Bandwidth Prediction in Low-Latency Chunked Streaming

Mile High Video – July 2019

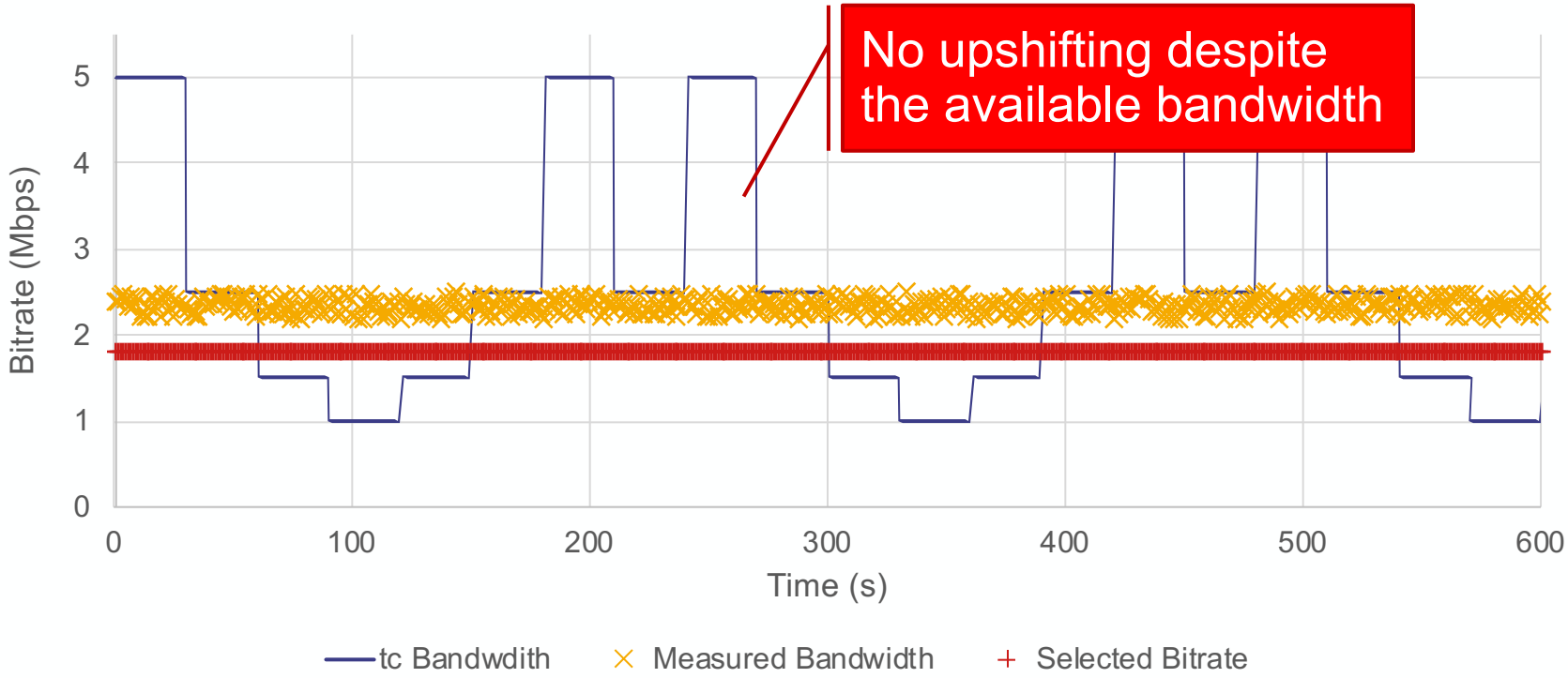
Ali C. Begen (on behalf My Colleagues)

