Streaming Analytics using CMCD and CMSD

Daniel Yermakov
TU Berlin
Berlin, Germany
daniel.yermakov@campus.tu-berlin.de

Maximilian Roschlau
TU Berlin
Berlin, Germany
m.roschlau@campus.tu-berlin.de

Neha Shrestha
TU Berlin
Berlin, Germany
neha.shrestha@campus.tu-berlin.de

***Abstract*—The Common Media Client Data (CMCD) specification defines data that is collected by a media client and is sent a Content Delivery Network (CDN) along with its HTTP requests. A discussion was raised during the development of the CMCD specification about the possibility of sending meta information and hints from the CDN to the streaming clients, which led to the concept of Common Media Server Data (CMSD). CMSD defines a structure for data transmitted in the response to a request from a media player for an HTTP Adaptive Streaming (HAS) media object. Existing projects in both CMCD and CMSD specification have shown that the use of client data and server data helps to improve the quality of streaming services and the buffering rate. In this paper, we have attempted to extend one of the existing projects from CMSD and implement and evaluate two of the use cases listed in the CMSD - Working Draft. While the implementation has been successful so far, we have reached our limits in performing meaningful tests due to hardware limitations.**

***Index Terms*—CMCD, CMSD, DASH**

I. Introduction

Streaming media occupies a large part of the data volume on the internet traffic today. As the demand for premium streaming services is rapidly growing, content providers are competing to provide high video quality and low buffering times to meet the viewer's increasing expectations. One technology used by content providers to meet viewer expectations is the Content Delivery Network (CDN). It tries to improve the quality of content delivered by streaming services by reducing the distance between the data and the users but is limited in its ability to manage continuously increasing streaming traffic. Specific information is required to address those limitations. One approach to improve the quality of streaming services is Server and Network Assisted Dash (SAND). SAND provides enhanced content delivery by improving the adaptation accuracy and reaction time [11]. While the SAND standard provided a framework for communication and information sharing, it left open the question of what information was relevant and usable [4]. Two specifications attempt to answer this question, namely the Common Media Client Data (CMCD) specification and the Common Media Server Data (CMSD) specification.

CMCD is an open specification created by the Consumer Technology Association (CTA) project which deals with different aspects of internet-delivered media. It defines data that is collected by a media client and is sent to the CDN along with its HTTP requests [4]. Furthermore, it allows content providers to get insights into the performance of their large-scale streaming operations and help the CDN provider increase the overall performance of the CDN itself. As mentioned in [10], values defined in the CMCD specification can be beneficial for log analysis, quality-of-experience (QoE) monitoring, and delivery optimization.

As mentioned in [6], the development of CMCD gave rise to the discussion about the possibility of sending meta information and hints from the CDN to the streaming clients. This discussion gave rise to the concept of CMSD, which is still work-in-progress and will be developed as a companion standard to CTA-5004. It defines the structure for data transmitted in the response to a request from a media player for an HTTP Adaptive Streaming (HAS) media object [14]. This response generally originates at an origin server and is distributed through a series of intermediaries to the player. As mentioned in the working draft of CMSD [14], the purpose of the CMSD specification is to define a standard by which every media server (intermediate and origin) can communicate data with each media object response and have it received and processed consistently by every intermediary and player, to improve the efficiency and performance of distribution and ultimately the quality of experience enjoyed by the end users.

This paper aims to experiment with CMSD implementations and their use cases. We studied existing projects in CMCD and CMSD specifications to analyse how the implementations worked and what the projects are already capable of. We then extended one of the existing CMSD specification projects for experimental purposes. In the remainder of this paper, we highlight related work in Section 2 and analyse existing projects in CMCD and CMSD in Section 3. In Section 4, we present our CMSD extension, which we then evaluate in Section 5. We conclude this paper with an outlook on future developments in Section 6.

