BOLA: Algorithm Analysis, Implementation, and Evaluation

Introduction

The BOLA (Buffer Occupancy based Lyapunov Algorithm) is designed for adaptive bitrate (ABR) streaming, achived by maximizing video quality while minimizing rebuffering [1].

This report includes the algorithm analysis, implementation, and evaluation of the BOLA algorithm implemented in Python.

1. Algorithm Analysis

The core idea of the BOLA algorithm is **Lyapunov optimization**, which selectst the best bitrates to maximize video quality under given constraints.

The BOLA algorithm has the following key parameters:

- Buffer Level $Q(t)$: The amount of video data (measured in seconds) buffered for playback.
- Bitrate S_m: The encoding rate for each video segment, where higher bitrates yield higher quality but require more bandwidth. This is provided in video metadata.
- Utility v_m: A non-decreasing function of the segment bitrate S_m, where $v_m = \ln(S_m/S_1)$, reflecting diminishing returns.
- Control Parameters V and γp: V is a control parameter balancing video quality and buffer occupancy. γp represents a penalty factor for rebuffering events, set by the user to determine the weight between quality and rebuffering.

The primary decision function for each segment in BOLA is: $\rho(t_k, a(t_k)) = \frac{V \cdot (v_m + \gamma p - Q(t_k))}{S_m}$

where $Q(t)$ is the buffer level at time t, S_m is the segment bitrate, and $v_m + \gamma p$ is the utility adjusted by the rebuffering penalty. The algorithm selects the bitrate m that maximizes ρ, balancing the trade-off between higher quality (bitrate) and lower rebuffering.

Besides, there are other components to tackle the challenges of network unstability. For example, a oscillations reduction introduced to avoid frequent bitrate switches.

2. Implementation

The Python code in `studentcode_122090513.py` implemented the core logic of the BOLA algorithm in function: `initialize_bola_state()` and `handle_bola()`.

Since the testing framework cannot support the full BOLA algorithm, for example, "abandoning the high bitrate segments when network bandwidth becomes slow during downloading" ([1] Fig. 6, line 22), the implementation is a simplified version.

a. Initial Setup

For `initialize_bola_state()` functionm, it initializes following values:

1. Bitrates: `bitrates`, which correspond to S_m in paper, passed in from `student_entrypoint` function.
2. Utility Values: `utilities`, which correspond to v_m in paper.
3. Control Parameters: `Vp` and `gp` are calculated based on the given formula, corresponding to V and γp in the paper.

b. Decision Function: `handle_bola()`

The `handle_bola()` function is responsible for selecting the optimal bitrate:

1. Score Calculation: For each available bitrate, the score is calculated using: $\text{score} = \frac{Vp \cdot (\text{utility} - 1 + \text{gp}) - \text{buffer\_level}}{\text{bitrate}}$ This formula helps us to find the best bitrates that balance the buffer level with utility.
2. Quality Selection: The function iterates over bitrates, choosing the one with the highest score.

3. Oscillations reduction: This rule designed to choose a lower bitrate if the available bandwidth is between two bitrates, since this might be a sign of network instability and cause rebuffering. This is similar to "BOLA-O" in the paper.

c. Entry Point & Contants Values

1. Entry Function: The `student_entrypoint()` function provide a interface for calling, it will initailiza the BOLA state and call hanlding function.
2. Logging: Since the testing framework did not show the detail of the value passed in, a logging part is added to show the selected values for each chunk.
3. Constants: The `BUFFER_TIME_DEFAULT`, `MINIMUM_BUFFER_S`, and `MINIMUM_BUFFER_PER_BITRATE_LEVEL_S` are set to `12`, `10`, and `2` respectively, as per the code implementation in JavaScript [2].

3. Evaluation

To test the Python implementation of BOLA, all test cases provided in the testing framework is tested, the result and analysis are as follows. Detailed log can be found in `grade_baseline.txt` and `grade_bola.txt`.

Test Results

Test Case	Algorithm	Avg. Bitrate (bps)	Buffer Time (s)	Switches	Score
testHD	Baseline	4,566,666.67	0.202	2	3,825,384.87
testHD	BOLA (Python)	4,116,666.67	0.202	2	3,448,430.88
testALThard	Baseline	2,166,666.67	72.18	22	8,535.21
testALThard	BOLA (Python)	866,666.67	1.001	1	757,427.81
badtest	Baseline	2,166,666.67	73.026	23	7,518.93
badtest	BOLA (Python)	866,666.67	1.012	1	757,000.57
testALTsoft	Baseline	3,816,666.67	27.341	10	407,852.94
testALTsoft	BOLA (Python)	1,716,666.67	0.784	5	1,086,826.99
testPQ	Both	50,000.0	246.381	0	0.16
testHDmanPQtrace	Both	50,000.0	242.584	0	0.20

Test Analysis

1. Streaming Quality: In `testHD`, both algorithms achieved high average bitrates with minimal buffering, though BOLA's bitrate was slightly lower.

2. Buffer Time: In harsh conditions like `testALThard`, and `badtest`, BOLA reduced the buffer times (e.g., 1.001s vs. 72.18s in `testALThard`) and minimized switches, resulting in smoother playback.

3. Switch Stability: Across tests, BOLA consistently minimized bitrate switches, enhancing stability and reducing disruptions. For example, in `testALTsoft`, BOLA had half the switches of the baseline.

4. Low Bandwidth Test: In poor network tests (`testPQ`, `testHDmanPQtrace`), both algorithms behaved similarly, keeping minimal bitrates.

Generally speaking, the BOLA algorithm performed well in differten tests, this makes it a good choice for ABR streaming.

References

[1] K. Spiteri, R. Urgaonkar and R. K. Sitaraman, "BOLA: Near-Optimal Bitrate Adaptation for Online Videos," in *IEEE/ACM Transactions on Networking*, vol. 28, no. 4, pp. 1698-1711, Aug. 2020, doi: 10.1109/TNET.2020.2996964.

[2] Dash-Industry-Forum. "Dash-Industry-Forum/dash.js: A reference client implementation for the playback of MPEG DASH via Javascript and compliant browsers." https://github.com/Dash-Industry-Forum/dash.js.