http://ali.begen.net

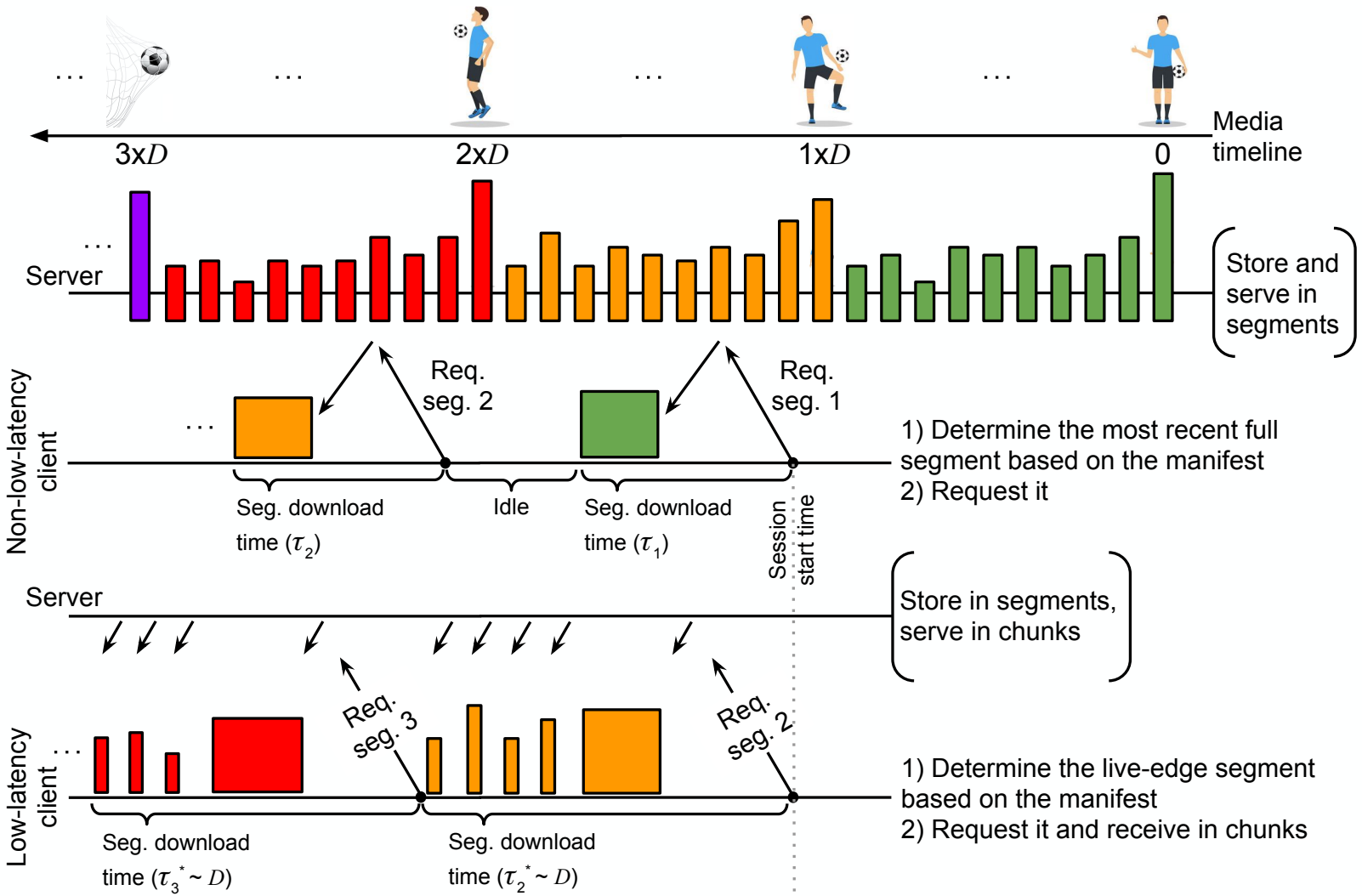
Published in ACM NOSSDAV 2019
DOI=10.1145/3304112.3325611

Bandwidth Measurement is Tricky

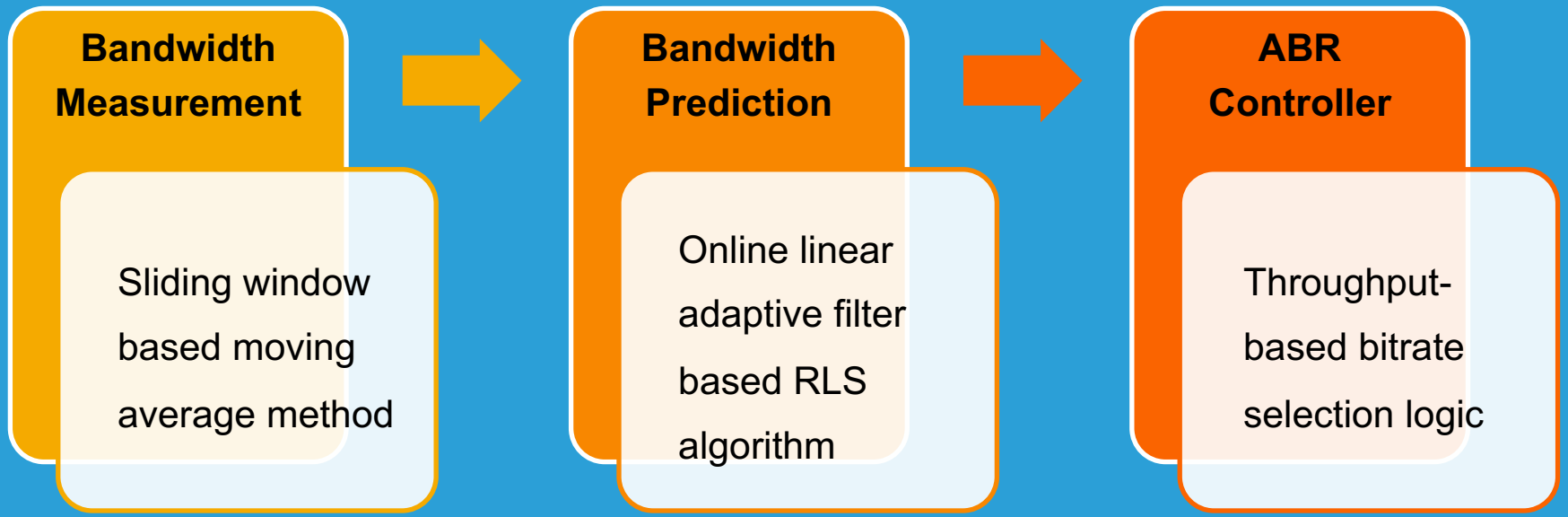
Live Twitch Data* (Nov. 2018)



