Control-Theoretic Approach for Dynamic Adaptive Video Streaming over HTTP

Xiaoqi Yin, Abhishek Jindal, Vyas Sekar, Bruno Sinopoli

Presenters: Seanna Zhang, Qianfan Zhang, Hongyu Wen, Yihan Wang

Mar 31, 2023

Quality of Experience (QoE)

- Increase of video streaming traffic to Internet
- Demand for better user experience
- Metrics
  - Startup delay
  - Duration of rebuffering
  - Average playback bitrate
  - Variability of bitrate delivery
  - Rendering quality

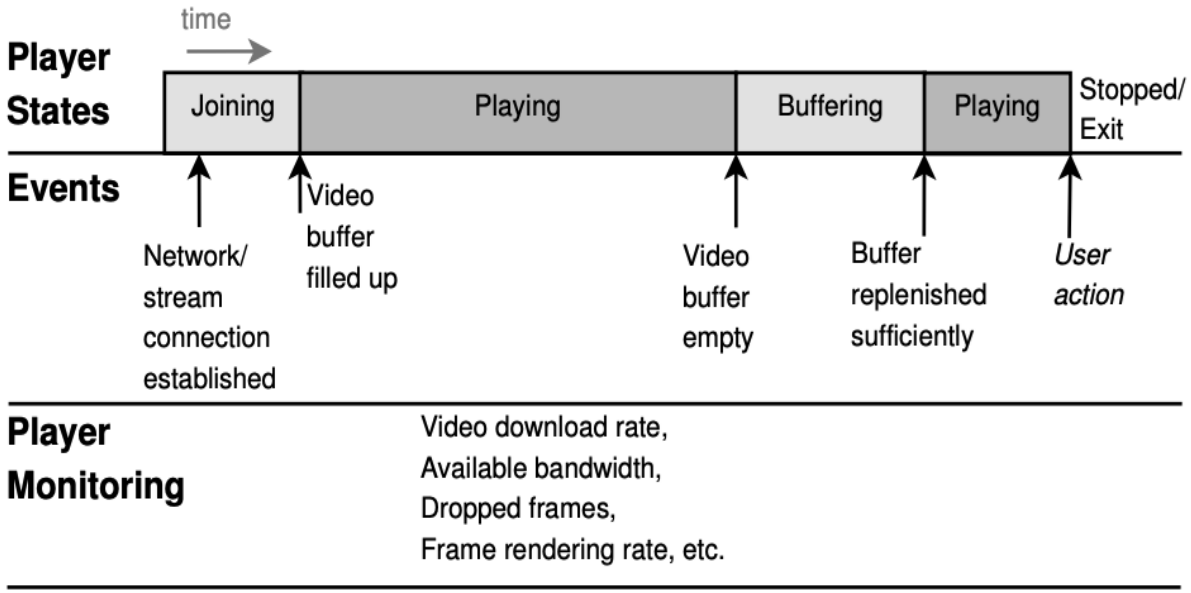


Image Credit: F. Dobrian, V. Sekar, A. Awan, I. Stoica, D. A. Joseph, A. Ganjam, J. Zhan, and H. Zhang. Understanding the Impact of Video Quality on User Engagement. In *Proc. ACM SIGCOMM*, 2011.

Adaptive Video Player

- HTTP-based delivery is predominated today
- Basic DASH architecture
  - Video is partitioned into chunks
  - Each chunk is encoded into different bitrate
  - Server sends manifest file with chunk information
  - Player uses some inputs to select bitrate level
  - Requested, downloaded, rendered

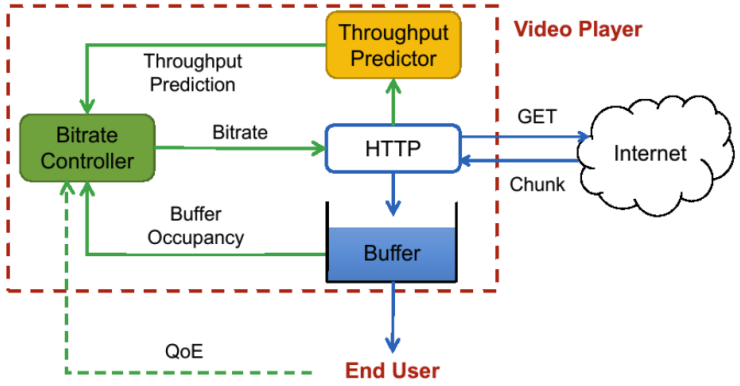


Figure 1: Abstract model of DASH players

Bitrate Adaptation Logic

- Motivation: Low QoE would lead user to leave video sessions → negative impact on revenue for content providers
- Bitrate adaptation logic is important to optimize user experience
  - Adaptive bitrate (ABR) algorithms
- Goal: to choose the bitrate level for future chunks to deliver the **highest** possible QoE

Challenge: Many Conflicting QoE Consideration

- Minimizing rebuffering events
- Delivering a high playback bitrate
- Minimizing startup delay
- Keeping playback smooth

Extreme solutions

- Always pick lowest bitrate
  - To minimize rebuffering events and startup delay
  - Conflict with delivering high bitrate
- Always select highest bitrate
  - Lead to many rebuffering events
- Goal of maintaining smooth playback
  - Optimal choice may switch bitrates for every chunk to minimize rebuffering and maximize average bitrate

Classes of Algorithms

- Rate-based algorithms
  - Estimate the available network throughput to decide
  - The estimation is easily to be impacted by several factors, leading inaccurate estimation
    - Network bandwidth fluctuation
- Buffer-based algorithms
  - Relies on buffer occupancy as metric to select
  - More stable to handle network fluctuations
  - Ignoring any available throughput information
- Each approach is isolation and work under specific environmental assumptions
- Need new approach to combine these two signals

Paper's Solutions/Contributions

- Client-side adaptation
  - Offer most immediately deployable alternative compare to solutions
    - Requires in-network support
    - Changes in server-side software
    - Modifies lower-layer transport protocols
  - Is best position to detect performance issue quickly and respond to dynamics
- Formulate bitrate adaptation as a stochastic optimal control problem
- Proposed Model Predictive Control (MPC) approach to solve optimization problem
- Designed a practical and fast table enumeration based algorithm, FastMPC
- Deployed a low-overhead implementation based on the open source reference video player dash.js
- Validated approach with different classes of algorithms using realistic trace-driven emulations

Control-Theoretic Model and MPC

- Previous works
- The video streaming model
- Optimization goal: QoE
- Class of algorithms
- Model predictive control

Previous works

- Lack of a uniformed evaluation
  - {bitrate, smoothness, rebuffer rate, startup delay}
  - Compared separately
- Lack of a general design space
  - Solely based on throughput estimation
  - Solely based on buffer occupancy

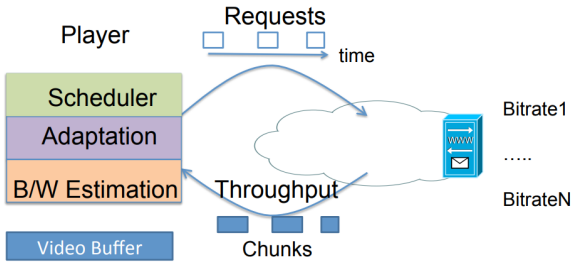
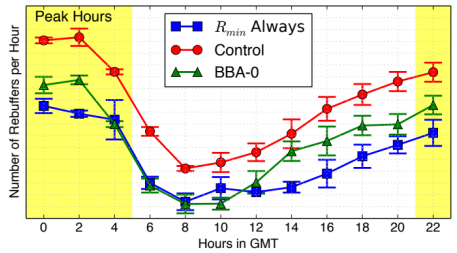
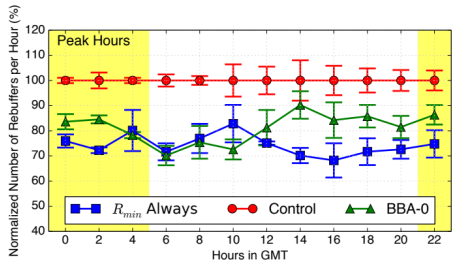


Figure 4: General framework of HTTP adaptive video streaming. The server supports multiple bitrate encodings, each a separate logically chunked file. The player issues GET requests for each chunk at a specific bitrate and adapts the bitrate based on the observed throughput.

