TOWARDS JOINT LOSS AND BITRATE ADAPTATION IN REALTIME VIDEO STREAMING

Dayou Zhang[1,†], *Kai Shen*[1,†], *Fangxin Wang*[1,2,*], *Dan Wang*[3] *and Jiangchuan Liu*[4]

[1]SSE and FNii, The Chinese University of Hong Kong, Shenzhen, China
[2] Peng Cheng Laboratory, Shenzhen, China
[3]Department of Computing, The Hong Kong Polytechnic University, Hong Kong
[4]School of Computing Science, Simon Fraser University, Canada
{dayouzhang, kaishen}@link.cuhk.edu.cn wangfangxin@cuhk.edu.cn
csdwang@comp.polyu.edu.hk jcliu@cs.sfu.ca

ABSTRACT

Recent years have seen booming development of realtime streaming services, highly improving user experience in remote work, online education, and entertainment. Unlike video-on-demand (VoD) or live services, realtime streaming service has extremely stringent delay requirements, rendering the TCP-based transmission no longer applicable. Existing works based on UDP (or its variants) either suffer from the packet loss problem or only focus on improving several QoS metrics, which cannot achieve satisfactory user QoE.

Our insight is to slightly sacrifice the bitrate and video quality to trade for the most significant delay to maximize the overall QoE. We propose `Oppugno`[‡], an integrated framework that achieves joint loss adaptation and bitrate adaption towards maximized QoE in realtime streaming services. `Oppugno` leverages existing UDP mechanisms and employs an advanced deep reinforcement learning algorithm Proximal Policy Optimization (PPO), to adaptively select optimal actions based on network conditions. Trace-driven experiments demonstrate the superiority of our framework, which outperforms the SOTA work by $3.9\% \sim 11.6\%$.

***Index Terms*—** Realtime video streaming, ABR Algorithm, quality of experience, reinforcement learning.

1. INTRODUCTION

The explosive development of Internet infrastructure empowered the rapid switch of multimedia applications from VoD services like Youtube to live broadcast services like Twitch.tv, providing the one-to-all realtime streaming experience. In recent years, the continuous spreading of the Covid-19 pandemic further stimulated the urgent demand for such realtime video conference applications as Zoom or even the immersive VR/AR-based chatting services [1] in Metaverse, calling for all-to-all realtime streaming services. As reported by Cisco [2], the realtime multimedia traffic on the Internet will be tripled from 2020 to 2022 and will account for 17% of the total Internet video traffic.

Unlike VoD or live services, realtime streaming service has extremely stringent requirements for the streaming delay, usually less than 150 ms [3], making the underlying TCP-based transmission protocol no longer applicable. Such service instead adopts the lightweight UDP-based transmission protocol, which may suffer from the packet loss problem if not handled properly, leading to video distortion and further undermining the quality of experience (QoE). Some existing approaches [4] mainly focus on bitrate adaptation for VoD services, while they all target TCP-based lossless scenarios and are insufficient to deal with UDP-based applications with loss. Other approaches try to optimize UDP transmission from codec, multi-path transmission [5], or fast retransmission, while they only focus on improving simple quality of service(QoS) metrics and usually cannot achieve the overall QoE optimization.

As the overall QoE of realtime streaming service consists of different components such as delay, video quality, and bitrate, in this paper, we argue that *the video quality and bitrate can be slightly sacrificed to trade for the most significant delay, such that the overall QoE can be maximized*. Our insight is two-fold. On the one hand, packets with bit error or packet loss can be *tolerated* at the expense of video quality to avoid retransmission delay and further improve QoE. On the other

*Fangxin Wang is the corresponding author.
†These authors contributed equally to this work.
‡Oppugno is a spell in Harry Potter that makes magical creatures attack the caster. It is a metaphor that we use an additional mechanism to mitigate the influence of packet loss.

