Grand Challenge on

Adaptation Algorithms for Near-Second Latency

Organized and sponsored by

Challenge Description

At Twitch, we have been successful in delivering ultra-low-latency streams to millions of viewers. However, as we drive towards near-second latency, we are finding that existing adaptation algorithms are not able to keep up - smaller player buffers do not provide enough time to respond to changing network conditions. We see this as a key challenge blocking the streaming community from reducing latency at scale.
Grand Challenge

- **Goal:** To build and test a low latency ABR algorithm
- **Given**
  - Fork of Dash.js modified to pre-request low latency segments
  - Low latency DASH server
- **Evaluation**
  - 5 network patterns to simulate network conditions
  - Quality of Experience
    - Bitrate
    - Bitrate switches
    - Rebuffering time
    - Live latency
    - Playback speed
Background

Live stream latency = end-to-end latency

Camera → Encoder → Streaming Server → Internet → Client

Network latency (Round Trip Time)
Background

Live stream latency = end-to-end latency

Camera → Encoder → Streaming Server → Internet → Client

Network latency (Round Trip Time)

Quality:
- High
- Med.
- Low

Playback Time:
- Low Resolution Video Segment
Live Stream Latency

STREAMING LATENCY AND INTERACTIVITY CONTINUUM

- COMMON HTTP LATENCIES TODAY
  - One-way streams of live events to large audiences; linear programming

- REDUCED LATENCY
  - OTT providers; live-streaming news and sports

- LOW LATENCY
  - UGC live streams; game streaming and e-sports

- NEAR REAL-TIME
  - Two-way web conferencing; telepresence; real-time device control (e.g., PTZ cameras, drones)

45+ seconds
18 seconds
05 seconds
01 second
<01 second

Graphic Credit: Wowza

Challenge Goal
Target Latency

- Challenge Requirements
  - 0.5 second segments
  - Live Latency: 1-2 seconds
  - Example: Target latency = 1 sec
Target Latency

- Challenge Requirements
  - 0.5 second segments
  - Live Latency: 1-2 seconds
- Example: Target latency = 1 sec

Camera → Encoder → Streaming Server → Internet → Client

Buffer: 0.4 s
Target Latency

- **Challenge Requirements**
  - 0.5 second segments
  - Live Latency: 1-2 seconds
  - Example: Target latency = 1 sec
Target Latency

- Challenge Requirements

- Example: Target latency = 1 sec

Run out of video in the buffer!
ABR Algorithms

• Throughput-based algorithms
  – Estimate network throughput available between the client and the server
  – Averaging of segment burst times
    • Sliding window
    • Expected weighted moving average (EWMA)
  – Examples: Festive, PANDA, and Squad

• Buffer-based algorithms
  – Use buffer level to decide on the bitrate of the next segment
  – Examples BBA, BOLA

• Hybrid algorithms
  – Use both through prediction and buffer level to decide on the bitrate of the next segment
  – Examples ELASTIC, MPC, DYNAMIC
  – DYNAMIC is the default ABR algorithm provided by the standard reference player dash.js

[ABD11] [JSZ12] [CCP13] [LZG14][YJS15] [HJM15] [XYJ16][SSS18]
Throughput Measurements

- Average Throughput determined by DYNAMIC’s Throughput Rule

\[ \text{segment throughput} = \frac{\text{size}_{\text{segment}}}{\text{end}_{\text{segment}} - \text{start}_{\text{segment}}} \]
Throughput Measurements

- Average Throughput determined by DYNAMIC’s Throughput Rule

\[
\text{segment throughput} = \frac{\text{size}_{\text{segment}}}{\text{end}_{\text{segment}} - \text{start}_{\text{segment}}}
\]

System does not react fast enough to network variations, causing stalls and variable resolutions.
STALLION ABR

- Unstable network conditions result in client bandwidth fluctuations during playback.
- Stallion uses a sliding window to measure the mean and standard deviation of both the bandwidth and latency.

<table>
<thead>
<tr>
<th>Notation</th>
<th>Meaning</th>
<th>Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C$</td>
<td>Throughput</td>
<td>$C_{safe} = \hat{C} - z_C \sigma_C$</td>
</tr>
<tr>
<td>$l$</td>
<td>Latency</td>
<td>$l_{safe} = \hat{l} + z_l \sigma_l$</td>
</tr>
<tr>
<td>$R$</td>
<td>Segment bitrate</td>
<td></td>
</tr>
<tr>
<td>$z$</td>
<td>Safety factor</td>
<td></td>
</tr>
<tr>
<td>$\sigma$</td>
<td>Sample standard deviation</td>
<td></td>
</tr>
<tr>
<td>$d$</td>
<td>Segment Length</td>
<td>$R_{max} = C_{safe} \times (1 - \frac{l_{safe}}{d})$</td>
</tr>
</tbody>
</table>
Implementation

