The Benefits of Server Hinting When DASHing or HLSing

May Lim
National University of Singapore
Singapore, Singapore

Mehmet N. Akcay
Ozyegin University
Istanbul, Turkey

Abdelhak Bentaleb
National University of Singapore
Singapore, Singapore

Ali C. Begen
Ozyegin University
Istanbul, Turkey

Roger Zimmermann
National University of Singapore
Singapore, Singapore

ABSTRACT

Streaming clients almost always compete for the available bandwidth and server capacity. Not every client's playback buffer conditions will be the same, though, nor should be the priority with which the server processes the individual requests coming from these clients. In an earlier work, we demonstrated that if clients conveyed their buffer statuses to the server using a Common Media Client Data (CMCD) query argument, the server could allocate its output capacity among all the requests more wisely, which could significantly reduce the rebufferings experienced by the clients.

In this paper, we address the same problem using the Common Media Server Data (CMSD) standard that is work-in-progress at the Consumer Technology Association (CTA). In this case, the incoming requests are scheduled based on their CMCD information. For example, the response to a request indicating a healthy buffer status is held/delayed until more urgent requests are handled. When the delayed response is eventually transmitted, the server attaches a new CMSD parameter to indicate how long the delay was. This parameter avoids misinterpretations and subsequent miscalculations by the client's rate-adaptation logic.

We implemented the server and client understanding/processing CMCD and CMSD, respectively. Our experiments show that the proposed CMSD parameter effectively eliminates unnecessary downshifting while reducing both the rebuffering rate and duration.

CCS CONCEPTS

• **Networks** → **Application layer protocols**; • **Information systems** → *Multimedia streaming*.

KEYWORDS

CMSD, CMCD, adaptive streaming, CDN, OTT, DASH, HLS, server assistance, network assistance, SAND.

ACM Reference Format:
May Lim, Mehmet N. Akcay, Abdelhak Bentaleb, Ali C. Begen, and Roger Zimmermann. 2022. The Benefits of Server Hinting When DASHing or HLSing. In *Mile-High Video Conference (MHV'22), March 1–3, 2022, Denver, CO, USA.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3510450.3517317

This work is licensed under a Creative Commons Attribution International 4.0 License.

MHV'22, March 1–3, 2022, Denver, CO, USA
© 2022 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-9222-8/22/03.
https://doi.org/10.1145/3510450.3517317

1 INTRODUCTION

In the early days of HTTP adaptive streaming, the content delivery network (CDN) providers did not take up the idea of HTTP servers or other network elements communicating with each other or streaming clients to improve the overall system's performance or the experience delivered to the individual clients. The concept of server and network assistance in streaming first flourished in 2013 and a few years later became an MPEG standard [3].

While there were many good use cases for this standard, it has not received sufficient interest from the industry for many years. Then, in 2019, the participants in the Consumer Technology Association's (CTA) WAVE project wanted to specify a solution to a long-standing problem: how could a streaming client relay media (*e.g.*, segment type/duration/format, content/session IDs) and playback (*e.g.*, current buffer length and latency) related information so that the CDN could tie the individual GET requests to playback sessions, harmonize its and client's logs to accurately generate dashboard metrics such as delivery performance, player software issues and viewer experience, and react to the time constraints implicit in media segment requests (*e.g.*, prioritize delivery for urgent requests).

As a result, CTA published the Common Media Client Data (CMCD; CTA-5004) [9] specification in Sept. 2020 for use and implementation without any royalties or additional licensing requirements. Many early implementations appeared right away. For example, the dash.js reference client [12] has been fully compliant with CMCD since v3.2.1 and there exist also open-source libraries for hls.js and ExoPlayer. The first evaluation results (*e.g.*, [8]) and capability demonstrations (*e.g.*, [15, 17]) came out soon after and many other studies are already underway. For the interested readers, the timeline of the developments in this space is detailed in [6]. An overview of CMCD and how it can potentially be used in practice is also provided in [7].

During the development of CMCD, an issue [10] was raised about the possibility of sending meta information and hints from the CDN to the streaming clients. The discussions eventually led to the concept of Common Media Server Data (CMSD), which will be developed as a companion standard to CTA-5004. As of Jan. 2022, the working group is on track to finalize the first version of this standard later in 2022 [11]. Figure 1 illustrates a media distribution system that uses CMCD and CMSD together.

