility when confronted with unseen network conditions [64].

Recent years have witnessed the rise of learning-based video transport design [18, 43, 50, 66, 69, 71]. Unlike the classical rule-based protocols, a learning-based approach typically trains a neural network model using a large data set that covers various network conditions. Whereas the data-driven model generally brings high performance on average, the lack of robustness due to its unexplainable and non-deterministic behaviors, remains as a major hindrance for at-scale deployment in production systems. In effect, regardless of the average performance, a single instance of catastrophic QoE degradation may cause users to abandon the application [49].

To demystify the tail performance of real-time video transport, we have conducted a measurement comparison over the state-of-the-art rule-based and learning-based algorithms. Our investigation leads to two key observations: *(i)* Learning-based approaches indeed exhibit gains in transport layer metrics, *e.g.*, higher throughput and lower delay on average. Yet surprisingly, the gains do not necessarily translate into QoE improvement—the application layer's frame delay and frame jitter may even become worse. An in-depth analysis

uncovers the reason: with learning-based algorithms, the transport layer performance metrics exhibit a long-tailed distribution. For instance, although the average packet-level RTT is low (*e.g.*, 28.96 ms), a small fraction of packets experience excessively large RTT (*e.g.*, 600 ms). Since a video frame consists of multiple packets, even a single outlier packet can result in large frame delay/jitter and adversely impact the playback smoothness. *(ii)* Learning-based methods occasionally output inaccurate bandwidth estimation, severe overshoots in particular, which leads to catastrophic behaviors, *e.g.*, frame skipping or even temporary video stalling. We find the root cause lies in that these methods, particularly the widely used reinforcement learning algorithms, inherently make decisions through trial-and-error exploration. They seek to maximize long-term cumulative rewards at the cost of occasional bandwidth overshoot or under-utilization, which results in poor long-tail performance. All in all, we find that learning-based methods fall short of robustness.

Motivated by the observations, we seek to answer a key question: *Can we design a real-time video transport algorithm that achieves high-performance both on average and at the tails?* To this end, we propose Loki[1], a hybrid model that simultaneously harnesses the deterministic nature of rule-based methods and the prediction capacity of learning-based approaches. When the network condition is unstable, Loki makes conservative video bitrate decisions that are typical of rule-based methods to avoid catastrophic QoE degradation. Otherwise, Loki can fully utilize the network capacity through reinforcement learning-like exploration.

Despite the simplicity of the idea at a high level, we identify two unique design challenges for Loki.

(i) How to make the rule-based and learning-based models compatible and hence "fusable"? The former is comprised of hard coded if-then" rules, whereas the latter adopts neural network (NN) based black-box feature representation. Prior work often has to adopt a "time division" scheme when integrating the two. For example, Orca [18] runs a rule-based transport-layer protocol [37] most of the time, while occasionally resorting to the learning-based TD3 [34] to regulate the base bitrate. Similarly, OnRL [69] uses the rule-based protocol [27] as a fallback only when it suspects that the learning model is behaving abnormally. However, our experiments reveal that such multiplexing approaches fail to exploit the integrated power of both and often lead to unsatisfactory performance (Sec. 2). In contrast, Loki aims for a deeper fusion and synergy between the two. Loki transforms a "white-box" rule based method (*i.e.*, GCC) into an equivalent "black-box" NN model through a customized imitation learning model. In this way, the two approaches become compatible and can be fused at the "feature-level" rather than decision-level.

(ii) How to ensure the feature fusion actually captures the advantages of rule-based and learning-based models respectively? We design a *dual-attention feature fusion mechanism* to meet this requirement. In particular, Loki treats the learned high-level features of the two NN models as a kind of attention or confidence coefficient. It prioritizes the feature set that has more confidence of achieving higher QoE, by giving it larger weights when co-determining an optimal decision. To materialize the attention mechanism, we train the integrated model in a large commercial interactive video system based on a state-of-the-art reinforcement learning algorithm [57]. After iteratively training, Loki finally evolves to be a self-contained model, with the ability of automatically reacting to new environments in a reliable way.

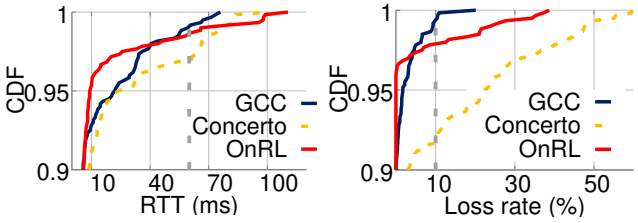


Figure 1: CDF of transport-layer metrics.

We implement and deploy Loki in a top commercial interactive mobile video system APP[2] that serves tens of millions of users. We have also deployed 3 other state-of-the-art solutions: the rule-based GCC [27], the learning-based OnRL [69], and the hybrid Orca [18] for benchmark comparisons. Our experiments involve 101 million video sessions in the production system, totaling 150 years of playback time. Compared with the baselines, Loki reduces the stall rate by 13.98%-27.27% and improves the video quality by 1.37%-5.71% on average. Meanwhile, Loki substantially cuts the long-tails, *e.g.*, it decreases the 95-percentile stall rate by 26.30%-44.24%. By tackling the long-tail QoE, Loki leads to 2.62%-4.68% longer averaged viewing time than the default GCC, which can translate into remarkable commercial value.

Contributions: Loki's key novelty and technical contributions can be summarized as follows:

(i) Through a measurement study, we investigate the long-tail performance issues of modern learning-based algorithms for real-time video adaptation (Sec. 2).

(ii) We propose a scheme to "blackboxify" rule-based algorithms, making them compatible and "fusable" with the learning-based algorithms (Sec.4). The scheme can potentially be applied to other rule-based protocol designs.

(iii) We design a dual-attention mechanism to tightly coordinate the rule-based and learning-based models, enabling deep feature level instead of decision level fusion (Sec. 5, 6).

(iv) We implement and deploy Loki on a commercial real-time video system, and validate its effectiveness in comparison with state-of-the-art solutions (Sec. 7, 8). To our knowledge, this is the first-of-its-kind field test of learning-based real-time video adaptation at a scale of several hundred million sessions.

2 MEASUREMENT AND FINDINGS

In this section, we present a microscopic measurement study to understand the tail performance of current learning-based algorithms for real-time video adaptation.

2.1 Measurement Methodology

Testbed implementation. We build an emulation testbed for real-time interactive video system based on WebRTC [14] – the de facto video communication framework adopted by mainstream real-time video/audio applications [7], *e.g.*, Google Hangouts, Facebook, and Amazon Chime. WebRTC implements both the transport layer and

[1]Loki is a figure in Norse mythology who's good at shape-shifting [16].

[2]We anonymize the APP due to legal issues.

Table 1: The panoramic comparisons of rule-based GCC [27], learning-based Concerto [71] and OnRL [69].

Metrics	Transport layer				Application layer			Catastrophic counts	
	Bitrate (Mbps)	Loss rate (%)	RTT (ms)	Packet stall (RTT, %)	Frame delay (ms)	Frame jitter (ms)	Frame stall (%)	Bitrate overshoot (/h)	Fps < 12 (/h)
GCC	1.0	2.01	41.54	3.4	37.41	178.62	1.46	222	52
Concerto	1.1	4.84	23	0.57	40.77	141.21	5.88	375	215
OnRL	1.17	1.94	28.96	2.0	41.83	219.9	2.04	463	73

application layer protocols. At the transport layer, it encapsulates a bitrate control algorithm based on Google Congestion Control (GCC); and at the application layer, it runs a video codec (*e.g.*, H.264, H.265) which generates the video frames for a given target bitrate. In our testbed setup, two PC hosts are connected by an Ethernet switch, one as video sender and the other as receiver. The sender uses a camera to capture real-world scenes, which are delivered to the receiver and rendered in real-time. We use the Linux traffic control (tc) tool [13] to emulate the end-to-end network bandwidth variation, which follows the traces collected from the commercial real-time video system. The traces cover users in more than 50 countries, consisting of fine-grained (*i.e.*, 1-second granularity) network statistics (throughput, packet delay, loss, *etc.*) and diverse network types (WiFi, 4G and wire lines).

State-of-the-art algorithms for real-time video delivery. We compare three typical algorithms for real-time video transport: *(i)* Google Congestion Control (GCC) [27], the default and most popular rule-based protocol inside the WebRTC framework. It is commonly regarded as a video transport algorithm since that its bitrate will be directly considered as the video sending bitrate by video codec. *(ii)* Concerto [71], an imitation learning [41] based approach that fuses the network information from transport-layer and application-layer in order to optimize video bitrate control. *(iii)* OnRL [69], an online reinforcement learning (RL) based algorithm for real-time video, which incorporates a robust learning mechanism, *i.e.*, deeming the legacy rule-based protocol as a "backup" when the RL model shows poor QoE.

2.2 Understanding the Limitations of Learning-based Designs

We experiment with different algorithms using the same set of 7 randomly selected real-time video traces with a total length of 5 hours. The resulting transport-layer and application-layer performance metrics are summarized in Tab. 1. We observe that: *(i)* The learning-based Concerto and OnRL indeed significantly improve transport layer performance, *e.g.*, 10%, 17% gains on video bitrate, 44.63%, 30.28% drops in RTT, respectively. In addition, OnRL has a 3.48% reduction in packet loss than GCC. The packet stall rate[3] is 83.24% and 41.18% lower than that of GCC. *(ii)* We examine close-to-human perceptible QoE metrics, *e.g.*, frame delay, frame jitter, and frame stall rate. We note that prior works, *e.g.*, Concerto and OnRL, focus on transport metrics, but haven't derived and looked into these QoE metrics. For instance, OnRL [69] reports packet stall but not frame stall. We surprisingly find that the transport-layer gains of Concerto and OnRL do not necessarily translate to QoE improvement at the application layer. The frame-level QoE metrics

[3]Here, the packet stall rate is specified as the ratio of packets whose RTT exceeds a certain threshold, consistent with the definition of *stall rate* in OnRL [69].

