IBC2023

IMPLEMENTING HLS / DASH CONTENT STEERING AT SCALE

Yuriy Reznik, Guillem Cabrera*, Ron Zekarias, Bo Zhang, Biswa Panigrahi, Nabajeet Barman*, Stuart Hicks*, Ted Krofssik, Andrew Sinclair, and Adam Waldron

Brightcove Inc, Boston, MA, USA
* Brightcove UK, Ltd., London, UK

ABSTRACT

Content Steering is a recent addition to both HLS and MPEG DASH standards, enabling dynamic routing of streaming content between different CDNs and delivery pathways. Already supported by DASH.js, HLS.js, and several other players, it dramatically simplifies the design of multi-CDN systems. No custom client plugins, DNS redirects, or CMS integrations are needed. However, for HLS/DASH Content Steering method to operate, it requires new elements: content steering servers. This paper discusses how to design and deploy them at scale. The proposed solution effectively reduces the logic on the content steering servers to stateless operations, in which all state variables become parameters of the client-server exchange. Such design enables servers to be deployed very inexpensively at the edge (by utilizing edge functions offered by many CDNs or edge platforms). Our proposed method is highly scalable, allows short response time, and enables a full spectrum of multi-CDN traffic optimizations: load balancing, failover protection, COGS- and QOE/QOS-based optimizations.

1. INTRODUCTION

Since its invention in the mid-1990s, Internet streaming has evolved from a pioneering concept to a mainstream technology used to deliver videos to viewers today [1-6]. This technology is exceptionally versatile. It reaches all IP-connected video devices of different screen sizes, mobility factors, and connection types (TVs, mobiles, PCs, etc.).

The two most widely deployed variants of streaming protocols today are called HTTP Live Streaming (HLS) [7] and Dynamic Adaptive Streaming over HTTP (DASH) [8]. Both are international standards. Both use HTTP as the underlying network protocol and employ Content Delivery Networks (CDNs) for media delivery [9,10]. The underlying principle is simple: the encoded media content is placed on the origin server first, and then CDN propagates, locally caches, and delivers it to a geographically dispersed population of viewers. Effectively the CDN manages the scale of the delivery.

However, CDNs have some limits. Some may not be available in all relevant regions; some may have internal capacity limits, and some may not have sufficient caches to support the delivery of vast collections of videos to the intended audience. CDNs may sometimes also experience outages or other technical failures, making them inaccessible for some time.

Considering such limits, large streaming operators increasingly employ multiple CDNs and so-called "CDN switching" technologies to adjust delivery paths content dynamically for streaming [10-12]. Among existing solutions are technologies using DNS-based switching, dynamic manifest updates, player-based switching, custom CMS integrations, etc. However,

most existing CDN switching solutions are complex and expensive to deploy and operate [9-13]. They also come with various drawbacks. Very few, for example, enable seamless in-stream switching without interrupting the continuity of the playback. And they are all proprietary, requiring much custom code to be written and supported for each deployment.

HLS / DASH Content Steering is a new standards-based technology [14-16] that promises to dramatically simplify the design of multi-CDN streaming systems.

In this paper, we first briefly overview the Content Steering technology, explain how it works, and explain its benefits for practical applications. We will then discuss a few challenges arising in this technology's design and deployment at scale. We will show that such challenges are all addressable by design turning HLS/DASH content steering servers into stateless functions deployable by advanced CDNs or edge platforms. Such a method is highly scalable, allows short response time, and enables a full spectrum of multi-CDN traffic optimizations: load balancing, failover protection, COGS- and QOE/QOS-based optimizations. The proposed design forms the basis of an open-source content steering system implementation, currently under development and validation study by the Streaming Video Technology Alliance (SVTA) [13,17]. We will explain the capabilities of this system, the set of streaming clients and CDN integrations it already supports, and the utilities it brings to the operators of streaming systems.

HLS / DASH CONTENT STEERING

HLS / DASH Content Steering is a relatively recent development. First, in April 2021, Apple proposed a technology called "HLS Content Steering Specification" [14]. Subsequently, in July 2022, a similar technology proposal, titled "Content Steering for DASH," was entered for community review by DASH-IF [16]. The DASH-IF proposal was effectively a subset of HLS content steering, preserving the syntax of the client-server exchanges. The corresponding changes in HLS and DASH standards have been implemented over the last two years [8,15]. As of today, Content Steering is already supported by the AVplayer framework [18], as well as HLS.js [19] and DASH.js [20] streaming players. Reference streams and related open-source tools are also available for the developer community through the efforts of DASH-IF, CTA WAVE, and SVTA forums [13,17].

To illustrate how the Content Steering mechanism works, in Figure 1, we depict an example streaming delivery system practicing it. This system employs two media CDNs, denoted

