深 圳 大 学 实 验 报 告

|  |  |  |  |
| --- | --- | --- | --- |
| 课程名称 | 计算机网络 | | |
| 实验名称 | 实验2：码率自适应传输 | | |
| 学院 | 计算机与软件学院 | | |
| 专业 | 软件工程（腾班） | | |
| 指导教师 | 张磊 | | |
| 报告人 | 黄亮铭 | 学号 | 2022155028 |
| 实验时间 | 2024年3月15日~2024年4月10日 | | |
| 提交时间 | 2024年4月10日 | | |

教务处制

# 实验目的与要求

* + - 1. 阅读并理解ABR算法。
      2. 复现一个ABR算法，根据网络带宽、缓冲区容量等因素，自适应选择视频段的码率并请求。
      3. 完成试验任务：引入网络波动、理解简单ABR算法和改进ABR算法。

# 实验过程

* + - 1. **引入网络波动：**下载实验所需要的库，引入fluctuation.h头文件，根据功能解析模块中的Network Fluctuation，修改源代码，模拟网络传输中的带宽波动。
         1. 在服务端引入fluctuation.h头文件；

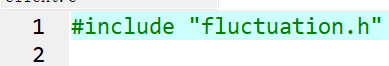


图1：引入头文件

* + - * 1. 在代码初始化阶段调用load\_fl()函数初始化网络波动相关信息；

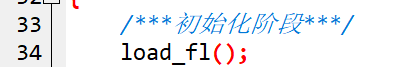


图2：调用load\_fl()函数

* + - * 1. 更改原文件发送函数send()为send\_fl()，参数一致；







图3：更改服务端文件函数名

* + - * 1. 在结束阶段调用release\_fl()函数释放为网络波动模拟模块所分配的系统资源；

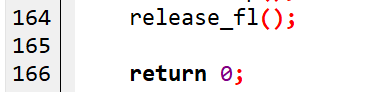


图3：调用release\_fl()函数

* + - 1. **简单ABR算法理解**

这个简单的自适应比特率（ABR）算法的主要目标是根据当前网络状况和播放器缓冲情况，动态地选择适当的视频码率，以提供最佳的观看体验

在这个简单ABR算法中，核心是动态调整视频码率，以平衡视频质量和网络带宽。当网络条件较好时，选择较高分辨率以提供更好的观看体验；而当网络条件较差时，选择较低分辨率以避免缓冲和卡顿现象。同时，通过等待一段时间来为下一个视频段制定码率决策，可以更好地适应网络状况的变化，提高整体的观看体验。

但是，在当前版本的代码框架中，需要等待一段时间才能为下一个视频段制定码率决策，这可能会导致不必要的等待时间。要优化这一点，可以考通过**异步处理或并行处理**来实现这一优化，以确保在等待视频段下载的同时，仍可以进行码率决策的制定。

ABR算法的主要步骤为：初始化下载队列，然后循坏请求视频段，同时统计每个视频段的下载时间，将视频段交付播放器并进行QoE记录和码率自适应决策。

* + - 1. **改进ABR算法**
         1. 简单的ABR算法

简单的码率自适应决策为：如果当前视频段的下载时间小于200ms，则以1080p的分辨率下载新的视频段；如果当前视频段下载时间不小于200ms但小于600ms，则以480p的分辨率下载新的视频段；否则以360p的分辨率下载新的视频段。

