Introduction

In this assignment, I perused the example code and its corresponding paper *A Buffer-Based Approach to Rate Adaptation: Evidence from a Large Video Streaming Service*, and found a bug in the helper function *prevmatch* (since the previous bitrate can never match with any of current available chunks' byte rates, the original version would always return the highest available bitrate, which contradicts to its original design intention. I modified its logic and fixed it.). Then, I did a little analysis of the remaining optional papers, and selected *BOLA: Near-Optimal Bitrate Adaptation for Online Videos* for code reproduction. Because I thought BOLA algorithm and given network system framework had the best compatibility: took the time to download a chunk or wait for a fixed time when not downloading as a unit slot, and made playback & buffer settlement and download decision at the beginning of each slot.

Algorithm Analysis

Basic symbols:

symbol	meaning
N	The number of chunks the video file is segmented into
p	The duration of each chunk in seconds
M	The number of available bitrates of each chunk
S_m	The size of any chunk encoded at bitrate index m in bits (assuming the bitrates are ascending)
v_m	The utility derived by the user from viewing S_m
Q_{max}	The finite buffer size in chunk
$w(t)$	The available bandwidth at time t in bits/second
k	The index of time slot. If the player decides to download a chunk, a slot lasts the time to complete downloading a chunk; otherwise the slot lasts a fixed duration of Δ second (Δ = 0.5 in this system framework). The time slot starts at t_k.
$a_m(t_k)$	The control function: =1 if the player downloads a chunk of bitrate index m in time slot k, = 0 otherwise
$Q(t_k)$	The buffer level at the start of slot k in number of chunks
K_N	The index of slot in which the N-th chunk is downloaded
T_{end}	The time when the player finishes playing back the last chunk
γ	The input weight parameter. It corresponds to how strongly we want to avoid rebuffering
V^D	The dynamic positive control parameter to allow a tradeoff between the buffer size and the performance objectives

Important formulas:

Utility function: $v_m = \ln\left(\frac{S_m}{S_1}\right)$

Input weight parameter: $\gamma = \frac{5}{p}$

Time-average expected playback utility:

$$\overline{v}_N \triangleq \frac{\mathbb{E}\left\{\sum_{k=1}^{K_N}\sum_{m=1}^{M} a_m(t_k)v_m\right\}}{\mathbb{E}\left\{T_{\text{end}}\right\}}$$

Expected smoothness:

$$\overline{s}_N \triangleq \frac{Np}{\mathbb{E}\left\{T_{\text{end}}\right\}} = \frac{\mathbb{E}\left\{\sum_{k=1}^{K_N}\sum_{m=1}^{M} a_m(t_k)p\right\}}{\mathbb{E}\left\{T_{\text{end}}\right\}}$$

Goal of the algorithm:

Design a control algorithm that maximizes the joint utility $\bar{v}_N + \gamma \bar{S}_N$ subject to the constraint that $Q(t_k) \le Q_{max}$.

Optimization problem in each decision:

$$Maximize\colon \begin{cases} 0 & , if \sum_{m=1}^{M} a_m(t_k) = 0 \\ \dfrac{\sum_{m=1}^{M} a_m(t_k)\,(V^D v_m + V^D \gamma p - Q(t_k))}{\sum_{m=1}^{M} a_m(t_k)\, S_m} & , otherwise \end{cases}$$

$$Subject\ to\colon \sum_{m=1}^{M} a_m(t_k) \le 1, a_m(t_k) \in \{0,1\}$$

Implementation

Capability problem:

The paper measures the buffer in seconds. However, in this system, the buffer is messured in bytes. My proposed solution is to estimate the buffer size Q_{max} in chunks as: $\frac{downloaded\ video\ time}{p} + \frac{remaining\ buffer\ space}{\frac{previous\ bitrate}{8} \times p}$.

In some test data (with PQ), available bitrates are inconsistent with actual chunks' bitrates. I used current actual chunks' bitrates of the same index m to estimate $\frac{previous\ bitrate}{8}$.

Psudocode of decision algorithm in each slot:

t = min(played time, remaining time) in second

t' = max(t / 2, 3p)

Q_{max}^D = min(Q_{max}, t' / p)

V^D = (Q_{max}^D – 1) / ($v_M + \gamma p$)

m* = arg max $\left(V^D v_m + V^D \gamma p - Q(t_k)\right)/S_m$

if (m* > last m*):

    r = measured throughput

    m' = max m such thatt $\frac{S_m}{p} \le \max\left(r, \frac{S_1}{p}\right)$

    if (m' > m*):

        m' = m

    else if (m < last m*):

        m' = last m*

    m* = m'

Evaluation

Here I compared the performance of Buffer-based implementation (Example code) and BOLA implementation under different configuations:

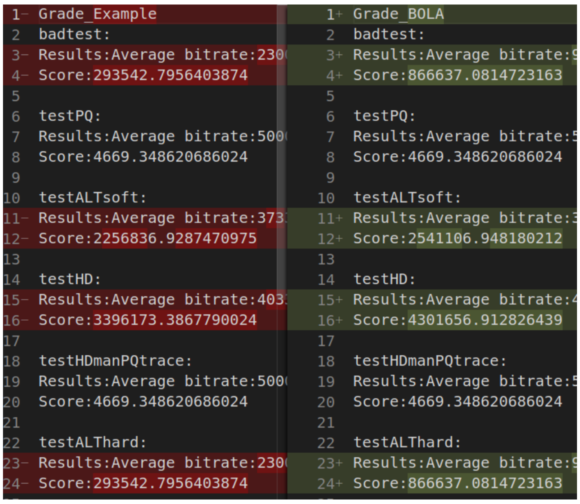
testALThard: test that have a unstable bandwidth that confuses ABR algos

testALTsoft: test that have a lot of alternating bandwidth

testHD: test that have high quality bandwidth and other params

testHDmanPQtrace: test that have high quality bandwidth but low params

testPQ: test that have low quality bandwidth and param, will rebuffer.



```
Grade_Example
badtest:
Results:Average bitrate:230
Score:293542.7956403874

testPQ:
Results:Average bitrate:500
Score:4669.348620686024

testALTsoft:
Results:Average bitrate:373
Score:2256836.9287470975

testHD:
Results:Average bitrate:403
Score:3396173.3867790024

testHDmanPQtrace:
Results:Average bitrate:500
Score:4669.348620686024

testALThard:
Results:Average bitrate:230
Score:293542.7956403874
```

```
Grade_BOLA
badtest:
Results:Average bitrate:9
Score:866637.0814723163

testPQ:
Results:Average bitrate:5
Score:4669.348620686024

testALTsoft:
Results:Average bitrate:3
Score:2541106.948180212

testHD:
Results:Average bitrate:4
Score:4301656.912826439

testHDmanPQtrace:
Results:Average bitrate:5
Score:4669.348620686024

testALThard:
Results:Average bitrate:9
Score:866637.0814723163
```

It can be seen that even though BOLA is not optimized for score calculation formula in grader.py, its performanxe is still better than the former.

How to run the code:

python3 simulator.py <tracefile.txt> <manifestfile.json>

python3 studentComm.py

in two separate terminals, or:

python3 grader.py

to see the performance in *grade.txt*

If you cannot run it using "python3", then try "python". Remember to replace all "python3" to "python" in grader.py.