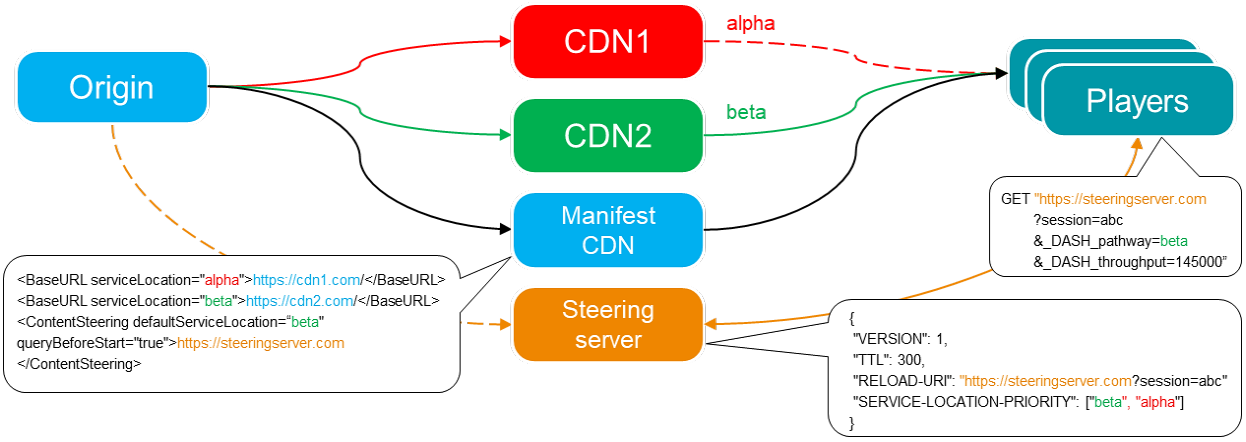

Figure 1 – DASH delivery system with two media CDNs and content steering servers managing switching between them.

CDN1 and CDN2, respectively. The URLs (or base URLs) of such CDNs, also called "pathways," have assigned names. In our system, we use names "alpha" and "beta" to refer to CDN 1 and 2, respectively. Both CDNs can deliver data, but only one is active at each moment in time. The system also deploys a server-side control element - the Content Steering server. We show relevant manifest declarations and exchanges between players and the steering servers in callouts.

As shown in Figure 1, the manifest defines the locations of CDNs and a steering server for use during a streaming session. In DASH, the corresponding syntax includes redundant BaseURL declarations and a ContentSteering descriptor:

```
<BaseURL serviceLocation="alpha">https://cdn1.com/</BaseURL>
<BaseURL serviceLocation="beta">https://cdn2.com/</BaseURL>
<ContentSteering defaultServiceLocation="beta"
queryBeforeStart="true">https://steeringserver.com>
</ContentSteering>
```

In HLS, the corresponding syntax includes using redundant variant streams pointing to different CDNs, with PATHWAY-ID annotations and a pointer to the steering server provided by the `#EXT-X-CONTENT-STEERING tag:`

```
#EXTM3U
#EXT-X-CONTENT-STEERING:SERVER-URI="https://steeringserver.com",PATHWAY-ID="beta"
#EXT-X-STREAM-INF:BANDWIDTH=1280000,PATHWAY-ID="alpha"
https://cdn1.com/hi/video.m3u8
#EXT-X-STREAM-INF:BANDWIDTH=1280000,PATHWAY-ID="beta"
https://cdn2.com/hi/video.m3u8
```

If an HLS manifest includes several variant streams per encoding ladder, the proper practice is to make all such variant streams available on both CDNs.

In principle, redundant variant streams and BaseURL declarations already existed in earlier HLS and DASH standards versions. Most existing clients already recognize them and use them to implement a basic failover logic for cases of significant network errors [21]. However, the ContentSteering elements are new, providing specific instructions to the clients about which CNDs to use.

When receiving a manifest with content steering elements present, the new HLS / DASH streaming players recognize the existence of steering servers and call them during the session. They issue HTTP GET requests to the steering server URI specified in the manifest. As part of the request, they may include various additional parameters. The parameters specified as recommended by both DASH and HLS specifications, are listed in Table 1.

Table 1 – Parameters communicated by HSL/DASH clients to steering servers.

HLS parameter	**DASH parameter**	**Description**
_HLS_pathway_	_DASH_pathway_	ID of the last pathway used by the client
_HLS_throughput_	_DASH_throughput_	Estimated throughput [bits / sec], as observer by the client in pulling data from the selected CDN

An example of a client's request communicating such parameters to the steering server is provided below:

```
GET "https://steeringserver.com?session=abc&_DASH_pathway=beta&_DASH_throughput=145000"
```

In this example, the client also passes the session ID as a custom parameter in addition to the pathway and throughput parameters.

In response to receiving such a request, the content streaming server generates a response indicating the preferred order of the CDNs (or pathways), the time to call the steering server again (TTL), and the SERVER-URI to use when calling the server next time.

Below, we provide an example of a response that the server can generate:

```
{
 "VERSION": 1,
 "TTL": 300,
 "RELOAD-URI": "https://steeringserver.com?session=abc"
 "SERVICE-LOCATION-PRIORITY": ["beta", "alpha"]
}
```

In this example, the server instructs the client to use pathway "beta" with a higher priority for streaming and then to call the server back in 300 seconds for the next update. The 300 seconds (5 minutes) TTL is a default response interval recommended by HLS specifications.

Once the client receives the steering server response, it checks if the top CDN specified matches the one currently used, and if not, it implements the switch.

The above-described syntax of the steering server response and client-server interactions are the same for HLS and DASH systems, enabling the same server to handle content steering operations.

IMPLEMENTING HLS / DASH CONTENT STEERING SYSTEM

Next, we study the implementation aspects of a multi-CDN streaming system employing the HLS / DASH Content Steering mechanism.

Centralized Steering Server-based Design

Figure 2 shows a possible implementation of the HLS/DASH content steering system. In this design, a single is responsible for all steering decisions. Conceptually, this is the most straightforward implementation of the system.