The work was supported in part by the National Key R&D Program of China with grant No.2018YFB1800800, the Basic Research Project No.HZQB-KCZYZ-2021067 of Hetao Shenzhen-HK S&T Cooperation Zone, by National Natural Science Foundation of China with Grant No.62102342, by Shenzhen Outstanding Talents Training Fund 202002, and by Guangdong Research Projects No.2017ZT07X152 and No.2019CX01X104. Dan Wang's work was supported in part by GRF 15210119, 15209220, 15200321, ITF-ITSP ITS/070/19FP, CRF C5026-18G, C5018-20G, PolyU 1-ZVPZ, and a Huawei Collaborative Project. Jiangchuan Liu's work was supported in part by an NSERC Discovery Grant.

(a) Loss rate = 0.2% (b) Loss rate = 1% (c) Loss rate = 2%

Fig. 1: The visual effect when experiencing different loss rate

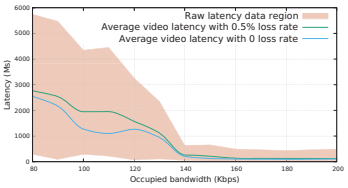


Fig. 2: Relation between bandwidth and latency

hand, extra bandwidth can be leveraged for redundant coding such that loss can be *corrected* without retransmission.

To achieve this goal, we propose `Oppugno`, a coupled framework compatible with existing loss control mechanisms that achieves joint loss and bitrate adaptation toward maximized QoE in realtime streaming. We first highlight the limitations of existing works and point out the opportunity space therein to optimize the QoE in realtime streaming scenes. Then we re-define the QoE and investigate the critical impact factors that affect user QoE. Oppugno focuses on utilizing the existing UDP loss control mechanisms, including FEC [6], UDP-Lite [7], and adaptive bitrate (ABR) control, which is compatible with existing realtime streaming architectures.

We design a holistic deep reinforcement learning (DRL) based approach that well captures the correlations among these mechanisms and environment dynamics for joint loss and bitrate adaptation. We leverage Proximal Policy Optimization (PPO) algorithm to train an intelligent agent based on past network conditions and adaptively select the best action. We conduct extensive trace-driven experiments, and the evaluation confirms Oppugno's superiority, with a $3.9\% \sim 11.6\%$ improvement compared with state-of-the-art solutions.

2. MOTIVATION

2.1. Limitations of Existing Works

Compared with VoD services, realtime streaming services have more stringent latency requirement, higher bandwidth requirement, and more diversified QoE compositions. Therefore, it is more difficult to guarantee the user's QoE in high-QoE-oriented realtime streaming services in current stage. Specifically, existing works still have their limitations in the following aspects:

(1) **TCP protocol is unsuitable for realtime streaming scenarios, while UDP-based streaming will bring packet loss problems.** In TCP-based transmission, clients receive data packets correctly and orderly through mechanisms such as timeout retransmission and congestion control, while these mechanisms make TCP fail to meet the low-latency requirement in realtime scenes. Thus, UDP (or those protocols built upon it) stands out as a promising alternative, meeting the low-latency requirement due to its light-weight design without handling packet loss and retransmission. The consequence however lies in the impact of the caused loss on user QoE, leading to video distortion or even frame lost, as shown in Figure 1.

(2) **Previous ABR algorithms cannot solve the QoE decrease caused by loss.** ABR algorithms are widely employed in video streaming to dynamically adapt to the bitrate of video transmission according to network throughput, thus optimizing video quality. Recently, there have been many works dedicated to improving ABR with learning based methods, e.g., QARC [8], Pensieve [4], and so on. However, such algorithms all consider reliable transmission based on TCP and are incapable of dealing with loss in unreliable UDP-based scenarios, not to mention optimize the overall QoE. More seriously, only using such ABR algorithms cannot distinguish errors caused by insufficient bandwidth or network transmission. As a result, in a poor-quality channel with much loss, the sender will keep reducing the bitrate erroneously, leading to QoE drop.

(3) **Emerging advanced transport protocols also cannot solve the problem of QoE adaption well.** Recently there emerges some new transport protocols towards realtime communication scenarios, e.g., QUIC [9] built on UDP leverages packet recovery, forward error correction, and multipath transmission to achieve low-latency requirement. Such protocols however are particularly designed to guarantee QoS rather than QoE, which is insufficient to simultaneously cover the important metrics in user QoE, including loss, bitrate and delay. Besides, deploying new transmission protocols across Internet infrastructure and devices requires quite a long time and huge overhead.