J. Jiang, V. Sekar, and H. Zhang. Improving Fairness, Efficiency, and Stability in HTTP-based Adaptive Video Streaming with FESTIVE. In Proc. CoNext, 2012.



(a) Number of rebuffers per playhour.



(b) Normalized number of rebuffers per playhour.

Figure 6: Number of rebuffers per playhour for the *Control*, $R_{\min}$ *Always*, and BBA-0 algorithms.

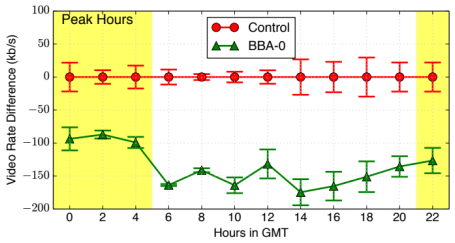


Figure 7: Comparison of video rate between *Control* and BBA-0.

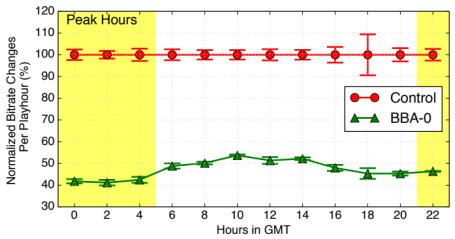


Figure 8: Average video switching rate per two hour window for the *Control* and *BBA-0* algorithms.

T.-Y. Huang, R. Johari, N. McKeown, M. Trunnell, and M. Watson. A Buffer-Based Approach to Rate Adaptation: Evidence from a Large Video Streaming Service. In Proc. ACM SIGCOMM, 2014.

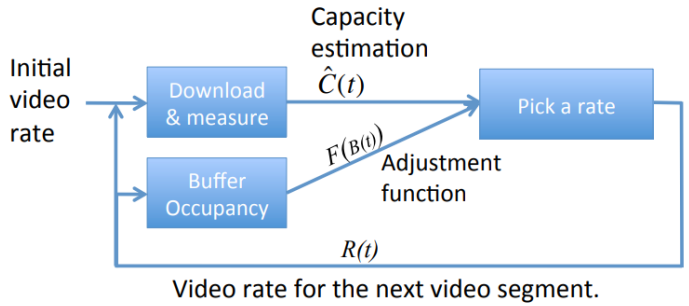
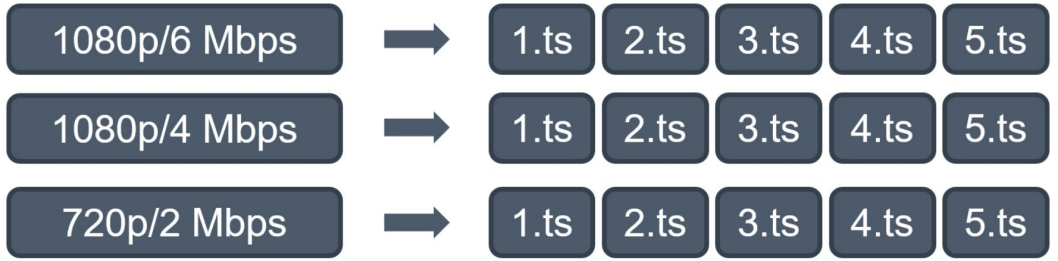


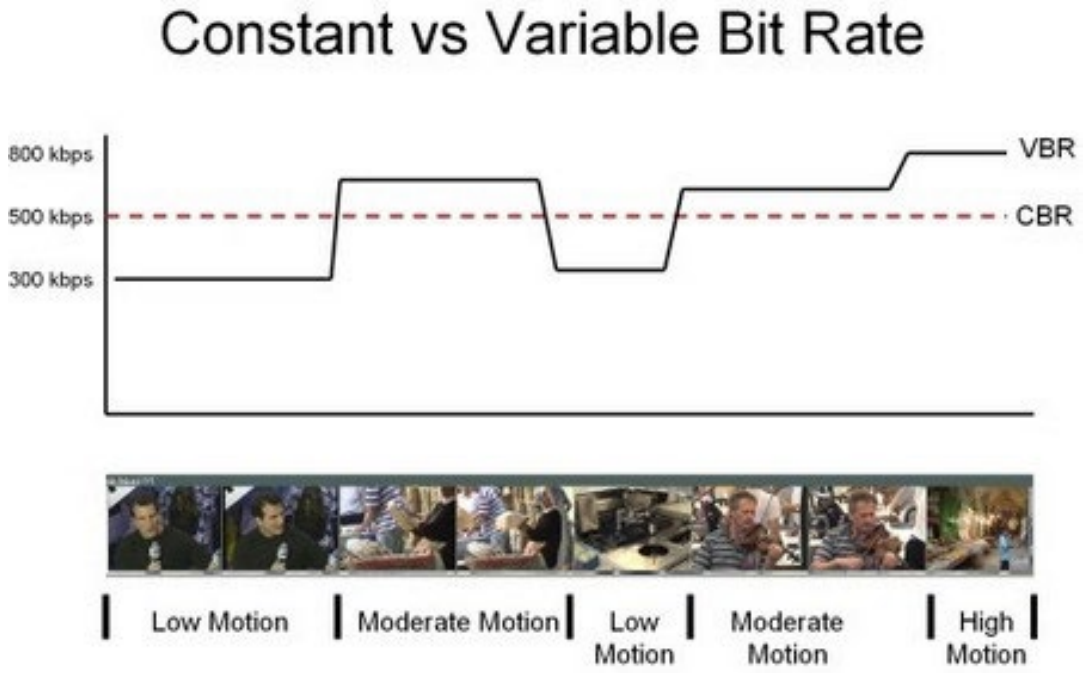
Figure 3: Current practice adjusts the estimation based on the buffer occupancy.

T.-Y. Huang, R. Johari, N. McKeown, M. Trunnell, and M. Watson. A Buffer-Based Approach to Rate Adaptation: Evidence from a Large Video Streaming Service. In Proc. ACM SIGCOMM, 2014.

The video streaming model: server-side

- Video stored as K consecutive *chunks* $1,2,\ldots,K$, each
  - Contains L seconds of video (e.g., $L = 4$ seconds)
  - Available in different bitrates (e.g., $\{350, 600, 1000\ 2000, 3000\}$ Kbps)
  - Size of a chunk $d_k(R_k) = L \cdot R_k$ if assuming constant bitrate encoding
    - For variable bitrate encoding, still have $d_k(R_k) \sim R_k$, but can differ across chunks





