# GAME THEORY BASED BITRATE ADAPTATION FOR DASH.JS REFERENCE PLAYER

*Abdelhak Bentaleb*[⋆]*, Ali C. Begen*[‡] *and Roger Zimmermann*[⋆]
[⋆]National University of Singapore, [‡]Ozyegin University
{bentaleb,rogerz}@comp.nus.edu.sg, ali.begen@ozyegin.edu.tr

## ABSTRACT

Most existing DASH adaptive bitrate (ABR) schemes are designed to behave in their own self-interest and do not perform consistently in all network environments. In this work, we provide a practical implementation, materials and demonstration of a game theoretical ABR scheme, termed GTA [1], in the dash.js reference player. The GTA approach optimizes the viewer experience across multiple players without requiring explicit communication, and maintains a high playback bitrate while reducing startup delay, and minimizing quality switches and stalls.

***Index Terms***— HAS, DASH, QoE, game theory, ABR.

## 1. INTRODUCTION

The advances in HTTP adaptive streaming (HAS) technologies are offset by the tremendous increase in demand for high-quality video. User-centric objectives such as personalization and improving viewer quality of experience (QoE) represent the primary concern for content and over-the-top (OTT) providers. Designing a robust and personalized video delivery solution suitable for large scale deployments is an important task.

As depicted in Fig. 1, in a Dynamic Adaptive Streaming over HTTP (DASH) system, two essential entities exist: the DASH server and clients. The DASH server is a plain HTTP server that stores the video content. Each video is segmented into small segments (2–15 seconds) and encoded into different representations (*i.e.*, bitrates and resolutions) along with the manifest files (media presentation description, MPD). A DASH client requests the MPD file of a content and then starts downloading the segments sequentially. The scheduler orchestrates the downloading process using the ABR controller to select the most appropriate representation for the next segment to be fetched. In doing so, the ABR controller may rely on a variety of heuristics [2] such as the estimated throughput and buffer occupancy.

To date, different client-driven ABR schemes have been proposed [3]. These schemes can be classified into three main categories based on the used heuristics: throughput-based, buffer-based and hybrid solutions. Existing client-driven ABR schemes achieve a good performance under certain circumstances while promoting the viewer's QoE with varying success. However, many of them suffer from either $(i)$ inconsistent performance in various network environments, settings and operating regimes, or $(ii)$ the use of fixed heuristics that broadly depend on specific settings and run under implicit assumptions or need extensive parameter tuning. Achieving an acceptable viewer QoE with the best trade-off between different QoE metrics and ABR objectives, and providing group fairness remains an open challenge.

Consequently, adapting the existing ABR schemes to different operating regimes is often difficult, and thus, DASH clients may suffer from low QoE, *i.e.*, frequent stalls, bitrate switches and long startup delays. In contrast to the existing ABR schemes, we designed GTA [1] based on game theory (GT) mechanisms [4]. The essential benefit of GTA is that it can concurrently optimize the QoE of multiple clients without requiring explicit communication between them. Deriving from our prior work [1], we demonstrate on a broad set of real-world network traces, different operating regimes and settings that GTA outperforms its competitors in the average QoE by 38.5%, and in the video stability by 62% without any stalls and no increase in startup delays. For more information on the GTA and its materials (the code source, demo, video samples, throughput profiles), visit our Web site [5].

## 2. THE GTA SOLUTION

For the theoretical foundation of GTA, we refer the readers to [1]. Our solution is implemented in the dash.js reference player [6], within three new main classes as shown in Fig. 2.

### 2.1. Inputs Class

This class manages the set of input variables that are obtained from the environment and GTA components. The bandwidth value is obtained from the PANDA and CS2P estimators, and the bitrate and SSIMplus [7] lists are extracted from the MPD file. The Buffer Controller provides the buffer size (see Fig. 1). The content type (CT), device resolution (DR) and service plan type (SPT) are taken from the video properties, device capabilities and subscription plan[1], respectively.

### 2.2. GTA Components Class

The GTA scheme comprises five essential components (gray boxes in Fig. 2), which are:

1. PANDA & CS2P Estimators: This component implements two algorithms for the throughput estimation of a segment using PANDA [8] and CS2P [9].

[1]Each client subscribes to one of the plans offered by the service provider.