The objective of the steering server is to direct traffic to each CDN in a way that achieves some beneficial effect. For example, it may perform failover control, increasing the system's

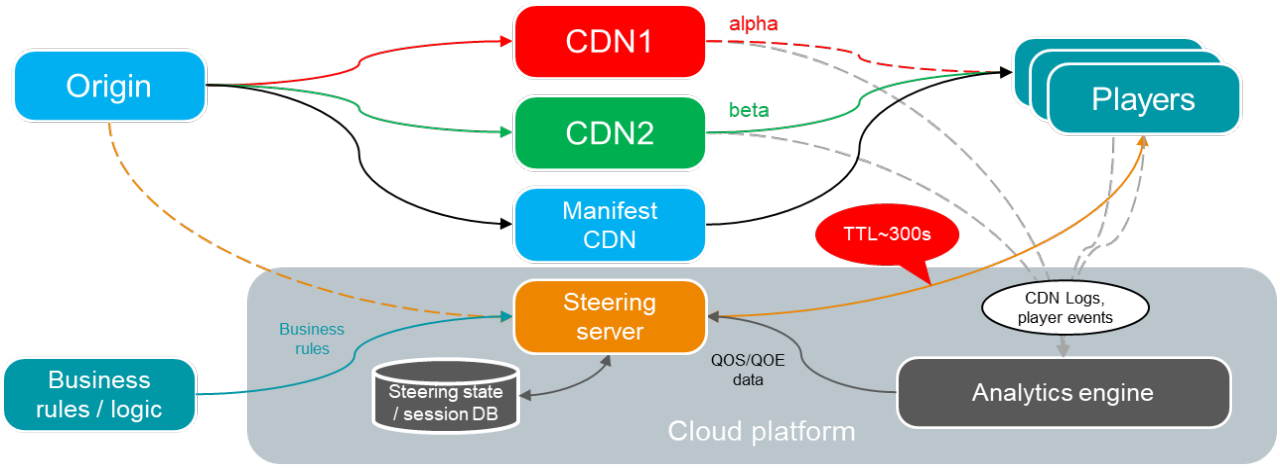

Figure 2 – Centralized server-based implementation of content steering system.

reliability. Or it may perform CDN load balancing, enabling broader distribution. It may also perform QOE/QOS- or COGS-type optimizations.

The steering server may receive at least two types of input information. First, to perform QOE or QOS-based optimizations, it will need to get QOE or QOS data about the system's performance. The usual source of such information is the *analytics engine*, collecting data from the streaming players, origin servers, and CDNs.

The other input that the steering server may receive is a set of *business rules* associated with each CDNs. Such data, for example, may include contract lengths, traffic- or dollar-level commits, per-GB edge traffic rates, etc.

Based on all such inputs, the content steering server decides how to direct traffic to achieve the desired utility (e.g., failover, load distribution, QOE/QOS-, or COGS-based optimization). We note that such decisions must be made periodically, as each client associated with each active session will call the server back at the TTL interval.

Limitations of the Centralized Server-based Design

We next will note some limitations of a system depicted in Figure 2.

The first one is *scalability*. Let us assume, for example, that we have an event watched by 6M of concurrent viewers. Then with 300 seconds TTL, the steering server will need to process at least 20K requests per second. That is a pretty high number! With conventional hardware and some non-trivial logic required for deriving each steering response, it may easily overload a single server or a cluster of servers. In other words, the architecture can't be that simple. It will likely need many servers and appropriate autoscaling and load-balancing logic.

The other issue is the *operating cost*. With the cloud-based implementation, processing each steering response involves compute-time and bandwidth-based costs. Such expenses can be considerable. At least as high as the costs of operating manifest origins, manifest CDNs, and maybe more.

The related issue is the response delay of the system. Reducing steering server TTL, as we just noted, goes against the scalability and costs of the system. Hence it will have to be relatively long, for example, 300 seconds or even longer.

However, such a long TTL dramatically reduces the utility and effectiveness of content steering! While 300 seconds (5 minutes) may be adequate for essential load balancing and CDN commit management tasks, it is inadequate for other objectives, such as QOS/QOE optimizations or rapid enough failover logic. When clients start buffering, directing them to another CDN 5 minutes later is too late!

In other words, we observe that the centralized server implementation of the HLS / DASH content steering method comes with many fundamental limits.

DISTRIBUTED, EDGE-BASED IMPLEMENTATION OF CONTENT STEERING

We next present an alternative implementation of the steering system addressing the above-described limits. Figure 3 shows the overall diagram of our proposed design.

First, instead of using a single content steering server responsible for all decisions in the system, the proposed design splits steering operations into two stages:

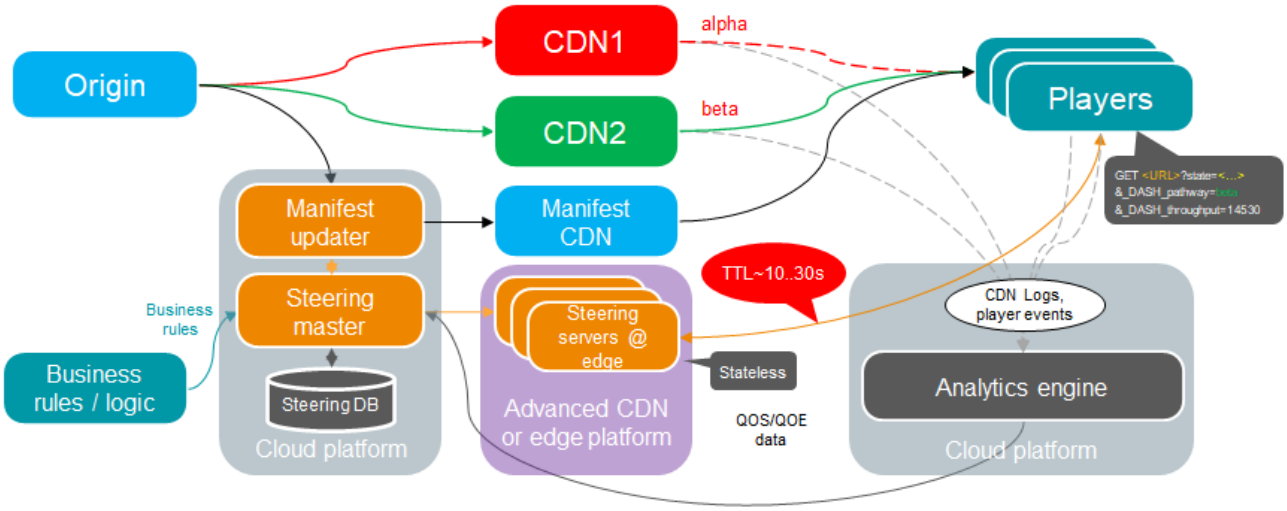

Figure 3 – Edge-based implementation of the content steering system.

- The first stage. Defines the *initial preferred CDN order* for each new streaming session and assigns steering servers to such sessions. We call the server (or a cluster of servers) producing such initial decisions – the *steering master*.
- The second stage. This stage produces all subsequent CDN steering decisions for each streaming session at TTL intervals. We use stateless functions and *edge computing platforms* to implement all such operations.

The proposed two-stage implementation has several key benefits:

- It becomes *massively scalable* - as scalable as CDNs / platforms responsible for executing edge functions;
- it also becomes much *more economical* to deploy - as bandwidth and per/requests costs at CDNs or edge platforms are significantly less expensive than egress traffic costs of cloud platforms
- it also becomes *more responsive*, allowing lower TTL response times between clients and the servers.

Reducing response time is crucial for enabling many additional utilities of the system. Thus, when TTL becomes shorter than the size of the player's buffer (e.g., 10-30 seconds), this automatically enables QOS and QOE-type optimizations — for example, prevention of buffering or allowing clients to use higher quality streams. Shorter response times are critical for graceful failover behavior, disaster recovery, and many other applications.

Regarding possible deployment options, the platforms currently supporting edge processing include AWS / CloudFront with Lambda @ Edge, Fastly's VCL, Akamai Edge Workers, CloudFront Functions, and others [9]. With the rollouts of 3GPP MEC-based services [23] and hybrid ecosystems such as 5G-EMERGE [24], the range of deployment options for such architecture will likely be even broader.

However, in all cases, for a steering server to be deployable at the edge, it must be reduced to a simple *stateless function*. We discuss this design aspect next.

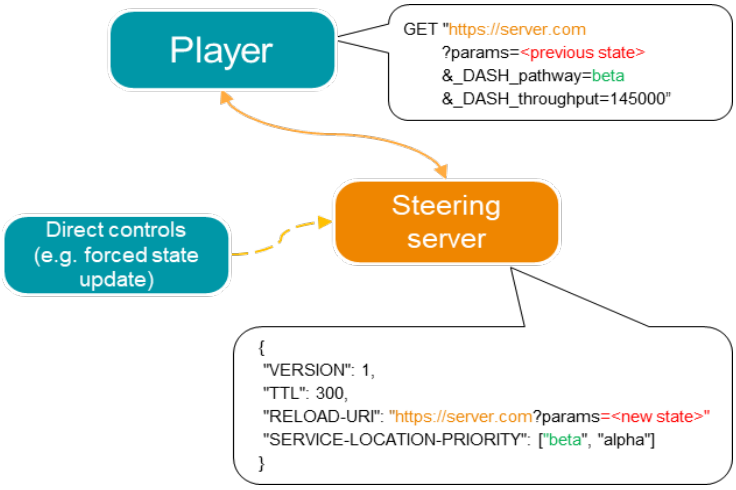

Figure 4 – Exchanges between clients and steering servers carrying the session-related state in the parameter string.

Stateless Implementation of Content Steering Servers

The critical element that enables us to turn the steering server into a stateless function is the *parameter string* used for communication between the streaming client and the server. This string can be specified as part of the SERVER-URI element in the manifest and as part of RELOAD-URI in the steering server response.

Hence, by encoding an internal state and passing it as a parameter string to the client, the server can recover it the next time the client calls it. Such a method allows the server to retain the full context of the session while being invoked as a stateless function on each client's request. We explain the dynamic of such client-server exchanges in Figure 4.

To pass an edge server an initial state, we encode such a state as part of the SERVER-URI string in the manifest. The manifest updater module depicted in Figure 3 does this for each new session.

In our current implementation, the edge server state variables include a few key characteristics of the encoding profile (minimum and maximum bitrates used by its renditions, media duration), current position in the stream, currently observed throughput statistics of all CDNs (as specific to player's region), and the CDN priority list as defined by the steering master. Such state variables allow our edge servers to perform in-session QOE-type delivery optimizations while adhering, to the extent possible, to CDN priorities as set by the steering master.

In our current implementation, we have also added a mechanism allowing CDN order decisions to be forced centrally for all edge servers in a particular region or working with some specific CDNs. Such a mechanism is necessary for testing, manual interventions, disaster recovery efforts, etc.

Distributed Decision Logic

We next discuss the distribution of the decision logic across players, edge servers, and the steering master server in our system. Figure 5 provides a diagram explaining this split.

