Reinforcement Learning Based Cross-Layer Congestion Control for Real-Time Communication

Haoyong Li[a], Bingcong Lu[a], Jun Xu[a], Li Song[a], Wenjun Zhang[a], Lin Li[b] and Yaoyao Yin[c]

[a]*School of Electronic Information and Electrical Engineering, Shanghai Jiao Tong University,*
Email: {koyong, lu, xujunzz, song_li, zhangwenjun}@sjtu.edu.cn
[b]*Migu Cultural Technology Co., Ltd.*
[c]*China Mobile Communications Co., Ltd.*

***Abstract*—Congestion control is a crucial part of Real-Time Communication (RTC), because it adjusts the current video bitrate according to the variable network environment, which determines the final Quality of Experience (QoE). Conventional congestion control algorithms only take packet-level delay and throughput into consideration. While in the application of RTC, video frame can provide a lot of reference information, because it is the basic unit of video processing and evaluation. Inspired by this, we propose CLCC, a reinforcement learning based Cross-Layer Congestion Control for WebRTC. CLCC uses not only packet-level information, but also frame-level information to decide the bitrate of the encoder. We evaluate CLCC on both random traces and LTE traces. Results show that CLCC outperforms GCC with improvements in average frame psnr of 0.4-0.6 and decreasing in average frame delay of 17%-23%.**

***Index Terms*—Real-time communication, cross-layer, congestion control, reinforcement learning**

I. Introduction

According to a Cisco report[1], by 2022, video traffic will account for 82% of all Internet traffic. In addition to traditional video streaming, Real-Time Communication (RTC), including video conference, live broadcast, interactive video, etc, has become another focus of multimedia services due to its advantage of low latency[2].

Since RTC asks for minimal latency and high video quality, the design for its congestion control is more challenging than it for traditional congestion control. Existing congestion control for RTC can be categorized into three types: model-based, RL-based and hybrid. Model-based congestion control includes Google Congestion Control (GCC)[3]. GCC uses a delay-based state machine model and a loss-based rule together to give the bitrate according to the network stats collected from RTCP packets. However, under unstable network environment, the fixed rule within the delay-based state machine often fails to track the rapidly changing bandwidth. Moreover, the loss-based rule of GCC takes all losses as flags of congestion, which may lead to cutting the bitrate so much that the loss-based rule dominates GCC. In order to better adapt to variable network environment and improve bandwidth utilization, many researchers have tried to combine reinforcement learning with congestion control. R3Net[4] (RL-based Recurrent Network for RTC) employs a deep neural network to output the bitrate directly base on the RTCP packet stats. Hybrid Receiver-side Congestion Control[5] (HRCC) combines deep reinforcement learning and GCC. HRCC uses a deep neural network to guide GCC with a multiplication factor periodically. These two algorithms aim at tracking the varying bandwidth better than GCC, however, these algorithms, as well as GCC, all have two unconsidered problems, stated as below.

On one hand, all these congestion controls only consider the packet-level information, while the frame-level information is the most important concern in the RTC system. Although the packet-level information has a certain correlation with the frame-level information, the packet-level information cannot fully represent the content of the frame-level information. Therefore, if the congestion control can adjust the bitrate basing on the frame-level information, the final QoE can be optimized more directly. In traditional video streaming systems, congestion control is often in the transport layer, while frame-level information is covered in the application layer, which are far apart. However, RTC system connects the transport layer and the application layer by letting congestion control guide the bitrate of the encoder. Under this circumstance, there may be more possibilities to further cooperate the congestion control with the encoder to maximize the final Quality of Experience (QoE)[6].

On the other hand, while congestion control for RTC should aim at optimizing the final QoE, these three congestion control algorithms all aim at achieving high bandwidth utilization. In fact, the increase in bandwidth utilization is not necessarily proportional to the improvement in video quality. [7] has done an experiment on the relationship between video bitrate and video quality. The results show that when the video bitrate achieves a certain level, the increasing rate of video quality tends to decrease, at this time, the video bitrate reaches a saturation point.