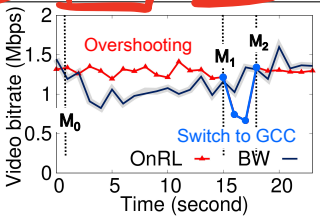


Figure 2: OnRL exhibits long-time overshooting than GCC.

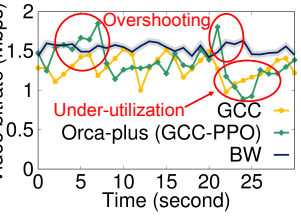


Figure 3: Orca-plus has more overshooting and under-utilization.

actually deteriorate, *i.e.*, 11.82%, 23.11% increase in terms of frame delay and frame jitter for OnRL, respectively. Worse still, the frame stall rate, which is close to the human perceptible video freezing effect, is 3.03× and 39.73% larger than that of GCC for Concerto and OnRL, respectively.

Long-tail performance. An in-depth analysis reveals that the performance gap stems from the long-tailed distribution of the transport-layer metrics. As a showcase, we run the three algorithms on a network trace collected from the same video session, and plot the CDF of RTT and packet loss rate with 1-second level granularity in Fig. 1. We observe that the fraction of excessively long RTT (*i.e.*, >160 ms) takes up 1.0% of the packets in GCC, in contrast to 3.17% and 1.5% in Concerto and OnRL, respectively. Similarly, the fraction of excessively large loss rate (*i.e.*, >10%) is 0.67% vs. 8.17% and 2.17%, respectively. As we know, a video frame can be rendered and played only after all its constituent packets arrive. So even a single lost or belated packet may increase the frame delay, which in turn causes a chain effect as nearby frames often rely on each other for decoding. For example, we have identified cases where a 10% loss rate or a 100 ms RTT can cause a large frame delay lasting for 1.2 seconds, which jeopardizes the video fluency.

Learning-based algorithms suffer from more catastrophic QoE degradation events. To examine the severity of the QoE degradation, we count the occurrences of catastrophic events, *i.e.*, bitrate estimation overshooting over 100 Kbps, and fps smaller than 12, empirically set according to previous studies [69]. The last two columns in Tab. 1 summarize the statistics of the 7 traces, normalized into 1-hour granularity. We can observe that: *(i)* Concerto and OnRL lead to more bitrate overshoots and excessive frame delay, and thus more low-fps events, *e.g.*, 3.13×, 40.38% compared with GCC, which severely damage the QoE. Whereas Concerto has fewer bitrate overshoots, the overshooting magnitude is much larger than OnRL, which ultimately results in higher loss rate and more low-fps counts.

2.3 Why are existing hybrid learning approaches ineffective?

The idea of integrating rule-based and learning-based approaches

emerged very recently. OnRL [69] and Orca [18] represent the state-of-the-arts. However, both essentially combine the two approaches in a "time division" manner, which lacks robustness as shown below.

OnRL incorporates an overuse detector to identify possible overshoots and switches to the conservative rule-based GCC, in the hope of reducing catastrophic behaviors [69]. However, we find that such a fallback operation only works *after the catastrophic event already started*. Fig. 2 showcases the issue: an overshoot event starts at M_0, OnRL detects the event and switches to GCC at M_1, about 15 seconds later.

Orca [18] uses a two-level control policy that alternates between rule-based and learning-based adaptation. It is originally designed for TCP congestion control. Here we adapt it for real-time video bitrate control and rename the new version as *Orca-plus*. More specifically, for a fair comparison with OnRL, we replace Orca's original TCP Cubic [37] and TD3 [34] methods with GCC and PPO, respectively. Orca-plus maintains the same input and output as OnRL, but its NN architecture and training methodology follow the original Orca. Our experiments reveal that Orca-plus shows more overshoots and under-utilization than the GCC, as showcased in Fig. 3 (extensive comparison in Sec. 8.1). Originally applied to congestion control, Orca's gains mostly rely on a large buffer to adjust its hybrid decisions. However, real-time video systems have a more stringent millisecond-level buffer [3], which leaves no time for Orca-plus to execute the hybrid decision. These observations motivate us to design a more pragmatic hybrid learning algorithm for real-time video.

3 SYSTEM OVERVIEW

Fig. 4 illustrates the end-to-end workflow of a real-time video system and our Loki architecture. During a real-time video session, the sender continuously captures video images, encodes them into video frames, which are then packetized and sent along the end-to-end network path. The receiver can reassemble the packets into frames, decode, render and finally playback the video. In practice, the workflow can be bidirectional, enabling interactive video applications.

The QoE of a real-time video largely depends on the sender's bitrate control algorithm, which tries to match the video quality with the instantaneous network bandwidth. However, the modern Internet is highly complex due to the myriads of heterogeneous links, highly dynamic due to a mixture of different traffic patterns and involvement of mobile links. These factors together make bitrate control a non-trivial task, especially under stringent latency constraints.

Unlike existing bitrate control algorithms, Loki is truly hybrid, consisting of two actors operating in parallel: a learning-based RL actor and a rule-based GCC actor, which are integrated into one NN entity through feature level fusion. The features comprise a plethora of NN weights, and preserve the high-level representation ability with respect to how the actors react to underlying network conditions. The deep integration is made possible through "blackboxification" of the rule-based GCC protocol, followed by a dual-attention fusion design.

Benefits of Loki's deep fusion architecture. By deeply fusing the learning- and rule-based actors, Loki attains the fundamental advantages: Loki's dual-attention fusion mechanism can avoid catastrophic decisions through instantaneous hybrid reactions. By nature, the rule-based protocols adopt variants of the additive-increase-multiplicative-decrease (AIMD) adaptation, which is slow in exploration but fast in degradation. When the network condition suddenly drops, the GCC actor is trained to generate a larger attention factor and weigh more in the decision, making the final action more conservative and "safe". On the other hand, as the network recovers, Loki can prioritize the reinforcement learning-based actor which excels at quick exploration. Besides, the inputs/outputs are usually generated at a fine-grained milliseconds level, in order to promptly respond to network dynamics. However, excessive bitrate fluctuation may undermine the user-perceivable QoE [32, 71]. To overcome this problem, Loki designs smooth-related reward functions and training methodologies to smooth both actors' bitrate decisions, and thus the video quality.

System Overview

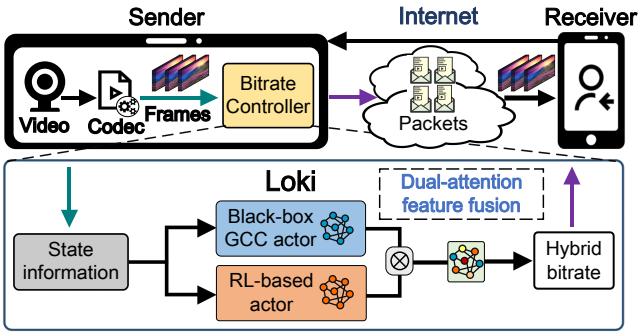


Figure 4: System overview of Loki.

BLACK-BOX RULE-BASED ACTOR

We first clarify the necessities of "blackboxifying" the GCC, and then present the details of our black-box GCC design.

4.1 Why Black-box GCC?

We first illustrate the feature characteristics of rule-based GCC and learning-based protocol in Fig. 5(a). The white-box rule-based GCC is represented by a handful of equations which map each predefined network state to a certain bitrate decision. The mapping is a sparse piecewise function that lacks fine-grained feature representation capability. On the other hand, the learning-based agent is built from delicate neural network structures, and incorporates a feature space with high-level representation ability. Clearly, the white-box version is inherently incompatible with the learning-based NN models at feature level. Although they can be combined at decision level [18, 69], how to multiplex between these two is a non-trivial decision by itself.

In Loki, we choose to fuse the two deeply at feature level. To this end, we make them compatible by transforming the "white-box" GCC into a "black-box" NN model. This equips the black-box GCC with feature representation ability. Now we can deeply integrate the two through a NN structure that captures the advantages of both policies, as illustrated in Fig. 5(b). We can further utilize the fused high-level feature to determine an action, which aims for an optimal decision for the underlying network condition.

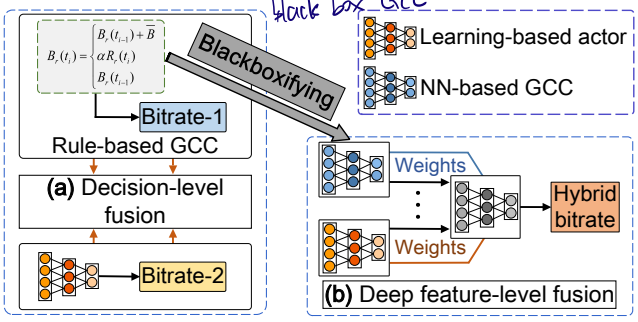


Figure 5: Fusion choices: (a) Decision-level fusion using the original rule-based GCC. (b) Feature-level fusion after blackboxifying the GCC.

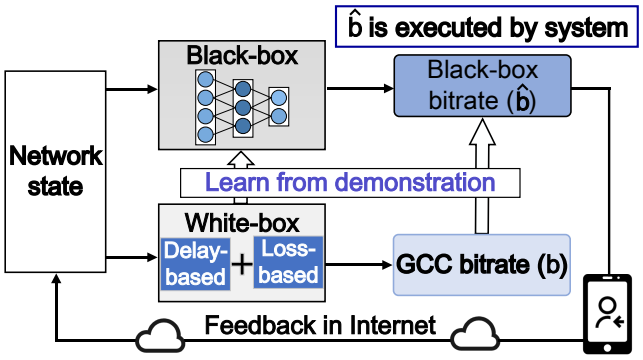


Figure 6: The imitation learning design for blackboxifying GCC.