根据上述的决策描述，我实现的代码如下图所示。

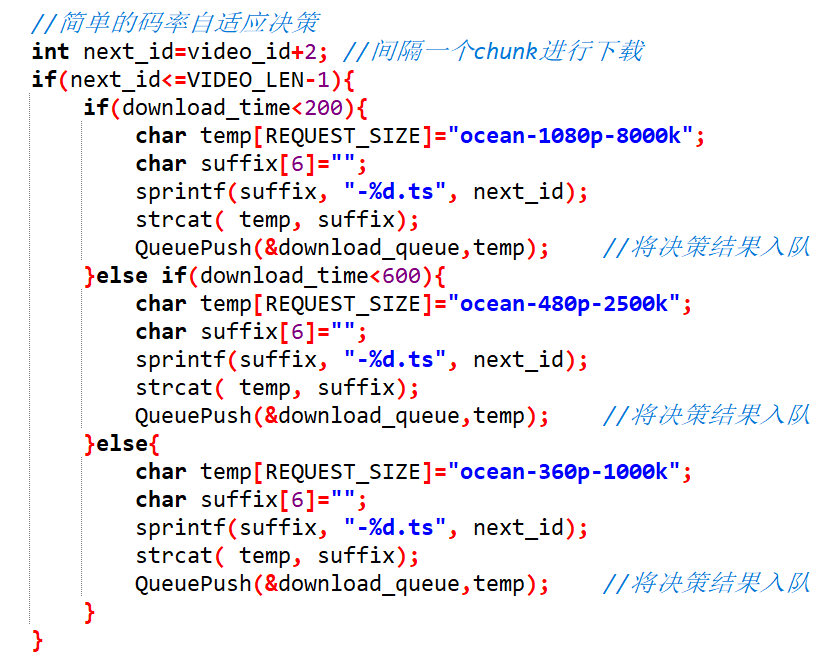


图4：简单ABR算法

* + - * 1. 改进的ABR算法I

通过查看我们需要下载的视频的文件，我们发现视频源中存在720p分辨率的视频文件。而简单的码率自适应决策中忽略了720p分辨率的视频文件，客户端只有1080p、480p和360p三种分辨率的视频文件可以选择。因此，在改进ABR算法I中，我们加入了720p分辨率的视频文件供客户端选择。

改进的ABR算法I的码率自适应决策为：如果当前视频段的下载时间小于200ms，则以1080p的分辨率下载新的视频段；如果当前视频段下载时间不小于200ms但小于400ms，则以720p的分辨率下载新的视频段；如果当前视频段下载时间不小于400ms但小于600ms，则以480p的分辨率下载新的视频段；否则以360p的分辨率下载新的视频段。

