ADAPTIVE STREAMING OF CONTENT-AWARE-ENCODED VIDEOS IN DASH.JS

Ali C. Begen[1,3], Mehmet N. Akcay[1], Abdelhak Bentaleb[2] and Alex Giladi[3]

[1] Ozyegin University (Turkey), [2] Atlastream (Singapore), [3] Comcast (USA)

ABSTRACT

In HTTP adaptive streaming, the client makes rate adaptation decisions based on the measured network bandwidth and buffer fullness. This simplifies the adaptation logic, however, it does often produce noticeable quality fluctuations during the streaming session. With content-aware encoding, one can improve the visual quality without increasing the total number of bits spent by carefully choosing where the bits are spent based on human perception. However, an adaptation logic that is unaware of the resulting variable-size segments may cause more stalls, which defeats the purpose of improving viewer experience through content-aware encoding. In this paper, we explain the design steps of a size-aware rate adaptation logic for one of the most popular DASH clients, namely dash.js, and show the improvements in rebuffering behavior and fetching top-resolution segments as a result of applying this logic.

INTRODUCTION

A common practice in adaptive streaming has been to run the transcoders in the constant bitrate (CBR) mode, resulting in more or less equal segment sizes for a given bitrate. That is, all the equal-duration segments produced out of each bandwidth-resolution pair in the encoding ladder (also called a representation in DASH terminology) would have a similar size in terms of bytes. This simplified the implementation of the rate adaptation for developers. However, natural content is not suitable for CBR encoding, since it exhibits a large variation in complexity in the temporal domain. Thus, during a playback session, even if the network conditions do not change and all the segments are requested from one particular representation, a naïve client that streams CBR-encoded segments inevitably experiences quality variation between the complex scenes or the ones involving more motion (e.g., an explosion) and the simple or static scenes (e.g., talking faces), unless the encoding bitrate is sufficiently high for even the complex scenes.

There is a good amount of prior research on this topic. The first study [2] introduced the concept of consistent-quality streaming and solved this problem using an optimization framework. The proposed method allocated the bits among the segments such that it yielded the best overall quality based on a metric that captured the presentation quality of a group of segments. Later studies (e.g., [3, 4, 5, 13]) tackled the same problem by making certain assumptions and simplifications to come up with a practical solution. For example, the problem at hand gets simplified when one produces each representation in a (near) constant-quality fashion (using capped variable bitrate (VBR) encoding) and takes the resulting segment size information into account in rate adaptation in addition to the available bandwidth and the amount of media there is in the playback buffer.

The available bandwidth varies over time and this is a significant constraint in any real-time streaming application, and since we can reliably predict the available bandwidth only for the next several seconds, not minutes, reliable rate adaptation decisions can only be made for a short horizon. In the case of live streaming, the output of the encoder is only known for a few segments and this limits the horizon further. For the segments that are yet to be encoded, actual sizes will be unknown. On the client side, we use the playback buffer as a breathing room for encoding bitrate variability. This buffer typically has a minimum and maximum size limit: if it is drained beyond the minimum threshold, the client should be conservative in rate adaptation as it may soon rebuffer and if more data arrives than the maximum size limit, the excess gets discarded. Normally, the goal is to keep a safe amount of data in the playback buffer between the minimum and maximum size limits.

In this paper, our target application is video-on-demand (VoD) streaming where the source content is pre-encoded and packaged, and the size information for each segment of every representation is known a priori. For this case, we develop a solution, implement it for dash.js [1] and test it in a number of different scenarios. It is worth noting that this solution is a component inside the rate adaptation logic already built in at the client side and it is orthogonal to the specific video codec/encoder. That is, it works with any open-source or commercial encoder and any video codec such as AVC, HEVC, VVC or AV1. We will investigate size-aware rate adaptation for live streaming in the future and publish the results in a follow-up work.

We believe in the reproducibility of the research. Therefore, our goal is to make the running code freely available [10]. We hope this code will eventually be merged into the main branch of dash.js and become a useful asset to developers and professionals.

Why dash.js

dash.js [1] is an initiative of the DASH Industry Forum to establish a production-quality framework for building media players that can play DASH content using client-side JavaScript libraries leveraging the Media Source Extensions API defined by the W3C. The dash.js code is covered by the BSD-3 license, thus, redistribution and use in source and binary forms, with or without modification is possible without cost or any license fees. Currently, it is one of the most popular open-source DASH client implementations with over a thousand forks and more than a hundred contributors. To date, several rate adaptation implementations have been developed for dash.js [12], which are referred to as adaptive bitrate (ABR) rules.

THE SIZE-AWARE RATE ADAPTATION (SARA) LOGIC

We propose SARA, a size-aware rate adaptation logic, aiming to improve the overall viewer experience and reduce the impact of content-aware encoding (CAE). SARA uses specific segment sizes, which are retrieved by the client ahead of time. These sizes are indicated in the Segment Index Box (`sidx`), defined in ISO/IEC 14496-12 and used in the DASH on-demand profile as well as in CMAF. For simplification purposes, our implementation emulates the above using an auxiliary manifest file in addition to the primary manifest (MPD) file. As normally, the MPD contains the adaptation sets, representations, segments, segment durations and URLs, and codec/encryption details, etc., while the auxiliary manifest stores the size information for each segment in every representation. Before the streaming session starts, the client fetches both manifest files.