II. Related Work

The use of client-side or server-side data to improve streaming quality has been studied frequently in recent years. [4] investigated the feasibility of CMCD in addressing the common problem in the streaming domain, i.e. efficient use of shared bandwidth by multiple clients. The same problem was addressed in [6] using CMSD standard. We discuss both findings of the paper in brief in section 3. In [5], a client-side request scheduler was presented that distributes requests for the video over multiple heterogeneous interfaces simultaneously. [3] provided an overview of CMCD standard and described possible application scenarios. It allows reader

to further explore in their own practical environments through an open source sample implementation. [3] investigated using client-side or server-side information in HTTP adaptive streaming technique. A number of new algorithms has been proposed and evaluated for both non-immersive and immersive media in different settings, ranging from low-latency live (LLL) to on-demand (VoD) streaming [7]. An approach is made for further investigation in CMSD specification to allow CDNs to send useful information back to the clients.

III. Existing Implementations

In this section, we provide an overview of the existing projects on CMCD and CMSD. We analysed one project on CMCD and two project on CMSD. This overview and analysis led us to select and extend one of the implementation for experimenting with CMSD.

A. CMCD - NUStreaming

As claimed in [8], this project is the first study to investigate the feasibility of CMCD and test its capabilities in the context of video delivery in the well-known problematic scenario where streaming clients compete for the available bandwidth. It consists of CMCD-aware clients built upon the initial CMCD implementation offered by the dash.js client. It further consists of a CMCD-aware server that uses NGINX with the NGINX Javascript (NJS) module to run the HTTP server and an NJS middleware application. The NJS application function includes three main functions: request processing and parsing, bandwidth allocation logic, and decision execution.

In this project, the authors designed a buffer aware bandwidth allocation algorithm to find the relation between the current buffer level and the bandwidth that should be allocated to a client. The CMCD parameters: buffer length (bl), max buffer (com. example-bmx), min buffer (com. example-bmn), buffer starvation (bs), and object type (ot) are used for this purpose. The main objective of this algorithm is to reduce the impact of the rebuffering events, which is known to be an important factor in maintaining a good viewer experience. This project evaluated the CMCD-aware system with the two bandwidth profiles Cascade and Spike in the following scenarios: i) an access link with 5-10 and ii) an aggregation link with 20-30 concurrent streaming sessions. In the first scenario with on-demand video session, their experiment showed that enabling CMCD with buffer-aware bandwidth allocation significantly reduces the average total rebuffering duration across all clients by 74% and 83%, and the total rebuffering duration for the client that suffered from the longest rebuffering duration by 72% and 78% and average rebuffering count across all clients by 57% and 66% for Cascade and Spike respectively. There was also a reduction in the average bit rate across all clients, but this was far less significant.

The second scenario aimed to show the benefits of using CMCD when a variable number of concurrent clients stream an on-demand video and compete at an aggregation link for the available bandwidth. The experiment showed that the benefits of buffer-aware bandwidth allocation start diminishing when the number of clients increases. This project further investigated the first scenario with low-latency live video sessions to determine whether using CMCD reduces rebuffering duration and helps the latency stay below the target value set by the application. The experiment showed that upon enabling CMCD, the clients achieved a reduction in average total rebuffering of 20% and 26%, in total rebuffering duration for the client that suffered from the longest rebuffering duration (s) of 25% and 22%, and in average rebuffering count across all clients of 36% and 21% for Cascade and Spike, respectively.

B. CMSD - NUStreaming

This project [9] attempted to investigate the feasibility of the still-work-in-progress CMSD specification and demonstrate its capability to improve the viewer experience. For this purpose, the authors designed a buffer-aware response scheduling algorithm for the server, where incoming requests are schedule based on their CMCD information. Client requests are scheduled with respect to the buffer level. Response to the clients with critical buffer level are responded first than the one with a healthy buffer level. This reduces the chances of rebuffering for the low-buffer clients while minimizing the impact on the performance of other clients. The delayed responses are included in a new CMSD parameter indicating the delay duration. Such information helps the clients avoid picking unnecessarily lower segment bitrates in their rate-adaptation logic.