根据上述的决策描述，我修改了简单ABR算法的代码，新增了720p分辨率的选择分支。

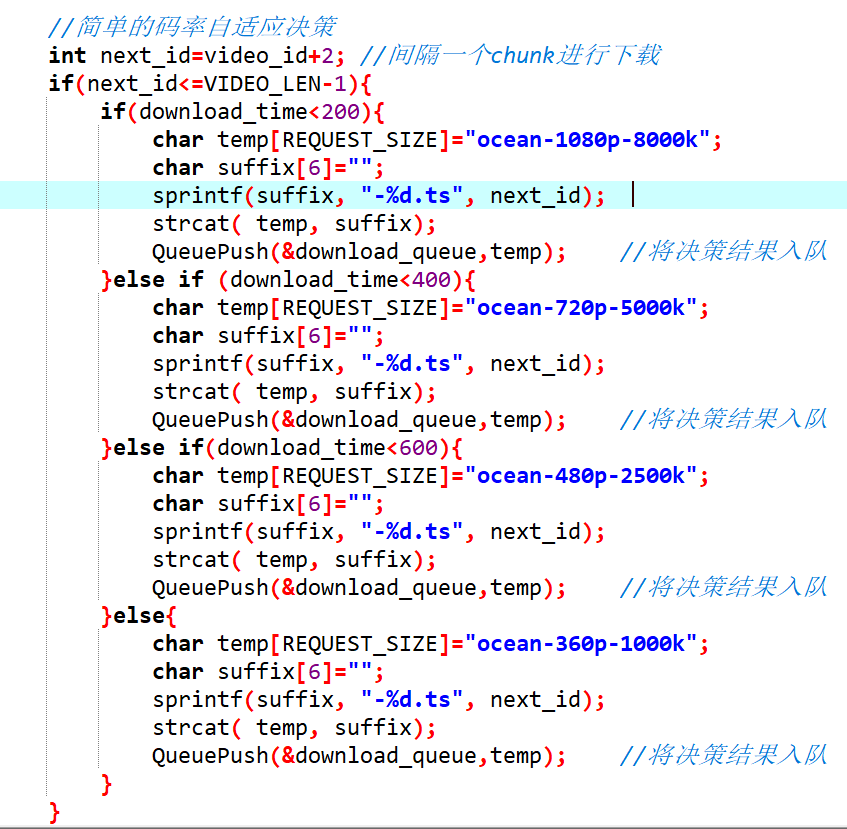


图5：改进ABR算法I

* + - * 1. 改进的ABR算法II（基于缓存补偿的视频码率自适应算法）

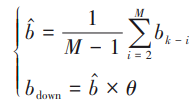
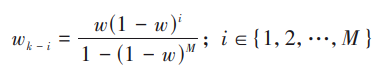
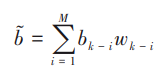
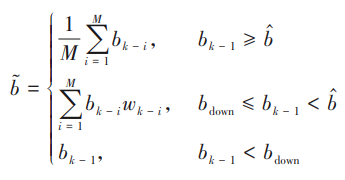
算法策略：在初始阶段，客户端始终请求低于当前带宽值的码率等级；在一段时间后，视频缓存时长累积至上切阈值，码率决策模块请求高于当前带宽值的码率级别；随时间推移，缓存时长消耗至下切 阈值以下，码率决策模块逐级切换码率等级并重新累积缓存时长。

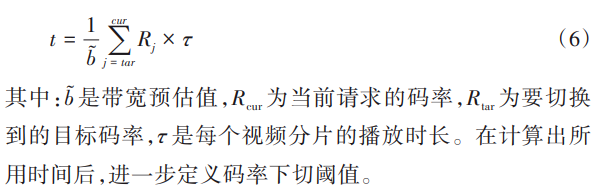
不妨设视频缓存时长最大为q\_max，显然最小值为0。进一步在视频缓存时长内划分缓存阈值，当缓存时长低于q\_min时，客户端将请求最低码率以快速积累缓存市场；当缓存时长高于q\_up时，客户端将以最高码率请求视频分片，提高画面质量。以最高码率请求视频分片时缓存进入消耗状态并向动态下切阈值 q\_down运动，当缓存时长 低于 q\_down时，码率决策模块将逐级切换至当前带宽下的最高 码率，缓存再次进入累积状态。



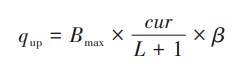
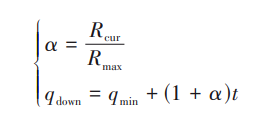
图6：视频缓存时长

算法设计：

1. 带宽预估：取前M（代码实现中为4）个实际下载速率的均值为下一时刻的带宽预估值（除去当前下载速率）记为，。若当前分片下载速率低于，则以作为下一时刻带宽预估值。若当前分片下载速率大于，则以前M（代码实现中为4）个实际下载速率的均值为下一时刻的带宽预估值。否则，我们对前M个视频分片的实际下载速率赋权重，然后计算得出下一时刻带宽预估值。权重计算公式如下：，下一时刻带宽预估值为：。综上所述，带宽预估值定义如下：。
2. 码率自适应：首先我们设置q\_min和B\_max（q\_min定义同前文，B\_max定义为缓存上限），然后计算出切换码率所需要的时间：



然后依次计算出和:

，其中，为上切阈值调整参数，取值为0.85时效果最好。

算法实现：

因为代码较长，这里只截图核心部分的代码：预测带宽值得getNextDownLoadRate()函数和根据预测带宽值选择不同码率的视频分片的getNextBitRate()函数。其余函数见代码压缩包或实验报告附页。