DYNAMIC ABR Controller

Throughput

Insufficient Buffer

Switch History

Dropped Frame

Client

Segment Request

ABR Decision

Video Segment

Video Player

Network And Buffer Statistics

ABR Controller

Logger

Modified/Changed Blocks in Red
Stallion Implementation

Stallion ABR Controller

- Stallion
- Switch History
- Dropped Frame

Client

- ABR Controller
- ABR Decision
- Network And Buffer Statistics
- Video Player
- Segment Request
- Video Segment

Streaming Server

Logger

Modified/Changed Blocks in Red

- https://github.com/Wimnet/twitch_challenge
Performance Evaluation

- Network profiles used to emulate varying network conditions

<table>
<thead>
<tr>
<th>Notation</th>
<th>Time Duration (s)</th>
<th>Data Rate (Kbps)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cascade</td>
<td>30,30,30,30,30</td>
<td>1200,800,400,800,1200</td>
</tr>
<tr>
<td>Intra Cascade</td>
<td>15,15,15,15,15,15,15,15</td>
<td>1000,800,600,400,200,400,600,800,1000</td>
</tr>
<tr>
<td>Spike</td>
<td>10,10,10</td>
<td>1200,300,500</td>
</tr>
<tr>
<td>Slow Jitters</td>
<td>5,5,5,5,5,5</td>
<td>500,1200,500,1200</td>
</tr>
<tr>
<td>Fast Jitters</td>
<td>0.25,5,0.1,1,0.25,5</td>
<td>500,1200,500,1200</td>
</tr>
</tbody>
</table>
Performance Evaluation

- Server and dash player run on Apple Macbook Pro
- Video (Big Buck Bunny) encoded in 3 bitrates (200 Kbps, 600 Kbps, 1000 Kbps)
- Each ABR algorithm was run 20 times on each network profile
- Compare vs. DYNAMIC, the default ABR algorithm provided by the standard reference player dash.js
Quality of Experience

- **Selected Bitrate**
- **Stall duration**
- **Live latency**
- **Playback Speed**
- **Number of bitrate switches**
- **QoE metric provided by the Challenge:**

\[
QoE = \sum_{s=1}^{S} (\alpha R_s - \beta E_s - \gamma L_s - \sigma |1 - P_s|) - \sum_{s=1}^{S-1} \mu |R_{s+1} - R_s|
\]

Results

\[ QoE = \sum_{s=1}^{S} (\alpha R_s - \beta E_s - \gamma L_s - \sigma |1 - P_s|) - \sum_{s=1}^{S-1} \mu |R_{s+1} - R_s| \]
Results

\[ QoE = \sum_{s=1}^{S} (\alpha R_s - \beta E_s - \gamma L_s - \sigma |1 - P_s|) - \sum_{s=1}^{S-1} \mu |R_{s+1} - R_s| \]
Results

\[ QoE = \sum_{s=1}^{S} (\alpha R_s - \beta E_s - \gamma L_s - \sigma |1 - P_s|) - \sum_{s=1}^{S-1} \mu |R_{s+1} - R_s| \]
Results

\[ QoE = \sum_{s=1}^{S} \left( \alpha R_s - \beta E_s - \gamma L_s - \sigma |1 - P_s| \right) - \sum_{s=1}^{S-1} \mu |R_{s+1} - R_s| \]
Results

\[ QoE = \sum_{s=1}^{S} \left( \alpha R_s - \beta E_s - \gamma L_s - \sigma |1 - P_s| \right) - \sum_{s=1}^{S-1} \mu |R_{s+1} - R_s| \]
Results

\[ QoE = \sum_{s=1}^{S} (\alpha R_s - \beta E_s - \gamma L_s - \sigma |1 - P_s|) - \sum_{s=1}^{S-1} \mu |R_{s+1} - R_s| \]
Summary

• Challenge Goal: To develop and test a low latency ABR algorithm
• Developed Stallion which relies on using the standard deviation of measured throughput to give a safe request for the segment bitrate
• Stallion shows 1.8x increase in bitrate, and 4.3x reduction in the number of stalls compared to DYNAMIC
• Beyond the challenge
  • Additional network profiles
  • Improve the safe throughput estimate
  • Add a machine learning system to auto tune Stallion parameters
Thank You!

STALLION: Video Adaptation Algorithm for Low-Latency Video Streaming

Craig Guttermman, Brayn Fridman, Trey Gilliland, Yusheng Hu, Gil Zussman
Columbia University
clg2168@columbia.edu