RTC system uses a lower video frame rate and resolution in order to ensure real-time performance. Additionally, RTC is mostly used in video conference, where there usually are light movement and relatively fixed scene. For these reasons, the bitrate required by RTC is inherently lower than that required by traditional video streaming system. Thus, continuous pursuit for a high bandwidth utilization may be a waste on the bandwidth. Moreover, when the network condition suddenly deteriorates, the video sent at an unnecessarily high bitrate will cause more serious network congestion.

Inspired by these factors, we firstly propose a method for the RTC congestion control module to obtain frame-level information. Base on this, we introduce a deep reinforcement learning based Cross-Layer congestion control: CLCC, which breaks the gap between transport layer and application layer. Results show that CLCC outperforms state-of-art congestion control for RTC on both frame-level metrics and packet-level metrics.

The rest of the paper is organized as following. Section II introduces our system architecture. Section III introduces the design of our deep reinforcement learning algorithm. Section IV introduces our training environment. Section V shows our test results against other state-of-art congestion control algorithms for RTC.

II. System Design

In this section, we introduce the original RTC system architecture and our modification.

We choose AlphaRTC[4] as the base of our RTC system, because it decouples the congestion control module from the RTC system to facilitate the design of congestion control for researchers. AlphaRTC is a fork of Google's WebRTC code base, but has replaced WebRTC's default congestion control with a receiver-side user-defined congestion control. Our system architecture and our modifications are demonstrated in Figure 1.

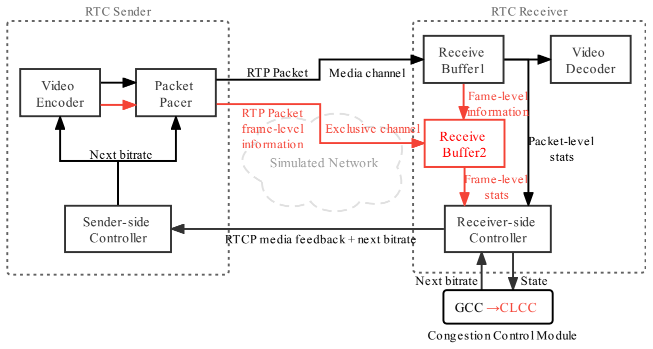


Fig. 1: System architecture and modification

The black part in Figure 1 is the original AlphaRTC system. On the receiver side, the receive buffer1 will receive the RTP packets from the sender, hand the video frame to the video decoder, and count the packet-level receiving rate, delay and loss rate according to the timestamp and sequence number of the packets. These packet-level stats will be delivered to the receiver-side controller, which will be sorted and forwarded to the congestion control module by the receiver-side controller. GCC inside the congestion control module will output the next bitrate every decision-making interval τ (200ms in this paper) based on the packet-level information. The congestion control module will return the next bitrate to the receiver-side controller, and the receiver-side controller will send it to the sender-side controller through RTCP packets. On the sender side, after the sender-side controller receives the next bitrate, it will deliver the next bitrate to the video encoder and the packet pacer. The pacer will send the encoded frames through RTP packets to the receiver.

The red part in Figure 1 is the modification we made in the RTC system. On the sender side, the pacer will not only send the video frames, but also send the frame-level information collected on the sender side, such as frame timestamp, PSNR, resolution, encoded time, etc, to the receiver. On the receiver side, the receive buffer1 will deliver the frame-level information collected on the receiver side, such as frame timestamp, frame arrival time, etc, to the receive buffer2. The receive buffer2 will collect the frame-level information on both the sender side and the receiver side, and output the frame-level stats to the receiver-side controller. The receiver-side controller will make a state including both frame-level and packet-level information, and deliver it to the congestion control module. CLCC inside the congestion control module will output the next bitrate every τ based on the state.