4.2 Design of Black-box GCC

At first blush, it seems straightforward to derive the black-box version of GCC via simple offline supervised learning. For instance, when running GCC, one can record a set of network states denoted by $\vec{I}$ (involving packet loss $\vec{l}_t$, delay jitter $\vec{j}_t$, *etc.*) and the corresponding optimal bitrate decision denoted by b (*i.e.*, the ground-truth label). Then, the collected dataset can be used to train a NN model in a supervised manner. However, we find that such supervised learning is far from generating a faithful black-box GCC model. The task of blackboxifying the rule-based GCC is fundamentally different from traditional *offline* supervised learning tasks like image classification, where the decision is "one shot", *i.e.*, the output does not impact follow-on task instances. In contrast, our GCC training is actually an *online* sequential decision task, *i.e.*, an output decision will have a direct impact on the next state $\vec{I}_{next}$, and so on.

To meet the challenge, we propose an imitation learning (IL) model [56] which is trained online and can faithfully imitate GCC's behavior. Unlike offline supervised learning, IL can deal with the online sequential decision problem, while preserving the ground-truth labels' demonstration ability. We illustrate the IL training framework in Fig. 6. Specifically, we train the black-box GCC model within the WebRTC framework at runtime, where the black-box GCC and its white-box counterpart run in parallel. The white-GCC generates a GCC bitrate (denoted as b), which is deemed as the expert label. Meanwhile, the black-box GCC will generate a bitrate $\hat{b}$. Then, $\hat{b}$ will be adopted by the WebRTC framework as the target video bitrate. Since the predicted $\hat{b}$ is actually executed, its impact on subsequent states will eventually manifest. We now elaborate on modules that are vital to the black-box GCC's online training, *i.e.*, the inputs, outputs, NN architecture, and customized loss functions.

Inputs and outputs. We specify the inputs and decision interval of the black-box GCC to be consistent with the rule-based counterpart. Within the WebRTC real-time video transport framework, the RTP and RTCP protocols are used to deliver video packets and feedback information, respectively [1]. At each RTCP feedback packet's arrival time t, the packet loss and packet delay jitter statistics [27, 69], denoted by $I_t = (\vec{l}_t, \vec{d}_t)$, are obtained and used as inputs to our model. As for the output, we use the underlying video bitrate options as a base value $\hat{b_u}$, and then specify a list as the black-box GCC's output options, $\mathcal{V}$: $\{0.85, 0.89, 0.96, 0.98, 0.99, 1.0005, 1.001, 1.0015, 1.002, 1.0025\}$, following the original GCC design [27]. Accordingly, the NN model will generate a probabilistic distribution represented as $\mathcal{P}$, and the maximum probability will be chosen as the target via a softmax function [17]. In consequence, the new bitrate $\hat{b^*}$ for the next RTCP interval is calculated in Eq. (1), and the $\hat{b_u}$ will be replaced by the newest $\hat{b^*}$.

$$\hat{b^*} = \mathcal{V}[max(softmax(\mathcal{P}))] \times \hat{b_u} \quad (1)$$

NN architecture for IL. We represent the black-box model with a NN architecture. The input $\vec{I} = (\vec{l}_t, \vec{j}_t)$, is first fed into two separate FC layers with 16 neuron units, and the resulting feature outputs are fused into a high-level feature map that generally encompasses both network loss and delay information. The fusion layer output will then be fed into two consecutive FCs with unit size of 64 and 32, respectively. Finally, an FC with softmax function outputs the probabilistic list of $\mathcal{V}$, which will be deeply fused with RL actor's feature as detailed in Sec. 6.

Weighted loss function and training details. For training the black-box GCC model while ensuring it preserves the same conservative behavior as the rule-based one, we customize the classical cross-entropy loss $\mathcal{L}$ into a weighted function. Specifically, we give a larger weight to $\mathcal{L}$ to penalize overshoots, similar to the scheme in [71]. Besides, we utilize the IL's dagger training loops [56] and adam optimizer [44] to maintain the training effectiveness, with a learning rate of 2.5×10^{-3}, batch size of 64, and LeakyReLU [63] activation function.

5 ADAPTIVE LEARNING-BASED ACTOR

The learning-based actor in Loki is based on PPO [57], a classical reinforcement learning algorithm. PPO is a policy optimization algorithm featuring lower training variance and easy convergence. PPO's potential in real-time adaptation has been validated in recent work [69]. We note that Loki can embrace other RL algorithms or training methodologies for its learning-based actor. In what follows, we proceed to describe the key modules and customized design of the learning-based actor.

RL agent's state and action. We regard Loki's learning actor as an agent that interacts with the network environment and provides instant response, *i.e.*, bitrate action. The state, *i.e.*, input for the RL model, at each RTCP feedback packet's arrival time t, is represented as a list $S_t = (\vec{l}_t, \vec{j}_t, \vec{d}_t, t_t)$, *i.e.*, packet loss, delay jitter, delay, and

the receiving throughput, respectively. The action, *i.e.*, the output of Loki, represents the target video bitrate of the sender. Unlike the wireless link bitrate adaptation, the video bitrate does not have fixed boundaries between different levels. So the bitrate levels are usually set empirically, as in the state-of-the-art video adaptation systems [50, 69]. Following this common practice, the action a_t of Loki, is chosen from a bitrate space with 10 discrete actions empirically set as follows: $\mathcal{A}: \{0.7Mbps, 0.83Mbps, \ldots, 1.87Mbps, 2.0Mbps\}$. The two extreme boundary values, *e.g.*, 0.7 Mbps and 2.0 Mbps, are consistent with those in our commercial real-time video system. Besides, other 8 levels are roughly equally separated in between. We note that the time period of RTCP feedback is usually around 50 ms. While the video codec cannot strictly follow each bitrate update in the millisecond-level granularity, it will take effect over a longer time scale, following the long-term discounted reward mechanism (Eq. (2)) in Loki.

NN structure. Following the state-of-the-art OnRL model [69], we first flatten the sequences S_t, and then feed them into two cascaded FC networks, with 64, 32 units and *tanh* activation [2], respectively. After the simple feature extraction, the intermediate features are fed into an FC layer with 10 units via linear activation, which is used for further attention fusion with GCC's NN-model (More details in Sec. 6).

RL's reward design. The target of RL is to maximize the long-term cumulative rewards. For a start, we adopt the classical *Power* [36] reward function design, defined as $R = throughput/delay$. Intuitively, R reflects the goal of maximizing throughput and minimizing network latency simultaneously. Considering that in real-time video, the network loss and video smoothness are vital to the QoE, we further customize the reward function, by adding the two key metrics: packet loss and video quality smoothness. More specifically, Loki's reward function is represented as follows:

$$\hat{R} = \alpha \times (throughput - loss)/delay - \beta \times |q_n - q_{n-1}| \quad (2)$$

Here, the throughput is measured by the receiver, which directly affects the video quality. The transport-layer packet delay and loss rate indicate the effects of Loki's bandwidth estimation. It is noteworthy that the two metrics, *i.e.*, throughput and loss, are incompatible in terms of unit, but they fall into meaningful range scopes and can be considered as numerics among normalized values, *i.e.*, the value of throughput usually falls within 0.5-3 Mbps, and the loss rate is in the range of 0.1-3%. Thus we combine the two directly in the reward function. q_n and q_{n-1} represent the hybrid agents' underlying and last video bitrates, respectively. Moreover, $|q_n - q_{n-1}|$ is used to enforce the RL agent's bitrate smoothness, so as to avoid large frame jitter [69]. We add two additional factors α, β to normalize these metrics into a consistent range to facilitate model training. We find that the training can converge well when these factors fall in the scope of 5~15 and 0.2~2, respectively. In practical deployment, we set $\alpha = 5, \beta = 0.5$. To enable the RL agent to fully explore link bandwidth and increase its reward when the delay is in a reasonable range, *e.g.*, {min-delay, 10 ms}, we set the delay value to the min-delay of the link in the reward function, so as to make the reward insensitive to small delay variation in the range. The RL-based PPO actor and attention-based feature fusion eventually go through a single training process, which will be described in Sec. 6.2.

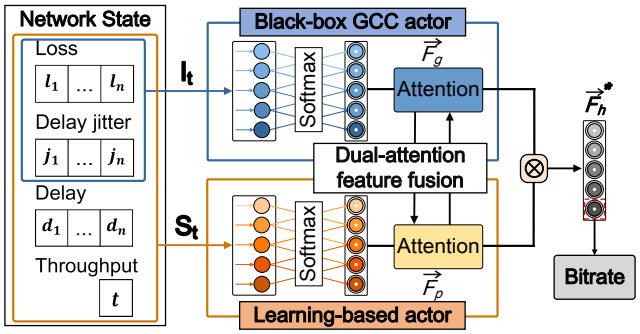


Figure 7: Loki's dual-attention feature fusion.

6 DUAL-ATTENTION FUSION

In this section, we describe Loki's dual-attention feature fusion design and its training methodologies.

6.1 Architecture Design

Attention-based feature fusion has been widely applied in machine learning tasks, particularly in image processing [22, 31, 55, 58]. Generally speaking, it aims to integrate different levels of features (*e.g.*, position and RGB channel in an image) in a self-attention manner, *i.e.*, paying more attention to the highly-weighted feature distilled by the neural network model, thus enhancing the feature extraction ability.

Basic attention fusion design. To realize Loki's principle of leveraging both actors' advantages, we model the hybrid bitrate adaption as an attention fusion task, and further custom-build a dual-attention feature fusion architecture. Fig. 7 shows the detailed workflow. First of all, we have two NN-based actors, *i.e.*, the black-box GCC, and the learning-based PPO actor. When a new network state arrives, the part $I_t = (\vec{l}_t, \vec{j}_t)$, is passed to the NN of the GCC actor, and the part $S_t = (\vec{l}_t, \vec{j}_t, \vec{d}_t, t_t)$ to PPO actor. Then, the two actors perform consecutive feature extraction by their own NN structures as described in Sec. 4 and Sec. 5, and finally output their last layer feature via the softmax operation, represented as $\vec{F}_g$ and $\vec{F}_p$, respectively. In particular, the last layer feature can be considered as a probabilistic list: $\vec{F}_g = [g_1, g_2, \ldots, g_n]$, and $\vec{F}_p = [p_1, p_2, \ldots, p_n]$, where each element represents the probabilistic value of generating one certain bitrate action. For instance, g_i is the probability of choosing the i_{th} bitrate in GCC's output space, *i.e.*, $\mathcal{V}$ list in Sec. 4. To deeply integrate the two actors, we perform feature fusion as follows,

$$\vec{F}_h = \vec{F}_g \otimes \vec{F}_p \quad (3)$$

where the operation $\otimes$ denotes the element-wise multiplication, the most widely used operation in attention fusion [22, 31]. Compared with straightforward element-wise summation and concatenation, $\otimes$ is more discriminative on different features from the two actors [58], which better matches our task. In this way, $\vec{F}_h$ has assimilated the basic hybrid features of the two actors.