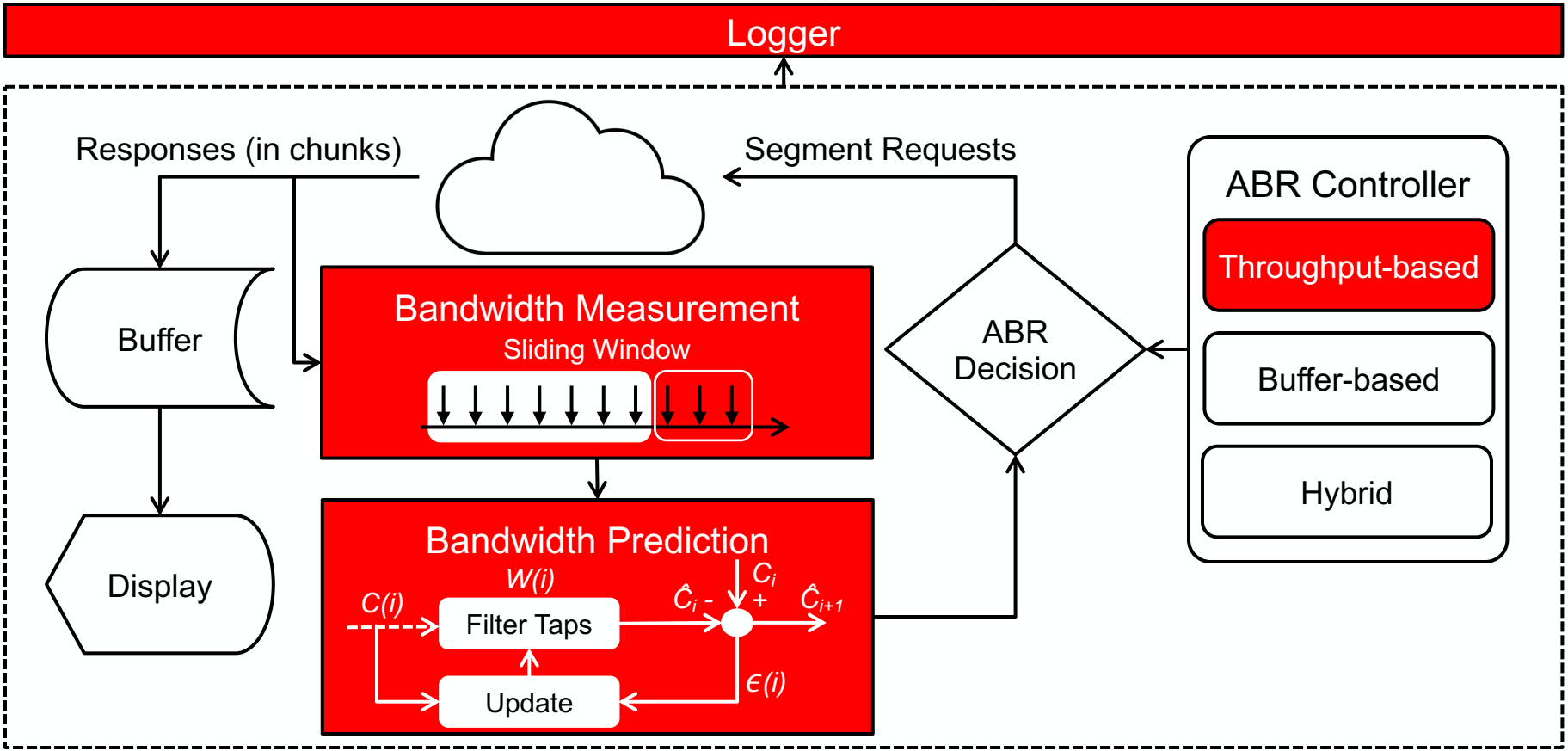
* Encoded at {0.18, 0.73, 1.83, 2.5, 3.1, 8.8} Mbps with three resolutions of {540p, 720p, 1080p}, and packaged with CMAF


