Playing Catch-Up: Evaluating Playback Speed Control in Low-Latency Live Streaming

Yu Liang[¶], Tomasz Lyko[¶], Mike Nilsson[§], Paul Farrow[§], Steve Appleby[§], Matthew Broadbent[*], Nicholas Race[¶]
[¶]Lancaster University, UK, [§]British Telecommunications, UK, [*]Edinburgh Napier University, UK
{y.liang20, t.lyko, n.race}@lancaster.ac.uk,{mike.nilsson, paul.farrow, steve.appleby}@bt.com, m.broadbent@napier.ac.uk

***Abstract*—The surge in popularity of live video streaming has spurred the development of various bitrate adaptation techniques, all aimed at enhancing user Quality of Experience (QoE). Compared to streaming Video-on-Demand, achieving low-latency live video streaming under fluctuating network conditions poses additional challenges. It requires finding the balance between rebuffering avoidance and latency, as a small client buffer is required to achieve low latency. Video players can also employ playback speed control to help optimize this balance. Specifically, when client buffer occupancy is high and hence latency is high, the player may increase playback speed to reduce the latency; and conversely, when client buffer occupancy is low and hence the risk of rebuffering is high, the player may reduce playback speed to increase buffer occupancy. Based on this rationale, a variety of playback speed control methods have been proposed. This paper evaluates, using a real-world testbed, the effectiveness of various playback speed control mechanisms when applied to a set of bitrate adaptation algorithms, with the evaluation also encompassing variations in target latency and network conditions. Our findings show a lack of coordination between adaptive bitrate (ABR) algorithms and playback speed control mechanisms. This leads us to conclude that there is a need for new playback speed control methods designed in conjunction with ABR algorithms.**

***Index Terms*—playback speed control, live streaming, bitrate adaption, low latency**

I. Introduction

The rapid growth of social media and improvements in Internet connectivity have propelled live streaming into a versatile tool for communication, entertainment, and more. Platforms such as YouTube Live [1] and Periscope [2] have leveraged this trend by enhancing user engagement through features like comments and chat in live video applications. Achieving low-latency live streaming poses a significant challenge, requiring adaptation to the constantly changing network conditions while also managing a constrained playback buffer. It is equally vital to carefully balance various QoE metrics, such as playback quality and playback latency [3].

In order to address this challenge, chunk-based streaming methodologies such as CMAF (Common Media Application Format) [4] and HTTP 1.1 chunked transfer encoding [5] have been introduced. In chunk-based streaming, video segments are subdivided into smaller chunks, facilitating encoding, transmission, and decoding in a seamless pipeline. A range of low-latency adaptive bitrate (ABR) algorithms have been proposed [6]–[8] that select an appropriate bitrate for each video segment, aiming to maximise the Quality of Experience. Additionally, video players employ a playback speed control module that allows the client to alter the video playback speed (faster/slower) in order to adjust the buffer depletion rate [9]. When the playback significantly trails behind the live event, the playback speed control module can increase the playback rate to catch up. Conversely, if latency is too low, resulting in a shallow video buffer and the risk of playback stalling, selecting a slower playback speed can gradually replenish the buffer to a secure state [10]. Therefore, the playback speed module plays a crucial role in controlling playback latency.