We provide a detailed description of the SARA logic and its associated components. As depicted in Figure 1, the SARA logic is implemented as a new ABR rule in the dash.js client. First, in addition to the primary MPD, it also needs a segment-size list as the input. Second, SARA uses a slightly modified bandwidth measurement and smoothing process. Third, SARA uses the smoothed bandwidth values to predict the bandwidth for the next *z* (called horizon in this paper) download steps and makes the ABR decisions based on the predicted bandwidth. These new or modified components for SARA are indicated in gray color in Figure 1, where we also have a logger component to output diagnostics data from the client.

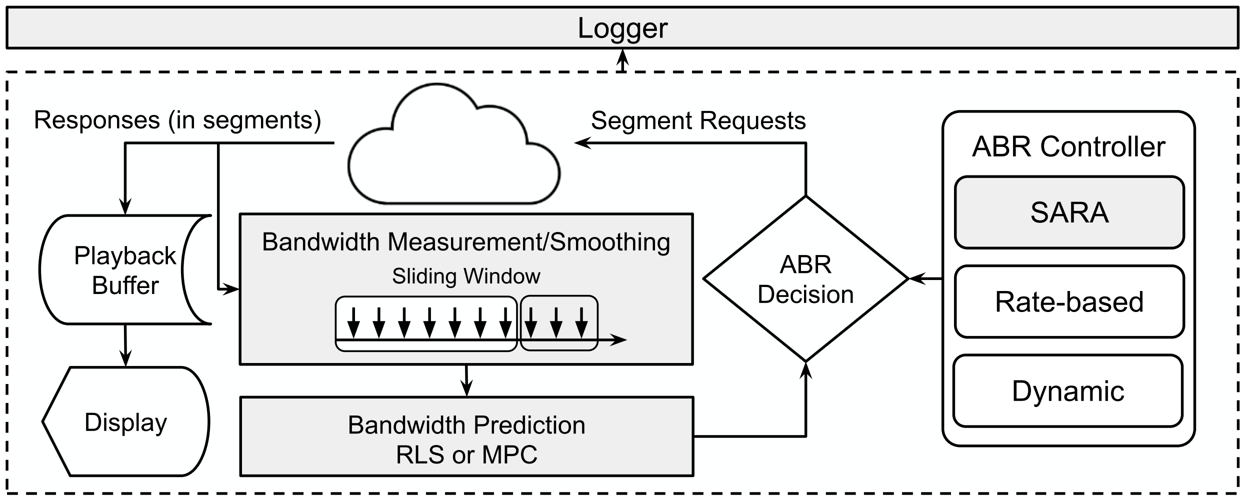

Figure 1: The SARA solution within the dash.js reference client. SARA's main components are highlighted in gray color.

By default, dash.js measures the bandwidth from the last downloaded segment by dividing its size by its download time and this method is called segment-based last bandwidth (SLBW). For smoothing, dash.js computes the exponentially weighted moving average (EWMA) of the last four of these measured values. For making an ABR decision, dash.js directly uses the computed smoothed value. That is, it does not perform any further prediction. These steps are illustrated in Figure 2(a).

SARA measures the bandwidth for the last downloaded segment in the same way (SLBW). However, for smoothing the measured values, SARA uses a method called segment-based sliding window moving average (SWMA), which simply computes the average of the last three segment downloads. The main difference is in the prediction stage, though. While dash.js has no built-in prediction component, we built two for SARA: Recursive Least Squares (RLS) [7] and Model Predictive Control (MPC) [8]. The steps for SARA are illustrated in Figure 2(b).

Bandwidth Measurement and Smoothing

For any rate adaptation logic, accurate bandwidth measurement and smoothing is essential. As explained above, after each downloaded segment, the bandwidth is computed by dividing the segment size by the delta between the times the download was completed and the request was sent. Due to many reasons such as the transient traffic in the network or the load on the server, measured bandwidth values may show a large variation. Thus, it is often a good idea to smooth out the measured values. For this purpose, the EWMA or SWMA method can be used. If desired, a safety factor of 90% can be used for the measured values as suggested in the default dash.js implementation.