III. Deep Reinforcement Learning Design

In this section, we introduce our deep reinforcement learning algorithm: the design of state space, reward function, action space and neural network model.

A. State Space

The state, an 8-dimension vector, $[d_{\text{pkt}}, d_{\text{pkt}}^{\text{grad}}, l_{\text{pkt}}, q_{\text{frame}}, r_{\text{frame}}, d_{\text{frame}}, d_{\text{frame}}^{\text{grad}}, a_{\text{last}}]$, can be mainly divided into three parts as shown in Table1: packet-level state, frame-level state and last action.

TABLE I: State Description

State		Description
Packet-level states	d_{pkt}	Average packet delay in the last time interval
	$d_{\text{pkt}}^{\text{grad}}$	Average packet delay gradient in the last time interval
	l_{pkt}	Burst loss rate in the last time interval
Frame-level states	q_{frame}	Average frame quality(PSNR) received in the last time interval
	r_{frame}	The largest frame resolution received in the last time interval
	d_{frame}	Average frame delay in the last time interval
	$d_{\text{frame}}^{\text{grad}}$	Average frame delay gradient in the last time interval
a_{last}		The last action made

In **Packet-level states**, the average packet delay d_{pkt} is the mean value of the relative packet delay in the last τ. The relative packet delay is the delay difference between each packet and the first packet. The average packet delay gradient $d_{\text{pkt}}^{\text{grad}}$ is the difference between the average packet delay and the last average packet delay. It represents the packet delay variation trend between two τ. The burst loss rate l_{pkt} is the ratio of the longest burst loss length to the total packet length in the last time interval. We choose burst loss rate rather than regular loss rate to distinguish random packet loss and network congestion packet loss to better guide the agent.

In **Frame-level states**, we use PSNR$_{\text{enc}}$, the PSNR obtained during encoding inside the encoder, to represent the frame quality q_{frame}. However, due to the resolution switching policy inside WebRTC, the resolution of frames may be reduced before sent into the encoder. Thus PSNR$_{\text{enc}}$ represents the difference between the encoded frame and the reduced-resolution frame, instead of the difference between the encoded frame and the original-sized frame. The PSNR$_{\text{enc}}$ of a low-resolution video frame can not represent the actual quality of the frame. For example, while the PSNR$_{\text{enc}}$ of a low-resolution frame may be high, the actual PSNR and quality of the frame might be quite low. To represent the frame quality with PSNR$_{\text{enc}}$, we multiply the PSNR$_{\text{enc}}$ with the ratio of its resolution to max resolution. In this way, the PSNR$_{\text{enc}}$ of a low resolution frame is multiplied with a small fraction, indicating a relativly low quality. The defined average frame quality q_{frame} is shown below:

$$q_{\text{frame}} = \frac{1}{n}\sum_{i=1}^{n}(\text{PSNR}_{\text{enc}\ i} \times \frac{\text{r}_i}{\text{maxR}}), \quad (1)$$

where n is the number of the frames received in the last τ, PSNR$_{\text{enc}\ i}$ is the PSNR$_{\text{enc}}$ of the i_{th} frame, r$_i$ is the resolution of the i_{th} frame and maxR is the resolution of the original video. The frame resolution r_{frame} is the ratio of the largest resolution of the frames received in the last τ to max resolution.

The average frame delay d_{frame} is the mean value of the frame delay in the last τ. The flow of one frame in WebRTC system is shown in Figure 2. The frame delay refers to the

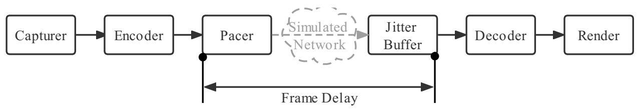


Fig. 2: WebRTC system frame flow

delay between the time when a frame is handed to the pacer on the sender side and the time when a frame is handed to the decoder on the receiver side. We choose these two points in the frame flow to maximize the influence of transportation congestion in frame delay. The average frame delay gradient $d_{\text{frame}}^{\text{grad}}$ is the difference between the average frame delay and the last average frame delay. It represents the frame delay variation trend between two τ.