978-1-5386-1737-3/18/$31.00 

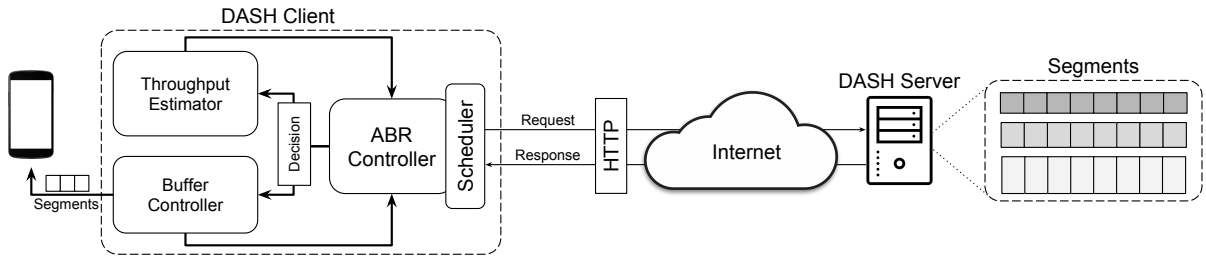


**Fig. 1**. The DASH delivery architecture with a dash.js player.

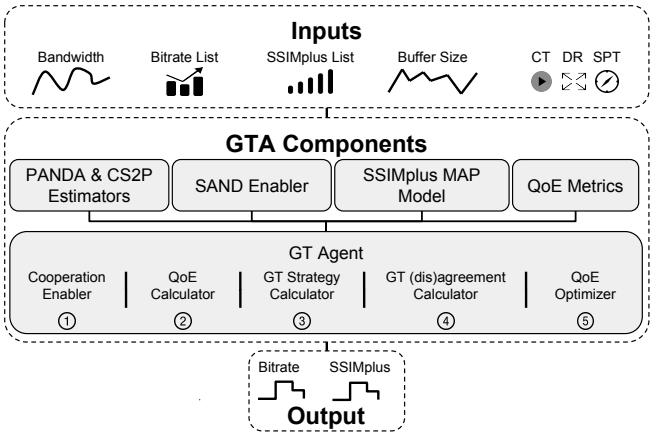


**Fig. 2**. A general overview of the GTA design.

2. SAND Enabler: This allows the integration of GTA with Server and Network-assisted DASH [10] by implementing various SAND-enabled communication interfaces.
3. SSIMplus MAP Model: This mplements a perceptual quality mapping model [11] based on the Structural Similarity Index plus (SSIMplus) [7]. This model maps three distinctive features (CT, DR and SPT) into one common perceptual quality space in which a set of clusters are constructed. Then, each DASH client joins one of the clusters when the cooperative mode is enabled.
4. QoE Metrics: This component integrates the QoE model and its metrics (*i.e.*, average quality, startup delay, stalls and quality switches) proposed in [11].
5. GT Agent: This develops the GTA decision rules for selecting the best bitrate based on a set of design steps (① - ⑤ in Fig. 2), which are explained in detail in [1].

Note that GTA operates in two modes: ($i$) the *cooperative* mode is enabled when multiple GTA players are deployed in a shared network environment, and ($ii$) the *non-cooperative* mode is activated when non-GTA players compete with a GTA player for the available bandwidth.

### 2.3. Output Class

At every segment downloading step, GTA outputs the optimal ABR decision by reaching the Pareto optimal (PO) Nash bargaining solution (NBS) [4]. The taken ABR decision represents the best bitrate with its corresponding SSIMplus perceptual quality.

## 3. IMPLEMENTATION

### 3.1. Implementation Overview

We provide a practical implementation of GTA (called *GTA.js*) on top of the open source, JavaScript-based dash.js reference player (v2.5) [6]. Fig. 3 highlights our implementation where the light gray boxes represent the additional classes and the dark gray ones denote the modified classes. These classes are explained at a conceptual level as follows: ($i$) At any segment downloading step, the Scheduler Controller manages and creates a request based on the bitrate decision that was made by an ABR rule. After that, it places the request in the XMLHttpRequest for requesting the desired segment from the corresponding DASH server via the method `xhr.send()`. The BaseURL Selector selects the DASH server from the set of available servers that are presented in the Manifest Attribute. ($ii$) The ABR Controller includes a set of ABR rules (*e.g.*, throughput-based, buffer-based and hybrid), which are used to select the appropriate bitrate decision respecting the throughput estimate and the buffer size for the next segment to be downloaded. ($iii$) The Buffer Controller monitors the playback buffer occupancy and maintains it within the safe region. ($iv$) The Throughput Estimator estimates the most recent segment's throughput and also maintains the mean of the last three throughput estimations. ($v$) The Inputs, GTA Components, Output and Logger are the new classes where the former three are used to select the bitrate based on the GT decision rule (see Section 2), while the latter is used to store different statuses of the player such as buffer size, throughput estimate and QoE metrics.