https://www.videoconverterfactory.com/tips/imgs-self/vbr-vs-cbr/vbr-vs-cbr-1.jpg

The video streaming model: client-side

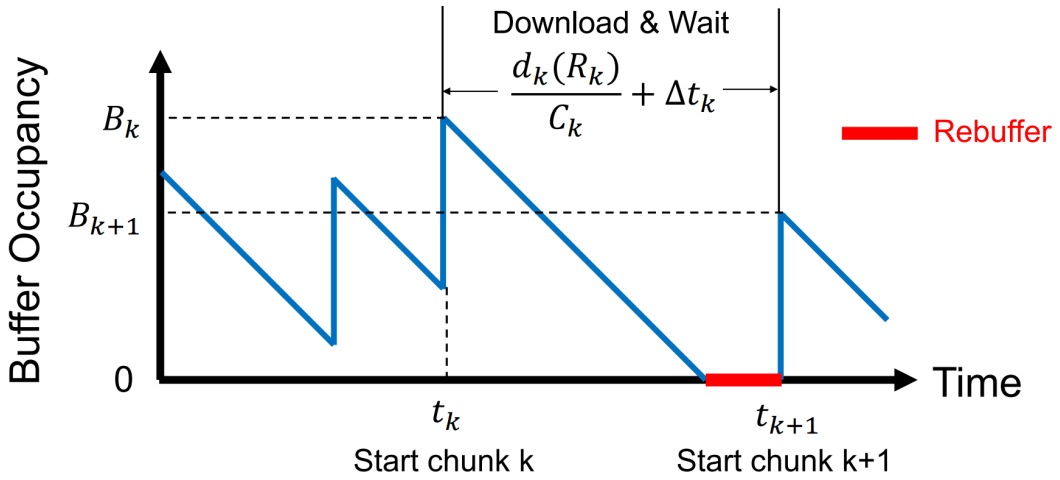
- Video chunks are requested and downloaded into a *playback buffer*
  - Contains downloaded yet unviewed video
  - Maximum buffer size (in seconds): B_{max} ($B_{max} = 30$ seconds in the paper)
- The download time for chunk k is $\frac{d_k(R_k)}{C_k}$
  - $d_k(R_k)$ is the size of chunk
  - C_k is the average throughput when downloading the chunk
- Decision: which bitrate R_k to choose for the next chunk

Discussion:

What are pros/cons for **client-side** vs **server-side** decision for adaptive bitrate?

The video streaming model: dynamics

- Given
  - L: chunk length
  - B_k: buffer size when starting download k
  - R_k: selected bitrate for k
  - $d_k(R_k)$: size of k ($L \cdot R_k$ if assume CBR encoding)
  - C_k: average throughput when downloading k
  - Δt_k: extra waiting time before download $k+1$
    - In the paper, $\Delta t_k = \left(\left(B_k - \frac{d_k(R_k)}{C_k}\right)_+ + L - B_{max}\right)_+$



- We have the buffer size when starting download $k+1$
  - $B_{k+1} = \left(\left(B_k - \frac{d_k(R_k)}{C_k}\right)_+ + L - \Delta t_k\right)_+$
- *Rebuffer event* happens when $B_k < \frac{d_k(R_k)}{C_k}$ with rebuffer time $\left(\frac{d_k(R_k)}{C_k} - B_k\right)_+$

Multiple objectives

- Select bitrate as high as possible
  - Maximize average video quality $\frac{1}{K}\sum_{k=1}^{K} q(R_k)$ for some function $q(\cdot)$ that maps a bitrate to some score
- Avoid frequent or large bitrate jumps
  - Minimize quality variations $\sum_{k=1}^{K-1} |q(R_{k+1}) - q(R_k)|$
- Avoid rebuffer events
  - Minimize total rebuffer time $\sum_{k=1}^{K} \left(\frac{d_k(R_k)}{C_k} - B_k\right)_+$
- Minimize startup delay T_s
  - Note that authors defined T_s as the "initial buffer occupancy"

Objectives combined: QoE

$$QoE = \frac{1}{K}\sum_{k=1}^{K} q(R_k) - \lambda \sum_{k=1}^{K-1} |q(R_{k+1}) - q(R_k)| - \mu \sum_{k=1}^{K} \left(\frac{d_k(R_k)}{C_k} - B_k\right)_+ - \mu_s T_s$$

- Parameters to choose:
  - Weights λ, μ, μ_s
  - Quality function for bitrate: $q(\cdot)$

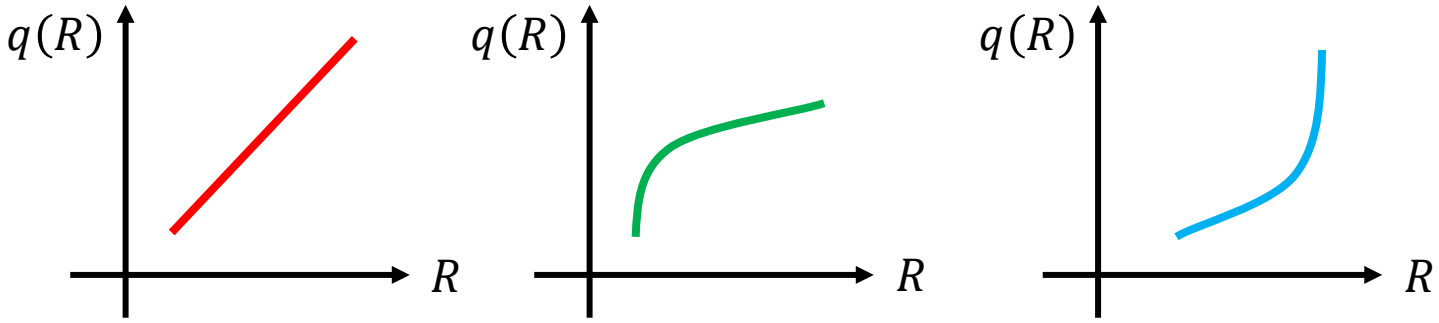
QoE: Choice of function q

$$QoE = \frac{1}{K}\sum_{k=1}^{K} q(R_k) - \lambda \sum_{k=1}^{K-1} |q(R_{k+1}) - q(R_k)| - \mu \sum_{k=1}^{K} \left(\frac{d_k(R_k)}{C_k} - B_k\right)_+ - \mu_s T_s$$

- In the paper, q is chosen to be identity, i.e., $q(R) = R$

Discussion:

Consider the following three functions. Which makes more sense to you? Or, in what scenario would you choose each one?



QoE: Choice of parameters λ, μ, μ_s

$$QoE = \frac{1}{K}\sum_{k=1}^{K} q(R_k) - \lambda \sum_{k=1}^{K-1} |q(R_{k+1}) - q(R_k)| - \mu \sum_{k=1}^{K} \left(\frac{d_k(R_k)}{C_k} - B_k\right)_+ - \mu_s T_s$$

- In this paper
  - "Balanced": $\lambda = 1, \mu = \mu_s = 3000$
    - 1 second rebuffer/start-up delay = reducing the average bitrate by 3000 Kbps
  - "Avoid instability": $\lambda = 3, \mu = \mu_s = 3000$
  - "Avoid rebuffering": $\lambda = 1, \mu = \mu_s = 6000$

The QoE maximization problem

$$QoE_1^K = \frac{1}{K}\sum_{k=1}^{K} q(R_k) - \lambda \sum_{k=1}^{K-1} |q(R_{k+1}) - q(R_k)| - \mu \sum_{k=1}^{K} \left(\frac{d_k(R_k)}{C_k} - B_k\right)_+ - \mu_s T_s$$

$$\max_{R_1,\cdots,R_K,T_s} \quad QoE_1^K \tag{6}$$

$$s.t. \quad t_{k+1} = t_k + \frac{d_k(R_k)}{C_k} + \Delta t_k, \tag{7}$$

$$C_k = \frac{1}{t_{k+1} - t_k - \Delta t_k} \int_{t_k}^{t_{k+1}-\Delta t_k} C_t \, dt, \tag{8}$$

$$B_{k+1} = \left(\left(B_k - \frac{d_k(R_k)}{C_k}\right)_+ + L - \Delta t_k\right)_+, \tag{9}$$

$$B_1 = T_s, \quad B_k \in [0, B_{max}] \tag{10}$$

$$R_k \in \mathcal{R}, \quad \forall k = 1, \cdots, K. \tag{11}$$

- Parameters:
  - $\lambda, \mu, \mu_s, q(\cdot)$
- Input:
  - Chunk length L and size $d_k(\cdot)$
  - Buffer capacity B_{max}
  - Throughput trace $\{C_t\}$
- Output:
  - Bitrate decisions $R_1, \ldots, R_K$
  - Startup delay T_s
  - Waiting time Δt_k (not used in this paper)