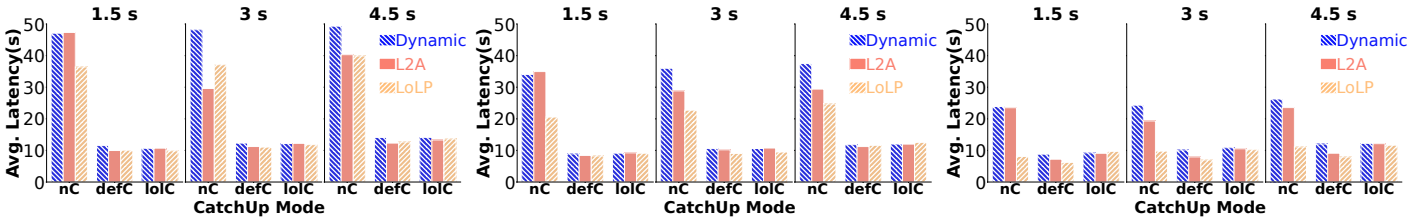
Despite the proposal of various playback speed control methods [7], [11], it remains unclear how these methods function in conjunction with different ABR algorithms. To assess the efficacy of existing playback speed control schemes, we undertake a thorough testbed-based evaluation by applying two such methods (the default playback control used by the Dash.js player and the playback control method proposed by LoLp [7]), to three representative ABR algorithms: LoLp [7], L2A [12], and Dynamic [13]. We have evaluated these playback control modules under realistic network conditions and various target latency settings. This paper makes the following contributions:

1) Comprehensive evaluation of playback speed control methods, applied to multiple low-latency ABR algorithms, under realistic network conditions and various target latency settings.
2) We present several insights into playback speed methods derived from the evaluation results, and motivated by these, propose future work that aims to enhance playback speed control.

II. Experimental Testbed Setup

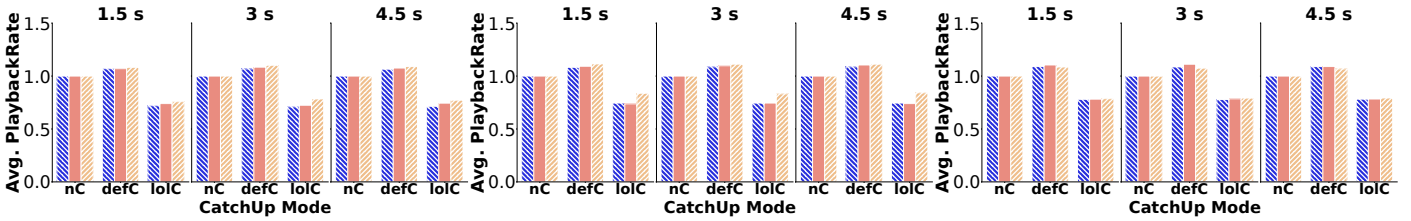
The experimental testbed is based on the LLL-CAdViSE architecture [16], modified to operate locally and with updates to the latest version of the Dash.js live streaming player (version 4.7.4) [15]. To enhance our video generation and analysis, we optimized server-side video processing and increased fault tolerance to ensure experiment accuracy. We also introduced an automated framework to efficiently conduct experiments and track related metrics such as download rates and buffer levels.

Player Setup. Dash.js acts as the live streaming player, configured with default settings. On the server side, video and audio content are dynamically generated in real-time, as per the LLL-CAdViSE architecture. Both the client and server components are hosted on a physical machine equipped



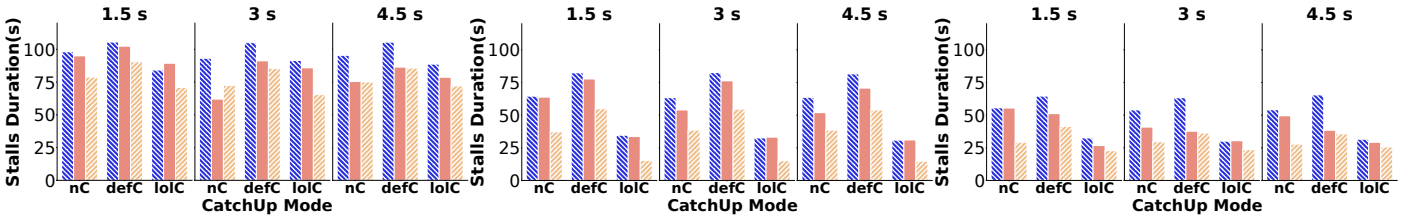
(a) Norway (b) Belgium (c) BT Sport

Fig. 1: Playback Latency with different speed control methods, ABR algorithms, network traces and target latencies.