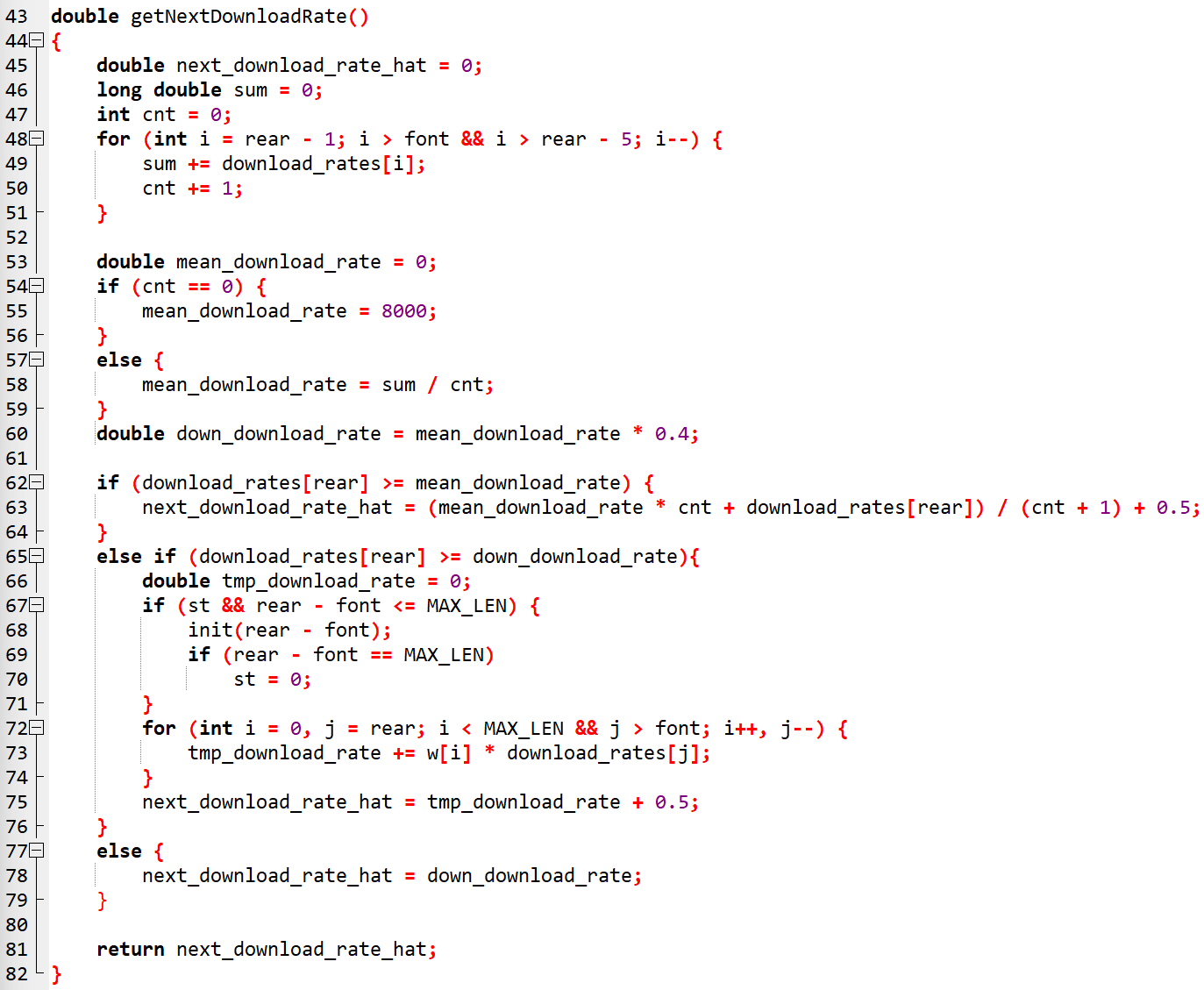
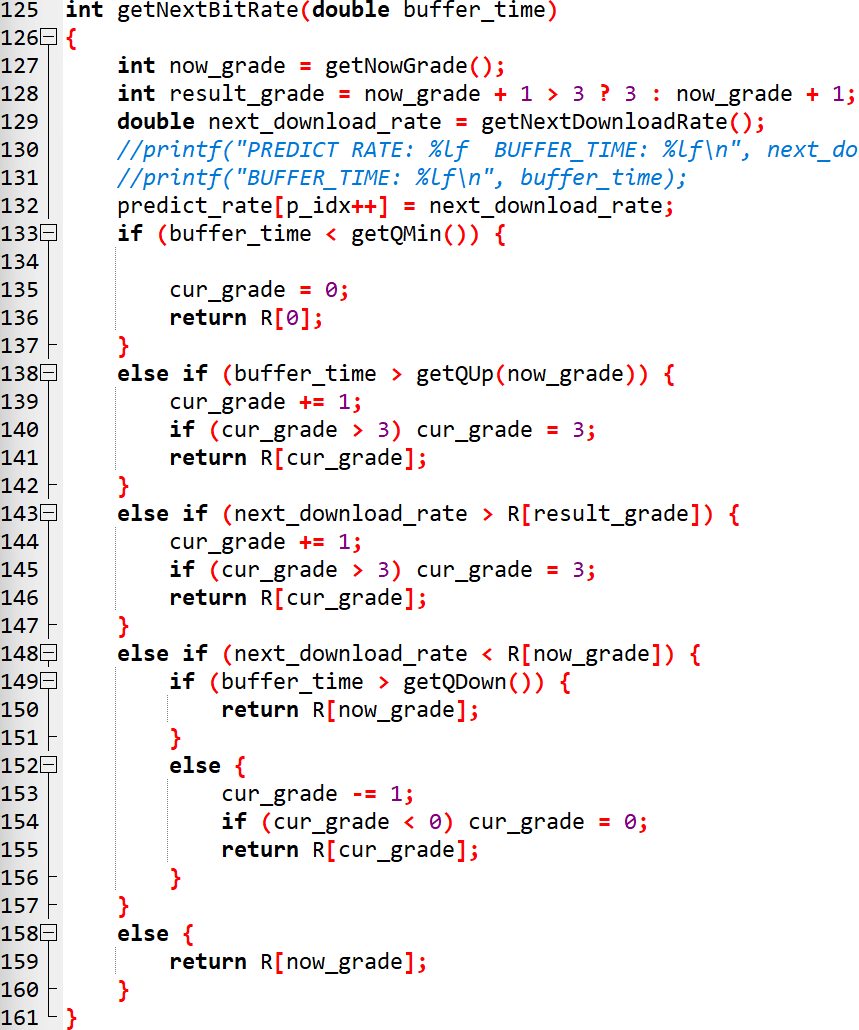
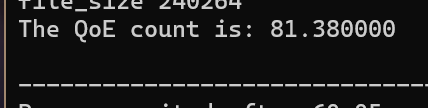
图6：getNextDownLoadRate()