The **online** QoE maximization problem

$$QoE_1^K = \frac{1}{K}\sum_{k=1}^{K} q(R_k) - \lambda \sum_{k=1}^{K-1} |q(R_{k+1}) - q(R_k)| - \mu \sum_{k=1}^{K} \left(\frac{d_k(R_k)}{C_k} - B_k\right)_+ - \mu_s T_s$$

$$\max_{R_1,\cdots,R_K,T_s} \quad QoE_1^K \tag{6}$$

$$s.t. \quad t_{k+1} = t_k + \frac{d_k(R_k)}{C_k} + \Delta t_k, \tag{7}$$

$$C_k = \frac{1}{t_{k+1} - t_k - \Delta t_k} \int_{t_k}^{t_{k+1}-\Delta t_k} C_t \, dt, \tag{8}$$

$$B_{k+1} = \left(\left(B_k - \frac{d_k(R_k)}{C_k}\right)_+ + L - \Delta t_k\right)_+, \tag{9}$$

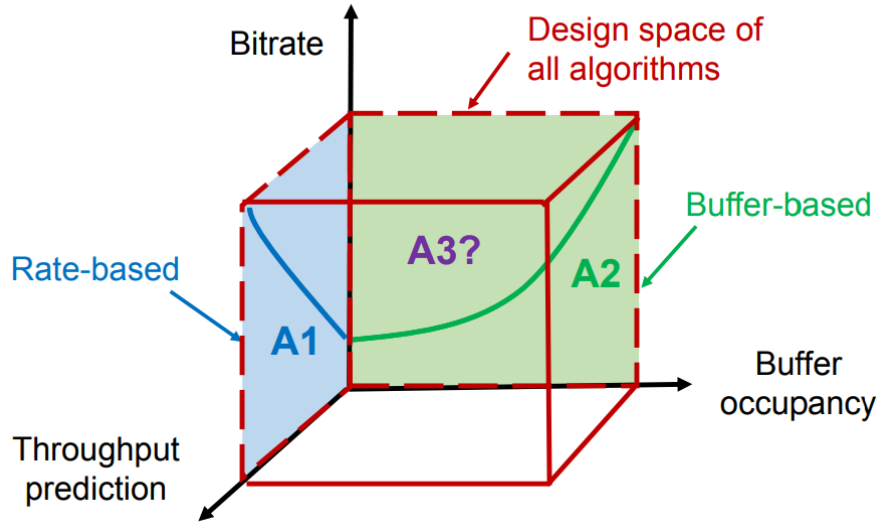
$$B_1 = T_s, \quad B_k \in [0, B_{max}] \tag{10}$$

$$R_k \in \mathcal{R}, \quad \forall k = 1, \cdots, K. \tag{11}$$

- Parameters:
  - $\lambda, \mu, \mu_s, q(\cdot)$
- Input:
  - Chunk length L and size $d_k(\cdot)$
  - Buffer capacity B_{max}
  - Throughput trace $\{C_t\}$ **Unknown in advance!**
- Output:
  - Bitrate decisions $R_1, \ldots, R_K$
  - Startup delay T_s
  - Waiting time Δt_k (not used in this paper)

Class of algorithms

- *Rate-based*: bitrate as a function of throughput prediction
  - $R_k = f\left(\{\hat{C}_t\}_{t>t_k}, \{R_i\}_{i<k}\right)$ given predictions $\{\hat{C}_t\}_{t>t_k}$ obtained by a *throughput predictor*
- *Buffer-based*: bitrate as a function of buffer occupancy
  - $R_k = f(B_k, \{R_i\}_{i<k})$
- Combined:
  - $R_k = f\left(B_k, \{\hat{C}_t\}_{t>t_k}, \{R_i\}_{i<k}\right)$
- This paper: focus on f rather than throughput predictor



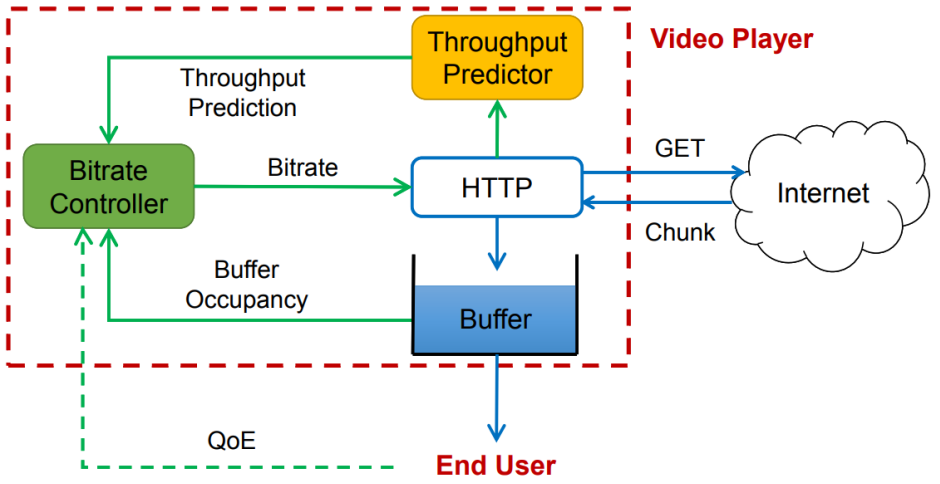


Figure 1: Abstract model of DASH players

Candidates

- PID (Proportional-integral-derivative) control:
  - Used to stabilize the system, cannot directly optimize QoE
  - Designed to work in continuous time and state space
- MDP-based (Markov decision process) control:
  - Must assume throughput dynamics follows Markov process
    - i.e., the throughput at step $k+1$ depends only on the throughput at step k.
  - Include last k throughput in the state?
    - Too many states!
    - Possible solution: Deep reinforcement learning

MPC (Model Predictive Control)

- Assumption: can predict throughput in short timescales
  - Not always true
- When deciding the bitrate R_k for chunk k:
  1. Predict future throughput $\{\hat{C}_t\}_{t\in[t_k,t_k+N]}$
  2. Optimize QoE_k^{k+N-1} to obtain R_k
  3. Download chunk k with bitrate R_k

Algorithm 1 Video adaptation workflow using MPC

1: Initialize
2: **for** $k = 1$ to K **do**
3: **if** player is in startup phase **then**
4: $\hat{C}_{[t_k,t_{k+N}]}$= ThroughputPred($C_{[t_1,t_k]}$)
5: $[R_k, T_s] = f_{mpc}^{st}\left(R_{k-1}, B_k, \hat{C}_{[t_k,t_{k+N}]}\right)$
6: Start playback after T_s seconds
7: **else if** playback has started **then**
8: $\hat{C}_{[t_k,t_{k+N}]}$= ThroughputPred($C_{[t_1,t_k]}$)
9: $R_k = f_{mpc}\left(R_{k-1}, B_k, \hat{C}_{[t_k,t_{k+N}]}\right)$
10: **end if**
11: Download chunk k with bitrate R_k, wait till finished
12: **end for**

Inaccurate throughput predictions

- What if the throughput predictor consistently make mistakes?
  - E.g., when Internet is congested
- RobustMPC: optimizes worst-case QoE given throughput in $\left[\underline{\hat{C}_t}, \overline{\hat{C}_t}\right]$

$$\max_{R_k,\cdots,R_{k+N-1}} \min_{C_t \in [\underline{\hat{C}_t}, \overline{\hat{C}_t}]} QoE_k^{k+N-1} \qquad (15)$$

$$s.t. \quad \text{Constraints (7) to (11)} \qquad (16)$$

THEOREM 1. *The robust MPC controller is equivalent to the regular MPC taking the lower bound of throughput as input, namely,*