The last action a_{last} is the last action made by the neural network.

B. Reward Function

Our reward function is also composed of packet-level reward R_{pkt} and frame-level reward R_{frame}.

The reward function should encourage the actions that increase the frame quality, and penalize the actions that lead to high packet delay, high packet loss rate and high frame delay. There is always room for improvement in video quality when $r_{\text{frame}} < 1$. However, when $r_{\text{frame}} = 1$, with the continuous improvement of the bitrate, the video quality may reach a bottleneck point. At this time, it is necessary for the agent to judge whether the quality improvement worth the corresponding delay risk brought by the increasing bitrate. For this reason, we choose receiving rate as the positive factor in the reward function when $r_{\text{frame}} < 1$, and we turn to PSNR$_{\text{enc}}$ after $r_{\text{frame}} = 1$.

We define our reward function as following:

$$R_{\text{pkt}} = \begin{cases} rate_{\text{pkt}} - 16 \times d_{\text{pkt}} - 6 \times l_{\text{pkt}}, & r_{\text{frame}} < 1 \\ -16 \times d_{\text{pkt}} - 6 \times l_{\text{pkt}}, & r_{\text{frame}} = 1 \end{cases} \quad (2)$$

$$R_{\text{frame}} = \begin{cases} -10 \times d_{\text{frame}}, & r_{\text{frame}} < 1 \\ \frac{4}{15} \times \text{PSNR}_{\text{enc}} - 10 \times d_{\text{frame}} & \\ +rate_{\text{pkt\_r}} - \frac{4}{15} \times \text{PSNR}_{\text{enc\_r}}, & r_{\text{frame}} = 1 \end{cases} \quad (3)$$

$$Reward = R_{\text{pkt}} + R_{\text{frame}}, \quad (4)$$

where $rate_{\text{pkt}}$ is the receiving rate(Mbps) in the last τ, PSNR$_{\text{enc}}$ is the average PSNR$_{\text{enc}}$ of the frames received in the last τ, $rate_{\text{pkt\_r}}$ and PSNR$_{\text{enc\_r}}$ are the values of $rate_{\text{pkt}}$ and PSNR$_{\text{enc}}$ when r_{frame} turns to 1.

C. Action Space

Unlike [4], in which the action is directly mapped to be the next bitrate, we map the action to be a gain coefficient to tune the next bitrate. The advantage of mapping the output this way is that it can prevent the excessive fluctuation of the bitrate, which will cause the encoded bitrate to be far from the target bitrate, thus making the congestion control invalid. Our network outputs an action in (0, 1) range, which is then mapped to (dec_max, inc_max) as a gain coefficient g to tune the bitrate as follows:

$$rate_t = g \times rate_{t-1}, \quad (5)$$

where $rate_t$ is the bitrate given to the receiver-side controller and g is the gain coefficient mapped from the action.

We choose a piecewise linear function to be the mapping function, which is defined as follows:

$$g = \begin{cases} \frac{action}{mid} \times (1 - dec\_max) + dec\_max, & action < mid \\ \frac{action - mid}{1 - mid} \times (inc\_max - 1) + 1, & action \geq mid \end{cases} \quad (6)$$

where $action$ is the output of the neural network and mid is the equilibrium point that tunes the bitrate with 1. Based on experimental experience, we choose the values of inc_max, dec_max and mid to be 1.1, 0.7 and 0.4.