### 3.2. GTA dash.js Integration

A detailed description of how to integrate GTA with dash.js as well as the testing guidelines are provided in the appendix of [1]. Succinctly, the programmer needs to follow these steps: ($i$) Download the dash.js player [6] and *GTA.js* from [5]. ($ii$) Add *GTA.js* to the /src/streaming/rules/abr folder. ($iii$) Replicate the same structure of *BolaRule.js* to integrate *GTA.js* by modifying the

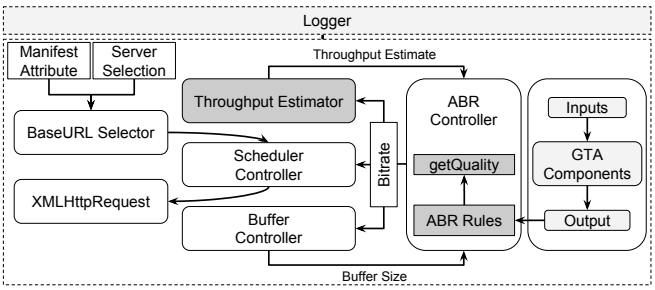


**Fig. 3**. GTA implementation in the dash.js reference player. The light gray boxes represent the new classes, and the dark gray ones are the modified classes.

following files: *ABRRulesCollection.js*, *MediaPlayer.js* and *MediaPlayerModel.js*. $(iv)$ In the file *MediaPlayerModel.js*, change the maximum (BUFFER_TO_KEEP) and minimum (DEFAULT_MIN_BUFFER_TIME) buffer occupancy thresholds to 36 and 12 seconds, respectively. Also, the programmer can use other buffer thresholds by modifying the corresponding buffer threshold values in both *GTA.js* and *MediaPlayerModel.js*. $(v)$ To log different player statuses, add *FileSaver.js*, *Blob.js* and *StorStatus.js* to the /samples/dash-if-reference-player/app/ folder. Then, call the function *savegtainfo()* (or *saveBOLAinfo()*, *saveRateinfo()* in the case of buffer-based and throughput-based schemes, respectively) from the function *onPlaybackEnded()* of the *PlaybackController.js*. This allows the player to log CSV files (*i.e.*, X_GTAinfoStatus_CT.csv, X_RateinfoStatus_CT.csv, X_BOLAinfoStatus_CT.csv, *etc.*) at the end of the streaming session that then can be used for graphing. For comparison, we also included *bba.js* and *elastic.js*, and both require the previous steps for their integration with dash.js.

## 4. EVALUATION AND RESULTS SUMMARY

To evaluate the efficiency of GTA, we conducted more than 500 hours of on-demand (VoD) video streaming experiments. These experiments are divided into two main test scenarios: one client and multiple clients. In particular, GTA is evaluated against the existing ABR schemes using large-scale, real-world, trace-driven experiments. Each experiment covers a broad set of realistic throughput variability profiles, different operating regimes and settings such as CT = {animation, movie, documentary, sports, news}, SPT = {platinum, gold, silver, bronze, basic}, DR = {240p, 360p, 480p, 720p, 1080p}, video parameters (*e.g.*, bitrate level, content resolution, and SSIMplus perceptual quality), and the number of clients.