Media timeline
3xD
2xD
1xD
0
Server
Store and serve in segments
Req. seg. 2
Req. seg. 1
Non-low-latency client
Seg. download time (τ_2)
Idle
Seg. download time (τ_1)
Session start time
1) Determine the most recent full segment based on the manifest
2) Request it
Server
Store in segments, serve in chunks
Req. seg. 3
Req. seg. 2
Low-latency client
Seg. download time ($\tau_3^* \sim D$)
Seg. download time ($\tau_2^* \sim D$)
1) Determine the live-edge segment based on the manifest
2) Request it and receive in chunks

ABR for Chunked Transfer Encoding (ACTE)

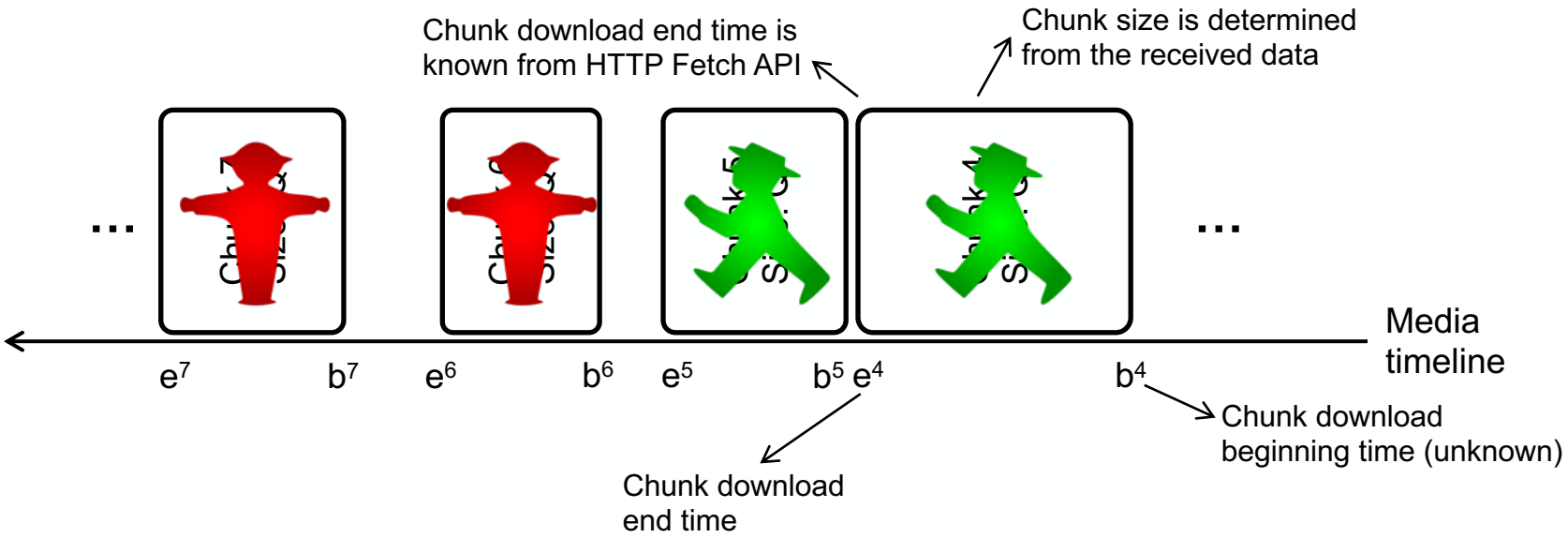


New/Modified Blocks in dash.js (in Red)



Bandwidth Measurement

Identifying the “Good” Chunks



- Compute the download rate for the chunks where the transmission is network limited
  - If there is a negligible idle period after a chunk download, use that chunk, otherwise disregard it