D. Neural Network Model

We train CLCC with an actor-critic framework, where except that actor has one sigmoid activation layer more than critical, the other layers are consistent. The neural network model we use is the same as [4]: a recurrent neural network with Gated Recurrent Units (GRU)

TABLE II: PPO Parameters

Parameter		Value
Exploration Value	0 - 50	0.25 - 0.15
	51 - 110	0.15 - 0.1
	111 - 180	0.1 - 0.01
Learning Rate	0 - 50	3e-3
	51 - 110	3e-4
	111 - 180	3e-5
Update Epoch		15
Clip Parameter		0.1
Adams_betas		(0.9, 0.999)
Sample per Episode		18000

TABLE III: Trace Main Limits

Bandwidth	Changing Rate	Loss Rate	Queue Size
[0.25,4]Mbps	[1,4]Hz	[0,5]%	[6,350]

IV. Implementation

In this section, we introduce our training data set, environment and our choice of parameters.

Our model is trained using Proximal Policy Optimization (PPO)[8]. The training parameters are shown in Table II.

We used four 1280x720 video conference scene videos for training. The four videos are from Xiph.org[9]: FourPeople, Johnny, KristenAndSara, vidyo1 and vidyo3. The length of these training traces and videos is 45 seconds.

We used 1000 generated traces for training, and used Mahimahi[10] to simulate these traces. We randomly generated these traces under the limits shown in Table III. Unlike [4] and [5], for stronger robustness and better realism, we run the RTC in real environment both in the training process and evaluating process.

V. Evaluation

In this section, we will introduce our testing data set and environment, and present our test results compared to the other congestion control algorithms.

A. Environment

To evaluate the performance of CLCC, we test CLCC and other algorithms in two groups of traces: random traces and LTE traces. The random traces are 100 traces generated in the same way as in Section IV, and the LTE traces are some traces given by DeepCC[11]. The tests on random traces show the general performance of CLCC, since these traces are similar to the traces it is trained in. The tests on LTE trace can show its general performance and robustness of CLCC under real traces. Our test video is vidyo4 from Xiph.org[9].

We use average packet receiving rate, average packet delay, average packet loss rate, average frame PSNR, average VMAF score, average frame delay and average frame drop rate as the evaluation metrics, because these metrics can indicate the effect of congestion control and its effect on the QoE of RTC video transmission. We run the tests for 5 times to eliminate the random errors. We will analyze the key indicator results and QoE results of these algorithms. Furthermore, we will present the running simulation to illustrate the difference of the behavior of CLCC against GCC and HRCC.

B. Key Indicator Results

The test results are shown in Figure 3 and 4. The average result is denoted by the star shaped dots, and each single result is denoted by the round dots.

Except for packet receiving rate, the other results show that CLCC has a better performance than GCC and HRCC. CLCC has a low packet receiving rate is because it will not always tend to increase the throughput, if it senses the bitrate is currently saturated, it will maintain the bitrate at that level.

On packet level, compared with GCC, CLCC has (5%, 7%) and (8%, 33%) advantages in packet delay and packet loss rate respectively. On frame level, compared with GCC, CLCC has (0.4, 0.6), (0.4, 1.8), (17%, 23%) and (25%, 30%) advantages in frame PSNR, VMAF score, frame delay and frame drop rate.

Although HRCC has high frame PSNR and VMAF score in the result of LTE traces, its performance in other aspects is poor. On one hand, it is due to the poor robustness of HRCC, on the other hand, HRCC sacrifices too much in terms of delay and frame drop rate in order to increase the bandwidth utilization.

C. QoE Results

We use two QoE functions to measure the performance of CLCC, GCC and HRCC, one on packet-level and one on frame-level. The packet-level QoE score indicates how well the congestion control utilizes the varying bandwidth while keeping a low packet delay and a low frame delay. The frame-level QoE score represents the final QoE of the RTC video transmission.