Furthermore, they extended the proof-of-concept of CMCD-DASH that conforms with the client and server-side CMCD specification to support CMSD functions. They tested the implemented system in a multi-client scenario via trace-driven network emulation. The system components in this project were similar to that of the CMCD project discussed in section III-A. This project used the CMCD implementation given by the dash.js client. The CMCD parameter: 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) were required to be sent from the clients to the server for the experiment. To support the CMSD functions, they extended the HTTPLoader.js class in the dash.js to parse the CMSD parameters sent by the server in the HTTP response headers. Furthermore, they modified the ThroughputHistory.js class so that the client could compute the throughput accurately. This experiment proved that the proposed CMSD parameter eliminate unnecessary downshifting while reducing both the rebuffering rate and duration.

Additionally, they implemented a proof-of-concept CMCD-CMSD system to incorporate with the proposed scheduling algorithm. They also designed test cases to evaluate their system in a multi-client environment where multiple clients concurrently streamed over a shared network. The two test cases are: (i) 10 VoD streaming clients, and (ii) 10 VoD streaming and 10 LLL streaming clients, respectively. Their experiment showed that enabling CMSD with the scheduling algorithm reduces average total rebuffering duration across all

clients(s) by 33% and 56%, and average rebuffering count across all clients by 30% and 27% for test cases (i) and (ii), respectively. The average video bitrate across all clients (Mbps) showed an average reduction of only 2%. Their experiment proved that the proposed CMSD parameter eliminate unnecessary downshifting while reducing both the rebuffering rate and duration.

C. Origin CMSD

Origin CMSD is another publicly available experimental project developed by the video delivery technology company, Unified Streaming [12]. This implementation aims to test and experiment with CMSD. The system components in this project consist of three docker containers in which one origin server is implemented using Apache and two intermediate servers are implemented using NGINX and Varnish Cache. The request on the client side goes through two intermediate servers, is then directed to the origin server where the response and CMSD headers and associated key-value pairs are generated. Currently, only the server-side components are implemented in this project. The client therefore requires to be simulated by the user. The client is either simulated via curl commands which respond with corresponding CMSD static and dynamic headers or with the DASH Industry Forum (DASH-IF) Reference Client. With this client, it is possible to load the local host media files and display metrics such as buffer length or latency. One way to observe the CMSD headers and exchanged key-value pairs is with in-build browser tools like the Google Chrome developer tools. It is also worth mentioning that the client can enable CMCD and send corresponding parameters with its request. However, the current implementation in this project cannot process CMCD requests yet. As the client-side components are still missing in this implementation, no evaluation can be made on CMCD and CMSD interaction and how this can lead to the performance improvements. Furthermore, 9 out of 23 defined key-value pair are currently working and not all key-value pairs listed in the CMSD working draft are implemented.

D. Comparison of the project

After studying the existing implementations in CMCD project and CMSD project, we tried to make a comparative evaluation between them. We observed that the CMSD project from NUStreaming used 6 CMCD parameters out of 18 parameters defined in [13]. On the other hand, Origin CMSD from Unified media does not use any CMCD metrics. Another difference in these project was the use of CMSD-specific headers. NUStreaming - CMSD project used the dynamic header buffer-based delay. This parameter is the authors' own extension and is not specified in the CMSD working draft. On the other hand, Unified Media used both static and dynamic headers, which makes 9 parameters out of 23 parameters defined in the CMSD working draft. In addition, the experiment set up in NUStreaming is different compared to Unified Media which illustrates the improvement in performance with CMSD. In their research, the authors of NUStreaming were able to show a 56% reduction in average rebuffering time. As a result, we can conclude that the Unified Media solution is more illustrative in terms of what data is exchangeable and the one from NUStreaming can be a better example of the functional specificity interaction. For this reason and because the project structure is better suited for adjustments, we decided to extend the implementation from NUStreaming - CMSD for our project. An overview of the comparison between the project is shown in Table I.