CMSD can help in many ways. For example:

- A client that would mistakenly downshift due to misinterpretation of a cache miss can be warned via CMSD.
- Oscillating clients [5] can be assisted and quickly stabilized via CMSD.

- Another client that would normally start the session with the lowest-bitrate segments can be hinted to fetch higher-bitrate segments.
- Caching storage capacity is limited in practice. Thus, not all encodings can be cached on every server and rate-adapting clients can get confused in certain circumstances [14]. CMSD also helps, in this case, using caching indications to list what is cached or not cached, letting the clients make more informed decisions.
- CMSD can let the clients know the latest (live-edge) segment in low-latency live streaming.
- CMSD can assist in synchronized viewing among the clients at distinct places.
- CMSD can let the clients know about the server-driven bandwidth measurements.

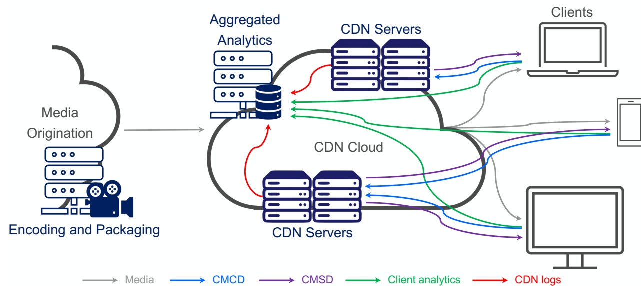

Figure 1: Client-CDN cooperation with CMCD/CMSD (reused from [6]).

We have three main contributions in this paper:

(1) This is the first attempt to investigate the feasibility of the still-work-in-progress CMSD specification and demonstrate its capability to improve viewer experience in one of the well-known problematic scenarios, *i.e.*, streaming clients competing for the available bandwidth [4]. To do so, we designed a buffer-aware response scheduling algorithm for the server. This algorithm schedules the response for each incoming segment request based on the reported (playback) buffer level. More precisely, the responses to the requests from clients with a healthy buffer level are delayed, whereas the responses to the requests from clients with a critical buffer level are delivered right away. This reduces the chances of rebuffering for the low-buffer clients while minimizing the impact on the performance of other clients. While doing this, the delayed responses include a new CMSD parameter indicating the delay duration. This information helps the clients avoid picking unnecessarily lower segment bitrates in their rate-adaptation logic.

(2) We extended our earlier proof-of-concept system [15] that conforms with the client and server-side CMCD specification to support CMSD functions. Our system comprises the open-source dash.js reference client [12] and the NGINX [1] server with a JavaScript module (NJS application) that implements the buffer-aware response scheduling algorithm. The entire system is publicly available at [16] to allow the scientific community and industry to reproduce or extend our results.

(3) We tested the implemented system in a multi-client scenario (a mix of on-demand (VoD) and low-latency live (LLL) streaming clients) via trace-driven network emulation. Our setup consisted of 10–20 concurrent streaming sessions sharing an access link with varying network conditions. The preliminary results show a reduction of 33% – 56% in rebuffering duration without degrading the video quality.

In the rest of this paper, we discuss our implemented system and its components in Section 2, followed by performance evaluation in Section 3. We conclude the paper in Section 4.

2 SYSTEM IMPLEMENTATION

The implemented proof-of-concept system is depicted in Figure 2. It consists of three main components: dash.js streaming clients, NGINX HTTP server and NetEm network emulator. The source code for the entire system is available at [16].

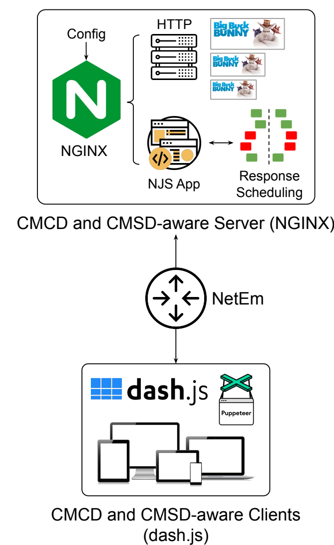

Figure 2: The implemented CMCD-CMSD system.