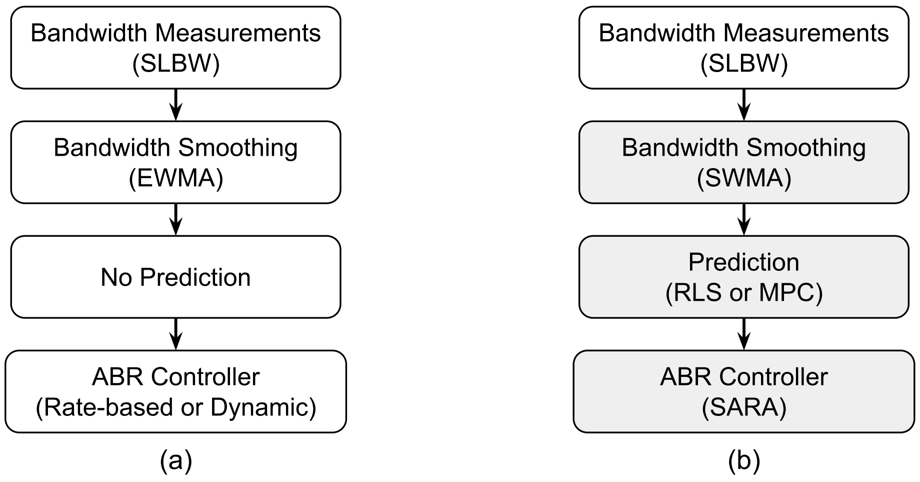
Bandwidth Measurements (SLBW)
Bandwidth Smoothing (EWMA)
No Prediction
ABR Controller (Rate-based or Dynamic)
(a)
Bandwidth Measurements (SLBW)
Bandwidth Smoothing (SWMA)
Prediction (RLS or MPC)
ABR Controller (SARA)
(b)

Figure 2: The steps involved in the default dash.js (a) and SARA (b).

Bandwidth Prediction

To perform a multi-step bandwidth prediction (for *z* steps), we developed two robust approaches: RLS and MPC, as shown in Figure 3. In a given streaming session, one of these approaches is picked by SARA and it uses the last *M* smoothed bandwidth values for making a prediction for the next *z* steps.

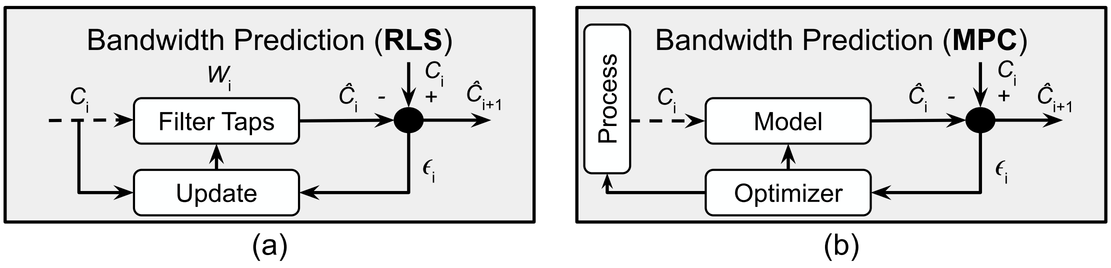

Figure 3: Bandwidth prediction approaches: RLS (a) and MPC (b).

Recursive Least Squares (RLS) Approach

RLS [7] is an online linear adaptive filter that recursively finds the filter coefficients that minimize a cost function relating to the input signals for accurate prediction. The RLS-based bandwidth prediction works in two phases at every segment download step *i*:

- Phase 1 (Filter Taps): It takes as an input a vector of the last *M* (*M* = 4 in this paper) smoothed bandwidth measurement values (denoted by C_i in Figure 3) given by the bandwidth measurement/smoothing component, and then recursively calculates the gain vector, inverse correlation matrix of the smoothed bandwidth measurement values and the estimated error (denoted by ϵ_i in Figure 3).
- Phase 2 (Update): These are used to update the filter taps vector (denoted by W_i in Figure 3) of length *M*, and finally return the bandwidth prediction for the next *z* steps.

RLS has three important benefits. First, it does not require extensive computational capabilities, allowing its deployment over practical real-time systems with commodity mobile devices. Second, it exhibits extremely fast convergence and does not need a prediction model. Third, it is robust against time-varying network conditions through its forgetting factor.

Model Predictive Control (MPC) Approach

MPC [8] is a widely deployed approach in the industry and has been proven to have a good forecasting and prediction performance. It falls into the model-based control category, which is known to be simple and practical, and it works in various environments. The MPC approach essentially selects the future bandwidth prediction by looking z steps ahead, considering the last M ($M = 4$ in this paper) smoothed bandwidth measurement values. At each segment download step i, the MPC approach works in two phases:

- Phase 1 (Process): It takes as an input a vector of the M most recent smoothed bandwidth values given by the bandwidth measurement/smoothing component.
- Phase 2 (Optimizer and Model): This represents the core of the MPC approach where given C_i and z, it predicts the future bandwidth using off-the-shelf MPC optimizer function $f_{mpc}(.)$ with a good accuracy (more than 90% in our case). Here, the objective function is to find the bandwidth prediction that minimizes the prediction error.

SARA ABR Rule

SARA selects the most suitable representation for the next segment to be downloaded from the list of available representations considering how much media is available in the playback buffer, minimum and maximum playback buffer sizes, predicted bandwidth and sizes of the candidate segments from the available representations.