TABLE I
COMPARISON BETWEEN NUSTREAMING AND UNIFIED MEDIA

Criteria	***NUStreaming - CMSD***	**Unified Media**
CMCD-Keys	6/18	Not implemented
CMSD-Keys	1/23	9/23
CMSD Header-Types	Dynamic	Dynamic, Static
Features	Experiment	

IV. IMPLEMENTATION

Among the use cases listed in the CMSD working draft [14], we selected two for our project and extended the implementation from the NUStreaming CMSD project.

The first use case we selected involves switching between multiple servers for load balancing purposes. The general idea behind this use case was to send the *duress (du)* key from the server when it is overloaded, so that the client is forced to change the origin to an alternative server. According to the CMSD working draft, this key is specified without value when the server is under pressure due to CPU, memory, disk, network IO, or other reasons. [1]

The second use case involves using the *max suggested bitrate (mb)* key, which the server can apply to limit the maximum bitrate on the client. It helps to improve the user experience and reduce the overall bandwidth, if requested by the internet service provider. [2]

A. Streaming server (NGINX)

As part of the server adaptation, we changed the *nginx.conf* file in such a way that it supports working with two servers and changed the main server logic file - *cmsd_njs.js* and *cmsd_njs2.js* for each of the server instances. Furthermore, we have added multiple endpoints to the servers to ensure appropriate information exchange.

To implement the first scenario, we created a server overload value that tells us when the server should transition to the overload scenario. The server load was simulated by specifying a single client load and specifying an extraneous server load. Since the thresholds for signalling duress are left to the discretion of the server operator [14], we have set the threshold at 60%. This value can be adjusted according to the desired scenario. In case the server is overloaded, it tries to reduce the number of clients by sending the du flag until the number of

[1] Similar to Appendix B: Use Case 3, but with server-side decision

[2] Appendix B: Use cases 4 and 10

clients runs out, or the load becomes acceptable (scenario on the Figure 2). Also, when a new client joins, the server checks if it could cause an overload, and if so, it immediately signals the client that it needs to join another server (scenario on the Figure 1).

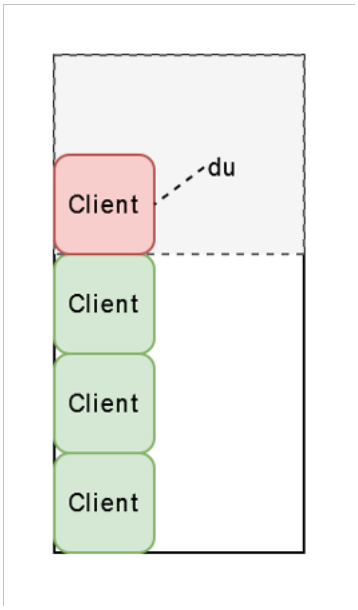

Fig. 1. New client provokes overload

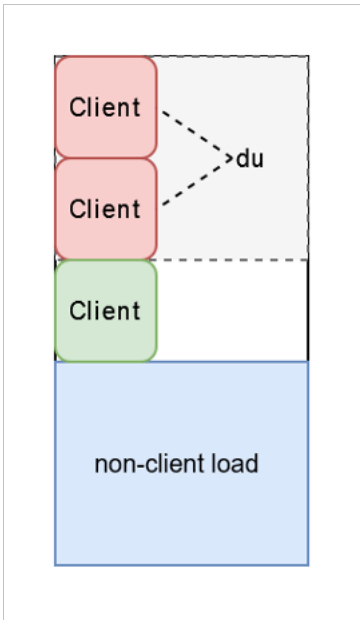

Fig. 2. Overload provoked by another reasons

To implement the second scenario, we created a simple decision-making schema, that allows the server to determine what the maximum bitrate for clients should be set to. In our case, we proceeded from the principles of fair sharing, and divided the bandwidth between clients in equal parts. Considering that in our case, both the server and clients work on the same machine, the bandwidth size is simulated just by setting the responsible variable to the desired value.