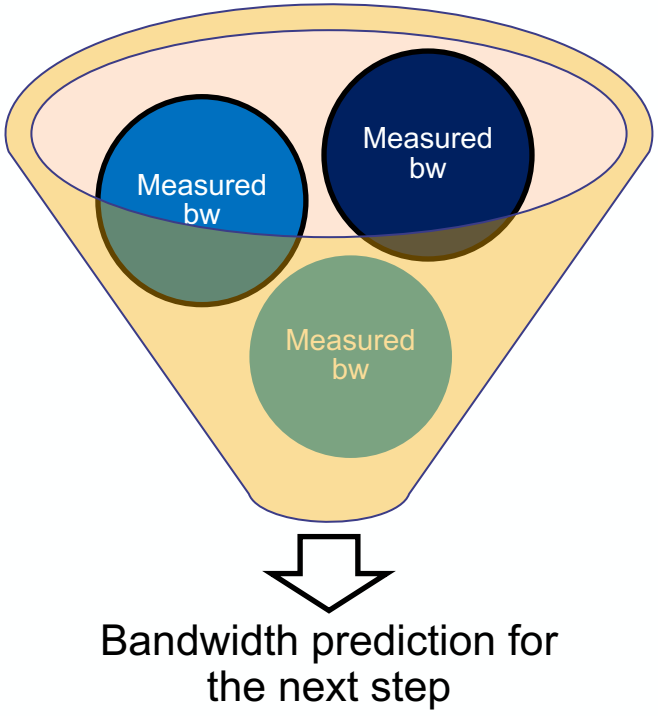
Bandwidth Measurement

Identifying the “Good” Chunks without the b^n Values

- Reasonable assumption: Idle periods cannot happen within a chunk, happen only between the chunks
  - Since the server pushes the chunks at full network speed
- For each chunk, compute its download rate, which equals its size divided by this chunk’s end time minus previous chunk’s end time
  - If this download rate is close (+/- 20%) to the average segment download rate, there must be significant idle time between these two chunks
    - Transmission is source limited
    - Disregard the current chunk
  - Else, the idle time is negligible
    - Transmission is network limited
    - The current chunk’s download rate is a good approximation of the available bandwidth
- Use a sliding window based moving average method over the last three successful chunk downloads

(Future) Bandwidth Prediction

Online Linear Adaptive Filter Using Recursive Least Squares (RLS)



ABR Controller

Throughput-Based Bitrate Selection Logic

- Find the best bitrate to pick to
  - Minimize the estimated error
  - Maximize QoE
- While respecting
  - Target latency
  - Network capacity
  - Buffer occupancy level

$$\mathcal{F}: \begin{cases} \textbf{find } r_i^\star, \arg\min \epsilon_i \text{ and } \arg\max \text{QoE}_i & \forall i > 0, \\ \text{s.t.} \quad l_i \leq l_{target}, \\ \qquad c_i \approx y_i, \text{ and } r_i^\star \approx \hat{c}_{i+1} \\ \qquad 0.5 \leq B_i \leq l_{target}, \end{cases}$$

Setup for Performance Evaluation

Experimental Setup

- Two machines, one running the modified dash.js player, one acting as a bandwidth shaping proxy
  - tc-NetEm to shape the network capacity according to DASH-IF's bandwidth variation profiles
  - iPerf to generate random TCP-based cross traffic ranging from 0.5 to 2 Mbps
- Origin and edge servers from Cloudflare with CMAF packaging and delivery enabled
- Content and Player Settings
  - Tears of Steel: https://mango.blender.org/download/
  - Segments of six seconds, chunks of 0.5 second
  - Video bitrate levels of {0.7, 1.3 and 2.0} Mbps
  - Min and max (target latency) buffer thresholds of 0.5 and 3.2 seconds, respectively

Setup for Performance Evaluation

Schemes Implemented

Bandwidth Measurement	ABR Schemes		
	Throughput-based	Buffer-based	Hybrid
SLBW	TH_{sl}	-	-
EWMA	TH_{ew}	-	-
SWMA	TH_{sw}	$BOLA_{sw}$	$Dynamic_{sw}$
WSSL	TH_{wss}	-	-

SLBW: Segment-based last bandwidth

EWMA: Chunk-based exponentially weighted moving average

SWMA: Chunk-based sliding window moving average

WSSL: Will's simple slide-load

Performance Metrics

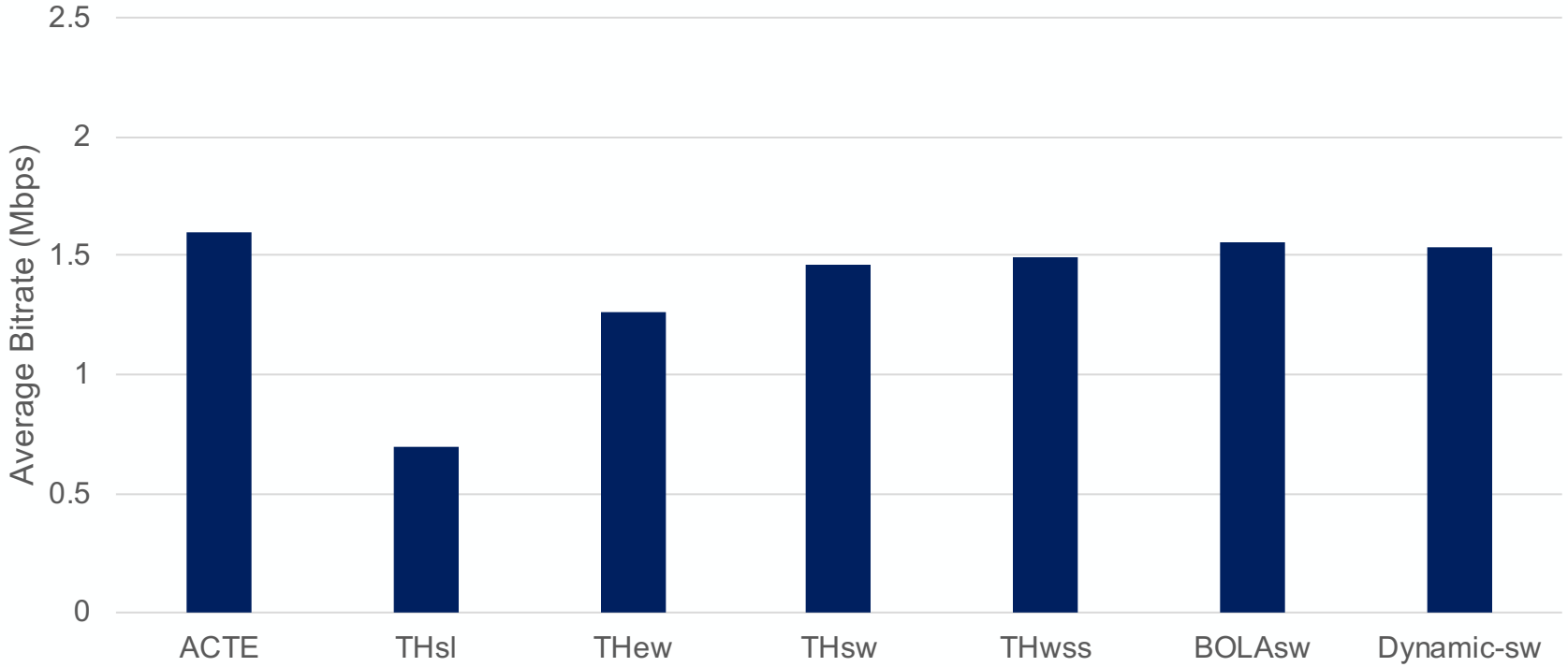
- Live latency
  - dash.js' live latency function (not including the encoding time)
- QoE models:
  - Yin QoE