First, we notice that streaming clients do all final switches. They follow the standards. They recognize the presence of all CDNs/pathways as declared in the manifests and the order of CDNs as provided by the content steering servers. They usually choose the top-priority listed CDN/pathway for delivery. However, in some cases, the clients may also select an

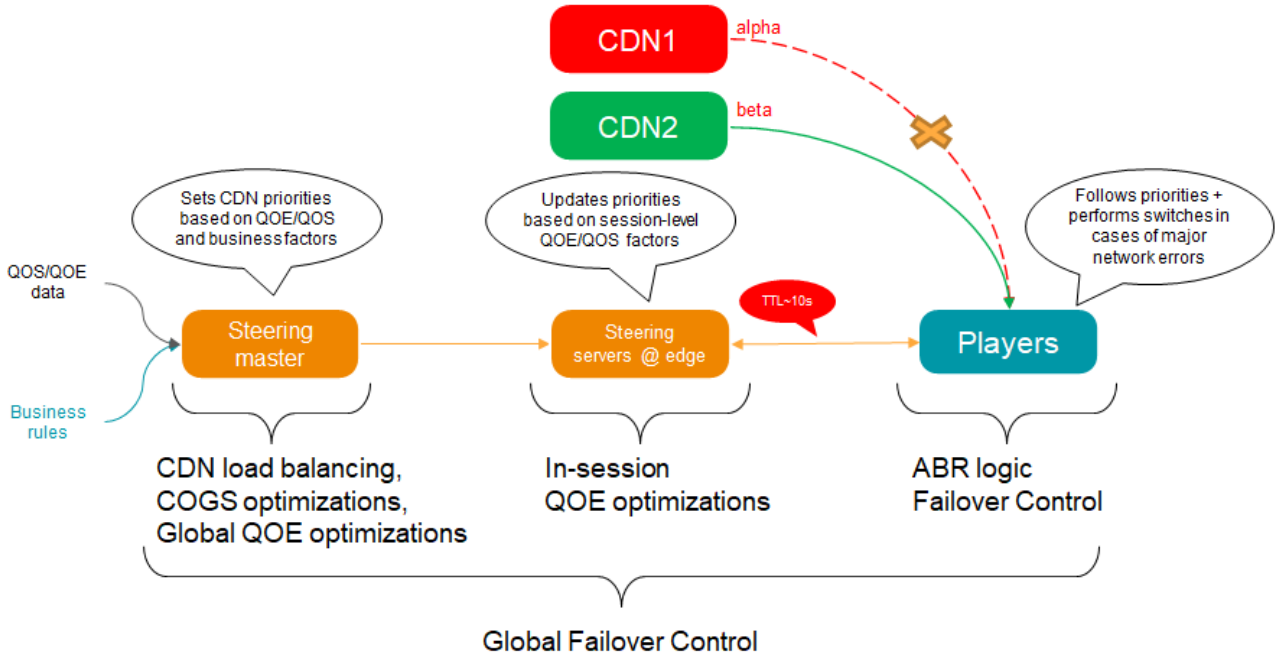

Figure 5 – Distribution of functions in a system with player-, edge-steering servers, and master server levels of control.

alternative CDN by picking the next one on the priority list. Usually, this happens in cases of significant network failures or lack of responses from the default CDN [21]. Effectively, the clients perform failover control logic.

However, each client only observes statistics for the CDN currently in use. It generally does not know what happens simultaneously with other CDNs in the system. Such knowledge is essential for QOS/QOE-type of optimizations. For these reasons, our system uses edge steering servers for in-session level QOE optimizations. As explained earlier, they receive performance statistics for all CDNs in the player's region as part of their initial state. Then they progressively update these statistics based on throughput values reported by the clients. With short enough TTL times, this becomes sufficient to detect degradation in the performance of the current CDN and force switch preventing buffering.

The master steering server in our proposed system architecture is responsible for all regional- or global-level optimizations. These include CDN load-balancing, COGS-based optimizations, CDN contracts commit-level control, etc. Such decisions don't usually require short TTLs, and the per-session granularity of CDN assignments is generally adequate. The regional- or global-level failover actions may also be started at the master server and propagated to edge servers.

With the described distribution of functions, the proposed system architecture can deliver multiple utilities in multi-CDN traffic management while being highly scalable, responsive, and simple to deploy and operate.

OPEN-SOURCE PROJECT IN SVTA

The essential elements of the described system – manifest updaters, steering servers, and testing and deployment scripts are now available as an open-source project within Streaming Video Technology Alliance (SVTA) [17]. Figure 6 shows the landing page of this project in SVTA GitHub.