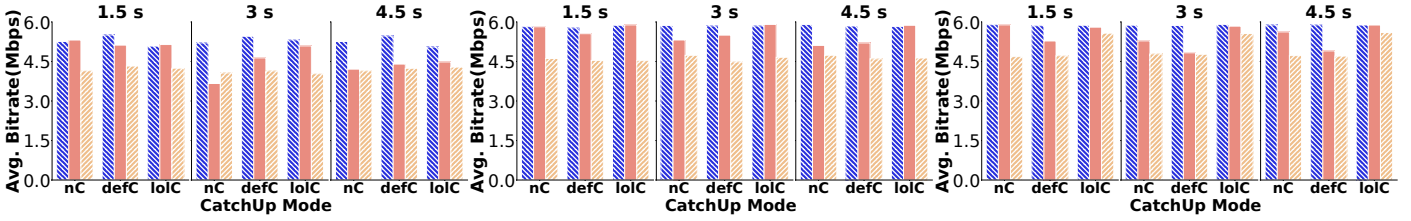
(a) Norway (b) Belgium (c) BT Sport

Fig. 2: Average playback rate with different speed control methods, ABR algorithms, network traces and target latencies.



(a) Norway (b) Belgium (c) BT Sport

Fig. 3: Playback stall time with different speed control methods, ABR algorithms, network traces and target latencies.



(a) Norway (b) Belgium (c) BT Sport

Fig. 4: Average Bitrate with different speed control methods, ABR algorithms, network traces and target latencies.

with a 13th Gen Intel(R) Core(TM) i9-13900KF processor running at 3.00 GHz with 24 cores and 64.0 GB of memory. The video content is provided at seven bitrates: 100Kbps, 365Kbps, 730Kbps, 1500Kbps, 3000Kbps, 4500Kbps, and 6000Kbps. The video content spans a duration of 200 seconds, divided into segments lasting 2 seconds each. Furthermore, each segment is subdivided into smaller chunks, each lasting 1 second. We set the target latency to 1.5s, 3s, and 4.5s, which represents the desired duration between the capture of a video frame and its display on the viewer's screen in live streaming.

To ensure the thoroughness of our practical tests, we use real-world network traces collected from a range of diverse environments. This includes a moving train in Norway [19], network conditions experienced in a car in Belgium [20], as well as traces obtained from CDN logs of the live BT Sport service [21]. Table I provides summary bandwidth information for the three network traces. We note that the Norway trace displays the highest level of instability, whereas the BT dataset appears to be the most stable. Bandwidth between the client and server is controlled by throttling via Chrome DevTools, facilitated by Selenium automation tools [18].

Speed Control Methods. We employ three playback speed modes: (i) No Catchup (*nC*), indicating playback speed remains at 1×; (ii) Default Catchup (*defC*) [15], which utilizes the catchup method proposed by Dash.js, increasing playback speed only when live latency deviates from the target value; and (iii) LoLp Catchup (*lolC*) [7], which controls playback speed based on current latency and buffer occupancy. It speeds up when current latency exceeds the target latency and buffer level surpasses a safe threshold, and slows down when current

TABLE I: Bandwidth information for the three network traces.

	Arithmetic Mean (Kbps)	Standard Deviation (Kbps)
Norway	2999.34	1865.15
Belgium	4116.16	1597.92
BT	4921.67	944.08

latency falls below the target latency and buffer level drops below the safe threshold. The parameters of these methods are set to their default settings.

III. Experimental Results

In this section, we evaluate the influence of different playback speed methods when using the three ABR algorithms with different network traces and values of target latency, using the metrics: playback latency, playback rate, rebuffering time and playback bitrate.