B. Streaming clients (dash.js)

On the client side, in response to a server request to switch, we change the connection to another predefined server URL (in our case, different localhost ports). In addition, the client remembers which server was unavailable, so as not to return to it later.

The response to a change of the maximum bitrate is performed by the built-in methods of the player, by adapting the responsible variable in its settings.

All logic adaptations for clients were made within the *index.html* file.

C. Command Line Interface

To simplify project management, we also implemented a command line interface (CLI) that allows convenient interaction with the server, such as starting or stopping it, as well as interacting with the implemented endpoint and therefore changing its internal parameters such as server load. Furthermore, the CLI contains a special monitor that allows to display real-time parameters such as the maximum bit rate, active server sessions or load. Additionally, it provides the possibility to display NGINX logs, start the dash.js client and run the experiment from [6] using the *batch_test.sh* script.

V. Evaluation

The features mentioned in the previous section have been fully implemented and tested by us. Clients freely switch between servers if one of them is indicated as overloaded. If a client switches to another server during streaming, the loading of the video continues where it left off before. Since the transition is currently still marked by a minimal interruption, there is still room for improvement here. The bitrate of the clients is adapted to the given properties of the bandwidth in accordance with the principle of fair sharing. Key values such as bandwidth size, overload threshold and single client load can be dynamically adapted across different setups.

A. Benchmarking

To test our solution in a multi-client situation, we wanted to use the provided NUStreaming test using the *batch_test.sh* script. In their experiment, the NUStreaming team used a physical machine running Ubuntu 18.04.5 LTS with dual 20-core Intel E5-2630 v4 @ 2.20GHz processors and 192 GB memory [4]. Unfortunately, we were limited here due to our equipment (the project was run on a Unix virtual machine). Even running the test with more than seven clients caused the virtual machine to crash. Implementing our own test setup with multiple clients running simultaneously was therefore not possible due to our hardware limitations. However, if the hardware requirements for a particular scenario are available, values such as server threshold and bitrate adaption can be adjusted accordingly and our adaptation of the NUStreaming

project can be used to perform the experiment without any problems.

Browser testing has shown that more than 30 player instances (in browser tabs) can be connected to a single server instance at the same time with 4 GB of RAM. However, one of the key factors was that both server instances, and all clients, shared the same machine. In addition, all players, likely, did not download videos simultaneously, due to the setting of the browser. Thus, such a test is not representative enough to evaluate our solution, but allows us to have a basic idea of the very minimal capabilities of our system when it is deployed in a comparable setup.

B. Discussion

Considering that we have made two key changes to the system, we can give the following assessment regarding their effectiveness.

Switching between servers - we believe that increasing the number of server instances will have a significant positive effect on performance and, as a result, on the user experience, given that this is the main principle of horizontal scaling.

Limiting the maximum bitrate - according to the draft, this feature also has a large positive impact on the user experience. However, in our case, it is of an instrumental nature, with rather primitive decision logic behind it. Given this, we can assume that this part of the implementation in its current form will not affect the efficiency of the system and the quality of the experience.

In conclusion, we can say that both implemented features were specified in the draft and, from a theoretical point of view, have a significant positive impact on the system. Our current implementation is however, in this case, more of an experimental prototype than a competitive one, and has not been optimized for real-life practical scenarios.

VI. Conclusion

Both CMCD and CMSD specification has recently emerged as an active cooperation paradigm that allows adaptive streaming clients and CDN servers to improve streaming performance. This information is beneficial to the CDN providers in understanding the root causes of QoE degradation, troubleshooting and improving the entire media distribution pipeline. As part of our project, we examined existing projects in the CMCD and CMSD specifications to analyse how the implementations work and what the projects are already capable of. We then extended the NUStreaming CMSD project for experimental purposes. Therefore, we selected two use cases that were listed in the CMSD working draft for our project and modified the NUStreaming project accordingly. The first use case we selected involves switching between multiple servers for load balancing purposes. The second use case concerns the use of the *max suggested bitrate (mb)* key, which the server can apply to limit the maximum bitrate on the client. Although performing tests with multiple clients running at the same time and corresponding measurements was not possible on our side due to hardware limitations, our extension of the NUStreaming project is able to perform such tests if the required hardware is available.