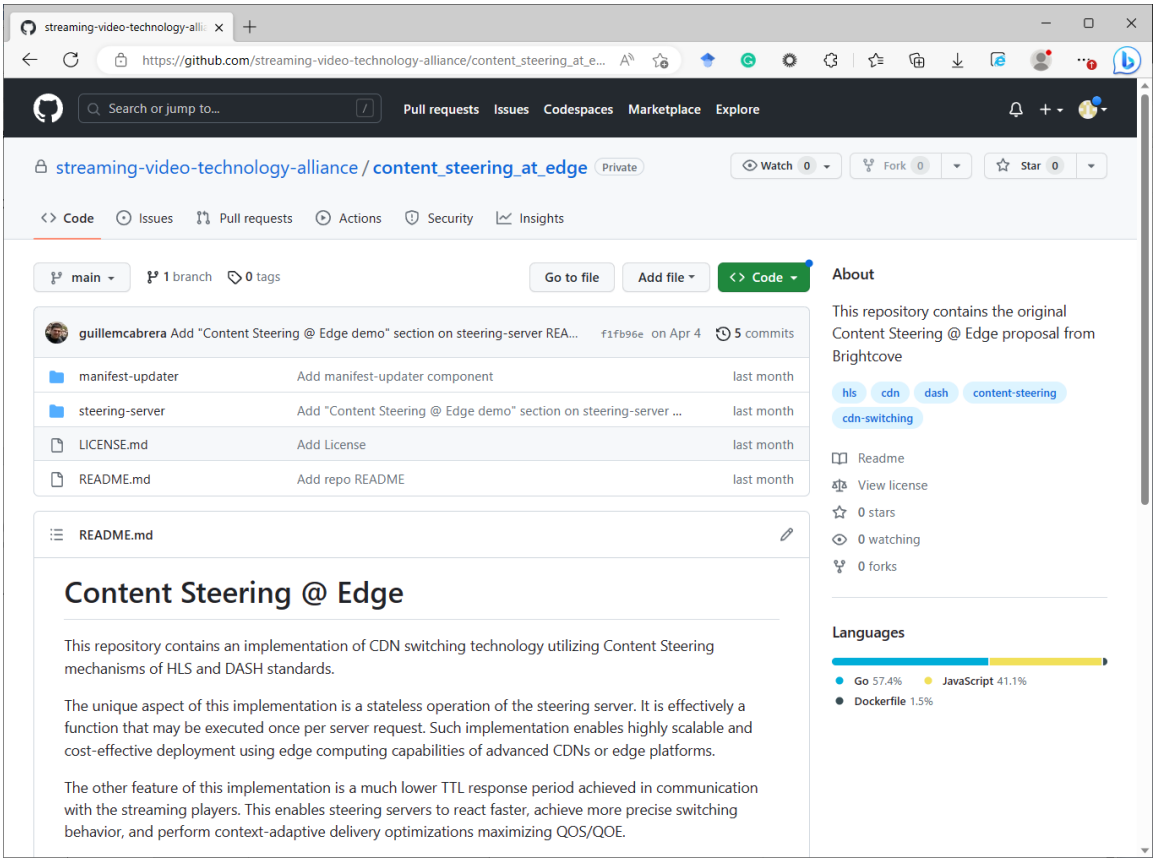

Figure 6 – Content Steering at Edge project page in SVTA GitHub.

This implementation supports HLS and DASH protocols and allows several deployment options. It includes steering servers implemented as standalone servers and edge functions deployable by AWS Lambda @ Edge. The manifest updaters allow deployments as standalone servers or as AWS Lambdas. The system works with DASH.js [20] and HLS.js [19] streaming players. The support for Video.js [24] and several related commercial streaming players [25] is currently being added.

Among functions immediately supported by this open-source project are:

- QOE/QOS optimizations (prevention of buffering)
- Automatic failover functions (switches in cases of failures of either CDNs)
- Manual steering controls (forced changes of CDN priority orders).

All these functions are available from a project demo page, as shown in Figure 7.

When operating this demo, the user can specify the protocol (DASH or HLS), sample content encoded using this protocol, and the streaming player. For testing the effectiveness of QOS/QOE and failover functions of the system, the user activates a network proxy /bandwidth throttling tool. By setting different network conditions for each CDN / pathway, the user can observe the effects of failover prevention of the QOE optimization functions of this system.

The immediate objective of this project is to provide a reference implementation of the HLS/DASH Content-Steering-based system and use it for performance study, along with other CDN solutions currently under investigation by the SVTA alliance [13]. Once fully validated and tested, this project promises to become a reference that would simplify subsequent developments and deployments of highly-scalable practical multi-CDN streaming solutions based on HLS / DASH content steering.