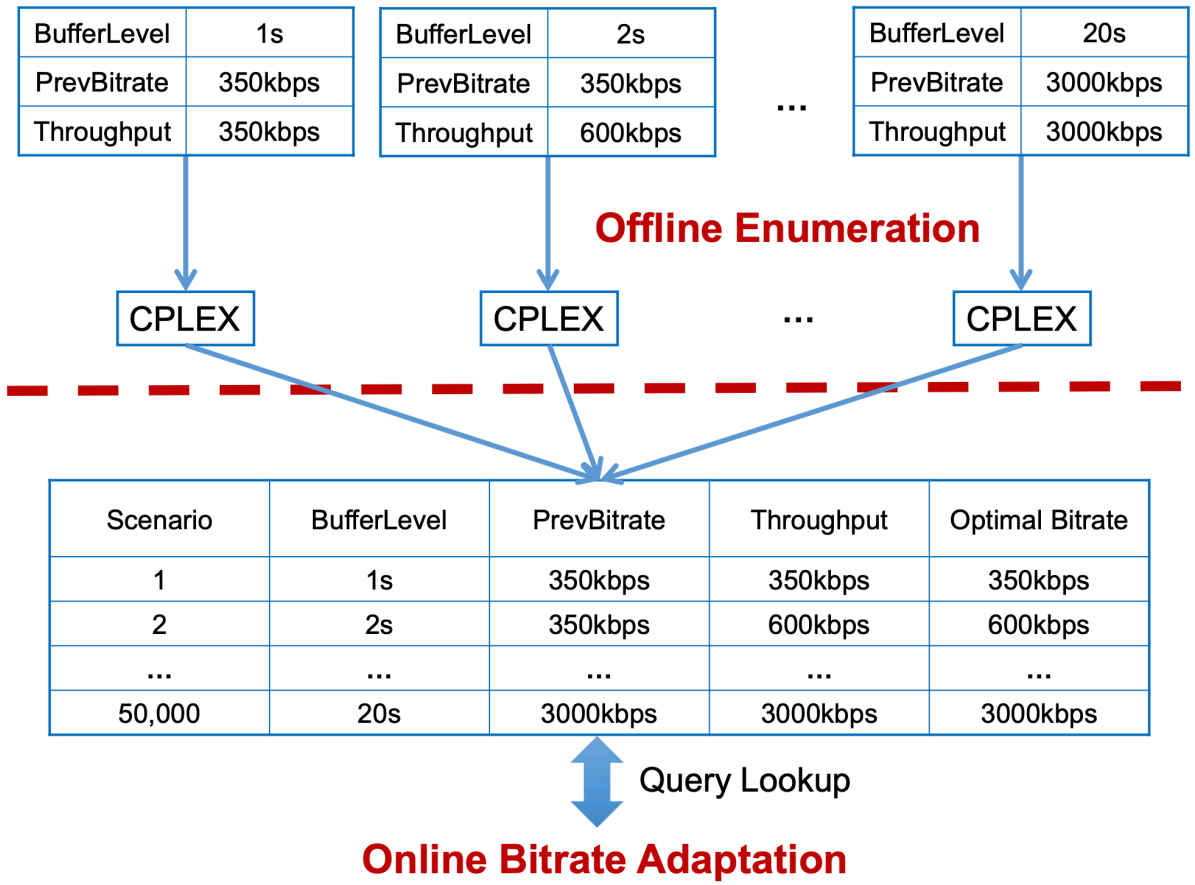
$$\begin{aligned} R_k &= f_{robustmpc}(R_{k-1}, B_k, [\underline{\hat{C}_t}, \overline{\hat{C}_t}]) \\ &= f_{mpc}(R_{k-1}, B_k, \underline{\hat{C}_t}) \end{aligned}$$

Practical Concerns

- Computational Overhead
  - ILP problem may take seconds to be solved
- Deployment
  - it may not be possible for video players to be bundled with the solver (CPlex or Gorubi)
- Don’t solve a ILP online

FastMPC Idea

- **Offline**: Enumerate the state-space and solve each specific instance
- **Online**: Map stored optimal control decisions to current operation conditions



FastMPC State-space

- R_{k-1}: Previous bitrates chosen
- B_k: Current buffer level
- $\hat{C}_{[t_k,t_{k+N}]}$: the predicted throughput for the next N chunks

Algorithm 1 Video adaptation workflow using MPC

1: Initialize
2: **for** $k = 1$ to K **do**
3: **if** player is in startup phase **then**
4: $\hat{C}_{[t_k,t_{k+N}]}$= ThroughputPred($C_{[t_1,t_k]}$)
5: $[R_k, T_s] = f^{st}_{mpc}\left(R_{k-1}, B_k, \hat{C}_{[t_k,t_{k+N}]}\right)$
6: Start playback after T_s seconds
7: **else if** playback has started **then**
8: $\hat{C}_{[t_k,t_{k+N}]}$= ThroughputPred($C_{[t_1,t_k]}$)
9: $R_k = f_{mpc}\left(R_{k-1}, B_k, \hat{C}_{[t_k,t_{k+N}]}\right)$
10: **end if**
11: Download chunk k with bitrate R_k, wait till finished
12: **end for**

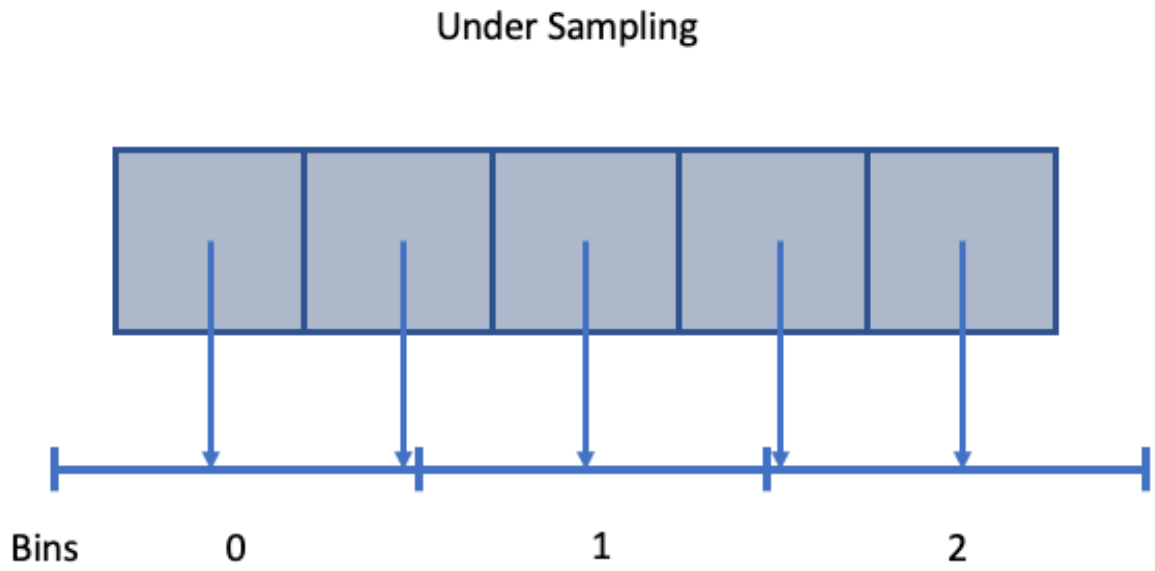
Practical Concerns

- Huge State Space
  - 100 possible values for the buffer level
  - 10 possible bitrates
  - A horizon of size 5 and 1000 possible throughput values
  - $100 \times 10 \times 1000^5 = 10^{18}$ rows in the table!