2.2. Opportunity and Insight

Given the above limitations, we are motivated to re-examine the QoE optimization problem in realtime streaming, exploring the relationship among those important metrics affecting QoE. We have conducted extensive measurements and have some key observations.

We first discuss the relationship between video quality and the streaming delay. Figure 1 describes the perceived video quality under different loss rates. Comparing Figure 1a with Figure 1b and Figure 1c, we can find that though a relatively high loss rate (e.g., 1%) will bring significant video

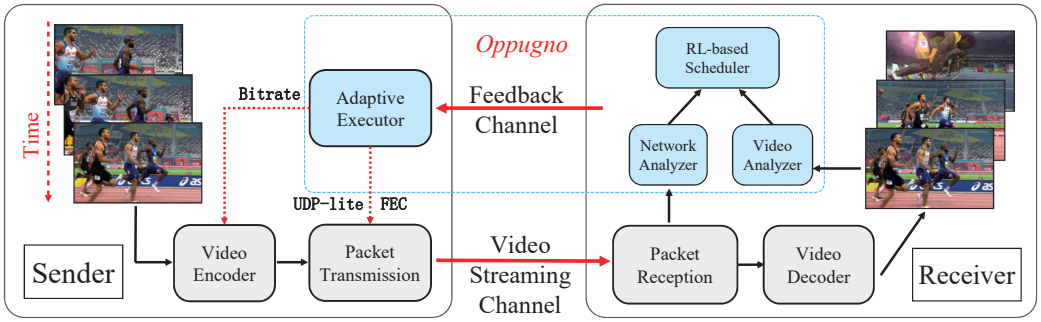


Fig. 3: The Oppugno Framework

quality degradation, people can hardly feel the quality degradation when the loss rate is quite low (e.g., 0.2%). Nevertheless, from the perspective of delay, accepting the 0.2% loss rate without retransmission of the erroneous packet can save an average of 50ms, which can even improve the overall QoE.

We then examine the relationship between the occupied bandwidth (determining the bitrate) and the streaming delay. We find that extra bandwidth can be leveraged for redundant coding and further correct the packet loss, reducing streaming delay by avoiding retransmission. As illustrated in Figure 2, when the transmission link has a 0.5% loss rate, an extra 160Kbps bandwidth is enough to mitigate the impact of loss.

With such observations above, we argue that different QoE metrics are highly correlated with each other, and the opportunity rises in comprehensively fine-tuning them to optimized user QoE in realtime streaming scenarios. Capturing the complicated correlations therein however is challenging, especially considering the uncertain environments such as future bandwidth and loss conditions. Thus, we are motivated to design a joint leaning-based loss and bitrate adaptation scheme, which can well leverage the historical experience to achieve an end-to-end adaptation.

3. DESIGN

In this section, we describe the design of Oppugno, a framework that generates a DRL-based algorithm approach to integrate the loss adaption and bitrate adaption for realtime video streaming services jointly. We start by explaining the design of QoE for realtime streaming. We then explain the framework design of Oppugno, and illustrate the DRL algorithm underlying Oppugno and its application to our framework.

3.1. QoE for Realtime Streaming

We followed the QoE derived from existing works [10, 4, 11] as

$$QoE = \sum_{i=1}^{N} q(R_n, L_n, D_n) - \alpha \sum_{i=1}^{N} T_n - \beta \sum_{i=1}^{N-1} |q(R_{n+1}, L_{n+1}, D_{n+1}) - q(R_n, L_n, D_n)| \quad (1)$$

where N is the total timeslots, R_n indicates the bitrate of timeslot n, L_n is the packets loss ratios*, and D_n is the discarded packets due to timeout. Therefore, $q(R_n, L_n, D_n)$ maps the bitrate, packet loss, and overdue packet discard to the video quality perceived by end-users. T_n represents the delay between sender and users. It is worth noting that excessive delays are not acceptable in our cases. The final term penalizes changes in video quality to favor smoothness. α and β are correlated coefficients to balance the relationships among terms. In our evaluation, we report the average QoE every timeslot to more accurately describe the user experience at each moment.