2.1 Streaming Clients (dash.js)

We used the CMCD implementation given by the dash.js client (v3.1.3) [12]. Specifically, the CmcdModel.js class implements various CMCD functions responsible for collecting the required values for the set of the CMCD parameters [7, 9] for other classes. For instance, it returns the buffer length (bl) and measured throughput (mtp) from BufferController.js and ThroughputHistory.js classes, respectively. Then, it generates the CMCD query string to be sent with the HTTP GET request using HTTPLoader.js. In our scenarios, we need the following CMCD parameters to be sent from the clients to the server: buffer length (bl), measured throughput (mtp), encoded bitrate (br), segment duration (d), maximum buffer threshold (com.example-bmx) and minimum buffer threshold (com.example-bmn). We note that maximum and minimum buffer thresholds were custom extensions introduced by us in [8].

To support the CMSD functions, we extended the HTTPLoader.js class to parse the CMSD parameters sent by the server in

the HTTP response headers. We made further changes to the ThroughputHistory.js class, where we subtracted the delay introduced by the server from the segment download time so that the client could compute the throughput accurately. It is worth mentioning that we used a throughput-based ABR scheme (abrThroughput.js) in this study.

2.2 HTTP Server (NGINX)

We used NGINX [1] that runs (*i*) an HTTP server that holds the source media segments of different representations for our content and the corresponding manifest files, and (*ii*) the NGINX JavaScript (NJS) middleware application. The NJS application implements the scheduler for the outgoing responses and simplifies the communication with the dash.js clients.

The NJS application has four essential functions: *CMCD request processing and parsing*, *response scheduling algorithm*, *CMSD response generation* and *decision execution.* NGINX consists of modules that are controlled by directives specified in the configuration file (nginx.conf), which enables the HTTP server and NJS application (cmsd_njs.js) to leverage the required functionalities. Below, we provide details for the NJS application (cmsd_njs.js) functions:

(1) *CMCD request processing and parsing*: This function processes each HTTP request received by NGINX with a URL path. It parses the query string to retrieve the values for the CMCD parameters and stores them in a JavaScript object (`cmcd_params`).
(2) *Response scheduling algorithm*: This function is responsible for calculating the response delay for each request based on the buffer length reported by the client. It uses the received CMCD parameter values (bl, mtp, br, d, com.example-bmx, com.example-bmn) sent by each client to make appropriate scheduling decisions. This algorithm protects the clients with low buffer levels from rebuffering. The main intuition behind it is to delay the responses proportionately to the reported buffer levels such that clients with an abundant buffer level are delayed more, while clients with a low buffer level are served sooner. This briefly frees up bandwidth that the abundant-buffer clients would have used, thereby allowing the low-buffer clients to have access to more bandwidth so that they can receive their segments sooner and avoid rebuffering. Our experiments show that we can improve the overall performance for the competing clients with proper calibration of the scheduling algorithm and without needing more network/server capacity. The other CMCD parameters are used to compute the response delay value (com.example-dl). We define three groups of clients based on their reported buffer length:
   - *Critical client*: The buffer length is less than its reported minimum buffer threshold (*i.e.*, $\text{bl} < \text{com.example-bmn}$). In this case, the response is processed right away (*i.e.*, the response delay is fixed to zero). Additionally, the response delay to be applied to the subsequent normal or abundant clients is calculated using: br × d / mtp (an approximation of the segment download time).
   - *Normal client*: The buffer length is between its minimum and maximum buffer thresholds (*i.e.*, $\text{com.example-bmn} \leq \text{bl} \leq \text{com.example-bmx}$). In this case, the response delay is calculated using:

$$\left(\frac{\text{bl} - \text{com.example-bmn}}{\text{com.example-bmx} - \text{com.example-bmn}}\right) \times \text{dl}_{current},$$

   where $\text{dl}_{current}$ is the response delay computed from the last critical client minus the time passed since that critical client was served ($\text{dl}_{current} \geq 0$).
   - *Abundant client*: The buffer length exceeds its maximum buffer threshold (*i.e.*, $\text{bl} > \text{com.example-bmx}$). In this case, the response delay is fixed to $\text{dl}_{current}$.
(3) *CMSD response generation*: This function generates the HTTP response header (`CMSD-Dynamic`) to convey the delay applied (with the newly defined CMSD parameter: com.example-dl) to the corresponding client. The client parses the response header to extract the delay, which is then subtracted from the segment download time when calculating the throughput. The client then performs rate adaptation as usual.
(4) *Decision execution*: This function applies the delay decisions to the corresponding client requests. As configured in `nginx.conf`, it uses the `echo_sleep` directive of the HTTP Echo Module for NGINX [2] to apply the per-request response delay.