Every downloaded segment is placed in the playback buffer, which cannot take more media than B_{max} (specified in terms of seconds). If the buffer level drops lower than a minimum size of B_{min} (also specified in terms of seconds), the client needs to be careful due to increased risk for a rebuffering. At step i, the SARA ABR rule computes the permutation table by looping through all the available representations and works as follows:

1. Compute the next download time (denoted by NDT_{i+1}) for each candidate segment from the available representations by dividing its size (denoted by NSS_{i+1}, which is available from the segment-size list) by the predicted bandwidth.
2. Compute the next buffer level (denoted by B_{i+1}) for each candidate segment from the available representations by adding the segment duration (denoted by τ) to the current buffer level (denoted by B_i) and subtracting the next download time (NDT_{i+1}).

SARA picks the segment from the representation with the highest encoding bitrate that ensures the next buffer level (B_{i+1}) $\geq$ B_{min}. If no candidate segment ensures this condition, the representation with the lowest encoding bitrate is selected. The overall flowchart of the SARA ABR rule is given in Figure 4, which is invoked after downloading each segment to determine the representation for the next segment. In the flowchart, the solid boxes handle the size-aware rate adaptation logic and the dashed boxes control the upshifting aggressiveness through the upshiftThreshold parameter, which is set to five in this paper.

There is an optional improvement for the SARA ABR rule to improve the likelihood of fetching higher-quality segments at the expense of a slightly increased risk of rebuffering, which is omitted from Figure 4 to simplify the flowchart. When the current buffer level (B_i) $\geq$ B_{min} and the actual encoding bitrate (denoted by E_{i+1} and computed by dividing the size of the candidate segment (NSS_{i+1}) by the segment duration (τ)) is equal to or higher than the predicted bandwidth, SARA may ignore checking the next buffer level (B_{i+1}) and pick the segment whose actual encoding bitrate (E_{i+1}) is the lowest among the ones that are higher than or equal to the predicted bandwidth.

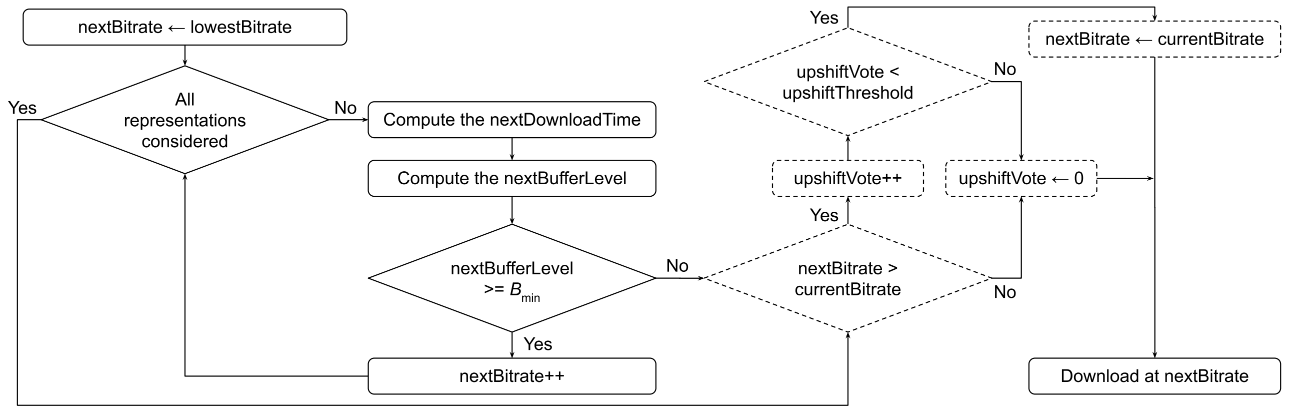
nextBitrate ← lowestBitrate
All representations considered
Yes
No
Compute the nextDownloadTime
Compute the nextBufferLevel
nextBufferLevel >= B_{min}
No
Yes
nextBitrate++
nextBitrate > currentBitrate
Yes
No
upshiftVote++
upshiftVote ← 0
upshiftVote < upshiftThreshold
Yes
No
nextBitrate ← currentBitrate
Download at nextBitrate

Figure 4: Flowchart of the SARA ABR rule.

SARA Working Example

Consider a hypothetical example with two-second segments and four representations advertised at 300, 500, 1000 and 2500 Kbps. The client predicts the next bandwidth to be 500 Kbps and is supposed to pick one of the candidate segments, whose actual sizes are of 200, 250, 500 and 1250 Kbits, respectively. Assume B_{min} is set to two seconds.

We demonstrate two cases. When the current buffer level is one second, SARA computes the next buffer levels as given in Table 1. Following the logic given in Figure 4, SARA picks the segment from the 1000 Kbps representation, whereas the rate-based logic would have picked the segment from the 500 Kbps representation.

Advertised Encoding Bitrate (Kbps)	Advertised Segment Size (Kbits)	Next Segment Size (Kbits)	Predicted Bandwidth (Kbps)	Current Buffer Level (s)	Next Download Time (s)	Next Buffer Level (s)
300	600	200	500	1.0	0.4	2.6
500	1000	250	500	1.0	0.5	2.5
1000	2000	500	500	1.0	1.0	2.0
2500	5000	1250	500	1.0	2.5	0.5

Table 1: SARA's computations when the current buffer level is one second.