1) Playback Latency: Figure 1 shows the playback latency of three ABR algorithms across the three network traces. Our observation reveals a notable reduction in playback latency across the tested ABR algorithms with the playback speed module enabled. For instance, with the target latency set to 1.5s for the Norway trace, employing the playback speed modules from the default Dash.js player and LoLp leads to 374.8% and 342.1% reduction in playback latency for L2A, respectively, compared to when the playback speed control is disabled. Similarly, with the target latency set to 3s for the Belgium trace, enabling the playback speed modules from the default Dash.js player and LoLp results in 239.8% and 239.2% decrease in playback latency for Dynamic ABR, respectively, compared to disabling the playback speed control.

2) Playback Rate: Figure 2 shows the average playback rate. We observe that when the playback speed control module is disabled, the playback speed remains at 1x. We note that the speed control module introduced by LoLp tends to decrease the speed to mitigate the risk of buffer depletion and minimize the chance of playback interruption. Conversely, the default Dash.js speed control module tends to increase the playback speed to alleviate playback latency. For example, for the Belgium trace, the average playback rate of LoLp is 1.115 under the default speed control and 0.8375 under its own speed control when the target latency is 1.5s. Similarly, when the target latency is 3s, the average playback rate of L2A under the Belgium trace is 1.0975 under the default speed control, while it is 0.745 under the LoLp speed control.

3) Rebuffering Time: Figure 3 shows the total rebuffering duration. The default Dash.js playback speed control module, which tends to accelerate playback, results in increased rebuffering. Conversely, the stall time associated with the LoLp speed control module is lower, attributed to its slower playback rate compared to the default playback speed control. For the Dynamic ABR and BT trace, when the target latency is 1.5s, the average rebuffering time under the default speed control is 64.6s, compared to 32.6s under the LoLp speed control, marking a 98.2% increase.

4) Playback Bitrate: Figure 4 shows the playback bitrate of the three ABR algorithms with the playback speed control methods. These results indicate that there is not a strong correlation between the speed control method and the playback bitrate of the three ABR algorithms. This suggests that the integration of ABR algorithms and playback speed control methods is not tightly coupled.

IV. Discussion

Based on the evaluation results, we can make the following observations:

- Playback speed control plays a crucial role in reducing playback latency, ultimately enhancing live streaming interactivity. Additionally, we can observe that the playback speed variation under the LoLp playback speed control module is greater than that under the default Dash.js playback speed control.
- The default playback speed control module aims to achieve lower latency by speeding up playback, but this is at the expense of increased rebuffering. Conversely, the LoLp playback speed control module tends to slow down the playback speed to reduce rebuffering, albeit at the cost of higher latency. These findings suggest that current playback speed control methods struggle to strike a balance between playback latency and rebuffering.
- Current playback speed control lacks sufficient coordination with ABR algorithms. Without proper coordination, ABR algorithms face challenges in achieving a balanced tradeoff between different playback metrics, such as playback bitrate and playback latency.

Building upon these observations, we plan to enhance the playback speed control module as part of future work. This requires designing a novel speed control method capable of seamlessly integrating with different ABR algorithms to achieve an optimal balance across various QoE metrics and target latency settings when operating under diverse network conditions.

V. Conclusion

In this paper, we have presented a comprehensive evaluation of playback speed control methods using a real-world testbed environment. Playback speed control plays a critical role in enhancing user experience by mitigating latency in live streaming scenarios. Our analysis of current playback speed control modules across various network conditions, target latency settings, and ABR algorithms reveals significant limitations, particularly their struggle to strike a balance between playback latency and rebuffering time, and their lack of coordination with underlying ABR algorithms. In response to these challenges, we aim to enhance the playback speed control module in the future to further elevate the QoE in low-latency live streaming.

Acknowledgements

This work was supported by the UK Engineering and Physical Sciences Research Council (EPSRC) and British Telecom (BT).

REFERENCES

[1] Pires, Karine, and Gwendal Simon. "YouTube live and Twitch: a tour of user-generated live streaming systems." Proceedings of the 6th ACM multimedia systems conference. 2015.