Incorporating domain-knowledge into fusion. Beyond the basic dual-attention fusion, we find that deeper customization is required to better handle two unique issues in typical real-time video adaptation, *i.e.*, *(i)* bitrate overshoot and *(ii)* slow-recovery. To this

end, we observe that the learning-based actor often causes overshooting while GCC often generates conservative bitrate actions. On the other hand, the RL actor performs well and can explore quickly when the network condition recovers from a dip, better than GCC's "slow recovery" [71]. Therefore, we further optimize the fusion $\vec{F_h}$ by leveraging the above domain knowledge.

Intuitively, when the network bandwidth is suspected to drop, we enlarge the black-box GCC's attention weights by multiplying a scaling factor. On the other hand, when the network bandwidth witnesses a growing trend, we weaken GCC's weights by appending a sigmoid operation. Formally, we adapt $\vec{F_g}$ as follows,

$$f(\vec{F_g}) = \begin{cases} e^{\beta * \vec{F_g}}, & argmax(\vec{F_g}) \leq \Gamma \\ Sigmoid(\vec{F_g}), & else \end{cases} \quad (4)$$

where β is an empirical parameter set to be 20 in our design, Γ represents the boundary index of NN-based GCC's decreasing factor, *e.g.*, Γ is set to be 4 in GCC's output list $\mathcal{V}$. The impacts of the two empirical parameters will be evaluated in Sec. 8.2. Then, the attentional feature $\vec{F_h}$ in Eq. (3) can be updated as:

$$\vec{\mathbf{F_h}}^* = f(\vec{F_g}) \otimes \vec{F_p} \quad (5)$$

so the final dual-attention bitrate decision should be:

$$Bitrate = \mathcal{A}[argmax(\vec{\mathbf{F_h}}^*)] \quad (6)$$

It is noteworthy that the basic fusion and its adaptation process become completely automatic after proper training. Compared with existing hybrid learning design, *e.g.*, OnRL, Orca, we emphasize that Loki has two advantages: (i) Loki is truly hybrid, consisting of two actors operating in parallel, thus the two actors can more quickly detect the network congestion in feature-level and millisecond-level granularity, better than the second-level detect-and-switch operations as observed in Sec. 2.3. (ii) As the network condition changes, Loki can immediately generate a hybrid bitrate that is co-determined by the two actors' attention feature fusion. The fused bitrate can automatically prevent untamed actions, so it is "safer" and can reduce catastrophic events. We will validate the advantage of Loki's deep "feature-level" fusion over existing "decision-level" fusion through micro-benchmark experiments (Sec. 8.2).

6.2 Training Methodology

In Loki, we adopt PPO's actor-critic architecture [54] to handle the training process. The actor is responsible for selecting an attentional bitrate, while the critic is in charge of evaluating the actor's policy. In Loki, the actor updates the gradient in the direction of increasing the advantage function $adv(\mathcal{A})$. The loss function $\mathcal{L}(\theta)$ is represented as:

$$\mathcal{L}(\theta) = \hat{\mathbb{E}}[min(\delta(\theta)adv(\mathcal{A}), clip(\delta(\theta), 1-\varepsilon, 1+\varepsilon)adv(\mathcal{A}))] \quad (7)$$

where $\delta(\theta) = \frac{p_{\theta_{new}}(s,a)}{p_{\theta_{last}}(s,a)}$, which represents the ratio of a new policy $p_{\theta_{new}}(s,a)$ and its last one $p_{\theta_{last}}(s,a)$. The basic idea of Eq. (7) is to use the $\delta(\theta)$ to avoid the incentive policy updates, by clipping operations when it's out of the interval [1-ε, 1+ε] (here ε is a hyper-parameter, and we empirically set it to be 0.2 [57]). Compared with the method of directly utilizing $p_\theta(s,a)$, $\delta(\theta)$ can make the updated policy smoother in terms of guaranteeing higher video QoE of Loki and simultaneously avoiding gradient oscillation.

On the other hand, the critic network mainly evaluates the actor's policy by generating a value function $\mathcal{V}(s)$, and the training

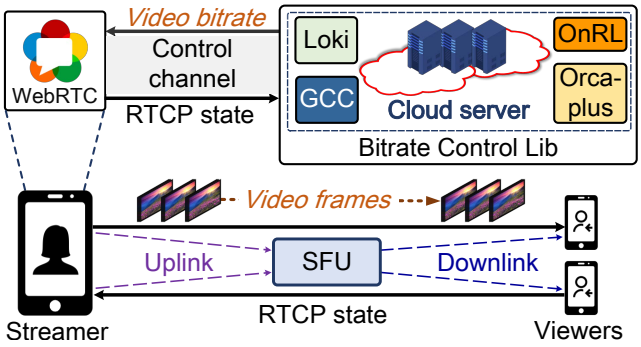


Figure 8: The implementation of Loki and 3 comparison algorithms on an operational system.

effectiveness is evaluated via an advantage function $adv(\mathcal{A})$, as follows,

$$adv(\mathcal{A}) = \hat{\mathcal{R}} + \gamma \mathcal{V}(s_{t+1}) - \mathcal{V}(s_t) \quad (8)$$

where $\hat{\mathcal{R}}$ represents the discount reward related to $\hat{R}$ in Sec. 5. $adv(\mathcal{A})$ will update the NN weights towards the direction where the actor's rewards are better than the expected averaged values $\mathcal{V}(s_t)$. Here γ serves as a decay factor, customized set to be 0.9. More training details about PPO can be found in [57, 69].

Overall training setup. The black-box GCC model is trained using the emulation testbed, and then the model is saved in a frozen *.pb* format, with a model size of 8.6 KB. We directly use the trained black-box model to perform further fusion with the PPO-based actor. The overall model size is 180.2 KB in *.ckpt* format, and will take up an average of 3.83% of 16-GB memory while training. In order to effectively learn and respond to the real network conditions in suit [65, 69], we train the dual-attention fusion network following an online learning manner, where the training is located in a server cluster related to a commercial video system (details in Sec. 7). In particular, when each session starts, the user loads a pre-trained model for fast adaptation with underlying network condition, and further trains its personalized model over time as the video session continues. In essence, Loki behaves like the online learning algorithm OnRL [69], which evolves along with the dynamic network variance and keeps optimizing its model during the whole training lifecycle.

7 IMPLEMENTATION AND DEPLOYMENT

7.1 Implementation on a Commercial Real-time Video System

Fig. 8 shows the implementation workflow of Loki. We integrate Loki into a mainstream commercial real-time video APP, which is used for e-commerce live shows and virtual interactive shopping by several hundred million users worldwide. The APP consists of the uplink, *i.e.*, from the live video streamer to a Selective Forwarding Unit (SFU) node [9], and downlink, *i.e.*, from SFU to the viewers. The uplink relies on the Real-Time-Communication (RTC) protocols, which are the optimization target of this work. The downlink can be considered as a live video content distribution system where each video frame can be redistributed by multiple SFUs. However, unlike VoD streaming servers, the client of viewer has a relatively short buffer lasting several hundred milliseconds (up to 1 second),

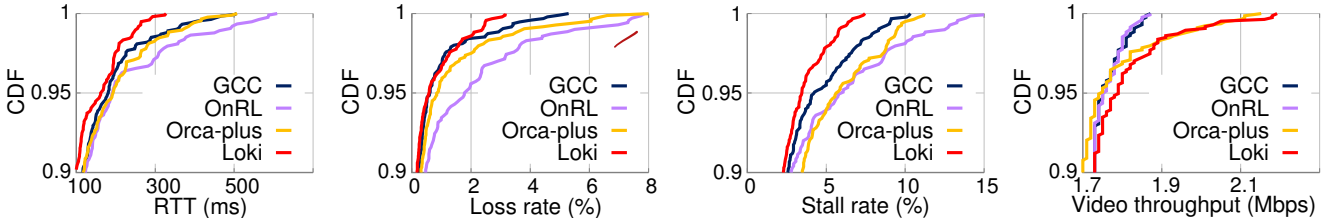


Figure 9: Loki significantly improves video QoE and solves the long tail issue.

to ensure the video frames can be viewed in real-time.

Loki is specifically designed for APP's uplink, which follows the same architecture as real-time video telephony systems. Specifically, the uplink builds upon WebRTC [5], an RTP-based real-time streaming framework, consisting of full-fledged bitrate controller modules (default is GCC) and video codec. Nonetheless, Loki can benefit the downlink and end-to-end APP video quality as well, as we will verify in Sec. 8.1. Our implementation substitutes the built-in bitrate controller of APP with state-of-the-art solutions: {Loki, OnRL, Orca-plus, GCC[4]}, which comprise a bitrate controller library. Ideally, these solutions should be deployed within the mobile app on the streamers' side. However, we find that many mobile devices do not support on-device training, which is crucial for the online learning of Loki. In addition, the mobile devices are constrained by their energy and CPU resource budget. Therefore, we make a compromise by deploying the bitrate controller on a cloud server cluster. During each video session, each sender (*i.e.*, the streamer side APP) maintains a control channel with a cloud server. The sender collects detailed network states (*i.e.*, RTCP feedback) from the receiver and sends them to the server. Then the server immediately outputs a bitrate action and returns this decision to the sender, which is used as the next sending bitrate. The control channel is extremely lightweight and consumes only about 1.5 KB per second on average. To accommodate massive scale evaluation and guarantee low latency, we configure 20 cloud servers with built-in service scheduling and load balancing functions. Each server is equipped with 65 GB memory and the RedHat Linux versions of 4.8.5, and can support the online training of at least 50 senders simultaneously.