图7：getNextBitRate()

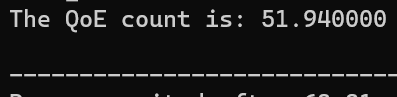
# 实验结果

* + - 1. **简单的ABR算法**

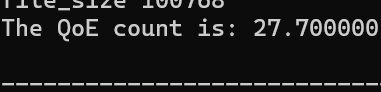
运行简单的ABR算法进行测试，评分结果如下图。



图：Trace1



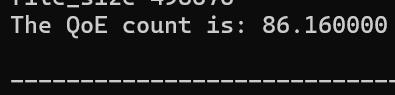
图：Trace2



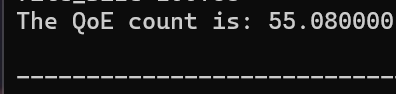
图：Trace3

* + - 1. **改进的ABR算法I**

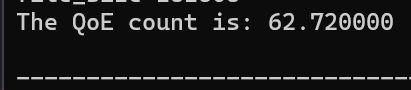
运行改进ABR算法I进行QoE指标结算，评分结果如下图：



图：Trace1



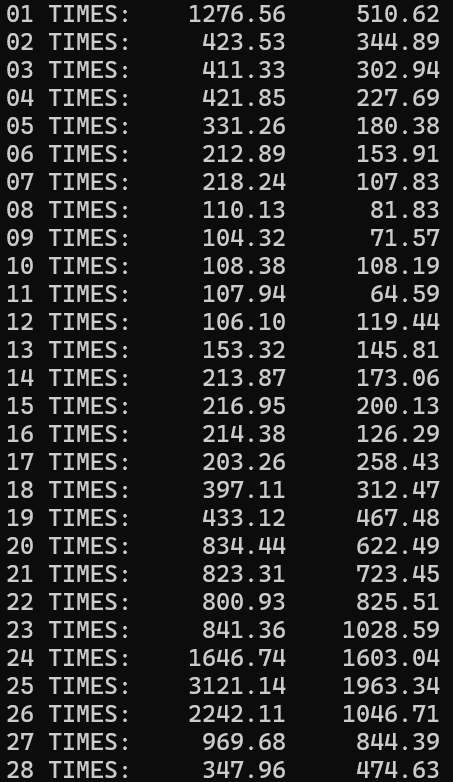
图：Trace2



图：Trace3

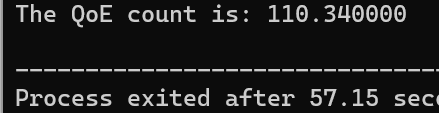
* + - 1. **改进的ABR算法II**

如下图所示（左：视频分片序号，中：实际带宽，右：预估带宽），改进ABR算法II对视频带宽值的预估非常准确，即使是在带宽突变的情况下，改进ABR算法II对带宽的预估情况仍然较好。