The packet-level QoE given by ACM MMSys'21 Grand Challenge includes throughput, packet delay and loss rate, which is shown below:

$$QoE_{\text{recv\_rate}} = 100 \times U, \tag{7}$$

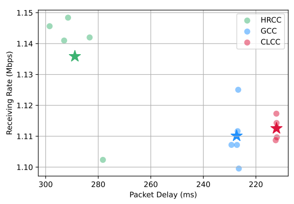
$$QoE_{\text{delay}} = 100 \times \frac{d_{\max} - d_{95\text{th}}}{d_{\max} - d_{\min}}, \tag{8}$$

$$QoE_{\text{loss}} = 100 \times (1 - L), \tag{9}$$

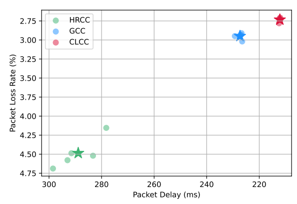
$$QoE_{\text{packet}} = 0.2 \times QoE_{\text{recv\_rate}} + 0.2 \times QoE_{\text{delay}} + 0.3 \times QoE_{\text{loss}}, \tag{10}$$

where U is the average bandwidth utilization of that trace test, $d_{\max}$, $d_{\min}$ and $d_{95\text{th}}$ is the maximum, the 95th percentile, and minimum packet delay (ms) of that trace test, L is the average packet loss rate of that trace test.

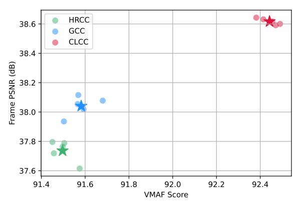
The frame-level QoE function we use is modified from the packet-level QoE function. We replaced the bandwidth utilization with PSNR, packet delay with frame delay, and packet loss rate with frame drop rate. Because a PSNR gap of just less than 1 may represent a very large video quality gap, we convert PSNR back to MSE and add an offset to reflect the difference in frame quality. Since the most important aspect of real-time communication performance is the p2p frame delay, we directly use the average frame delay to calculate the QoE of frame delay. The definition of the frame-level QoE function is shown below:



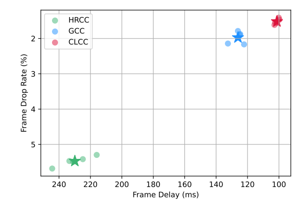
(a) Receiving Rate - Packet Delay



(b) Packet Loss Rate - Packet Delay

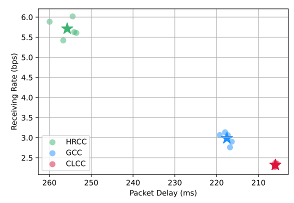


(c) Frame PSNR - VMAF Score

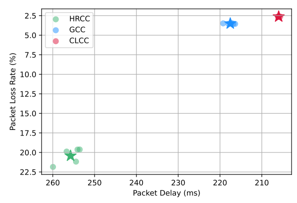


(d) Frame Drop Rate - Frame Delay

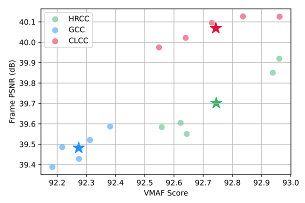
Fig. 3: Average Performace Results: Random Traces



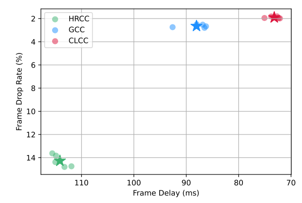
(a) Receiving Rate - Packet Delay



(b) Packet Loss Rate - Packet Delay

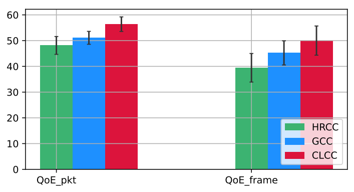


(c) Frame PSNR - VMAF Score

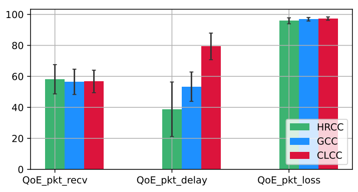


(d) Frame Drop Rate - Frame Delay

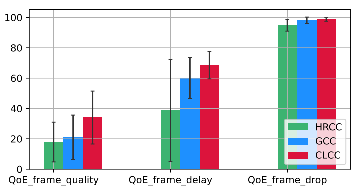
Fig. 4: Average Performace Results: LTE Traces