In Loki's real-world deployment, we also implement a backup mechanism, *i.e.*, letting the APP sender automatically fall back to the default GCC controller upon connection failure happening between the sender APP and server cluster, so as to guarantee users' worst-case experience at runtime. Besides, we note that two key modules within Loki's workflow will cause latency issues. One is the link between the APP sender and the viewer side, and the other is the link between APP streamer/sender and cloud server. The former has a high possibility of generating relatively high RTT, especially in highly dynamic networks. We will demonstrate Loki's robustness over such networks in Sec. 8.1. The latter latency is usually around 10 ms, which will have negligible impacts on real-time video transmission, as corroborated in OnRL [69].

7.2 Large-scale Deployment

We deploy Loki on the APP live video system, and setup a large-scale field test involving 392 streamers across 16 countries and 156 cities. Tab. 2 summarizes our deployment statistics. Most streamers perform at least one live video session each day, and each live video has about 3000 real-time viewers on average. The field trial lasted over 3 months (from Jan. 10 to Mar. 16, 2021), and generated about 101 million video sessions. We filter out the video sessions shorter than 2 seconds because they do not represent any meaningful viewing experience for QoE evaluation. The final evaluation involves over 73 million sessions with a total playback time of 1.316 million hours (150 years long).

Table 2: Deployment scale in a commercial real-time video system over 3 months.

Algorithms	Live video counts	Sessions (all)	Sessions ≥ 2s	Avg. live time (h)	Total play time (h)
GCC	2517	34,569,487	25,651,556	2.77	417175
OnRL	2128	22,217,624	15,929,805	2.73	325,569
Orca-plus	2022	21,934,513	15,829,700	2.64	286,700
Loki	2047	22,913,621	16,234,772	2.58	286,565
Overall	8714	101,635,245	73,645,833	2.68	1,316,009

For a fair comparison, we randomly divide the 392 streamers into 4 equal-sized groups. In each round of trials (lasting 24 hours), we assign the 4 solutions in the bitrate control lib to the 4 groups. In the next round, we randomly shuffle the assignment. After multiple rounds, each streamer will experience all the 4 bitrate controllers, in a statistically fair manner. Under such setup, each bitrate control algorithm experiences over 15 million sessions, and each live video session lasts more than 2 hours.

8 EVALUATION

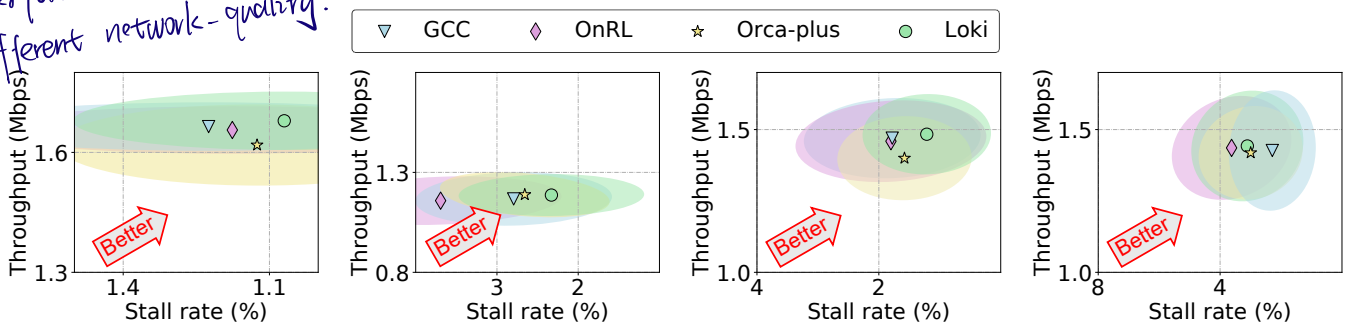
In this section, we evaluate Loki from two aspects: *(i)* We demonstrate Loki's advantages over the 3 baselines through large scale field tests (Sec. 8.1). *(ii)* We conduct a deep dive analysis for the two key design modules, *i.e.*, black-box GCC and the dual-attention based fusion, so as to understand the effectiveness of Loki's hybrid feature-level fusion (Sec. 8.2).

8.1 System-level Evaluation in The Wild

Evaluation metrics. Loki mainly controls the streamer-to-SFU network path which runs the WebRTC-based real-time video protocol stack, so we pay more attention to metrics along this path, *i.e.*, *packet loss*, *RTT* and *throughput* on the transport layer, and *PSNR*[5], *fps*, *stall rate* (the fraction of fps<12) on the application layer. Besides, we also log the viewer-side metrics, including *average playback time*,*stalling count per hour* (*i.e.*, counting the occurrences of

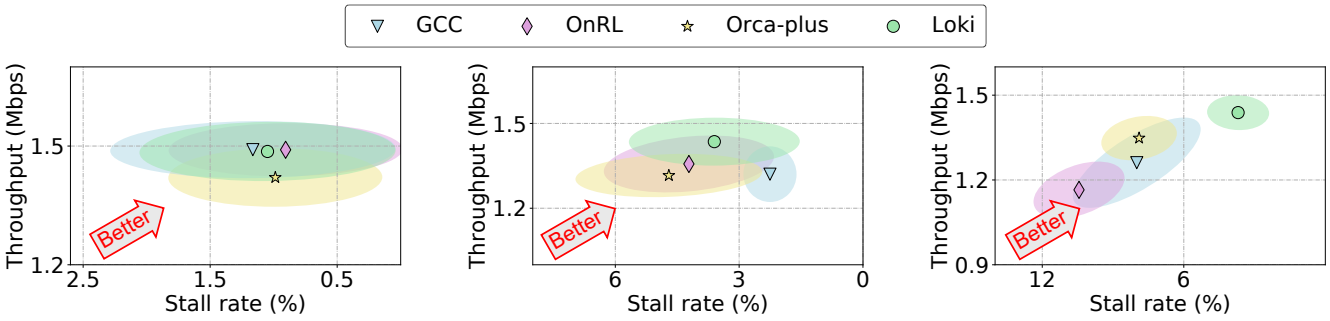
[4]We note that the GCC that we used is actually not the vanilla GCC in [27], but an optimized GCC: GCC-β [62] aiming to optimize GCC's relatively conservative bitrate control policy.

[5]The PSNR is indicated by a metric named quantization parameter (QP) that represents the frame quality [20].



(a) High-quality (57.39% traces) (b) Low-quality (42.61% traces) (c) WiFi (80.38% traces) (d) 4G (19.62% traces)

Figure 10: Loki achieves consistent QoE performance over diverse network conditions.



(a) RTT ≤ 100 ms (b) 100 ms < RTT < 200 ms (c) RTT ≥ 200 ms

Figure 11: Loki achieves better QoE performance across different RTT regimes.

Table 3: QoE comparison statistics.

Metrics	Stall rate (fps<12)	Stall rate (fps<5)	Video th. (Mbps)	PSNR	Fps
GCC	1.86	0.40	1.46	31.02	**19.47**
OnRL	2.20	0.77	1.45	30.63	19.45
Orca-plus	1.89	0.56	1.40	30.83	16.45
Loki	**1.60**	**0.35**	**1.48**	**31.06**	19.46

an empty playback buffer), *low video quality count per hour* (*e.g.*, fps is smaller than 5.), *frame jitter*[6], and *maximized frame jitter*. These metrics are also used internally by the commercial APP's engineering team to monitor the QoE.

Loki achieves the maximum QoE. We summarize the QoE-related metrics over all field tested sessions in Tab. 3. We observe that, *(i)* Loki achieves remarkable gains in terms of stall rate, *e.g.*, 13.98%, 27.27%, 15.34% reduction than GCC, OnRL and Orca-plus, respectively. Meanwhile, its video throughput improves by 1.37%, 2.07%, 5.71%, and PSNR by 0.13%, 1.40%, 0.75%, with nearly the same averaged fps. *(ii)* More importantly, Loki remarkably reduces the fraction of catastrophic events. In terms of extremely low fps (< 5) which indicates video freezing [6], it achieves a reduction by 54.55% and 37.5% compared with OnRL and Orca-plus, respectively. The result validates the advantage of Loki's "feature-level" fusion mechanism, in comparison to the "time multiplexing".

Loki significantly cuts the performance tail. We analyze the tail distribution of diverse metrics in Fig. 9. Clearly, Loki significantly alleviates the long-tailed QoE effects, especially when compared to the existing learning-based solutions OnRL and Orca-plus. In particular, Loki significantly decrease the 95-percentile of RTT and loss rate by 13.26% and 67.37% over OnRL, and 14.17%, 31.35% over Orca-plus. Moreover, it even outperforms the rule-based GCC by 6.96% and 1.84%. The tail-cutting gains at transport layer transform into higher QoE gains at application layer. For instance, Loki decreases the 95-percentile tails of stall rate by 26.30%, 44.24%, 41.39%, improves tail video throughput by 2.17%, 1.76%, 2.03%, compared with GCC, OnRL and Orca-plus, respectively.

Loki performs consistently over diverse networks. We further conduct a breakdown analysis to show Loki's performance across different network environments, *e.g.*, high quality paths (throughput is higher than the averaged value) vs. low quality paths, WiFi vs. 4G, and 3 kinds of RTT regions (*e.g.*, 0-100ms, 100ms-200ms, and more than 200ms). We plot the averaged QoE and the corresponding *std.* in Fig. 10, and Fig. 11. We observe that: *(i)* Loki maintains the best QoE performance over both high quality and low quality networks. It effectively decreases the average stall rate, by 12.66%, 9.06%, and 4.95%, relative to GCC, OnRL, and Orca-plus in high quality paths. Moreover, its advantage becomes more prominent for low quality paths, *i.e.*, 16.63%, 36.92%, and 12.33% reduction. *(ii)* Loki has relatively consistent high performance in both WiFi and 4G networks. Other baseline algorithms' performance dramatically varies across two network conditions. For instance, GCC performs well in terms of stall rate in 4G, but degrades remarkably in WiFi. Orca-plus performs reasonably well in 4G networks, but has drastically low video throughput in WiFi. In contrast, Loki has the lowest stall rate and highest throughput in WiFi. Note that WiFi access takes up a dominating fraction of 80.38% of the total

[6]Frame jitter reflects network congestion: larger frame jitter indicates stalling event.