- Unacceptable memory cost and loading delay
- Non-trivial offline computation cost

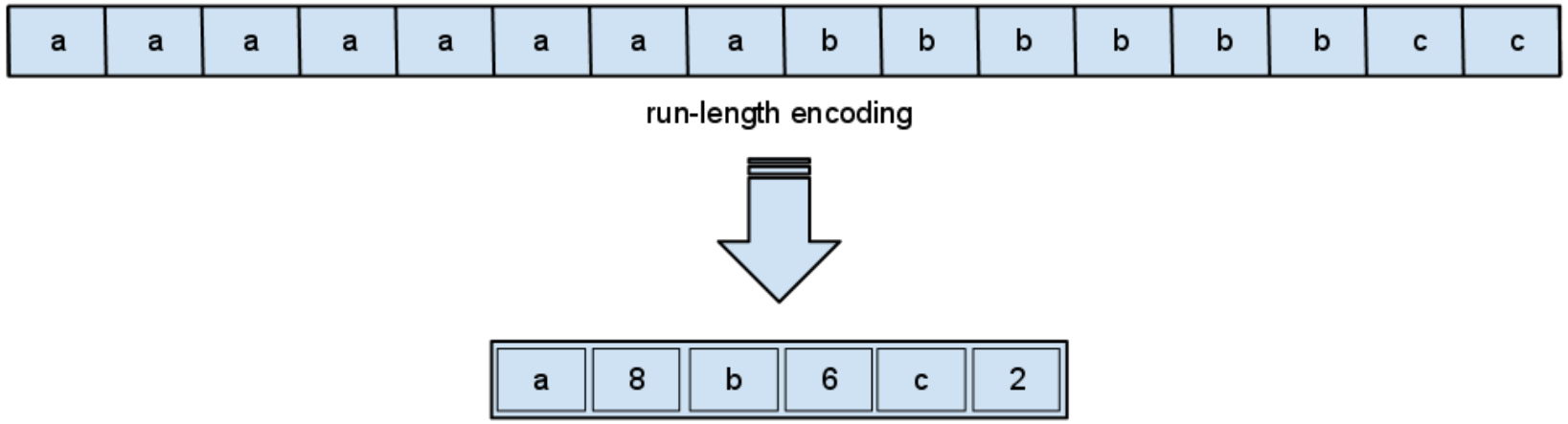
Optimizing FastMPC Performance

- Grouping values into bins
- Use 100 bins for 1000 possible throughput values (0 ~ 999)
  - (111, 278, 578, 430, 869) -> (115, 275, 575, 435, 865)
- Reduce the number of rows



Optimizing FastMPC Performance

- Run-length encoding
  - **runs of data** (sequences in which the same data value occurs in many consecutive data elements) are stored as a single data value and count
  - the optimal solutions for several similar scenarios will likely be the same
- Reduce memory cost of each row
- table occupies less than 60 kB after optimization

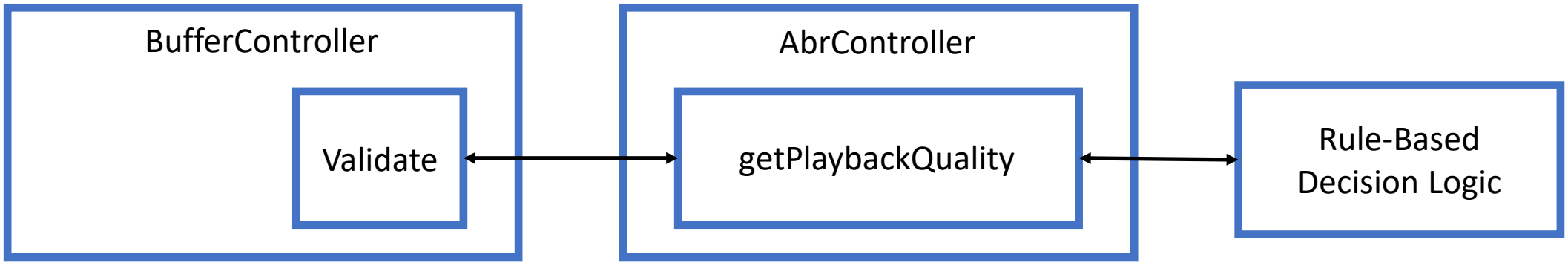


Implementation

- Adobe OSMF framework

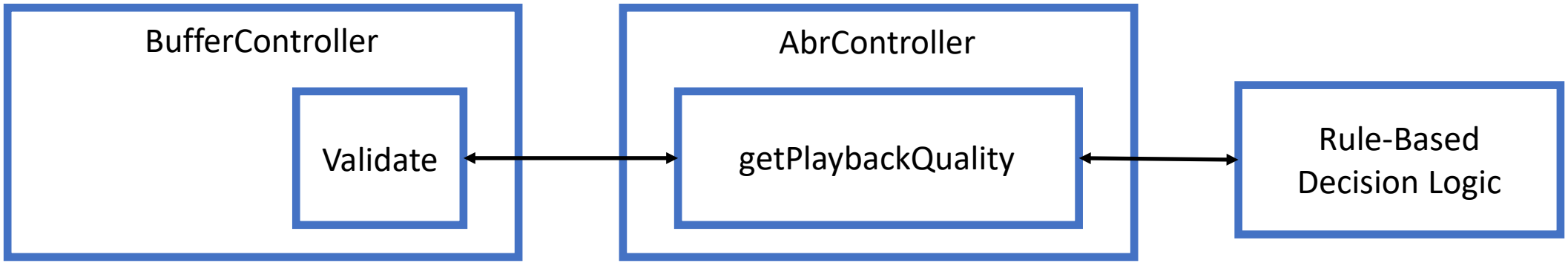
- HTML5-based players

dash.js Overview



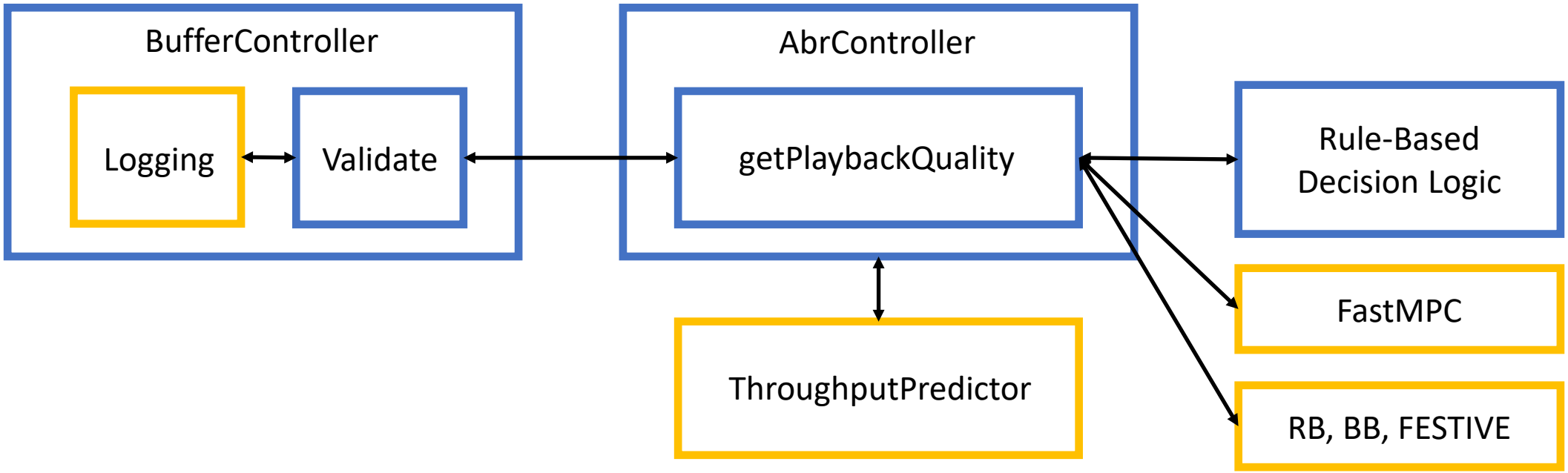
- **DownloadRatioRule** selects bitrate based on the “download ratio”
  - Download ratio: (play time of last chunk) / (its download time)
- **InsufficientBufferRule** chooses bitrate depending on whether the buffer level has reached a lower limit recently to avoid rebuffers

dash.js Modifications



- Periodically call Validate → always call it at the start of each chunk
- Chunk download in parallel → chunk download completely sequential

dash.js Extensions



- Harmonic mean of previous 20 samples

Evaluation

- Settings
  - Throughput variability traces
  - Video-specific parameters
  - Algorithm configurations
  - QoE parameters
- Performance & overhead analysis