[2] Siekkinen, Matti, Enrico Masala, and Teemu Kämäräinen. "A first look at quality of mobile live streaming experience: the case of periscope." Proceedings of the 2016 Internet Measurement Conference. 2016.

[3] Li, Weihe, et al. "A learning-based approach for video streaming over fluctuating networks with limited playback buffers." Computer Communications 214 (2024): 113-122.

[4] Lyko, Tomasz, et al. "Evaluation of CMAF in live streaming scenarios." Proceedings of the 30th ACM Workshop on Network and Operating Systems Support for Digital Audio and Video. 2020.

[5] Yadav, Praveen Kumar, et al. "Playing chunk-transferred DASH segments at low latency with QLive." Proceedings of the 12th ACM Multimedia Systems Conference. 2021.

[6] Gutterman, Craig, et al. "Stallion: video adaptation algorithm for low-latency video streaming." Proceedings of the 11th ACM Multimedia Systems Conference. 2020.

[7] Bentaleb, Abdelhak, et al. "Catching the Moment With LoL $^{+}$ in Twitch-Like Low-Latency Live Streaming Platforms." IEEE Transactions on Multimedia 24 (2021): 2300-2314.

[8] Lyko, Tomasz, et al. "Improving quality of experience in adaptive low latency live streaming." Multimedia Tools and Applications 83.6 (2024): 15957-15983.

[9] Aladag, Omer F., et al. "Content-aware playback speed control for low-latency live streaming of sports." Proceedings of the 12th ACM Multimedia Systems Conference. 2021.

[10] Sun, Liyang, et al. "Tightrope walking in low-latency live streaming: Optimal joint adaptation of video rate and playback speed." Proceedings of the 12th ACM Multimedia Systems Conference. 2021.

[11] Li, Yunlong, et al. "Fleet: improving quality of experience for low-latency live video streaming." IEEE Transactions on Circuits and Systems for Video Technology (2023).

[12] Karagkioules, Theo, et al. "Online learning for low-latency adaptive streaming." Proceedings of the 11th ACM Multimedia Systems Conference. 2020.

[13] Spiteri, Kevin, Ramesh Sitaraman, and Daniel Sparacio. "From theory to practice: Improving bitrate adaptation in the DASH reference player." ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM) 15.2s (2019): 1-29.

[14] W. Li, J. Huang, J. Liu, W. Jiang and J. Wang, "Learning Audio and Video Bitrate Selection Strategies via Explicit Requirements," IEEE Transactions on Mobile Computing, vol. 23, no. 4, pp. 2849-2863, 2024.

[15] "Dash.js." https://github.com/Dash-Industry-Forum/dash.js.

[16] B. Taraghi, H. Hellwagner and C. Timmerer, "LLL-CAdViSE: Live Low-Latency Cloud-Based Adaptive Video Streaming Evaluation Framework," IEEE Access, vol. 11, pp. 25723-25734, 2023.

[17] W. Li, J. Huang, Y. Liang, J. Liu and F. Gao, "Synthesizing Audio and Video Bitrate Selections via Learning from Actual Requirements," Proceedings of the IEEE ICME, 2012, pp. 1-6.

[18] Vila, Elior, Galia Novakova, and Diana Todorova. "Automation testing framework for web applications with Selenium WebDriver: Opportunities and threats." Proceedings of the International Conference on Advances in Image Processing. 2017.

[19] H. Riiser, P. Vigmostad, C. Griwodz, and P. Halvorsen, "Commute Path Bandwidth Traces from 3G Networks: Analysis and Applications," Proceedings of the ACM MMSys, 2013, pp. 114-118.

[20] J. van der Hooft et al., "HTTP/2-Based Adaptive Streaming of HEVC Video Over 4G/LTE Networks," IEEE Communications Letters, vol. 20, no. 11, pp. 2177-2180, 2016.

[21] "ABR Throughput Traces derived from CDN logs of BT Sport 1 service," https://github.com/lancs-net/ABR-Throughput-Traces.