Adapting your player logic to your use case: how to use ABR to your advantage

Streaming Media East – Track D
Wednesday, May 17, 2017
10:30 to 11:30 am
Streamroot: Who are we?

- Pioneers in hybrid video delivery systems to accompany exponential growth in OTT traffic
- Experts in HTML5 video through integrations into open-source & proprietary players
- Consultants in the transition from Flash to HTML5
Presentation Outline

I. Introduction: What are we trying to accomplish? Why does this matter?

II. Basic building blocks: bandwidth-based algos
   - Constraints
   - Parameters
   - Examples: hls.js & dash.js

III. Addition logic: buffer-based algorithms
   - Concept, pros and cons, hybrid approaches
   - Examples: BOLA

IV. Real-life example: Dailymotion

V. Going further
   - The key to improving: testing and iterating
I. Design goals
I. Why ABR?

Multiplicity of network conditions and devices $\rightarrow$ need to dynamically select resolution

HTTP / TCP stack $\rightarrow$ removal from the transport protocol $\rightarrow$ client-level estimation & decisions

Source: FESTIVE diagram of HTTP streaming
I. Design Goals

1. Maximize efficiency – stream at the highest bitrate possible
2. Minimize rebuffering – avoid underrun and playback stalls
3. Encourage stability – switch only when necessary

(4. Promote fairness across network bottlenecks)
II. Building blocks: bandwidth estimation
# II. Basic building blocks

<table>
<thead>
<tr>
<th>CONSTRAINTS</th>
<th>PARAMETERS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Screen size / Player size</td>
<td>Buffer size</td>
</tr>
<tr>
<td>Dropped frame threshold</td>
<td>Bandwidth estimation</td>
</tr>
<tr>
<td>Startup time / Rebuffering recovery</td>
<td><strong>Bonus</strong>: P2P Bandwidth</td>
</tr>
</tbody>
</table>
II. The Basics: Constraints

1. Screen & Player Size
   Resolution should never be larger than the actual size of the video player

2. Dropped frame rate
   Downgrade when too many dropped frames per second

3. Startup time
   Fetch the lowest quality first when the buffer is empty
Available bandwidth estimation \(\rightarrow\) maximize bitrate

Estimate the available bandwidth based on last segment
Available bandwidth = size of chunk / time taken to download

Basic bandwidth-based algorithm:

**Estimation**

- Network conditions
- Estimate bandwidth from last segment stats

**Bitrate selection**

- Choose first bitrate < estimated bandwidth
- Schedule download
II. The Basics: Bandwidth estimation

More complex bandwidth-based algorithm:

Estimation: take history into account!
II. The Basics: Bandwidth estimation

More complex bandwidth-based algorithm:

- **Estimation**: Take into account historical values, not just the last segment.
- **Smoothing**: Apply a smoothing function to the range of values obtained.
  - Possible functions: average, median, EWMA, harmonic mean
II. The Basics: Bandwidth estimation

More complex bandwidth-based algorithm:

Quantizing: quantize the smoothed bandwidth to a discrete bitrate
II. The Basics: Buffer control mechanism

Buffer fill size → Minimize rebuffering ratio

Rebuffering ratio = buffering time / (buffering time + playback time)

Abandon strategy
EXAMPLES
Examples: HLS.js & DASH.js

Both are dominant HTML5 MSE-based players


https://github.com/video-dev/hls.js
Examples: algorithms used

<table>
<thead>
<tr>
<th>HLS.js</th>
<th>DASH.js</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 Main:</td>
<td>2 Main:</td>
</tr>
<tr>
<td>Bandwidth Estimation</td>
<td>ThroughputRule</td>
</tr>
<tr>
<td>Abort rules</td>
<td>AbandonRequestsRule</td>
</tr>
<tr>
<td>2 secondary:</td>
<td>2 secondary:</td>
</tr>
<tr>
<td>CapLevel</td>
<td>BufferOccupancyRule</td>
</tr>
<tr>
<td>FPS (Dropped frames per second)</td>
<td>InsufficientBufferRule</td>
</tr>
</tbody>
</table>
Screen & player size: HLS.js

Checks the max CapLevel corresponding to current player size

Frequency: every 1000 ms

https://github.com/dailymotion/hls.js/blob/master/src/controller/cap-level-controller.js#L68
Dropped frames: HLS.js

Calculates the dropped frames per second ratio.