Throughput Variability Traces (Dataset)

- Broadband Dataset (FCC)
  - 6 million data points, each contains the average throughput during a 5s interval
  - Concatenated to match the video length
  - Randomly pick 1000 traces, average throughputs from 0-3Mbps
- Mobile Dataset (HSDPA)
  - 30min of continuous 1s measurement of the avg throughput in the MOBILE environment
  - Randomly pick 1000 traces
- Synthetic Dataset
  - Random state S_t, which models the number of users sharing the link
  - Gaussian throughput based on S_t

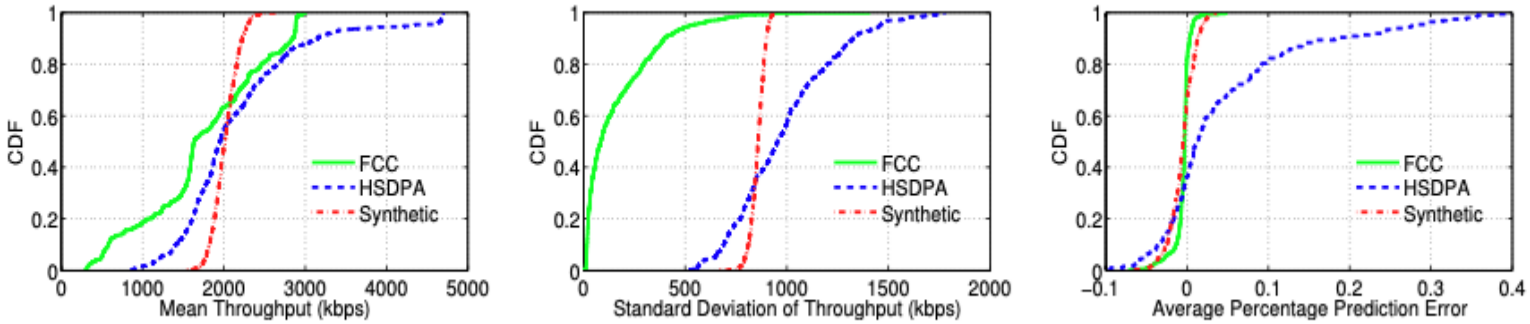


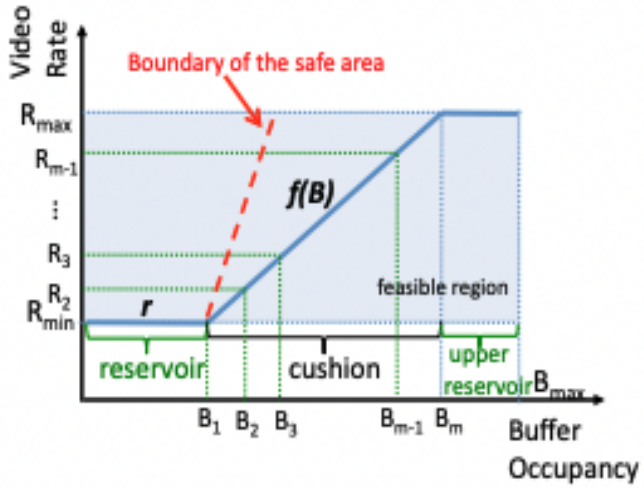
Figure 7: Characteristics of datasets

Video Parameters

- Envivio video
  - 260s: 65 * L=4 (chunk size)
  - Bitrate levels = {350, 600, 1000, 2000, 3000} kbps ~ {240p, 360p, 480p, 720p, 1080p}
  - $\mathrm{B}_{max} = 30s$
  - $Q(x) = Identity(x) = 30s$
  - QoE weights

Algorithms

- RB: R = [maximum available bitrate <= harmonic prediction (5 chunks)]
- BB: R = f(B)
  - cushion = 10s, r = 5s
- FastMPC:
  - Look-ahead h = 5
  - Past 5 chunks for rate prediction
    - MPC-OPT: perfect predictor
  - 100 bins for buffer levels
  - 100 bins for throughput prediction
- RobustMPC: $\underline{\hat{C}_t} = \frac{\hat{C}_t}{1+err}$
- dash.js: a rule-based algorithm
- FESTIVE: a rate-based algorithm



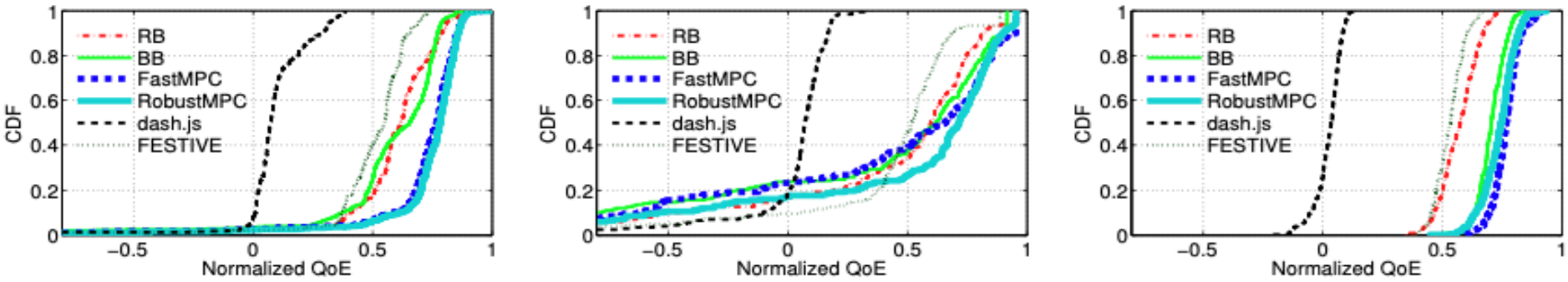
BB algorithm baseline
Huang et al. SIGCOMM2014

Metric

- Normalized QoE:
  - QoE(OPT) : offline optimal with perfect future information
  - n-QoE = QoE / QoE(OPT)

Real Video Player Evaluation

- 2 computers with the same setting: client & server
- Large space to improve (avg 70% QoE(OPT))
- Rule-based method is the worst