Following the same logic, when the current buffer level is 10 seconds, SARA computes the next buffer levels as given in Table 2 and picks the segment from the 2500 Kbps-representation, whereas the rate-based logic would have again picked the segment from the 500 Kbps-representation.

Advertised Encoding Bitrate (Kbps)	Advertised Segment Size (Kbits)	Next Segment Size (Kbits)	Predicted Bandwidth (Kbps)	Current Buffer Level (s)	Next Download Time (s)	Next Buffer Level (s)
300	600	200	500	10.0	0.4	11.6
500	1000	250	500	10.0	0.5	11.5
1000	2000	500	500	10.0	1.0	11.0
2500	5000	1250	500	10.0	2.5	9.5

Table 2: SARA's computations when the current buffer level is 10 seconds.

RESULTS

For the testing, we used a virtualized DASH streaming setup, which can be obtained from [6]. In this setup, we created three different nodes for the server to serve the DASH content, client to run different ABR rules and network emulator to throttle the bandwidth according to a bandwidth profile. For the bandwidth profiles, we used three of them: Cascade, Twitch and LTE, which are plotted for a duration of 10 minutes in Figure 5. These bandwidth profiles are repurposed for this work from [11].

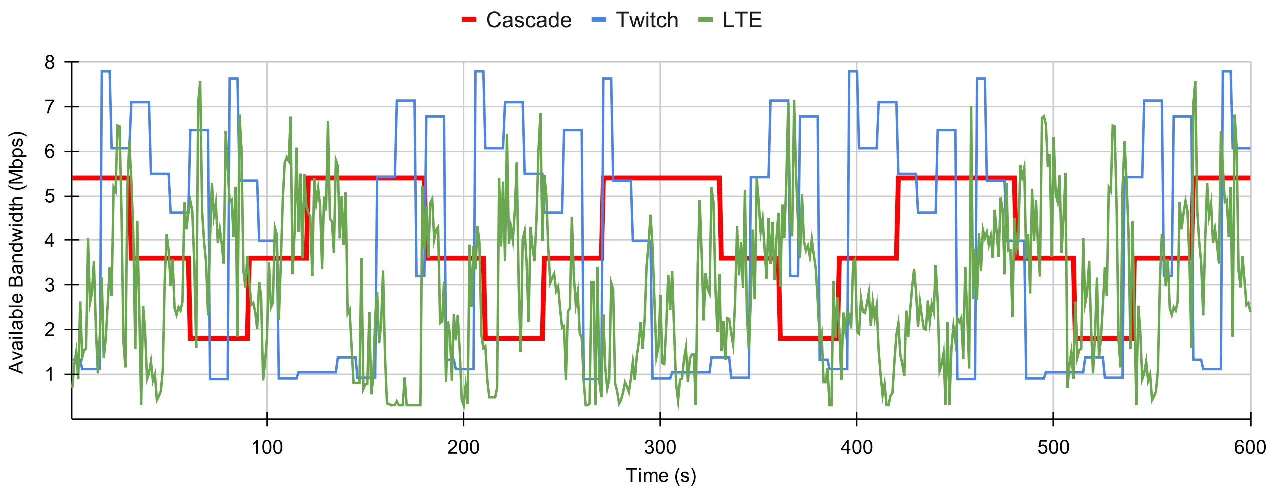

Figure 5: Bandwidth profiles used in the testing.

For the test content, we created a 10-minute video mixed from a variety of videos that exhibited different scene complexities. The mixed video was encoded by a professional encoder with the CAE mode enabled. The choice of codec was AVC, and the frame rate and segment duration were set to 25 fps and two seconds, respectively. We generated four representations at 360p at 850 Kbps, 540p at 1690 Kbps, 720p at 2720 Kbps and 1080p at 5540 Kbps, where the bitrates indicate the average bitrate at that resolution, although the segment encoding bitrates varied significantly due to CAE as shown in Figure 6.