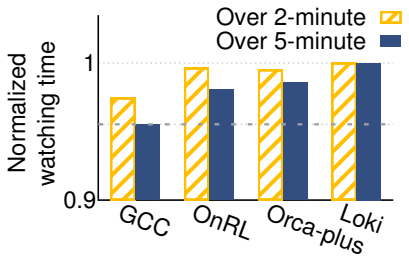


Figure 12: Loki's viewers watch for longer time.

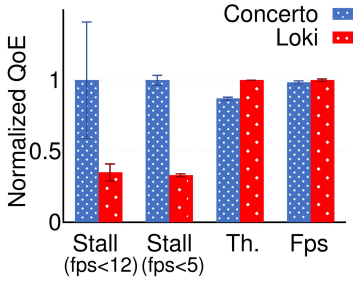


Figure 13: Loki performs better than Concerto.

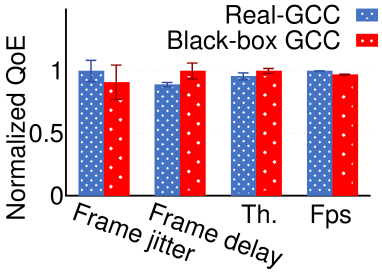


Figure 14: Black-box and real GCC show highly consistent QoE.

sessions. On the other hand, in 4G networks, Loki's stall rate is very similar to the most competent learning-based Orca-plus (only 0.11% deviation). These results clearly validate Loki's robustness which is attributed to its hybrid learning mechanism. *(iii)* Loki achieves the best QoE over the larger RTT region (*e.g.*, more than 200ms), *e.g.*, 6.68% gain in video throughput, and 53.16% decrease in stall rate than the most competent scheme Orca-plus. Note that larger RTT usually indicates higher network dynamics, larger buffer size or longer path length [25, 52]. The result again verifies Loki's ability to cut the long-tail.

Loki benefits end viewers indirectly. We collected viewer side performance metrics, and find that Loki also outperforms its counterparts on most of the QoE-related metrics: stalling counts per hour, low frame-quality counts per hour, and maximum frame jitter exhibit 0.79%-8.04%, 0.63%-9.66%, 1.72%-11.63% reduction than the other three methods. The frame jitter of Loki is larger (3.21 ms) than the best competitor (GCC), but is 14.65 ms larger than the best (Orca-plus). Overall, the few ms reductions of frame jitter are unlikely to disturb the QoE of viewers, because the client side has up to 1-second of buffer to smooth the slight frame instability.

In addition, we also examine the viewers' watching time, as shown in Fig. 12. Loki extends the mean watching duration by 2.62%, 0.37%, 0.5% in comparison to GCC, OnRL, and Orca-plus among all >2 minute sessions, and 4.68%, 1.97%, 1.44% among all >5 minute sessions. The result indicates Loki can deliver a higher quality viewing experience for users. Note that even a mild gain in viewing time can bring huge impacts on commercial systems that serve millions of users [49].

We emphasize again that the benefit of Loki in the current commercial APP is more prominent along the streamer-to-SFU path which runs a real-time video transport. The SFU-to-viewer downlink is not purely controlled by Loki. Instead, it adopts a CDN-like architecture, as in non-real-time VoD streaming systems. The SFU servers act as the video distribution hubs and perform extra actions such as frame buffering and skipping. The ongoing development of APP is trying to rebuild the downlink using WebRTC just like the uplink. As this materializes, we expect the viewer side performance will be boosted to another level.

Comparing with Concerto. We further add a testbed experiment to compare Loki with a recently developed offline learning-based algorithm, Concerto [71]. We run both algorithms over 5 randomly selected traces from the real-time video APP, lasting about 5 hours in total, and illustrate their normalized QoE-related metrics in Fig. 13. We can find that Loki achieves consistently better performance than Concerto, not only the stall rate (fps <12) is reduced by 65%, *i.e.*, 1.84% reduction in the absolute value, but also the video quality is higher, *e.g.*, 13% improvements on video throughput, and 1.5% gains in fps. More surprisingly, Loki also remarkably reduces the fraction of low fps (<5) counts by 67%. These results further corroborate the robustness of Loki.

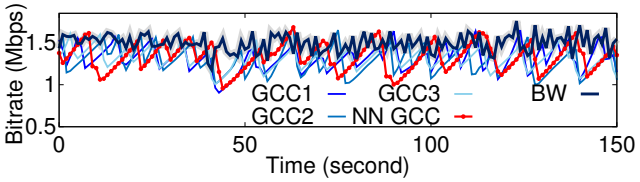


Figure 15: NN-based black-box GCC imitates GCC's behaviors faithfully.

8.2 Loki Deep Dive

In this section, we conduct micro-benchmark experiments in the emulation testbed as detailed in Sec. 2, to study the contribution of different design modules in Loki. It is noteworthy that the local testbed also contains a cloud-based structure similar to the real-world deployment, where we deploy the bitrate controller lib and detailed controlled algorithms.

The cloning effect of black-box GCC. Now we demonstrate that the black-box GCC faithfully clones and can replace the original white-box counterpart, so as to facilitate Loki's hybrid architecture. In particular, we train a black-box model using 20 hours of randomly selected network traces collected from APP. As a sanity check, we run both models over another 3 randomly selected network traces unseen in the training. We repeat each experiment 3 times, and then record the application layer QoE metrics. From Fig. 14, we can observe that the black-box GCC shows nearly the same performance as its white-box counterpart, *e.g.*, similar throughput and fps, and only slight deviation in frame jitter (40 ms vs. 36 ms) which makes no difference at the user end.

Fig. 15 further showcases the instantaneous bandwidth estimation of a black-box GCC vs. the white-box GCC repetitively running 3 times on the same sample network trace. We make two observations: *(i)* The black-box GCC behaves like the vanilla GCC, *i.e.*, exhibiting fast degradation and slow recovery, which demonstrates that it has mastered the white-box's decision-making mechanism through imitation learning. *(ii)* Although the bandwidth estimation curves are not perfectly aligned, these are expected randomness, because even if we repeat the white-box GCC on the same network trace, the output time series do not coincide perfectly. As illustrated

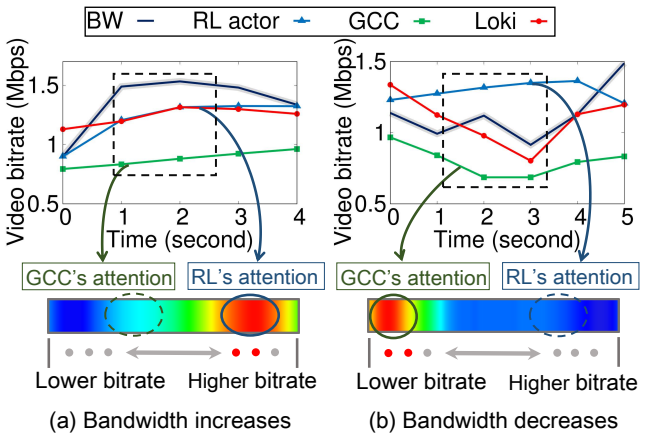


Figure 16: Looking into Loki's attention mechanism.

in Fig. 15, the bitrate prediction gap among 3 repetitions GCC1, GCC2, and GCC3 is 0.145 Mbps on average, which is comparable with the gap between them and the black-box GCC, *i.e.*, 0.156 Mpbs.

Understanding Loki's dual-attention fusion. We now demystify the operation details of Loki's dual-attention fusion model, so as to understand the reasons behind its robustness against various network conditions. We showcase and visualize two typical network condition changes in Fig. 16, where we make the following observations: *(i)* In Fig. 16(a), when the bandwidth increases, as highlighted in the dotted box, GCC generates overly conservative bitrates, whereas the RL-based actor performs quick exploration to recover. The attention heatmap at the bottom of the figure demonstrates that the current attention (the "red zone") focuses on the "higher bitrate" region, *i.e.*, the dual-attention fusion mechanism automatically prioritizes the RL agent's decision. *(ii)* In the second scenario, as shown in Fig. 16(b), when the bandwidth suddenly drops, GCC decreases its bitrate quickly in a multiplicative manner, while the RL actor still maintains its high bitrate estimations for a while. In this case, Loki automatically generates a bitrate that is closer to GCC's behavior, *i.e.*, the focused attention red zone lies in the lower bitrate region. To summarize, Loki's dual-attention fusion mechanism can agilely gauge the underlying network condition variations, and generates a reliable hybrid bitrate decision to best match the network condition, thus preventing the occasional but fatal overshooting/underestimation behavior.

Ablation study. We further run an ablation experiment to demonstrate the benefit of Loki's attention-based fusion learning, when comparing it against three mainstream fusion mechanisms. For fair comparisons, we only modify the fusion design, and keep other parts unchanged. *(a)* "Average fusion": using the average of the two actors' bitrate decisions as the finial bitrate; *(b)* "Switch fusion": switching between two actors, *e.g.*, using one actor for some time and then switch to the other actor upon the detection of abnormal outcome. A representative solution is OnRL [69], which we implement in this experiment; *(c)* "Early fusion": concatenating two actors' features in the early learning process, and then continuing to distill the joint features by cascading neural network layers; *(d)* Loki's attention-based late-fusion manner. We verify the QoE performance of the above 4 methods under the same 1-hour network trace, and then plot 3 normalized QoE metrics in Fig. 17. We observe that, *(i)* Generally, "feature-level" fusion mechanisms, *i.e.*, the early fusion and Loki's attentional late fusion, outperform the two "decision-level" fusion designs, *i.e.*, Average and Switch. The Average scheme is merely a hard combination between the incompatible actors and does not account for the fine feature characteristics. As for the Switch fusion, only one actor runs at any time, thus Switch cannot exploit the benefit from both actors simultaneously, *e.g.*, the in-use RL actor cannot detect abnormal behaviors in time and causes a lagging effect in its decision as observed in Sec. 2.3. *(ii)* For the two "feature-level" designs, the early fusion shows 52.71% lower throughput and 66.67% higher frame jitter than Loki. Since the two actors have different inputs, it is ineffective to fuse the two incompatible inputs at the beginning of feature extraction.