(a) FCC dataset (b) HSDPA dataset (c) Synthetic dataset

Figure 8: Real experiment results with different throughput traces

Real Video Player Evaluation

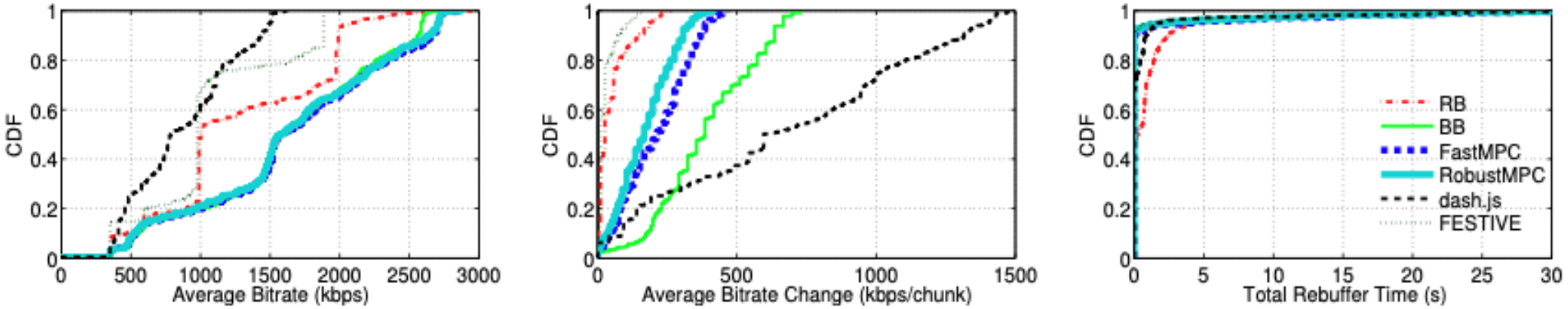


Figure 9: Detailed performance for FCC dataset

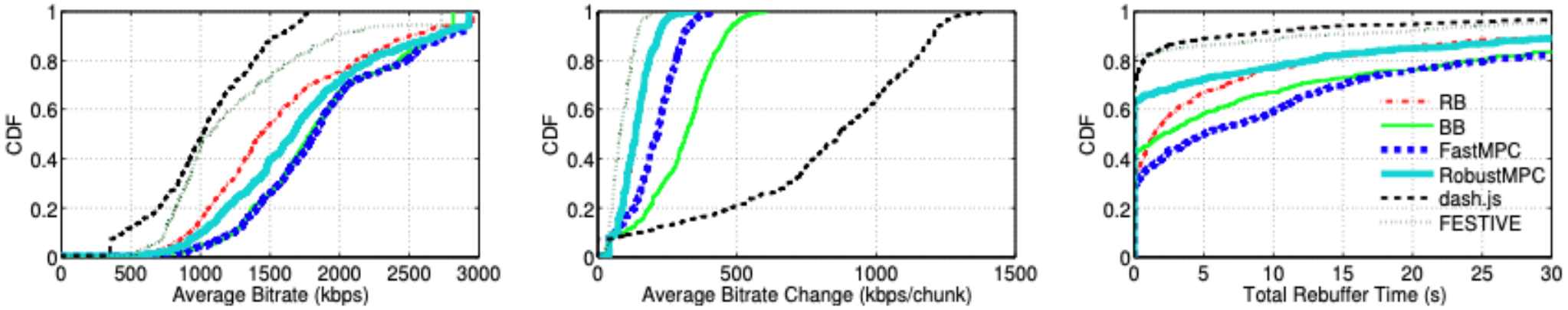
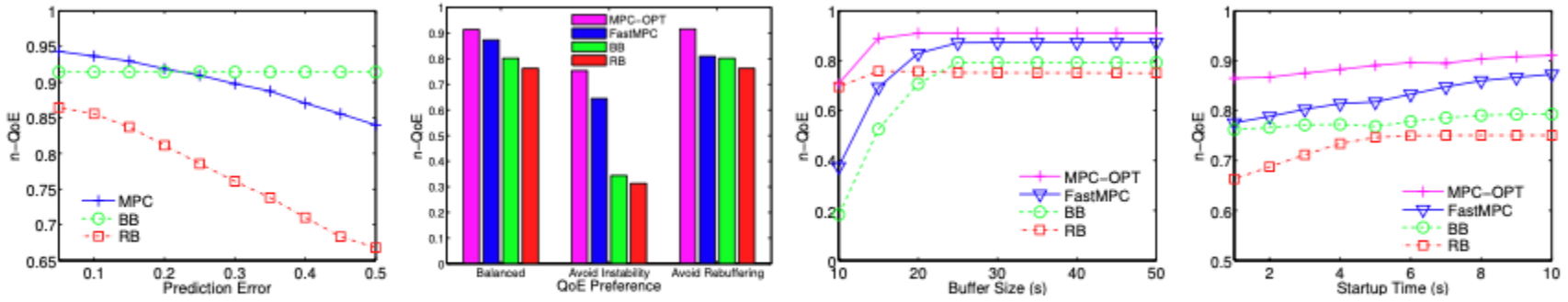


Figure 10: Detailed performance for HSDPA dataset

Sensitivity Analysis: Ablations

- Throughput prediction
  - Analyze general prediction err instead of a specific type (e.g. harmonic mean)
- QoE preference
- Buffer size
  - RB least affected, >25s buffer affects less to all algorithms
- Start-up delay
  - More delay, better rebuffering schedule
- Birate levels:
  - BB&MPC: more levels, better performance
  - RB: w/ increasing #levels, performance first goes up then degenerates



(a) Prediction error (b) QoE preferences (c) Buffer size (d) Startup time

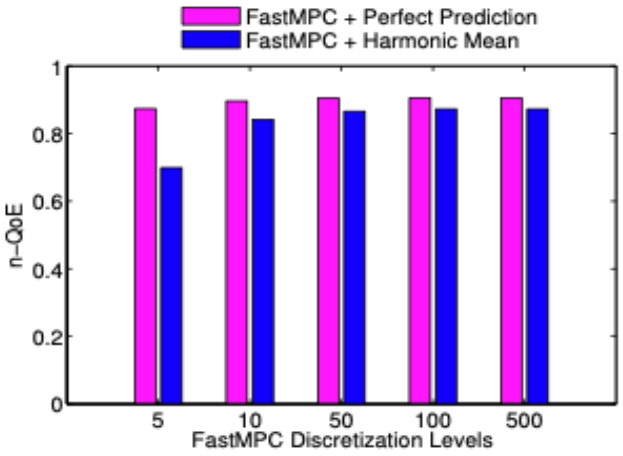
Figure 11: Sensitivity analysis vs. operating conditions

Overhead Analysis

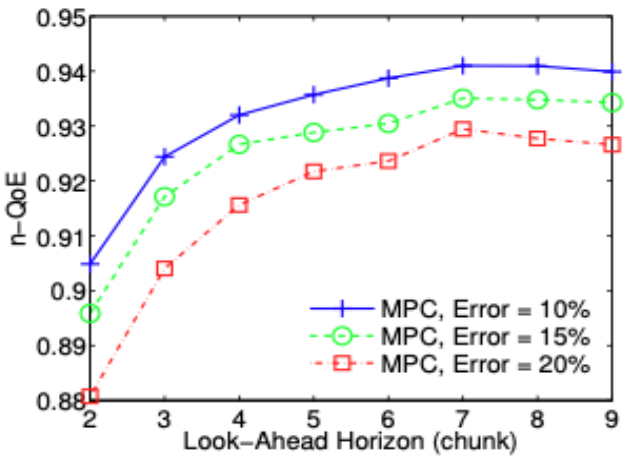
- RB/BB/FastMPC: similar cpu usage
- FastMPC
  - Discretization level
  - Lookahead horizon

Discretization levels	Extra JavaScript code size	
	Full table	Run length coding
50	25.0 kB	19.1 kB
100	100 kB	56.4 kB
200	400 kB	141 kB
500	2.50 MB	451 kB

Table 1: FastMPC table size



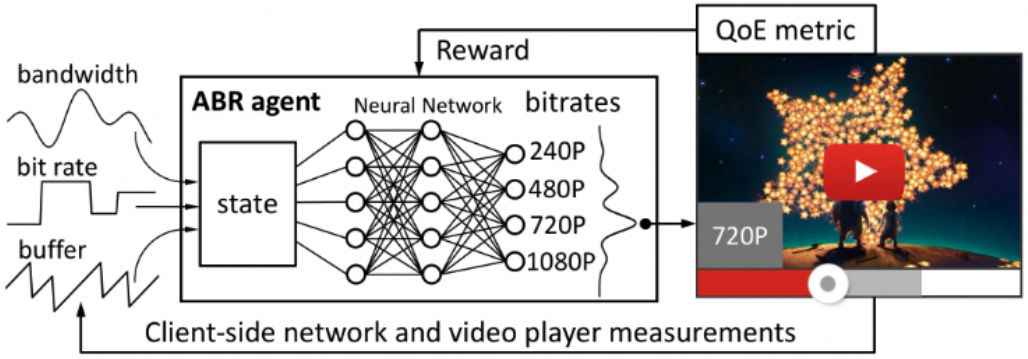
(a) Discretization



(b) Look-ahead horizon

Summary & Future Work

- Summary
  - RobustMPC outperforms other existing methods
    - FastMPC does not show advantages in the mobile setting, but works good for other settings
  - FastMPC implementation is efficient
    - Near-zero cpu overhead & only 60kB extra memory usage
  - FastMPC is more stable compared to RB/BB
- Future Work
  - More advanced techniques for birate decision (e.g. NN)



Mao et al.
SIGCOMM2017