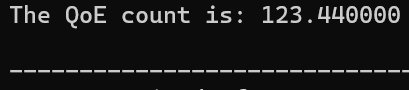


图：带宽预估（Trace1）

运行改进ABR算法I进行QoE指标结算，评分结果如下图：

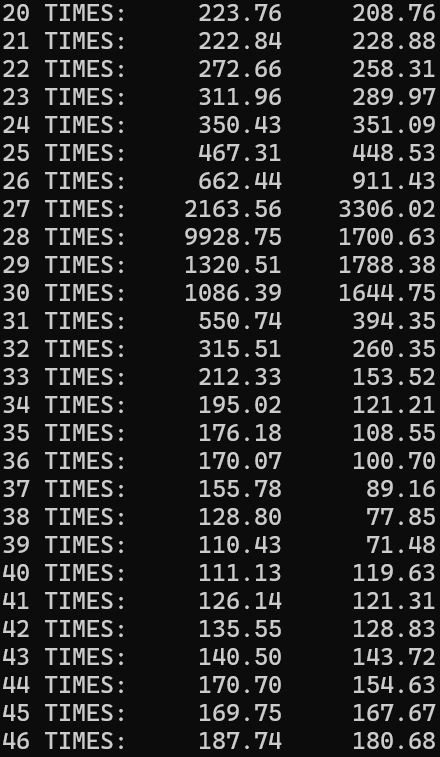


图：Trace1（平均）



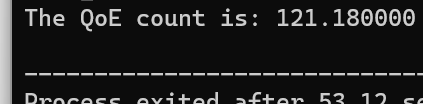
图：Trace1（最佳）

如下图所示（左：视频分片序号，中：实际带宽，右：预估带宽），改进ABR算法II对视频带宽值的预估非常准确，即使是在带宽突变的情况下，改进ABR算法II对带宽的预估情况仍然较好。

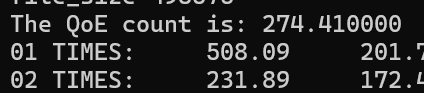


图：带宽预估（Trace2）

运行改进ABR算法I进行QoE指标结算，评分结果如下图：

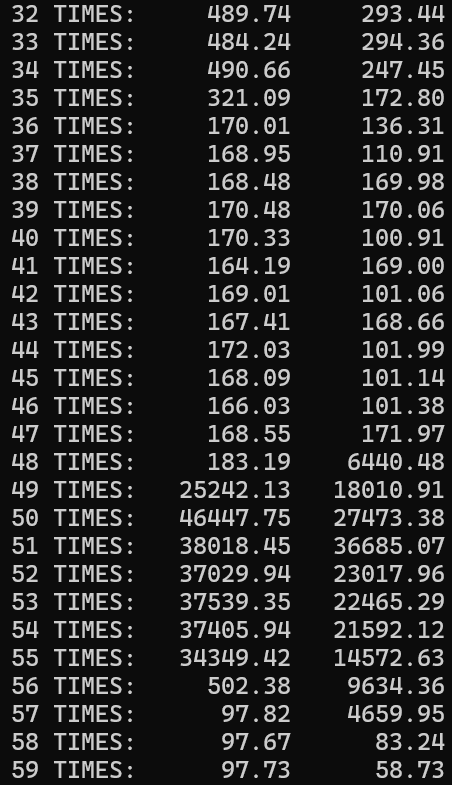


图：Trace2（平均）



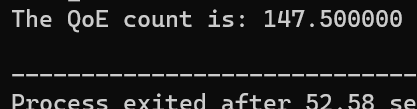
图：Trace2（最佳）

如下图所示（左：视频分片序号，中：实际带宽，右：预估带宽），改进ABR算法II对视频带宽值的预估非常准确，即使是在带宽突变的情况下，改进ABR算法II对带宽的预估情况仍然较好。