3.2. The Oppugno Framework

The framework of Oppugno is demonstrated in Figure 3, which mainly consists of a sender, a receiver and two channels for video streaming and information feedback. The sender encodes the raw video, and transmits the video packets to the receiver through the video streaming channel. The receiver decodes the received video packets and repairs the packet loss through the FEC data. Based on the user's QoE and the network status analysis in the previous period (including bandwidth and packet loss rate), Oppugno adaptively adjusts the bitrate and loss control and informs the sender accordingly.

3.2.1. Receiver-driven Control

The whole adaptation is driven by the receiver since the client's network status and video quality can be obtained in time to ensure fast response. Oppugno consists of a network analyzer for network status collection, a video analyzer for video quality calculation, a RL scheduler for control policy making, and an adaptive executor to execute the packet sending. As the core part of the Oppugno, the scheduler employs a deep reinforcement learning based scheme. Past data collected from the network analyzer and video analyzer is used to train the scheduler towards an adaptive RL agent for optimized bitrate and loss control. After the training, the scheduler is used to make control decisions, which are later applied to the adaptive executor at the sender for execution. It is

*There can be different reasons for packet loss, such as bit errors or congestion. We do not distinguish the specific loss reasons in our work.

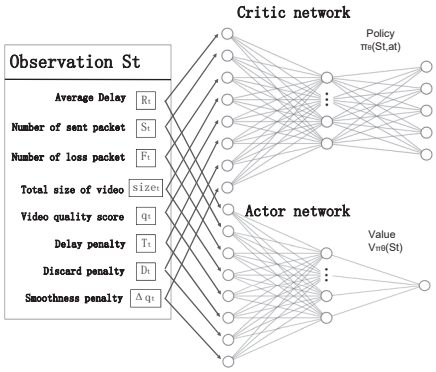


Fig. 4: The PPO learning algorithm

worth noting that the RL-based scheduler can employ an online learning model that updates the model in realtime to suit the environment changes rapidly.

3.2.2. Leveraging Existing UDP Mechanisms

To be compatible with current realtime streaming architectures, we employ the existing UDP control mechanisms for joint loss and bitrate adaptation. In particular, we utilize Forward Error Correction (FEC) and UDP-Lite in our prototype implementation. FEC [6] is a kind of streaming codec mechanism that uses redundant packets to allow the decoder to recover lost packets without explicit retransmission. According to the current network bandwidth, we can effectively reduce the retransmission delay by selecting different FEC levels (i.e., deciding different redundant encoding levels) at the expense of corresponding bandwidth overhead. UDP-Lite [7] is a variant of the original UDP that allows the packet to be partially checked with a specified percentage. With this mechanism, the receiver can tolerate partial packet error, allowing a certain degree of error to be accepted.

3.3. Algorithm Design

3.3.1. Basic Training Algorithm

In Oppugno, we use Proximal Policy Optimization [12], a state-of-the-art actor-critic reinforcement learning algorithm. Specifically, the agent samples an action based on the current environment state and the probability distribution given by the *policy* $\pi_\theta\left(S_t, a_t\right)$. After executing each action, the simulated environment will give us corresponding feedback, which will be used as observations of the environment for the agent to make the next decision. The traditional actor-critic is a policy gradient method, and the principle is as follows:

$$\nabla_\theta \mathbb{E}_{\pi_\theta} = \hat{\mathbb{E}}_t \left[\nabla_\theta \log_{\pi_\theta}\left(a_t \mid s_t\right) \hat{A}_t\right] \tag{2}$$

This formula uses π_θ to represent policy and all its parameters. $\hat{A}_t$ is the advantage function, representing the difference between action a_t and the average value of all actions. The idea is to use the policy's trajectory to estimate the gradient of the expected total reward, while the advantage does not require all interactions with the environment and could update the policy at any time.