### 4.1. Testing Methodology

We explain the testing methodology briefly:

- Throughput Profiles: We used four throughput profiles from realistic public datasets and synthetic hidden Markov models. These datasets [1] include DASH-IF test cases and guidelines, FCC Broadband, 3G/HSDPA Mobile and Synthetic. To avoid trivial ABR decisions where the maximum decision would always be the optimal solution, we filtered the throughput values and considered only throughput traces whose values were in the range of [0.25,...,6] Mbps. Also, we used various inter-variation durations {1, 4, 10, 30, 60, 75} seconds, round-trip times (RTT) {38, 50, 75, 88, 100} ms, and packet loss ratios {0.06, 0.08, 0.09, 0.12} %. In each dataset, we averaged all of the used throughput traces respecting the total duration of the video.
- ABR Schemes: We compared GTA against many well-known ABR schemes. In our demonstration, we show the performance of GTA compared to the conventional ABR schemes of dash.js [6] including throughput-based, buffer-based (BOLA) and hybrid.
- Video Samples and Parameters: In the DASH server, we considered different genres from the DASH dataset [12]. These genres are *animation* (Big Buck Bunny), *movie* (Valkaama), *documentary* (Of Forests And Men), and *sports* (Red Bull Playstreets). Each video is 600 seconds long and encoded with H.264/AVC into either {3, 5, 6, 10 or 20} bitrates with different resolutions {240, 360, 480, 720, 1080}p, and segment durations {1, 2, 4, 10} in seconds.

### 4.2. Single-Client Scenario

Our testing setup consisted of two Ubuntu (16.04 LTS) machines, one running the dash.js player and one for the DASH server. The player was executing in a Google Chrome browser (v60), and the server was an Apache HTTP server (v2.4). Both machines were connected via a Cisco router, and tc-NetEm (https://goo.gl/GNSbHt) was used to throttle the available bandwidth of the link between the player and the server based on the throughput profiles. We used PANDA for estimating the throughput, and we set 8 and 32 seconds as the minimum and maximum buffer occupancy thresholds, respectively.

### 4.3. Multi-Client Scenario

In the multi-client scenario, 100 players competed for 170 Mbps of total capacity minus random cross traffic of 10–70 Mbps. We grouped the players using the SSIMplus map model (*i.e.*, mapping CT, SPT, DR into a perceptual quality model) into disjoint clusters. Each cluster contained 20 players and the other parameters remained the same as in the single-client scenario. In this setup, we used five workstations, each executing 20 players, and each player running Google Chrome with Selenium and Xvfb for a headless browser.

### 4.4. Results Summary

Since similar results were obtained for various settings and operating regimes, we only present the results of CT =

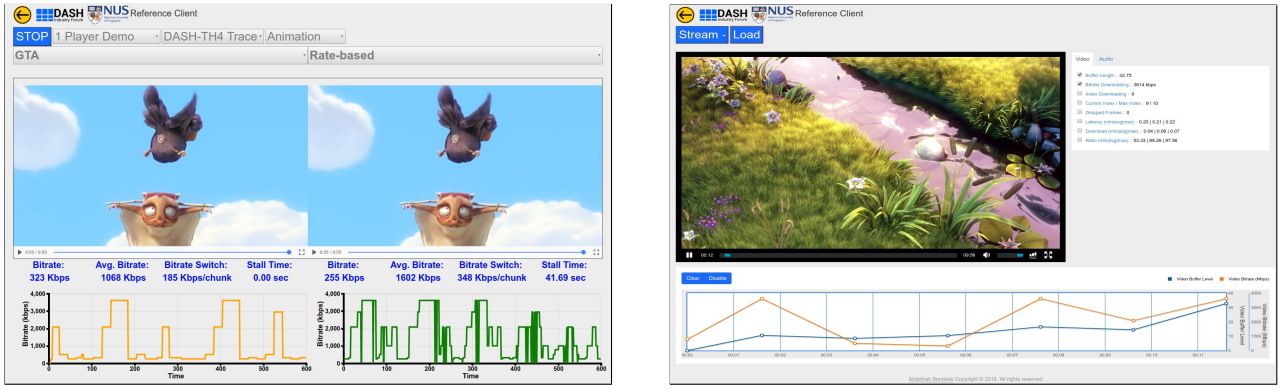


**Fig. 4**. GTA demonstration: offline demo (left figure) and online demo (right figure); available at [5].

animation, segment duration = 4 seconds, 9 levels of bitrates, and inter-variation durations {1, 4, 30, 75} seconds. Table 1 summarizes the main findings of GTA compared to the dash.js ABR rules in the scenarios of one and 100 GTA players (the results of 100 players are presented inside parentheses).