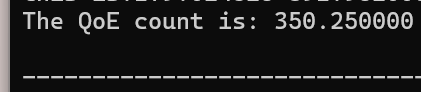


图：带宽预估（Trace3）

运行改进ABR算法I进行QoE指标结算，评分结果如下图：



图：Trace3平均



图：Trace3最佳

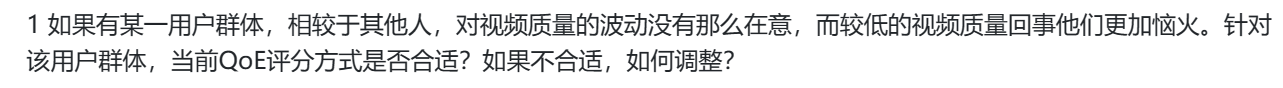
# 实验分析

* + - 1. 简单的ABR算法实现非常简单，代码结构清晰易懂。但是存在不能适应网络环境的动态变化的缺点，在网络波动较大的时候尤其明显。实验结果表明简单的ABR算法并不能很好地适应网络波动。
      2. 改进的ABR算法I只在简单的ABR算法之上增加了一个720P的选择分支，让客户端在480P和1080P之间增加了一个选项。该选项使得客户端在网络波动时可以选择更加合适的码率播放视频，提高了视频播放质量和用户体验。实验结果表明增加的720P分支在一定程度上改善了简单ABR算法存在的缺点。
      3. 改进的ABR算法II是基于缓存补偿的视频码率自适应算法，能预测下一时刻的网络带宽，同时也能根据视频缓存时长动态调整请求的码率等级。实验结果表明该算法对网络带宽的预测比较准确，能适应网络环境的动态变化，即使是网络波动较大的时候也能有良好的表现。QoE评分也比上述两者高，但是并不是很明显。原因可能是因为算法复杂度较前两者更高，而视频总时长较短，无法体现绝对优势。
      4. 通过对比三种不同的ABR算法，可以发现改进的ABR算法II在提高用户观看体验方面具有明显优势。然而，实际应用中需要考虑算法的复杂度、实现成本以及调整参数的难度等因素，选择最适合具体场景和需求的算法。

# 实验总结

* + - 1. 本次实验我实现了[基于缓存补偿的视频码率自适应算法](E://DownFromEdge/Buffer%20compensation%20based%20video%20bitrate%20adaptation%20algorithm.pdf)这篇论文里面的ABR算法。
      2. 我查阅了fluctuation.h中的源码，了解了如何通过基础延迟和随机延迟组合模拟网络波动。
      3. 通过本次实验，我了解了经典ABR算法的分类：基于带宽预测、基于缓冲区容量和基于神经网络学习等。
      4. 通过本次实验，我了解了ABR算法的基本原理和优化方法。

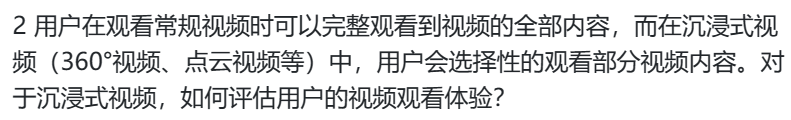
# 思考题



当前QoE评分方式可能不太合适。因为它对视频卡顿的惩罚程度较高，而对高视频质量的奖励程度较低，同时对视频质量波动的惩罚也较高。这会导致客户端更加倾向播放低质量视频以维持视频不卡顿。

调整方法：1）增加对较低视频质量的惩罚；2）适当降低对卡顿时间的惩罚；3）适当提高对高质量视频的奖励；4）适当降低视频质量波动的惩罚程度。

例如设置参数（参照实验平台）：alpha=0.005, beta=0.001,gamma=0..005,新增参数c=0.001。



1. 考虑视频的加载速度、流畅度、缓冲时间等技术性能指标，这些指标直接影响到用户的观看体验。
2. 监测用户在沉浸式视频中的交互行为，包括观看方向、转头频率、放大缩小等操作。这些交互行为可以提供关于用户兴趣和注意力焦点的重要信息。
3. 考虑用户在不同部分视频内容上的观看时长和观看频率。通过分析用户对不同内容区域的观看持续时间和频率，可以了解用户对视频中各个部分的关注程度。
4. 收集用户的反馈和评价，了解他们对沉浸式视频观看体验的主观感受。这可以通过用户调查、问卷调查、用户评论等方式进行收集。

# 附页

改进ABR算法II头文件全部代码如下所示

#include <math.h>

#define MAX\_LEN 4

#define N 100000

#define W 0.4

#define Qmin 2000

#define Bmax 10000

//const int MAX\_LEN = 4, N = 1e5 + 10;