However, the learning rate of this traditional policy gradient method is challenging to control. Learning will be slow and potentially fall to a local optimum if the learning rate is too low. Meanwhile, if the learning rate is too high, it will cause more significant fluctuations, and it will not be easy to converge. Therefore, we used a better-performing method - PPO. The essential formula of PPO is:

$$L^{clip}\left(\theta\right) = \begin{cases} \left(1-\epsilon\right) A_t, & r_t\left(\theta\right) \leq 1-\epsilon \ \ and \ \ A_t < 0, \\ \left(1+\epsilon\right) A_t, & r_t\left(\theta\right) \geq 1+\epsilon \ \ and \ \ A_t > 0, \\ r_t\left(\theta\right) A_t, & otherwise. \end{cases} \tag{3}$$

$r_t\left(\theta\right)$ is defined as follows:

$$r_t\left(\theta\right) = \frac{\pi_\theta\left(a_t \mid s_t\right)}{\pi_{\theta_{old}}\left(a_t \mid s_t\right)} \tag{4}$$

As the equation shows, PPO compares the new policy with the old policy and uses this coefficient $r_t\left(\theta\right)$ to limit the update range. With this improvement, PPO is more stable, reliable, and has better overall performance. Furthermore, the implementation of PPO has the same level of complexity as the original actor-critic method. We present more details in Algorithm 1.

Algorithm 1 The Proximal Policy Optimization Algorithm, Actor-Critic Style

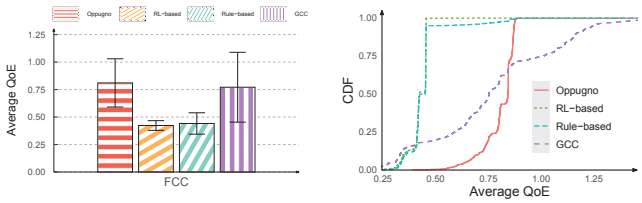
Input: Initial policy $\pi_{\theta_{old}}$

for iteration=1,2,... **do**
  for actor=1,2,...,N **do**
    Run policy $\pi_{\theta_{old}}$ in environment, and collect set of partial trajectories
    Estimate the advantages $\hat{A}_1$,..., $\hat{A}_T$
  end for
  Optimize surrogate $L_t\left(\theta\right)$, with K epochs and minibatch size $M \leq NT$
  $\theta_{old} \leftarrow \theta$
end for

3.3.2. Observation and Action Space

Oppugno's observation space contains the measured values of last time slot, including delay, sent packets, packet loss and video size. Intuitively, the increase or decrease of any variable in the action space has a fixed impact on the quality. Therefore, we also try to feed different scores or penalties to the PPO algorithm, including video quality score, delay penalty, discard penalty, and smoothness penalty. With the guidance of these values, Oppugno indeed performs better.

Our decision space contains three independent variables: bitrate, FEC ratio, and UDP-Lite ratio. The number of combinations between them is enormous. However, this makes



(a) Average QoE (b) CDF of QoE

Fig. 5: QoE under FCC dataset

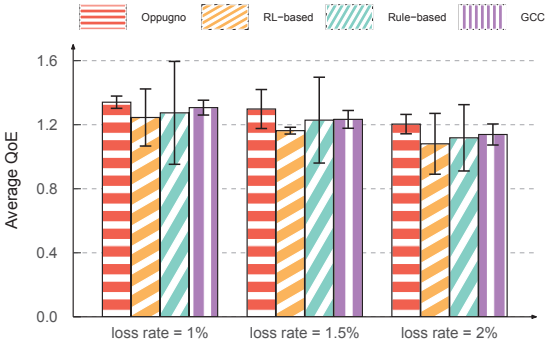


Fig. 6: QoE under different network environments

the probability of each action taken by the policy is tiny and ultimately cause it difficult for the RL algorithm to learn a good policy. To mitigate this problem, we did the following thinking: choosing a combination among three different parameters is actually finding a point in a three-dimensional space. If we turn this process into continuous walking in this three-dimensional space, we only need to define actions as different travel directions. Specifically, taking new action is like walking in different directions in this three-dimensional space or staying at the current position. In this way, we have reduced the number of action spaces from hundreds to seven, making the implementation of Oppugno become possible.

4. EVALUATION

In this section, we evaluate the performance of Oppugno with real-world datasets and a broad range of synthesized datasets.

4.1. Methodology

Network Traces: We use two real-world datasets in our experiments: (1) the FCC dataset [13] containing over 1 million throughput and round-trip-time traces collected in natural environments; (2) the HSDPA dataset [14] on mobile throughput covers multiple usage scenarios such as buses, trains, and cities, which together with the FCC dataset is used to synthesize environments with various conditions to verify the generalization of Oppugno.