References

[1] Akcay, M. (2021). Improving Server and Client-Side Algorithms for Adaptive Streaming of Non-Immersive and Immersive Media. Proceedings Of The 12Th ACM Multimedia Systems Conference. https://doi.org/10.1145/3458305.3478461

[2] Begen, A. (2021). Manus manum lavat. Proceedings Of The Applied Networking Research Workshop. https://doi.org/10.1145/3472305.3472886

[3] Begen, A., Bentaleb, A., Silhavy, D., Pham, S., Zimmermann, R., & Law, W. (2021). Road to Salvation: Streaming Clients and Content Delivery Networks Working Together. IEEE Communications Magazine, 59(11), 123-128. https://doi.org/10.1109/mcom.121.2100137

[4] Bentaleb, A., Lim, M., Akcay, M., Begen, A., & Zimmermann, R. (2021). Common media client data (CMCD). Proceedings Of The 31St ACM Workshop On Network And Operating Systems Support For Digital Audio And Video. https://doi.org/10.1145/3458306.3461444

[5] Evensen, K., Kaspar, D., Griwodz, C., Halvorsen, P., Hansen, A., & Engelstad, P. (2011). Improving the performance of quality-adaptive video streaming over multiple heterogeneous access networks. Proceedings Of The Second Annual ACM Conference On Multimedia Systems. https://doi.org/10.1145/1943552.1943560

[6] Lim, M., Akcay, M., Bentaleb, A., Begen, A., & Zimmermann, R. (2022). The benefits of server hinting when DASHing or HLSing. Proceedings Of The 1St Mile-High Video Conference. https://doi.org/10.1145/3510450.3517317

[7] Mehmet N. Akcay. 2021. Improving Server and Client-Side Algorithms for Adaptive Streaming of Non-Immersive and Immersive Media. In Proceedings of the 12th ACM Multimedia Systems Conference (MMSys '21). Association for Computing Machinery, New York, NY, USA, 383–387. https://doi.org/10.1145/3458305.3478461

[8] NUS-OzU. CMCD-aware System. [Online] Available: https://github.com/NUStreaming/CMCD-DASH. Accessed on Jul. 20, 2022.

[9] NUS-OzU. CMSD-DASH. [Online] Available: https://github.com/NUStreaming/CMSD-DASH. Accessed on Jul. 30, 2022.

[10] Silhavy, D. (2022). dash.js - Common-Media-Client-Data - Video-Dev. Video-Dev. Retrieved 31 July 2022, from https://websites.fraunhofer.de/video-dev/dash-js-common-media-client-data-cmcd/.

[11] Uitto, M., & Heikkinen, A. (2016). SAND-assisted encoding control for energy-aware MPEG-DASH live streaming. 2016 24Th International Conference On Software, Telecommunications And Computer Networks (Softcom). https://doi.org/10.1109/softcom.2016.7772179

[12] Unified Streaming. Origin CMSD. [Online] Available: https://github.com/unifiedstreaming/origin-cmsd. Accessed on Jul. 20, 2022.

[13] Vijayanagar, K. (2022). Common-Media-Client-Data (CMCD) Explanation with Sample Player - OTTVerse. OTTVerse. Retrieved 31 July 2022, from https://ottverse.com/common-media-client-data-cmcd/.

[14] Begen, A., Evans, J., Law, W. CMSD- Working draft [Online] Available: https://docs.google.com/document/d/1BlTHfbF2VGSlA4vLx1fMssWqiGWYuzhmTfQ8VeyxF8g/ Accessed on Jul. 31, 2022.