2.3 Network Emulation (NetEm)

For network emulation, we used the *Cascade* bandwidth profile that is publicly available in our repository [16]. The bandwidth throttling between the clients and server is performed using the NetEm tool [13] through the tc command.

3 PERFORMANCE EVALUATION

3.1 Scenarios and Setup

We implemented a proof-of-concept CMCD-CMSD system that incorporates our proposed scheduling algorithm. We also designed test cases to evaluate our system in a multi-client environment where multiple clients concurrently streamed over a shared network. The two test cases contained: (*i*) 10 VoD streaming clients, and (*ii*) 10 VoD streaming and 10 LLL streaming clients, respectively. The concurrent streaming sessions shared an access link with varying network conditions enabled via trace-driven network emulation.

Our test machine used Ubuntu 18.04.6 LTS, AMD Ryzen 7 3700X 8-Core CPU and 32 GB memory. The dash.js clients ran in Google Chrome browser (v96.0.4664.110) with headless mode, which was enabled by Puppeteer (v5.5.0) and Node.js (v15.14.0), and used the default throughput-based ABR scheme (abrThroughput.js). The network emulation was performed using tc NetEm, which varied the bandwidth as specified in the *Cascade* bandwidth profiles. Table 1 summarizes the bandwidth values used in our tests (CascadeX10 was used for the first test case with 10 clients and CascadeX20 was used for the second test case with 20 clients). Each profile changed the throttled bandwidth every 30 seconds and looped for the video duration of 10 minutes (which equates to the duration of one test).

Profile Name	Values (Mbps)	# of Clients
CascadeX10	100, 40, 20, 10, 20, 40	10
CascadeX20	200, 80, 40, 20, 40, 80	20

Table 1: The bandwidth profiles applied in both test cases. Bandwidth values were updated at 30-second intervals and the profiles looped for the test duration of 10 minutes.

We used two dash.js-compliant video datasets that were encoded, per Akamai's encoding recommendations, with H.264 codec at 30 fps and a bitrate ladder of five representations: 180p@0.4 Mbps, 360p@0.8 Mbps, 432p@1.5 Mbps, 576p@2.5 Mbps, 720p@4.0 Mbps. Both datasets used the *Big Buck Bunny* video with a duration of 10 minutes, however, they varied by segment duration: the first dataset used a segment duration of four seconds to simulate VoD streaming sessions while the second dataset used one second to simulate LLL streaming sessions. The videos were then stored on our NGINX HTTP server. The clients and NJS application used minimum and maximum buffer thresholds of [B_{min} = 4 s | B_{max} = 20 s] for VoD and [B_{min} = 2 s | B_{max} = 6 s] for LLL streaming sessions.

3.2 Results and Analysis

The primary goal in our evaluation is to show how we can potentially use a scheduling algorithm with CMCD and CMSD to reduce rebuffering without degrading video quality when multiple clients compete for the available bandwidth. We ran each test case five times and averaged the results. We compared using CMSD with scheduling against using no CMSD or scheduling using the following metrics:

- Avg. BR: Average video bitrate across all clients (Mbps).
- Min. BR: Average video bitrate for the client with the lowest average bitrate (Mbps).
- Avg. RD: Average total rebuffering duration across all clients (s).
- Max. RD: Total rebuffering duration for the client with the longest rebuffering duration (s).
- Avg. RC: Average rebuffering count across all clients.

From the results shown in Table 2, we see that enabling CMSD with the scheduling algorithm reduces Avg. RD by 33% and 56%, and Avg. RC by 30% and 27% for test cases (*i*) and (*ii*), respectively. The Avg. BR remains more or less unchanged with an average reduction of only 2%. CMSD helps preserve the average bitrate (quality) even as we introduce a delay on the server side, since the delay is made known to the client via CMSD and accounted for in the client-side mtp calculations. We believe that this is a promising start for investigations on practical CMSD usage and we hope to continue exploring this and other use cases.