Baselines: We compare Oppugno to the following baseline video streaming algorithms:

(1) Rule-based ABR: It is a lightweight ABR algorithm that estimates the bandwidth using the weighted average of the last 5-second bandwidth, and then selects the corresponding bitrate.

(2) RL-based ABR: We employ the RL-based ABR algorithm proposed in Pensieve [4] but decrease the decision interval to 0.4 seconds to fit the realtime scenario.

(3) GCC: We implement the basic idea of GCC [15], a well-known UDP congestion control algorithm in our context. It estimates network congestion through packet loss rate from sender and delay from receiver, and reduces the sending rate when congestion happens.

Experiment Setup: Our experiment follows the QoE setting in Eq. 1, where $q(R_n, L_n, D_n) = c_r R_n + c_l L_n + c_d D_n$. The hyperparameters of c_r, c_l and c_d are set as 1, -1, and -0.5 respectively to balance the three components. The penalty term coefficient α and β are set as 0.1 and 0.2. The learning rate in our PPO algorithm is set as $3e^{-4}$.

4.2. Performance with Real-world Data

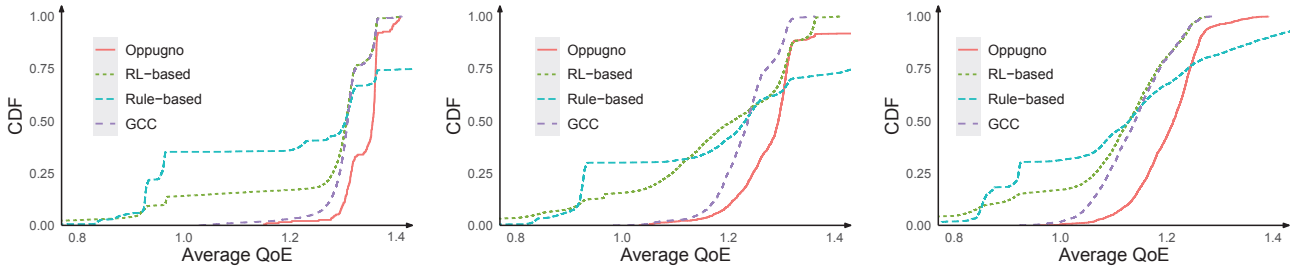
We first test the performance of Oppugno under real-world network conditions. We utilize the FCC dataset as it contains all the metrics we need including bandwidth, loss, delay, etc. We demonstrat the overall QoE score achieved by Oppugno compared with the baseline methods in Figure 5a and the detailed CDF plot in Figure 5b. We have two key observations.

First, Oppugno is able to outperform all the baseline methods with higher QoE, with 5.1% higher QoE than GCC and 92.6% higher QoE than RL-based ABR under the real-world dataset. This result confirms the effectiveness of our joint design to integrate the loss adaptation and bitrate adaption together for QoE optimization. An interesting result is that both rule-based ABR and RL-based ABR are inefficient to deal with this condition, achieving much worse performance even than GCC. This is largely because of the impact of re-transmission, and further indicates that the loss problem is quite significant in realtime streaming so that traditional ABR algorithms without considering loss is not applicable.

Second, Oppugno can achieve more stable and concentrated QoE score than the baselines. Figure 5a and the curve in Figure 5b show a smaller variance of Oppugno than GCC. This indicates that GCC is quite sensitive to packet loss and insufficient to handle conditions with large loss ratio. Instead, the QoE score of Oppugno is mainly concentrated within the range of 0.5 to 1.0, reflecting that Oppugnu can well mitigate the loss impact, even under a poor network condition with large loss. As to the ABR algorithms, they achieve quite concentrated but overall poor QoE performance, which infers that they fail to provide high QoE in a lossy environment.