//const double W = 0.6;

const int R[] = { 1000, 2500, 5000, 8000 };

double download\_rates[N];

double w[MAX\_LEN];

int font = 0, rear = 0;

int cur\_grade = 3;

int st = 1;

double predict\_rate[N];

int p\_idx;

void init(int len)

{

double beta = 1, alpha = W;

for (int i = 0; i < len; i++) {

beta \*= (1 - W);

}

beta = 1 - beta;

for (int i = 0; i < len; i++) {

alpha \*= (1 - W);

w[i] = alpha / beta;

}

}

void updateDownQueue(int file\_size, long download\_time)

{

double alpha = 1;

if (download\_time <= 0) {

download\_time = 1;

alpha = 0.85;

}

double download\_rate = (double)file\_size / download\_time / alpha;

download\_rates[++rear] = download\_rate;

if (rear - font > 4) font++;

}

double getNextDownloadRate()

{

double next\_download\_rate\_hat = 0;

long double sum = 0;

int cnt = 0;

for (int i = rear - 1; i > font && i > rear - 5; i--) {

sum += download\_rates[i];

cnt += 1;

}

double mean\_download\_rate = 0;

if (cnt == 0) {

mean\_download\_rate = 8000;

}

else {

mean\_download\_rate = sum / cnt;

}

double down\_download\_rate = mean\_download\_rate \* 0.4;

if (download\_rates[rear] >= mean\_download\_rate) {

next\_download\_rate\_hat = (mean\_download\_rate \* cnt + download\_rates[rear]) / (cnt + 1) + 0.5;

}

else if (download\_rates[rear] >= down\_download\_rate){

double tmp\_download\_rate = 0;

if (st && rear - font <= MAX\_LEN) {

init(rear - font);

if (rear - font == MAX\_LEN)

st = 0;

}

for (int i = 0, j = rear; i < MAX\_LEN && j > font; i++, j--) {

tmp\_download\_rate += w[i] \* download\_rates[j];

}

next\_download\_rate\_hat = tmp\_download\_rate + 0.5;

}

else {

next\_download\_rate\_hat = down\_download\_rate;

}

return next\_download\_rate\_hat;

}

int getSwitchGrade()

{

double max\_rate = getNextDownloadRate();

int tar\_grade = 0;

for (int i = 0; i < 4; i++) {

if (max\_rate > R[i]) {

tar\_grade = i - 1;

}

}

return tar\_grade;

}

int getNowGrade()

{

return cur\_grade;

}

double getSwitchingTime(double now\_bit\_rate\_idx)

{

double t = 1 / getNextDownloadRate();

int tmp = 0;

for (int i = max(getSwitchGrade(), now\_bit\_rate\_idx); i >= min(getSwitchGrade(), now\_bit\_rate\_idx); i--) {

tmp += R[i];

}

return t \* tmp;

}

double getQMin()

{

return Qmin;

}

double getQDown()

{

double alpha = R[getNowGrade()] / R[0];

return Qmin + (1 + alpha) \* getSwitchingTime(getNowGrade());

}

double getQUp(int now\_bit\_rate\_idx)

{

return Bmax \* 0.85 \* (now\_bit\_rate\_idx + 1) / 4 ;

}

int getNextBitRate(double buffer\_time)

{

int now\_grade = getNowGrade();

int result\_grade = now\_grade + 1 > 3 ? 3 : now\_grade + 1;

double next\_download\_rate = getNextDownloadRate();

if (buffer\_time < getQMin()) {

cur\_grade = 0;

return R[0];

}

else if (buffer\_time > getQUp(now\_grade)) {

cur\_grade += 1;

if (cur\_grade > 3) cur\_grade = 3;

return R[cur\_grade];

}

else if (next\_download\_rate > R[result\_grade]) {

cur\_grade += 1;

if (cur\_grade > 3) cur\_grade = 3;

return R[cur\_grade];

}

else if (next\_download\_rate < R[now\_grade]) {

if (buffer\_time > getQDown()) {

return R[now\_grade];

}

else {

cur\_grade -= 1;

if (cur\_grade < 0) cur\_grade = 0;

return R[cur\_grade];

}

}

else {

return R[now\_grade];

}

}

指导教师批阅意见

成绩评定

指导教师签字：

年 月 日

注：1、报告内的项目或内容设置，可根据实际情况加以调整和补充。

2、教师批改学生实验报告时间应在学生提交实验报告时间后10日内。