**Table 1**. Summary of the average results. Percentage improvements of GTA over the existing schemes, at scale.

| GTA Improvement vs. | Throughput-based % | Buffer-based % | Hybrid % |
|---|---|---|---|
| Quality Stability | 37.5 (15) | 72 (43) | 63.5 (41) |
| Viewer QoE | 35.5 (29) | 33.5 (23) | 19.5 (15) |
| Stalls | 6.1 (7.4) | 1.1 (4.7) | 1.7 (7.2) |
| Startup Delay | 44 (61) | 25 (32) | 18 (12) |

## 5. DEMONSTRATION

We provide an online platform [5] that comprises all the materials of GTA including the GTA demo, the design paper, experimental videos, and the source code with its related files (*i.e.*, MPDs, throughput profiles and video dataset). As shown in Fig. 4, our demonstration consists of two types: offline and online. In the offline demo, we consider two main scenarios, *i.e.*, one player and multiple players. We used this demo to simplify the emulation of the obtained results for nine bitrate levels [1] via an interactive, easy and effective demonstration. The users are given the opportunity to select the test scenario, different content types, throughput profiles and the ABR schemes to compare against each other. Moreover, for each ABR scheme, the viewer QoE metrics of the video session are reported in blue text, and the bitrate and its variability are shown in a real-time graph. On the other hand, the online demo is similar to DASH-IF's reference player. This demo emulates a realistic scenario in real time without requiring hard-coded information (the ABR decision is performed online based on the rule logic). Both demonstrations provide users with the ability to explore various operating regimes and settings. Furthermore, the GTA source code is available, which enables a user to build a customized solution.

## ACKNOWLEDGMENTS

This work was supported in part by the National Natural Science Foundation of China under Grant 61472266, in part by the NUS (Suzhou) Research Institute, and in part by grant 31T102-UPAR-1-2017 from UAE University.

## 6. REFERENCES

[1] A. Bentaleb, A. C. Begen, R. Zimmermann, and S. Harous, "Want to play DASH? a game theoretic approach for adaptive streaming over HTTP," in *ACM Multimedia Systems Conference (MMSys)*, 2018.

[2] C. Zhou, X. Zhang, L. Huo, and Z. Guo, "A control-theoretic approach to rate adaptation for dynamic HTTP streaming," in *IEEE Visual Communications and Image Processing (VCIP)*, 2012, pp. 1–6.

[3] J. Kua, G. Armitage, and P. Branch, "A survey of rate adaptation techniques for dynamic adaptive streaming over HTTP," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 3, pp. 1842–1866, 2017.

[4] Z. Han, *Game theory in wireless and communication networks: theory, models, and applications*. Cambridge University Press, 2012.

[5] GTA Demo Platform, [Online] Available: http://streaming.university/GTA/, 2018.

[6] DASH Reference Player, [Online] Available: https://goo.gl/SKZa77, 2018.

[7] Z. Duanmu, K. Zeng, K. Ma, A. Rehman, and Z. Wang, "A quality-of-experience index for streaming video," *IEEE Journal of Selected Topics in Signal Processing*, vol. 11, no. 1, pp. 154–166, 2017.

[8] Z. Li, X. Zhu, J. Gahm, R. Pan, H. Hu, A. C. Begen, and D. Oran, "Probe and adapt: Rate adaptation for HTTP video streaming at scale," *IEEE Journal on Selected Areas in Communications*, vol. 32, no. 4, pp. 719–733, 2014.

[9] Y. Sun, X. Yin, J. Jiang, V. Sekar, F. Lin, N. Wang, T. Liu, and B. Sinopoli, "CS2P: improving video bitrate selection and adaptation with data-driven throughput prediction," in *ACM SIGCOMM*, 2016.

[10] E. Thomas, M. van Deventer, T. Stockhammer, A. C. Begen, and J. Famaey, "Enhancing MPEG-DASH performance via server and network assistance," *SMPTE MIJ*, vol. 126, no. 1, pp. 22–27, 2017.

[11] A. Bentaleb, A. C. Begen, R. Zimmermann, and S. Harous, "SDNHAS: An SDN-enabled architecture to optimize QoE in HTTP adaptive streaming," *IEEE Transactions on Multimedia*, vol. 19, no. 10, pp. 2136–2151, 2017.

[12] S. Lederer, C. Müller, and C. Timmerer, "Dynamic adaptive streaming over HTTP dataset," in *ACM Multimedia Systems Conference (MMSys)*, 2012, pp. 89–94.