$$\underbrace{\sum_{i=1}^{K} q(r_i)}_{\text{Average Bitrate}} - \underbrace{\delta_1 \sum_{i=1}^{K-1} \left| q(r_{i+1}) - q(r_i) \right|}_{\text{Bitrate Switches}} - \underbrace{\delta_2 T_{stall}}_{\text{Stall Duration}} - \underbrace{\delta_3 T_{sd}}_{\text{Startup Delay}}$$

  - ITU-T Rec. P.1203 QoE (Mode 0): bitrate, stall duration, frame rate, and resolution.
- Bandwidth prediction accuracy
  - Root Mean Square Error (RMSE) to compute the differences between the measured and predicted bandwidth values

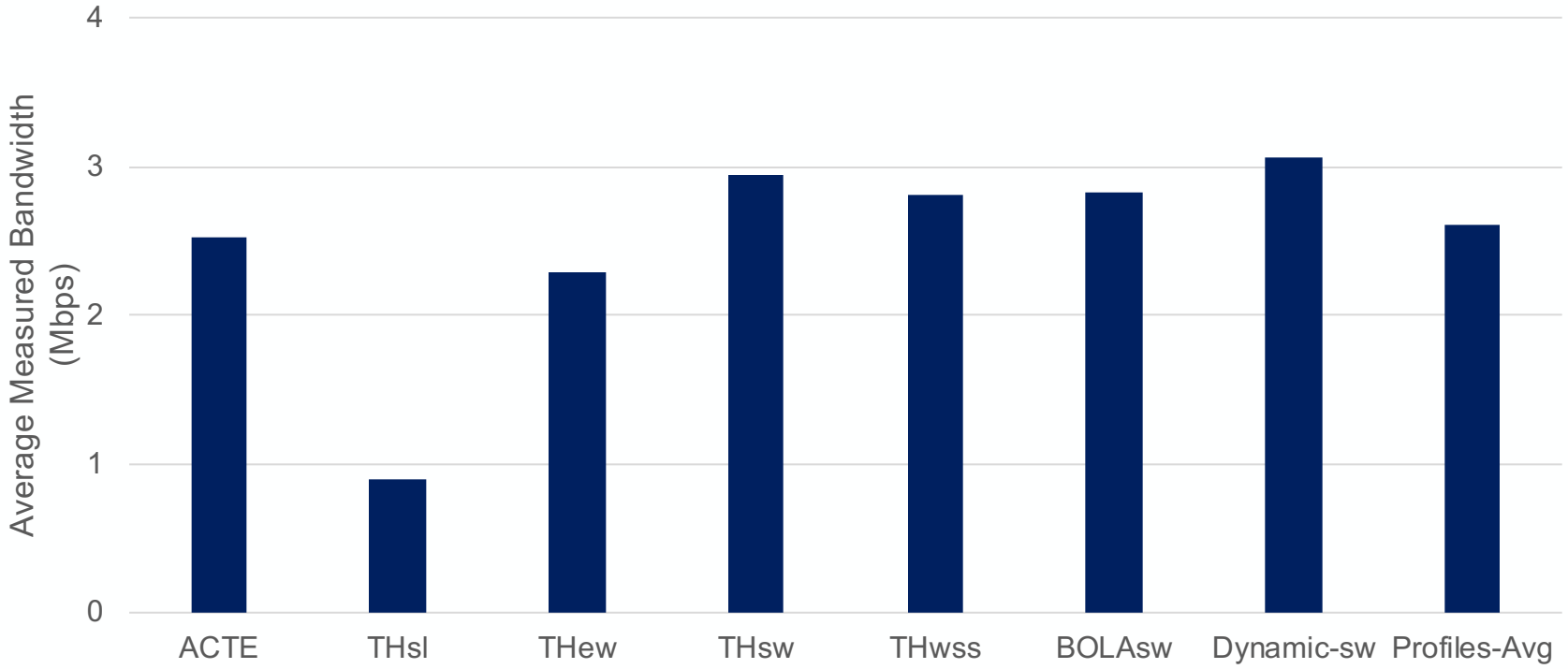
Average Selected Bitrate

28.6% Improvement by ACTE over Other Schemes



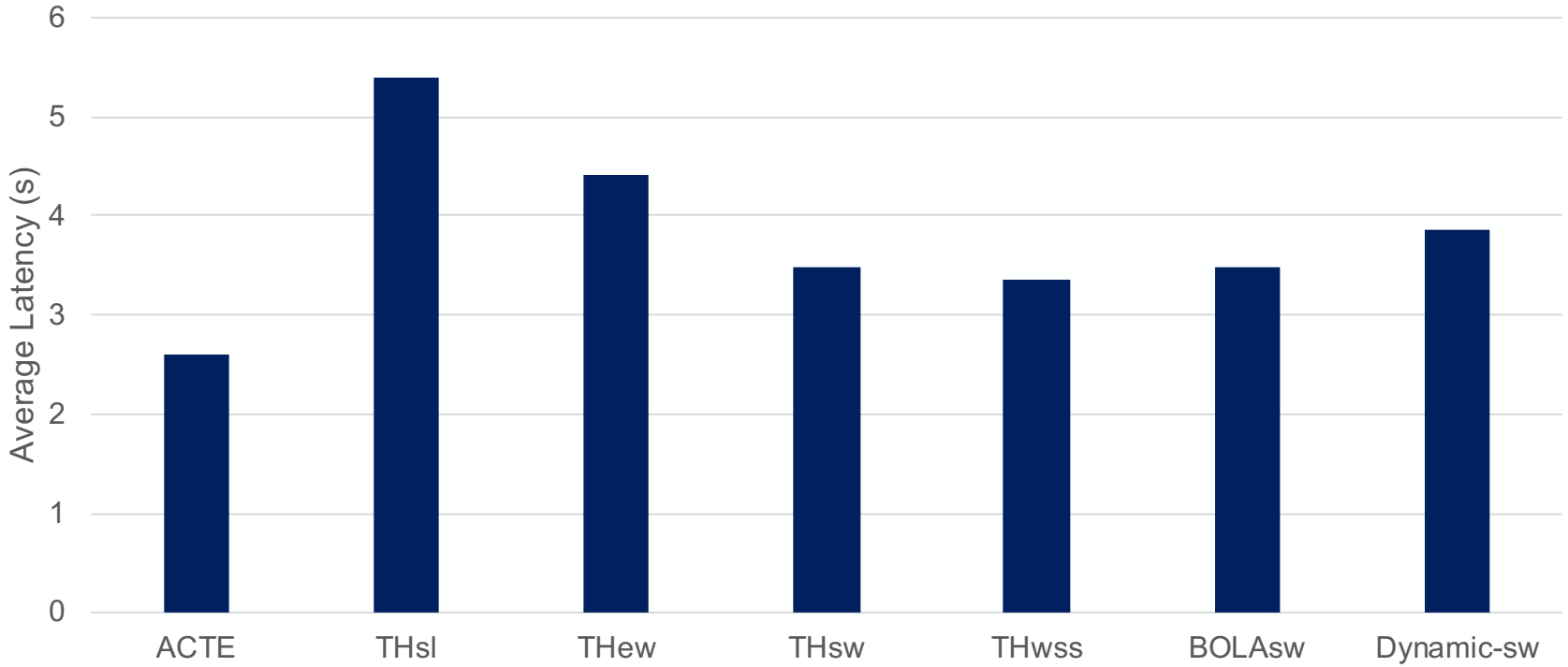