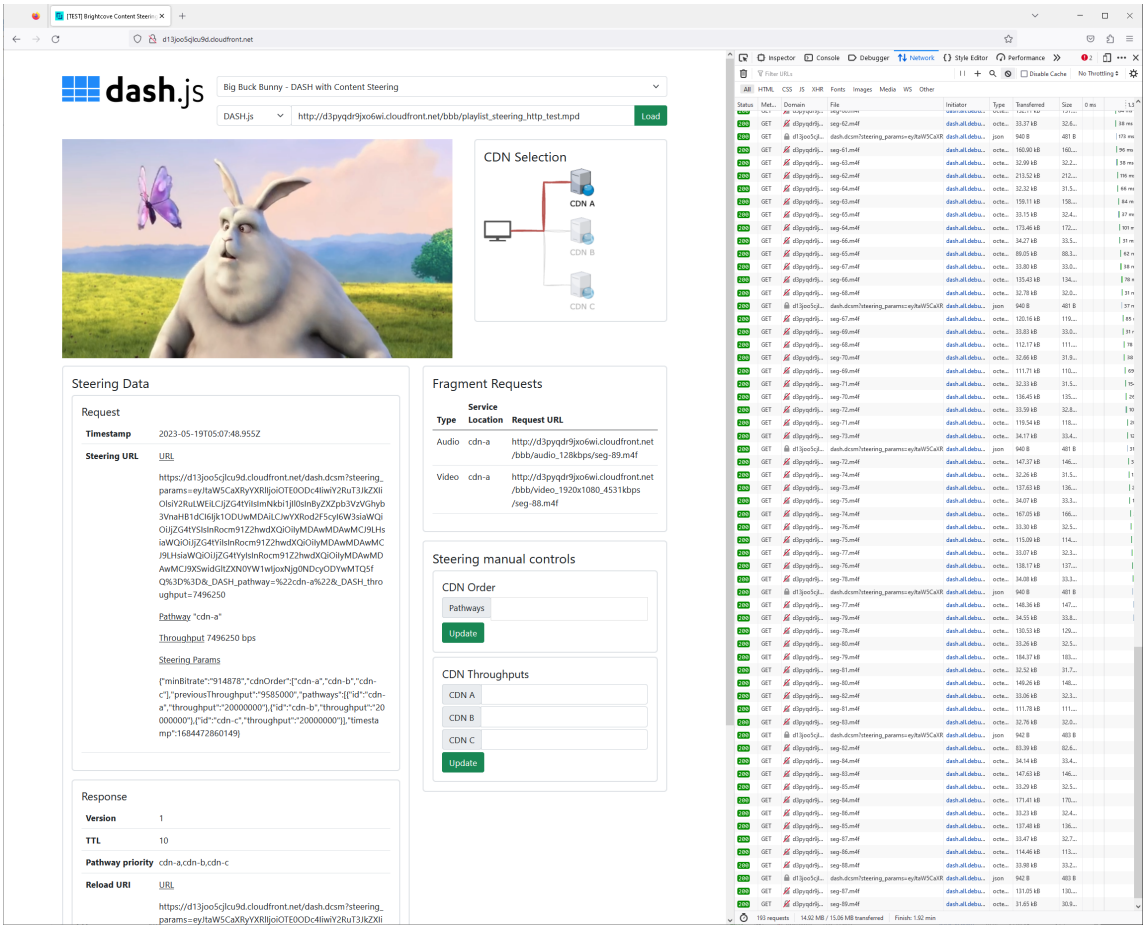

Figure 7 – Test/demo page of edge-based content steering system.

EXPERIMENTAL STUDY

This section reports preliminary experimental results using the SVTA open-source content steering framework.

For testing, we used the well-known "Big Buck Bunny" video sequence [26]. We have encoded it for HLS and DASH streaming using Brightcove CAE encoder [27]. The resulting ladder included four streams with parameters listed in Table 1. We used 2-second segments and an identical file format (CMAF) for both HLS and DASH versions of the streams.

Table 1 – Characteristics of encoded video streams used in our test experiment

Rendition	Video codec	Bitrate [bps]	Resolution	Framerate
1	H.264/AVC	4530860	1920x1080	30
2	H.264/AVC	2445034	1280x720	30
3	H.264/AVC	1419255	1024x576	30
4	H.264/AVC	783322	640x360	30

We subsequently placed all streams on two HTTP servers and with extra instrumentation added to control their respective output bandwidth/throughputs. We called the URLs leading

to these servers "CDN-A "and "CDN-B," respectively. Figure 8 shows the pattern shape used to modulate the bandwidth along each pathway during streaming sessions.

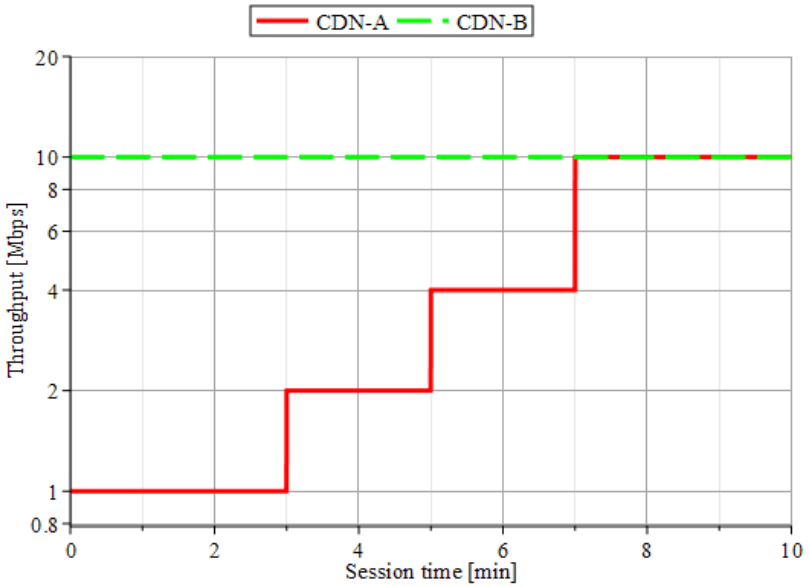

Figure 8 – Test pattern used to limit client-side throughput of CDNs/pathways used for delivery.

As easily observed, the test pattern is straightforward, with CDN A being worse than CDN B, particularly at the beginning. It also effectively tests the range of bitrates as present in the encoding ladder (Table 1). If a player stays with CDN A longer, it will likely switch and buffer more throughout the session. Each session is about 10 minutes long (the length of the "Big Buc Bunny" sequence).

To measure system performance, we have instrumented streaming clients to report standard performance metrics, such as streaming startup time, the number of buffering events, the number of seconds spent buffering, current rendition bitrate, etc. The startup time reflects the difference between "loadstart" and "loadeddata" events in HTML5. The rest of the metrics are reported natively by clients (HLS.js and DASH.js, respectively).

For testing, we have considered the following possible systems/scenarios:

1) DASH playback without content steering,
2) DASH playback with SVTA edge content steering,
3) HLS playback without content steering
4) HLS playback with SVTA edge content steering.

In all cases, CDN-A served as a default (higher priority) pathway at the beginning of the test session. We used the SVTA edge steering server to control subsequent CDN priorities dynamically. In all cases, we repeated each session/experiment 10 times. Table 2 shows the resulting average statistics.

Table 2 – Performance metrics and test results collected.