4 CONCLUSIONS

Both CMCD and CMSD specifications have been developed to promote better cooperation between adaptive streaming clients and CDN servers to improve streaming performance. Beyond defining the specification, designing and validating the use cases are equally important to ensure adoption. In this work, we conducted a preliminary study to design and evaluate a proof-of-concept system that applied CMCD, CMSD and a buffer-aware response scheduling algorithm to manage a multi-client shared network scenario. The results show that the system reduces the average rebuffering duration and average rebuffering count without any noticeable degradation in the video quality.

REFERENCES

[1] High Performance Load Balancer Web Server. [Online] Available: https://www.nginx.com/. Accessed on Jan. 10, 2022.
[2] HTTP Echo Module. [Online] Available: https://www.nginx.com/resources/wiki/modules/echo/. Accessed on Jan. 10, 2022.
[3] ISO/IEC 23009-5:2017 Information technology — Dynamic adaptive streaming over HTTP (DASH) — Part 5: Server and network assisted DASH (SAND). [Online] Available: https://www.iso.org/standard/69079.html. Accessed on Jan. 10, 2022.
[4] S. Akhshabi, L. Anantakrishnan, A. C. Begen, and C. Dovrolis. What happens when HTTP adaptive streaming players compete for bandwidth? In *ACM NOSSDAV*, 2012 (DOI: 10.1145/2229087.2229092).
[5] S. Akhshabi, L. Anantakrishnan, C. Dovrolis, and A. C. Begen. Server-based traffic shaping for stabilizing oscillating adaptive streaming players. In *ACM NOSSDAV*, 2013 (DOI: 10.1145/2460782.2460786).
[6] A. C. Begen. Manus manum lavat: media clients and servers cooperating with common media client/server data. In *ACM Applied Networking Research Wksp. (ANRW)*, 2021 (DOI: 10.1145/3472305.3472886).
[7] A. C. Begen, A. Bentaleb, D. Silhavy, S. Pham, R. Zimmermann, and W. Law. Road to salvation: streaming clients and content delivery networks working together. *IEEE Communications Magazine*, 59(11):123–128, 2021 (DOI: 10.1109/MCOM.121.2100137).
[8] A. Bentaleb, M. Lim, M. N. Akcay, A. C. Begen, and R. Zimmermann. Common media client data (CMCD): Initial findings. In *ACM NOSSDAV*, 2021 (DOI: 10.1145/3458306.3461444).
[9] Consumer Technology Association. CTA-5004: Web Application Video Ecosystem–Common Media Client Data, Sept. 2020.
[10] cta-wave/common-media-client data. No common-media-server-data? [Online] Available: https://github.com/cta-wave/common-media-client-data/issues/19. Accessed on Jan. 10, 2022.
[11] cta-wave/common-media-server data. [Online] Available: https://github.com/cta-wave/common-media-server-data. Accessed on Jan. 10, 2022.
[12] DASH-IF. DASH Reference Client. [Online] Available: https://reference.dashif.org/dash.js/. Accessed on Jan. 10, 2022.
[13] S. Hemminger et al. Network emulation with netem. In *Linux conf au*, volume 5, page 2005. Citeseer, 2005.
[14] D. H. Lee, C. Dovrolis, and A. C. Begen. Caching in HTTP adaptive streaming: friend or foe? In *ACM NOSSDAV*, 2014 (DOI: 10.1145/2597176.2578270).
[15] NUS-OzU. CMCD-DASH. [Online] Available: https://github.com/NUStreaming/CMCD-DASH. Accessed on Jan. 10, 2022.
[16] NUS-OzU. CMSD-DASH. [Online] Available: https://github.com/NUStreaming/CMSD-DASH. Accessed on Jan. 10, 2022.
[17] S. Pham, M. Avelino, D. Silhavy, T.-S. An, and S. Arbanowski. Standards-based streaming analytics and its visualization. In *ACM MMSys*, 2021.

	CascadeX10		CascadeX20	
Metric	**CMSD**	**NO CMSD**	**CMSD**	**NO CMSD**
Avg. BR	3.46	3.55	3.20	3.26
Min. BR	3.15	3.27	2.45	2.59
Avg. RD	3.52	5.26	0.51	1.16
Max. RD	15.0	14.5	4.15	8.21
Avg. RC	1.52	2.18	0.40	0.55

Table 2: Results for the test cases (*i*) 10 VoD streaming clients (CascadeX10), and (*ii*) 10 VoD and 10 LLL streaming clients (CascadeX20).