(a) QoE Result

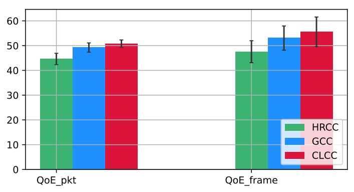


(b) Packet-Level QoE Composition

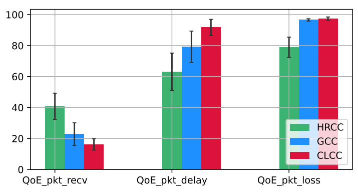


(c) Frame-Level QoE composition

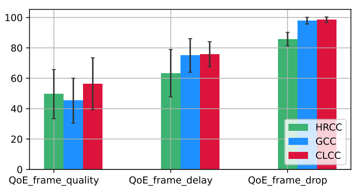
Fig. 5: QoE Results: Random Traces



(a) QoE Result



(b) Packet-Level QoE Composition



(c) Frame-Level QoE composition

Fig. 6: QoE Results: LTE Traces

$$QoE_{\text{quality}} = 100 \times \frac{\text{MSE(PSNR)} - \text{MSE}_{\text{min}}}{\text{MSE}_{\text{max}} - \text{MSE}_{\text{min}}} \tag{11}$$

$$QoE_{\text{delay}} = 100 \times \frac{d_{\text{avg}}}{3} \tag{12}$$

$$QoE_{\text{drop}} = 100 \times (1 - D) \tag{13}$$

$$QoE_{\text{frame}} = 0.2 \times QoE_{\text{quality}} + 0.2 \times QoE_{\text{delay}} + 0.3 \times QoE_{\text{drop}} \tag{14}$$

where MSE(PSNR) is the MSE value of the PSNR of that trace test, which is calculated by $10^{\frac{\text{PSNR}}{10}}$, MSE_{min} is the MSE value of the PSNR_{min}, MSE_{max} is the MSE value of the PSNR_{max}, d_{avg} is the average frame delay (ms) of that trace test and D is the average frame drop rate of that trace test. We set the values of PSNR_{min} and PSNR_{max} to be 36 and 42.

Figure 5 and 6 shows the results of the QoE test. CLCC outruns GCC and HRCC both on packet-level and frame-level. CLCC has a large advantage over the other two algorithms in both latency and video quality. The reason why advantage of CLCC's frame delay under LTE trace test is not as great as that of random trace is the bandwidth range of LTE trace is much larger than that of random trace, and the bandwidth fluctuation of LTE trace is not as frequent as that of random trace.