System under test	Startup time [ms]	Buffering events	Buffering seconds	Average bitrate	Rendition switches
DASH, no steering	1231	35	80.1	783.8	2
DASH, with content steering	1216	2	21.1	4327.1	1
HLS, no streering	1579	8	30.6	2371.3	2
HLS, with content steering	1834	1	11.5	4497.1	1

While these results are preliminary and limited to a few artificial cases, we find them highly encouraging. We see that steering brings significant improvements in reducing the number of buffering events and number of seconds spent buffering. We also see a significant (2-4x) increase in the average bitrate delivered to the viewers. And interestingly enough, the number of rendition switches is also lower due to intelligent choices of CDNs.

All these factors indicate that content steering has excellent potential for delivering improved QOE, reliability, and effectiveness of multi-CDN delivery systems.

CONCLUSIONS

This paper reviewed the Content Steering technology recently introduced in both HLS and DASH standards. We have discussed the advantages of this technology, and we have also identified some challenges with its realization and deployment at scale. To address these challenges, we have proposed a distributed deployment model utilizing edge processing functions of modern CDNs and edge platforms. We have shown that this model is highly scalable, allows short response time, and enables a full spectrum of multi-CDN management functions: load balancing, failover protection, and COGS- and QOE/QOS-based optimizations. The proposed system has been implemented and contributed to an open-source project within Streaming Video Technology Alliance. With additional refinements, testing, and validations performed now as a community effort, we believe it could provide a helpful reference enabling the industry to accelerate developments and deployments of highly-efficient multi-CDN streaming solutions.

REFERENCES

[1] D. Wu, Y.T. Hou, W. Zhu, Y-Q. Zhang, and JM Peha, "Streaming video over the internet: approaches and directions," IEEE Trans. CSVT, vol. 11, no. 3, pp. 282-300, 2001.

[2] B. Girod, M. Kalman, Y.J. Liang, and R. Zhang, "Advances in channel-adaptive video streaming," Wireless Comm. and Mobile Comp., vol. 2, no. 6, pp. 573-584, 2002.

[3] G. J. Conklin, G. S. Greenbaum, K. O. Lillevold, A. F. Lippman, and Y. A. Reznik, "Video coding for streaming media delivery on the internet," IEEE Trans. CSVT, vol. 11, no. 3, pp. 269-281, 2001.

[4] Bentaleb, B. Taani, A. C. Begen, C. Timmerer, R. Zimmermann, “A Survey on Bitrate Adaptation Schemes for Streaming Media Over HTTP,” in IEEE Communications Surveys & Tutorials, vol. 21, no. 1, 2019, pp. 562-585.

[5] Y. A. Reznik, K. O. Lillevold, A. Jagannath, and X. Li. 2021. Towards Understanding of the Behaviour of Web Streaming. In 2021 Picture Coding Symposium (PCS). 1–5.

[6] Y. Reznik, X. Li, K.O. Lillevold, R. Peck, T. Shutt, and P. Howard, "Optimizing Mass-Scale Multi-Screen Video Delivery," SMPTE Motion Imaging Journal, vol. 129, no. 3, pp. 26-38, April 2020.
[7] R. Pantos, and W. May, "HTTP live streaming, RFC 8216," https://tools.ietf.org/html/rfc8216, 2017.
[8] ISO/IEC 23009-1:2022, "Information technology - Dynamic adaptive streaming over HTTP (DASH) - Part 1: Media presentation description and segment formats," October 2022.
[9] Mind Commerce, "CDN Market by Technology, Platform, Application, Service Type, Customer Type, and Industry Verticals 2021 – 2027", 2021.
[10] EBU TR 068, "CDN Architectures Demystified," EBU, Geneva, June 2022. https://tech.ebu.ch/publications/tr068
[11] D. Hassoun, "How to Jump-Start Your Multi-CDN Strategy and Deliver Every Time", Oct. 2019: https://www.streamingmedia.com/Articles/ReadArticle.aspx?ArticleID=134765
[12] Muvi blog post: Multi-CDN switching methods: https://www.muvi.com/blogs/multi-cdn-switching-in-streaming-businesses.html
[13] SVTA investigation of approaches to CDN delivery https://www.svta.org/2023/01/03/investigating-approaches-to-multi-cdn-delivery
[14] HLS Content Steering Specification (v1.2b1) https://developer.apple.com/streaming/HLSContentSteeringSpecification.pdf
[15] RFC 8216, Section 7: Content steering https://datatracker.ietf.org/doc/html/draft-pantos-hls-rfc8216bis#section-7
[16] DASH-IF CTS Version 0.9.0 – https://dashif.org/docs/DASH-IF-CTS-00XX-Content-Steering-Community-Review.pdf
[17] Content Steering at Edge, an open-source project: https://github.com/streaming-video-technology-alliance/content_steering_at_edge (available to SVTA members)
[18] AVFoundation, https://developer.apple.com/av-foundation/
[19] Hls.js player, https://github.com/video-dev/hls.js/
[20] DASH.js player, https://github.com/Dash-Industry-Forum/DASH.js
[21] P. Cluff, "Survive CDN failures with redundant streams", September 2020, https://www.mux.com/blog/survive-cdn-failures-with-redundant-streams
[22] 3GPP MEC, https://www.3gpp.org/news-events/partner-news/mec
[23] 5G-EMERGE project, https://www.5g-emerge.com/
[24] Video.js player, https://videojs.com/
[25] Brightcove VideoCloud system, https://videocloud.brightcove.com
[26] Blender Foundation, Big Buck Bunny video sequence, https://peach.blender.org/
[27] Brightcove Context Aware Encoding, https://www.brightcove.com/en/products/online-video-platform/context-aware-encoding/
[28] Charles proxy, https://www.charlesproxy.com/documentation/proxying/throttling/