**Existing CDN switching architectures:**

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Pros</th>
<th>Cons</th>
</tr>
</thead>
<tbody>
<tr>
<td>DNS-based</td>
<td>This is the simplest of all solutions since the source video URL always remains constant.</td>
<td>Switch delay is more time-consuming, ranging from 300 seconds to even five minutes in case of CDN failures. This can immensely hamper the user QoE.</td>
</tr>
<tr>
<td>On-the-fly manifest rewrite</td>
<td>Better user experience due to midstream switching eliminating the need for hard refresh during video playback. No matter the volume of simultaneous session resets, this method reduces the chances of a cascade effect that may hamper the video workflow.</td>
<td>Rewriting the manifest can sometimes bring about errors. Midstream switching is not completely seamless, and takes time for the server to understand that a particular CDN is unavailable.</td>
</tr>
<tr>
<td>Server-side</td>
<td>It is a relatively simple CDN switching method to implement since changes happen in the server itself that is easier for the operator to control.</td>
<td>Page loading may take some time, adding to delays. Since CDN switching is based on the collective data from many clients, it does necessarily consider the unique conditions of the actual clients.</td>
</tr>
<tr>
<td>Client-side</td>
<td>QoS data is almost accurate as it is fetched based on individual clients' local and real-time performance metrics. Seamless midstream CDN switching is possible.</td>
<td>It is a complex procedure to implement when built in-house due to the code complexity of the algorithms that requires detailed planning.</td>
</tr>
</tbody>
</table>

https://www.svta.org/2023/01/03/investigating-approaches-to-multi-cdn-delivery/
HLS / DASH CONTENT STEERING

General concept

Pros:
- Standards based
- The same steering protocol & server is used for both HLS and DASH
- Simple integration – no need to patch players!
- Complements the existing BaseURL redundancy / failover behavior mechanisms
**CONTENT STEERING SERVER**

**Direct implementation**

- **Origin** → **CDN1** (alpha)
- **Origin** → **CDN2** (beta)
- **Manifest CDN** → **Players**

**Challenges**

- **TTL time:** 300s *default* is too long! Suitable for basic CDN load balancing. Not suitable for QOE optimizations!
- **Scalability:** the steering server should be at least as scalable as manifest CDN!
- **Costs:** reducing TTL will increase number of requests and traffic to the steering server!
**CONTENT STEERING @ EDGE**

**Proposed architecture**

- **Origin**
- **CDN1**
- **CDN2**
- **Players**

**Benefits**

- Scales well with CDNs or edge platforms.
- TTL can be smaller; comparable to player buffer delay; Can be used to optimize QOE!
OPEN SOURCE PROJECT IN SVTA

Elements:
- Component 1: manifest updater inserting content steering information in the manifests (Golang)
- Component 2: steering server implementation with several deployment variants (Node.js, Lambda @ Edge, etc.)
- Component 3: DASH and HLS players: DASH.js and HLS.js – existing open source projects
Standalone edge server & player:
- Maintaining preferred CDN order
- Failover functions
  - reaction to network failures at either pathway
  - forced updates
- QOE optimizations
  - reactions to degraded performance of current CDN
  - lowering buffering rate

With master server (not part of open source):
- CDN load balancing
- Overall QOE optimizations
- Delivery cost optimizations