If > 0.2, bans the level forever → goes into restricted capping levels

fpsDroppedMonitoringThreshold
fpsDroppedMonitoringPeriod

https://github.com/dailymotion/hls.js/blob/master/src/controller/fps-controller.js#L33
Startup time strategies

DASH.js

Always loads the lowest quality when buffer empty

HLS.js

First segment loaded from the first level in the playlist (capped by the cap level rule), then continues with normal ABR rule.

https://github.com/dailymotion/hls.js/blob/master/src/controller/stream-controller.js#L131
Bandwidth estimation: HLS.js smoothing and quantization

Adding levels quantization to reduce oscillation

(Bonus) P2P bandwidth estimation: HLS.js

CDN

Request time = 3 seconds

P2P Cache

Media Buffer

Request time = 0ms!
Bandwidth fragment abort rule

HLS.js

1. fragment onProgress()
2. requestDelay > 0.5
   - yes
   - no
3. fragDuration?
   - yes
   - no
4. Can another level avoid starvation?
   - yes
   - no
5. ABORT segment load & Load new level

DASH.js

1. Fragment Loader
   - onprogress()
   - doLoad()
   - onload()
   - HTTP REQUEST
2. HTTP RESPONSE
3. CDN
4. AbandonRequest Rule
   - estimateTimeOfDownload()
   - true
   - false
   - true
   - false
   - true
   - SwitchRequest (newQuality, STRONG)

https://github.com/dailymotion/hls.js/blob/master/src/controller/abr-controller.js#L51
(Bonus) DASH.JS complementary buffer-based rules

Prevents from switching down when buffer is large enough (mostly for VOD)

III. Another approach: buffer-based algorithms
III. Buffer based algorithms

Pitfalls of bandwidth-based estimations:

- Not resilient to sudden network fluctuations
- Often leads to bitrate oscillations
- Biased by HTTP/TCP calls on the same device/network

→ What if we only used the buffer size?
III. Buffer-based algorithms: concept

IF Output Rate $>$ Input Rate THEN Switch Down

IF Input Rate $>$ $K \times$ Output Rate THEN Switch Up

IF BufferSize $>$ Buffer Threshold THEN Stay at Max Rate
Advantages of buffer-based algorithms:

• Fewer estimation issues
• Particularly effective when throughput is highly variable

BUT:

• Too conservative during ramp-up phase
• Issues when you have different chunk sizes (VBR)
• Issues when you don’t have enough buffer!

→ HYBRID Buffer-level based algorithms, with bandwidth measurement used only when needed (at start-up)
III. Buffer-based algorithms: pros and cons

→ HYBRID Buffer-level based algorithms, with bandwidth measurement used only when needed

For VOD:
On startup, while BufferSize<Bmax, Use bandwidth-based algo to ramp up more aggressively.

For Live and short-content VOD:
In the case Playlist length << Bmax, Always use the Bandwidth-based approach
DASH.js BOLA, another approach


Uses Utility Theory:

Maximize $V_n + y S_n$

Where:
$V_n$ is the bitrate utility
$S_n$ is the playback Smoothness
$y$ is the tradeoff weight parameter
IV. Real-time QoS Improvements: Dailymotion
IV. Real-time QoS Improvements: Dailymotion

World’s 2\textsuperscript{nd} largest video hosting platform

\textbf{dailymotion}

>100 million video views each day

63% Desktop
31% Mobile
8% Tablet
1% SmartTV

© 2017 Streamroot - All rights reserved
1. Start taking historical values.
2. Use EWMA to smooth.
IV. Real-time QoS Improvements: Dailymotion
Improving hls.js’s ABR logic