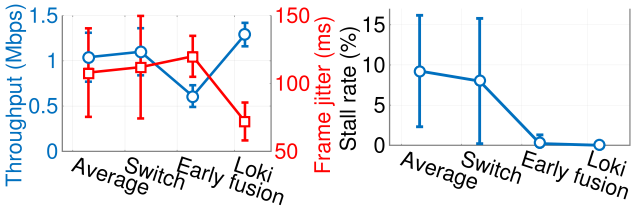


Figure 17: Ablation study of Loki.

Table 4: Impact of the attentional parameters of Loki.

Metrics	β_1	β_2	β_3	Γ_1	Γ_2	Γ_3
Stall rate (%)	2.23	0.0	**0.0**	2.67	0.0	1.20
Video Th. (Mbps)	1.12	1.14	**1.28**	1.16	1.29	1.14

Impact of the attentional parameters of Loki. We examine Loki's attention fusion with different parameter configurations in Eq. (4), including the enlarged ratio of β (*i.e.*, $\beta_1 = 1$, $\beta_2 = 10$, $\beta_3 = 20$), and the threshold Γ (*i.e.*, $\Gamma_1 = 3$, $\Gamma_2 = 4$, $\Gamma_3 = 5$). The results in Tab. 4 demonstrate that Loki performs better for larger β, *i.e.*, both β_2 and β_3 lead to zero stalling. Note that even larger β will make the array of $e^{\beta * \vec{F_g}}$ in Eq. (4) out of bounds when NN training. In addition, the stall rate increases as Γ decreases from 4 to 3, and $\Gamma_3 = 5$ leads to larger stalling as it is larger than the bitrate variance boundary of 4 in GCC's output list $\mathcal{V}$. Besides, we find that the throughput is less sensitive to β and Γ. In all our field tests and micro-benchmarks, we have adopted $\beta = 20$, $\Gamma = 4$ as the default configurations.

9 RELATED WORK

Real-time video transport is driving many essential applications in the digital society. Unlike the VoD streaming with multiple seconds of playback buffer [19], real-time video imposes a much more stringent latency requirement, around few hundreds of milliseconds only [3]. A plethora of research has been devoted to optimizing the QoE of real-time video with the same design goal, *i.e.*, designing a bitrate controller that generates appropriate video resolution and frame rate to match the instantaneous network conditions. The proposed approaches generally fall into two categories: *(i)* classic rule-based design, and *(ii)* contemporary learning-based adaptation algorithms. The rule-based algorithms commonly use the packet loss and delay metrics [26, 27, 30, 53, 64, 68] as the

key indicators of network condition, and adapt based on heuristic AIMD-like algorithms. Notwithstanding, a universal set of prescribed rules can hardly fit the increasingly heterogeneous modern Internet [50, 60, 70]. To overcome such limitations, learning-based designs have emerged in recent years. To name a few, Concerto [71], OnRL [69], LiveNAS [43], and NEMO [67] are designed to optimize live video streaming, telephony and broadcasting services. Pensieve [50], ABRL [49], and Puffer [65] target VoD streaming through content delivery networks (CDNs). These algorithms commonly customize a deep learning model, particular the widely-used reinforcement learning, for video bitrate control.

Recently, the idea of hybridizing rule-based and learning-based protocols is gaining traction. For instance, Orca [18] proposes to regularly alternate between rule-based and learning-based decisions to improve traditional TCP performance. OnRL [69] switches to a rule-based protocol when its learning agent detects abnormal QoE. Stick [40] is designed to improve the buffer-based ABR QoE in VoD streaming by adjusting its buffer bounds, by fusing an RL-based model with the traditional buffer-based method at "decision-level", instead of the "feature-level". The limitations of such a time-multiplexing design are evident in our experimental results (Sec. 2). In contrast, Loki represents a deeper fusion scheme, allowing the two rules to work in sync to co-determine a bitrate that safely explores the network bandwidth, without causing long-tail glitches.

Fusion learning has become a key ingredient in many deep learning tasks, *e.g.*, image classification [23], object detection [48], and scene segmentation [33], which usually assembles diverse levels of features from different branches or layers. In general, fusion learning models follow either an early fusion or late fusion architecture. Early fusion usually operates through simple concatenation or summation operations over the original input features, and late fusion executes directly on the last feature layer. Recent years, another kind of feature fusion based on attention mechanism is attracting much interest. These algorithms mimic human attention behaviors [28, 31, 46], by adaptively integrating different local features stemmed from the same input, *e.g.*, position attention and channel attention. Unlike prior work that targeted video content processing, Loki adapts the attention mechanism to enable hybrid decision making between rule-based and learning-based models. Our design may be extended to tackle the long-tail performance issues of a more broad class of learning-based network protocols.

10 DISCUSSION

Generalization of Loki. Loki's "feature-level" integration provides a framework of combining the advantages of different video transport methods. As a first instance, our Loki design realizes fusion between the rule-based GCC and RL-based PPO. Nevertheless, the framework can be used to integrate any other rule-based algorithms, *e.g.*, Cubic [37], BBR [26], PCC-family [29, 30, 53], and any RL agent algorithms, *e.g.*, A3C[54], DDPG [47], SAC[38], and TD3 [34]. The key lies in blackboxifying the rule-based algorithms faithfully, which is consisting of two steps: *(i)* Collecting and inducting the input signal and decision information of the rule-based algorithms as the NN models' inputs and outputs, separately. *(ii)* Customizing a NN structure based on the rule-based protocol's traits, and then training the model in the practical system. Afterwards, we can adopt Loki's dual-attention mechanism to perform "feature-level" fusion.

On-device training of Loki. Loki currently relies on cloud servers to support the training, due to the lack of on-device light-weighted NN training API, and the concern of consuming much of the end device's limited resource (Sec. 7). However, as more mobile devices are equipped with specialized neural processing units (NPUs), neural network training on mobile devices will be pragmatic soon [59]. As future work, we will develop an on-device hybrid learning framework, and transform Loki's cloud-based training to on-device learning, so as to eschew the overhead of deploying training servers.

Other design spectra. In Loki, we have tried blackboxifying a rule-based algorithm in order to fuse it with a learning-based model. There exists alternative design spectra, *e.g.*, whiteboxifying the NN model and fusing it with the rule-based algorithm. However, it is hard to whiteboxify an RL model, compared with supervised learning models. More importantly, even though we have a whiteboxified RL model, it can only be integrated with the rule-based protocol at the "decision-level", since there are no hidden features in the whiteboxified models. On the other hand, safe reinforcement learning algorithms have been explored in recent work [21, 35, 51], which aim at handling the safety problems in the RL's exploration phase. At first blush, the safe RL may solve the robustness issue in Loki's real-time video transmission. However, we find that existing safe RL algorithms [21, 51] act like a "decision-level" fusion policy, *e.g.*, letting a shielder correct the action when the chosen action is regraded to possibly cause performance degradation. As we have verified in Fig. 17, the "decision-level" fusion mechanism is not as good as Loki's "feature-level" fusion.

11 CONCLUSION

We have designed and evaluated Loki, a robust hybrid solution that integrates the learning-based and rule-based real-time video adaptation algorithms through deep "feature" level, instead of "decision" level fusion. Real-world deployment and massive field evaluation demonstrate that Loki can substantially improve the tail performance of state-of-the-art learning-based algorithms. In this way, Loki addresses the common concern on the reliability of learning-based algorithms, thus paving the way for their deployment in commercial real-time video systems. We believe the design paradigm of fusing rule-based and learning-based algorithms at feature level can help enhance the robustness of general data-driven network protocols.

ACKNOWLEDGMENTS

We appreciate the insightful feedback from the anonymous shepherd and reviewers 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 Youth Top Talent Support Program, the BUPT Excellent Ph.D. Students Foundation (CX2020204) and AIR Program.

REFERENCES

[1] 2005. RTP, RTCP and RTMP: Internet protocol for Real-time Multimedia Communication. https://www.cse.wustl.edu/~jain/books/ftp/rtp.pdf.

[2] 2017. An overview of activation functions used in neural networks. https://adl1995.github.io/an-overview-of-activation-functions-used-in-neural-networks.html.

[3] 2019. Intro To Live Streaming Latency. https://www.boxcast.com/blog/live-stream-video-latency.

[4] 2019. The minimum internet speeds for streaming video in 2019. https://www.zen.co.uk/blog/posts/zen-blog/2019/04/18/the-minimum-internet-speeds-for-streaming-video-in-2019.

[5] 2019. WebRTC Chrome Report Notes. https://webrtc.github.io/webrtc-org/release-notes/.

[6] 2019. What Is Frame Rate (Fps) And Why Does It Matter In Live Streaming? https://www.boxcast.com/blog/what-is-frame-rate-fps-and-why-does-it-matter-in-live-streaming.

[7] 2020. 10 Massive Applications Using WebRTC. https://bloggeek.me/massive-applications-using-webrtc/.

[8] 2020. Consumer Internet Data Traffic Worldwide By Application Category From 2016 To 2022. https://www.statista.com/statistics/454951/mobile-data-traffic-worldwide-by-application-category/.

[9] 2020. How Different WebRTC Multiparty Video Conferencing Technologies Look Like on the Wire. https://testrtc.com/different-multiparty-video-conferencing/.

[10] 2020. Isolate Together: 6 Platforms to Help You Connect Online. https://www.everythingzoomer.com/style/2020/12/15/gift-ideas-for-the-animal-lover/.

[11] 2020. Reasons of eCommerce Video Can Help Sell Online. https://demoduck.com/blog/ecommerce-video-online-sales/.

[12] 2020. Robotics, Smart Wearable Technologies, and Autonomous Intelligent Systems for Healthcare. https://onlinelibrary.wiley.com/doi/full/10.1002/aisy.202000071.