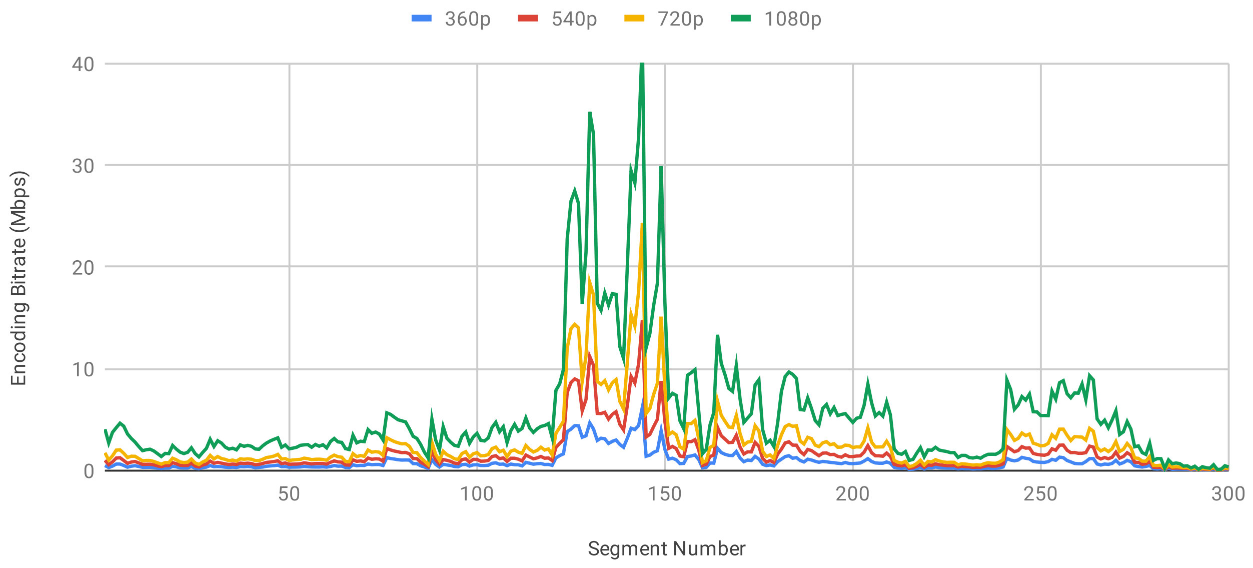

Figure 6: Segment encoding bitrates for the test content.

To compare different ABR rules, we recorded the total downloaded (TD) video size (in megabytes), total rebuffering duration (TRD, in seconds), long-rebuffering count (LRC), short-rebuffering count (SRC) and percentage of segments rendered at 720p or higher (HD). Here, a short rebuffering refers to a rebuffering that lasts less than a single frame duration (40 milliseconds at 25 fps) and all other rebufferings are considered a long rebuffering. We compare two built-in rate ABR rules (namely, rate-based and dynamic) with three flavors of the SARA ABR rule (namely, SARA-Basic, SARA-RLS and SARA-MPC). Here, SARA-Basic considers segment sizes in rate adaptation but does not use the RLS or MPC bandwidth prediction, it rather simply uses the smoothed bandwidth value from Figure 2(b).

In the client, we used ($B_{\min}$, $B_{\max}$) = (6, 30) in seconds. The playback started after the duration of the media in the playback buffer exceeded $B_{\min}$.

Comparisons

Table 3 and Figure 7 present the results for five ABR rules for three bandwidth profiles for the 10-minute test content. In these runs, SARA-RLS and SARA-MPC performed a two-step bandwidth prediction. Further, SARA-RLS used the following parameters: initial input variance estimate parameter for the inverse correlation matrix of size M x M (σ) = 0.001 and forgetting factor (λ) = 0.999. These parameters are further discussed later in this section.

There are a number of important observations from these results:

- All three SARA ABR rules experience not only a smaller total number of rebufferings but also a shorter total duration of rebufferings. This vastly improves the viewer experience. This is largely due to the fact that SARA takes segment sizes into account in its decisions such that it can refrain from fetching larger-than-expected segments if there is a risk of rebuffering and it can carry on fetching smaller-than-expected segments if there is no risk of rebuffering.
- The rate-based and dynamic ABR rules have the best HD performance for the Cascade profile (i.e., when the network is more stable) but are never able to reach 1080p quality, while SARA-Basic, SARA-RLS and SARA-MPC fetch 87, 115 and 120 segments, respectively, at 1080p.
- For the Twitch profile, the rate-based and dynamic ABR rules can fetch few 1080p segments and the SARA flavors fetch about half of the segments at 1080p.
- For the LTE profile, the rate-based and dynamic ABR rules fetch very few 1080p segments whereas the SARA flavors fetch about one third of the segments at 1080p.

If we check the logs for the segments between #100 and #200 (where the test content shows the largest encoding bitrate variation as shown in Figure 6), we observe that most rebufferings happen between segments #100 and #200 for the rate-based and dynamic ABR rules. The HD performance for the rate-based and dynamic ABR rules also drops during this period meaning that the share of the 720p and 1080p segments fetched decreases. At the same time, all three SARA flavors increase the share of the HD segments proving that SARA can easily deal with segments encoded with CAE.

Bandwidth Profile	ABR Rule	TD (MB)	TRD (s)	LRC	SRC	HD (%)
Cascade	Rate-based	184.00	4.92	6	11	88.33
	Dynamic	165.88	3.17	10	8	80.67
	SARA-Basic	178.56	1.26	3	5	65.33
	SARA-RLS	196.10	0.40	1	1	65.00
	SARA-MPC	198.06	0.80	2	1	66.67
Twitch	Rate-based	180.88	2.69	6	8	61.67
	Dynamic	186.57	2.54	10	6	68.00
	SARA-Basic	231.48	1.40	3	7	74.00
	SARA-RLS	199.01	1.00	3	4	55.00
	SARA-MPC	203.62	1.02	4	5	51.33
LTE	Rate-based	188.78	1.90	5	12	87.00
	Dynamic	130.19	1.30	6	7	53.00
	SARA-Basic	140.38	1.27	5	6	40.67
	SARA-RLS	146.10	1.00	3	7	43.67
	SARA-MPC	161.47	0.95	4	5	62.33

Table 3: Comparison of the five ABR rules for the three bandwidth profiles.