EWMA:

Shaka Player:

```javascript
shaka.abr.EwmaBandwidthEstimator = function() {
  /**
   * A fast-moving average.
   * Half of the estimate is based on the last 3 seconds of sample history.
   * @private {!shaka.abr.Ewma}
   */
  this.fast_ = new shaka.abr.Ewma(3);

  /**
   * A slow-moving average.
   * Half of the estimate is based on the last 10 seconds of sample history.
   * @private {!shaka.abr.Ewma}
   */
  this.slow_ = new shaka.abr.Ewma(10);
}
```

IV. Real-time QoS Improvements: Dailymotion

Comparing the results - live

Rebuffering, percentile ranks

67.4% native
70.3% hls.js
78.6% flashls
IV. Real-time QoS Improvements: Dailymotion

Comparing the results - live

Rebuffering, percentile ranks

66.29% native
78.03% flashls
69.05% hls.js,s=6,f=1
70.01% hls.js,s=0,f=0
79.15% hls.js,s=9,f=3
IV. Real-time QoS Improvements: Dailymotion

Number of level switches - live

hls.js, s=9, f=3

hls.js, s=0, f=0
IV. Real-time QoS Improvements: Dailymotion

Conclusions

With the right workflow, big improvement in less than 2 weeks of testing.

Numbers are use-case specific (segment length, network, etc.).

Lots of other options to test → a continuous effort to improve QoS!
V. Going further
V. Going further: understand your use case

Streaming type: VOD / Live?

Devices: Desktop? Mobile? STBs?

Service: Premium? Ad based?

With data, personalize even more:
- Per country
- Per ISP
- Per network connection type
- Per viewer habits!
## V. Going further: how to improve in practice

### HLS.js

1. Tweak the parameters
   
   [https://github.com/dailymotion/hls.js/blob/master/API.md#fine-tuning](https://github.com/dailymotion/hls.js/blob/master/API.md#fine-tuning)

2. Write your own rules!

   AbrController: AbrController  
   capLevelController: CapLevelController,  
   fpsController: fpsController

### DASH.js

1. Tweak the Parameters
   

2. Write your own rules
   

V. Going further: test and iterate!

Tweaking algorithms is easy, creating your forks too.

You’ve got the power!

- Know what is important to you (buffering, max bitrate, bandwidth savings...)
- Compare and cross with QoS analytics to understand your audiences
- Test and iterate: AB testing allows you to compare changes in real-time

→ Significant improvements without even changing your workflow!
Questions?
Further Reading / Contact Us

Nikolay Rodionov, Co-Founder and CPO, nikolay@streamroot.io
Erica Beavers, Head of Partnerships, erica@streamroot.io

Additional resources:

*Probe and Adapt: Rate Adaptation for HTTP Video Streaming At Scale.* Zhi Li, Xiaoqing Zhu, Josh Gahm, Rong Pan, Hao Hu, Ali C. Begen, Dave Oran, Cisco Systems, 7 Jul 2013.


*ELASTIC: a Client-side Controller for Dynamic Adaptive Streaming over HTTP (DASH).* Luca De Cicco, Member, IEEE, Vito Caldaralo, Vittorio Palmisano, and Saverio Mascolo, Senior Member, IEEE.

*BOLA: Near-Optimal Bitrate Adaptation for Online Videos.* Kevin Spiteri, Rahul Urgaonkar, Ramesh K. Sitaraman, University of Massachusetts Amherst, Amazon Inc., Akamai Technologies Inc.

*A Buffer-Based Approach to Rate Adaptation: Evidence from a Large Video Streaming Service.* Te-Yuan Huang, Ramesh Johari, Nick McKeown, Matthew Trunnell, Mark Watson, Stanford University, Netflix