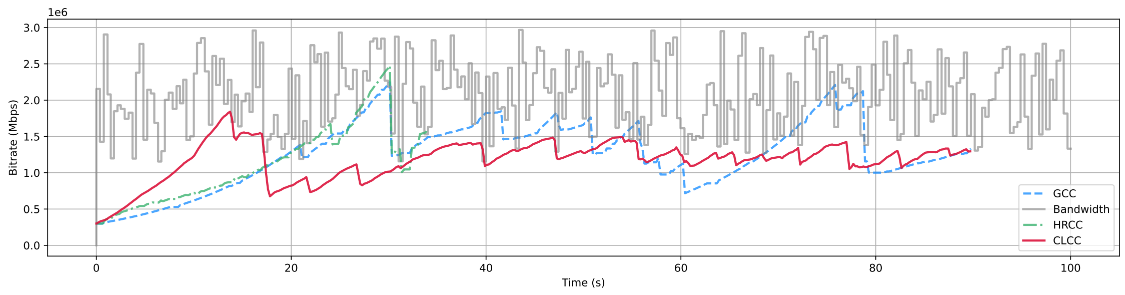


Fig. 7: Trace Behavior

D. *Trace Performance*

Figure 7 is a running simulation we choose to explain the behavior of CLCC. First of all, it can be seen that at the beginning of the simulation, CLCC climbs up faster than GCC and HRCC, proving that it can track the changing bandwidth better before the bitrate is saturated. Secondly, when it encounters a bandwidth cliff, it will reduce the bitrate lower than GCC and HRCC to stable the frame delay and packet delay. Thirdly, after the frame resolution is raised to the maximum level and the bitrate is saturated, CLCC will not actively increase the bitrate as before, but maintain a slow level. So when congestion occurs, it will be more easy for CLCC to adjust the bitrate to overcome it. HRCC is unable to run the entire simulation process because its robustness was not strong enough, resulting in poor performance in an environment that it is not familiar with enough.

VI. Conclusion

In this paper, we presented CLCC, a reinforcement learning based Cross-Layer Congestion Control for WebRTC that uses not only packet-level information, but also frame-level information to decide how to tune the bitrate. Tests on random traces prove that it can maintain low delay, low frame drop rate and high video quality at the same time, in other words, high QoE. Tests on LTE traces further prove the robustness of CLCC. We believe this paper can provide some inspiration for the low-latency and high-quality RTC application.

References

[1] *Video Will Account for 82% of All Internet Traffic by 2022, Cisco Says | Fierce Video.* https://www.fiercevideo.com/video/video-will-account-for-82-all-internet-traffic-by-2022-cisco-says.

[2] "Cisco Visual Networking Index: Forecast and Trends, 2017–2022". In: (2018), p. 38.

[3] Gaetano Carlucci et al. "Analysis and Design of the Google Congestion Control for Web Real-Time Communication (WebRTC)". In: *Proceedings of the 7th International Conference on Multimedia Systems*. Klagenfurt Austria: ACM, May 2016, pp. 1–12. ISBN: 978-1-4503-4297-1. DOI: 10.1145/2910017.2910605.

[4] Joyce Fang et al. "Reinforcement Learning for Bandwidth Estimation and Congestion Control in Real-Time Communications". In: *arXiv:1912.02222 [cs]* (Dec. 2019). arXiv: 1912.02222 [cs].

[5] Bo Wang et al. "A Hybrid Receiver-side Congestion Control Scheme for Web Real-time Communication". In: *Proceedings of the 12th ACM Multimedia Systems Conference*. Istanbul Turkey: ACM, June 2021, pp. 332–338. ISBN: 978-1-4503-8434-6. DOI: 10.1145/3458305.3479970.

[6] R. Jain. "Quality of Experience". In: *IEEE MultiMedia* 11.1 (Jan. 2004), pp. 96–95. ISSN: 1070-986X.

[7] Tianchi Huang et al. "QARC: Video Quality Aware Rate Control for Real-Time Video Streaming Based on Deep Reinforcement Learning". In: *Proceedings of the 26th ACM International Conference on Multimedia*. Seoul Republic of Korea: ACM, Oct. 2018, pp. 1208–1216. ISBN: 978-1-4503-5665-7. DOI: 10 . 1145 / 3240508 . 3240545.

[8] John Schulman et al. *Proximal Policy Optimization Algorithms*. 2017. DOI: 10.48550/ARXIV.1707.06347. URL: https://arxiv.org/abs/1707.06347.

[9] *Xiph.org Video Test Media [derf's collection].* https://media.xiph.org/video/derf/.

[10] Ravi Netravali et al. "Mahimahi: accurate record-and-replay for HTTP". In: *USENIX Association* (2015).

[11] Soheil Abbasloo, Chen-Yu Yen, and H. Jonathan Chao. "Wanna Make Your TCP Scheme Great for Cellular Networks? Let Machines Do It for You!" In: *IEEE Journal on Selected Areas in Communications* 39.1 (2021), pp. 265–279. DOI: 10 . 1109 / JSAC . 2020 . 3036958.