Average Measured Bandwidth

96.6% Prediction Accuracy by ACTE



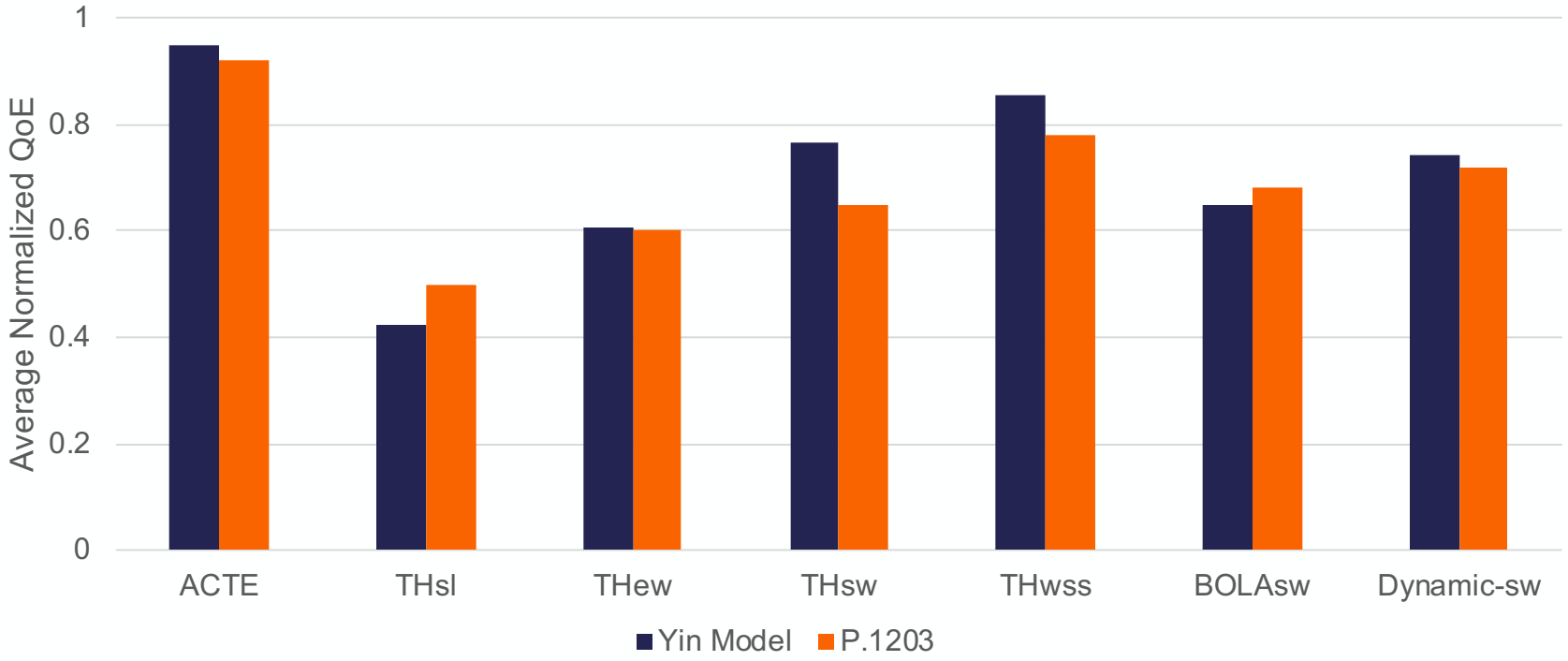
Average Live Latency

36.2% Improvement by ACTE over Other Schemes



Average Normalized QoE

49.3% Improvement by ACTE over Other Schemes



Overall Results

	Avg. Buffer Occupancy	Avg. # of Switches	Avg. # of Stalls and Duration (s)	Avg. Startup Delay (s)
ACTE	2.1 to 3.0 (2.5)	17	3 & 0.76	0.71
TH_{sl}	3.6 to 5.0 (4.3)	0	2 & 0.86	1.46
TH_{ew}	1.9 to 3.9 (2.9)	18	21 & 66	1.06
TH_{sw}	1.9 to 3.5 (2.8)	24	27 & 33	1.03
TH_{wss}	2.0 to 3.1 (2.5)	23	16 & 9	0.88
$BOLA_{sw}$	1.6 to 3.0 (2.3)	20	58 & 119	1.66
$Dynamic_{sw}$	1.6 to 3.0 (2.4)	30	53 & 68	0.92

ACTE Outperforms the Existing ABR Schemes

- Consecutive numbers represent the results
  - Summary of the average results. Percentage improvements of ACTE's over the other scheme

ACTE vs.	TH_{sl}	TH_{ew}	TH_{sw}	TH_{wss}	$BOLA_{sw}$	$Dynamic_{sw}$	Average
Avg. Selected Bitrate	128.6	23.1	6.7	6.7	0	6.7	28.6
Avg. Live Latency	53.7	44.4	30.6	21.9	30.6	35.9	36.2
Avg. # of Switches	N/A	5.6	29.2	26.1	15.0	43.3	23.8
Avg. # of Stalls	-50.0	85.7	88.9	81.3	94.8	94.3	65.8
Avg. Stall Duration	11.6	98.8	97.7	91.6	99.4	98.9	83.0
Avg. Startup Delay	51.4	33.0	31.1	19.3	57.2	22.8	35.8
Avg. N-QoE	126.2	58.3	25.0	11.8	46.2	28.4	49.3