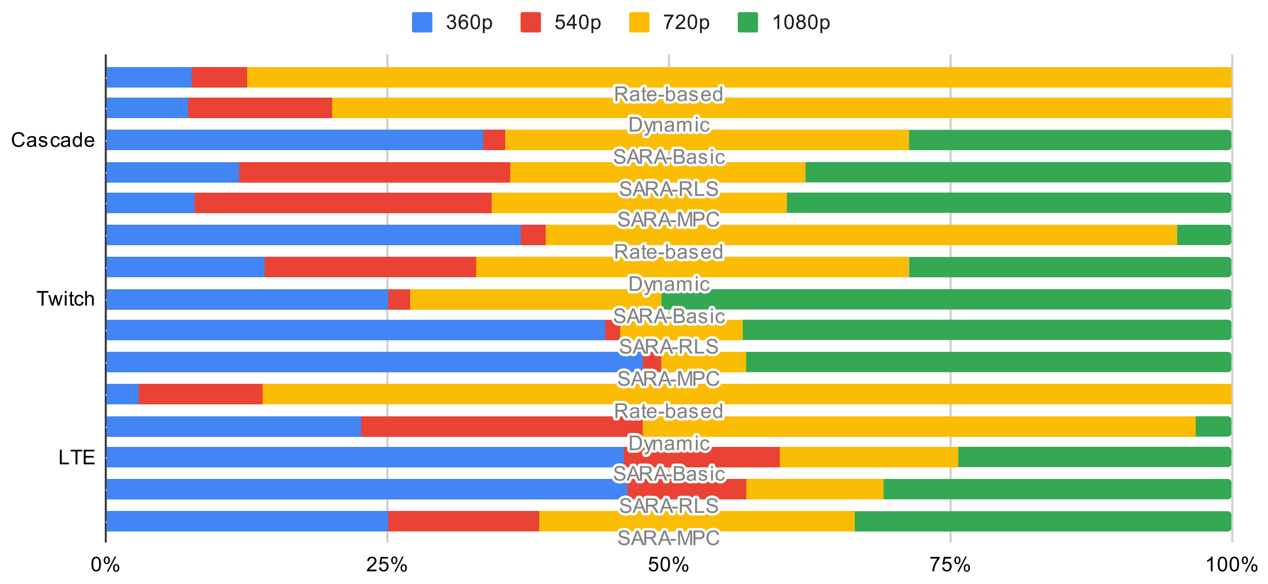

Figure 7: Distribution of the segment resolutions for the five ABR rules.

Single vs. Multi-Step Bandwidth Prediction

As it can be seen in Table 4, having a two-step (as opposed to a single-step) prediction improves the streaming performance for both SARA-RLS and SARA-MPC. The bandwidth prediction accuracy has a direct effect on rebuffering duration and HD performance. The results also show that SARA-MPC produces better HD performance results than SARA-RLS when the network conditions become more dynamic (e.g., the LTE profile compared to the Cascade profile), since in this case the RLS approach makes more conservative predictions and underperforms in upshifting.

Bandwidth Profile	ABR Rule	TD (MB)	TRD (s)	LRC	SRC	HD (%)
Cascade	SARA-RLS-1	192.23	0.73	2	1	63.66
	SARA-RLS-2	196.10	0.40	1	1	65.00
	SARA-MPC-1	198.25	0.91	3	1	65.00
	SARA-MPC-2	198.06	0.80	2	1	66.67
Twitch	SARA-RLS-1	203.55	1.25	4	4	54.00
	SARA-RLS-2	199.01	1.00	3	4	55.00
	SARA-MPC-1	199.86	1.15	4	6	50.00
	SARA-MPC-2	203.62	1.02	4	5	51.33
LTE	SARA-RLS-1	139.86	1.25	4	7	38.00
	SARA-RLS-2	146.10	1.00	3	7	43.67
	SARA-MPC-1	149.89	1.33	5	6	48.66
	SARA-MPC-2	161.47	0.95	4	5	62.33

Table 4: Single vs. multi-step bandwidth prediction (SARA-RLS: $\sigma = 0.001$, $\lambda = 0.999$).

Parameter Tuning for SARA-RLS

In the RLS approach, an important parameter is the forgetting factor (λ), which is defined as at what weight(s) previous smoothed values contribute to the result [7]. A lower forgetting value means the earlier samples are forgotten faster, which is a more appropriate behavior for more dynamic samples. This is confirmed by checking the results for the Twitch and LTE profiles in Table 5, as the results with the (σ_2, λ_2) pair are better considering the percentage of the HD segments and total rebuffering duration, whereas for the Cascade profile, the results with the (σ_1, λ_1) pair are better.