[13] 2020. TC control-Linux page. https://man7.org/linux/man-pages/man8/tc-tbf.8.html.

[14] 2020. WebRTC Home. https://webrtc.org/.

[15] 2021. Interactive Video Wall Market Research Report by Vertical, by Offerings, by Region. https://www.researchandmarkets.com/reports/4896751/interactive-video-wall-market-research-report-by.

[16] 2021. Introduction to Loki. https://en.wikipedia.org/wiki/Loki.

[17] 2021. Softmax Function-Deep AI. https://deepai.org/machine-learning-glossary-and-terms/softmax-layer.

[18] Soheil Abbasloo, Chen-Yu Yen, and H. Jonathan Chao. 2020. Classic Meets Modern: a Pragmatic Learning-Based Congestion Control for the Internet. In *SIGCOMM '20: Proceedings of the 2020 ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication, Virtual Event, USA, August 10-14, 2020*. 632–647.

[19] 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.

[20] Hamed Ahmadi Aliabad, Sandro Moiron, Martin Fleury, and Mohammed Ghanbari. 2010. No-Reference H.264/AVC Statistical Multiplexing for DVB-RCS. In *Personal Satellite Services - Second International ICST Confernce, PSATS 2010, Rome, Italy*, Vol. 43. Springer, 163–178.

[21] 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), New Orleans, Louisiana, USA, February 2-7, 2018*. 2669–2678.

[22] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural Machine Translation by Jointly Learning to Align and Translate. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

[23] Sean Bell, C. Lawrence Zitnick, Kavita Bala, and Ross B. Girshick. 2016. Inside-Outside Net: Detecting Objects in Context with Skip Pooling and Recurrent Neural Networks. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*. IEEE Computer Society, 2874–2883.

[24] Per Block, Marion Hoffman, Isabel J Raabe, Jennifer Beam Dowd, Charles Rahal, Ridhi Kashyap, and Melinda C Mills. 2020. Social network-based distancing strategies to flatten the COVID-19 curve in a post-lockdown world. *Nature Human Behaviour* (2020), 1–9.

[25] Thiago B Cardozo, Ana Paula C da Silva, Alex B Vieira, and Artur Ziviani. 2014. Bufferbloat systematic analysis. In *2014 International Telecommunications Symposium (ITS)*. IEEE, 1–5.

[26] Neal Cardwell, Yuchung Cheng, C. Stephen Gunn, Soheil Hassas Yeganeh, and Van Jacobson. 2017. BBR: congestion-based congestion control. *Commun. ACM* 60, 2 (2017), 58–66.

[27] 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.

[28] Yimian Dai, Fabian Gieseke, Stefan Oehmcke, Yiquan Wu, and Kobus Barnard. 2020. Attentional Feature Fusion. *CoRR* abs/2009.14082 (2020).

[29] Mo Dong, Qingxi Li, Doron Zarchy, Philip 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, Oakland, CA, USA, May 4-6, 2015*. 395–408.

[30] 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.

[31] Deng-Ping Fan, Wenguan Wang, Ming-Ming Cheng, and Jianbing Shen. 2019. Shifting More Attention to Video Salient Object Detection. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*. 8554–8564.

[32] 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.

[33] Jun Fu, Jing Liu, Haijie Tian, Yong Li, Yongjun Bao, Zhiwei Fang, and Hanqing Lu. 2019. Dual Attention Network for Scene Segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*. 3146–3154.

[34] Scott Fujimoto, Herke van Hoof, and David Meger. 2018. Addressing Function Approximation Error in Actor-Critic Methods. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholm, Sweden, July 10-15, 2018*. 1582–1591.

[35] Javier García and Fernando Fernández. 2015. A comprehensive survey on safe reinforcement learning. *J. Mach. Learn. Res.* 16 (2015), 1437–1480. http://dl.acm.org/citation.cfm?id=2886795

[36] Alfred Giessler, J. D. Haenle, Andreas König, and E. Pade. 1978. Free Buffer Allocation - An Investigation by Simulation. *Comput. Networks* 2 (1978), 191–208.

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

[38] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. 2018. Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholm, Sweden, July 10-15, 2018*. 1856–1865.

[39] Bo Han, Yu Liu, and Feng Qian. 2020. ViVo: visibility-aware mobile volumetric video streaming. In *MobiCom '20: The 26th Annual International Conference on Mobile Computing and Networking, London, United Kingdom, September 21-25, 2020*. ACM, 11:1–11:13.

[40] Tianchi Huang, Chao Zhou, Rui-Xiao Zhang, Chenglei Wu, Xin Yao, and Lifeng Sun. 2020. Stick: A Harmonious Fusion of Buffer-based and Learning-based Approach for Adaptive Streaming. In *39th IEEE Conference on Computer Communications, INFOCOM 2020, Toronto, ON, Canada, July 6-9, 2020*. 1967–1976.

[41] Ahmed Hussein, Mohamed Medhat Gaber, Eyad Elyan, and Chrisina Jayne. 2017. Imitation Learning: A Survey of Learning Methods. *ACM Comput. Surv.* 50, 2 (2017), 21:1–21:35.

[42] Werner Alexander Isop, Christoph Gebhardt, Tobias Nägeli, Friedrich Fraundorfer, Otmar Hilliges, and Dieter Schmalstieg. 2019. High-Level Teleoperation System for Aerial Exploration of Indoor Environments. *Frontiers Robotics AI* 6 (2019), 95.

[43] Jaehong Kim, Youngmok Jung, Hyunho Yeo, Juncheol Ye, and Dongsu Han. 2020. Neural-Enhanced Live Streaming: Improving Live Video Ingest via Online Learning. In *SIGCOMM '20: Proceedings of the 2020 ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication, Virtual Event, USA, August 10-14, 2020*. ACM, 107–125.

[44] Diederik P. Kingma and Max Welling. 2014. Auto-Encoding Variational Bayes. In *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*.

[45] Kyungjin Lee, Juheon Yi, Youngki Lee, Sunghyun Choi, and Young Min Kim. 2020. GROOT: a real-time streaming system of high-fidelity volumetric videos. In *MobiCom '20: The 26th Annual International Conference on Mobile Computing and Networking, London, United Kingdom, September 21-25, 2020*. ACM, 57:1–57:14.

[46] Xiang Li, Wenhai Wang, Xiaolin Hu, and Jian Yang. 2019. Selective Kernel Networks. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*. 510–519.

[47] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. 2016. Continuous control with deep reinforcement learning. In *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016*.

[48] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott E. Reed, Cheng-Yang Fu, and Alexander C. Berg. 2016. SSD: Single Shot MultiBox Detector. In *Computer Vision - ECCV 2016 - 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016 (Lecture Notes in Computer Science, Vol. 9905)*. Springer, 21–37.

[49] Hongzi Mao, Shannon Chen, Drew Dimmery, Shaun Singh, Drew Blaisdell, Yuandong Tian, Mohammad Alizadeh, and Eytan Bakshy. 2020. Real-world Video Adaptation with Reinforcement Learning. *CoRR* abs/2008.12858 (2020).

[50] 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.

[51] 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)*.

[52] Nick McKeown, Guido Appenzeller, and Isaac Keslassy. 2019. Sizing router buffers (redux). *Comput. Commun. Rev.* 49, 5 (2019), 69–74.

[53] Tong Meng, Neta Rozen Schiff, Philip Brighten Godfrey, and Michael Schapira. 2020. PCC Proteus: Scavenger Transport And Beyond. In *SIGCOMM '20: Proceedings of the 2020 ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication, Virtual Event, USA, August 10-14, 2020.* ACM, 615–631.

[54] 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.

[55] Volodymyr Mnih, Nicolas Heess, Alex Graves, and Koray Kavukcuoglu. 2014. Recurrent Models of Visual Attention. In *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada.* 2204–2212.

[56] 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.

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

[58] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*. 5998–6008.

[59] Ji Wang, Bokai Cao, Philip S. Yu, Lichao Sun, Weidong Bao, and Xiaomin Zhu. 2018. Deep Learning towards Mobile Applications. In *38th IEEE International Conference on Distributed Computing Systems, ICDCS 2018, Vienna, Austria, July 2-6, 2018.* IEEE Computer Society, 1385–1393.

[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)*.

[62] Leilei Wu, Anfu Zhou, Xiaojiang Chen, Liang Liu, and Huadong Ma. 2019. GCC-beta: Improving Interactive Live Video Streaming via an Adaptive Low-Latency Congestion Control. In *2019 IEEE International Conference on Communications, ICC 2019, Shanghai, China, May 20-24, 2019.* IEEE, 1–6. https://doi.org/10.1109/ICC.2019.8761256

[63] Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. 2015. Empirical Evaluation of Rectified Activations in Convolutional Network. *CoRR* abs/1505.00853 (2015).

[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* (Taipei, Taiwan) *(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.

[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] Hyunho Yeo, Chan Ju Chong, Youngmok Jung, Juncheol Ye, and Dongsu Han. 2020. NEMO: enabling neural-enhanced video streaming on commodity mobile devices. In *MobiCom '20: The 26th Annual International Conference on Mobile Computing and Networking, London, United Kingdom, September 21-25, 2020.* ACM, 28:1–28:14.

[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* (London, United Kingdom) *(SIGCOMM '15)*. ACM, 509–522.

[69] Huanhuan Zhang, Anfu Zhou, Jiamin Lu, Ruoxuan Ma, Yuhan Hu, Cong Li, Xinyu Zhang, Huadong Ma, and Xiaojiang Chen. 2020. OnRL: improving mobile video telephony via online reinforcement learning. In *MobiCom '20: The 26th Annual International Conference on Mobile Computing and Networking, London, United Kingdom, September 21-25, 2020.* 29:1–29:14.

[70] Huanhuan Zhang, Anfu Zhou, Ruoxuan Ma, Jiamin Lu, and Huadong Ma. 2021. Arsenal: Understanding Learning-based Wireless Video Transport via In-depth Evaluation. *IEEE Transactions on Vehicular Technology* (2021).

[71] 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.