4.3. Performance of Generalization

Oppugno performs well on the public dataset in the experiments above, while the practical environment can be more complex with various network conditions. To evaluate the



(a) loss rate = 1%

(b) loss rate = 1.5%

(c) loss rate = 2%

Fig. 7: The CDF of the QoE by emulations run over synthesized network traces

ability of Oppugno to generalize to different network conditions, we synthesize network traces with different loss rate based on the FCC dataset and HSDPA dataset. Figure 6 shows the average QoE of different methods over different loss rate and Figure 7 polts the detailed CDF plots. The results reveal more characteristics of these methods. We can find that at different loss rate ranging from 1% to 2%, Oppugno can achieve better QoE than all the baseline methods, with 3.9% QoE improvement than GCC and 11.6% QoE improvement than RL-based ABR when loss rate is 2%. This result indicates that Oppugno has good generalization ability to adapt to environments with different loss. For the baseline methods, GCC outperforms other ABR-based methods with different loss rate. This observation demonstrates that loss adaption is actually a more dominant problem in the realtime streaming scenario. Solutions with only bitrate adaptation cannot deal with bit errors and packet losses, rendering larger QoE drop.

5. CONCLUSION

We present Oppugno, a framework that combines loss adaptation and bitrate adaptation with the PPO reinforcement learning algorithm to tackle the realtime video streaming scenarios. Unlike traditional ABR algorithms, Oppugno takes the trade-off between bandwidth and acceptance of packet errors into consideration. With these improvements, Oppugno outperformed the existing ABR algorithm by $3.9\% \sim 11.6\%$ over the simulation of two famous datasets.

6. REFERENCES

[1] R. P. Singh, M. Javaid, R. Kataria, M. Tyagi, A. Haleem, and R. Suman, "Significant applications of virtual reality for covid-19 pandemic," *Diabetes & Metabolic Syndrome: Clinical Research & Reviews*, vol. 14, no. 4, pp. 661–664, 2020.

[2] T. Barnett, S. Jain, U. Andra, and T. Khurana, "Cisco visual networking index (vni) complete forecast update, 2017–2022," *Americas/EMEAR Cisco Knowledge Network (CKN) Presentation*, 2018.

[3] Zoom Corporation, "Meeting and phone statistics," https://support.zoom.us/hc/en-us/articles/202920719-Meeting-and-phone-statistics, 2021.

[4] H. Mao, R. Netravali, and M. Alizadeh, "Neural adaptive video streaming with pensieve," in *Proceedings of ACM SIGCOMM*, 2017.

[5] Y. Guan, Y. Zhang, B. Wang, K. Bian, X. Xiong, and L. Song, "Perm: Neural adaptive video streaming with multi-path transmission," in *Proceedings of IEEE INFOCOM*, 2020.

[6] P. Frossard, "Fec performance in multimedia streaming," *IEEE Communications Letters*, vol. 5, no. 3, pp. 122–124, 2001.

[7] A. K. Shukla, G. S. Kahlon, and M. Dubey, "Udp lite: Efficient model for video transmission by udp lite protocol," in *Proceedings of IEEE ICACCCT*, 2016.

[8] T. Huang, R. Zhang, C. Zhou, and L. Sun, "Qarc: Video quality aware rate control for real-time video streaming based on deep reinforcement learning," in *Proceedings of ACM MM*, 2018.

[9] J. Iyengar and M. Thomson, "Quic: A udp-based multiplexed and secure transport," *Internet Engineering Task Force, Internet-Draft draftietf-quic-transport-17*, 2018.

[10] X. Yin, A. Jindal, V. Sekar, and B. Sinopoli, "A control-theoretic approach for dynamic adaptive video streaming over http," in *Proceedings of ACM SIGCOMM*, 2015.

[11] X. Zhang, Y. Ou, S. Sen, and J. Jiang, "Sensei: Aligning video streaming quality with dynamic user sensitivity.," in *Proceedings of NSDI*, 2021.

[12] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[13] Office of Engineering and Technology, "Raw data measuring broadband america 2021," https://www.fcc.gov/oet/mba/raw-data-releases, 2021.

[14] R. Haakon, V. Paul, G. Carsten, and H. Pål, "Commute path bandwidth traces from 3g networks: analysis and applications," in *Proceedings of IEEE MMSys*, 2013.

[15] G. Carlucci, L. De Cicco, S. Holmer, and S. Mascolo, "Analysis and design of the google congestion control for web real-time communication (webrtc)," in *Proceedings of IEEE MMSys*, 2016.