Bandwidth Profile	ABR Rule	TD (MB)	TRD (s)	LRC	SRC	HD (%)
Cascade	RLS-2 (σ_1,λ_1)	196.10	0.40	1	1	65.00
	RLS-2 (σ_2,λ_2)	191.87	0.75	3	3	63.66
Twitch	RLS-2 (σ_1,λ_1)	199.01	1.00	3	4	55.00
	RLS-2 (σ_2,λ_2)	199.57	0.75	3	3	58.00
LTE	RLS-2 (σ_1,λ_1)	146.10	1.00	3	7	43.67
	RLS-2 (σ_2,λ_2)	144.30	0.67	3	5	50.00

Table 5: SARA-RLS two-step prediction with $\sigma_1 = 0.001$, $\lambda_1 = 0.999$, $\sigma_2 = 0.01$, $\lambda_2 = 0.99$.

SOURCE CODE and DEMO

As mentioned in the introduction, we are pleased to offer the source code and an offline demo for SARA at [10]. Future improvements including any bug fixes will also be documented at the same link. We hope the SARA ABR rule will eventually be integrated to the official dash.js repo and used by developers as the share of content-aware-encoded videos increases. The offline demo can be used to play the output videos resulting from different ABR rules for different bandwidth profiles. On the demo page, several viewer experience metrics are reported and the bitrate is plotted in a real-time graph.

CONCLUDING REMARKS

With content-aware encoding, it is possible to use the bits in a smarter way to improve the visual quality of the video content. However, this creates a problem during streaming, most of the time ending with more rebufferings and lower-quality segment downloads. To address these problems, we developed a size-aware rate adaptation (called SARA) logic that uses the segment size information and two different multi-step bandwidth prediction approaches. Our results show major improvements in rebuffering behavior as well as in the percentage of the playback time spent at the top resolution. The results are promising, and we plan on implementing SARA for the RDK's AAMP player as it is another widely used open-source player [9].

REFERENCES

[1] DASH-IF. DASH Reference Client. [Online] Available: https://reference.dashif.org/dash.js/. Accessed on May 10, 2021

[2] Zhi Li, Ali C. Begen, Joshua Gahm, Yufeng Shan, Bruce Osler and David Oran, "Streaming video over HTTP with consistent quality," in Proc. ACM Multimedia Systems Conf. (MMSys), Singapore, Mar. 2014 (DOI=10.1145/2557642.2557658)

[3] Ali C. Begen, "Quality-aware HTTP adaptive streaming," in Proc. Int. Broadcasting Convention Conf. (IBC), Amsterdam, The Netherlands, Sept. 2015 (DOI=10.1049/ibc.2015.0004)

[4] Yanyuan Qin, Shuai Hao, Krishna R. Pattipati, Feng Qian, Subhabrata Sen, Bing Wang and Chaoqun Yue, "Quality-aware strategies for optimizing ABR video streaming QoE and reducing data usage," in Proc. ACM Multimedia Systems Conf. (MMSys), 2019

[5] W. Cooper, S. Farrell and K. Subramanian, "QBR metadata to improve streaming efficiency and quality," SMPTE Annual Technical Conf. and Exh., 2017

[6] DASH-Streaming-Setup. [Online] Available: https://github.com/fg-inet/DASH-streaming-setup. Accessed on May 10, 2021

[7] Engel, Y., Mannor, S., & Meir, R., "The kernel recursive least-squares algorithm," IEEE Trans. Signal Processing, 52(8), 2275-2285, Aug. 2004

[8] Agachi, P. S., Cristea, M. V., Csavdari, A. A. and Szilagyi, B., "2. Model predictive control". Advanced Process Engineering Control: De Gruyter, 2016, pp. 32-74 (DOI=10.1515/9783110306637-003)

[9] RDK Central. Advanced Adaptive Media Player. [Online] Available: https://developer.rdkcentral.com/video/documentation/components/open-sourced_components/advanced_adaptive_media_player/. Accessed on May 10, 2021

[10] SARA - IBC'21 Demo Page. [Online] Available: http://streaming.university/demo/ibc21-sara/. Accessed on May 10, 2021

[11] Abdelhak Bentaleb, Mehmet N. Akcay, May Lim, Ali C. Begen and Roger Zimmermann, "Catching the moment with LoL+ in Twitch-like low-latency live streaming platforms," IEEE Trans. Multimedia, to appear (DOI=10.1109/TMM.2021.3079288)

[12] Abdelhak Bentaleb, Bayan Taani, Ali C. Begen, Christian Timmerer and Roger Zimmermann, "A survey on bitrate adaptation schemes for streaming media over HTTP," IEEE Commun. Surveys Tuts., vol. 21/1, pp. 562-585, Firstquarter 2019 (DOI=10.1109/COMST.2018.2862938)

[13] Parikshit Juluri, Venkatesh Tamarapalli, and Deep Medhi, "SARA: Segment aware rate adaptation algorithm for dynamic adaptive streaming over HTTP," in Proc. IEEE Int. Conf. Communication Workshop (ICCW), 2015